

**Raimund Ubar**

# **Digitaalsüsteemide diagnostika**

**I**

**Diagnostiline modelleerimine**

**TTÜ**

KIRJASTUS

Selle õppevahendi väljaandmist  
toetas Eesti Infotehnoloogia Sihtasutus  
Tiigriülikooli projekti kaudu

Autoriõigus © Raimund Ubar 2005  
Autoriõigus © Tallinna Tehnikaülikool 2005

Kaane kujundus: Tiina Ubar Sauter

## Sisukord

Eessõna .....	5
Sissejuhatus .....	8
1. Digitaalsüsteemide diagnostiline modelleerimine .....	15
1.1. Boole'i diferentsiaalaparaat .....	16
1.1.1. Boole'i osatuletised .....	16
1.1.2. Boole'i osatuletiste omadusi .....	19
1.1.3. Boole'i vektortuletised .....	21
1.1.4. Kordsed Boole'i osatuletised .....	23
1.1.5. Liitfunktsioonide diferentseerimine .....	25
1.1.6. Boole'i osatuletised järjestikskeemidele .....	27
1.1.7. Boole'i difentsiaalid .....	30
1.1.8. Digitaalskeemide diagnostika üldvõrrand .....	34
1.2. Otsustusdiagrammid .....	36
1.2.1. Binaarsete otsustusdiagrammide mõiste .....	37
1.2.2. Struktuurselt sünteesitud binaarsed otsustusdiagrammid .....	39
1.2.3. Diagnostikaoperatsioonid SSBOD-mudelil .....	45
1.2.4. Otsustusdiagrammide üldjuhtum .....	49
1.2.5. Otsustusdiagrammid juhtautomaatidele .....	55
1.2.6. Otsustusdiagrammid digiaalsüsteemidele registersiirete tasandil .....	63
1.2.7. Otsustusdiagrammid mikroprotsessorite diagnostiliseks modelleerimiseks .....	75
1.3. Kokkuvõtteks .....	77
2. Rikete modelleerimine digitaalsüsteemides .....	80
2.1. Rikete klassifikatsioon .....	81

2.1.1.	Defektid, rikked, vead .....	81
2.1.2.	Rikete mudelite üldised omadused .....	82
2.1.3.	Konstantsed rikked .....	85
2.1.4.	Lühised, katkestused ja viiterikked .....	87
2.2.	Konstantsete rikete omadused .....	94
2.2.1.	Rikete liiasus .....	94
2.2.2.	Rikete dominantsus- ja ekvivalentsussuhted .....	96
2.2.3.	Rikete maskeerumine .....	101
2.3.	Funktsionaalne rikkemudel .....	104
2.3.1.	Rikete modelleerimine Boole'i diferentsiaalvõrranditega .....	104
2.3.2.	Füüsiliste defektide kujutamine loogikatasandile .....	108
2.3.3.	Ühendusahelate defektide kujutamine loogikatasandile .....	114
2.3.4.	Rikete hierarhiline esitus .....	117
2.4.	Kõrgtasandi rikete mudelid .....	119
2.4.1.	Mikroprotsessori funktsionaalne rikete mudel .....	120
2.4.2.	Registersiirete tasandi rikete mudelid .....	124
2.4.3.	Rikete modelleerimine otsustusdiagrammidega .....	126
2.5.	Kokkuvõtteks .....	132
	Kirjandus .....	134
	Lühendid .....	138
	Mõisteregister .....	139

## Eessõna

*“Eksimine on inimlik.  
Et aga tõeliselt kõik ära rikkuda,  
on vaja arvutit.”  
(insenerifolkloor)*

Viimase kümne aastaga on inimkond produtseerinud rohkem teadmisi kui kogu eelneva tsivilisatsiooni vältel. Selle tulemusena on tekkinud meeletu kiirusega arenev tehismaailm, kus materiaalsed artifaktid on suurendanud inimese tugevust ja kiirust ning kognitiivsed artifaktid (arvutid) on teinud teda targemaks ja kavalamaks.

Arvutite võimsus kahekordistub pooleteise aastaga. Aastast 1950 kuni tänaseni on see kasvanud 10 miljardit korda, mis on palju rohkem kui võimsuse kasv, mille tõi kaasa üleminek dünaamiidilt vesinikupommile. Hiilivalt pealetungiv infotehnoloogiline *ubikvism* (nähtamatud arvutid kõikjal ja ei kusagil – *ubiquitous computing*) on muutmas ulmeraamatute stsenaariumid tõelisuseks.

Tehnoloogia keerukus kasvab, aga keerukus kätkeb endas ootamatusi ja mittekavandatut. Elu on täis ohte, aga võrku ühendatud arvutid, mis juhivad ja kontrollivad tehismaailma, on täis veelgi suuremaid ohte: arvutirikete põhjustatud lennukatastroofid, õnnetused keemiatööstuses, tuumajaamades... Tehissüsteemide ohutus ja turvalisus on muutumas sageli olulisemaks kui niisuguste süsteemide täiuslikkus.

Esimest arvutiviga maailmas dateeritakse aastaga 1945, kui koiliblikas sattus Harvardis arvuti Mark II kahe relee vahele. 1991. aasta Lahesõjas tabab Iraaki suunatud raket USA enda raketibaasi, kuna ümardamisvea tõttu arvutas kompuuter valesti raketi trajektoori. 1995 plahvatab Ariane-5 juba 40 sek pärast starti tarkvaravea tõttu inertsiaalsüsteemis. 1996 saadab Intel turule mikroprotsessori, mille jagamistehe töötab riistvara vea tõttu valesti. 1998 oli USA kogukahjum arvutivigadest 11 miljardit dollarit. 43% firmadest, keda on tabanud arvutisüsteemi andmekatastroof, pankrotistub.

Suurenev sõltuvus tehnoloogiast teeb inimese kergesti haavatavaks ja abituks olukordades, kus tehnika hakkab tõrkuma. Harilikult märkamegi seda sõltuvust alles siis, kui saabub õnnetus, milleks me valmis ei olnud.

Möödunud sajandi keskel eeskätt lennuki- ja raketitööstuse vajadusel kujunes välja inseneriteadus, mida nimetatakse *tehniliseks diagnostikaks*. Eriti suure arenguhoo sai see teadus arvutite ilmumisel tehismaailma, mille põhjuseks oli tehissüsteemide, s.t arvutite keerukuse kiire kasv. Paradoksaalne on, et keerukuse teooria loodi digitaalse tehismaailma jaoks, mis oma olemuselt on lõplik, aga mitte analoogmaailma jaoks, mille dimensiooniks on lõpmatus.

Submikrontehnoloogia ning juba käivitunud investeeringud nanotehnoloogiasse elektroonikas võimaldavad inseneridel luua järjest keerukamaid mikrokipe – integraalskeeme. Arvutid ja arvutivõrgud ühesainsas mikroskeemis on saanud tänapäeval tavapäraseks kõrgtehnoloogiliseks tooteks.

Mida keerukam on aga süsteem, seda kindlamini kehtib Murphy seadus: *kui miski võib untsu minna, siis ta ka läheb*. Keerukus tähendab rohkete vigade võimalusi nii süsteemide projekteerimisel, valmistamisel kui ka kasutamisel. Seega, mida keerukam on süsteem, seda olulisem on süsteemi testimine, diagnostika ja veakindlus. Kulutused elektroonikasüsteemide testimisele ulatuvad sageli kuni 70%-ni (!) kõikidest tootiskuludest tööstuses. Kuidas projekteerida süsteeme nii, et nad oleksid kergemini (odavamalt) ja paremini testitavad, see on uus inseneriprobleem süsteemitehnikas.

Elektroonika-, arvuti- ja süsteemiinseneride hariduse oluliseks koostisosaks on tänapäeval saanud teadmised ja oskused skeemide, seadmete ja süsteemide testimise ning diagnostika alal. Süsteemide usaldatavusega ja veakindlusega seotud distsipliin on hakatud tehnikaülikoolides õpetama üha rohkem ja põhjalikumalt.

Laiemaski mõttes tähendab tehniline diagnostika kui õppeaine intellektuaalset harjutust põhjuste ja tagajärgede mõtestamisel ning tal on seetõttu küllaltki oluline didaktiline tähendus insenerihariduse kujundamisel. *Diagnostika* on aine, mis õpetab esitama õigeid küsimusi, plaanima mõttekaid eksperimente ning analüüsima eksperimentide tulemusi. Tehissüsteemide maailm on rikas eri valdkondade iseärasuste poolest, aga tema ühiseks siduvaks jooneks on süsteemide loogiline keerukus. Seetõttu ongi just digitaalsüsteemid tehismaailma kõige iseloomulikumateks esindajateks tehnilise diagnostika probleemide uurimisel ja mõistmisel.

Viimastel aastatel on ilmunud rohkesti võõrkeelset õppe-, teadus- ja teatmealast kirjandust elektronskeemide, arvutite ja digitaalsüsteemide testimise ning diagnostika kohta.

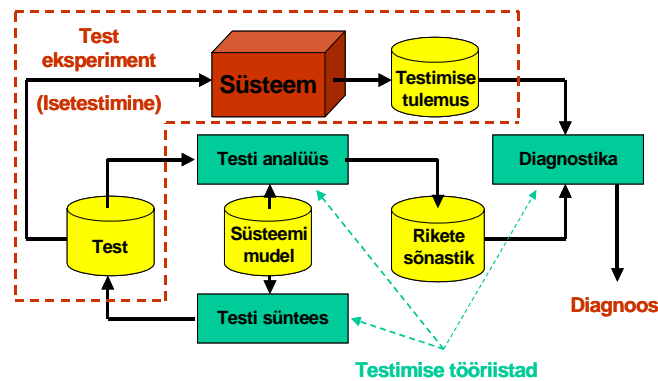
Selles raamatus tehakse valdkond eesti lugejale kättesaadavaks emakeeles.

Raamat on plaanitud mitmeosalisena. Esimeses osas käsitletakse digitaalsüsteemide diagnostilise modelleerimise teoreetilisi aluseid, et kujundada põhi diagnostikaprobleemide matemaatiliseks uurimiseks ning analüüsiks, nende lahendamise algoritmimiseks ja automatiseerimiseks. Järgmistes osades vaadeldakse diagnostikatestide sünteesi ja analüüsi probleeme, testprogrammide automaatse genereerimise meetodeid, rikete simuleerimist ja diagnoosi ehk rikete asukoha lokaliseerimist, süsteemide projekteerimise ja diagnostika omavahelisi seoseid, seda, kuidas luua hõlpsamini testitavaid ja diagnoositavaid süsteeme, kuidas projekteerida iseennast testitavaid seadmeid ja kuidas tagada süsteemide usaldatavus ning veakindlus.

## Sissejuhatus

Digitaalsüsteemide *tehnilise diagnostika* eesmärk on testimise abil kindlaks teha, kas süsteem on töökorras ja kui ta ei ole töökorras, siis lokaliseerida rikete asukoht nende kõrvaldamiseks. Laiemas mõttes tähendab *diagnostika* nii *testimist* kui rikete lokaliseerimist, kitsamas mõttes ainult *rikete lokaliseerimist* ehk diagnoosi.

Diagnostika tähtsamaid omavahel seotud ülesandeid ja nende lahendamiseks kasutatavaid tööriistu esitab ülevaatlikult joonis 1. Keskseks objektiks on siin testimisele ja diagnoosimisele kuuluv *tehniline süsteem*. *Testeksperiment* selle süsteemiga võidakse läbi viia kas spetsiaalse välise testimisseadme (automaatse *testri*) abil või süsteemi enda ressursside poolt, mida nimetatakse *isetestimiseks*.



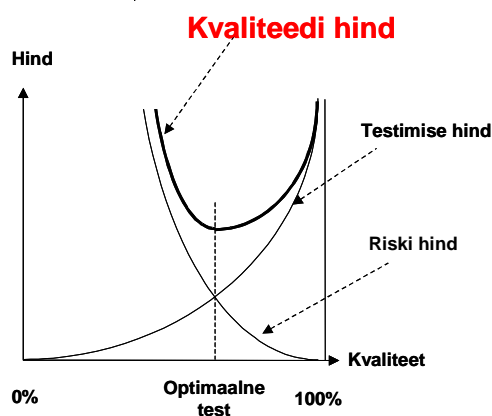
Joonis 1. Testimise tööriistad

Mõlemal testimise juhul on vaja rakendada *testi* ehk *testvektorite* jada, mis on eelnevalt genereeritud ja mida hoitakse testri mälus või



genereeritakse jooksvalt (ehk *on-line*) süsteemi sisseehitatud erivahendite poolt isetestimise käigus. *Testi kvaliteet* (avastatavate rikete protsent kõigi võimalike rikete suhtes) tehakse kindlaks *testi analüüsi* tööriistadega, milleks on harilikult tarkvaralised testide analüsaatorid ehk *rikete simulaatorid*. Nende abil konstrueeritakse ka *rikete sõnastik*, mida saab tarvitada tehnilise diagnoosi panemiseks. *Testide sünteesiks* kasutatakse väga keerulisi tarkvaratööriistu – testide generaatoreid.

Mida tähendab *testigeneraator*? See on programm, mis analüüsib sadadest tuhandetest loogikaelementidest koosnevat elektronskeemi (õigemini selle mudelit), määrab kindlaks kõik skeemis esineda võivad *rikked* – lühised ja katkestused elementide sees ning ühenduste vahel – ja seejärel hakkab iga sellise rikke jaoks otsima niisugust signaalide kogumit (*testi*), mille rakendamisel skeemi sisenditele avastataks nimetatud rike juhul, kui see skeemis ette tuleb. Kui skeem koosneks mõnest tuhandest elemendist, siis sünteesiks parimad generaatorid selle testi mõne minutiga. Mõnekümne tuhande elemendi puhul võib kuluda aega juba tunde ja edasi kasvava keerukuse puhul suureneks aeg juba eksponentsiaalselt. Reaalsed digitaalsüsteemid koosnevad miljonitest elementidest. Kuidas genereerida niisugustele süsteemidele teste, jäädes seejuures realistliku aja piiridesse?



*Joonis 2. Testimise ja riski maksumuse vahekord*

Tehnilise diagnostika filosoofiliseks põhiküsimuseks ongi – *kui palju testida* uurimise all olevat süsteemi olemaks kindel, et süsteem on töökorras. Teiseks tähtsaks, aga rohkem juba tehniliseks põhiküsimuseks on: kus asub rike, kui testimise teel on juba kindlaks tehtud, et süsteem ei

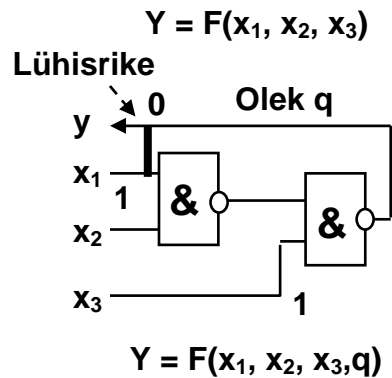
tööta. Mõlema probleemi lahendus koosneb kahest poolest – testide sünteesist (genereerimisest) ja testide analüüsist.

*Testimine* on tegelikult kvaliteedi ja selle tagamiseks vajalike kulutuste ehk raha dilemma. Mida rohkem ja põhjalikumalt testime, seda kindlamalt garanteerime süsteemi kvaliteedi (loe: *usaldusväirsuse*). Aga testimine tähendab kulutusi. Mida rohkem testime, seda rohkem kulutame aega ja ressursse ehk kokku võttes raha. Samas, kui ei testiks üldse või testiks liiga vähe, siis *riskiksime* vigade tekkimisega süsteemi töös võimalike avastamata jäänud rikete tõttu, mida lõppkokkuvõttes trahvide, kompensatsioonide jms näol tuleks kinni maksta. Nii testimisel kui *riskil* on hind. Kusagil on aga kesktee, nagu näitavad joonisel 2 olevad kõverad.

Kvaliteedi hinna kõveral on miinimum, mida ilmselt tulekski taotleda. See miinimum tähendab sisuliselt parimat kulutuste ja riski vahekorda.

Probleem on vaid selles, et täpseid kõveraid ja parimat suhet kulutuste ja riski vahel on raske välja arvutada.

Esimeseks raskuseks on testi *100%-lise kvaliteedi* määrang. Kuidas kindlaks teha, kas test annab 100%-lise garantii süsteemi töökorras oleku kohta? Üheks võimaluseks oleks läheneda probleemile terve mõistuse loogika abil: kui süsteem töötab, siis on ta ka töökorras. Aga kuidas kindlaks teha, et süsteem töötab?



*Joonis 3. Lühisriike digitaalskeemis*

Võtame näiteks 32-bitise *summaatori*. Selleks et kindlaks teha, kas summaator töötab õigesti, tuleks läbi summeerida

$$2^{64} = 18\,446\,700\,000\,000\,000\,000 \approx 10^{19}$$

arvupaari. Selle sooritamiseks kuluks isegi 1 GHz summeerimissageduse juures kolm sajandit (!). Aga isegi sellest ei piisa (!).

Kujutleme *lühisriket* joonisel 3 näidatud skeemi ühe sisendi ja väljundi vahel. 3-sisendilise *kombinatsioonskeemi*  $y = F(x_1, x_2, x_3)$  täielikuks testimiseks oleks vaja  $2^3 = 8$  testvektorit. Aga lühise tõttu pole enam tegemist kombinatsioonskeemiga. Rikke tulemusena on tekkinud triger ehk *järjestikskeem* (lõplik automaat) olekuga  $q$ , mille väljundfunktsioonil  $y = F(x_1, x_2, x_3, q)$  pole mitte 3 vaid on 4 argumenti. Seetõttu oleks täielikuks testimiseks vaja  $2^4 = 16$  testvektorit. Pöördudes tagasi summaatori juurde, tähendaks ühe oleku lisandumine *testimisaja* kahekordistumist ehk kolme sajandi asemel kuus sajandit. Kuna võimalike sekventsiaalsete lühiste arv võib suuremas skeemis olla küllalt suur ja kuna arvesse tuleks võtta ka lühiste suvalisi kombinatsioone, siis näemegi, et kõigi testimiseks vajalike testvektorite arvu ülemine piir kipub tegelikult kujunema lõpmata suureks.

Väljapääs olukorrast oleks loobuda niisuguse *funktsionaalse testimise* käsitlusest ja asendada see *struktuurse* lähenemisega. Näiteks 32-bitist summaatorit võib kirjeldada kombinatsioonskeemina, mis realiseerib 32 loogikafunktsiooni paari

$$s_i = \overline{a_i b_i c_i} \vee \overline{a_i b_i} c_i \vee \overline{a_i} b_i c_i \vee a_i b_i c_i$$

$$c_{i+1} = a_i b_i \vee a_i c_i \vee b_i c_i$$

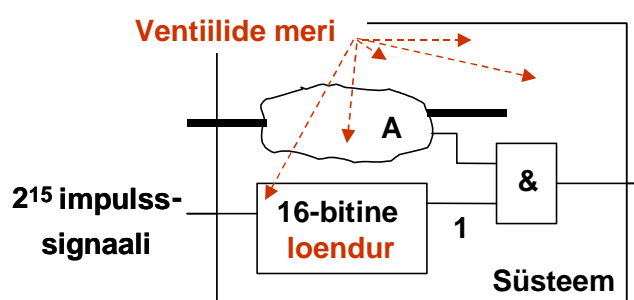
Kokku oleks niisuguses skeemis  $32 \cdot 3$  sisendit,  $32 \cdot 18$  sisendharu ja  $32 \cdot 7$  ühendust ehk kokku 896 juhet. Igal juhtmel võib kujutleda kahte võimalikku *loogilist riket* “konstant 0” ja “konstant 1”. Seega oleks kõikide rikete koguarv 1792. Iga niisuguse rikke avastamiseks oleks vaja leida üks testvektor ehk kahe summeeritava arvu paar. See oleks ühtlasi ka *testimise keerukuse* mõõt, mis oleks juba päris realistlik. Edaspidi näeme (raamatu teises osas), et tegelikult piisaks suvalise sõnapikkusega *järjestikülekandega* summaatori testimiseks ka ainult 8-st arvupaarist.

On lihtne näha, et struktuurse ehk *riketepõhise* lähenemisviisi puhul kujuneb ülesande keerukus mõõtmalt väiksemaks kui funktsionaalse lähenemise puhul. Ometigi pörkame ka siin kohe raskustele: skeemis võivad ju põhimõtteliselt tekkida ka *mitmekordsed* rikked. On lihtne arvutada, et  $n$ -juhtmelises skeemis on kõigi võimalike kordsete rikete arv  $3^n - 1$ , ehk 32-bitise summaatori puhul  $3^{1792} - 1$ . Iga juhe võib olla kolmes võimalikus

olekus: “konstant 0”, “konstant 1” või “korras”. Rikete loetlemisel ei tule arvesse olukord, kus kõik juhtmed on korras. Seega viib ka struktuurse lähenemise korral täiuslikkuse taotlemine meid keerukuse lõksu ja praktiliste lahenduste leidmiseks tuleb ülesannet mõistlikult lihtsustada. Üheks niisuguseks võimaluseks oleks eeldus, et skeemis võib alati korraga esineda ainult üks rike. Sellise eelduse põhjenduseks on, et kordsete rikete tekkimise tõenäosus on väike: kui ühe rikke tõenäosus oleks näiteks 1 tuhandik, siis kahe üheaegse rikke tõenäosuseks oleks 1 miljondik. Nii või teisiti on selline eeldus ikkagi kompromiss reaalsete testimisvõimaluste ja saavutatava testimiskvaliteedi vahel.

Kohe kerkivad aga uued probleemid. Konstantsete rikete mudel loogikaskeemide tasemel ei ole piisavalt täpne esindamiseks reaalseid füüsilisi defekte transistoriskeemide tasandil. Kui asendame aga analüüsi eesmärgil loogikaskeemid transistoriskeemidega, sattume uuesti keerukuse kiire kasvu umbteele.

Nii jõuamegi digitaalsüsteemide diagnostikas uue üldisema põhiküsimuse juurde: kuidas tagada süsteemide üha kasvava keerukuse puhul testimise kvaliteeti.

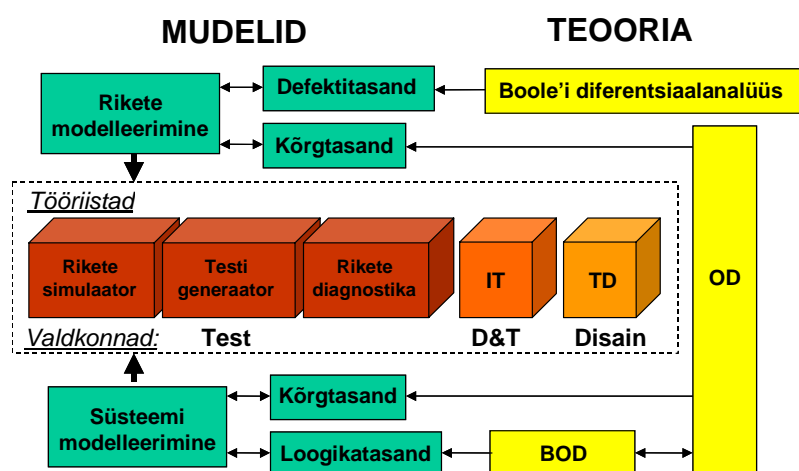


Joonis 4. Hierarhiline süsteemi modelleerimine

Keerukusega saaks võidelda, kui süsteemi analüüsi (testide sünteesi ja rikete simuleerimist) viia läbi kõrgematel tasanditel väiksema detailsusastmega mudelitel. Samal ajal, nagu eeltoodust juba järeldada võis, kaotaksime niiviisi analüüsi täpsuses ega saaks tagada testide kvaliteedi adekvaatset hindamist. Teisalt saaks täpsust tagada küll, kui viia testide analüüs läbi, modelleerides süsteemi defektipõhiselt võimalikult madalal näiteks transistoriskeemide tasandil. See aga suurendaks taas uurimisobjekti keerukust ega oleks praktiliselt kasutatav meetod.

Väljapääsuks ummikust oleks keerukate digitaalsüsteemide testimisel *hierarhiline* lähenemisviis.

Joonisel 4 on *digitaalsüsteem*, mida võib käsitleda kui struktuuritult *ventiilide merd* (*sea of gates*) või siis hierarhiliselt, kus üksikutele ventiilide piirkondadele on seatud vastavusse kõrgema tasandi funktsioonid. Näiteks ühele ventiilide grupile on joonisel 4 vastavusse seatud 16-bitise *loenduri* funktsioon. Oletame, et on vaja genereerida test ploki A võimalike rikete avastamiseks süsteemi väljundis. Selleks et rikete tõttu tekkivad veasignaalid leviksid süsteemi väljundisse, kus neid avastada saaks, tuleks JAA-elementi alumisele sisendile saata signaal 1. Kui me ei teaks, et selle sisendiga ühenduses olev skeem on loendur ja seda skeemi tuleks käsitleda kui „ventiilide merd”, oleks loogikameetodite abil väga raske tuletada niisugune sisendsignaalide jada, mis tekitaks meil JAA-elementi sisendis vajaliku signaali 1. Taoline tuletamine võtaks ka väga võimsal arvutil aega minuteid kui mitte tunde. Teades aga, et nimetatud ventiilide grupp on 16-järguline loendur, oleks ka ilma arvutusteta selge, et signaali 1 saamiseks loenduri väljundis on vaja selle sisendile rakendada  $2^{15}$  impulsist koosnevat signaalide jada.



Joonis 5. Diagnostikaprobleemide üldplaan

Hierarhiline lähenemisviis tähendab seda, et osa süsteemi analüüsi viiakse läbi komponentide tasemel, mis on vähem keerukad ja mis seetõttu võimaldavad töötada madalamatel ja detailsematel (ventiilide või isegi

transistoride) tasanditel. Nii õnnestuks adekvaatselt käsitleda füüsilikali *defekte* ja leida tingimusi, kuidas neid aktiveerida, nii et nende mõjusid ka kõrgematel tasanditel jälgida saaks. Defektidest tingitud veasignaalide analüüs saaks seejärel toimuda juba pealiskaudsemaid kõrgema tasandi mudeleid kasutades.

Toodust tuleneb, et keerukate süsteemide diagnostikaprobleemide mõistmiseks ja nende lahendamiseks tuleb tunda mitmesuguseid süsteemide eri tasanditel töötavaid matemaatilisi teooriaid, mudeleid ja meetodeid.

Joonisel 5 on digitaalsüsteemide *diagnostika probleemide* üldpilaan. Probleemid võib jagada kolme valdkonda, mis on omavahel läbi põimunud: *testimine*, *disain* ja *test* ning *testikõlblik disain*. *Testi* valdkond ühendab testide sünteesi ja analüüsi probleeme, mida võiks rühmitada kui rikete simuleerimine, testide genereerimine ja diagnoos. *Disaini ja testi* (D&T) valdkonda kuuluvad küsimused, kuidas projekteerida (disainida) isetestivaid süsteeme (ITS). *Disaini ja testi* probleemid on siin tihedalt läbi põimunud. *Testikõlblik disain* (TD) aga haarab probleeme, kuidas projekteerida süsteemi nii, et see oleks võimalikult hästi, kvaliteetselt ja kergesti testitav.

Probleemide lahendus seisneb vajalike tööriistade väljatöötamises, valmis ehitamises ja tulemuslikus rakendamises. Tööriistade all mõtleme siin eritarkvara (rikete simulaatorid, testigeneraatorid, rikete diagnoosi ja lokaliseerimise tarkvara). Tööriistade väljatöötamise aluseks on meetodid ja algoritmid, mis töötavad süsteemi ehk diagnostikaobjekti matemaatilistel mudelitel. Hierarhilise lähenemisviisi realiseerimiseks peab meil olema nii madala kui ka kõrgtasandi mudeleid. Algoritmide väljatöötamiseks eri tasanditel ja eri mudelitel tuleb kasutada mitmesuguseid teooriaid.

Paljude võimalike teoreetiliste aparatuuride hulgast on käesolevas raamatus põhjalikumaks käsitlemiseks välja valitud kaks: *Boole'i diferentsiaalalgebra* ja *otsustusdiagrammid*, mis kokku võimaldavad anda ühtse ja kompaktse käsitluse põhilistest digitaalsüsteemide diagnostika probleemidest. Boole'i diferentsiaalaparaat võimaldab siduda kaks põhilist diagnostikaprobleemi – testide sünteesi ja rikete diagnoosi ühteainsasse Boole'i diferentsiaalvõrrandisse. Boole'i tuletiste abil osutub võimalikuks ka defektide kujutamine füüsilikatasandilt loogikatasandile, mis oleks esimene samm defektide hierarhilisel modelleerimisel. *Binaarsete otsustusdiagrammide* (BOD) abil õnnestub kompaktselt esitada põhilisi testide sünteesi ja analüüsi algoritme. Loobudes aga binaarsuse kitsendusest, õnnestub neid algoritme otsustusdiagrammide (OD) üldjuhu abil üldistada digitaalsüsteemide kõrgematele tasanditele.

## 1. Digitaalsüsteemide diagnostiline modelleerimine

Digitaalsüsteemide *diagnostilise modelleerimise* eesmärk on analüüsida süsteemide käitumist erinevate rikete korral, seletada põhjusti, miks süsteem käitub konkreetses olukorras ebanormaalselt, genereerida teste rikete avastamiseks, analüüsida testide võimet avastada rikkeid ja lahendada palju muid spetsiifilisi süsteemide diagnostika ülesandeid. Raskeks muudab nimetatud probleemide lahendamise digitaalsüsteemide kasvav keerukus. Väljapääsuks olukorrast on hierarhiline lähenemisviis meetodil *jaga ja valitse (divide and conquer)*, mis eeldab seda, et diagnostiline modelleerimine peaks olema võimalik nii kõrgemal näiteks arvuti käsusüsteemi tasandil kui ka madalamal näiteks ventiil- või koguni transistoriskeemide tasandil.

Digitaalsüsteemide diagnostikaprobleemid on võrdlemisi hästi läbi töötatud loogikaskeemide tasandil, Formaalseks matemaatiliseks aparaadiks sobib siin kõige paremini Boole'i diferentsiaalaparaat, mida traditsioonilistes *diskreetse matemaatika* või *arvutiteooria* kursustes harilikult ei käsitleta. Nimetatud aparaadist arusaamine aitab paremini mõista diagnostikaprobleemide kompleksi kui tervikut. Siiski pole sellest teoriast suurt kasu süsteemide uurimiseks kõrgematel tasanditel, kus bittide ja loogikatehete asemel kasutatakse mõisteid nagu sõnad, mikroprogrammid, masinakäsud. Sobivaks modelleerimise vahendiks süsteemide diagnostilisel analüüsil kõrgematel tasanditel on otsustusdiagrammid, mille erijuhuks on binaarsed otsustusdiagrammid, mis võimaldavad diagnostilist analüüsi loogikatasemel.

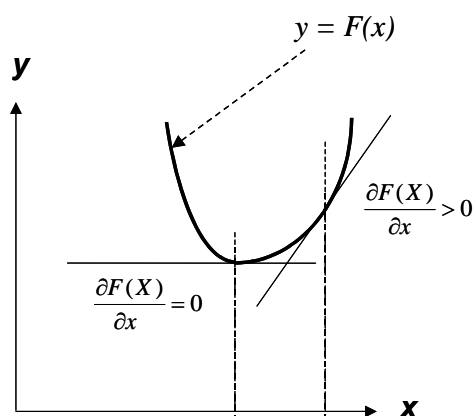
Käesolevas peatükis käsitlemegi nimetatud kahte modelleerimisviisi – Boole'i diferentsiaalaparaati ja otsustusdiagramme.

## 1.1. Boole'i diferentsiaalaparaat

*Boole'i diferentsiaalaparaati* käsitletakse traditsioonilistes diskreetse matemaatika kursustes suhteliselt harva. Põhjalikumalt on seda aparaati käsitletud töödes [Ake 59, Tha 71, Boc 75, Boc 89]. Peamiseks Boole'i diferentsiaalarvutuse rakendusala on olnud digitaalskeemide uurimine dünaamikas. Selles õppevahendis kasutame Boole'i diferentsiaalaparaati digitaalskeemide diagnostiliseks modelleerimiseks.

### 1.1.1. Boole'i osatuletised

Matemaatilises analüüsis mõistetakse funktsiooni  $y = F(x)$  tuletise  $dy/dx$  all selle funktsiooni juurdekasvu ja argumendi juurdekasvu suhte piirväärtust argumendi juurdekasvu lähenemisel nullile. Geomeetrilise interpretatsiooni järgi on tuletis  $dy/dx$  funktsiooni graafiku  $dy/dx$  puutuja tõus punktis, mille abstsiss on  $x$ . Tuletist võib interpreteerida ka funktsiooni muutumise kiirusena. Joonisel 1-1 esitatud funktsiooni näitel on selge, et funktsiooni tuletisel võib põhimõtteliselt olla igasuguseid väärtusi, alates miinus lõpmatuseni kuni pluss lõpmatuseni.



Joonis 1-1. Funktsiooni tuletise geomeetiline interpretatsioon



Samasuguse tuletise mõiste võib sisse viia ka Boole'i funktsioonide klassi jaoks. Kuna aga *Boole'i funktsiooni*  $y = F(x)$  puhul nii  $y$  kui ka  $x$  võivad omada vaid kahte väärtust 0 ja 1, siis ei saa me enam rääkida funktsiooni muutumise kiirusest, vaid võime rääkida vaid sellest, kas funktsiooni väärtus muutub argumenti väärtuse muutudes või ei muutu. Seega saab ka *Boole'i funktsiooni tuletise* väärtus omada vaid kahte väärtust 0 ja 1, kusjuures 0 vastaks sellele, et  $y$ -i väärtus ei muutu  $x$ -i väärtuse muutudes ja, vastupidi, 1 vastaks sellele, et  $y$ -i väärtus muutub, kui  $x$ -i väärtus muutub.

Kuna Boole'i funktsioonid on tüüpiliselt  $n$ -muutuja funktsioonid  $y = F(X)$ , kus  $X = (x_1, x_2, \dots, x_n)$ , ja  $n \geq 1$ , siis tuleb siin kasutada *Boole'i osatuletise* mõistet, mis kirjeldab  $y$ -i muutumist mingi konkreetse  $x_i \in X$  muutumise suhtes. Niisugust muutumist kirjeldab funktsioon

$$\frac{\partial F(X)}{\partial x_i} = F(x_1, \dots, x_i, \dots, x_n) \oplus F(x_1, \dots, \bar{x}_i, \dots, x_n) \quad (1-1)$$

ehk lihtsamal kujul

$$\frac{\partial F(X)}{\partial x_i} = F(x_1, \dots, x_i = 0, \dots, x_n) \oplus F(x_1, \dots, x_i = 1, \dots, x_n) \quad (1-2)$$

Boole'i funktsioon (1-2) leiab suurepärase rakenduse digitaalskeemide diagnostikas. Esitagu meil funktsioon  $y = F(X) = F(x_1, x_2, \dots, x_n)$  kombinatsioonskeemi, millel on  $n$  sisendit. Olgu sellel skeemil sisend  $x_i$  katki, nii et signaal seal kunagi läbi ei lähe. Matemaatiliselt võiks see tähendada näiteks, et  $x_i \equiv 0^1$ . Kui selles sisendis rakendada nüüd signaali  $x_i = 1$ , siis rikke tõttu leiaks seal aset signaali  $x_k$  väärtuse muutus  $1 \rightarrow 0$ . Niisuguse rikke avastamiseks skeemi väljundis  $y$  huvitab meid olukord, kus ka  $y$  muutuks  $x_i$  muutudes. Lähtudes eeltoodud Boole'i osatuletise interpretatsioonist, saaksime taolist olukorda kirjeldada võrrandi

$$\frac{\partial F(X)}{\partial x_i} = 1 \quad (1-3)$$

lahendite abil. Iga diferentsiaalvõrrandi (1-3) lahend oleks testiks (ehk testvektoriks), mis avastab rikke  $x_i \equiv 0$  funktsiooniga  $y = F(X)$  esitatud kombinatsioonskeemi sisendis. Võrrandi lahendamiseks tuleks ta vasak pool

---

<sup>1</sup> Sõltuvalt tehnoloogiast võib see mõnikord tähendada ka teist olukorda, nimelt et  $x_i \equiv 1$ . Konkreetselt käsitleme seda probleemi edaspidi (vt. 2.1.4).

teisendada *disjunktivseks normaalkujuks*. Iga konjuktsioon annaks siis ühe võimaliku lahendi.

*Näide 1-1*

Vaatleme kombinatsioonskeemi joonisel 1-2.

Skeemi funktsiooniks on  $y = x_1x_2 \vee x_3x_4$ . Valemit (1-2) kasutades leiaksime võrrandi

$$\frac{\partial y}{\partial x_1} = x_3x_4 \oplus (x_2 \vee x_3x_4) = 1 \quad (1-4)$$

Võrrandit (1-4) disjunktivsele normaalkujule (DNK) teisendades saaksime

$$\frac{\partial y}{\partial x_1} = x_2\overline{x_3} \vee x_2\overline{x_4} = 1 \quad (1-5)$$

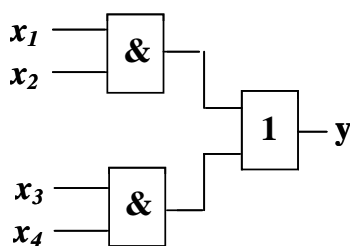
mis annaks kaks võimalikku testvektorit sisendi  $x_1$  testimiseks

$$x_1x_2x_3x_4 = -10-$$

ja

$$x_1x_2x_3x_4 = -1-0$$

kus “-” tähendab, et väärtus jääb määramatuks (ehk pole oluline).



*Joonis 1-2. Kombinatsioonskeem*

Huvitav on märkida, et mõlema lahendi puhul jääb testitav muutuja  $x_1$  ise määramatuks, mis peabki nii olema. Kuna testime just muutujat  $x_1$ , siis

peamegi talle põhimõtteliselt lubama mõlemat võimalikku väärtust: õiget ja väära (rikke korral).

### 1.1.2. Boole'i osatuletiste omadusi

Näitest 1-1 ilmneb, et võrrandi (1-4) teisendamine DNK-ks pole sugugi lihtne. Teisenduse lihtsustamiseks vaatame mõningaid Boole'i osatuletiste omadusi.

1. Olgu funktsioon  $P(X)$  esitatav kujuna  $P(X) = F(X) \wedge G(X)$ , kusjuures osaavaldis  $F(X)$  on sõltumatu muutujast  $x_i \in X$ . Siis on diferentseeritavat avaldist võimalik lihtsustada järgneva teisenduse abil:

$$\frac{\partial [F(X)G(X)]}{\partial x_i} = F(X) \frac{\partial G(X)}{\partial x_i} \quad (1-6)$$

2. Olgu funktsioon  $P(X)$  esitatav kujuna  $P(X) = F(X) \vee G(X)$ , kusjuures  $F(X)$  on sõltumatu muutujast  $x_i \in X$ . Siis on diferentseeritavat avaldist võimalik lihtsustada järgnevalt:

$$\frac{\partial [F(X) + G(X)]}{\partial x_i} = \frac{\partial G(X)}{\partial x_i} \quad (1-7)$$

Üsna endastmõistetavad on ka järgmised kaks omadust:

$$\frac{\partial \overline{F(X)}}{\partial x_i} = - \frac{\partial F(X)}{\partial x_i} \quad (1-8)$$

$$\frac{\partial \overline{\overline{F(X)}}}{\partial x_i} = \frac{\partial F(X)}{\partial x_i} \quad (1-9)$$

Kasutades esitatud omadusi, on võimalik üsna märgatavalt lihtsustada ka päris keerukate Boole'i avaldiste diferentseerimist.

*Näide 1-2*

Olgu vaja diferentseerida muutuja  $x_5$  järgi järgmist avaldist:

$$y = x_1 x_2 \vee x_3 (\overline{x_2 x_4} \vee \overline{x_1} (x_4 \vee (\overline{x_5} \vee \overline{x_2 x_6}))) \vee \overline{x_1 x_3} \quad (1-10)$$

Kuna osaavaldis

$$F(X) = x_1 x_2 \vee x_1 \overline{x_3}$$

ei sõltu muutujast  $x_5$ , siis võime selle diferentsiaali märgi alt välja viia, lihtsustades diferentseeritavat osa. Kasutades vaheldumisi teisendusi (1-6) ja (1-7) saaksime

$$\begin{aligned} \frac{\partial y}{\partial x_5} &= \frac{\partial (x_1 x_2 \vee x_1 \overline{x_3})}{\partial x_5} = \frac{\partial (x_1 x_2 \vee x_1 (x_4 \vee (x_5 \vee x_2 x_6)))}{\partial x_5} = \\ &= \frac{\partial (x_1 x_2 \vee x_1 x_3 x_2 x_4)}{\partial x_5} = \\ &= \frac{\partial (x_1 (x_4 \vee (x_5 \vee x_2 x_6)))}{\partial x_5} = \\ &= \frac{\partial (x_4 \vee (x_5 \vee x_2 x_6))}{\partial x_5} = \\ &= \frac{\partial x_5}{\partial x_5} = 1 \end{aligned}$$

Kuna  $\partial x_5 / \partial x_5 = 1$ , siis olemegi lahti saanud tülkast operatsioonist  $\oplus$  ja testide leidmine avaldisele (1-10) muutuja  $x_5$  testimiseks taandub nüüd juba DeMorgani teisendustele ning sulgude avamisele

$$\begin{aligned} \frac{\partial y}{\partial x_5} &= (x_1 x_2 \vee x_1 \overline{x_3}) x_3 (x_2 x_4) x_1 x_4 (x_2 x_6) \frac{\partial x_5}{\partial x_5} = \\ &= (\overline{x_1} \vee \overline{x_2})(\overline{x_1} \vee x_3) x_3 (x_2 \vee \overline{x_4}) \overline{x_1} x_4 (x_2 \vee \overline{x_6}) = \\ &= \overline{x_1} x_4 x_3 x_2 \vee \dots = 1 \end{aligned}$$

Üheks võimalikuks testvektoriks leiame  $x_1 x_2 x_3 x_4 x_5 = 0110-$ . Olgu märgitud, et testide genereerimisel praktilisest küljest lähtudes, ei olegi vaja tervet disjunktiivset normaalkuju kätte saada kõikide lahendite leidmiseks, vaid piisab ka ühestainsast testvektorist.

### 1.1.3. Boole'i vektortuletised

Paljudel juhtudel on vaja analüüsida Boole'i funktsiooni tundlikkust mitme muutja väärtuse üheaegse muutumise suhtes. Niisugusteks juhtudeks on näiteks

- mitme rikke üheaegne olemasolu skeemis või
- ühest ja samast rikkest tuleneva vigase signaali levimine skeemis mitme hargneva signaalitee kaudu.

Niisugustel juhtudel võib testvektorite leidmiseks kasutada *vektortuletisi*, kus uuritakse funktsiooni tundlikkust korraga mitme argumenti väärtuse muutumise suhtes. Näiteks kahe muutuja  $x_i$  ja  $x_j$  muutumise korral saaksime vektortuletise defineerida järgmiselt

$$\frac{\partial F(X)}{\partial(x_i, x_j)} = F(x_1, \dots, x_i, x_j, \dots, x_n) \oplus F(x_1, \dots, \bar{x}_i, \bar{x}_j, \dots, x_n) \quad (1-11)$$

*Näide 1-3*

Vaatleme taas skeemi joonisel 1-2, mille funktsiooniks on  $y = x_1 x_2 \vee x_3 x_4$ . Olgu korraga vigased nii sisend  $x_2$  kui ka  $x_3$ . Kahe rikke põhjuseks võib olla ka üksainus rike skeemis, millest tingitud veasignaali levib mööda kahte haru. Testvektori niisuguse kordse vea avastamiseks leiame võrrandist

$$\frac{\partial F(X)}{\partial(x_2, x_3)} = x_1 \bar{x}_4 \vee \bar{x}_1 x_4 \vee x_1 x_4 (\bar{x}_2 \bar{x}_3 \vee x_2 x_3) = 1$$

Võrrandil on neli lahendit:

- 1) 1--0
- 2) 0--1
- 3) 1001
- 4) 1111

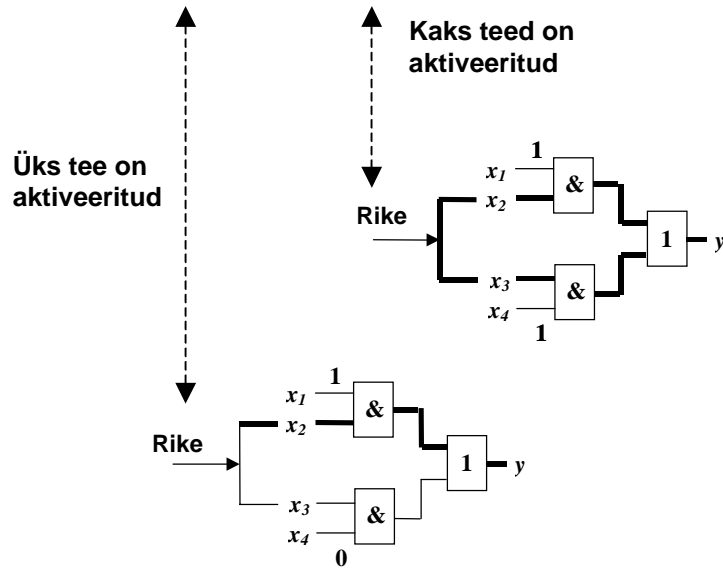
Lahenditel on ka oma kindel füüsikaline tähendus (joonis 1-3).

Esimene lahend 1--0 lubab levida ühelainsal veasignaali  $x_2$  läbi skeemi, blokeerides teise veasignaali  $x_3$  levi. Sama omadus on ka teisel lahendil, kusjuures nüüd levib viga vaid sisendist  $x_3$ . Mõlemal juhul võivad signaalide muutused sisendites  $x_2$  ja  $x_3$  olla suvalised  $0 \rightarrow 1$  või  $1 \rightarrow 0$  (kokku 4 võimalust).

Kolmanda ja neljanda lahendi puhul on aga vea levikuks aktiveeritud mõlemad harud, kusjuures mitte kõik vigade kombinatsioonid pole enam

avastatavad: väljundisse levib viga vaid siis, kui mõlemad veasignaaliid on ühesuunalised. Kui veasignaaliid muutuvad vastupidises suunas, siis nad kompenseerivad teineteist ja vead jäävad avastamata (vead *maskeeruvad*).

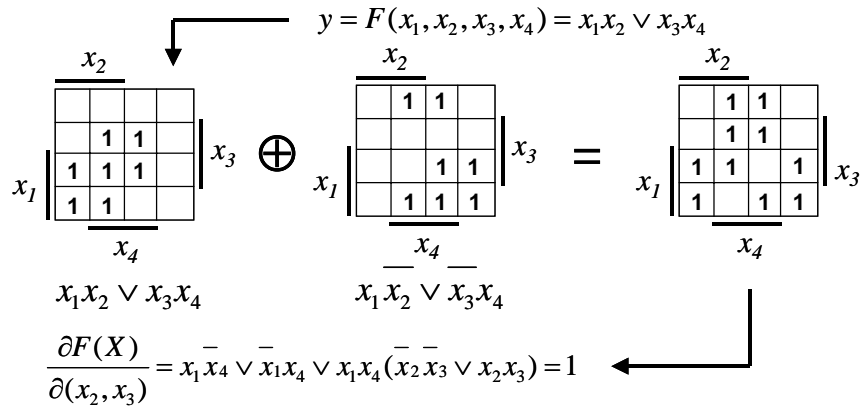
$$\frac{\partial F(X)}{\partial(x_2, x_3)} = \overline{x_1}x_4 \vee x_1\overline{x_4} \vee \overline{x_1}\overline{x_4}(x_2x_3 \vee x_2\overline{x_3}) = 1$$



*Joonis 1-3. Vektortuletise füüsikaline interpretatsioon*

Näites 1-3 vaadeldud olukord illustreerib ühte keerukamat probleemi testide genereerimisel. Kui *veasignaaliid* levib mitut haru pidi, siis oleks vaja tegeleda maskeerumise kontrolliga, mis teeb testide genereerimise algoritmi keerukamaks ja aeglasemaks. Algoritm lihtsustuks, kui lubaksime veasignaaliid levida alati vaid ühte haru pidi, blokeerides teised harud. Samas aga, kui nii viisi teeksime, sünnib oht välistada testi leidmine juhul, kui üksnes mitme haru kaudu veasignaaliid levitav test eksisteerib.

Nn lihtsustatud testide genereerimise algoritmid, mis aktiveerivad vaid üksikuid teid, korreleeruvad hästi ka faktiga, et Boole'i vektortuletisi arvutada on keerukas. Üheks võimaluseks neid arvutada on *Carnaugh' kaartide* kasutamine (joonis 1-4), mis aga sobib vaid väikese muutujate arvu korral.



Joonis 1-4. Vektortuletiste arvutamine Carnaugh' kaardi abil

### 1.1.4. Kordsed Boole'i osatuletised

Olgu antud Boole'i funktsioon  $y = F(X) = F(x_1, x_2, \dots, x_n)$ . Kordne tuletis kahe muutuja  $x_i$  ja  $x_j$  järgi analoogiliselt tavalise matemaatilise analüüsiga on defineeritud järgmiselt:

$$\frac{\partial^2 y}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left( \frac{\partial y}{\partial x_j} \right) \quad (1-12)$$

Kordset Boole'i osatuletist saab kasutada kordsete rikete analüüsiks, näiteks testide genereerimiseks kordsele rikkele. Olgu skeemis, millele vastab funktsioon  $y = F(x_1, x_2, \dots, x_n)$ , vigased kaks sisendit  $x_i$  ja  $x_j$ . Võiks püstitada ülesande genereerida test, mis avastaks ühe kahest rikkest, näiteks rikke sisendis  $x_j$ , tingimusel, et teine rike sisendis  $x_i$  ei maskeeriks esimest.

Rikke avastamise tingimus sisendis  $x_j$  oleks

$$\frac{\partial y}{\partial x_j} = F_1(X) = 1 \quad (1-13)$$

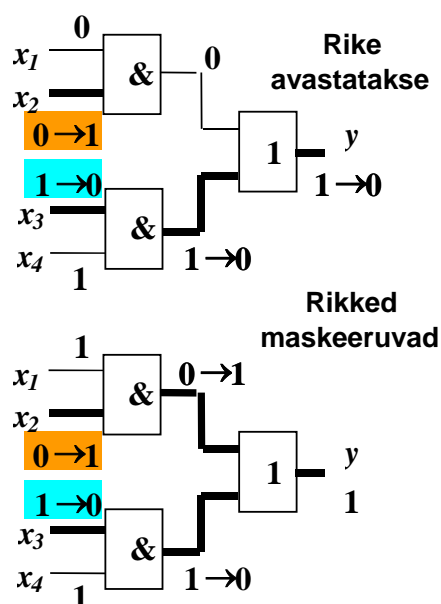
Selleks et viga sisendis  $x_i$  ei saaks mõjutada tingimust (1-13), s.t maskeerida riket sisendis  $x_j$ , tuleks lahendada võrrand

$$\frac{\partial F_1(X)}{\partial x_i} = 0 \quad (1-14)$$

Ehk teiste sõnadega tuleks lahendada võrrandisüsteem

$$\frac{\partial^2 y}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left( \frac{\partial y}{\partial x_j} \right) = 0 \quad (1-15)$$

$$\frac{\partial y}{\partial x_j} = 1$$



Joonis 1-5. Kordse tuletise kasutamine rikete maskeerumise vältimiseks



#### Näide 1-4

Olgu antud kombinatsioonskeem, millele vastab funktsioon  $y = x_1x_2 \vee x_3x_4$  ja kus on korraga vigased nii sisend  $x_2$  kui ka  $x_3$ . Joonisel 1-5 on illustreeritud kahte juhtu. Ülemisel skeemil illustreeritakse testi  $x_1x_2x_3x_4 = 0011$ , mille saame saame, kui lahendame võrrandisüsteemi (1-15)

$$\frac{\partial^2 y}{\partial x_2 \partial x_3} = \frac{\partial}{\partial x_2} \left( \frac{\partial y}{\partial x_3} \right) = x_1x_4 = 0$$
$$\frac{\partial y}{\partial x_3} = \overline{x_1x_4} \vee \overline{x_2x_4} = 1$$

Alumisel skeemil aga illustreeritakse olukorda, kus on küll genereeritud test  $x_1x_2x_3x_4 = 1011$  rikkele sisendis  $x_3$  võrrandi (1-13) lahendamisega

$$\frac{\partial y}{\partial x_3} = \overline{x_1x_4} \vee \overline{x_2x_4} = 1 \longrightarrow \overline{x_2x_4} = 1$$

aga rike jääb maskeerituks teise rikke tõttu sisendis  $x_2$ , kuna pole täidetud tingimust (1-14).

### 1.1.5. Liitfunktsioonide diferentseerimine

Keerukamate digitaalskeemide puhul ei ole otstarbekas neid esitada väljundite Boole'i funktsioonide süsteemina kahel põhjusel:

- väljundfunktsioonide keerukuse tõttu muutuvad ka Boole'i diferentsiaalvõrrandid liiga keeruliseks;
- diagnostika vaatenurgast ei piisa ainult sisendite testimisest, vaid vaja on testida ka skeemi sisepunkte.

Seetõttu esitatakse digitaalskeeme enamasti komponentide (või alamskeemide) võrguna, kus igale komponendile (või alamskeemile) vastab mingi lihtsam Boole'i funktsioon. Väljundfunktsioone saab sellisel juhul vaadelda liitfunktsioonidena, mille argumentideks võivad olla nii sisendmuutujad kui ka skeemi sisepunktidele vastavad funktsioonid.

Vaatleme skeemi joonisel 1-6, mis on esitatud *liitfunktsioonina*

$$y = F(X) = F(X, x_k) = F(X, F_k(X_k))$$

kus

$$x_k = F_k(X_k)$$

$$X_k \subseteq X$$

Niisuguse funktsiooni tuletis mingi muutuja  $x_i \in X_k$  järgi võib avalduda Boole'i funktsioonide korrutisena

$$\frac{\partial y}{\partial x_i} = \frac{\partial F(X, F_k(X_k))}{\partial x_i} = \frac{\partial F(X, x_k)}{\partial x_i} = \frac{\partial y}{\partial x_k} \frac{\partial F_k(X_k)}{\partial x_i} = \frac{\partial y}{\partial x_k} \frac{\partial x_k}{\partial x_i} \quad (1-16)$$

Üldjuhul, kui liitfunktsioon on esitatud *superpositsioonide* ahelana

$$y = F_k(F_{k-1}(\dots F_2(x_1, X), \dots, X)) \quad (1-17)$$

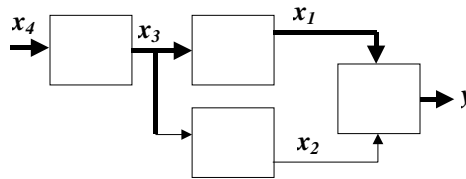
saame korrutise

$$\frac{\partial y}{\partial x_1} = \frac{\partial F_k}{\partial F_{k-1}} \frac{\partial F_{k-1}}{\partial F_{k-2}} \dots \frac{\partial F_2}{\partial x_1} \quad (1-18)$$

Olgu märgitud, et niisugune *ahelreegel* liitfunktsiooni osatuletise esitamiseks on lihtsustatud ja kehtib ainult erijuhul, mis diagnostika seisukohalt vastab sellele, et veasignaali levib vaid ühte teed pidi.

*Näide 1-5*

Vaatleme skeemi joonisel 1-6, kus eraldi on välja toodud neli olulist muutujat. Teisi muutujaid hulgast  $X$  pole näidatud.



**Joonis 1-6.** Testide genereerimine liitfunktsiooni osatuletiste abil

Arvutades tuletist  $\partial y/\partial x_4$ , võime kasutada liitfunktsiooni tuletise arvutamise ahelreeglit

$$\frac{\partial y}{\partial x_4} = \frac{\partial y}{\partial x_1} \frac{\partial x_1}{\partial x_3} \frac{\partial x_3}{\partial x_4}$$

aga ainult sel juhul, kui veasignaali levik paralleelselt teiste harude kaudu on blokeeritud. Selleks võib kasutada kahte võimalikku lisatingimust:

- 1) kordsete tuletiste põhimõtet

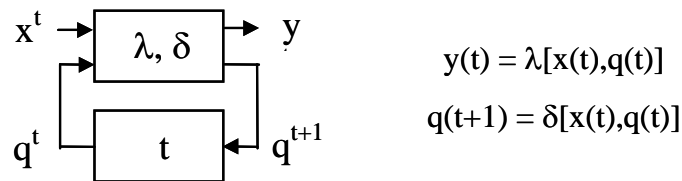
$$\frac{\partial y}{\partial x_2} \left( \frac{\partial y}{\partial x_1} \right) = 0$$

- 2) või veasignaali leviku blokeerimist juba enne komponenti  $y = F(x_1, x_2, X)$

$$\frac{\partial x_2}{\partial x_3} = 0$$

### 1.1.6. Boole'i osatuletised järjestikскеemidele

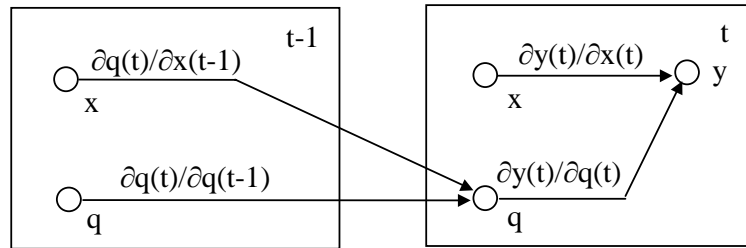
Järjestikскеemide esitamiseks kasutatakse *Huffmanni mudelit* (joonis 1-7), millele vastab kaks tüüpi funktsioone: *väljundfunktsioon* ja *siirdefunktsioon*.



Joonis 1-7. Järjestikскеemide mudel

Mõlemad funktsioonid on tavalised Boole'i funktsioonid, kus aeg  $t$  on parameetrik ja seob konkreetseid muutujate väärtusi erinevate taktidega. Nii tulebki vaadelda ühte muutujat  $x(t)$  parameetri  $t$  eri väärtuste korral  $t_1$  ja  $t_2$  sisuliselt kui kahte erinevat muutujat  $x(t_1)$  ja  $x(t_2)$ . Öeldut arvesse võttes toimub niisuguste *ajast sõltuvate Boole'i funktsioonide* diferentseerimine

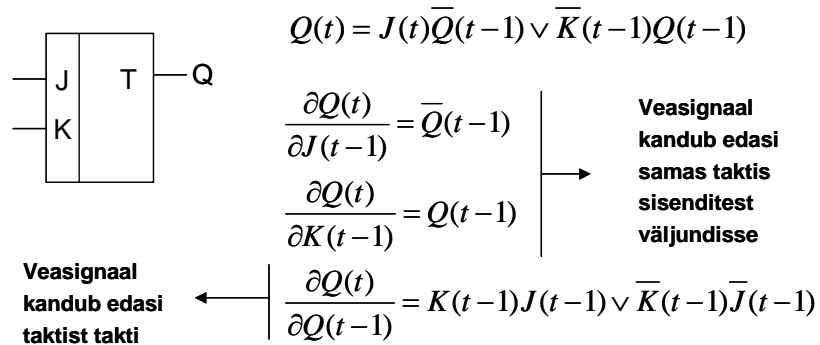
sarnaselt ajast sõltumatute Boole'i funktsioonidega. Joonisel 1-8 on illustreeritud Huffmanni mudelist tulenevat nelja Boole'i osatuletise tüüpi.



*Joonis 1-8. Järjestikskemide osatuletiste tüübid*

*Näide 1-6*

Joonisel 1-9 on *JK-trigeri* funktsioon ja sellest tulenevad Boole'i osatuletised.

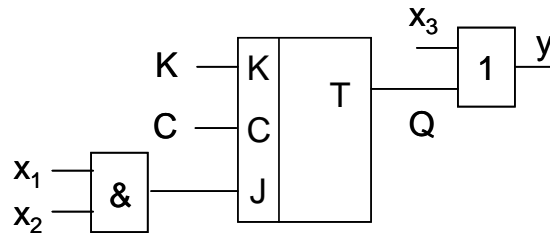


*Joonis 1-9. JK-trigeri Boole'i osatuletised*

Kui seni vaatlesime Boole'i diferentsiaalvõrrandite kasutamist üksikute testvektorite genereerimiseks, siis ajafunktsioonide diferentseerimise eripäraks on võimalus formaliseerida järjestikskemide *testjadade* genereerimist.

Näide 1-7

Genereerime testi joonisel 1-10 esitatud järjestikskeemi sisendile  $x_2$ , kasutades Boole'i tuletiste arvutamise ahelmeetodit võrrandi (1-19) kujul.



Joonis 1-10. Järjestikskeem

$$\frac{\partial y(t)}{\partial x_1(t-\tau)} = \frac{\partial y(t)}{\partial Q(t)} \frac{\partial Q(t)}{\partial J(t-1)} \frac{\partial J(t-1)}{\partial x_1(t-1)} = \overline{x_3(t)} \overline{Q(t-1)} x_2(t-1) = 1 \quad (1-19)$$

Võrrandi vasakut poolt koostama hakates polnud veel teada testjada pikkus, seetõttu sai ajaline nihe muutujate  $y$  ja  $x_1$  vahel fikseeritud esialgu veel määramatu väärtusega sümbolmuutujaga  $\tau$ . Liitfunktsiooni osatuletiste jada valmimisel selgub ka selle muutuja lõplik väärtus ehk otsitav ajaline nihe  $\tau = 1$ .

Võrrandi vasakul pool olevast tingimusest  $Q(t-1) = 0$  tuleneb uus tingimus  $K(t-2) = 1$ . Asendades nüüd selle tingimuse võrrandi (1-19) vasakusse poolde, saame lõpliku võrrandi

$$x_3(t)K(t-2)x_2(t-1) = 1 \quad (1-20)$$

millest tuleneb tabelis 1-1 esitatud 3-taktiline testjada.

Tabel 1-1

3-taktiline testjada skeemile joonisel 1-10

$t$	$x_1$	$x_2$	$x_3$	$Q'$	$K$	$y$
1					1	
2	$D$	1		0		
3			0	$D$		$D$

Tühjad ruudud tabelis vastavad määramata väärtustele. *Sümboliga D* tähistame rikke tõttu potentsiaalselt muutuvat väärtust. Apostroof muutuja  $Q$  juures tähendab muutuja väärtust eelmises taktis.

### 1.1.7. Boole'i diferentsiaalid

Testeksperimentide läbiviimist ja rikete lokaliseerimist (diagnoosimist) saab modelleerida *Boole'i diferentsiaalide* abil.

Klassikalisest matemaatilisest analüüsist teame, et funktsiooni  $y = F(x)$  *diferentsiaaliks*

$$dy = \frac{\partial y}{\partial x} dx \quad (1-21)$$

nimetatakse selle funktsiooni juurdekasvuga ekvivalentset lõpmatult vähenevat suurust, mis on võrdeline argumenti juurdekasvuga, kusjuures funktsioonil on diferentsiaal olemas siis ja ainult siis, kui funktsioon on diferentseeruv. Boole'i funktsioonide puhul, kus argumentide arv on suur, kasutame taas *osadiferentsiaali* mõistet

$$d_x y = \frac{\partial y}{\partial x} dx$$

Boole'i diferentsiaali  $dx$  all mõistame muutuja  $x$  väärtuse muutumist mingi rikke tõttu  $dx = 1$ , kui  $x$  väärtus muutub (esineb viga), ja  $dx = 0$ , kui  $x$  väärtus ei muutu (viga ei olnud). Boole'i funktsiooni osadiferentsiaal  $d_x y$  väljendab testeksperimenti tulemust  $d_x y = 1$ , kui  $y$  väärtus muutub  $x$  väärtuse muutudes, ja  $d_x y = 0$ , kui ei muutu. Tingimuseks, et niisugune muutus toimub, ongi eelpool käsitletud osatuletise võrdumine 1-ga  $\partial y/\partial x=1$ .

Boole'i funktsiooni *täisdiferentsiaal* avaldub valemina

$$\begin{aligned} dy &= F(X) \oplus F(X \oplus dX) = \\ &= F(x_1, \dots, x_i, \dots, x_n) \oplus F(x_1 \oplus dx_1, \dots, x_i \oplus dx_i, \dots, x_n \oplus dx_n) \end{aligned} \quad (1-22)$$

kus

$$dX = (dx_1, dx_2, \dots, dx_n)$$

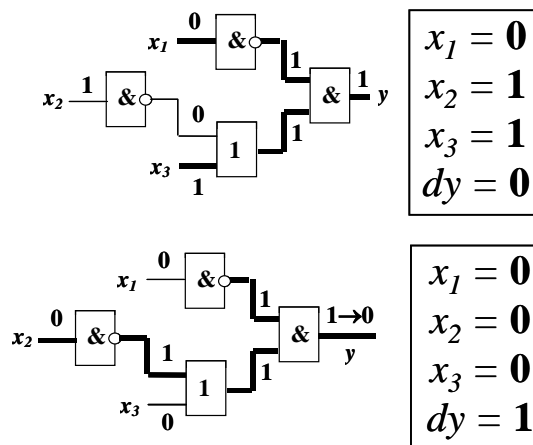
Võrreldes valemite (1-22) valemiga (1-11), on kerge näha, et Boole'i täisdiferentsiaal kujutab endast kõigi Boole'i vektortuletiste hulga parameetrilist kuju. Täpsemalt väljendub nimetatud omadus järgmises täisdiferentsiaali kujus

$$dy = \frac{\partial y}{\partial x_1} dx_1 \overline{dx_2} \dots \overline{dx_n} \vee \dots \vee \frac{\partial y}{\partial x_n} \overline{dx_1} \overline{dx_2} \dots \overline{dx_{n-1}} dx_n \vee \dots \vee \frac{\partial y}{\partial(x_1 x_2)} dx_1 dx_2 \overline{dx_3} \dots \overline{dx_n} \vee \dots \vee \frac{\partial y}{\partial(x_1 x_2 \dots x_n)} dx_1 dx_2 \dots dx_n \quad (1-23)$$

või näiteks kahe muutuja funktsiooni  $y = F(x_1, x_2)$  puhul

$$dy = \frac{\partial y}{\partial x_1} dx_1 \overline{dx_2} \vee \frac{\partial y}{\partial x_2} \overline{dx_1} dx_2 \vee \frac{\partial y}{\partial(x_1 x_2)} dx_1 dx_2 \quad (1-24)$$

Põhiline Boole'i täisdiferentsiaali roll digitaalskeemide diagnostilises analüüsis seisneb võimaluses dešifreerida *diagnostikaeksperimentide* tulemusi, teiste sõnadega määrata vigaste väljundsignaalide põhjusi.



Joonis 1-11. Kaks diagnostikaeksperimenti

Näide 1-8

Olgu antud kombinatsioonskeem funktsiooniga

$$y = \bar{x}_1(\bar{x}_2 \vee x_3) \quad (1-25)$$

ja selle täisdiferentsiaaliga

$$dy = y \oplus (\bar{x}_1 \oplus dx_1)((\bar{x}_2 \oplus dx_2) \vee (x_3 \oplus dx_3)) \quad (1-26)$$

Viime läbi kaks diagnostikaeksperimenti (joonis 1-11) testvektoritega  $T_1 = 011$  ja  $T_2 = 000$ . Oletame, et esimesel juhul test läheb läbi, s.t väljundi väärtus on  $y = 1$ , mis vastab oodatavale tulemusele. Eksperimenti tulemuseks saame seega  $dy = 0$ . Oletame, et teisel juhul test ei lähe läbi, s.t väljundis avastatakse oodatava signaali 1 asemel 0. *Diagnostikaeksperimenti tulemus* vastab nüüd diferentsiaalile  $dy = 1$ .

Asetame vektoritega  $T_1$  ja  $T_2$  määratud sisendmuutujate väärtused ning testide tulemused  $dy$  avaldisse (1-26) ning saame kahe testeksperimentiga määratud võrrandisüsteemi

$$\begin{cases} dy = 1 \oplus \bar{dx}_1(dx_2 \vee \bar{dx}_3) = 0 \\ dy = 1 \oplus \bar{dx}_1(\bar{dx}_2 \vee dx_3) = 1 \end{cases}$$

Lihtsustades avaldise ja teisendades ka esimese avaldise tõeseks, saame

$$\begin{cases} \bar{dx}_1^0(dx_2^1 \vee \bar{dx}_3^1) = 1 \\ dx_1^0 \vee dx_2^0 \bar{dx}_3^0 = 1 \end{cases} \quad (1-27)$$

Et diferentsiaale muutujate erinevate väärtuste tõttu üheselt mõista, viime diferentsiaalide tähistusse sisse ülaindeksid, mis märgivad muutujate õigeid väärtusi. Näiteks  $dx_2^1 = 1$  tähendab, et muutuja  $x_2$  väärtus peaks olema 1, aga rikke tõttu on tegelikult 0.

Süsteemi (1-27) esimesest võrrandist tuleneb üheselt mõistetav tulemus

$$\bar{\partial x}_1^0 = 1$$

mis tähendab, et sisendis  $x_1$  rike  $x_1 \equiv 1$  kindlasti puudub. Rohkem ei tulene üheselt mõistetavaid fakte enam kummastki süsteemi (1-27) võrrandist.



Küll aga tuleb rohkesti informatsiooni, kui kaks võrrandit loogiliselt omavahel korrutada, mis võtab kokku mõlemast eksperimentidest saadava informatsiooni

$$\overline{dx_1}^0(dx_2^1 dx_2^0 \overline{dx_3}^0 \vee dx_2^0 \overline{dx_3}^0 \overline{dx_3}^1) = 1 \quad (1-28)$$

Kasutades lihtsustust

$$dx_k^0 dx_k^1 = 0 \quad (1-29)$$

mis tähendab, et ühes ja samas sisendis (või ühel ja samal juhtmel)  $x_k$  ei saa olla korraga mõlemat tüüpi rikked  $x_k \equiv 1$  ja  $x_k \equiv 0$ , saame võrrandist (1-28) järgmise tulemuse:

$$\overline{dx_1}^0 dx_2^0 \overline{dx_3}^0 \overline{dx_3}^1 = 1 \quad (1-30)$$

Tõene lause (1-30) ongi *diagnos* ehk kahe testeksperimenti tulemuse lahtimõtestamine. Lause (1-30) väidab, et

- sisendis  $x_1$  puudub rike  $x_1 \equiv 1$ ,
- sisendis  $x_2$  on rike  $x_2 \equiv 1$ ,
- sisend  $x_3$  on korras (mõlemat tüüpi rikked puuduvad).

#### *Boole'i täisdiferentsiaali omadused*

Olulisteks *diferentsiaalioperaatori d* omadusteks on

- Boole'i võrrandite diferentseerimise võimalus ja
- Boole'i täisdiferentsiaali avaldise *invariantsus*.

Olgu antud Boole'i võrrand

$$F_1(X) = F_2(X)$$

Siis kehtib ka järgmine võrrand

$$dF_1 = dF_2$$

Niisugust tehet nimetatakse Boole'i võrrandi täisdiferentseerimiseks.

Olgu antud liitfunktsioon

$$F_k(X, F_i(X))$$

Siis vastavalt Boole'i täisdiferentsiaali definitsioonile (1-22) saame

$$dF_k(X, F_i(X)) = F_k(X, F_i(X)) \oplus F_k(X \oplus dX, F_i(X \oplus dX))$$

Aga kuna

$$F_i(X \oplus dX) = F_i(X) \oplus dF_i$$

siis

$$dF_k = F_k(X, F_i(X)) \oplus F_k(X \oplus dX, F_i \oplus dF_i) \quad (1-31)$$

Avaldisest (1-31) järgneb, et üldine formaalne Boole'i täisdiferentsiaali avaldis ei sõltu sellest, kas muutujad on sõltumatud või ei ole sõltumatud. Teiste sõnadega, Boole'i täisdiferentsiaali avaldis on invariantne oma muutujate suhtes.

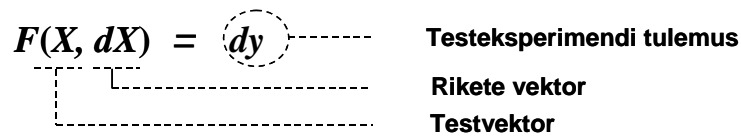
Sellel Boole'i täisdiferentsiaali invariantse omadusel põhineb võimalus arvutada diferentsiaale ka liitfunktsioonidele. See aga omakorda laseb keerukaid Boole'i avaldise tükeldada lihtsamateks osadeks ja sel moel lihtsustada diferentseerimist.

### 1.1.8. Digitaalskeemide diagnostika üldvõrand

Boole'i funktsioonide täisdiferentsiaali võrand kätkeb võimalust lahendada kõiki digitaalskeemide diagnostikaga seotud ülesandeid. Kaheks põhiülesandeks loetakse

- testide genereerimist (kui *pöördülesannet*) ja
- rikete diagnoosi (kui *otseülesannet*).

Kõigepealt kujutab täisdiferentsiaal endast testeksperimenti mudelit, kus on ühendatud korruga signaalide mõjutamine (stimuleerimine) testvektoritega, rikete modelleerimine ja testi tulemuste arvesse võtmine (joonis 1-12).



**Joonis 1-12.** Diagnostika üldvõrrand

Diagnostika üldvõrrandina esitatavast *diagnostika eksperimentide mudelist* tulenevad otseselt kahe diagnostika põhiülesande formuleeringud, mis seisnevad kahte tüüpi võrrandi lahendamise leidmises (joonis 1-13)

*Otseülesanne*

**Testide genereerimine:**  $dX, dy = 1$  – on antud  
 $X$  – tuleb leida

*Pöördülesanne*

**Rikete diagnoos:**  $X, dy$  – on antud  
 $dX$  – tuleb leida

**Joonis 1-13.** Diagnostika põhiülesannete tulenemine üldvõrrandist

*Rikete simuleerimist* võib vaadelda kui rikete diagnoosi ülesande erijuhtu, kus imiteeritakse tahtlikult eksperimendi viga  $dy = 1$  ja viiakse sel põhjal läbi diagnoos: leitakse, millised vektoriga  $dX$  määratud rikked avastab antud test  $X$ .

*Näide 1-9*

Vaatleme veel kord funktsiooni

$$y = \bar{x}_1(\bar{x}_2 \vee x_3)$$

täisdiferentsiaali

$$dy = y \oplus (\bar{x}_1 \oplus dx_1)((\bar{x}_2 \oplus dx_2) \vee (x_3 \oplus dx_3)) \quad (1-32)$$

Testide genereerimisel antakse ette mingi rikete kombinatsioon, näiteks  $dX = 011$ , mis tähendab, et soovitakse genereerida testi kordsele

rikkele sisendites  $x_2$  ja  $x_3$ . Rikke avastamine tähendab ühtaegu veel ka tingimust  $dy = 1$ .

Võrrand (1-32) võtab nüüd kuju

$$y \oplus \overline{x_1}(x_2 \vee \overline{x_3}) = 1 \quad (1-33)$$

Võrrandi (1-33) lahendiks on test, mis avastab etteantud rikete kombinatsiooni. Kerge on näha, et sama ülesande sõnastuse oleksime saanud ka kasutades vektorosatuleti (vt p 1.1.3).

Rikete diagnostika ülesande lahendamisel antakse ette mingi testvektor, näiteks  $X = 011$ , ja testi tulemus, näiteks  $dy = 1$ . Asetades nimetatud väärtused võrrandisse (1-32), saame uue lihtsustatud võrrandi

$$y \oplus \overline{dx_1}(dx_2 \vee \overline{dx_3}) = 1. \quad (1-34)$$

Rikete diagnoos leitakse võrrandi (1-34) lahendist.

Praktikas nii otse- kui ka pöördülesandeid enamasti lihtsustatakse. Testide genereerimisel antakse ette vaid üksik konkreetne rike  $dx_k = 1$  ja kõik ülejäänud rikete avaldumiste võimalused välistatakse (kõigi ülejäänud muutujate  $dx_i \in dX$  jaoks võetakse  $dx_i = 0$ ). Sellisel juhul, nagu selgelt ilmneb valemist (1-23), redutseerub täisdiferentsiaali üldavaldis Boole'i osatuletiseks.

## 1.2. Otsustusdiagrammid

Boole'i diferentsiaalparaadi puuduseks on, et teda saab kasutada üksnes digitaalsüsteemide abstraktse esituse loogikatasandil. Palju laiemaid võimalusi annab digitaalsüsteemide graafiline esitus *otsustusdiagrammide* (OD) kujul, kus kõik operatsioonid taanduvad graafi (otsustusdiagrammi *topoloogilisele analüüsile*). Otsustusdiagrammide erijuhuks on *binaarsed otsustusdiagrammid* (BOD).

Binaarsete otsustusdiagrammide töötlemiseks diagnostika seisukohalt on levinud kaks kontseptsiooni: otsustusdiagrammide manipuleerimine ehk loogikaoperatsioonide läbiviimine ja graafidel läbiviidav topoloogiline analüüs. Esimesel juhul ei välju me loogika algebra raamidest ja graafidega opereerimise meetodite üldistamine digitaalsüsteemide kõrgematele

tasanditele pole võimalik. Teisel juhul, kus graafidega töö aluseks on topoloogiline analüüs, on meetodite üldistamine loogikatasandilt kõrgematele tasanditele suhteliselt lihtne.

Vaatleme otsustusdiagrammide analüüsi topoloogilist kontseptsiooni, et graafide abil mõista digitaalsüsteemide diagnostika ühtsust nii süsteemide madalatel kui ka kõrgematel esitustasanditel.

### 1.2.1. Binaarsete otsustusdiagrammide mõiste

1959. aastal võttis binaarsed otsustusdiagrammid esmakordselt kasutusele C.Y.Lee [Lee 59] nime all *binaarsed otsustusprogrammid*. 1976 kasutati Tallinna Tehnikaülikoolis [Uba 76] esmakordselt BOD mudelit digitaalskeemide diagnostikas, tookord *alternatiivsete graafide* nime all. Tegelikult veel varemgi, kuna juba 1974 kaitses oma diplomitöö BOD kasutamisest digitaalskeemide simuleerimise kiirendamiseks TTÜ-s Virve Vaher käesoleva raamatu autori juhendamisel. Paar aastat hiljem võeti samad graafid kasutusele ka USAs *binaarsete otsustusdiagrammide* nime all [Ake 78]. Täna on BOD-teooria laialt kasutatav ja kiiresti arenev [Bry 86, Min 96, Sas 96, Dre 98].

Boole'i funktsiooni

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

esitav BOD on *orienteeritud tsükliteta graaf*

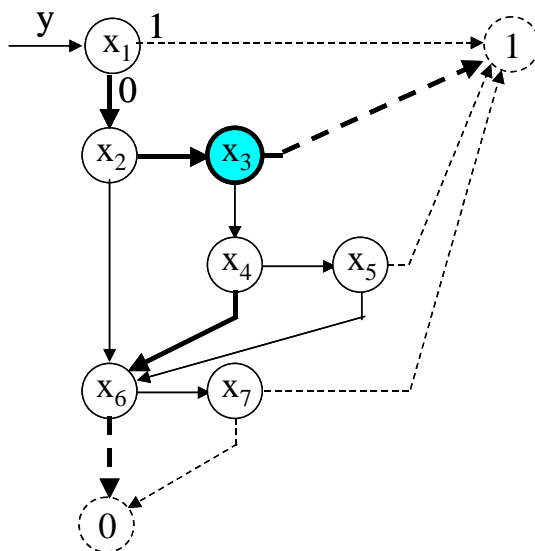
$$G_y = (M, X, \Gamma)$$

kus  $M$  on *graafi tippude hulk*,  $X = \{x_1, x_2, \dots, x_n\}$  funktsiooni argumentide hulk,  $x(m) \in X$  on muutuja, mis on vastavusse seatud tipuga  $m \in M$ , ja  $\Gamma(M, X)$  on funktsiooniga  $y = F(X)$  üheselt määratud relatsioon, kus  $\Gamma(m, x(m))$  määrab tipu  $m$  järglase vastavalt *tipumuutuja*  $x(m)$  väärtusele. Vektoriga  $X^* \in \{0, 1\}^n$  on graafis määratud liikumine tipust tippu. 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 *terminaltippu*  $m^{T,e}$ ,  $e \in \{0, 1\}$ , mida 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 terminaltippu  $m^{T,e}$ , kehtib  $y = F(X^*) = e$ .



**Joonis 1-14.** Binaarne otsustusdiagramm

*Näide 1-10*

Joonisel 1-14 on esitatud BOD, mis vastab Boole'i funktsioonile

$$y = x_1 \vee x_2 (x_3 \vee x_4 x_5) \vee x_6 x_7$$

Edaspidi esitame graafe selliselt, et 1-kaared lähevad paremale ja 0-kaared alla. Sellisel juhul ei ole vaja kaartele panna tähistusi 0 ja 1. Olgu antud vektor  $X = 0110101$ . Liikudes selle vektori järgi, läbime tippe  $x_1, x_2, x_3, 1$ . Terminaltippu 1 jõudmine tähendab, et Boole'i funktsiooni väärtus antud vektori korral on 1.

Binaarseid otsustusdiagramme võib interpreteerida järgmistel viisidel:

- BOD on *binaarne programm* Boole'i funktsioonide arvutamiseks, kus tippudele vastavad *if-then-else* käsud,
- BOD on *andmestruktuur*, mida kasutades võib töötada mõni teine programm (näiteks BOD-le vastava funktsiooni simuleerimiseks ehk väärtuse arvutamiseks);
- BOD on *viis esitada teavet* mingist digitaalskeemist (selle interpretatsiooni juurde tuleme edaspidi vt 1.2.4).

Teiselt poolt ei ole binaarne otsustusdiagramm muud kui võimalus esitada kombinatsioonskeemide funktsioone, käsitledes esitatavat skeemi *musta kastina*. Skeemide diagnostiliseks modelleerimiseks oleks aga vaja käsitleda ka skeemi sisemist struktuuri, s.t simuleerida rikkeid skeemi sisepunktides. Kui modelleerida BOD abil skeeme ventiilide tasandil diagnostika eesmärgil, on kaks võimalust:

- kujutada skeemi ventiilide võrguna ja esitada igat ventiili BOD kujul või
- konstrueerida igale skeemi sisepunktile selle funktsiooni esitamiseks eraldi BOD.

Viimasel juhul on BOD-i puuduseks tema keerukuse eksponentsiaalne kasv kombinatsioonskeemide keerukuse kasvades.

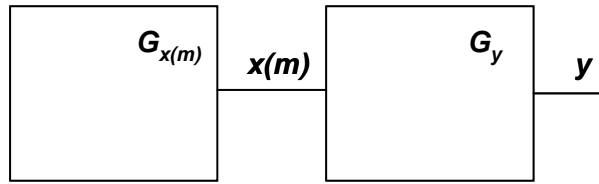
Järgnevalt vaatleme ühte BOD-i eriklassi, nn *struktuurselt sünteesitud BOD*-sid (SSBOD), kus nimetatud puuduste mõju väheneb.

## 1.2.2. Struktuurselt sünteesitud binaarsed otsustusdiagrammid

Struktuurselt sünteesitud BOD-i mudel võeti kasutusele *ventiiltasandil* esitatud kombinatsioonskeemide diagnostiliseks modelleerimiseks signaaliteede terminites [Uba 76, Pla 80, Uba 96].

SSBOD sünteesi aluseks on binaarsete otsustusdiagrammide *superpositsiooni meetod* [Uba 76]. Olgu antud kaks kombinatsioonskeemi (joonis 1-15), millest üks on esitatud BOD-ga  $G_y = (M_y, X_y, \Gamma)$ , ja ühele tipule  $m \in M_y$  vastav muutuja  $x(m) \in X_y$  on teise kombinatsioonskeemi väljundiks, mis omakorda on esitatud ühe teise BOD-ga  $G_{x(m)} = (M_{x(m)}, X_{x(m)}, \Gamma)$ . Otsustusdiagrammide superpositsioon seisneb antud juhul

graafi  $G_y$  tipp  $m \in M_y$  asendamises graafiga  $G_{x(m)}$ . Operatsioon seisneb järgmiste sammude läbiviimises.



**Joonis 1-15.** Kaks omavahel ühendatud skeemi

*Algoritm 1-1. BOD superpositsioon*

1. Tipp  $m$  koos kahe väljuva kaarega eemaldatakse graafist  $G_y$ .
2. Terminaltipud  $m^{T,e}$ ,  $e \in \{0,1\}$  eemaldatakse graafist  $G_{x(m)}$ .
3. Kõik terminaltippudega  $m^{T,e}$  seotud olnud kaared graafis  $G_{x(m)}$  ühendatakse tippu  $m \in M_y$  vastavate järglastega  $m^e$  graafis  $G_y$ .
4. Kõik tippu  $m$  sisenenud kaared graafis  $G_y$  ühendatakse graafi  $G_{x(m)}$  algtipuga ehk juurega  $m_0$ .

Algoritmi 1-1 teel saadud graafe nimetamegi *struktuurselt sünteesitud binaarseteks otsustusdiagrammideks* (SSBOD).

*Näide 1-11*

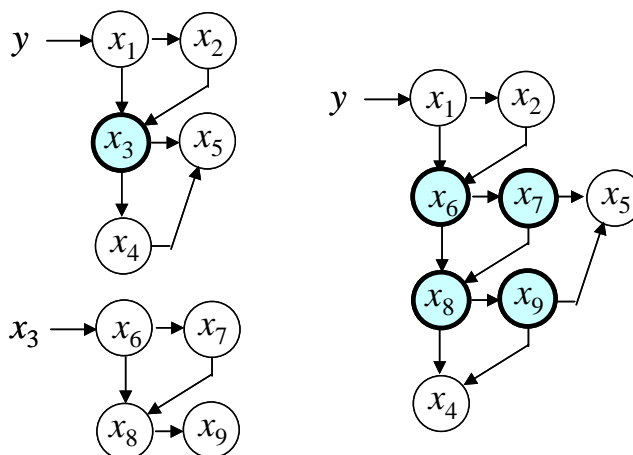
Kahe BOD-i superpositsiooni operatsioon on illustreeritud joonisel 1-16. Funktsiooni  $y = x_1x_2 \vee (x_3 \vee x_4)x_5$  esitava graafi  $G_y$  tipp  $m$  asendatakse teise graafiga  $G_{x,3}$ , mis esitab funktsiooni  $x_3 = x_6x_7 \vee x_8x_9$ . Uus graaf  $G_y$  esitab funktsiooni

$$y = x_1x_2 \vee (x_6x_7 \vee x_8x_9 \vee x_4) x_5$$

Graafika lihtsustamiseks on joonisel 1-16 (ja edaspidi) terminaltipud koos sinna suubuvate kaartega ära jäetud. Kuna kokkuleppeliselt kõik 1-kaared suunduvad alati paremale ja kõik 0-kaared alla, siis 1-kaarte puudumist tuleb interpreteerida, et graafist väljumine paremale tähendab



sisenemist 1-terminali ja 0-kaarte puudumist tuleb interpreteerida, et graafist väljumine alla tähendab sisenemist 0-terminali.

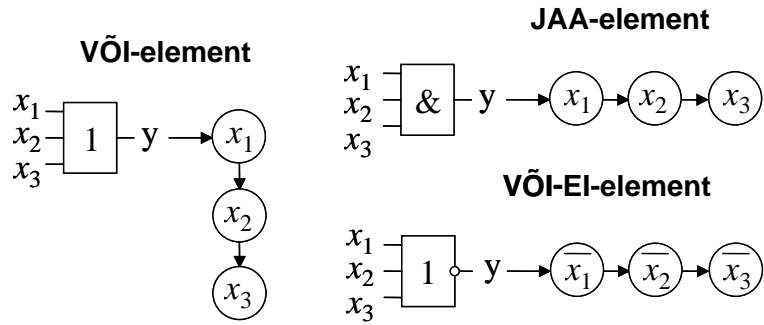


*Joonis 1-16. Binaarsete otsustusdiagrammide superpositsioon*

Kasutades BOD superpositsiooni võib suvalise ventiiltasemel esitatud kombinatsiooniskeemile vastavusse seada struktuurselt sünteesitud BOD.

Olgu antud ventiiltasemel kombinatsiooniskeem, kus iga ventiil (loogikaelement) on esitatud binaarse otsustusdiagrammiga. Niisuguste elementaardiagrammide näited on toodud joonisel 1-17.

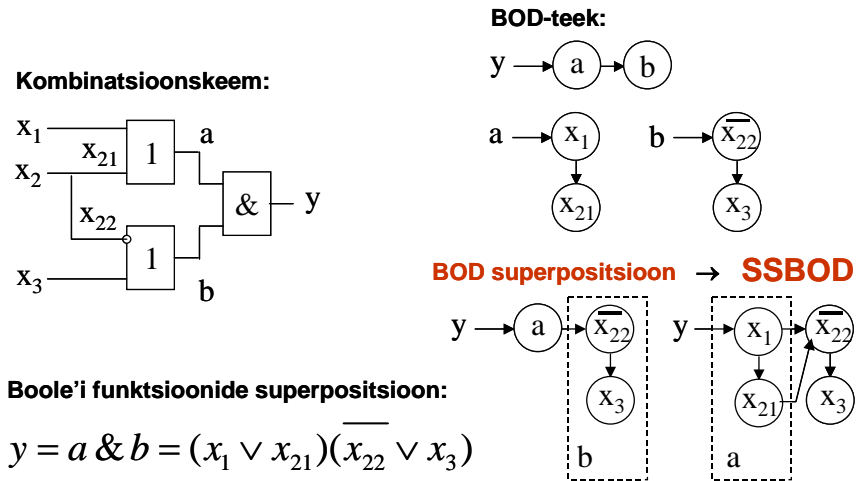
Alustades skeemi väljundelemendile vastavast graafist ja kasutades *iteratiivselt* superpositsiooni protseduuri, saaksime graafide võrgu asemel üheainsa graafi (iga iteratsiooniga redutseerime mudelit ühe tippu ja ühe graafi võrra). Selleks et redutseerida mudelit (tippude koguarvu) maksimaalselt, tuleks superponeerida graafe üksnes puukujuliste alamskeemide ulatustes. Sellisel juhul saaksime ventiilide võrgu asemel *puukujuliste alamskeemide* ehk *makrode* võrgu, kus igale makrole vastab üks SSBOD.



Joonis 1-17. Elementaasete binaarsete otsustusdiagrammide näited

Näide 1-12

Joonisel 1-18 on esitatud näide SSBOD genereerimisest puukujulisele kombinatsiooniskeemile. Võrdluseks on toodud skeemile vastava Boole'i funktsiooni superpositsioon.



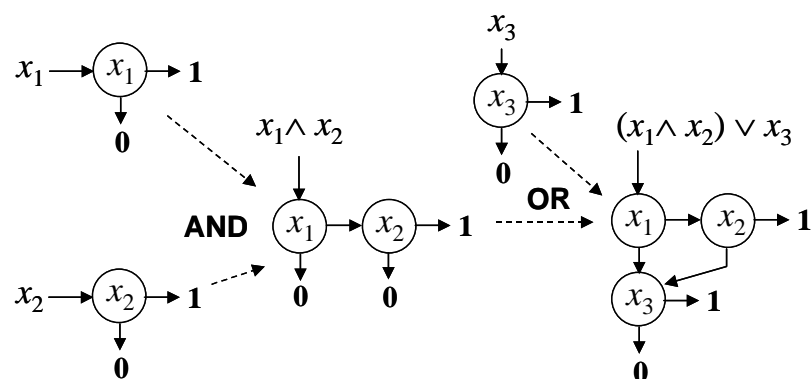
Joonis 1-18. SSBOD-mudeli süntees kombinatsiooniskeemile

SSBOD mudeli genereerimist antud kombinatsioonskeemile alustame joonisel 1-18 väljundelemendist JAA ja sellele vastavast graafist  $G_y$ , mis koosneb kahest tipust  $a$  ja  $b$ . JAA elemendi sisend  $a$  on samal ajal graafiga  $G_a$  esitatud VÕI elemendi väljundiks. Graaf  $G_a$  koosneb kahest tipust  $x_1$  ja  $x_{21}$ . Kõigepealt asendame tipu  $a$  graafis  $G_y$  graafiga  $G_a$ , seejärel tipu  $b$  graafis  $G_y$  graafiga  $G_b$ , mis koosneb tippudest  $x_{22}$  ja  $x_3$ . Lõplik graaf, mis esindab antud puukujulist kombinatsioonskeemi ehk makrot, kust on välja jäetud hargnev sisend  $x_2$ , koosneb tippudest  $x_1$ ,  $x_{21}$ ,  $x_{22}$ , ja  $x_3$ .

Olgu märgitud, et samale ventiilskeemile saaks sünteesida SSBOD-mudelit kasutades ka BOD *sümboolset manipuleerimist* [Bry 86]. Näide elementaarsete loogikaoperatsioonide kasutamisest BOD sünteesiks Boole'i funktsioonile  $y = x_1x_2 \vee x_3$  või sellele vastavale kahest elemendist koosnevale ventiilskeemile alates sisenditest kuni väljundini [Sas 98] (vastupidi superpositsiooni protseduurile, mis toimub väljundist sisendite suunas) on esitatud joonisel 1-19.

Sümboolse manipuleerimise meetod graafidega loogikaoperatsioone sooritades on läbi viidav vaid graafi ventiilide kaupa ehitades. Superpositsioonimeetod on üldisem, kuna võimaldab suvalisi graafe (funktsioone) kokku ühendada.

SSBOD mudelit võib kasutada otseselt ventiiltasandil esitatud kombinatsioonskeemidele testide genereerimiseks.

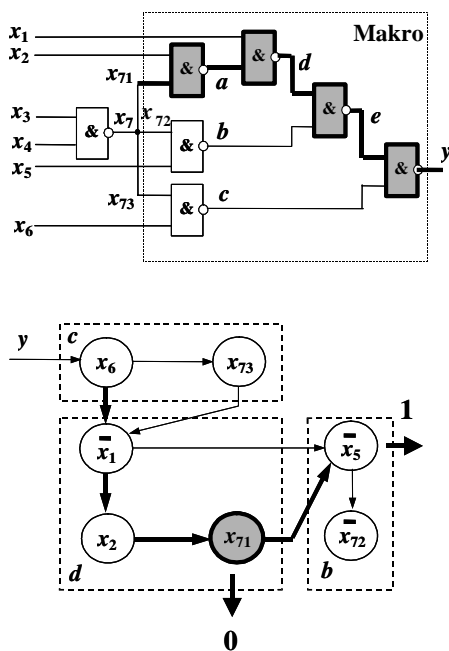


Joonis 1-19. BOD mudeli süntees loogikaoperatsioone kasutades

Erinevalt tavalistest binaarsetest otustusdiagrammidest kehtib SSBOD-mudeli jaoks järgmine omadus: igale tipule  $m$  graafis  $G_y$ , mis kirjeldab mingis ventiiltasandil esitatud kombinatsiooniskeemis  $N$  puukujulist alamskeemi  $N_y$ , vastab üheselt mingi signaalitee  $l(m)$  skeemis  $N_y$ . See üksühene vastavus SSBOD tippude ja kombinatsiooniskeemi signaaliteede vahel tuleneb otseselt SSBOD sünteesist superpositsiooni meetodil.

Näide 1-13

Joonisel 1-20 on esitatud puukujuline makro ja sellele vastav struktuurselt sünteesitud otsustusdiagramm.



Joonis 1-20. Kombinatsiooniskeem ja selle BOD-mudel

Igale tipule  $m$  tipumuutujaga  $x(m)$  selles graafis joonisel 1-20 vastab signaalitee kombinatsiooniskeemis, mis algab muutujaga  $x(m)$  tähistatud ühendusest ja levib väljundini  $y$ . Tipumuutuja on invertteeritud, 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 1-20). Tipumuutuja  $x_1$  on aga *inverteeritud*, kuna talle vastav signaalitee  $x_1, d, e, y$  sisaldab paaritud arvu invertoreid. Skeemipunktis  $x_7$  on kolm hargnemist, kus iga haruga  $x_{7,k}$  ( $k = 1, 2, 3$ ) algavale ja väljundis  $y$  lõppevale signaaliteele skeemis vastab konkreetne sama muutujaga  $x_{7,k}$  tähistatud tipp graafis.

Kirjeldatud SSBOD-mudelil on rida eeliseid üldisema pelgalt Boole'i funktsioonide esitamiseks mõeldud BOD-mudeliga võrreldes.

Esiteks, SSBOD modelleerib lisaks skeemi funktsioonile ka skeemi struktuuri – igale graafitipule vastab üks ja ainus signaalitee skeemis. Sellest tulenevalt on skeemi *struktuursete rikete* esitamine mudelis vahetu – skeemi rikkeid võib vaadelda graafitippude atribuutidena.

Teiseks, kombinatsiooniskeeme võib diagnostika eesmärgil modelleerida SSBOD abil nii ventiilide kui ka kõrgemal makrode tasandil, kusjuures makroks võib olla suvaline osa skeemist (alamskeem). Erinevalt tavapärasest modelleerimisest ventiilide tasandil, kus iga ventiilitüübi esitamiseks on vajalik tema konkreetse mudeli hoidmine andmeteegis, ei ole SSBOD puhul selliste mudeltekide olemasolu vajalik. Kolmandaks, diagnostilise analüüsi meetodeid, mis on välja töötatud SSBOD-mudeli alusel loogikatasandil silmas pidades, on võimalik vahetult üldistada kõrgema tasandi otsustusdiagrammide mudeli jaoks digitaalsüsteemide kõrgemaid esitustasandeid arvestades [Uba 81, Uba 83, Uba 96], vt. 1.2.4. – 1.2.7.

### 1.2.3. Diagnostikaoperatsioonid SSBOD mudelil

Vaatleme kahe põhilise diagnostikaülesande lahendamist struktuurselt sünteesitud otsustusdiagrammidel: pöördülesannet ehk testide genereerimist ja otseülesannet ehk rikete diagnoosi.

#### *Testide genereerimine*

Olgu antud  $n$  sisendiga puukujuline skeem  $N_y$ , millele vastab mingi Boole'i funktsioon  $y = F(X) = F(x_1, x_2, \dots, x_n)$  ja SSBOD  $G_y$ . Genereerides testi tipule  $m$  graafis  $G_y$ , leitakse sisendsignaali vektor, mis on testiks sisendile  $x(m)$  skeemis  $N_y$  ja tervele reale riketele signaaliteel  $l(m)$ . Sama testi leidsime, lahendades Boole'i diferentsiaalvõrrandi (1-3).

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

*Algoritm 1-2. Testi genereerimine SSBOD tipule*

1. Aktiveerida tee  $l_m$  SSBOD algtipust tipuni  $m$ .
2. Aktiveeritakse 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.

*Näide 1-14*

Vaatleme testi genereerimist graafi  $G_y$  tipule muutujaga  $x_{71}$  joonisel 1-20, millele vastab signaalitee skeemis, alates punktist  $x_{71}$  läbi ventiilide  $a,d,e,y$ .

Aktiveerime *algoritmi 1-2* kohaselt tee  $l_m$  läbi tippude<sup>2</sup>  $x_6, \neg x_1, x_2$ , 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  $x_1, x_2, x_5, x_6, x_7 = 1100-$ . Kuna  $x_7$  on skeemi sisepunkt, siis tuleb antud testi vaadelda kui makrokomponendi *lokaalset* sisendvektorit. Kuidas lõplikku testi saada antud haru testimiseks, käsitleme edaspidi. Leitud testvektor aktiveerib skeemis rasvaselt näidatud signaalitee. Ka testi leidmiseks vajalikud aktiveeritud teed graafis on näidatud rasvaselt.

Kui kasutaksime antud ülesande lahendamiseks Boole'i diferentsiaalparaati, tuleks lahendada võrrand:

$$\begin{aligned} \frac{\partial y}{\partial x_{7,1}} &= \frac{\partial (x_6 x_{7,3} \vee (\overline{x_1} \vee x_2 x_{7,1}) (\overline{x_5} \vee \overline{x_{7,2}}))}{\partial x_{7,1}} = \\ &= \overline{x_6} \overline{x_{7,3}} (\overline{x_5} \vee \overline{x_{7,2}}) x_1 x_2 = x_1 x_2 \overline{x_5} \overline{x_6} \vee \dots = 1 \end{aligned}$$

---

<sup>2</sup> Lihtsuse mõttes nimetame tippe  $m$  mõnikord ka neile vastavate tipumuutujate  $x(m)$  järgi.

Saaksime sama lahendi. Võrrandist nähtub, et lahendeid on veelgi rohkem, kuid kõik need sõltuvad muutuja  $x_7$  väärtusest. Näiteks üks lahenditest  $x_1, x_2, x_5, x_6, x_7 = 11-00$  tähendab seda, et saaksime testida sisendis  $x_7$  vaid signaalimuutusi  $0 \rightarrow 1$ , aga mitte muutusi  $1 \rightarrow 0$ . Selles mõttes on esimene lahend üldisem, kuna muutuja  $x_7$  väärtusest mittesõltumine võimaldab testida sisendis  $x_7$  mõlemasuunalisi signaalimuutusi.

#### *Rikete diagnostika*

Olgu antud testvektor  $T = (X^*, e)$ , kus  $e \in \{0, 1\}$  on funktsiooni muutuja  $y$  väärtus sisendvektori  $X^*$  korral. Vektor  $X^*$  aktiveerib mingi tee  $l(T)$  graafis terminaltipuni  $m^{T,e}$ . Tipp  $m \in l(T)$  sellel teel on siis ja ainult siis testitud, kui lisaks teele  $l(T)$  on aktiveeritud veel mingi teine tee  $l_{m,-e}$  tipust  $m^{-e}$  terminalini  $m^{T,e}$ .

Nimetatud tingimusest tuleneb järgmine *testvektorite diagnostilise analüüsi* algoritm.

#### *Algoritm 1-3. Testvektori T diagnostiline analüüs SSBOD-mudelil*

1. Simuleerida läbi testiga aktiveeritud tee  $l(T)$  potentsiaalselt testitavate tippude leidmiseks. Olgu selle tee terminaltipuks  $m^{T,e}$ .
2. Iga tipu  $m \in l(T)$  jaoks jätkata simuleerimist alates selle tipu järglasest  $m^{-e} \notin l(T)$ , et kontrollida, kas teine tee  $l_{m,-e}$  terminaltipu  $m^{T,-e}$  on samuti aktiveeritud; kui on, siis antud vektor  $T$  testib tipu  $m$ .

#### *Näide 1-15*

Vaatleme veel kord joonist 1-20. Lokaalne testvektor joonisel esitatud makro jaoks  $T = (11001, 1)$  aktiveerib graafis tee  $l(T) = (x_6, -x_1, x_2, x_{7,1}, -x_5, m^{T,1})$ . Järgnevalt kontrollime aktiveeritud tee  $l_{m,0}$  olemasolu tippudele sellel teel  $l(T)$ . Tippudel  $x_6$  ja  $-x_1$  see puudub, mõlema tipu puhul tipumuutuja väärtuse muutumine ei muuda funktsiooni  $y$  väärtust. Tippude jaoks  $x_2$  ja  $x_{7,1}$  ei olegi vaja seda kontrolli läbi teha – ühendus 0-terminaliga on vahetu. Ka tipu  $-x_5$  jaoks tee  $l_{m,0}$  puudub. Seega testib vektor  $T$  antud graafis üksnes tippe  $x_2$  ja  $x_{7,1}$  ehk vastavalt mõlemat signaaliteed skeemis, alates sisenditest  $x_2$  ja  $x_{7,1}$  läbi ventiilide  $a, d, e$  ja  $y$ .

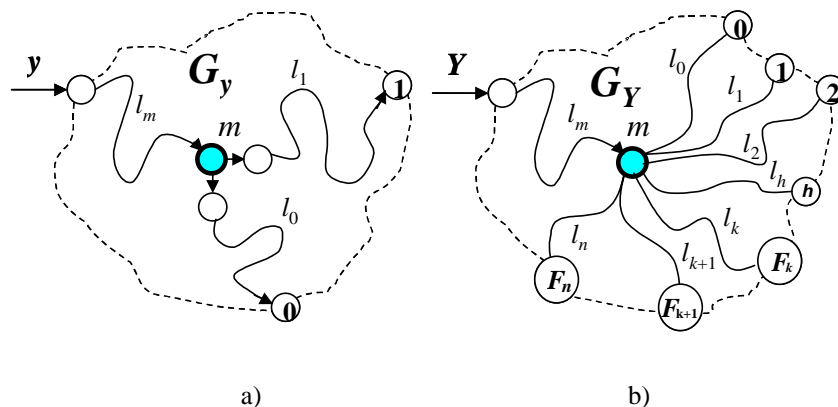
Kui testvektorit rakendada makro sisenditele ja väljundsignaali  $y$  väärtus osutub vigaseks, tuleb kahtlustada rikkeid tippudes  $x_2$  või  $x_{7,1}$  ehk neile vastavatel signaaliteedel skeemis.

Vaadeldud operatsioonide algoritmid põhinevad SSBOD mudeli topoloogilisel töötusel ja seetõttu võib siin Boole'i algebra konteksti kõrvale jätta.

Omistame aktiveeritud tee graafis põhimõtteliselt uude tähenduse: selle asemel, et käsitleda aktiveeritud teed kui Boole'i diferentsiaalvõrrandi lahendit ehk ühte termini disjunktiivses või konjunktiivses normaalkujus, võime seda vaadelda kui tagajärje-põhjuse (testide genereerimine) või põhjuse-tagajärje (rikete diagnostika) funktsiooni.

Kui tahame kavandada testi mingite objektide (graafitippude) käitumise kontrollimiseks, siis tuleb konstrueerida niisugune topoloogiline situatsioon graafis, kus graafiga esitatava funktsiooni väärtus hakkab sõltuma sellest, millises suunas testitavast tipust väljutakse.

Binaarses otsustusdiagrammis  $G_y$  (joonis 1-21a) tuleb selleks aktiveerida kolm teed (nn *kolmjalg*): tee  $l_m$  graafi algusest testitavasse tipu  $m$  ja tipu  $m$  naabritest kaks teed  $l_0$  ja  $l_1$  vastavalt 0-terminali ja 1-terminali. Sõltuvalt tipust  $m$  väljumise suunast, saame funktsioonile erinevad väärtused.



**Joonis 1-21.** Binaarne ja kõrgtaseme otsustusdiagramm

Üldistades nüüd otsustusdiagrammi mõistet kõrgtaseme otsustusdiagrammiks  $G_Y$  (joonis 1-21), lubame muutujatele  $x(m)$  rohkem



väärtusi kui 0 või 1 ning laseme ka terminaltippude arvu vabaks. Terminalides võivad, kuid ei pea enam olema konstandid  $0, 1, 2, \dots, h, \dots, k-1$ , vaid seal võivad olla ka suvalised funktsionaalsed avaldised  $F_k, F_{k+1}, \dots, F_n$ . Tipu  $m$  testimise ülesanne topoloogilises mõttes jääb aga ikka samaks: on vaja aktiveerida algtee  $l_m$  ja abiteed  $l_0, l_1, \dots, l_{k-1}, l_k, l_{k+1}, \dots, l_n$  tipu  $m$  naabritest vastavalt terminalidesse  $0, 1, \dots, k-1, F_k, F_{k+1}, \dots, F_n$ .

#### 1.2.4. Otsustusdiagrammide üldjuhtum

Olgu *digitaalsüsteem* kirjeldatav funktsioonina  $Y = F(X) = F(X_1, X_2, \dots, X_n)$ , kus  $X_j \in X$  on täisarvulised muutujad *määramispiirkondadega*  $V(X_j)$  ning funktsioon  $F$  on esitatatav kujul  $F: V(X_1) * V(X_2) * \dots * V(X_n) \rightarrow V(Y)$ , kus  $V(Y)$  on funktsiooni *muutumispõirkond* ja  $*$  tähistab *Descartes'i korrutist*. Defineerime digitaalsüsteemi  $y = F(X)$  esitavat otsustusdiagrammi (OD) üldjuhul kui *orienteeritud tsükliteta graafi*  $G_Y = (M, X, \Gamma)$ , kus  $M = \{m\}$  on *tippude* hulk,  $X = \{X_1, X_2, \dots, X_n\}$  on funktsiooni argumentide hulk,  $X(m) \in X$  on muutuja, mis on seotud tipuga  $m$ , ning  $\Gamma(M, X)$  on funktsiooniga  $Y = F(X)$  üheselt määratud relatsioon, kus  $\Gamma(m, X(m))$  määrab tipu  $m$  järglase vastavalt *tipumuutuja*  $X(m)$  määramispiirkonnale  $V(X(m))$ . Vektoriga  $X^*$  on graafis määratud liikumine tipust tippu. Igal tipul  $m$  on  $|V(X(m))|$  väljuvat kaart ning vastavalt  $k \leq |V(X(m))|$  naabrit  $m^e$ ,  $e \in V(X(m))$ . Graafis on  $N$  *terminaltippu*  $m^{T,i}$ ,  $i = 1, 2, \dots, N$ ,  $N \geq 2$ , märgenditega  $x(m^{T,i})$ , mis võivad olla konstandid, muutujad  $X_j \in X$  või funktsionaalsed avaldised muutujatest  $X_j \in X$ .

Me ütleme, et tipumuutuja  $X(m)$  väärtus aktiveerib mingi tipu  $m$  väljundkaare. Mingi tee on graafis aktiveeritud siis, kui kõik seda teed moodustavad kaared on aktiveeritud. OD on aktiveeritud väärtusele  $e \in V(Y)$  siis, kui graafis on aktiveeritud tee algtipust (juurest) mingi terminalini  $m^T$ , nii et  $X(m^T) = e$ . Otsustusdiagramm  $G_Y = (M, X, \Gamma)$  esitab funktsiooni  $Y = F(X)$  siis ja ainult siis, kui iga vektori  $X^*$  jaoks, mis viib graafis terminaltippu  $m^T$ , kehtib  $F(X^*) = X(m^T)$ .

Sõltuvalt digitaalsüsteemi klassist või süsteemi esitustasemest võime klassifitseerida eri tüüpi otsustusdiagramme erinevate muutujate ja tippude interpretatsioonidega.

Vaatleme kahte otsustusdiagrammide sünteesi meetodit: süsteemi *protseduurse mudeli sümbolsimuleerimine* ja süsteemi struktuurse OD-võrgu superpositsioon. Viimasel juhul peavad süsteemi komponentide OD-mudelid olema eelnevalt juba sünteesitud.

### OD süntees süsteemi protseduurse mudeli sümbolsimuleerimise abil

Olgu antud mingi süsteemi protseduurne kirjeldus. Sümbolsimuleerimise teel genereeritakse puu protseduuri kõigist võimalikest teedest (süsteemi *tööviisidest*). Iga tee jaoks leitakse kõigi vajalike tingimuste hulk  $C$ , mille korral tee realiseerub, ja tegevuste (operatsioonide) hulk  $P$ , mis antud tingimustel realiseeruvad. Hulgad  $C$  ja  $P$  on aluseks OD-mudeli konstrueerimisel.

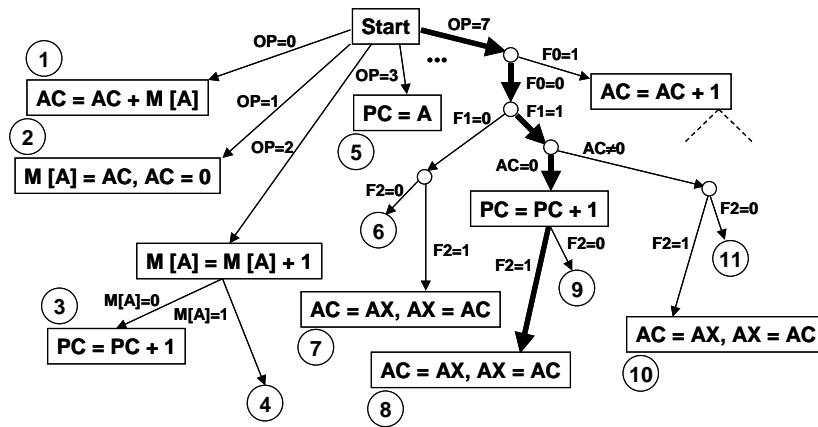
```
BEGIN
Memory state:      M
Processor state:   PC, AC, AX
Internal state:    TMP
Instruction format: IR = OP. A. F0. F1. F2.
Execution process: EXEC:
                  BEGIN
                  DECODE OP ⇒ (
0: AC ← AC + M? A?
1: M[A] ← AC, AC ← 0
2: M[A] ← M[A] + 1,
   IF M[A] = 0 THEN PC ← PC + 1
3: PC ← A
.....
7: IF F0 THEN AC ← AC + 1
   IF F1 THEN IF AC = 0 THEN PC ← PC + 1
   IF F2 THEN (TMP ← AC, AC ← AX, AX ← TMP)
                  END
END
```

### Joonis 1-22. Fragment protsessori protseduursest kirjeldusest

Vaatleme näitena fragmenti ühe protsessori protseduursest kirjeldusest joonisel 1-22.

Protsessori andmemuutujateks on  $M$ ,  $PC$ ,  $AC$  ja  $AX$ , juhtmuutujateks on käsuformaadi väljad: operatsioonikood  $OP$  ja loogikatingimused (lipud)  $F0$ ,  $F1$  ja  $F2$ . Täisarvulise muutuja  $OP$ , mille määramispiirkonnaks on  $V(OP) = \{0,1,2, \dots,7\}$  ja binaarmuutujate  $F0$ ,  $F1$  ja  $F2$  kõigi väärtuste läbi simuleerimisel genereeritaks puu joonisel 1-23.

Iga haru selles puus kujutab ühte võimalikku teed läbi protseduuri joonisel 1-22. Puu kaarte on märgitud tingimused hulgast  $C$ , mille puhul kaar realiseerub, puu tippudes on aga esitatud operatsioonid hulgast  $P$ , mis vastava tee läbimisel täide viiakse. Sümbolsimuleerimise puu alusel koostatakse loetelu kõikvõimalikest paaridest – tingimuste alamhulgast ja operatsioonidest (süsteemi töörežiimidest), mis antud tingimustel toimuvad.



Joonis 1-23. Sümbolsimuleerimise puu

Näide niisugusest loetelust joonisel 1-23 toodud puu jaoks on tabelis 1-2.

Tabel 1-2

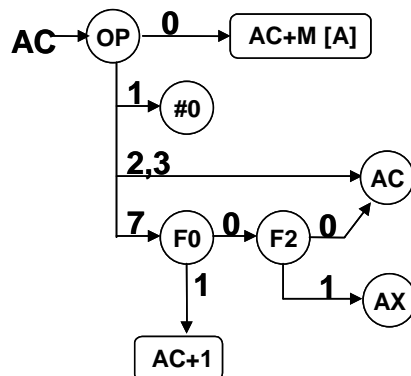
Nimistu süsteemi töörežiimidest

No	Input assertions	Output assertions
1	OP = 0	AC = AC + M[A]
2	OP = 1	M[A] = AC, AC = 0
3	OP = 2, M[A] + 1 = 0	M[A] = M[A] + 1, PC = PC + 1
4	OP = 2, M[A] + 1 ≠ 0	M[A] = M[A] + 1
5	OP = 3	PC = A
6	OP = 7, FO=0, F1=0, F2=0	NO CHANGE
7	OP = 7, FO=0, F1=0, F2=1	AC = AX, AX = AC
8	OP = 7, FO=0, F1=1, AC=0, F2=1	AC = AX, AX = AC
9	OP = 7, FO=0, F1=1, AC=0, F2=0	NO CHANGE

Tabeli paremal poolel esitatud operatsioonide hulgast ekstraheerime funktsiooni muutujad (avaldiste vasakud pooled)  $Y = \{AC, AX, M, PC\}$ . Nende muutujate jaoks on võimalik konstrueerida otsustusdiagrammid, kusjuures tingimustega tabeli 1-2 vasakust pooldest märgendame OD mitteterminaalseid tippe ja operatsiooniavaldiste paremate pooltega tabeli paremast pooldest märgendame OD terminaltipud. Nii näiteks muutuja AC jaoks, viies läbi vajalikud lihtsustused, leiame järgmise sõltuvuse:

$$AC = (OP = 0) (AC + M[A]) \vee (OP = 1) (0) \vee [(OP = 2) \vee (OP = 3)] AC \vee \\ \vee (OP = 7) [(F0 = 0) ((F2 = 0) AC \vee (F2 = 1) AX) \vee \\ \vee (F0 = 1) (AC + 1)]$$

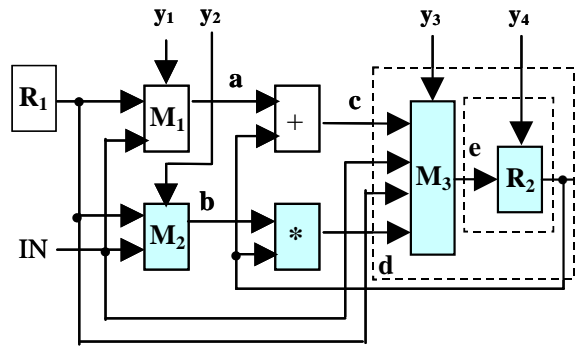
millele vastab otsustusdiagramm joonisel 1-24.



**Joonis 1-24.** Otsustusdiagramm funktsiooni muutuja AC jaoks

#### *OD süntees superpositsiooni meetodil*

Joonisel (1-25) on esitatud digitaalsüsteemi *operatsioonautomaat*, mis koosneb järgmistest *operatsioonielementidest*: registritest  $R_1$  ja  $R_2$ , kolmest multiplexerist  $M_1$ ,  $M_2$ ,  $M_3$ , summaatorist ja korrutajast. Süsteemi juhtmuutujateks on sisendsignaalid  $y_1, y_2, y_3, y_4$  ning andmesisendiks sõnamuutuja  $IN$ . Komponentide  $M_1, M_2, M_3$  ja  $R_2$  funktsioonid on tabelis 1-3. Nendele funktsioonidele vastavad otsustusdiagrammid  $G_{M,1}, G_{M,2}, G_{M,3}, G_{R,2}$  on joonisel 1-26.



Joonis 1-25. Digitaalsüsteemi operatsioonautomaat

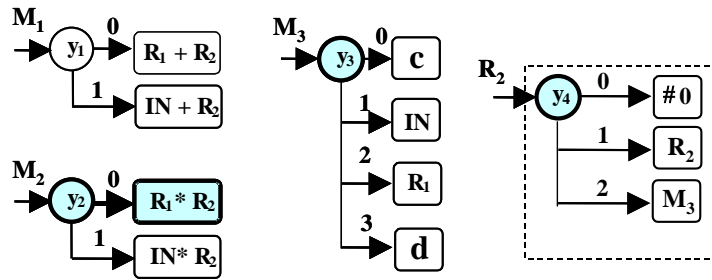
Tabel 1-3

Joonisel 1-25 esitatud operatioonelementide funktsioonid

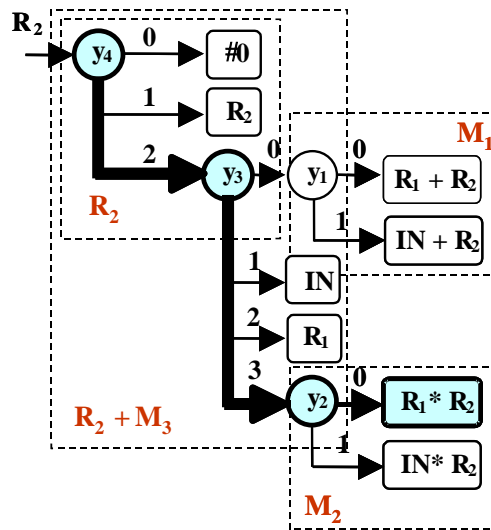
M <sub>1</sub>		M <sub>2</sub>		M <sub>3</sub>		R <sub>2</sub>		
y <sub>1</sub>	Function	y <sub>2</sub>	Function	y <sub>3</sub>	Function	y <sub>4</sub>	Operation	F
0	M <sub>1</sub> = R <sub>1</sub>	0	M <sub>2</sub> = R <sub>1</sub>	0	M <sub>3</sub> = M <sub>1</sub> + R <sub>2</sub>	0	Reset	R <sub>2</sub> = 0
1	M <sub>1</sub> = IN	1	M <sub>2</sub> = IN	1	M <sub>3</sub> = IN	1	Hold	R <sub>2</sub> = R' <sub>2</sub>
				2	M <sub>3</sub> = R <sub>1</sub>	2	Load	R <sub>2</sub> = M <sub>3</sub>
				3	M <sub>3</sub> = M <sub>2</sub> * R <sub>2</sub>			

Viies järgnevalt läbi superpositsiooni protseduuri, kus kõigepealt, asendades graafis  $G_{R,2}$  tipu  $M_3$  graafiga  $G_{M,3}$ , seejärel saadud graafis tipu  $c$  graafiga  $G_{M,1}$  ja tipu  $d$  graafiga  $G_{M,2}$ , saame uue superponeeritud graafi  $G_{R,2}$  joonisel 1-27, mis esitab kompaktselt operatsioonautomaati joonisel 1-25.

Otsustusdiagrammi  $G_{R,2}$  eeliseks võrreldes traditsioonilise komponentide võrguga joonisel 1-25 on efektiivsuse tõus süsteemi simuleerimisel ja diagnostilisel modelleerimisel. Näiteks selle asemel, et juhtvektori  $y_1, y_2, y_3, y_4 = 0032$  simuleerimiseks viia läbi lähteskeemis joonisel 1-25 arvutusi  $a = R_1, b = R_1, c = a + R_1, d = b * R_1, e = d$ , ja  $R_2 = e$ , tuleks OD-mudelil  $G_{R,2}$  joonisel 1-27 läbi käia aktiveeritud tee läbi tippude  $y_4, y_3$  ja  $y_2$  terminaltippu, kus oleks vaja sooritada üksainus tehe  $R_2 = R_1 * R_2$ .



Joonis 1-26. OD-d süsteemi komponentide jaoks joonisel 1-25



Joonis 1-27. Otsustusdiagramm operatsioonautomaadi jaoks joonisel 1-25

Diagnostilise analüüsi efektiivsus otsustusdiagrammidel tõuseb sellest, et tagajärje-põhjuse funktsionaalsed suhted on siin paremini korrastatud ning järeldused tulenevad vahetult. Näiteks väära signaali avastamisel testvektori  $y_1, y_2, y_3, y_4 = 0032$  puhul operatsioonautomaadi väljundis

tulenevad võimalikud veapõhjused otseselt aktiveeritud teel läbitud tippudest.

Testide genereerimise tõhusus tõuseb OD-mudelit kasutades antud juhul sellest, et operatsioonautomaadi alamfunktsioonide superponeerimise tõttu õnnestub testi genereerida vahetult sisendmuutujate peal, ilma et oleks vaja läbi viia täiendavaid sisemuutujate väärtuste arvutamisi.

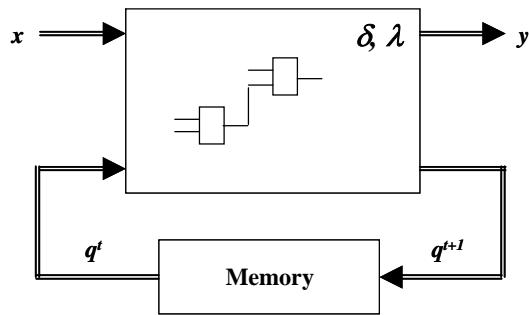
Näiteks selleks et sünteesida testi korrutajale joonisel 1-25 toodud skeemi abil, oleks vaja levitada nii stiimuleid, alates skeemisisenditest läbi skeemikomponentide testitava korrutaja sisenditeni kui ka transportida korrutaja reaktsioone läbi skeemikomponentide väljundregistrini  $R_2$ . Kasutades aga OD-mudelit on vaja üksnes leida terminaltipp korrutaja funktsiooniga ja aktiveerida graafil tee, mis viib sellesse tippu.

### 1.2.5. Otsustusdiagrammid juhtautomaatidele

Digitaalsüsteemide *juhtautomaate* võib esitada nii struktuuri kui ka funktsioonide kaudu. Esimesel juhul vaatleme automaati *Hufmanni* mudelina (joonis 1-28) [Gri 99], kus automaat on dekomponeeritud kombinatsioonskeemiks ja mäluskeemiks, ning teisel juhul vaatleme automaati väljundite ja üleminekute diagrammina.

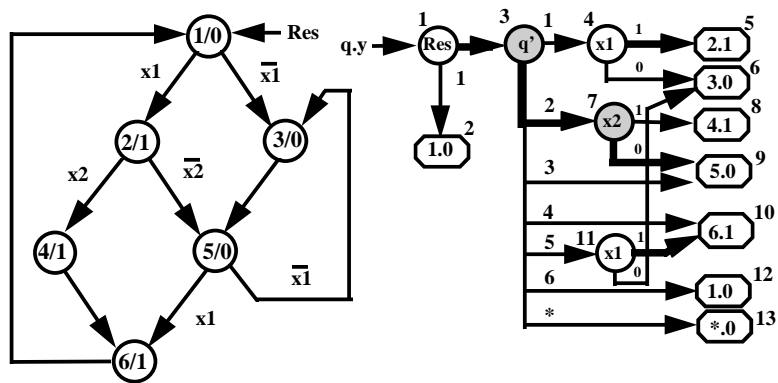
Struktuurse käsitusviisi puhul, kus automaadi kombinatsioonskeemi osa, mis realiseerib *väljundfunktsioone*  $\lambda$  ja *siirdefunktsioone*  $\delta$ , esitatakse ventiilide võrguna, võime kasutada SSBOD mudelit juba vaadeldud kujul. Testide genereerimise efektiivsuse tõstmist silmas pidades aga pole automaatide struktuurset käsitlemisest ventiilide tasandil kasu ülesande keerukuse tõttu. Lõplike automaatide diagnostilise analüüsi efektiivsuse tõstmiseks oleks vaja neid käsitleda just kõrgematel funktsionaalsetel tasanditel.

Jättes lahtiseks automaadi olekute ja sisend-väljundmuutujate kodeerimise, võime esitada automaadi sisend-, väljund- ja olekumuutujad täisarvulistena, mis võimaldab automaadi funktsioonide esitamisel loobuda keerukast loogikatasandist.



Joonis 1-28. Lõpliku automaadi Hufmanni mudel

Piirdudes praegu üksnes Moore'i automaadi käsitlemisega, võime nii väljundmuutuja  $y$  kui ka olekumuutuja  $q$  esitada *konkatenatsiooni* abil üheainsa liitmuutujana  $q.y$  ja konstrueerida automaadi *integreeritud funktsiooni* (vektorfunktsiooni) kujul  $q.y = f(q',x)$ , kus  $q'$  tähistab automaadi eelmist olekut. Niisuguse funktsiooni saaks esitada üheainsa vektorotsustusdiagrammina (joonis 1-29).



Joonis 1-29. Lõpliku automaadi üleminekute-väljundite graaf ja otsustusdiagramm



### Näide 1-16

Joonisel 1-29 on esitatud juhtautomaadi olekute ja üleminekute ühitatud diagramm ja OD-mudel. Otsustusdiagramm esitab automaadi liit-ehk vektorfunktsiooni  $q.y = F(q', Res, x_1, x_2)$ , kus  $q.y$  on konkatenatsioon täisarvulisest muutujast  $q$  määramispiirkonnaga  $V(q) = \{1,2,3,4,5,6\}$  ja väljundmuutujast  $y$ , mis on binaarne,  $Res$  (*reset*) on automaadi binaarne *seademuutuja* ( $Res = 1$  viib automaadi algolekusse  $q = 0$ ) ja  $x_1$  ning  $x_2$  on automaadi binaarsed sisendmuutujad. Apostroofiga ( $q'$ ) tähistame automaadi olekut eelmisel taktil. Terminaltipud on mudelis märgendatud *liitkonstantidega* – liitmuutuja  $q.y$  võimalike väärtustega (vastavalt, järgmine olek ja väljundi  $y$  väärtus). Võimalike rikete tõttu tekkida võiva *illegaalse* (*keelatud*) oleku tähistamiseks kasutame täрни \* (mudelis eeldame, et illegaalse oleku puhul on automaadi väljundmuutuja  $y$  väärtus 0).

Võib vaadelda kahte äärmuslikku juhtu lõplike automaatide funktsionaalsel esitusel:

- 1) kasutame automaadi modelleerimiseks vaid kolme täisarvulist abstraktset muutujat sisendi, väljundi ja olekute esitamiseks ning vastavalt sellele ka vaid kahte (või ühte integreeritud) otsustusdiagrammi üleminekute (siirete) ja väljundkäitumise kirjeldamiseks;
- 2) automaadi sisendid, väljundid ja olekud kodeeritakse; sellisel juhul võib automaati esitada binaarfunktsioonide süsteemina.

### Näide 1-17

Vaatleme otsustusdiagrammide sünteesi juhtautomaadile *VHDL kirjeldusest*. Joonisel 1-30 on esitatud neljast *protsessist* koosnev juhtautomaadi VHDL kirjeldus. Iga protsessi väljundmuutuja käitumise diagnostiliseks modelleerimiseks sünteesime sellele protsessile vastava otsustusdiagrammi *sümbolsimuleerimise* meetodil.

Lihtsustuste läbiviimiseks võtame kasutusele järgmise tähistuse. Olgu

$$CL = \{c\uparrow, c, c\downarrow, \neg c\}$$

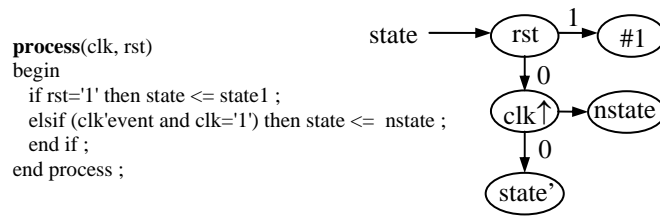
loogikamuutujate hulk tähistamaks erinevat tüüpi taktisignaali seotud protsesside aktiveerimistingimusi.

```

entity rd_pc is
  port
    ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
      reg : out bit ; outreg : out bit ; fin : out bit ) ;
end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
  (state1, state2);
  signal state, nstate: STATETYPE ; signal enable_in : bit ;
  signal reg_cp_comb : bit ;
begin
  seq: process(clk, rst)
  begin
    if rst='1' then state <= state1 ;
    elsif (clk'event and clk='1') then state <= nstate ;
    end if ;
  end process ;
  process(clk, enable)
  begin
    if clk='1' then enable_in <= enable ;
    end if ;
  end process ;
  process(clk, reg_cp_comb)
  begin
    if clk='0' then reg_cp <= reg_cp_comb ;
    end if ;
  end process ;
  comb: process (state, rb0, enable_in)
  begin
    case state is
      when state1 => outreg <= '0' ; fin <= '0' ;
      if (enable_in='0') then nstate <= state1 ;
        reg <= '1' ; reg_cp_comb <= '0' ;
        else nstate <= state2 ; reg <= '1' ;
        reg_cp_comb <= '1' ;
      end if ;
      when state2 =>
      if (rb0='1') then nstate <= state2 ; reg <= '0' ;
        reg_cp_comb <= '1' ; outreg <= '0' ; fin <= '0' ;
        elsif (enable_in='0') then nstate <= state1 ; reg <= '0' ;
        reg_cp_comb <= '0' ; outreg <= '1' ; fin <= '1' ;
        else nstate <= state2 ; reg <= '0' ;
        reg_cp_comb <= '0' ; outreg <= '0' ; fin <= '1' ;
      end if ;
    end case ;
  end process ;
end archi_rd_pc ;

```

*Joonis 1-30. Juhtautomaadi VHDL kirjeldus*

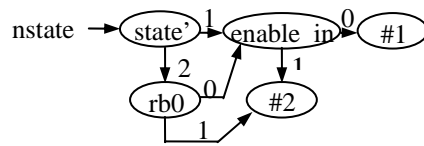


**Joonis 1-31.** VHDL protsessi “state” otsustusdiagramm

```

comb: process (state, rb0, enable_in)
begin
  case state is
    when state1 => outreg <= '0' ; fin <= '0' ;
      if (enable_in='0') then nstate <= state1 ;
        reg <= '1' ; reg_cp_comb <= '0' ;
      else nstate <= state2 ; reg <= '1' ;
        reg_cp_comb <= '1' ;
      end if ;
    when state2 =>
      if (rb0='1') then nstate <= state2 ; reg <= '0' ;
        reg_cp_comb <= '1' ; outreg <= '0' ; fin <= '0' ;
      elsif (enable_in='0') then nstate <= state1 ; reg <= '0' ;
        reg_cp_comb <= '0' ; outreg <= '1' ; fin <= '1' ;
      else nstate <= state2 ; reg <= '0' ;
        reg_cp_comb <= '0' ; outreg <= '0' ; fin <= '1' ;
      end if ;
    end case ;
  end process ;

```



**Joonis 1-32.** VHDL protsessi “nstate” otsustusdiagramm

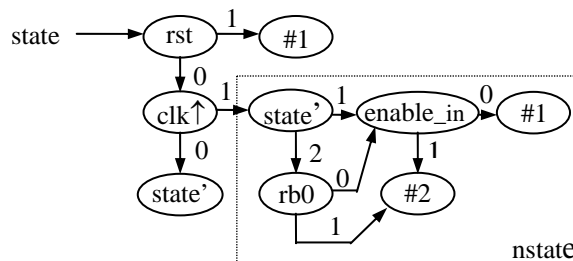
VHDL kirjelduse kontekstis tähendab  $c\uparrow$ , et protsess on aktiveeritud, kui  $clk'event \ \& \ clk = '1'$  on tõene,  $c$  tähendab, et protsess on aktiveeritud, kui  $clk = '1'$  on tõene,  $c\downarrow$  tähendab, et protsess on aktiveeritud, kui  $clk'event \ \& \ clk = '0'$  on tõene ja  $\neg c$  tähendab, et protsess on aktiveeritud, kui  $clk = '0'$  on tõene. Nagu varemgi tähendab apostroof muutuja juures, et muutuja väärtus tuleb võtta eelneval ajahetkel. Defineerime ka järjestuse

$$c\uparrow < c < c\downarrow < \neg c$$

määramaks, millises järjekorras protsessid ühe ja sama taktperioodi vältel aktiveeritakse

VHDL protsessidele *state* ja *nstate* vastavad otsustusdiagrammid  $G_{state}$  ja  $G_{nstate}$  on esitatud vastavalt joonistel 1-31 ja 1-32. DD sünteesiks on kasutatud peatükis 1.2.4 vaadeldud sümbolistlikku simuleerimise meetodit. Graafi  $G_{nstate}$  sünteesil on seejuures simuleeritud üksnes muutuja *nstate* käitumist. Ülejäänud väljundmuutujatele *reg.*, *reg\_cp*, *outreg* ja *fin* tuleks sünteesida nende vastavad otsustusdiagrammid.

Kahest graafist  $G_{state}$  (joonis 1-31) ja  $G_{nstate}$  (joonis 1-32) koosnevat DD-mudelit saab lihtsustada, viies läbi graafide *superpositsiooni*. Selleks asendame tipu *nstate* graafis  $G_{state}$  graafiga  $G_{nstate}$ . Uus superponeeritud graaf on joonisel 1-33, kusjuures iseseisev graaf  $G_{nstate}$  on nüüd mudelist kadunud, olles graafi  $G_{state}$  alamgraaf.



**Joonis 1-33.** Protsessi “state” otsustusdiagramm pärast superpositsiooni

Joonisel 1-34 on toodud VHDL protsessi *enable\_in* otsustusdiagramm  $G_{enable\_in}$ . Kuna muutuja *enable\_in* on tipumuutujaks graafis  $G_{state}$ , siis on nüüd võimalik mudelit veelgi lihtsustada, asendades tipu *enable\_in* graafis  $G_{state}$  graafiga  $G_{enable\_in}$ . Uus graaf  $G_{state}$  on joonisel 1-35a.

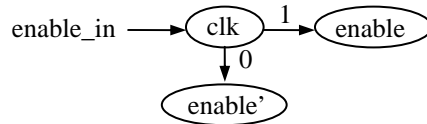
Võttes arvesse taktsignaali seotud muutujate ajalisi järgnevust  $c\uparrow < c < c\downarrow < \neg c$  ning ka seda, et taktsignaali ennast ei pea sünkroonskeemide mudelis

ilmtingimata otseselt esitama [Lev98], võime läbi viia täiendavad lihtsustused ja esitada otsustusdiagrammi  $G_{state}$  veelgi kompaktsemalt joonisel 1-35b.

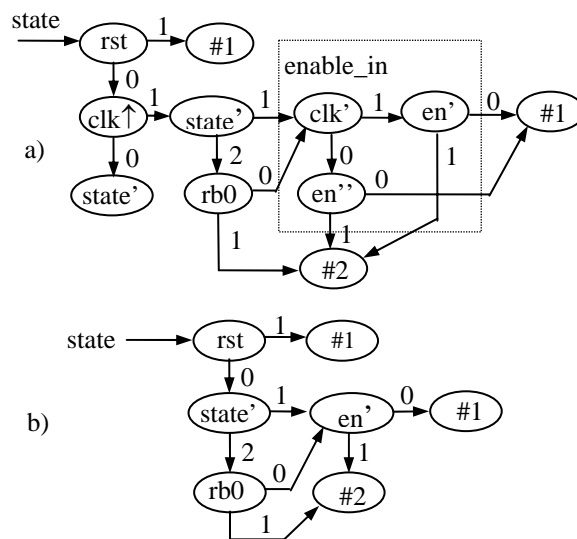
```

process(clk, enable)
begin
  if clk='1' then enable_in <= enable ;
  end if ;
end process ;

```

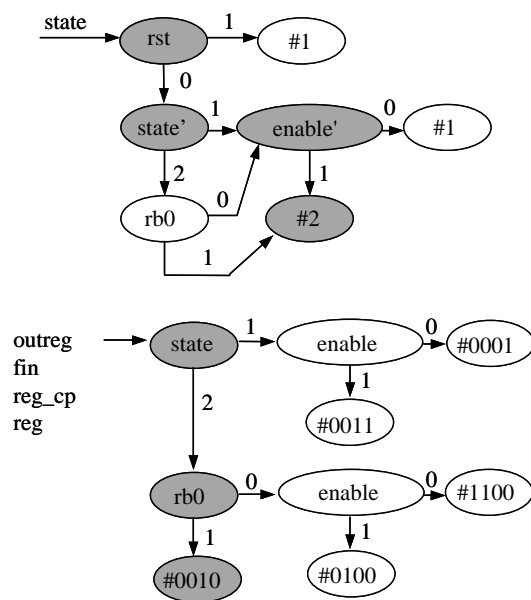


**Joonis 1-34.** Protsessi “enable” otsustusdiagramm



**Joonis 1-35.** Tipu “enable\_in” superpositsioon graafis  $G_{state}$

Superpositsiooni tulemusena on nüüd osa signaale esialgselt VHDL-mudelist (joonis 1-30), mille kaudu protsessid kommunikeeruvad, ja ka vastavatest originaalotsustusdiagrammidest elimineeritud. Nii näiteks on elimineeritud graafist  $G_{state}$  (joonis 1-31) muutujad  $clk\uparrow$  ja  $nstate$ , graafist  $G_{nstate}$  (joonis 1-32)  $enable\_in$  ja graafist  $G_{enable\_in}$  (joonis 1-34)  $clk'$  ja  $en''$ . Lihtsustuste tulemusena on esialgse VHDL-mudeli üleminekufunktsioonide esitus graafina  $G_{state}$  joonisel 1-35 väga kompaktne. See omakorda tõstab modelleerimise kiirust ja vähendab otsinguruumi mahtu testide genereerimisel.



**Joonis 1-36.** Juhtautomaadi VHDL kirjelduse lõplik esitus OD-mudelina

Joonisel 1-36 on esitatud VHDL keeles kirjeldatud juhtautomaadi lõplik OD-mudel kahe graafina. Esimene graaf  $G_{state}$  esitab automaadi üleminekute funktsiooni, teine graaf väljundite funktsiooni. Väljundmuutuja on vektoriaalne, selle argumentideks on binaarmuutujad  $outreg$ ,  $fin$ ,  $reg\_cp$  ja  $reg$ . Kasutades vektormuutujat, on võimalik väljundfunktsiooni mudelit esitada kompaktselt üheainsa vektorgraafina.

OD-mudel võimaldab lihtsustada testide genereerimist võrreldes esialgse VHDL kirjeldusega, kuna kogu funktsionaalses mõttes ebaoluline info on OD-mudelist eemaldatud. Käies OD-mudelis joonisel 1-36 läbi kõik teed ja aktiveerides neid, saaksime genereerida vaid kuuest vektorist koosneva funktsionaalse testjada, mis on ära toodud tabelis 1-4. Tabelis värvitud testvektorile vastavad aktiveeritud teed on OD-mudelis joonisel 1-36 samuti värviga esile toodud.

**Tabel 1-4**

*Testide genereerimine OD-mudelil joonisel 1-30 esitatud automaadile*

time	1	2	3	4	5	6
rst	1	0	0	0	0	0
enable	0	1	1	1	0	0
rb0	x	x	1	0	0	0
state	1	1	2	2	2	1
outreg	0	0	0	0	1	0
fin	0	0	0	1	1	0
reg_cp	0	1	1	0	0	0
reg	1	1	0	0	0	1

### 1.2.6. Otsustusdiagrammid digitaalsüsteemidele registersiirete tasandil

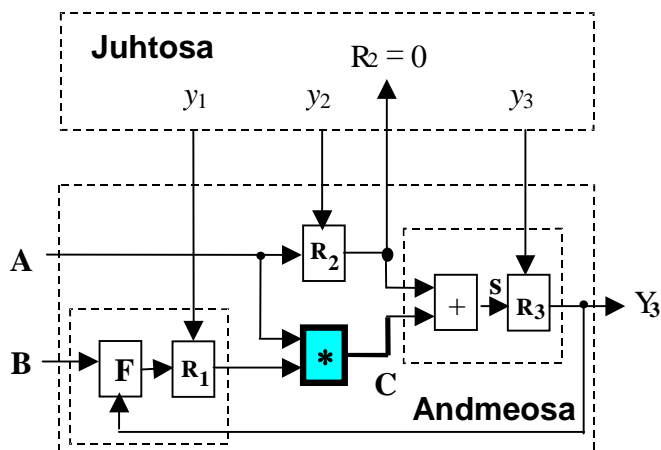
Digitaalsüsteemide kirjeldamiseks OD mudeli abil *registersiirete tasandil* tuleb kõigepealt esitada süsteem sobiva komponentide või alamsüsteemide võrguna ja seejärel kirjeldada alamsüsteemide funktsioone otsustusdiagrammidena. Alamsüsteemiks on otstarbekas valida register koos sisendloogikaga nii viisi, et alamsüsteemi sisendmuutujateks oleksid süsteemi primaarsed sisendid või registrid. Niisugune mudel võimaldab efektiivselt läbi viia nn *tsükli põhise simuleerimist*, kus iga süsteemi takti (tsükli) jaoks arvutatakse uus süsteemi olek registermuutujate uute väärtuste näol.

Üldjuhul võib registersiirete tasandil esitatud süsteemi käsitleda kui kompositsiooni juht- ja operatsioonautomaadist. Juhtautomaadi oleku- ja väljundmuutujaid võib seejuures interpreteerida vastavalt kui *mikrokäskude*

aadresse ja juhtõnu (või juhtvälju) ja operatsioonautomaadi muutujaid kui andmesõnu.

*Näide 1-18*

Vaatleme juhtautomaadist ja operatsioonautomaadist koosnevat digitaalsüsteemi joonisel 1-37.



*Joonis 1-37. Digitaalsüsteem*

Operatsioonautomaat koosneb kolmest registrist  $R_1$ ,  $R_2$  ja  $R_3$ , kolmest funktsionaalsest plokist – korrutajast, summaatorist ning eriooperatsiooni-  
elemendist  $F$ . Süsteemi andmesisendid on  $A$  ja  $B$  ja andmeväljundiks on  $Y$ .

Juhtautomaat juhib andmeosa juhtsignaalidega  $y_1$ ,  $y_2$  ja  $y_3$ , ning juhtautomaadi sisendiks on operatsioonautomaadis genereeritav loogikatingimus  $R_2 = 0$ . Operatsioonautomaat on dekomponeeritud neljaks alamsüsteemiks väljundmuutujatega  $R_1$ ,  $R_2$ ,  $R_3$  ja  $C$ . Alamsüsteemide funktsioonid on esitatud tabelis 1-5.

Ehkki tsükli põhise modelleerimise idee kohaselt, kus alamsüsteeme moodustavad registrid koos oma sisendloogikatega, peaks korrutaja kuuluma registriga  $R_3$  määratud alamsüsteemi, on siiski otstarbekam eraldada korrutaja ja summaator iseseisvate OD-mudeliga esindatud sõlmedeks. See võimaldab OD



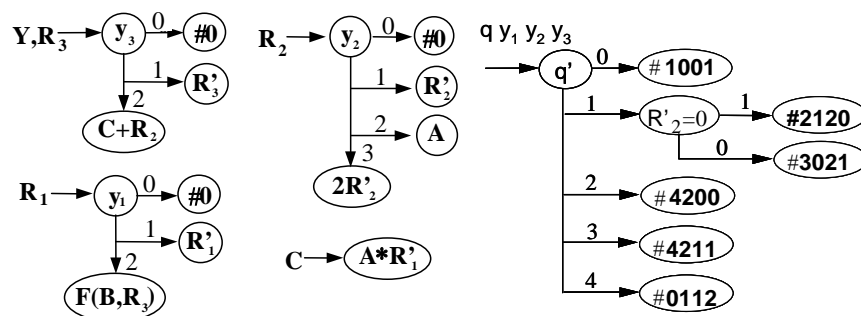
terminaltippudega siduda iseseisvaid standardoperatsioone nagu  $R_i * R_j$  ja  $R_i + R_j$  ning vältida keerulisi ebastandardseid liitfunktsioone näiteks  $R_i * R_j + R_h$ .

**Tabel 1-5**

*Digitaalsüsteemi alamsüsteemide funktsioonid*

Block	Control	RTL operation	Function
R <sub>1</sub>	y <sub>1</sub> = 0	R <sub>1</sub> = 0	Reset
	y <sub>1</sub> = 1	R <sub>1</sub> = R' <sub>1</sub>	Hold
	y <sub>1</sub> = 2	R <sub>1</sub> = F(B, R' <sub>1</sub> )	Special
R <sub>2</sub>	y <sub>2</sub> = 0	R <sub>2</sub> = 0	Reset
	y <sub>2</sub> = 1	R <sub>2</sub> = R' <sub>2</sub>	Hold
	y <sub>2</sub> = 2	R <sub>2</sub> = A	Load
	y <sub>2</sub> = 3	R <sub>2</sub> = 2 * R' <sub>2</sub>	Shift
R <sub>3</sub>	y <sub>3</sub> = 0	R <sub>3</sub> = 0	Reset
	y <sub>3</sub> = 1	R <sub>3</sub> = R' <sub>3</sub>	Hold
	y <sub>3</sub> = 2	R <sub>3</sub> = C + R' <sub>2</sub>	Add
C	None	C = A * R' <sub>1</sub>	Multiply

Joonisel 1-38 on esitatud OD-mudel digitaalsüsteemile joonisel 1-36. Mudel koosneb viiest otsustusdiagrammist:  $G_{q,y_1,y_2,y_3,y_4}$  juhtautomaadile ja  $G_C$ ,  $G_{R,1}$ ,  $G_{R,2}$ , ja  $G_{Y,R,3}$  operatsioonautomaadi neljale alamsüsteemile.



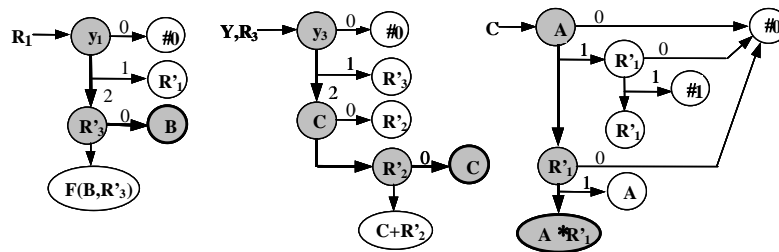
**Joonis 1-38.** Otsustusdiagrammide mudel joonisel 1-32 esitatud süsteemile

Tabel 1-6

Juhtautomaadi üleminekute ja väljundite tabel

State	Condition	Nstate	Control signals		
			$y_1$	$y_2$	$y_3$
$q'$		$q$			
0		1	0	0	1
1	$R_2 = 0$	2	1	2	0
1	$R_2 \neq 0$	3	0	2	1
2		4	2	0	0
3		4	2	1	1
4		0	1	1	2

Juhtautomaadi üleminekute ja väljundite funktsioonid on tabelis 1-6 ning otsustusdiagrammina  $G_{q,y_1,y_2,y_3,y_4}$  joonisel 1-38. Mitteterminaalsed tipud on märgendatud eelmise oleku muutujaga  $q'$  ja automaadi sisendmuutujatega  $x_1$  ning  $x_2$ , terminaltipud aga konstantidega, mis omistatakse graafi vektormuutujale  $q.y_1.y_2.y_3.y_4$ .

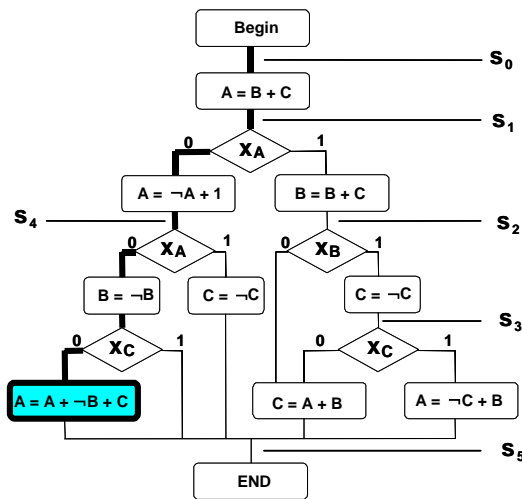
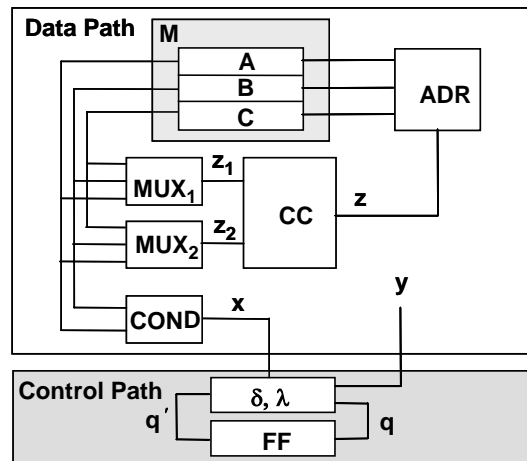


Joonis 1-39. Transparentsusfunktsioonid otsustusdiagrammidel

Joonisel 1-39 on näidatud, kuidas saab kujutada OD-mudelisi alamskeemide funktsioone kui ka *transparentsuse* (läbipaistvuse) tingimusi, mis lubaksid läbi alamskeemide transportida stiimuleid ja veasignaale.

Näiteks on graafi  $G_{R1}$  viidud lisaks kaks tippu  $R'_3$  ja  $B$ , selleks et ilmutatult näidata, kuidas oleks võimalik juhtida vastavat alamskeemi, nii et see muutuks transparentseks signaalide saatmisel sisendist  $B$  registrisse  $R_1$ .

Graafist on näha, et nimetatud töörežiimi aktiveerimiseks tuleks tingimuste  $y_1 = 2$  ja  $R'_3 = 0$  abil aktiveerida tee algtpust  $y_1$  terminaltippu  $B$ .



Joonis 1-40. Digitaalsüsteem ja tema mikroprogramm

Graafis  $G_{Y,R3}$ , on esitatud töörežiimid, mis võimaldavad aktiveerida transparentsed signaaliteed registrist  $R'_2$  väljundisse  $Y$  (kui  $C = 0$ ) registri  $R'_2$  käitumise jälgimiseks ja korrutajast  $C$  väljundisse  $Y$  (kui  $R'_2 = 0$ ) korrutaja käitumise jälgimiseks. Graafis  $G_{Y,R3}$  on värvitult illustreeritud tee, mida tuleks aktiveerida, et väljundis  $Y$  saaks jälgida korrutaja väljundisignaale. Tee aktiveerimise tingimusteks on  $y_3 = 2$ ,  $R'_2 = 0$ . Summaator muutub nendel tingimustel transparentseks: 0 liitmine korrutise tulemusele ei muuda selle väärtust.

Graafis  $G_C$  on lisatud neli täiendavat terminaltippu, mis on tähistatud muutujatega  $R'_1$  ja  $A$  ning konstantidega 0 ja 1. Uues graafis on esitatud korrutamisseadme töörežiimid, mis võimaldavad aktiveerida transparentsed signaaliteed registrist  $R'_1$  väljundisse  $C$  (kui  $A = 1$ ) registri  $R'_1$  käitumise jälgimiseks ja sisendist  $A$  väljundisse  $C$  (kui  $R'_1 = 0$ ) sisendi  $A$  jälgimiseks. Lisaks on sisse viidud võimalus asetada muutuja olekutesse 0 või 1, aktiveerides tee terminaltippu vastavalt valitud konstandile. Värvitult on graafis  $G_{Y,R3}$  illustreeritud tee, mida tuleks aktiveerida, et väljundis  $C$  saaks jälgida seadme põhifunktsiooni – korrutamist.

Kõigi nimetatud töörežiimide (skeemi transparentseks tegemine, asetamine olekutesse 0 või 1) aktiveerimine on tüüpülesanneteks testide genereerimisel, mistõttu vajalike lahendite kiire leidmine mudelist on väga oluline.

#### Näide 1-19

Joonisel 1-40 on digitaalsüsteem koos selle käitumist kirjeldava mikroprogrammiga. Süsteem koosneb nii juht- kui andmeosast.

Juhtosa on esitatud väljundfunktsiooniga  $y = \lambda(q', x)$  ja siirdefunktsiooniga  $q = \delta(q', x)$ . Täisarvuline väljundmuutuja  $y$  esitab mikrokäsku – vektorit, mille komponentideks on mikrokäsuväljad  $y = (y_M, y_{z1}, y_{z2})$  vastavalt mäluregistri valikuks mikrooperatsiooni (mikrotehte) tulemuse salvestamiseks ( $y_M$ ), mikrooperatsiooni valikuks ( $y_{z1}$ ) ning mäluregistrite adresseerimiseks operandide valikul ( $y_{z1}, y_{z2}$ ). Väljundfunktsiooni argumentideks on Boole'i sisendmuutujad  $X = (x_A, x_C)$ , mis esitavad andmeosa poolt genereeritavaid loogikatingimusi, ning täisarvuline olekumuutuja  $q'$ , kus apostroof viitab olekule eelmisest taktist. Juhtautomaat on esitatav kahe otsustusdiagrammiga  $q = G_q(q', x)$ ,  $y = G_y(q', x)$ ,

Andmeosa koosseisu kuuluvad mäloplokk  $M$ , mis koosneb kolmest registrist  $A, B, C$  koos adresseerimisskeemiga  $ADR$ , ning mis on esitatav kolme otsustusdiagrammiga  $A = G_A(y_M, z)$ ,  $B = G_B(y_M, z)$ ,  $C = G_C(y_M, z)$ ,

andmetöötusplokk  $CC$ , mida esitab graaf  $z = G_z(y_z, z_1, z_2)$ ; ja kaks multiplekserit  $z_1 = G_{z,1}(y_{z,1}, M)$  ning  $z_2 = G_{z,2}(y_{z,2}, M)$ . Loogikatingimuste arvutamiseks on plokk COND funktsiooniga  $x = G_x(A, C)$ .

Süsteemi kõrgtaseme struktuurne mudel on niisiis esitatav üheksast otsustusdiagrammist koosneva süsteemina

$$N_1 = \{G_q, G_y, G_A, G_B, G_C, G_z, G_{z,1}, G_{z,2}, G_x\}$$

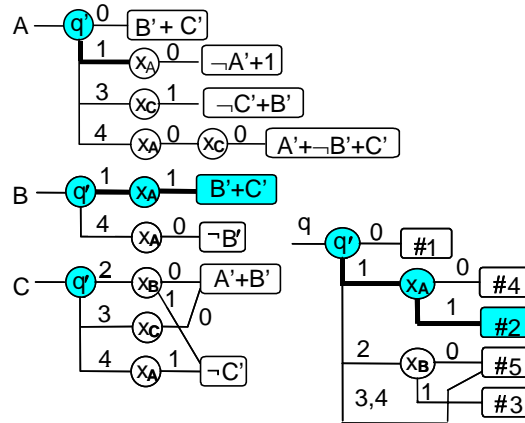
Superponeerides graafi  $A = G_A(y_M, z)$  genereerime nüüd registri  $A$  modelleerimiseks uue kompaktsema mudeli

$$\begin{aligned} A &= G_A(y_M, z) = G_A(y_M, G_z(y_z, z_1, z_2)) = \\ &= G_A(y_M, G_z(y_z, G_{z,1}(y_{z,1}, M), f_4(y_{z,2}, M))) = \\ &= G_A(y_M, y_z, y_{z,1}, y_{z,2}, M) = G_A(y, M) = \\ &= G_A(G_y(q', x), M) = G^*_A(q', A, B, C) \end{aligned}$$

Korrates samasugust graafide superponeerimise võtet ka otsustusdiagrammidele  $G_q, G_B, G_C$  saame kogu süsteemi esitamiseks uue kompaktsema mudeli:

$$N_2 = \{G_q, G^*_A, G^*_B, G^*_C\}$$

kus  $G_q$  esitab juhtautomaati ning graafid  $G^*_A, G^*_B$ , ja  $G^*_C$ , esitavad registreid  $A, B$  ja  $C$  koos sisendloogikaga. Süsteemi uus otsustusdiagrammide kompaktne mudel on joonisel 1-41.



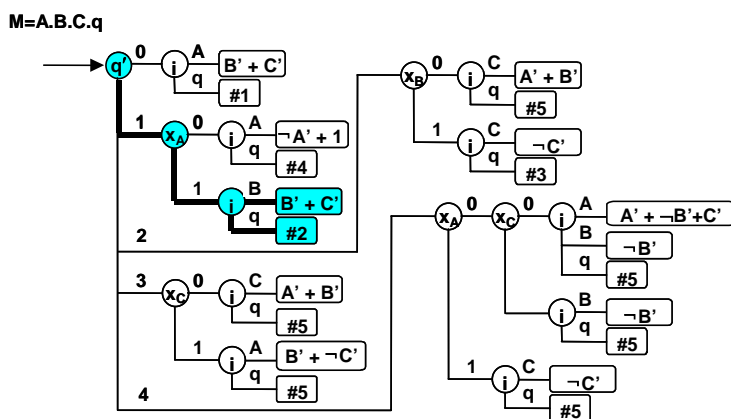
Joonis 1-41. Joonisel 1-40 esitatud süsteemi otsustusdiagrammid

Joonisel 1-42 on konstrueeritud sama digitaalsüsteemi vektorotsustusdiagramm  $G_{q,A,B,C}$ , kus kõik süsteemi esialgse mudeli graafid  $G_q$ ,  $G_A$ ,  $G_B$ , ning  $G_C$  on ühitatud ühte vektorgraafi. Graaf esitab vektorfunktsiooni  $M = A.B.C.q$  ja võimaldab arvutada üheainsa sisenemisega graafi kogu süsteemi olekut ehk registreite  $A,B,C$  ja juhtautomaadi oleku  $q$  uusi väärtusi.

Vektorgraafide puhul on võimalik terminaltippe esitada kahel viisil: kas igale vektorikomponendile vastava avaldise eri tipuna nii nagu käesoleval juhul joonisel 1-42 või koondada vektoravaldised ühteainsasse tippu nii nagu juhtautomaatide otsustusdiagrammides joonistel 1-29, 1-36, 1-38.

Esimesel juhul tuleb mudelisse viia uut tüüpi tipumuutuja, nn *aadressimuutuja*  $i$ , mille ülesandeks on valida graafis haru vastavalt simuleeritavale vektori komponendile. *Aadressitipust*, mis on märgendatud aadressimuutujaga, väljub üldjuhul maksimaalselt  $n$  haru, kus  $n$  on simuleeritava vektori komponentide arv. Juhul kui mingi komponendi väärtus simuleeritava vektori korral ei muutu, siis vastav väljundharu aadressitipul puudub.

Nii näiteks liikudes joonisel 1-42 piki märgistatud teed, sattume aadressitippu  $i$ , mis suunab meid arvutama vektori komponente  $A$  ja  $q$ . Nagu graafist näha, siirdudes olekust 0 olekusse 1 säilitavad komponendid  $B$  ja  $C$  oma väärtusi, seega ei ole neid vaja adresseerida ega arvutada.



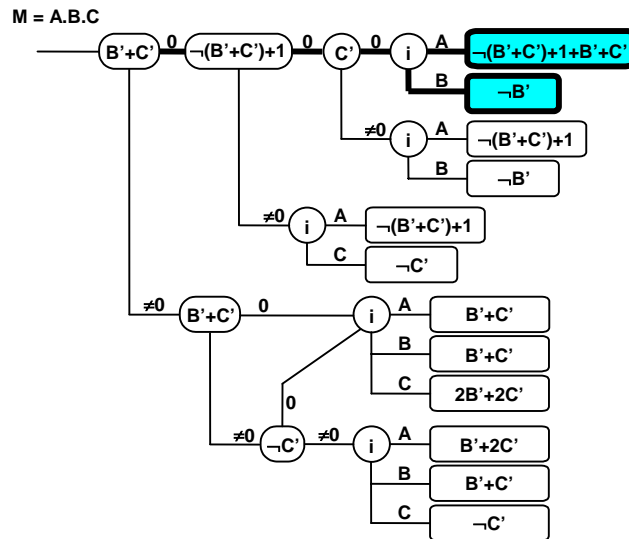
Joonis 1-42. Joonisel 1-40 esitatud süsteemi vektorotsustusdiagramm

Vektorotsustusdiagrammide kasutamise abil on veelgi võimalik tõsta süsteemi simuleerimise efektiivsust võrreldes komponentide võrgu simuleerimisega. Kui komponentide simuleerimisel eraldi graafe kasutades komponentmuutujate väärtuste arvutamine toimuks eraldi eri sammudena, siis vektorgraafis  $G_{q^i A^i B^i C^i}$  toimub süsteemi uue oleku arvutamine ühiselt kõigi funktsioonimuutujate  $q, A, B$  ja  $C$  jaoks korraga.

Joonisel 1-42 on halli värviga ja jämedate joontega illustreeritud võrreldav töömaht testvektori  $q^i=1, x_A = 1$  simuleerimisel. Vektorgraafil käiakse selle vektori simuleerimisel läbi 3 graafi sisetippu  $q^i, x_A$ , ja  $i$ . Aadressstipul  $i$  on vaid kaks kaart, kuna muutujate  $A$  ja  $C$  väärtused antud testvektori korral ei muutu ja neid pole vaja arvutada. Sama testvektori simuleerimisel eraldi muutujate graafides (joonis 1-41) tuleks läbi käia 7 sisetippu 4-s graafis (4 korda  $q^i$  ja 3 korda  $x_A$ ), seega üle 2 korra rohkem.

*Vektorotsustusdiagrammide komprimeerimine ajas*

Digitaalsüsteemide simuleerimise kiiruse edasiseks tõstmiseks on võimalik teostada otsustusdiagrammides esitatud funktsioonide superponeerimist ka ajas.



*Joonis 1-43. Ajas superponeeritud vektorotsustusdiagramm*

Joonisel 1-43 on toodud digitaalsüsteemi esitus vektorotsustusdiagrammina  $G_{A,B,C}$ , kus olekumuutujad ja ka olekugraaf  $G_q$  on elimineeritud ning kogu arvutusprotsess on taandatud üheleainsale ajamomendile, millele vastab lõpphetk, mil mikroprogramm saab täidetud. Niisuguse vektorgraafi  $G_{A,B,C}$  saamiseks tuleks simuleerida takthaaval graafi  $G_{q,A,B,C}$  üle kõigi olekumuutuja väärtuste  $q' = 0,1,2,3,4$  ning fikseerida igal hetkel (vastava  $q'$  väärtuse juures) registermuutujate  $A$ ,  $B$  ja  $C$  sümbolväärtused (avaldised nende muutujate arvutamiseks). Loogikatingimuste  $x_A$  ja  $x_C$  asemele graafi sisetippudes tuleb asetada nende muutujate arvutamiseks antud hetkel (antud  $q'$  väärtusel) kehtivad avaldised. Niisugust graafi manipuleerimist, mille tulemusena tekib uus graaf, nimetatakse *sümbolsimuleerimiseks*.

Sümbolsimuleerimise puhul ei asendata muutujaid nende konkreetsete väärtustega antud ajahetkel, vaid avaldistega, mille järgi neid tuleks hetkel arvutada. Seega ei teisendu otsustusdiagrammi sümbolsimuleerimise tulemusena esialgne graaf väärtuseks, vaid uuesti graafiks (puuks).

Nii näiteks pärast vektormuutuja  $M = A.B.C.q$  simuleerimist graafis  $G_{q,A,B,C}$  (joonis 1-42) tingimisel  $q' = 0$ , saame tulemuseks  $A = B' + C'$  ja  $q = 1$ . Vastav *harugraaf*, alates kaarest  $q' = 0$ , elimineeritakse. Edasi jätkatakse graafi simuleerimist väärtusel  $q' = 1$ . Loogikatingimus  $x_A$  (antud simuleerimise hetkel  $x_A$  on samane tingimusega  $A = B' + C' \neq 0$  ja  $x_A = 0$  on samane tingimusega  $A = B' + C' = 0$ ) asendatakse graafis loogikatingimusega  $B' + C' \neq 0$  ning jätkatakse graafi simuleerimist mõlema juhu jaoks  $B' + C' = 0$  ja  $B' + C' \neq 0$ .

Jätkates nüüd sümbolsimuleerimist graafis  $G_{q,A,B,C}$  mööda teed  $q' = 1$ ,  $x_A = 0$  (ehk  $B' + C' = 0$ ), arvutame:  $A = \neg A' + 1 = \neg(B' + C') + 1$  ja  $q' = 4$ . Vastav harugraaf graafis  $G_{q,A,B,C}$  alates kaarest  $x_A = 0$  elimineeritakse. Seejärel valime samas graafis uue haru  $q' = 4$  ning siirdume tippu  $x_A$ . Loogikatingimus  $x_A$  (antud juhul  $x_A$  on samane tingimusega  $A = \neg(B' + C') + 1 \neq 0$ ) asendatakse tingimusega  $\neg(B' + C') + 1 \neq 0$  ning jätkatakse graafi simuleerimist mõlema juhu jaoks  $\neg(B' + C') + 1 = 0$  ja  $\neg(B' + C') + 1 \neq 0$ .

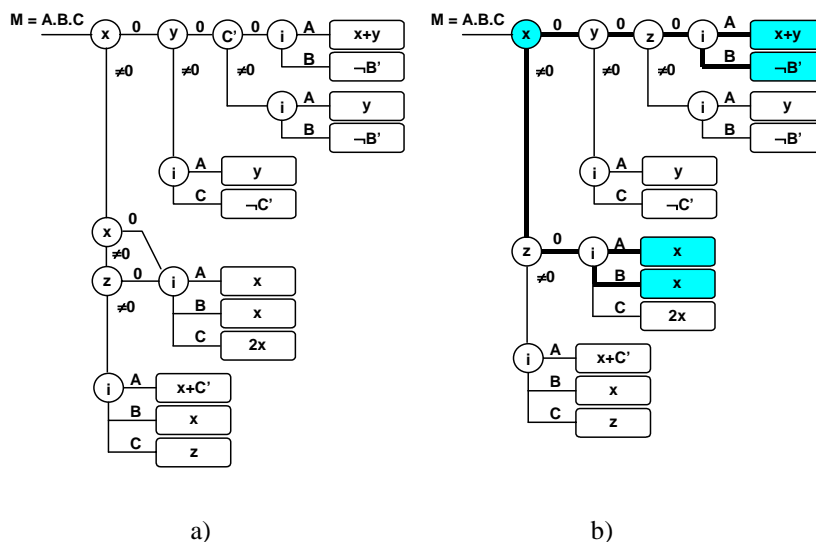
Graafi  $G_{q,A,B,C}$  (joonis 1-42) sümbolsimuleerimise tulemuseks on graaf  $G_{A,B,C}$  (joonis 1-43), kus olekutipp  $q'$  on elimineeritud ja sisetippudes on Boole'i muutujad asendatud nende väärtuste arvutamiseks vajalike funktsionaalsete avaldistega.

Joonisel 1-44a on esitatud sama graaf, kus on kasutusele võetud abimuutujad vahepealsete tulemuste salvestamiseks



$$\begin{aligned}
 x &= B' + C' \\
 y &= \neg(B' + C') + 1 \\
 z &= \neg C'
 \end{aligned}
 \tag{1-35}$$

Abimuutujate kasutamine võimaldab vältida korduvaid arvutusi. Joonisel 1-44b on veel kord esitatud sama otsustusdiagramm optimeeritud kujul, kus on elimineeritud muutuja  $x$ . Kerge veenduda, et mõlemad vektordiagrammid joonisel 1-44 on funktsionaalselt ekvivalentsed.



**Joonis 1-44.** Ajas superponeeritud abimuutujatega vektorotsustusdiagramm

Võrreldes joonistel 1-43 ja 1-44a esitatud kompaktsed vektorotsustusdiagramme mikroprogrammiga joonisel 1-40, võib esimesel pilgul öelda, et erinevus peaaegu puudub. Nii on see tõepoolest traditsioonilist simuleerimist silmas pidades. Iga tee otsustusdiagrammides joonistel 1-43 ja 1-44 kujutab tegelikult ka mingit ühte teed läbi mikroprogrammi joonisel 1-40. Näiteks rasvaselt eristatud teele mikroprogrammis joonisel 1-40, mille tulemusena arvutatakse uued väärtused registrites  $A$  ja  $B$ , vastab joonisel 1-43 rasvaselt ära märgistatud harugraaf (ning, vastavalt, joonisel 1-44b rasvaselt märgistatud teed läbi sisetippude  $x$ ,  $y$ ,  $z$ ,  $i$  terminaaltippudesse  $x + y$  ja  $\neg B'$ ).

Testide genereerimist silmas pidades on erinevus mikroprogrammi (joonis 1-40) ja otsustusdiagrammide vahel aga suur. Kui mikroprogramm modelleerib üksnes järjestikulist andmete arvutamise protsessi, siis otsustusdiagrammid esitavad selle kõrval veel ilmutatud kujul tulemusepõhjuse vahekordi, mis võimaldab kergesti läbi viia diagnostilist analüüsi. Näiteks avastades vigase tulemuse  $A^*$  registris  $A$  pärast mikroprogrammi läbimist jämedalt märgistatud teed mööda joonisel 1-40, oleks mikroprogrammi abil suhteliselt tülikas kindlaks teha võimalikku rikke mõjumispunkti (asukohta).

Joonisel 1-44b esitatud otsustusdiagrammis aga saaksime diagnostilist analüüsi suhteliselt lihtsalt korraldada ning vea asukoha lokaliseerida järgmisel viisil.

Vaadeldud operatsioonile (rasvane tee joonisel 1-40), mille tulemusena avastati viga  $A^*$ , vastab joonisel 1-44b aktiveeritud tee läbi tippude  $x$ ,  $y$ ,  $z$ ,  $i$  ja  $x + y$ , kus muutujate  $x$ ,  $y$ ,  $z$  väärtused on arvatud valemite (1-35) abil. Olgu operatsiooni tulemuseks saadud väärtus  $A = A^* \neq x + y$ . Läbitud tipud  $x$ ,  $y$ ,  $z$ ,  $x + y$  viitavad võimalikele vea põhjustele, mis tuleks üksteise järel läbi analüüsida.

Kui vea põhjuseks oleks sisetipp  $x$ , oleks graafis aktiveeritud tee sellest tipust alla ( $x \neq 0$ ) läbi tippude  $z$  ( $z = 0$ ),  $i$  ( $i = A$ , terminaltippu  $x$ ). Kui viga tõesti oleks sisetipus  $x$ , peaks operatsiooni tulemusena saadud vigase väärtuse jaoks kehtima ka tingimus  $A^* = x$ , kuna vea tõttu jõuti ju terminaaltipu  $x + y$  asemel terminaaltippu  $x$ . Kui  $A^* \neq x$ , siis ei saa viga olla graafi sisetipus  $x$  ning tuleks analüüsida teisi läbitud tippe. Sarnaselt eelneva arutlusega leiame, et vea põhjuseks võiksid olla graafis läbitud sisetipud  $y$  või  $z$ , juhul, kui kehtiks  $A^* = y$ . Neid veapõhjusi saaksime eristada, kui analüüsiksime täiendavalt registri sisu  $C$ . Kui selle sisu oleks inverteeritud  $C^* = \neg C$ , siis oleks vea põhjuseks tipp  $y$ , sest siis aktiveeruks graafis tee tipust  $y$  alla ( $y \neq 0$ ) läbi tipu  $i$  ( $i = C$ ), terminaaltippu  $\neg C$ . Vastasel juhul, kui registri  $C$  väärtus on jäänud muutumata  $C^* = C$ , peab viga olema tipus  $z$ . Juhul kui  $A^* \neq y$ , võiks vea põhjuseks olla adresseerimistipp  $i$ , seda aga ainult juhul, kui  $A^* = \neg B$ . Kui ka see tingimus pole täidetud  $A^* \neq \neg B$ , siis saab vea põhjuseks olla vaid terminaltipp  $x + y$  ise.

Viimasel juhul on viga süsteemi operatsioonautomaadis (andmeosas), eelmistel juhtudel oleks viga aga juhtautomaadis.

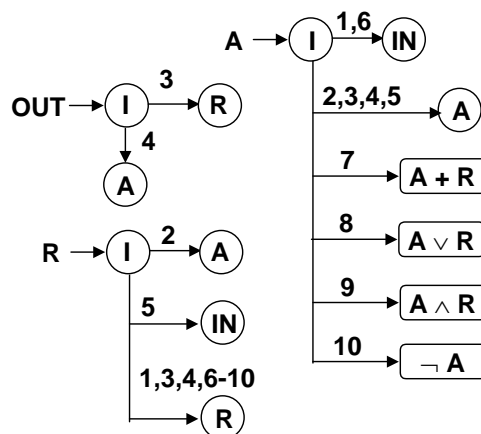
## 1.2.7. Otsustusdiagrammid mikroprotsessorite diagnostiliseks modelleerimiseks

Analoogiliselt registersiirete tasandil esitatud digitaalsüsteemidele on ka käsusüsteemidega kirjeldatud mikroprotsessorite diagnostiliseks modelleerimiseks võimalik kasutada otsustusdiagramme.

I <sub>1</sub> : MVI A,D	A = IN	I <sub>6</sub> : MOV A,M	A = IN
I <sub>2</sub> : MOV R,A	R = A	I <sub>7</sub> : ADD R	A = A + R
I <sub>3</sub> : MOV M,R	OUT = R	I <sub>8</sub> : ORA R	A = A ∨ R
I <sub>4</sub> : MOV M,A	OUT = A	I <sub>9</sub> : ANA R	A = A ∧ R
I <sub>5</sub> : MOV R,M	R = IN	I <sub>10</sub> : CMA A,D	A = ¬ A

Joonis 1-45. Hüpooteetilise mikroprotsessori käsusüsteem

Otsustusdiagrammide sünteesi aluseks tuleks sel juhul võtta mikroprotsessorite käskude kirjeldused registersiirete tasandil. Käskude eristamiseks tuleks sisse viia uus registertaseme *käsumuutuja*, mille väärtusteks oleksid mikroprotsessori käsu numbrid või koodid.



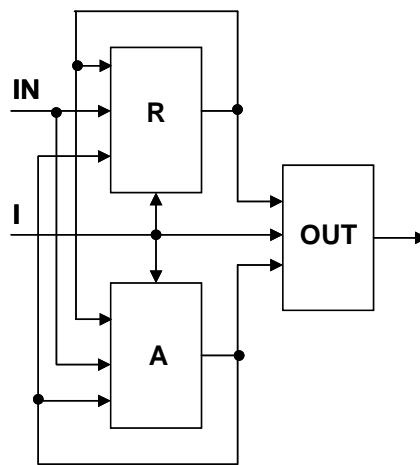
Joonis 1-46. Mikroprotsessori käsusüsteemi otsustusdiagrammid

Näide 1-20

Joonisel 1-45 on hüpoteetilise mikroprotsessori käsusüsteem ja joonisel 1-46 sellele vastav otsustusdiagrammide võrk. Käsusüsteem sarnaneb tabelites 1-2 või 1-3 toodud töörežiimide nimistuga ja otsustusdiagrammide süntees toimub sarnaselt graafi  $G_{AC}$  sünteesile joonisel 1-24.

Joonisel 1-46 toodud graaf  $G_{OUT}$  vastab mikroprotsessori väljundkäitumisele ja graafid  $G_A$  ning  $G_R$  kirjeldavad vastavalt registre  $A$  ja  $R$  sisendloogika funktsioone. Mudeli andmemuutujad on vektoriaalsed: sisendmuutuja  $IN$ , väljundmuutuja  $OUT$  ja registermuutuja  $R$ . Juhtmuutuja (käsumuutuja)  $I$ , mille väärtustele vastavad käsukoodid  $I_k, k = 1, 2, \dots, 10$ , on täisarvuline.

Otsustusdiagrammide mudeli erinevuseks võrreldes käsusüsteemiga on see, et põhitähelepanu pole pööratud sellele, mis toimub ühe või teise konkreetse käsu ajal, vaid sellele, kuidas käituvad eri käskude puhul mikroprotsessori käsusüsteemi sise- ja väljundmuutujad.



Joonis 1-47. Mikroprotsessori struktuurskeem

On lihtne märgata, et graafid OD-mudelis on muutujate kaudu omavahel seotud. Näiteks graafimuutujad  $R$  ja  $A$  on argumentideks ehk muutujateks väljundgraafis  $G_{OUT}$ . Seega sisuliselt kujutab graafide süsteem

joonisel 1-46 endast võrku, mis on esitatav mikroprotsessori käitumist kujutava struktuurskeemina joonisel 1-47.

Teiste sõnadega, minnes traditsiooniliselt mikroprotsessori *protseduurset mudelilt* üle otsustusdiagrammide mudelile, saame uue struktuurse esituse, mis võimaldab vahetumalt formuleerida testide sünteesi ja analüüsi ülesandeid, kasutades traditsioonilisi struktuurseid testide sünteesi ja analüüsi meetodeid. Digitaalsüsteemide diagnostika otsesteks objektideks on rikked, aga rikkeid saab vahetumalt modelleerida *struktuursetel* mudelitel, võrreldes protseduursete või *funktsionaalsete* (musta kasti tüüpi) esitustega.

Detailsemaks mikroprotsessori kirjeldamiseks *käsusüsteemi tasandil* oleksid järgmised võimalused:

1) mikroprotsessori *käsuformaadi* võib dekomponeerida väljadeks, kus igal väljal on oma spetsiifiline funktsioon: *operatsioonikood*, esimese registri aadress, teise registri aadress jne. Igale väljale võib vastavusse seada eri muutuja, mida võib vaadelda käsumuutuja kui vektormuutuja ühe komponendina;

2) käsutäitmise protsessi võib vaadelda *käsu tsükli* ja niisuguste tsükli esitamiseks võib kasutusele võtta analoogiliselt automaatide olekumuutujaga *tsüklimuutuja*. Tsüklimuutuja väärtustele vastaksid käsutäitmise erinevad tsüklid: käsu väljalugemine, esimese operandi lugemine, teise operandi lugemine, tulemuse kirjutamine mälli jne. Juhul kui tsükli arv on konstantne iga konkreetse käsu jaoks ja üleminek olekust olekusse ei sõltu andmetest, võib olekumuutuja graafist loobuda ning käsitleda olekumuutujat nagu aadressmuutujat vektorgraafides, viies aadressi dimensiooni kõrval mudelisse sisse ka aja dimensiooni.

### 1.3. Kokkuvõtteks

Peatükis vaadeldi digitaalsüsteemide diagnostilise modelleerimise kahte matemaatilist aparaati: Boole diferentsiaalalgebrat ja graafiteooriat ehk otsustusdiagramme.

Boole'i diferentsiaalaparaadi tähendus praktiliste diagnostikaülesannete keerukust silmas pidades on rohkem tunnetuslik. Konkreetselt lahendada saab selle aparaadi abil vaid lihtsaid testide genereerimise, rikete simuleerimise ja diagnostika ülesandeid mittekeerukate loogikaskeemide jaoks. Mikroprotsessori puhul pole selle aparaadiga kahjuks midagi teha.

Küll on aga Boole'i diferentsiaalaparaat oma kompaktsuse ja korrektsuse poolest sobiv testide sünteesi ja analüüsi matemaatilise probleemi formuleerimiseks ning probleemidevaheliste seoste lihtsaks kirjeldamiseks.

Suurepäraseks näiteks oleks siin testide genereerimise ja rikete diagnostika probleemide ühtne formuleerimine diagnostika üldvõrrandi abil (alapunktis 1.1.8), mis võimaldab mõlemat probleemi vaadelda teineteise pöördprobleemidena.

Teiseks iseloomulikuks näiteks on olemusliku selgituse andmine tavapärasele insenerlikule lihtsustusele, kus piisavaks loetakse testide genereerimist üksikutele riketele, ignoreerides nende kordse esinemise võimalust. Selgituse sellele probleemile saame alapunktis 1.1.7, kus näeme, kuidas Boole'i osatuletis, mille abil formuleeritakse testide genereerimise ülesannet, tuleneb erijuhuna Boole'i täisdiferentsiaalset, mis modelleerib skeemi kõikide võimalike rikete ja nende kombinatsioonide korral.

Samas on Boole'i diferentsiaalaparaadil ka praktiline tähendus füüsikaliste defektide loogilisel modelleerimisel, mida käsitletakse järgmises peatükis.

Niisiis võimaldab Boole'i diferentsiaalaparaat formuleerida korrektselt probleemi. Kui probleem on formuleeritud, saab lahendamiseks otsida juba sobivamaid meetodeid ja mudeleid. Üheks niisuguseks mudeliks valiti käesolevas raamatus otsustusdiagrammide mudel.

Kui enamus digitaalsüsteemide diagnostika probleeme loogikaskeemide tasandil on peaaegu täiuslikult lahendatud Boole'i algebrat ja Boole'i diferentsiaalalgebrat kasutades, siis kõrgemate tasandite jaoks on vastav matemaatiline aparaat kaua puudunud. Samas on keerukate miljonitest loogikaelementidest koosnevate süsteemide diagnostikaprobleemide lahendamine ainuvõimalik kõrgematel tasanditel "töötavate" meetodite kaasamisega. Niisugusteks tasanditeks oleksid protseduurse registertaseme kirjeldused (näiteks VHDL keeles), protsessorite käsusüsteemid jne.

Kui Boole'i algebral põhinevaid diagnostikameetodeid ei ole võimalik otseselt üldistada kõrgematele tasanditele, siis graafiteoorial põhinevat topoloogilist käsitlust kasutades osutub üldistamine võimalikuks.

Üldjoontes seisneb niisuguse üldistamise võimalus järgnevas. Kõigepealt „tõlgime” Boole'i algebra meetodid topoloogilisteks meetoditeks, kasutades Boole'i algebra ühte tööriista binaarseid otsustusdiagramme. Seejärel eemaldame otsustusdiagrammidelt binaarsuse kitsenduse ja rakendame juba olemasolevaid “ümbertõlgitud” topoloogilisi meetodeid neid küll pisut modifitseerides juba kõrgematel tasanditel kirjeldatud süsteemide puhul.

Peatükis vaadeldud meetodeid kasutame edaspidi konkreetsete insenerlike diagnostikaprobleemide käsitlemisel. Järgmises peatükis vaadeldakse diagnostika alusprobleemi – rikete modelleerimist. Edaspidi aga võtame käsile otsesed diagnostikaprobleemid, milleks on: testide genereerimine, rikete simuleerimine ja rikete diagnostika.

## 2. Rikete modelleerimine digitaalsüsteemides

Füüsikaliste *defektide* põhjusi on palju ja võimalike defektide arv on suur. Pole mõeldav defekte üksikhaaval käsitleda ei testide genereerimisel ega ka defektide diagnoosil. Samal ajal aga tuleks *testimise kvaliteeti* mõõta avastatavate defektide kattega ehk protsendiga kõigi võimalike defektide suhtes. Niisugune kvaliteedi mõõtmine on praktiliselt võimatu. Seetõttu tuleks defekte modelleerida ja testimise kvaliteeti mõõta füüsikalise tasandiga või transistorskeemidega võrreldes kõrgematel digitaalsüsteemide abstraktsetel esitustasanditel.

*Rikete modelleerimise* eesmärk ongi asendada praktiliselt lõpmatu võimalike defektide hulk samaväärse lõpliku rikete hulgaga. *Rikete mudelid* kasutatakse nii testide genereerimisel kui ka füüsikaliste defektide lokaliseerimiseks ning neid tuleks vaadelda kui silda, mis ületab lõhet füüsikalise reaalsuse ja abstraktse matemaatika vahel.

Kõige sagedamini on kasutatud ja kasutatakse *konstantse rikke mudelit*, mille järgi digitaalskeeme vaadeldakse kui ventiilide võrku ja võimalikeks rikeks on loogikakonstandid 1 või 0 ventiilide sisendites ja väljundites. Mitme selle rikkemudeli omaduste ärakasutamine aitab vähendada otsest käsitlemist nõudvate konstantsete rikete hulka.

Samal ajal on selgeks saanud, et konstantsete loogikarikete mudel on siiski piiratud. Ühest küljest kasvab loogikataseme käsitlemist vajavate rikete hulk süsteemide keerukuse kasvades liiga kiiresti. Teisalt ei kata aga see rikkemudel piisavalt adekvaatselt *transistorskeemide* reaalseid defekte. Käesolevas peatükis vaadeldakse, kuidas käsitleda rikkeid ja tagada kõrge testimise kvaliteet üha keerukamaks muutuvates digitaalsüsteemides, kuidas on võimalik kasutada Boole'i diferentsiaalaparaati füüsikaliste defektide kujutamiseks loogikatasandile ja milline peaks olema rikete mudel, mis



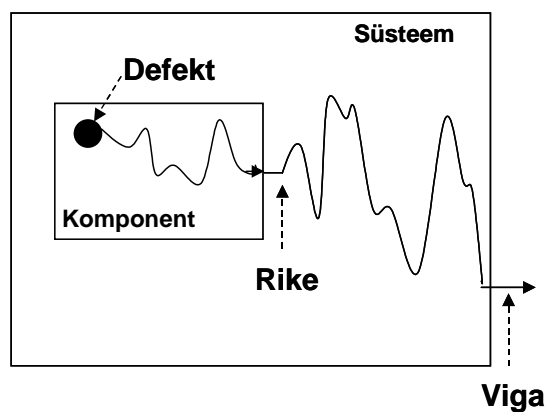
poleks väga keerukas, aga samas ometi esindaks täpsemalt reaalsel füüsikalist defektide maailma.

## 2.1. Rikete klassifikatsioon

Vaatleme, mille poolest erinevad üksteisest niisugused mõisted nagu defektid, rikked ja vead. Uurime põhilisi rikete tüüpe, nende mudeleid ning rikete põhiomadusi.

### 2.1.1. Defektid, rikked, vead

Terminoloogia korrastamiseks püüame kõigepealt defineerida põhimõisted. Termineid *defekt*, *rike* ja *viga* kasutatakse nii teaduslikus kirjanduses kui ka elus sageli erinevates ja kattuvates tähendustes. Aga need mõisted ei ole sünonüümid. Käesolevas raamatus kasutame definitsioone, mis on esitatud defektide ja nende testimisele pühendatud monograafias [Ben 98].



*Joonis 2-1. Defekt, rike ja viga digitaalsüsteemis*

*Defektiks* elektroonikasüsteemis nimetame mittekavandatud erinevust disaini ja selle implementatsiooni vahel riistvaras. Defekt kutsub esile

erinevuse elektroonskeemis võrreldes selle spetsifikatsiooniga. Defekt tekib kas seadme valmistamisel või tema kasutamise ajal.

*Rikkeks* nimetame defekti kujutamist abstraktsel funktsionaalsel tasandil. Rike on defekti matemaatiline mudel. Joonisel 2-1 on kujutatud defekt, rike ja viga omavahelises suhtes. Kusagil transistoriskeemi tasandil tekib *füüsikaline defekt*, mis avaldub selles, et defektse komponendi väljundsignaal võtab väära püsiva väärtuse, mida võib tõlgendada loogikatasandil kui konstantset väära loogikaväärtust ehk *riket*. Seda väärtust kasutatakse skeemi loogikamudeliga manipuleerides, näiteks testi genereerides, mille ülesandeks on antud defekti avastamine ehk *rikke levitamine* väljundisse, kus signaalid oleksid jälgitavad. Väljundisse levitatud väär signaal on tõlgentatav avastatud *veana*.

*Veaks* nimetamegi väärat väljundsignaali, mida põhjustab defektne süsteem. Vea põhjuseks on mingi defekt.

### 2.1.2. Rikete mudelite üldised omadused

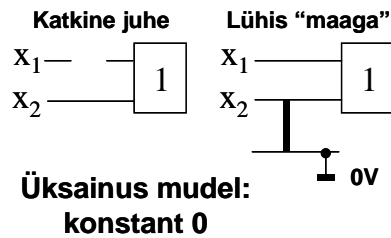
Füüsikaliste *defektide modelleerimiseks* abstraktsete rikete mudelite abil on terve rida olulisi motivatsioone:

- väheneb defektide käsitlemise keerukus: mitut erinevat füüsikalist defekti esindab üksainus rikkemudel (joonis 2-2);
- ühtainust matemaatilist rikkemudelit võib kasutada erinevate tehnoloogiate puhul, kus defektide füüsikalised tähendused võivad erineda;
- rikete loogikamudeleid võib edukalt kasutada juhtumitel, kui rikete füüsikaline tähendus ja efekt pole päris selge;
- rikete mudelid annavad võimaluse töötada füüsikalisest tasandist kõrgematel abstraktsetel tasanditel.

#### *Näide 2.1*

Joonisel 2-2 on näidatud kaks defekti: katkine juhe ja lühis maaga. Mõlemad defektid on loogiliselt modelleeritavad ühe ja sama rikkena – konstantne 0.

### Kaks defekti:



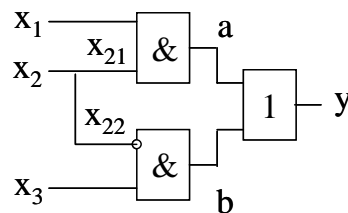
Joonis 2-2. Kahte defekti modelleerib üksainus rike

Rikete mudelid võivad olla *otsesed* ja *kaudsed*.

*Otsese rikkemudeli* puhul võib rikkeid loendada. Näiteks konstantne loogikarike on otsene, sest ta on loendatav. Kui skeemis on elementide vahel  $n$  ühendust ja igal ühendusel võib olla kaks riket – konstantne 0 ja konstantne 1, siis on kõigi rikete arv skeemis  $2n$ . *Kaudsed rikked* ei ole loendatavad, need defineeritakse harilikult mingi iseloomuliku omadusena.

Rikete mudelid võivad olla *struktuursed* ja *funktsionaalsed*.

*Struktuursed rikked* on seotud konkreetse skeemi struktuuriga, need modifitseerivad näiteks komponentidevahelisi ühendusi. *Funktsionaalsed rikked* aga on seotud komponentide funktsionaalsete mudelitega, need modifitseerivad näiteks komponentide funktsioone.



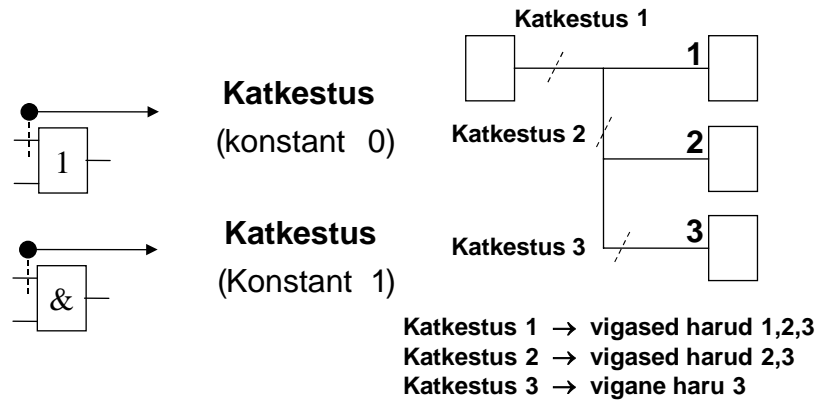
Joonis 2-3. Kombinatsioonskeem

Näide 2-2

Vaatleme skeemi joonisel 2-3. Struktuurseteks riketeks selles skeemis võivad näiteks olla katkine juhe  $a$ , lühis juhtmete  $x_1$  ja  $x_2$  vahel või vigane komponent. Funktsionaalse rikke näiteks on aga funktsiooni muutus: õige funktsiooni asemel realiseerub mingi teine funktsioon

$$y = x_1x_2 \vee \overline{x_2x_3} \Rightarrow y = \overline{x_2x_3}.$$

Struktuursete rikete puhul eeldatakse vahel ka, et skeemi komponendid on korras ja rikked võivad olla ainult komponentidevahelistes ühendustes.



**Joonis 2-4.** Mittevastavus defektide ja konstantse rikkemudeli vahel

Mitte alati pole ventiilskeemides võimalik luua adekvaatset vahet vahel reaalsete struktuursete rikete ja konstantse rikke mudeli vahel. Näiteks võib katkine juhe erinevate loogikaelementide puhul tähendada nii konstanti 0 kui ka konstanti 1 (joonis 2-4). Kui juhe hargneb, siis, olgugi et on tegemist ühe ja sama muutujaga, tuleb ometigi katkestusi erinevates kohtades modelleerida erinevalt (joonis 2-4).

### 2.1.3. Konstantsed rikked

*Konstantseid* rikkeid (*SAF mudel – stuck-at fault*) modelleeritakse, fikseerides konstantse väärtuse signaalijuhtmele, millele korras skeemis vastaks Boole'i muutuja. Signaalijuhtmed on loogikaskeemides ventiilide või trigerite sisenditeks või väljunditeks. Eeldatakse, et konstantsed rikked mõjutavad vaid ventiilidevahelisi ühendusi, aga mitte ventiile endid. Iga ühendusjuhe võib omada kahte tüüpi rikkeid: *konstant 0 (stuck-at-0, SAF0)* või *konstant 1 (stuck-at-1, SAF1)*. Konstantse rikke näiteid on joonisel 2-4.

*Üksiku konstantse rikke* mudel on kõige levinum rikkemudel digitaalskeemide ja süsteemide diagnostikas: igal aja hetkel eeldatakse vaid ühe rikke olemasolu, kusjuures see rike on püsiv (*permanentne*)<sup>3</sup>. Üksiku konstantse rikke mudelil on järgmised omadused:

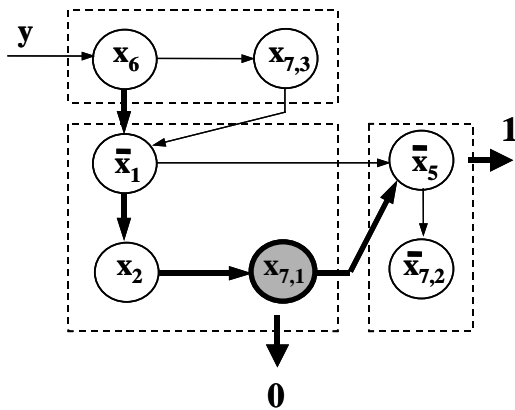
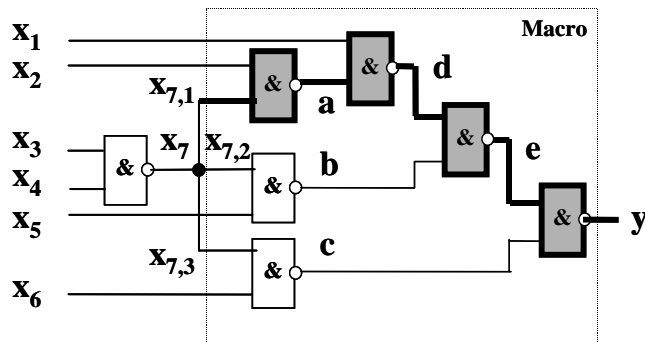
- ainult üks signaalijuhe võib olla vigane;
- signaalijuhtmel on püsivalt konstantne signaal 1 või 0;
- konstantne signaal võib olla ventiili sisendis või väljundis.

Konstantse rikke mudel sai tööstuslikuks standardiks 1959. aastal. On ennustatud konstantse rikke mudelist loobumist, kuid paljudel põhjustel ja mitme konstantse rikke hea omaduse tõttu on ta endiselt laialt kasutusel. Niisugusteks omadusteks on

- mudeli lihtsus;
- selle käsitlemise hõlpsus;
- mudeli loogiline käitumine – simuleerimine on vahetu ja deterministlik;
- moodsustavus: avastamise või mitteavastamise kindlakstegemine on lihtne;
- kohandatavus: mudelit saab kergesti rakendada suvalistel süsteemi abstraktse esituse tasanditel: ventiilid, transistorid, süsteemid, registersiirete tasand jne.

---

<sup>3</sup> Eksisteerib ka *ajutise (transient) rikke* mudel, mida käesolevas raamatus ei vaadelda



Joonis 2-5. Testi genereerimine konstantsele rikkele

Näide 2-3

Vaatleme testi genereerimist konstantsele rikkele  $x_{71} \equiv 0$  skeemis joonisel 2-5. Kasutades *Boole'i osatuletisi*, tuleks lahendada diferentsiaalvõrrand

$$\frac{\partial y}{\partial x_{71}} = \frac{\partial (x_6 x_{73} \vee (\bar{x}_1 \vee x_2 x_{71})(\bar{x}_5 \vee \bar{x}_{72}))}{\partial x_{71}} = 1$$

lisatingimusel

$$x_7 = 1$$

Kasutades SSBOD-mudelit joonisel 2-5 tuleks

- aktiveerida tee tipuni  $x_{71}$ , omistades teel kohatud muutujatele väärtused  $x_6 = 0, x_1 = 1, x_2 = 1$ ,
- fikseerida muutuja  $x_7$  väärtus  $x_7 = 1$ ,
- aktiveerida tipust  $x_{71}$  tee terminali 1, fikseerides  $x_5 = 0$ .

Väärtuse  $x_7 = 1$  tagamiseks sisenditel tuleks lahendada võrrand  $x_3 x_4 = 1$ .

Vaatamata konstantse rikke mudeli headele omadustele, ei saa nii mitmeidki reaalseid füüsikalisi defekte tema abil ometi modelleerida põhiliselt just *CMOS-tehnoloogia* puhul. See on konstantse rikke mudeli puudus. Mõnikord püütakse mudeli "võimsust" suurendada, lubades *kordsete rikete* olemasolu. Kuid eriti see reaalsete defektide katet siiski ei suurenda, pealegi on kordse rikke mudel keerukas: rikete arv  $n$  signaalijuhtmega skeemis on  $3^n$ . Ka testide genereerimise ja rikete simuleerimise algoritmid muutuvad kordsete rikete mudeli puhul keerukaks.

#### 2.1.4. Lühised, katkestused ja viiterikked

Tulenevalt konstantse rikke puudustest on võetud kasutusele terve rida täiendavaid rikete mudeleid nagu lühised, katkestused, viiterikked jne. Teatud rikkeid tuleks modelleerida ainult transistoride tasemel. Rikete modelleerimise keerukuse vähendamiseks on püütud välja töötada ka palju kõrgtaseme rikkemudeleid.

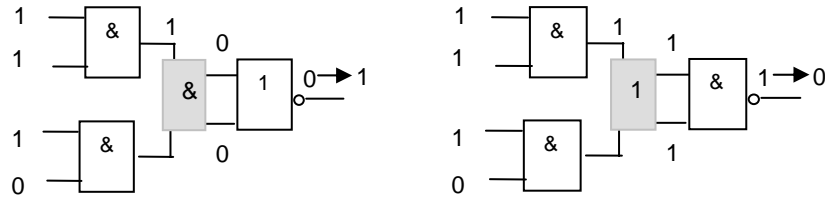
##### *Lühised*

*Lühised (bridges, shorts)* modelleerivad kõiki neid füüsikalisi defekte, mis põhjustavad mittekavatsetud elektrilisi kontakte kahe või rohkema signaalijuhtme vahel.

Lühiseid võib iseloomustada lineaarsete või mittelineaarsete omadustega, mis on seotud lühistakistuse eri väärtustega vahemikus 0 kuni

1 M $\Omega$ . Teatud kriitilistest takistuse vahemikest väljaspool võib reaalne lühis ka mitte enam põhjustada rikke efekti.

Lühised võib jämedalt jagada kahte gruppi: *ventiilidevahelised* lühised (*inter-gate shorts*), *ventiilisised* ehk transistoride lühised (*intra-gate shorts*).



**Joonis 2-6.** *Ventiilidevahelised kombinatoorsed lühised*

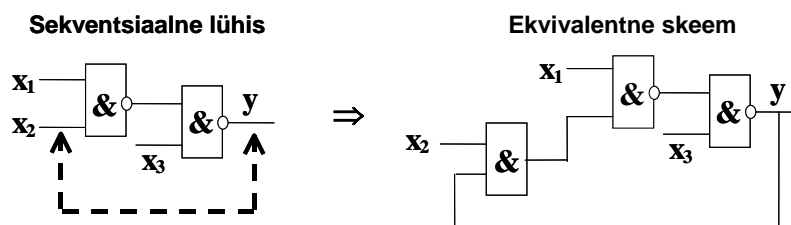
Ventiilidevahelisi lühiseid võib omakorda klassifitseerida järgmiselt:

- *JAA-tüüpi* lühised (*wired-AND short*) või *0-domineerimisega* lühised (vt näidet joonisel 2-6a);
- *VÕI-tüüpi* lühised (*wired-OR short*) või *1-domineerimisega* lühised (vt näidet joonisel 2-6b);
- Mittemääratud lühise tüüp, kus lühis sõltub tehnoloogiast, mida kasutatakse.

Lühised võivad olla *kombinatoorsed* ja *sekventsiaalsed*.

*Sekventsiaalsed lühised* põhjustavad kombinatsioonskeemides sekventsiaalset käitumist, siis kui lühis tekitab elektrilise tagasiside (joonis 2-7).



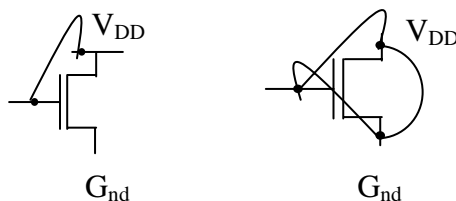


*Joonis 2-7. Ventilidevaheline sekventsiaalne lühis*

*Kombinatorsete lühiste* testimise protsent konstantse rikke mudelit kasutades on harilikult kõrge. Sekventsiaalsete lühiste testimine on aga keerukas, sest üldiselt eeldavad testide genereerimise meetodid, et olekute arv skeemis ei kasva (sekventsiaalne lühis põhjustab aga täiendava oleku tekkimise skeemis) ning et kombinatsiooniskeemid rikke puhul jäävad ikka kombinatsiooniskeemideks (sekventsiaalse rikke tõttu muutub aga kombinatsiooniskeem sekventsiaalseks ehk järjestikскеemiks).

*Lühised transistorskeemides*

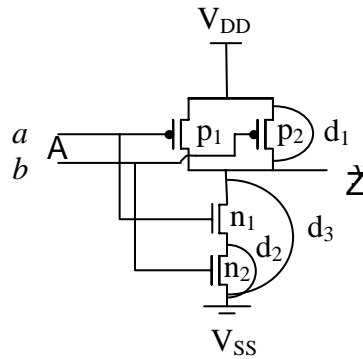
Transistorskeemides võib põhimõtteliselt eristada kuut tüüpi lühist kõikide transistori terminalide vahel: ventiil, *neel (drain)*, *läte (source)*, *põhimik (substrate)* ja lühised  $V_{DD}$  ja  $G_{nd}$  sõlmedega (joonis 2-8). Mainitud rikete tegelik avaldumine sõltub lühistakistuse suuruselt. Mõningaid ventiiliseseid transistortaseme rikkeid võib modelleerida ka konstantse rikke mudeliga.



*Joonis 2-8. Transistori lühised*

Näide 2-4

Joonisel 2-9 on illustreeritud CMOS-tehnoloogias realiseeritud JAA-EI ventiili transistorskeem, milles on kolm lühisriket  $d_1$ ,  $d_2$ ,  $d_3$ .



Joonis 2-9. Lühised transistortaseme CMOS JAA-EI ventiilis

Eeldades, et lühistakistus on oluliselt madalam kui avatud transistori sisetakistus, võib defekte  $d_1$  and  $d_3$  modelleerida loogikatasandi konstantse rikke mudeliga, aga defekti  $d_2$  mitte.

Defekt  $d_1$  põhjustab lühise toiteallika VDD ja väljundsõlme  $y$  vahel. Rakendades testvektorit  $(a,b) = 11$ , avanevad transistorid  $n_1$  ja  $n_2$ . Eeldades, et avatud transistoride sisetakistus on oluliselt kõrgem defekti  $d_1$  põhjustatud lühistakistusest, võib defekti lugeda ekvivalentseks konstant 1 rikkega väljundis  $y$ .

Samasuguse põhjenduse saab anda ka defekti  $d_3$  taandamisele konstantseks rikkeks  $d_3 \equiv 1$ .

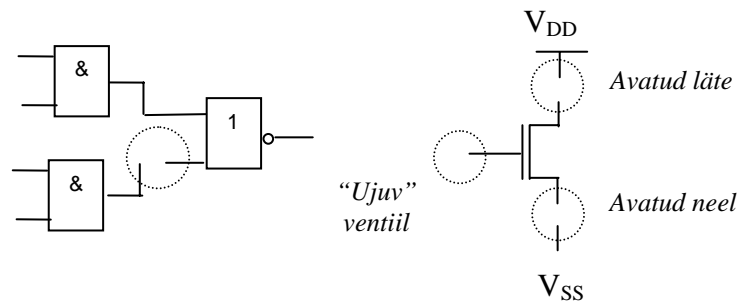
Defekti  $d_2$  võib avastada konstantse rikke testiga eeldusel, et lühisrikke takistus ja avatud transistori  $n_1$  takistus on oluliselt väiksemad kui avatud transistori  $p_2$  takistus. Sellise tingimuse täitmisel avastab defekti  $d_2$  testvektor  $(a,b) = 10$ , mis avab transistorid  $n_1$  ja  $p_2$ .

### *Katkestused*

*Katkestusteks (opens)* nimetatakse järgmist tüüpi defekte:

- kontakti puudumine ühendustes,
- katkenud juhtmed,
- praod metallitud kihtides jne.

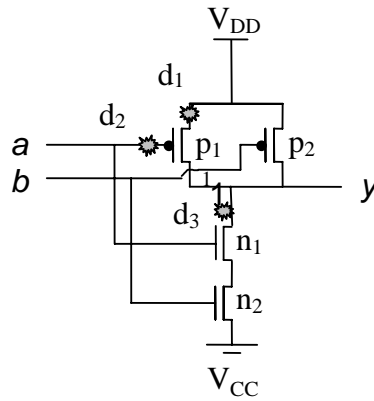
Katkestuste käitumine varieerub ja neid on tihti raske ette ennustada. Katkestuste avaldumine ei sõltu üksnes defekti suurusest või ulatusest, vaid ka temperatuurist, taktsagedusest, asukohast, tehnoloogilistest parameetritest jne. Katkestused võivad lokaliseeruda nii ventiilide tasandil kui ka transistoride tasandil (joonis 2-10).



**Joonis 2-10.** Katkestused ventiilskeemis ja transistoris

### *Näide 2-5*

Joonisel 2-11 on illustreeritud CMOS-tehnoloogias realiseeritud JAA-EI ventiili transistoriskeem, milles on kolm katkestusdefekti  $d_1$ ,  $d_2$ ,  $d_3$ , mis avaldavad mõju transistori  $p_1$  käitumisele.



**Joonis 2-11.** Katkestused CMOS JAA-EI transistorskeemis

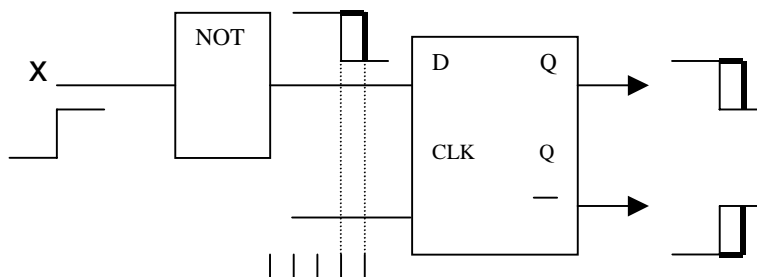
Konstantse rikke mudelit kasutades genereeritakse antud JAA-EI elemendile järgmine testvektorite (a,b) hulk:  $T_1 = (11)$ ,  $T_2 = (10)$ ,  $T_3 = (01)$ . Vaadeldavad katkestusdefektid aga ei ole avastatavad ühegi üksiku testvektoriga sellest hulgast. Vektor  $T_2$  viib väljundi väärtusele 1 transistori  $p_2$  abil ja vektor  $T_3$  viib väljundi väärtusele 1 transistori  $p_1$  abil. Defektide  $d_1$ ,  $d_2$ ,  $d_3$  olemasolul aga ei vii vektor  $T_3$  transistori olekusse 1. Nimetatud testvektorite kolmik avastaks need defektid, kui vektorid viidaks läbi järjestuses:  $T_2, T_1, T_3$ .

Üldjuhul on katkestusdefekti avastamiseks transistorskeemides vaja kahe testvektori jada  $T_1, T_2$ . Esimene vektor  $T_1$  initialsiseerib skeemi väljundi vajalikule loogikatasemele, teine vektor  $T_2$  üritab muuta skeemi olekut testitava transistori abil.

#### Viiterikked

Ajastus (*timing*) ehk ajalised viited (*delay*) on olulised parameetrid sisend-väljundsignaalide suhetes. Defektide omaduste uurimine on näidanud, et enamik CMOS-transistoride defekte põhjustavad just viiterikkeid (viiteid signaalide levikus), võrreldes näiteks teiste nn *katastroofaalse* (püsivate) defektidega. Näiteks *takistuslikud* lühisrikked (*resistive bridges*) lühistakistusega väljaspool kriitilise takistuse alasid põhjustavad just viiteid signaalide levikus ehk viiterikkeid. Viiterikked moodustavad defektide klassi, mis ei kuulu ei lühiste ega ka katkestuste

kategooriasse. Viiterike tähendab seda, et signaal hilineb väljundisse ja seetõttu konkreetsel hetkel (takti lõpus) oodatav väljundsignaal on vigane (joonis 2-12).



*Joonis 2-12. Viiterike*

Viiterikked võivad tekkida paljudest põhjustest, mis võiksid olla näiteks

- tehnoloogilise protsessi defektid;
- transistoride *lävipingete* nihked;
- suurenenud *parasiitmahtuvus*;
- signaalide ajastuse ebatäpsus.

Viiterikkeid võib seostada nii ventiilidega kui ka signaaliteedega, mistõttu viiterikete mudeleid klassifitseeritakse järgmiselt:

- *ventiili viiterikke* mudel (digitaalskeeme projekteeritakse, lähtudes elementide nominaalsetest viidetest; defekt ventiilis aga võib põhjustada nominaalsest palju suurema viite);
- *signaalitee viiterikke* mudel (mudel võtab kumulatiivselt arvesse kõigi ventiilide viiterikked antud signaaliteel)

## 2.2. Konstantsete rikete omadused

Üks olulisemaid probleeme testvektorite genereerimisel ja rikete simuleerimisel digitaalsüsteemides on skeemide keerukus – hiiglasuur rikete hulk, kus iga rikke kohta tuleb võtta mingi seisukoht (genereerida talle test või kindlaks teha, kas konkreetne olemasolev test suudab seda riket avastada või mitte).

Probleemist ülesaamiseks tuleks mingil moel püüda seda rikete hulka vähendada, määrata kindlaks antud rikke olulisus, suhe teistesse riketesse. Et niisugust analüüsi teha, peaks tundma rikete omadusi nagu testitavus, liiasus, ekvivalentsus teiste riketega, dominantsus teiste rikete suhtes jne. Tundes rikete niisuguseid omadusi, osutub võimalikuks kogu rikete hulgast selekteerida välja mingi olulisemate rikete alamhulk ja tegelda edaspidi vaid nende väljavalitud riketega, teades sel juhul näiteks, et kui testid on võimelised avastama kõik rikked sellest valikhulgast, avastavad nad ka kõik ülejäänud rikked.

### 2.2.1. Rikete liiasus

Olgu  $y = f(X)$  mingi kombinatsioonskeemi  $C$  loogikafunktsioon ja  $R(C) = \{r\}$  kõigi võimalike rikete hulk selles skeemis. Rikke  $r \in R(C)$  olemasolul transformeerub antud skeem  $C$  üheks uueks vigaseks kombinatsioonskeemiks  $C^r$  funktsiooniga  $y = f^r(X)$ . Igat sisendvektorit  $t_i = X_i^*$  võib vaadelda kui testi kombinatsioonskeemile, sisendvektorite jada  $T = (t_1, t_2, t_n)$  aga nimetame *testjadaks*.

Ütleme, et testvektor  $t$  avastab rikke  $r \in R(C)$  antud skeemis siis ja ainult siis, kui kehtib  $f(X) \neq f^r(X)$ .

Riket  $r \in R$  nimetatakse *testitavaks*, kui eksisteerib vähemalt üks testvektor  $t$ , mis avastab rikke  $r$ , vastasel juhul on rike  $r$  *mittetestitav*.

Kombinatsioonskeemi  $C$  nimetatakse *liiaseks*, kui eksisteerib mittetestitav rike  $r \in R(C)$ . Sellist skeemi on alati võimalik lihtsustada, eemaldades temast vähemalt ühe ventiili või ventiili sisendi. Mittetestitavat riket võib analoogia põhjal samuti nimetada *liiaseks rikkeks*.

Oletame näiteks, et SAF1 rike (konstant 1) JAA-ventiili sisendis on mittetestitav. See aga tähendab, et ventiili funktsioon ei muutu antud rikke olemasolul, mistõttu võime panna sellesse sisendisse permanentselt konstandi 1. Aga  $n$ -sisendiga JAA-ventiil konstandiga 1 ühes sisendis on ekvivalentne  $(n - 1)$ -sisendiga JAA-ventiiliga, mis on saadud, eemaldades

ventiilist konstantse signaaliga sisendi. Kui JAA-ventiili sisend SAF0 on mittetestitav, siis võib selle JAA-ventiili üldse skeemist eemaldada ja asendada konstantse 0-signaaliga.

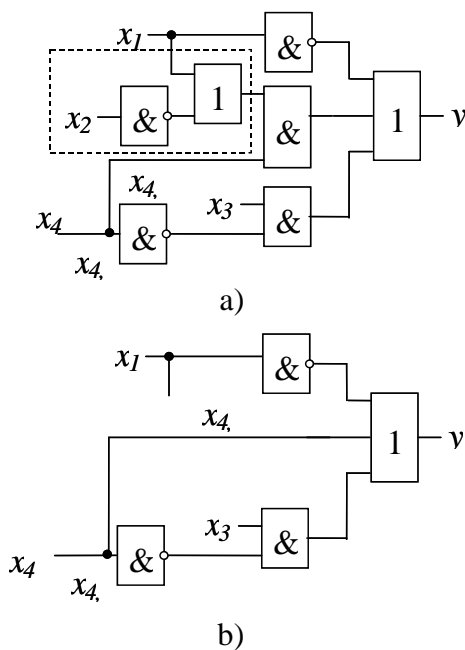
Näide 2-6

Vaatleme skeemi joonisel 2-13a. Püüdes lahendada *Boole'i diferentsiaalvõrrandit*

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

leiame, et

$$\frac{\partial y}{\partial x_2} \equiv 0$$



Joonis 2-13. Liase kombinatsioonskeemi lihtsustamine

Seega on sisend  $x_2$  skeemis liiane ja ta võib sealt koos invertoriga eemaldada.

Kuna Boole'i tuletis  $\partial y / \partial x_2$  on konstantne suvalise  $x_2$  väärtuse korral, siis on mõlemad rikked  $x_2 \equiv 0$  ja  $x_2 \equiv 1$  selles sisendis mittetestitavad ja ka VÕI elemendi võib skeemist eemaldada ning selle järel kohe ka järgmise JAA-lemendi. Kombinatsioonskeemi funktsioon on nüüd võtnud kuju

$$y = \overline{x_1} \vee x_{4,1} \vee \overline{x_3 x_{4,2}}$$

ja sellele vastav lihtsustatud kombinatsioonskeem on esitatud joonisel 2-13b. Diferentseerides saadud funktsiooni harumuutuja  $x_{4,2}$  järgi, saame:

$$\frac{\partial y}{\partial x_{4,2}} = \overline{x_1 x_3} = 1.$$

Siit näeme, et ehkki rike  $x_{4,2} \equiv 1$  on avastatav vektoriga  $(x_1, x_3, x_4) = 110$ , ei ole rike  $x_{4,2} \equiv 0$  seevastu testitav. Eemaldades nüüd invertori sisendis  $x_4$  ja seejärel mittetestitava konstandi 1 rikke tõttu ka invertorile järgneva JAA-lemendi sisendi, saame uueks lihtsustatud funktsiooniks

$$y = \overline{x_1} \vee x_4 \vee x_3.$$

Liiased mittetestitavad rikked põhjustavad tõelise probleemi testvektorite genereerimisel. Testide genereerimine on protseduur, kus lahendit tuleb otsida kõikvõimalike testvektorite või testjadade hulgast. Mitteliaste ehk testitavate rikete puhul me leiame testi harilikult kiiresti, skaneerides enamasti vaid väikest osa kogu otsimisruumist. Liaste rikete puhul aga tuleb läbida kogu otsinguruum, mis on väga töömahukas, ja kus saavutatavaks tulemuseks on pealegi teadasaamine, et otsitavat testi ei leidugi.

Seetõttu, kui eemaldaksime liiased skeemi osad juba nende projekteerimise käigus enne testide genereerimist, suudaksime testvektorite sünteesi kiirust märgatavalt tõsta.

## 2.2.2. Rikete dominantsus- ja ekvivalentsussuhted

Olgu  $y = f(X)$  mingi kombinatsioonskeemi  $C$  loogikafunktsioon ja  $R(C) = \{r\}$  kõigi võimalike rikete hulk selles skeemis.



### *Rikete ekvivalentsus*

Kaks riket  $r \in R(C)$  ja  $s \in R(C)$  on *funktsionaalselt ekvivalentsed* siis ja ainult siis, kui  $f^r(X) \equiv f^s(X)$ .

Ütleme, et testvektor  $t$  eristab kahte riket  $r \in R(C)$  ja  $s \in R(C)$  siis, kui  $f^r(X) \neq f^s(X)$ . Selliseid rikkeid nimetatakse *eristatavateks*. Ei ole olemas testi, mis eristaks kahte funktsionaalselt ekvivalentset riket.

Funktsionaalse ekvivalentsuse suhe lahutab kogu rikete hulga  $R(C)$  funktsionaalselt *ekvivalentseteks klassideks*. Testide genereerimise seisukohalt on piisav vaadelda ainult ühte esindajat igast *ekvivalentsusklassist*.

Iga  $n$ -sisendilise ventiiliga võime seostada  $2(n+1)$  üksikut konstantset riket. JAA-EI ventiili puhul kõik konstant 0 rikked sisendites ja konstant 1 rike väljundis on funktsionaalselt ekvivalentsed. Kõiki neid rikkeid võib esindada üheainsa rikkega, näiteks väljundi konstant 1 rikkega. Seega iga  $n$ -sisendilise JAA-EI ventiili puhul piisab tegeleda testide genereerimise ajal  $2(n+1)$  rikke asemel vaid  $n+2$  üksiku konstantse rikkega.

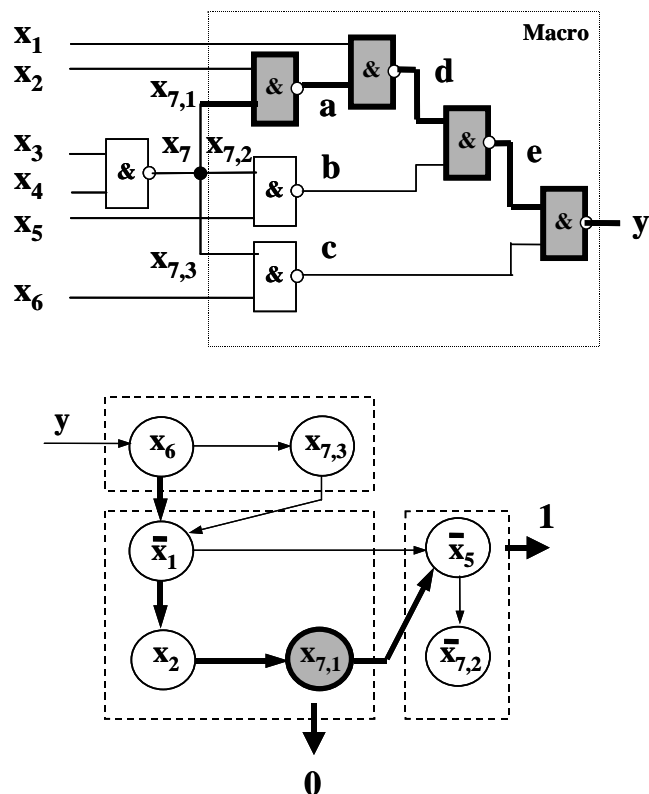
Niisugust oluliste rikete hulga redutseerimise meetodit nimetatakse *ekvivalentsete rikete kollapsiks (equivalence fault collapsing)*.

Kui lisaks rikete avastamisele on testimise ülesandeks ka rikete lokaliseerimine, tuleb genereerida niisugusi teste, mis mitte üksnes ei avasta rikkeid, vaid suudavad neid ka eristada. *Täielik rikkeid lokaliseeriv test* suudab skeemis eristada igat riket eristatavate rikete hulgast. Ehk teisiti, täielik rikkeid lokaliseeriv test suudab diagnoosida rikkeid kuni *funktsionaalse ekvivalentsuse* täpsuseni. Maksimaalse *diagnoosi resolutsiooni* (eristatavuse) määrabki funktsionaalse ekvivalentsuse suhe antud rikete hulgal.

Struktuurselt sünteesitud otsustusdiagrammide konstrueerimisel toimub automaatselt ka ekvivalentsete rikete kollaps: kui iga tipp SSBOD-mudelil esindab ühte *signaaliteed* vastavas puukujulises kombinatsioonskeemis, siis igale *tipurikkele* otsustusdiagrammis vastab samuti üks rikete alamhulk ventiilide sisendites tipule vastaval signaaliteel.

### *Näide 2-7*

Vaatleme kombinatsioonskeemi ja sellele vastavat SSBOD-mudelit joonisel 2-14.



Joonis 2-14. Rikete kollaps SSBOD-mudelis

Joonisel 2-14 on puukujuline makro  $C$  kombinatsioonskeemist ja talle vastav SSBOD. Rikete hulk makros on  $R(C)$ . Igale tipule  $m$  tipumuutujaga  $x(m)$  selles graafis vastab signaalitee  $P(m)$  kombinatsioonskeemis, mis algab muutujaga  $x(m)$  tähistatud ühendusest ja levib väljundini  $y$ . Testida tippu  $m$  graafis rikkele  $x(m) \equiv e$ ,  $e \in \{0,1\}$  hulgast  $R(C)$  tähendab testida rikete alamhulka  $F(x(m),e)$  signaaliteel  $P(m)$ .

Olgu  $P(m) = (x(m), x_1, \dots, x_n)$ , kus  $x(m)$  on sisendmuutuja signaaliteel  $P(m)$  ja  $x_1, \dots, x_n$  on ventiilide väljundmuutujad sellel teel. Siis on graafi tippu  $m$  rikkega  $x(m) \equiv e$  ekvivalentne järgmine rikete hulk signaaliteel  $P(m)$ :

$$F(x(m),e) = \{x(m) \equiv e, x_1 \equiv e \oplus INV_1, \dots, x_n \equiv e \oplus INV_n\}$$

kus  $INV_i = 0$ , kui invertorite arv sõlmest  $x_i$  kuni makro väljundini on paarisarv, ja  $INV_i = 1$  vastupidisel juhul.

Näiteks rike  $x_2 \equiv 1$  otsustusdiagrammi tipus  $x_2$  on ekvivalentne järgmise rikete alamhulgaga signaaliteel  $P(x_2)$ :

$$F(x_2,1) = \{x_2 \equiv 1, a \equiv 0, d \equiv 1, e \equiv 0, y \equiv 1\}$$

### Rikete dominantsus

Kui testimise eesmärk on üksnes rikete avastamine, siis lisaks ekvivalentsuse suhtele võib kasutada ka rikete dominantsuse suhet, mis aitab samuti redutseerida käsitlemist vajavate rikete hulka.

Olgu  $T_r$  niisuguste testvektorite hulk, mis avastavad rikke  $r \in R(C)$ . Rike  $r$  domineerib rikke  $s \in R(C)$  üle siis ja ainult siis, kui rikked  $r$  ja  $s$  on funktsionaalselt ekvivalentsed testide hulga  $T_r$  jaoks.

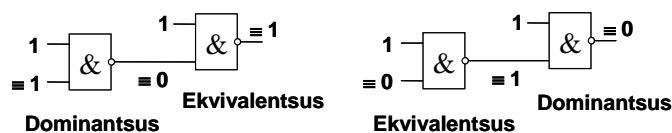
Kui rike  $r \in R(C)$  domineerib rikke  $s \in R(C)$  üle, siis iga test  $t$ , mis avastab rikke  $s$ , avastab ka rikke  $r$ . Seetõttu, kui testide genereerimise ülesandeks on ainult rikete avastamine, pole oluline käsitleda domineerivaid rikkeid  $r$ , sest genereerides testvektori rikkele  $s$ , me leiame automaatselt testi, mis avastab ka domineeriva rikke  $r$ .

### Rikete ekvivalentsus ja dominantsus

A	B	C	D	Rikete klassid
1	1	1	0	A/0, B/0, C/0, D/1 → Ekvivalentsusklass
0	1	1	1	A/1, D/0
1	0	1	1	B/1, D/0
1	1	0	1	C/1, D/0

→ Dominantsusklass

### Rikete kollaps



Joonis 2-15. Rikete kollaps ekvivalentsuse/dominantsuse suhete abil

### *Rikete kollaps*

Rikete ekvivalentsuse ja dominantsuse suhteid võib kasutada oluliste rikete hulga minimeerimiseks, mida tuleb kasutada testide genereerimisel ja rikete simuleerimisel.

Vaatleme kolme sisendiga JAA-EI elementi joonisel 2-15. Tabelis on esitatud ventiili täielik test ja iga testvektori jaoks on määratud ekvivalentsus- ja dominantsusklassid. Dominantsusklasside puhul on oluline, et selles klassis on konkreetse testvektori puhul alati üks rike, mida võib avastada ka mõne teise testvektoriga. Näiteks väljundriket  $D \equiv 0^4$  avastab koguni kolm erinevat vektorit. Seepärast see rike domineeribki temaga seotud olevates dominantsusklassides teiste rikete üle.

### *Rikete kollapsi reeglid*

1. *Ekvivalentsusreegel*: suvaline rike ekvivalentsusklassis võib saada valitud selle klassi esindajaks, jättes ülejäänud ekvivalentsusklassi rikked vaatluse alt välja.
2. *Dominantsusreegel*: dominantsusklassidest võib üksnes domineerivaid rikkeid vaatlusest välja jätta.

Vaatleme nüüd lihtsat kahe ventiiliga kombinatsioonskeemi joonisel 2-15. Testimisele kuuluvate rikete minimeerimiseks antud skeemis võib nüüd kasutada mõlemat rikete kollapsi reeglit.

Näiteks testvektori 110 puhul (vasakpoolne skeem) valime ekvivalentsusreegli põhjal väljundventiili sisendrikke  $\equiv 0$  klassi esindajaks ja seejärel, seades selle rikke dominantsussuhtesse rikkega  $\equiv 1$  skeemi sisendis, valime sisendi rikke  $\equiv 1$  kõigi rikete esindajaks vaadeldaval signaaliteel.

Testvektori 111 puhul aga (parempoolne skeem) valime dominantsusreegli põhjal väljundventiili sisendrikke  $\equiv 1$  klassi esindajaks ja seejärel seades selle rikke ekvivalentsussuhtesse rikkega  $\equiv 0$  skeemi sisendis, valime sisendi rikke  $\equiv 0$  kõigi rikete esindajaks vaadeldaval signaaliteel.

---

<sup>4</sup> Rikke  $x \equiv e$ ,  $e \in \{0,1\}$ , tähistamiseks kasutatakse ka notatsiooni  $x/e$ .

Vaadeldud näitest tuleneb vahetult üldine rikete kollapsi reegel puukujuliste skeemide jaoks: puukujulistes kombinatsioonskeemides on vaja testida üksnes sisendrikkeid.

Üldistades seda reeglit kombinatsioonskeemide üldjuhtumi jaoks, saame järgmise üldreegli.

#### *Rikete kollapsi üldreegel kombinatsioonskeemide jaoks*

Kombinatsioonskeemide testimiseks konstanttüüpi riketele on vajalik ja piisav testida

- konstantsed rikked skeemi kõigi hargnemispunktide kõikides harudes
- konstantsed rikked skeemi kõigis sisendites.

### **2.2.3. Rikete maskeerumine**

#### *Kordsed rikked*

Seni oleme vaadelnud vaid üksikuid rikkeid skeemis. Oleme teadlikult lihtsustanud niigi rasket rikete modelleerimise ülesannet, kuna juba üksikute rikete hulk digitaalsüsteemides võib osutada käsitlematult suureks, rääkimata siis suvaliste rikete kombinatsioonide arvust.

Olgu skeemis  $n$  signaalijuhet, siis üksikute konstantrikete hulk selles skeemis sisaldab  $2n$  riket. Kõikvõimalike kordsete rikete hulk sisaldab aga kokku  $3^n - 1$  riket, kus igal juhtmel võib olla 3 üksteist välistavat olekut: konstant 0, konstant 1 ja "korras" juhe.

Kui pisut lihtsustada ülesannet ja anda ette nn *lubatud rikete kordsus*  $k$ , s.t suurim korruga eksisteerivate rikete arv võib olla  $k$ , siis niisugusel juhul oleks kõigi võimalike rikkesituatsioonide (rikete kombinatsioonide) arv  $N$ :

$$N = \sum_{j=1}^k \{C_j^n\} 2^j$$

kus  $C_j^n$  on kombinatsioonide arv  $n$ -ist  $j$  kaupa.

Ülaltoodust näeme, et kordsete rikete arv on väga suur. Seega oleks mõeldamatu nende nummerdamine ja käsitlemine vähegi suuremas

kombinatsiooniskeemis. Küll aga tuleb kordsete rikete võimalust siiski arvestada testide genereerimisel just rikete vastastikuse maskeerimise ohu vältimiseks.

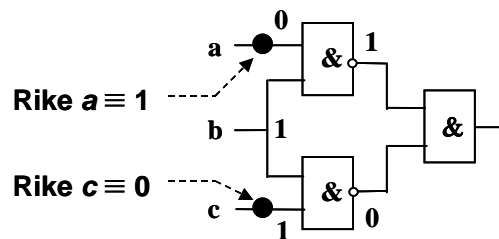
#### *Rikete maskeerumine*

Olgu  $y = f(X)$  mingi kombinatsioonskeemi  $C$  loogikafunktsioon ja  $R(C) = \{r\}$  kõigi võimalike rikete hulk selles skeemis. Olgu  $T_r$  niisuguste testvektorite hulk, mis avastavad rikke  $r \in R(C)$ .

Rike  $s \in R(C)$  maskeerib riket  $r \in R(C)$  testi  $T_r$  puhul siis ja ainult siis, kui ükski testvektor  $t \in T_r$  ei avasta kordset riket  $\{r, s\} \subset R(C)$ .

#### *Näide 2-8*

Vaatleme skeemi joonisel 2-16, kus on kaks riket  $a \equiv 1$  ja  $c \equiv 0$ , mis maskeerivad teineteist.



**Joonis 2-16.** Kahe rikke maskeerumine kombinatsioonskeemis

Testvektor  $t = (a, b, c) = 011$  on ainus võimalik vektor, mis avastaks rikke  $c \equiv 0$ . Aga see vektor ei avasta riket  $c \equiv 0$ , kui samal ajal on skeemis ka teine rike  $a \equiv 1$ . Seega võime öelda, et rike  $a \equiv 1$  maskeerib testvektori  $t$  rakendamisel riket  $c \equiv 0$ .

Olgu  $T$  kõigi võimalike testvektorite hulk ja  $T_r \subseteq T$  kõigi nende võimalike testvektorite hulk, mis avastavad rikke  $r \in R(C)$ .

Me ütleme, et rike  $s \in R(C)$  maskeerib riket  $r \in R(C)$  siis ja ainult siis, kui ükski testvektor  $t \in T_r$  ei avasta kordset riket  $\{r, s\} \subset R(C)$ .

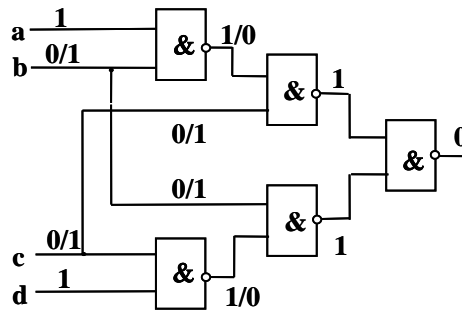
### Rikete ringmaskeeruvus

Olgu  $T$  täielik testvektorite hulk, mis avastavad skeemis  $C$  kõik üksikud konstantrikked hulgast  $R(C)$ . Kordne rike  $R' \subset R(C)$  võib aga maskeeruda niisuguse täieliku üksikrikete testi  $T$  korral, ehkki kõik rikked  $r \in R'$  eraldi oleksid testiga  $T$  avastatavad.

#### Näide 2-9

Vaatleme skeemi joonisel 2-17. Test  $T = \{1111, 0111, 1110, 1001, 1010, 0101\}$  avastab kõik üksikud konstantrikked selles skeemis. Skeemis on aga kordne rike  $R' = \{b \equiv 1, c \equiv 1\}$ . Ainus testvektor, mis avastab mõlemat riket üksikult on  $1001 \in T$ . Mõlema rikke koosinemisel aga ei testita enam kumbagi selle vektoriga, sest rike  $b \equiv 1$  maskeerib riket  $c \equiv 1$  ja rike  $c \equiv 1$  maskeerib omakorda riket  $b \equiv 1$ .

Niisugust maskeerumist nimetatakse *ringmaskeerimiseks* (*circular masking*).



Joonis 2-17. Rikete ringmaskeerumine kombinatsioonskeemis

Eelnevatest näidetest on näha, et kordsete rikete olemasolu võib mõjutada testide kvaliteeti. Seetõttu, ehkki me pole suuteliselt käsitlema kõiki võimalikke kordseid rikkeid ja konstrueerima kõigile neile sobivaid testvektoreid, on olemas veel teine võimalus – genereerida igale üksikule rikkele testi niiviisi, et see rike saaks avastatud antud testi poolt suvalise kordse rikete kombinatsiooni korral. Niisuguseid meetodeid vaatleme edaspidi.

## 2.3. Funktsionaalne rikkemudel

Digitaalsüsteemide testide genereerimise tulemuslikkus on oluliselt sõltuv süsteemi ja selle rikete diagnostilise modelleerimise viisidest. On piisavalt hästi näidatud, et konstantse rikke mudel ei suuda garanteerida piisavat testimise kvaliteeti. Põhjuseks on, et konstantse rikke mudel ignoreerib tegelikku transistorskeemide käitumist (eriti *CMOS-tehnoloogia* korral) ega kajasta adekvaatselt reaalseid füüsilisi defekte transistorskeemides, mis tihtilugu pole esitatavad ei üksikute ega ka kordsete konstantsete ricketena loogikatasandil.

Samal ajal vajame ometigi füüsiliste defektide käsitlemiseks rikete simuleerimise tööriistadega loogikatasandi mudeleid, kuna need võimaldavad vähendada simuleerimise keerukust (rida füüsilisi defekte saab modelleerida ühe ja sama loogikatasandi rikkega) ja loogikarikete mudeli kasutamise korral oleme simuleerimisel sõltumatud konkreetsest tehnoloogiast.

Selles peatükis vaatleme meetodit, kuidas modelleerida füüsilisi defekte üldistatud diferentsiaalvõrranditega, et kujutada neid füüsiliselt tasandilt loogikatasandile. Vaatleme ja analüüsime transistortasandi rikete tüüpe, et veenduda meetodi üldisuses. Defineerime ka uue funktsionaalse rikkemudeli ja näitame, kuidas seda mudelit saab käsitleda kui *universaalset liidest*, mille läbi kujutada rikkeid suvaliselt alumiselt tasandilt kõrgemale. Niisugune rikete kujutamine tasandilt tasandile teeb võimalikuks hierarhilise lähenemisviisi testimisele, mis on ka ainuvõimalik strateegia tänapäevaste järjest keerukamaks muutuvate digitaalsüsteemide testimiseks.

### 2.3.1. Rikete modelleerimine Boole'i diferentsiaalvõrranditega

Vaatleme digitaalskeemi ühte skeemisisest komponenti  $G$ , mis realiseerib Boole'i funktsiooni  $y = f(x_1, x_2, \dots, x_n)$ . Võtame kasutusele täiendava Boole'i muutuja  $d$  mingi suvalise füüsilise defekti tähistamiseks komponendis. Olgu teada ka defektse komponendi (defektiga  $d$ ) realiseeritav Boole'i funktsioon

$$y = f^d(x_1, x_2, \dots, x_n),$$

kus üldjuhul võib mõni muutuja ka puududa, juhul kui defekt  $d$  välistab nende mõju komponendi väljundile.



Võtame komponendi  $G$  jaoks kasutusele üldistatud parameetrilise funktsiooni (parameetrik on defekt  $d$ ):

$$y^* = f^*(x_1, x_2, \dots, x_n, d) = \bar{d}f \vee df^d \quad (2-1)$$

mis kirjeldab komponendi käitumist ühel ajal kahel võimalikul viisil – töökorras skeemina ja defektsena (defektiga  $d$ ). Defektse oleku korral on defektimuutuja  $d$  väärtuseks 1 ja korras skeemi puhul  $d = 0$ . Teiste sõnadega  $y^* = f^d$ , kui  $d = 1$ , ja  $y^* = f$ , kui  $d = 0$ .

Boole'i diferentsiaalvõrrandi

$$W^d = \frac{\partial y^*}{\partial d} = 1 \quad (2-2)$$

lahend kirjeldab tingimust (kitsendust), mille täitmisel defekti mõju on aktiveeritud komponendi väljundisse, s.t korras skeem ja defektne skeem annaksid kitsenduse (2-2) täitmisel väljundis erinevad signaalid.

Parameetiline defekti  $d$  modelleerimine diferentsiaalvõrranditega (2-1) ja (2-2) lubaks kasutada kitsendusi  $W^d = 1$  nii testide genereerimisel defekti  $d$  avastamiseks kui ka rikete simuleerimisel kontrollimaks, kas defekt  $d$  on antud testvektoriga avastatav. Esimesel juhul tuleks lahendada võrrand (2-2), teisel juhul tuleks arvutada võrrandi (2-2) vasaku poole väärtust.

Et leida tingimus  $W^d$  etteantud defekti  $d$  jaoks, peame kõigepealt leidma defektse komponendi funktsiooni  $f^d$  kas loogilise tuletamise teel defektse komponendi jaoks konstrueeritud *ekvivalentskeemist* või viima läbi defektse komponendi mudeli simuleerimine või korraldama reaalsed eksperimendid defektse komponendiga, et kindlaks teha defekti mõju komponendi käitumisele.

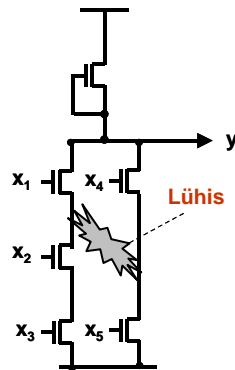
#### Näide 2-10

Vaatleme transistorskeemi, mis on joonisel 2-18 ja mis realiseerib funktsiooni

$$y = x_1 x_2 x_3 \vee x_4 x_5$$

Lühis skeemis, nii nagu on näidatud joonisel, muudab skeemi funktsiooni järgmiselt:

$$y^d = (x_1 \vee x_4)(x_2 x_3 \vee x_5)$$



*Joonis 2-18. Lühisega transistorskeem*

Võttes lühise tähistamiseks kasutusele muutuja  $d$ , konstrueerime antud defekti analüüsimiseks komponendi üldistatud funktsiooni ja diferentseerime seda muutuja  $d$  suhtes. Viies läbi vajalikud lihtsustused, leiame diferentsiaalvõrrandi lahendid, mis oleksidki defekti avastamise tingimusteks transistorskeemi väljundis

$$\begin{aligned} \frac{\partial y^*}{\partial d} &= \frac{\partial((x_1 x_2 x_3 \vee x_4 x_5) \bar{d} \vee (x_1 \vee x_4)(x_2 x_3 \vee x_5) d)}{\partial d} = \\ &= \frac{\partial(x_1 x_2 x_3 (\bar{d} \vee d) \vee x_4 x_5 (\bar{d} \vee d) \vee (x_1 x_5 \vee x_2 x_3 x_4) d)}{\partial d} = \\ &= \frac{\partial(x_1 x_2 x_3 \vee x_4 x_5 \vee (x_1 x_5 \vee x_2 x_3 x_4) d)}{\partial d} = \\ &= \overline{x_1 x_2 x_3 x_4 x_5} (x_1 x_5 \vee x_2 x_3 x_4) \\ &= \overline{x_1 x_2 x_4 x_5} \vee \overline{x_1 x_3 x_4 x_5} \vee \overline{x_1 x_2 x_3 x_4 x_5} = 1 \end{aligned}$$

Lahenditeks on järgmine tingimuste hulk:

$$T = \{10x_01, 1x_001, 01110\}$$

Igatüht nendest tingimustest võiks kasutada kui testvektorit komponendi sisendis antud defekti avastamiseks.

Toodud näidet võiks veel vaadelda kui kontranaidet tõestamaks konstantsete rikete mudeli ebaadekvaatsust. Vaatleme järgnevat tabelit.

**Tabel 2-1.**

*100%-lise KR-testi ja defektitesti võrdlus*

No	100% KR test					Defekti test				
	x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	x <sub>4</sub>	x <sub>5</sub>	x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	x <sub>4</sub>	x <sub>5</sub>
1	1	1	1	0	-	1	0	-	0	1
2	0	-	-	1	1	1	-	0	0	1
3	0	1	1	0	1	0	1	1	1	0
4	1	0	1	1	0					
5	1	1	0	-	0					

Tabelis 2-1 on viis testvektorit, mis avastavad 100%-liselt kõik konstantrikked (KR) antud skeemis (vasakul), ja kolm testvektorit, mis on genereeritud spetsiaalselt antud defekti jaoks (paremal). KR-testvektorite jaoks on toonitud ruutudega näidatud, millised rikked konkreetne testvektor avastab. 0(1) toonitud ruudus näitab, et reale vastav testvektor avastab veerule vastavas sisendis konstantrikke 1(0).

Tabelist näeme, et ükski kolmest testvektorist, mis avastavad antud lühise, ei tulene KR-testi vektoritest (ühtegi neist ei saaks tuletada, valides sobivalt muutujate vabu ehk määramata väärtusi).

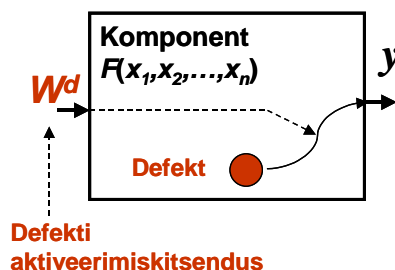
Näitest järeldub seega, et väga hea konstantsete rikete avastamiseks mõeldud test ei pruugi sugugi olla enam piisavalt hea, kui *lubatud rikete klassi* laiendada suvalistele füüsikalistele defektidele. Siit aga tuleneb vajadus üldisema rikete modelleerimise meetodi järele võrreldes sellega, mida võimaldab konstantrikete mudel.

### 2.3.2. Füüsiliste defektide kujutamine loogikatasandile

Eespool kirjeldatud Boole'i diferentsiaalvõrrandile toetuv rikete modelleerimise meetod on üldistatud lähenemisviis suvaliste defektide kujutamiseks füüsiliselt tasandilt loogikatasandile. Meetodi abil võib suvalist füüsilist defekti kirjeldada kitsendusena  $W^d = 1$ , mis tuleb täita defekti aktiveerimiseks komponendi väljundisse (joonis 2-19).

Väara signaali ilmnenist komponendi väljundis võime Boole'i diferentsiaali põhimõttel kirjeldada tingimusena  $dy = 1$ .

Võttes kasutusele *funktsionaalse rikke mudeli* mõiste, võime kirjeldada defekti  $d$  paariga  $(dy, W^d)$ . Rike avaldub nüüd puhtalt komponendi funktsiooni kaudu, rakendades komponendi sisenditele tingimusega  $W^d = 1$  määratud vektorit. Riket, kas avastatakse, kui  $dy = 1$ , või ei avastata, kui  $dy = 0$ . Komponenti madala taseme *transistorskeemi* struktuur pole enam oluline, komponenti ennast võib vaadelda musta kastina. Põhimõtteliselt pole ka tingimuste  $W^d$  leidmisel vaja teada komponendi struktuuri, kui on teada selle defektne funktsioon  $f^d$ .



Joonis 2-19. Defekti aktiveerimine komponendis

Teisest küljest võib Boole'i diferentsiaalvõrrandit (2-2) interpreteerida ka kui füüsilise defekti kujutamist füüsiliselt tasandilt loogikatasandile: füüsilise defekti asemel võib loogikatasandil käsitleda viga  $dy = 1$  (ehk konstantriket  $y \equiv 1$ ), juhul kui kitsendus  $W^d = 1$  on täidetud.

Järgmised näited demonstreerivad meetodi kasutatavust transistorskeemi defektide kujutamisel loogikatasandile.

Lühised (stuck-on) transistorskeemis

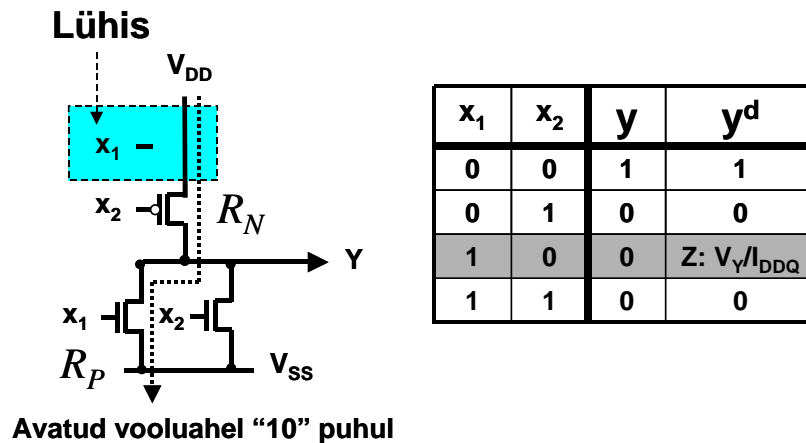
Näide 2-11

Transistortasemel esitatud VÕI-EI-elementi (joonis 2-20) käitumist vaadeldava lühisrikke korral ei saa kirjeldada rangelt loogiliselt. Sisendvektor 10 produtseerib voolu juhtiva ahela sõlmest  $V_{DD}$  sõlmeni  $V_{SS}$  ja vastav pingeniivo väljundsõlmes  $Y$  ei ole võrdne ei  $V_{DD}$  ega  $V_{SS}$ -ga, vaid selle asemel kujuneb *pinge jagunemisest* avatud transistoride *sisetakistustel* voolu juhtivas ahelas

$$V_Y = \frac{V_{DD}R_P}{(R_P + R_N)}$$

Sõltuvalt sisetakistuste suhtest ja vastavast väljundisse  $y$  ühendatud järgnevate komponentide ümberlülitumise *lävipingetest* võib defekt avalduda veana väljundis  $d$ , aga võib ka mitte avalduda. Tähistame sümboliga  $Z$  *määramatu signaali* väljundis. Siis võib defektse komponendi funktsiooni kirjeldada järgmiselt:

$$y^d = \overline{x_1} \cdot \overline{x_2} \vee x_1 \cdot \overline{x_2} Z$$



Joonis 2-20. Lühisrike (stuck-on) transistorskeemis

Kui nüüd  $\overline{x_1 x_2} = 1$ , siis  $y^d = Z$ . Kasutades valemeid (2-1) ja (2-2) saame

$$y^* = \overline{d(x_1 \vee x_2)} \vee d(\overline{x_1 x_2} \vee x_1 \overline{x_2} Z)$$

$$W^d = \partial y^* / \partial d = \overline{x_1 x_2} Z = 1$$

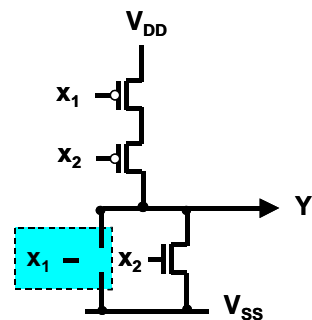
Võrrandist lähtub, et defekti aktiveerimiseks on vajalikud tingimused  $x_1 = 1, x_2 = 0$ , aga defekti avastamiseks on tarvilik ka  $Z = 1$ . Viimase tingimuse täitmine sõltub küll juba konkreetsest elektriskeemist.

*Katkestused (stuck-open) transistorskeemis*

*Näide 2-12*

Vaatleme katkestusriket VÕI-EI-elementis joonisel 2-21. Mõningate testvektorite puhul ilmneb, et vooluahelat  $V_{DD}$  ja  $V_{SS}$  vahel ei tekigi. Selle tulemusena säilitab väljundsõlm oma eelmise loogilise väärtuse. Kombinatoorne element on hakanud käituma kui *dünaamiline mälu*element.

### Katkestus



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

**Vooluahela puudumine**  
 $V_{DD}$  ja  $V_{SS}$  vahel "10" puhul

*Joonis 2-21. Katkestusriike (stuck-open, stuck-off) transistorskeemis*

Defektse ventiili funktsioon on

$$y^d = \overline{x_1 x_2} \vee x_1 \overline{x_2} y'$$

kus  $y'$  vastab väljundsignaali väärtusele, mis oli salvestatud defektse ventiili väljundis eelmisel ajahetkel. Kasutades nüüd avaldisi (2-1) ja (2-2), saame

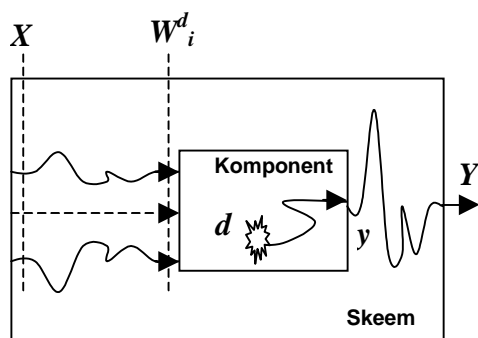
$$\begin{aligned} y^* &= \overline{d(x_1 \vee x_2)} \vee d(\overline{x_1 x_2} \vee x_1 \overline{x_2} y') = \\ &= \overline{x_2}(\overline{x_1} \vee dy') \end{aligned}$$

$$W^d = \partial y^* / \partial d = \overline{x_1 x_2} y' = 1$$

Siit aga järgneb, et defekti aktiveerimiseks vajalikud tingimused on  $x_1 = 1, x_2 = 0, y' = 1$ . Teiste sõnadega, et testida antud katkestusriket, oleks vaja rakendada 2-vektorilist testjada: kõigepealt vektorit 00, et viia väljund olekusse  $y = 1$ , ja seejärel vektorit 10, et avastada defekt.

#### Defektide esitamine defektide tabelina

Üldjuhul võib ühel ja samal defektil  $d$  olla hulk erinevaid aktiveerimistingimusi  $W^d_i$ , mis leitakse diferentsiaalvõrrandi kõigi lahendite hulganä. Olgu antud komponent  $C_y$ , mille väljundiks on  $y$ , koos kõigi võimalike defektide hulgaga  $D$ .



Joonis 2-22. Defekti aktiveerimine kogu skeemi tasandil

Olgu  $W^{F,d}(y) = \{W^d_i\}$  kõigi loogikatingimuste hulk, mis aktiveeriksid defekti  $d \in D$  väljundisse  $y$ , ning  $W^F(y) = \{W^{F,d}(y)\} \subseteq W(y)$  loogikatingimuste hulk, mis kokku aktiveeriksid kõik defektid hulgast  $D$  väljundisse  $y$ . Siin on  $W(y)$  kõigi võimalike komponendi  $C_y$  sisendvektorite hulk.

Näiteks joonisel 2-18 toodud defekti  $d$  avastamiseks väljundis  $y$  võiks kasutada kolme erinevat tingimust (tabel 2-1)

$$W^{F,d}(y) = \{10x01, 1x001, 01110\}$$

Arvestades seda, et mõned sisendmuutujate väärtused on siin määramata, on kõigi võimalike lahendite ehk aktiveerimistingimuste arv antud juhul koguni 5. Oluline oleks kõik need võimalused testide generaatori jaoks ette anda, sest mitte igauks neist ei pruugi olla lahenduv terve skeemi tasandil, s.t mitte igauhe jaoks nendest ei pruugi eksisteerida skeemi sisendvektor.

Joonisel 2-22 on illustreeritud testi genereerimine defekti  $d$  jaoks kogu skeemi ulatuses. Kõigepealt valitakse üks tingimus  $W^d_i \in W^{F,d}(y)$  defekti aktiveerimiseks komponendi väljundisse  $y$ . Seejärel lahendatakse vastavad Boole'i võrrandid tingimuse  $W^d_i = 1$  rahuldamiseks skeemi sisenditel ning aktiveeritakse *veasignaal* komponendi väljundist  $y$  skeemi väljundisse  $Y$ . Nende protseduuride tulemusena leitakse defekti  $d$  avastav testvektor  $X$  skeemi sisendis. Võib juhtuda, et sellist vektorit ei eksisteerigi. Siis tuleks valida mingi teine tingimus  $W^d_i \in W^{F,d}(y)$  ja korrata sama protseduuri. Kui testvektorit ei leita mitte ühegi tingimuse jaoks hulgast  $W^{F,d}(y)$  tuleb defekti  $d$  lugeda *funktsionaalselt liiaseks* ning selle testimine polegi antud skeemis oluline.

Loogikatingimuste hulka

$$W^F(y) = \{W^{F,d}(y)\} \subseteq W(y)$$

võib esitada *defektide tabelina* ehk maatriksina  $DT = \|w_{i,j}\|$ , kus  $w_{i,j} = 1$ , kui defekt  $i$  on avastatav komponendi lokaalse sisendvektoriga  $j$ , ning  $w_{i,j} = 0$ , kui defekt  $i$  ei ole avastatav vektoriga  $j$ . Niisugune defektide esitus loogikaelemendile AND2,2/NOR2 funktsiooniga  $y = \overline{AB} \vee CD$  on esitatud tabelis 2-2.



**Tabel 2-2**

*Defektide tabel loogikaelemendile AND2,2/NOR2*

i	Fault $d_i$	Erroneous function $f^{d_i}$	Input patterns $t_j$															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	B/C	$\text{not}((B \cdot C) \cdot (A + D))$				1								1	1	1		
2	B/D	$\text{not}((B \cdot D) \cdot (A + C))$				1								1	1		1	
3	B/N9	$B \cdot \text{not}(A)$	1	1	1						1	1	1	1				
4	B/Q	$B \cdot \text{not}(C \cdot D)$	1	1	1							1	1	1		1	1	1
5	B/VDD	$\text{not}(A + (C \cdot D))$										1	1	1				
6	B/VSS	$\text{not}(C \cdot D)$														1	1	1
7	A/C	$\text{not}((A \cdot C) \cdot (B + D))$				1				1					1	1		
8	A/D	$\text{not}((A \cdot D) \cdot (B + C))$				1				1					1	1		
9	A/N9	$A \cdot \text{not}(B)$	1	1	1		1	1	1					1				
10	A/Q	$A \cdot \text{not}(C \cdot D)$	1	1	1		1	1	1						1	1	1	
11	A/VDD	$\text{not}(B + (C \cdot D))$					1	1	1									
12	C/N9	$\text{not}(A + B + D) + (C \cdot \text{not}((A \cdot B) + D))$		1			1	1			1	1						
13	C/Q	$C \cdot \text{not}(A \cdot B)$	1	1		1	1	1	1	1	1	1		1				
14	C/VSS	$\text{not}(A \cdot B)$				1				1					1			
15	D/N9	$\text{not}(A + B + C) + (D \cdot \text{not}((A \cdot B) + C))$			1		1	1	1		1		1					
16	D/Q	$D \cdot \text{not}(A \cdot B)$	1		1	1	1	1	1	1	1		1	1				
17	N9/Q	$\text{not}((A \cdot B) + (B \cdot C \cdot D) + (A \cdot C \cdot D))$				1												
18	N9/VDD	$\text{not}((C \cdot D) + (A \cdot B \cdot D) + (A \cdot B \cdot C))$														1		
19	Q/VDD	SA1 at Q				1				1				1	1	1	1	1
20	Q/VSS	SA0 at Q	1	1	1		1	1	1		1	1	1					

Igale skeemikomponendile *komponentide teegis* lisatakse täiendav *diagnostiline informatsioon* niisuguse defektide tabeli näol. Neid tabelleid komponentide teegis saab kasutada *defektipõhisel testide genereerimisel* ja ka testide *diagnostilise analüüsi* ehk defektide avastamise protsendi arvutamiseks.

Tabelit 2-2 analüüsid on kerge märgata, et kõikide defektide avastamiseks ei olegi vaja kõiki sisendvektoreid kasutada. Näiteks kõigi 20 defekti avastamiseks piisab vaid 4-st vektorist 3, 6, 8 ja 12. Seega, elemendi defekte võiks kirjeldada kompaktselt ka niisuguse *funktsionaalse rikke mudelina*

$$W^F(y) = \{W^{F,d}(y)\} = \{0011, 0110, 1000, 1100\}$$

Nagu juba eespool mainitud, võib skeemi tasandil teste genereerides selguda, et mõni nendest vektoritest pole realiseeritav, misjuhul tuleks kõigi defektide katmiseks tabelis 2.2 kasutada hoopis teisi selles mudelis mitte sisalduvaid sisendvektoreid.

### 2.3.3. Ühendusahelate defektide kujutamine loogikatasandile

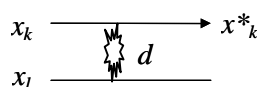
Vaatleme digitaalskeemis ühte komponenti  $C$ , mida kirjeldab Boole'i funktsioon  $y = f(x_1, x_2, \dots, x_n)$ . Kirjeldame *komponendi lähikämbust*, kus on võimalik lühiste tekkimine komponendi ja ümbruse vahel, signaalijuhtmete hulga  $E_c = \{x_{n+1}, \dots, x_p\}$ . Võtame kasutusele sama täiendava Boole'i muutuja  $d$  füüsiliste defektide modelleerimiseks alamskeemis  $(C, E_c)$  komponendi  $C$  ja tema lähikämbuse  $E_c$  vahel, mis võiksid mõjutada signaali väärtust komponendi väljundis  $y$ . Olgu teada defektse alamskeemi  $(C, E_c)$  funktsioon

$$y = f^d(x_1, x_2, \dots, x_n, x_{n+1}, \dots, x_p) \quad (2-3)$$

Võtame komponendi  $C$  ja selle lähikämbusega  $E_c$  määratud alamskeemi  $(C, E_c)$  kirjeldamiseks defekti  $d$  võimaliku olemasolu korral kasutusele *üldistatud parameetrilise funktsiooni* (parameetriks on defekt  $d$ )

$$\begin{aligned} y^* &= f^*(x_1, x_2, \dots, x_n, x_{n+1}, \dots, x_p, d) = \\ &= (\bar{d} \wedge f) \vee (d \wedge f^d) \end{aligned}$$

mis kirjeldab alamskeemi käitumist ühel ajal kahel võimalikul viisil – töökorras skeemina ja defektse (defektiga  $d$ ). Defektse oleku korral on defektimuutuja  $d$  väärtuseks 1 ja korras skeemi puhul  $d = 0$ . Teiste sõnadega  $y^* = f^d$  kui  $d = 1$ , ja  $y^* = f$ , kui  $d = 0$ . Diferentsiaalvõrrandi (2-2) lahendid kirjeldavad tingimusi (kitsendusi), mille korral defekti  $d$  mõju on aktiveeritud väljundisse  $y$ .



**Joonis 2-23.** Lühis kahe juhtme vahel

#### Näide 2-13

Vaatleme lühis kahe juhtme  $x_k$  ja  $x_l$  vahel joonisel 2-23. Defektne juhtme  $x_k$  funktsioon *lühisdefekti*  $d$  puhul JAA-tüüpi lühismudeli kohaselt oleks  $x^d_k = x_k x_l$ . Üldistatud funktsiooni saame nüüd kujul

$$x_k^* = \bar{d}x_k \vee dx_k^d = \bar{d}x_k \vee d(x_k x_l)$$

funktsioonina

$$x_k^* = f(x_k, x_l, d)$$

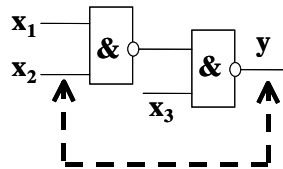
nii *defektimuutujast*  $d$ , kui ka lähiümbruse muutujast  $x_l$ , mis kirjeldab kahe juhtme paari käitumist nii rikke korral ( $d = 1$ ) kui ka rikke puudumisel ( $d = 0$ ).

Boole'i diferentsiaalvõrrandi

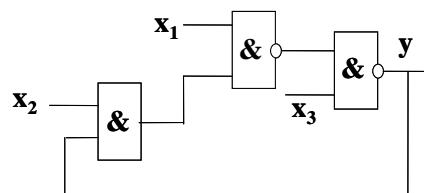
$$W^d = \partial x_k^* / \partial d = x_k \bar{x}_l$$

lahend kirjeldab tingimust, mille täitmise korral defekti  $d$  toime on jälgitav juhtmel  $x_k$  (joonis 2-23). Tingimus  $W^d = x_k \bar{x}_l = 1$  tähendab: selleks et avastada lühist juhtmete  $x_k$  ja  $x_l$  vahel, jälgides signaali juhtmel  $x_k$ , tuleb saata juhtmele  $x_k$  signaal 1 ja juhtmele  $x_l$  signaal 0.

### Lühis põhjustab tagasiside



### Ekvivalentskeem



Joonis 2-24. Lühis, mis põhjustab tagasiside

#### Näide 2-14

Lühis kahe juhtme  $x_k$  ja  $x_l$  vahel joonisel 2-24 põhjustab tagasiside. JAA-tüüpi lühisriket modelleeriv *ekvivalentskeem* on esitatud samuti joonisel 2-24. Üldistatud *parameetriline funktsioon*, mis kirjeldab skeemi käitumist nii defekti puudumisel kui ka selle olemasolul, võtab järgmise kuju:

$$y^* = \overline{d}(x_1x_2 \vee \overline{x_3}) \vee d(x_1x_2y \vee \overline{x_3}) = x_1x_2(\overline{d} \vee y')\overline{x_3}$$

Boole'i diferentsiaalvõrrandi

$$W^d = \partial y^* / \partial d = x_1x_2x_3\overline{y}' = 1$$

lahend kirjeldab tingimust (kitsendust), mille puhul defekt  $d$  on aktiveeritud väljundisse  $y$  (joonis 2-24). Apostroof  $y$  juures tähendab, et  $y$  väärtus kuulub eelmise ajahetke (takti) juurde.

Tingimus

$$W^d = x_1x_2x_3\overline{y}' = 1$$

tähendab, et lühise  $d$  testimiseks on vaja kahest vektorist koosnevat *testjada*: esiteks, vektorit, mis produtseerib skeemi väljundis väärtuse  $y = 0$  (näiteks  $x_3 = 0$ ), seejärel defekti avastamiseks vektorit  $x_1 = 1, x_2 = 1, x_3 = 1$ .

Näide demonstreerib, et defektide testimiseks vajalikud tingimused võivad üldjuhul puudutada mitut ajahetke, mille tulemusel tuleb genereerida mitte üks vektor, vaid mitmest testvektorist koosnev testjada. Ka näeme siit, et *Boole'i diferentsiaalvõrrandi* lahendamisel põhinev meetod võimaldab universaalselt käsitleda ühtselt nii kombinatoorseid lühiseid kui ka võrdlemisi keerukat sekventsiaalsete lühisdefektide juhtu, kus lühisest tingituna kombinatsioonskeem muutub järjestikskeemiks (või antud järjestikskeemi olekute arv kasvab).

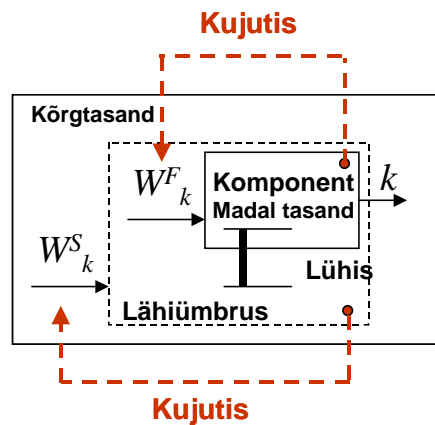
*Funktsionaalse rikke mudelit*, mis on esitatav paarina  $(dy, W^d)$ , võib vaadelda esiteks meetodina suvaliste füüsikaliste defektide kujutamiseks füüsikaliselt tasandilt loogikatasandile ja teiseks universaalse meetodina rikete modelleerimiseks testide genereerimise ja rikete simuleerimise hierarhilises käsitluses. Tingimusi  $W^d$  defektide  $d$  aktiveerimiseks võib seejuures kasutada kitsendustena kõrgematel (loogika või registersiirete)

tasanditel kas testide genereerimiseks või rikete simuleerimiseks osutamata tähelepanu madalama taseme rikete (nt. defektide) füüsilisele olemusele.

### 2.3.4. Rikete hierarhiline esitus

*Rikete defineerimise meetod* mudeli  $(dy, W^d)$  kaudu võimaldab unifitseerida skeemi (või süsteemi) komponentide ja nende lähiümbruse *diagnostilist modelleerimist*, pööramata tähelepanu komponendi ega ka selle lähiümbruse struktuursetele omadustele. Mõlemal juhul kirjeldab mudel  $(dy, W^d)$ , kuidas madalama tasandi defekti  $d$  (kas defekti komponendis või selle lähiümbruses) tuleks aktiveerida kõrgemale tasandile sõlmeni  $y$ .

Kitsendusi  $W^d$  võib kasutada nii hierarhilisel testide genereerimisel kui ka rikete simuleerimisel. Vaatleme sõlme  $k$  joonisel 2-25 kui komponendi  $M_k$  väljundit, millele vastab muutuja  $y_k$ . Seame sõlmega  $k$  vastavusse rikete hulga  $R_k = R_k^F \cup R_k^S$ , kus  $R_k^F$  on rikete alamhulk komponendis  $M_k$  väljundiga  $y_k$  ja  $R_k^S$  on komponendiga  $M_k$  seotud struktuursete rikete (defektide) alamhulk komponendi  $M_k$  lähiümbruses.

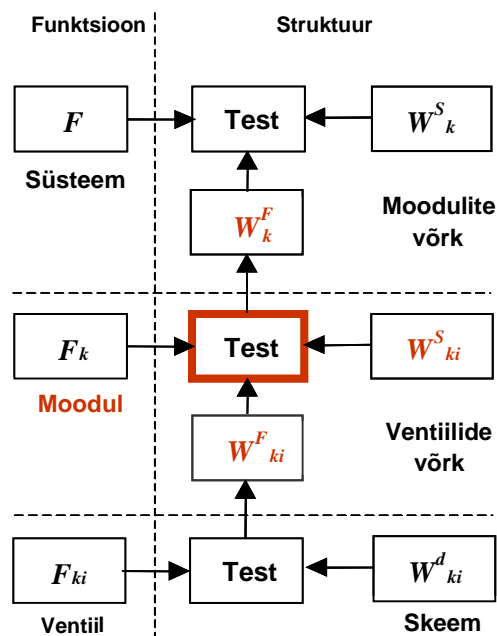


Joonis 2-25. Lühis, mis põhjustab tagasiside

Olgu  $W^d \in W(y_k)$  tingimus, mille täitmise korral rike  $d \in R_k$  muudab muutuja  $y_k$  väärtust ( $dy_k = 1$ ). Olgu  $W_k^F \subset W(y_k)$  tingimuste hulk, mis

aktiveerivad defekte  $d \in R_k^F$ , ja  $W_k^S \subset W(y_k)$  nende tingimuste hulk, mis aktiveerivad defekte  $d \in R_k^S$ .

Kasutades tingimusi  $W_k^F$  ja  $W_k^S$ , seame üles rikete kujutised madalamalt tasandilt kõrgemale tasandile hierarhilise testide genereerimise eesmärgil, aga ka vastupidi, kõrgemalt tasandilt madalamale tasandile hierarhilise rikete simuleerimise või diagnoosi eesmärgil.



*Joonis 2-26. Hierarhiline testimise kontseptsioon*

Et kujutada rikkeid  $d \in R_k$  testide genereerimise eesmärgil madalamalt tasandilt kõrgemale muutuja  $y_k$  diferentsiaaliks  $dy_k$  on vaja leida lahend võrrandile  $W^d = 1$ . Teiste sõnadega, kui kitsendus  $W^d = 1$  on täidetud, siis defekti  $d \in R_k$  olemasolu korral avaldub see signaali  $y_k$  muutusena  $dy_k = 1$ .

Rikete simuleerimisel (või rikete diagnoosil) võib muutuja  $y_k$  väärat väärtust ( $dy_k = 1$ ) formaalselt interpreteerida implikatsioonina

$$dy_k \rightarrow d_1 W^{d_1} \vee d_2 W^{d_2} \vee \dots \vee d_n W^{d_n} \quad (2-4)$$

kus  $\forall j = 1, 2, \dots, n: d_j \in R_k$ . Teiste sõnadega, kõrgtasandi sündmusele  $dy_k = 1$  seatakse vastavusse madalama tasandi sündmus  $d_j$  siis, kui tingimus  $W^{d_j} = 1$  on täidetud.

Hierarhilise testimise käsitluse eesmärgil peaksime konstrueerima iga komponendi  $M_k$  jaoks skeemis *rikete nimistu*  $R_k$  koos vastavate kitsendustega  $W^d$  iga rikke jaoks  $d \in R_k$ . Kitsenduste hulga  $W^F_k$  võib genereerida madalamal tasandil kõigile komponendi riketele  $d \in R^F_k$ . Kitsenduste hulga  $W^S_k$  struktuursete rikete jaoks  $d \in R^S_k$  komponendi lähieesmärges võib genereerida Boole'i diferentsiaalanalüüsi abil eelmistes punktides kirjeldatud meetodil.

Joonisel 2-26 on illustreeritud hierarhilise testimise kontseptsiooni kolme tasandi süsteem-moodul-ventiil vahel, nagu eelnevalt kirjeldatud. Funktsionaalse lähenemise korral kasutatakse informatsiooni ainult vaadeldava tasandi objekti funktsionaalse käitumise kohta. Struktuurse lähenemisviisi puhul on testimise objektideks rikked struktuurse võrgu komponentides ja nendevahelistes ühendustes.

## 2.4. Kõrgtasandi rikete mudelid

Et kiirendada testide simuleerimist ja rikete katte hindamist, on välja töötatud kõrgtasandi (funktsionaalse või käitumusliku tasandi) rikkemudeleid.

*Kõrgtasandi rikkemudelid* väljendavad madala tasandi füüsiliste defektide mõju süsteemi käitumisele kõrgematel funktsionaalsetel tasanditel – mikrotehete või operatsioonide täitmisel registersiirete tasandil või arvuti käskude täitmisel käsusüsteemide arhitektuursel tasandil. Kõrgtasandi rikkemudelit loetakse heaks, kui selle abil genereeritud testidel on hea konstantsete rikete või füüsiliste defektide kate (avastamise võime).

Rikete kõrgtasandil modelleerimise põhiidee seisneb süsteemi kõrgtasandi kirjelduse modifitseerimises rikete sisseviimisega nii, et süsteemi esialgselt kirjeldusest tekiks vigaseid versioone. Niiugust lähenemisviisi nimetatakse *mudeli häirimiseks (model perturbation)* [Gup 85]. Mudelit saab häirida mitmel viisil, näiteks tõeväärtustabeli modifitseerimisega, mikrotehete modifitseerimisega jne. Nimetatud idee on

realiseeritud ühel või teisel viisil erinevates kõrgtasandi rikete mudelites, mis on välja töötatud spetsiaalselt näiteks mikroprotsessoritele [Tha 80] pisut üldisemal moel registersiirete tasandil esitatud süsteemidele [She 88] või VHDL-keeles kirjeldatud digitaalsüsteemidele [Gho 91] jne. On tehtud katseid töötada välja eri kõrgtasandi rikete mudeleid spetsiaalsetele süsteemikomponentidele nagu dekodeerid, multiplekserid, mälud, programmeeritavad maatriksid jne.

Kõrgtasandi rikete mudelid võivad olla otsesed ja kaudsed [Abr 95]. *Otsene rikkemudel* identifitseerib iga rikke eraldi ja iga rike selles mudelis on üheks konkreetseks testide genereerimise eesmärgiks. *Kaudne rikkemudel* identifitseerib rikete klasse, kus rikked omavad mingeid sarnaseid omadusi, nii et kõiki ühe klassi rikkeid saaks käsitleda sarnaste protseduuride abil. Kaudse mudeli eeliseks on, et ta ei nõua rikete otsest nummerdamist klassi sees.

Enamik kõrgtasandi rikkeid, mida selles peatükis käsitleme, võib samastada nn *adresseerimise rikete (addressing faults)* [Abr 95] tüübiga. Iseloomulikumateks näideteks on sõna adresseerimine mälus, registri valik vastavalt konkreetse välja sisule protsessori *käsuvormingus*, tehtekoodi dekodeerimine, et kindlaks teha, millist käsku on vaja täita arvutis jne. Üldiseks põhimõtteks selles mudelis on kasutada  $n$ -bitist aadressi, et valida ühte  $2^n$  võimalikust objektist. Millal iganes üks objekt  $i$  on valitud, adresseerimise rike võib tähendada ühte kolmest võimalusest:

- ei valita ühtegi objekti;
- valitakse objekt  $j$  objekti  $i$  asemel;
- valitakse objekt  $j$  lisaks objektile  $i$ .

Veelgi üldisemalt: võidakse valida üks hulk objekte  $\{j_1, j_2, \dots, j_k\}$  objekti  $i$  asemel või lisaks objektile  $i$ .

Adresseerimisrikete mudeli puhul on oluline kontrollida, et adresseeritud funktsioon täidetakse ja et selle kõrval ei toimuks muid tegevusi, mis pole ette nähtud. Seda viimast olulist aspekti ignoreerivad tihti lihtsustamise eesmärgil mitmesugused teised heuristilised rikete mudelid.

#### **2.4.1. Mikroprotsessori funktsionaalne rikete mudel**

Väga levinud on lähenemisviis, kus mikroprotsessori erinevate alamsüsteemide nagu andmetöötluse alamsüsteemi (operatsioonautomaadi) ja juhtseadme (juhtautomaadi) jaoks kasutatakse erinevaid rikete mudeleid



[Tha 80, Abr 95]. Kuna kõrgtaseme operatsioonid toimuvad kõrgtasandil, mis on inimesele hästi jälgitav, siis tuleks anomaaliate puhul süsteemi käitumises rääkida pigem *vigadest* kui rikestest, mis vastab ka paremini nende sõnade igapäevasele tähendusele eesti keeles. Säilitamiseks aga eespool defineeritud suhet sõnade vahel – defekt, rike, viga, kus rike tähendab defekti mudelit, viiakse protsessori funktsioonide „materialiseerimise” eesmärgil tinglikult sisse nn *mehhanismi* mõiste. Mehhanismiks loetakse mingit tinglikku osa aparatuurist, mis realiseerib või vastutab teatavate funktsioonide täitmise eest.

Kõrgtasandi rikete mudeli konstrueerimise seisukohalt võib mikroprotsessori jagada järgmisteks mehhanismideks: registreite dekodeerimise, käskude dekodeerimise, andmete säilitamise, andmete edastamise ja andmete töötlemise mehhanismid.

Niisiis, kõrgtasandi rikked, mis mõjutavad mikroprotsessori käitumist, võib jagada järgmisteks konkreetsete mehhanismidega seotud rikete klassideks:

- *adresseerimise rikked*, mis mõjutavad registreite dekodeerimise mehhanismi,
- *adresseerimise rikked*, mis mõjutavad käskude dekodeerimise ja mikrotehete järjestamise mehhanisme,
- rikked andmete säilitamise mehhanismis,
- rikked andmete edastamise mehhanismis,
- rikked andmete töötlemismehhanismis.

#### *Rikked registreite dekodeerimise mehhanismis*

Registreite dekodeerimise vigade põhjuseks võivad olla defektsed multiplekserid või defektsed demultiplekserid.

Kui on antud mingi *andmeallika* aadress, siis multiplekseris on võimalikud järgmised anomaaliad, mis moodustavad registreite dekodeerimise mehhanismi rikete mudeli:

F1: ei valitagi allikat,

F2: valitakse vale allikas,

F3: valitakse rohkem kui üks allikas, kusjuures multiplekseri väljund määratakse JAA või VÕI tehtega (sõltuvalt *tehnoloogiast*) üle kõigi andmesõnade, mis on valitud.

Demultiplekserite puhul on aga võimalikud järgmised *käitumisvead*, mis lisanduvad rikete mudelisse:

F4: ei valitagi sihtkohta;

F5: valitud sihtkoha asemel või sellele lisaks valitakse üks teine või veelgi rohkem sihtkohti.

#### *Rikked käskude dekodeerimise mehhanismis*

Mikroprotsessori käsku *I* võib vaadelda kui ühte *mikrokäskude* jada, kus iga mikrokäsk väljastab juhtsignaale, mille toimel realiseeruvad paralleelselt mikrotehted. *Mikroteheteks* on elementaarsed andmesiirded või tehted andmetega.

Adresseerimise vead, mis mõjutavad käskude täitmist või mikrotehete järjestust, võivad kutsuda esile järgmisi effekte (käskude dekodeerimise mehhanismi rikete mudel):

F6: üks või enam mikrotehet jäävad aktiveerimata käsu *I* poolt;

F7: mikrotehteid aktiveeritakse ekslikult;

F8: õigete mikrokäskude asemel või nendele lisaks aktiveeritakse komplekt mikrokäske, mida ei peaks aktiveerima.

Kirjeldatud rikete mudel on üldine, kuna see võimaldab defektide modelleerimiseks nii *osalist käskude täitmist* kui ka nn “uute” käskude tekkimist, mis pole ette nähtud mikroprotsessori käsusüsteemiga.

#### *Rikked andmete säilitamise mehhanismis*

Kuna registreid võib vaadelda kui ühte *mälumassiivi*, siis seda tüüpi rikete jaoks võib kasutada traditsioonilisi *mäluseadmete rikete mudeleid* [Goo 91, Abr 95, Mou 00]. Vead andmete säilitamisel mälus võivad toimuda järgmiselt (andmete säilitamise mehhanismi mudel):

F9: ühes või enamas mälualemendis võivad olla rikked konstant 1 või konstant 0;

F10: ühes või enamas mälualemendis võivad andmete edastamisel tekkida vead  $0 \rightarrow 1$  või  $1 \rightarrow 0$ ;

F11: kaks või rohkem mälualemendi paari võivad olla seotud, s.t kui ühes mälualemendis olek muutub näiteks viisil  $x \rightarrow y$ , siis temaga paaris olevas mälualemendis muutub olek kas  $x \rightarrow y$  või  $y \rightarrow x$ , kusjuures  $x, y \in \{0,1\}$  ja  $y \neq x$ .

On näidatud [Tha 77], et rikked *mäludekoodris* ja ka sisend-väljund-registris on interpreteeritavad vigadena mälumassiivis.

#### *Rikked andmete edastamise mehhanismis*

Andmete edastamise funktsioon hõlmab kogu andmesiiret magistraalide kaudu registritest registritesse kas otse või läbi andmetöötlemehhanismi. Andmete edastamise mehhanismi rikete mudel koosneb järgmistest riketest:

F12: ühel või enamal signaalijuhtmel võivad olla rikked konstant 1 või konstant 0;

F13: üks või enam signaalijuhet võivad olla lühises ja vastavalt lühisrikke mudelile (JAA-tüüpi või VÕI-tüüpi lühiste) võivad realiseerida vastavalt, JAA- või VÕI-funktsioone lühises olevate juhtmete peal.

#### *Rikked andmete töötlemise mehhanismis*

Andmete töötlemise mehhanismi jaoks pole hästi töötavaid rikkemudeleid välja pakutud. Ainsaks mõistlikuks viisiks oleks kasutada *hierarhilist lähenemisviisi*, mille puhul mehhanismi kuuluvad funktsionaalsed plokid esitatakse mingil madalamal tasandil, näiteks ventiilskeemide tasandil, ja lokaalsed testid genereeritakse sellel samal madalal tasandil. Neid lokaalseid teste kasutatakse lõplike *testprogrammide* koostamisel taas kõrgtasandil.

Ehkki käesolevas peatükis kirjeldatud rikete mudel on laialt levinud testide genereerimiseks mikroprotsessoritele, on selle mudeli puuduseks liigne spetsialiseeritus, mis raskendab testprogrammide sünteesi

automatiseerimist. Ka on mudel liialt orienteeritud kitsale mikroprotsessorite klassile, mistõttu pole üldjuhul digitaalsüsteemidele kasutatav.

## 2.4.2. Registersiirete tasandi rikete mudelid

*Registersiirete rikete mudel* on üles ehitatud kajastamaks teatavat hulka eri tüüpi funktsioneerimise anomaaliaitena esitatavaid *funktsionaalseid rikkeid*. Rikete hulk on koostatud registersiirete tasandit kirjeldavate keelte *lauseanalüüsi* tulemusena.

Registersiirete tasandit kirjeldavas keeles on lausel üldjuhul järgmine struktuur [She 88]:

$$K: (T,C) R_d \leftarrow f(R_{S1}, R_{S2}, \dots, R_{Sn}), \rightarrow N \quad (2-5)$$

kus

- $K$  - *märgend*,
- $T$  - ajaliselt määratud tingimus (*ajatingimus*), mis on tarvilik lausega määratud tegevuste realiseerimiseks;
- $C$  - *loogikatingimus*, mis on tarvilik lause (tegevuse) realiseerumiseks;
- $R_d$  - *sihtregister*, kuhu kirjutatakse tegevuse tulemus;
- $R_{Si}$  - *andmeallikas* (register, andmesisend);
- $f$  - *tehe*, mis sooritatakse andmeallikatest  $R_{Si}$  saadud operandidega;
- $\leftarrow$  - tähistab *andmesiiret* (tehte tulemuse kirjutamist sihtregistrisse  $R_d$ );
- $\rightarrow N$  - tähistab siiret järgmise lause (tegevuse) juurde aadressiga  $N$ .

Toodud tähistuste alusel võime eristada üheksat tüüpi *funktsionaalseid rikkeid*, mida kasutatakse registersiirete tasandil digitaalsüsteemide diagnostilisel modelleerimisel:

- RT1: märgenditega seotud rike ( $K/K'$ ), mis tähendab, et antud tegevus realiseeritakse vaid siis, kui järgmise tegevuse valik toimub suunamisega märgendi  $K'$  poole;
- RT2: ajatingimuste ülesseadmisega seotud rikked ( $T/T'$ ), kus ajatingimuse  $T$  asemel kontrollitakse hoopis ühte teist ajatingimust  $T'$ ;
- RT3: loogikatingimuste ülesseadmisega seotud rikked ( $C/C'$ ), kus tingimuse  $L$  asemel kontrollitakse hoopis ühte teist loogikatingimust  $L'$ ;
- RT4: registrite (sihtregistrite  $R_d$  või andmeallikate  $R_{s,i}$ ) dekodeerimisega seotud rikked ( $R_i/R_i'$ ), kus registri  $R_i$  asemel valitakse hoopis teine register  $R_i'$ ;
- RT5: tehete dekodeerimisega seotud rikked ( $f/f'$ ), kus tehete  $f$  asemel realiseeritakse hoopis teine tehe  $f'$ ;
- RT6: juhtimisega seotud rikked, kus järgmine lause valitakse märgendi  $N'$ , mitte aga märgendi  $N$  järgi;
- RT7: rikked andmete salvestamisel või säilitamisel ( $(R_s)/(R_s)'$ ), mis tähendab, et registri  $R_s$  sisu ( $R_s$ ) võtab mingi defekti tõttu uue kuju ( $R_s$ )';
- RT8: rikked andmete edastuse mehhanismis ( $\leftarrow/\leftarrow'$ ), mis tähendab, et toimub mingi viga andmete edastamisel andmeallikast sihtkohta;
- RT9: rikked andmete töötlemise mehhanismis ( $(f)/(f)'$ ), mis tähendab, et tehe  $f$  küll toimub, aga tehete tulemus ( $f$ )'erineb oodatavast tulemusest ( $f$ ).

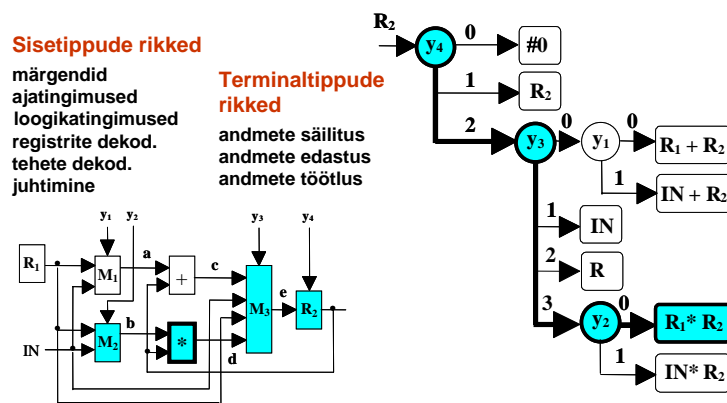
Kirjeldatud rikete mudel on üsna üldine ja sisukas, kuna igat digitaalsüsteemi on võimalik kirjeldada registertasandi keeltes selliste lausetega. Mudeli lihtsustamiseks on võimalik läbi viia ka rikete kollapsi protseduure, toetudes rikete ekvivalentsuse ja dominantuse suhetele.

Mudeli puuduseks on tema kohmakus – liiga palju erinevaid rikete tüüpe, mis kõik vajavad eraldi defineerimist ja eri käsitlust.

### 2.4.3. Rikete modelleerimine otsustusdiagrammidega

Eespool kirjeldatud kõrgtasandi rikete mudelid on liigselt spetsialiseeritud ja nõrgalt seotud süsteemi funktsioonide ning struktuuri mudeliga, s.t ei tulene otseselt süsteemi mudelist, vaid rikete mudel tuleb lisaks veel spetsiaalselt defineerida. Rikete mudelis kasutatavate rikete tüüpide rohkus teeb keeruliseks mudeliga töötamise. Tüüpide rohkuse tõttu on raske välja töötada ühtseid algoritme testide genereerimiseks ja rikete simuleerimiseks nii, nagu see osutus võimalikuks loogikatasandil konstantse rikke mudelit kasutades.

Otsustusdiagrammide kasutamine annab paremad võimalused *universaalse rikkemudeli* väljatöötamiseks ka kõrgematel modelleerimise tasanditel. OD-mudel kajastab sisuliselt täpselt sama informatsiooni, mida esitatakse registersiirete tasandi traditsiooniliste keelte (nt VHDL-keele) abil, kuid palju korrastatumal ja diagnostilise modelleerimise seisukohast otstarbekamal kujul, kuna on selgelt välja toodud põhjuse-tagajärje suhted. OD-mudelis on esitatavad kõik eespool vaadeldud rikete mudelid, kusjuures neid ei tule lisaks süsteemi mudelile täiendavalt konstrueerida, vaid kõrgtasandi rikete mudel tuleneb otseselt otsustusdiagrammist endast (loomulikul viisil): iga tipuga on seotud mingi konkreetne rikete tüüp ja hulk, kusjuures kõikide rikete käsitlus on standardne kõikide rikkettüüpide jaoks ning kõikide tippude jaoks. Praktilisest küljest lähtudes on aga siiski otstarbekam eristada OD sisetippude ja terminaltippude käsitlust.



Joonis 2-27. Kõrgtasandi rikete mudel otsustusdiagrammidel

Vaatleme digitaalsüsteemi (operatsioonautomaadi andmeosa) ja selle otsustusdiagrammi joonisel 2-27. OD terminaltippudele vastavad andmete säilituse funktsioonid (tipud muutujatega  $R_1$ ,  $R_2$  ja  $IN$ ), andmete edastuse funktsioonid (tipud muutujatega  $R_1$ ,  $R_2$  ja  $IN$ ) ning andmete töötlemise funktsioonid (tipud avaldistega  $R_1 + R_2$ ,  $IN + R_2$ ,  $R_1 * R_2$ ,  $IN + R_2$ ), sisetippudele aga tingimuste kontrolli ja juhtimise funktsioonid. Joonisel 2-27 esitatud graafi sisetippudel kontrollitakse juhtsignaalide väärtusi. Üldjuhul võidakse otsustusdiagrammide sisetippudes kontrollida ka *siirdeadresse* (märgendeid), aja- ning loogikatingimusi, ja *juhtsõnade* koodivälju, mille järgi adresseeritakse nii andmete allikaid (registreid, magistraale, sisendeid) kui ka selekteeritakse sooritatavaid tehteid.

Iga aktiveeritud tee otsustusdiagrammis juurtipust kuni terminaaltipuni kirjeldab digitaalsüsteemi ühte konkreetset *töörežiimi*, mida saaks kirjeldada ka ühe lausega (2-5) mingis registersiirete tasandi kirjeldamise keeles. Seega on kõik selle lause (2-5) komponendid nii või teisiti esitatud ka otsustusdiagrammis. Kõik rikked, mis võivad mõjutada antud töörežiimi, on võimalik siduda OD tippudega töörežiimile vastaval aktiveeritud teel. Iga rike, mis on seotud konkreetse tipuga aktiveeritud teel, põhjustab kõrvalekalde sellelt teelt, mis tähendab, et jõutakse mingisse teise terminaltippu, kus sooritatakse mingi teine tehe ettenähtud tehte asemel.

OD terminaltippudega, millele vastavad andmetöötlusega seotud funktsioonid, on olukord pisut keerulisem. Nii nagu juba eespool mainitud on siin üldjuhul raske välja pakkuda hästi töötavat kõrgtaseme rikete mudelit. Ainsaks mõistlikuks meetodiks oleks läheneda probleemile hierarhiliselt ja sünteesida *lokaalsed testvektorid* terminaltippude testimiseks madalamal tasandil. Näiteks kui *tipuavaldiseks* on aritmeetiline liitmisetehe, siis selle tehte testimiseks vajalikud operandid tuleks genereerida summaatori ventiiltaseme mudeli peal.

#### *Universaalse rikkemudeli definitsioon*

Igas digitaalsüsteemi kirjeldava otsustusdiagrammi tipus võivad aset leida järgmised rikked:

- D1: tipust väljuv kaar on alati (konstantselt) aktiveeritud (sellesse tippu jõudes jätkame teed graafil alati just selle kaare kaudu sõltumata sellest, milline väärtus on tipumuutujal);
- D2: tipust väljuv kaar on katki ehk avatud (sellesse tippu jõudes katkeb edasine liikumine graafis, mistõttu ühtki tehet ei toimu);

D3: tipust väljutakse mingi teise kaare või korruga mitme teise kaare kaudu ettenähtud kaare asemel või sellele lisaks (sellisel juhul võidakse jõuda mitmesse terminaltippu korruga ning võidakse sooritada korruga mitu tehet, kusjuures saadud tehete tulemused võetakse kokku JAA- või VÕI tehtega vastavuses süsteemis realiseeritud *tehnoloogiale*).

**Tabel 2-3.**

*Rikete mudelite vastavus*

DD-mudeli rikked	Registersiirete rikked	Mikroprotsessori rikked	
		Käsitasetasand	Mikrokäsitasetasand
D1	RT1 – RT6	F1	F1, F6
D2		F2, F4	F2, F4
D3		F3, F5	F3, F5, F7, F8
Terminaltipp	RT7 – RT9	F9 - F13	F9 – F13

On kerge näha, et otsustusdiagrammide jaoks defineeritud ühtne rikkemudel katab eespool vaadeldud mikroprotsessori ja registersiirete tasandi rikkemudeleid ja on sisuliselt üldistuseks kõikidele käsitletud kõrgtasandi rikete mudelitele.

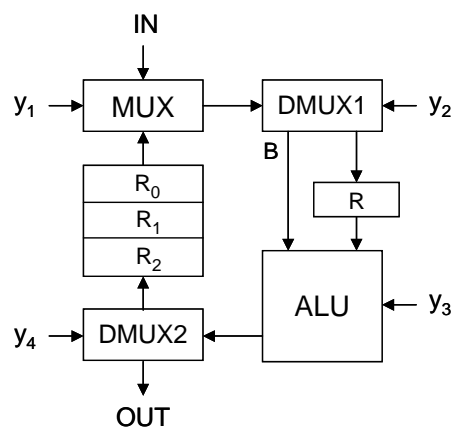
Universaalse otsustusdiagrammi rikkemudeli vastavus registersiirete tasandi ja mikroprotsessori rikete mudelitele on tabelis 2-3.

Vaatleme järgnevalt mitmesuguste digitaalsüsteemide rikete modelleerimise näiteid otsustusdiagrammidel.

*Registersiirete tasandi rikked*

Joonisel 2-28 on digitaalsüsteemi *andmetöötlusplokk*, mille komponentide funktsioonid on tabelis 2-4. Seade koosneb neljast registrist R, R<sub>0</sub>, R<sub>1</sub>, R<sub>2</sub>, multiplekserist MUX, kahest demultiplekserist DMUX ning aritmeetikaseadmest ALU.





Joonis 2-28. Digitaalsüsteemi andmetötlusplokk

Tabel 2-4.

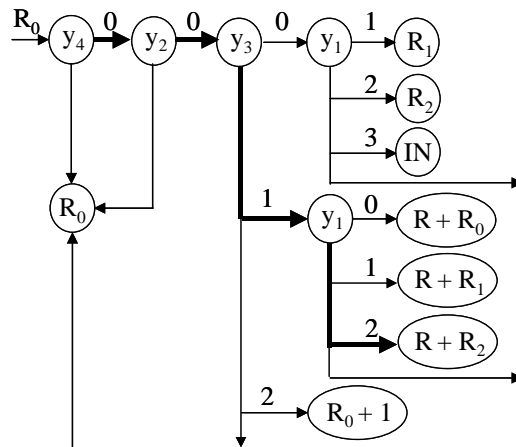
Andmetötlusploki funktsioonid

MUX		DMUX1		ALU		DMUX2	
$y_1$	Tehe	$y_2$	Tehe	$y_3$	Tehe	$y_4$	Tehe
0	MUX = $R_0$	0	B = MUX	0	ALU = B	0	$R_0$ = ALU
1	MUX = $R_1$	1	R = MUX	1	ALU = B + R	1	$R_1$ = ALU
2	MUX = $R_2$	2	∅	2	ALU = B + 1	2	$R_2$ = ALU
3	∅	3		∅	3	∅	3

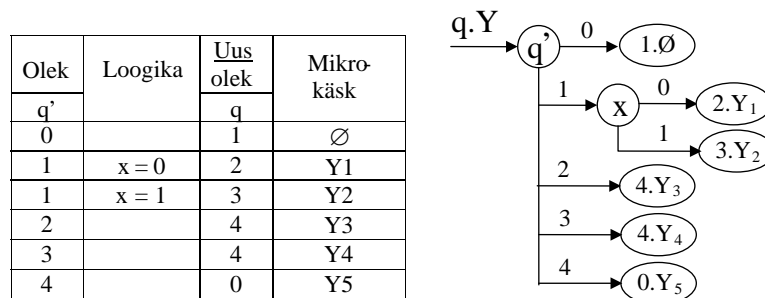
Registri  $R_0$  käitumine on esitatud otsustusdiagrammiga joonisel 2-29, kus argumentideks on juhtmuutujad (mikrokäsu operatsiooniosa väljad)  $y_1$ ,  $y_2$ ,  $y_3$ ,  $y_4$ , sisendmuutja  $IN$  ning registermuutujad  $R$ ,  $R_0$ ,  $R_1$ ,  $R_2$ .

Registersiirete tasandi rikked RT2 – RT5 on otseselt kaetud OD-mudeli sisetippude riketega D1- D3, mida saab kergesti interpreteerida kui aja- või loogikatingimuste rikkeid või registreid ning operatsioonide dekodeerimise (adresseerimise) rikkeid. Nii näiteks vastab joonisel 2-29 tipp  $y_1$  registreid/andmeallikate dekodeerimisele (MUX joonisel 2-28), tipp  $y_2$  esitab loogikatingimust (demultiplekser DMUX2), tipp  $y_3$  vastab mikrooperatsioonide dekodeerimisele (ALU) ning tipp  $y_4$  sihtregistri dekodeerimisele (DMUX2).

Rikete klassid RT1 (mürgendiga seotud rikked) ja RT6 (juhtimisega seotud rikked) esindavad juhtseadme rikkeid, mida samuti on kerge interpreteerida otsustusdiagrammidel.



Joonis 2-29. Digitaalsüsteemi andmeosa otsustusdiagramm



Joonis 2-30. Juhtseadme olekute/siirete tabel ja otsustusdiagramm

Joonisel 2-30 on toodud juhtseadet kirjeldav lõpliku automaadi mudel (olekute ning siirete tabel) ning sellele vastav otsustusdiagramm, mis esitab vektorfunktsiooni  $q.Y = \delta(x, q'). \lambda(x, q')$ , kus  $x$  on Boole'i muutuja

(loogikatingimus),  $q$  on arvutatav olekumuutuja,  $q'$  on eelmisele ajahetkele vastav olekumuutuja ja  $Y$  on juhtseadme väljundmuutuja (mikrokäsk). Rikete klassidele RT1 ja RT6 vastavad graafi sisetipu  $q'$  ja terminaltippude  $(q, Y)$  rikked. Tipu  $x$  rike otsustusdiagrammis vastab rikete klassile RT2 või RT3.

Rikete klasside RT7 – RT9 vastavus joonisel 2-29 toodud otsustusdiagrammi terminaltippude riketega on toodud tabelis 2-5.

**Tabel 2-5.**

*Juhtseadme olekute/siirete tabel ja otsustusdiagramm*

Registersiirete tasandi rikked	Otsustusdiagrammi terminaltippude rikked
RT7	$R_0, R_1, R_2, R$
RT8	$R_0, R_1, R_2, R, IN$
RT9	$R + R_0, R + R_1, R + R_2, R_0 + 1$

*Mikroprotsessorite kõrgtasandi funktsionaalsed rikked*

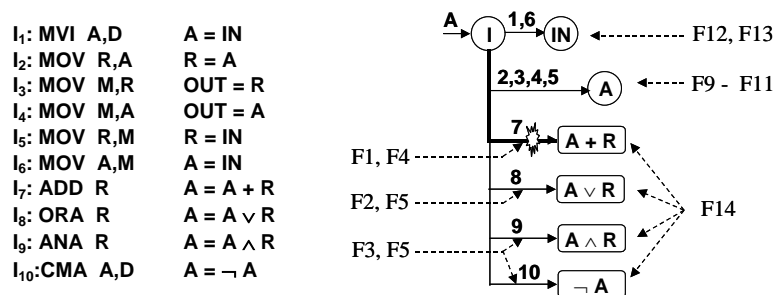
Joonisel 2-31 on toodud eespool vaadeldud hüpoteetilise mikroprotsessori otsustusdiagrammi mudel koos võimalike rikete esitamiseks vastavuses vaadeldud rikete klassidega F1 – F13.

Oletame, et testitakse käsku  $I_7$ , mille tulemusena mudelis peaks aktiveeruma kaar  $I = 7$  ja toimuma operatsioon  $A = A + R$ . Mudelis on illustreeritud järgmisi rikkeid.

Rikete klasse F1 ja F4 esindab katkestus kaarel  $I = 7$ : operatsiooni  $A = A + R$  ei toimu (D1). Rikete klassidele F2 ja F5 vastab mudelis viga, kus kaare  $I = 7$  asemel aktiveerub mingi teine kaar, näiteks  $I = 8$  ning realiseerub vale käsk  $I_8 : A = A \vee R$  (D2). Rikete klasse F3 ja F5 esindab mudelis viga, kus kaare  $I = 7$  asemel (või temaga koos) aktiveerub korrara mitu kaart ning realiseerub kombinatsioon kahest (või rohkemast) käsust  $I_9 : A = A \wedge R$  ja  $I_{10} : A = \neg A$  (D3). Kombinatsiooni tüüp (näiteks loogiline JAA või VÕI) sõltub mikroprotsessori *tehnoloogilisest realisatsioonist* [Abr 86, Abr 95].

Rikkeklassidele F6, F7 ja F8, mis kajastavad vigu mikrokäskude tasandil, antud mudelil vasteid ei leitud, kuna tegemist on käsutasandiga. Juhul kui sama graafil joonisel 2-31 kajastaks mikrooperatsioone, vastaksid riketele F6, F7 ja F8, vastavalt rikked F1, F2 ja F3.

Rikete klassidele F9, F10 ja F11 vastavad vead mälulementides või registrites, mida kajastavad mudelis terminaltipud, kus märgenditeks on mälumuutujad näiteks A. Rikete klassidele F12 ja F13 vastavad vead andmete edastamisel näiteks sisendmagistraali IN kaudu, aga ka registritest (näiteks tipp A). Rikete klassile F14 vastavad vead andmete töötlemisel (tipud  $A + R$ ,  $A \vee R$ ,  $A \wedge R$ ,  $\neg A$ ).



Joonis 2-31. Mikroprotsessori kõrgtasandi rikete modelleerimine

Otsustusdiagrammi tipurikke mudel võimaldab katta erinevaid rikete mudeleid süsteemi erinevatel abstraktsetel esitustasanditel. OD tipurikete füüsikaline sisu tuleneb sellest, millist konkreetset osa digitaalsüsteemist antud tipp modelleerib, milline sisuline tähendus on sellel tipul. Sõltuvalt digitaalsüsteemi või selle osa kirjeldamise detailsuse astmest otsustusdiagrammis sõltub ka tipurikke mudeli interpretatsioon ja detailsuse aste.

Näiteks SSBOD-mudelis sisaldab tipu  $m$  rikkemudel kaks konstantset riket  $\{x(m) \equiv 0 \text{ ja } x(m) \equiv 1\}$ , millele vastab üks ekvivalentsete rikete klass SSBOD-ga modelleeritavas ventiiltaseme skeemis. Seega katab SSBOD-mudel täielikult klassikalise ventiilskeemidele vastava konstantse rikete mudeli.

Toetudes eelnevale, võime öelda, et OD tipurikke mudel on tegelikult klassikalise ventiiltaseme konstantse rikete mudeli üldistus digitaalsüsteemide kõrgtasandil esitatud kirjelduste jaoks. Kui konstantse rikete

*model* toetub Boole'i algebrale ja on seetõttu kasutatav ainult loogikatasandil, siis OD tipurikke *model* on kasutatav nii loogikatasandil kui ka digitaalsüsteemide kõrgematel tasanditel.

## 2.5. Kokkuvõtteks

Käsitlesime defektide ja rikete modelleerimise meetodeid digitaalsüsteemides. Kuna funktsionaalne lähenemisviis – testida süsteemi kõikvõimalikke funktsioone – osutub reaalse objektide keerukuse tõttu ebapraktiliseks, siis võimaldab struktuurne ricketel põhinev lähenemisviis saavutada küllaltki häid ja rahuldavaid tulemusi. Struktuurse lähenemisviisi puhul on diagnoosiobjektiks rike, mitte funktsioon. Testi genereerimine tähendab leida iga võimaliku rikke jaoks seda avastav testvektor või -jada.

Selgus, et kõigi võimalike rikete loendamine ja testimine on praktiliselt võimatu. Küll aga osutub võimalikuks testimisülesannet lihtsustada, viies kõigepealt läbi rikete analüüsi, et välja valida ainult need rikked, mille testimine on oluline. Oluliste rikete väljaselgitamiseks on vaja teada rikete tüüpe, osata klassifitseerida rikkeid ning tunda rikete omadusi nagu liiasust, ekvivalentsust, maskeeruvust.

Keerukuse vastu võitlemiseks võib testide genereerimisel kasutada ka kõrgtasandi rikete mudeleid. Aga viimaste puhul pole alati selge, kui hästi need esindavad reaalseid füüsilisi defekte. Seetõttu, ehkki nad võimaldavad testide sünteesi oluliselt kiirendada, tuleb testide kvaliteedi kindlakstegemiseks simuleerida saadud teste siiski ka madala tasandi rikete (defektide) hulgal, mistõttu kahaneb taas testide sünteesi efektiivsus.

Kõige tähtsamaks meetodiliseks võtteks võitluses keerukusega rikete käsitlemisel on nende hierarhiline modelleerimine funktsionaalse rikke-mudeli abil. Nimetatud *model* võimaldab kujutada reaalseid defekte füüsilikatasandilt loogikatasandile ja ka suvalise tasandi rikkeid järgmisele kõrgemale tasandile. Funktsionaalne rikkemudel on üldiseks aluseks hierarhilistele meetoditele nii testide sünteesil kui ka analüüsil.

## Kirjandus

- [Aba 89] M. Abadir et al. Logic Design Verification via Test Pattern Generation. *IEEE Trans. Comput-Aided Des. Integrated Circuits Syst.*, Vol. CAD-7, No 1, pp. 138-149.
- [Abr 86] J.A. Abraham. Fault modeling in VLSI. *VLSI testing. North Holland* 1986, pp.1-27.
- [Abr 95] M. Abramovici, M.A. Breuer, A.D. Friedman. Digital Systems Testing & Testable Designs. *Computer Science Press*, 1995, 653 p.
- [Ake 59] S.B. Akers. On a Theory of Boolean Functions. *J. SIAM*, 7 (1959) H.4.
- [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.
- [Ben 98] B. Bennetts: Failures & Fault Models, Design for Testability Course Material. *Bennetts Associates*, February 1998.
- [Boc 75] D. Bochmann. Einführung in die Strukturelle Automaten Theory. *VEB Verlag Technik Berlin*, 1975, 234 S.
- [Boc 89] D. Bochmann, R. Ubar. Fehler in Automaten. *VEB Verlag Technik Berlin*, 1989, 216 S.
- [Bry 86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35 (8): 677-691, 1986.
- [Bus 00] M.L. Bushnell, V.D. Agrawal. Essentials of Electronic testing. *Kluwer Acad. Publishers*, 2000, 690 p.
- [Dre 98] R. Drechsler, B. Becker. Binary Decision Diagrams. *Kluwer Academic Publishers*, 1998, 200 p.

- [Gho 91] S. Ghosh, T.J. Chakraborty. On Behavior Fault Modelling for Digital Designs. Kluwer Academic Publishers. *J. of Electronic testing: Theory and Applications*, 2, 1991, pp. 135-151.
- [Gho 92] A. Ghosh et. al. Sequential Logic Testing and Verification. *Kluwer Academic Publishers*, 1992, 214 p.
- [Goo 91] A.J. van de Goor. Testing Semiconductor Memories. Theory and Practice. *J. Wiley & Sons*, 1991. 512 p.
- [Gri 99] R.P. Grimaldi. Discrete and Combinatorial Mathematics. *Addison-Wesley Longman, Inc.*, 1999.
- [Gup 85] A.K. Gupta, J.R. Armstrong. Functional Fault modelling and Simulation for VLSI Devices. *22<sup>nd</sup> Design Automation Conference*, 1985, pp. 720-726.
- [Hur 98] S.L. Hurst. VLSI Testing. Digital and Mixed Analog/Digital Techniques. *The Institution of Electrical Engineers*, Exeter, 1998, 532 p.
- [Jha 90] N.K. Jha, S. Kundu. Testing and Reliable Design of CMOS Circuits. *Kluwer Academic Publishers*, 1990, 231 p.
- [Jha 03] N. Lha, S. Gupta. Testing of Digital Systems. *Cambridge University Press*, 2003, 1000 p.
- [Kun 97] W. Kunz, D. Stoffel. Reasoning in Boolean Networks. Logic Synthesis and Verification Using Testing Techniques. *Kluwer Academic Publishers*, 1997, 230 p.
- [Kär 96] R. Kärger. Diagnose von Computern. *B.G. Teubner, Stuttgart*, 1996, 416 S.
- [Lal 97] P.K. Lala. Digital Circuit Testing and Testability. *Academic Press*, 1997, 199 p.
- [Lee 59] C.Y. Lee. Representation of Switching Circuits by Binary Decision Programs. *The Bell System Technical Journal*, July 1959, pp. 985-999.
- [Lev98] R. Leveugle, R. Ubar. Synthesis of Decision Diagrams from Clock-Driven Multi-Process VHDL Descriptions for Test Generation. *Proc. of the 5<sup>th</sup> International Conference on Mixed Design of Integrated Circuits and Systems*. Lodz (Poland), June 18-20, 1998, pp. 353-358.
- [Min 96] S. Minato. Binary Decision Diagrams and Applications for VLSI CAD. *Kluwer Academic Publishers*, 1996, 141 p.

- [Mol 04] P. Molitor, J. Mohnke. Equivalence Checking of Digital Circuits. Fundamentals, Principles, Methods. *Kluwer Academic Publishers*, 2004, 262 p.
- [Mou 00] S. Mourad, Y. Zorian. Principles of Testing Electronic Systems. *J. Wiley & Sons, Inc.*\_New York, 2000, 420 p.
- [Nic 03] N. Nicolici, B.M. Al-Hashimi. Power-Constrained Testing of VLSI Circuits. *Kluwer Academic Publishers*, 2003, 178 p.
- [Nov 05] O. Novak, E. Gramatova, R. Ubar. Handbook of Electronic Testing. CTU Printhouse, Prague, 2005, 400 p.
- [Pla 80] M. Plakk, R. Ubar. Digital Circuit Test Design using the Alternative Graph Model. Automation and Remote Control, Vol.41, No 5, part 2, Nov. 1980, *Plenum Publishing Corporation*, USA, pp. 714-722
- [Pra 95] D.K. Pradhan. Fault-Tolerant Computer System Design. *Prentice Hall PTR*, New Jersey, 1995, 550 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.
- [Sac 98] M. Sachdev. Defect Oriented Testing for CMOS Analog and Digital Circuits. *Kluwer Academic Publishers*, 1998, 308 p.
- [Sas 96] T. Sasao, M. Fujita. Representations of Discrete Functions. *Kluwer Academic Publishers*, 1996, 331 p.
- [Sel 83] A.V. Seleznjov, B.T. Dobritsa, R.R. Ubar. Projektirovanije Avt. Sistem Kontrolja Bortovovo Oborudovanija Letatel'nyh Apparatov. *Mashinostrojenije*, Moskva, 1983, 224 p.
- [She 88] L. Shen, S.Y.H. Su. A Functional Testing Method for Microprocessors. *IEEE Transactions on Computers*, Vol.37, No. 10, 1988, pp.1288-1293.
- [Tha 71] A. Thaise. Boolean Differential Calculus. *Philips Res. Repts.*, vol. 26 (1971), pp.229-246.
- [Tha 77] S.M. Thatte, J.A. Abraham. Testing of Semiconductor Random Access Memories. *Proc. of 7<sup>th</sup> Int. Symp. on Fault-Tolerant Computing*, Los Angeles, June 1977, pp. 81-87.
- [Tha 80] S.M. Thatte, J.A. Abraham. Test Generation for Microprocessors, *IEEE Trans. On Computers*, Vol. C-29, No. 6, pp.429-441, June 1980.



- [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 81] R. Ubar. Vektorielle Alternative Graphen und Fehlerdiagnose für digitale Systeme. *Nachrichtentechnik/Elektronik*, (31) 1981, H.1, S.25-29.
- [Uba 83] R. Ubar. Test Generation for Digital Systems on the Vector Alternative Graph Model. *Proc. of the 13th Annual Int. Symposium on Fault Tolerant Computing*, Milano, Italy, 1983, pp. 374-377.
- [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 pp. 141-157, 1998.

## Lühendid

ALU	- <i>Arithmetic Logic Unit</i> 128
BOD	- binaarne otsustusdiagramm 14
CMOS	- <i>Complementary Metal Oxide Semiconductor</i> 87
DMUX	- demultiplekser 128
DNK	- disjunktiivne normaalkuju 18
D&T	- disain ja test 14
ITS	- isetestiv süsteem 14
KR	- konstantrike 107
KR-test	- konstantrikke test 107
MUX	- multiplekser 128
OD	- otsustusdiagramm 14
SAF	- <i>Stuck-at Fault</i> (konstantne rike) 94
SAF0	- <i>Stuck-at-0 Fault</i> (konstantne rike 0) 85
SAF1	- <i>Stuck-at-1 Fault</i> (konstantne rike 1) 85
SSBOD	- struktuurselt sünteesitud binaarsed otsustusdiagrammid 39
TD	- testikõlblik disain 14
VHDL	- <i>Very High Scaled Integrated Circuits Description Language</i> 57
V <sub>DD</sub>	- <i>Voltage Drain Drain</i> 110
VSS	- <i>Voltage for Substrate &amp; Sources</i> 110

## Indeks

### A

ahelreegel 26  
ajatingimus 124  
aktiveerimistingimus 111  
alternatiivsed graafid 37  
andmeallikas 121, 124  
andmesiire 124  
andmestruktuur 39  
andmete edastamise mehhanism 123  
andmete säilitamise mehhanism 122  
andmete töötlemise mehhanism 123  
andmetöötlusplokk 128  
arvutiteooria 15

### B

binaarne otsustusdiagramm (BOD) 14, 36  
    BOD tipurike 97  
    BOD topoloogiline analüüs 36  
    BOD superpositsioon 39, 41, 42  
    elementaarne BOD 42  
    struktuurselt sünteesitud (SSBOD) 39  
    testi genereerimine BOD tipule 46  
binaarne otsustusprogramm 37  
binaarne programm 39  
Boole'i  
    diferentsiaal 30  
    diferentsiaalalgebra 14  
    diferentsiaalaparaat 16  
    diferentsiaalvõrrand 95, 104, 116

- funktsioon 17
- muutuja 50
- Boole'i funktsioon (BF) 17
  - disjunktiiivne normaalkuju 18
  - osadiferentsiaal 30
  - täisdiferentsiaal 30
- Boole'i tuletis 17
  - ajafunktsiooni tuletis 27
  - kordne tuletis 23
  - liitfunktsiooni tuletis 25
  - osatuletis 17, 19, 86
  - osatuletise omadused 19
  - tuletis
  - vektortuletis 21

## C

- Carnaugh' kaart 22
- CMOS-tehnoloogia 87, 90, 104

## D

- defekt 81
  - funktsionaalselt liiane 112
  - füüsikaline 14, 80, 108
  - katastroofiline 92
  - ühendusahelate 114
- defekti
  - aktiveerimine 108, 111
  - kirjeldamine 108
  - kujutamine 108
- defektimuutuja 115
- defektide tabel 111, 112
- defektipõhine modelleerimine 12
- defektipõhine testide genereerimine 113
- Descartes'i korrutis 49
- diagnoos 33
- diagnoosi resolutsioon 97
- diagnostika 6
  - kitsamas tähenduses 8
  - laiemas tähenduses 8
  - otseülesanne 34
  - probleemid 13

- pöördülesanne 34
- eksperiment 31
- eksperimendi mudel 35
- eksperimendi tulemus 32
- üldvõrrand 34
- diagnostiline analüüs 47, 113
- diagnostiline informatsioon 113
- diferentsiaalioperaator 33
- digitaalsüsteem 13, 49
- disain 14
  - disain ja test 14
  - testikõlblik disain 14
- diskreetne matemaatika 15
- dominantsusreegel 100
- dünaamiline mälulement 110

## **E**

- ekvivalentskeem 105, 116
- ekvivalentsusklass 97
- ekvivalentsusreegel 100

## **F**

- funktsioon
  - ajafunktsioon 27
  - integreeritud funktsioon 56
  - liitfunktsioon 25
  - siirdefunktsioon 27
  - vektorfunktsioon 56, 70
  - väljundfunktsioon 27
  - üldistatud parameetiline funktsioon 105, 114, 116
- funktsiooni
  - differentsiaal 30
  - muutuja 51
  - muutumispiirkond 49
  - määramispiirkond 49
  - tuletis 16
- funktsionaalne ekvivalentsus 97
- füüsikaliste defektide kujutamine 108

## **G**

- graaf

harugraaf 72  
orienteeritud, tsükliteta 37, 49  
vektorgraaf 70

#### graafi

aadresstipp 70  
algtipp (juur) 40  
kaar (0-kaar, 1-kaar) 37  
mitteterminaalne tipp 51  
terminaltipp 37, 49  
0-terminal, 1-terminal 37  
tee, aktiveeritud tee 37  
tipp 37, 49  
tipumuutuja 37, 49  
tipu järglane (naaber) 37

#### **H**

hierarhiline testimise kontseptsioon 118, 123  
Huffmanni mudel 27, 55

#### **I**

illegaalne olek 57  
invariantsus 33  
isetestimine 8

#### **J**

JK-triger 28  
JK-trigeri Boole'i osatuletised 28  
juhtautomaat 55  
juhtumuutuja 50  
juhtsõna 127  
järjestikskeem 11

#### **K**

keelatud olek 57  
kolmjalg 48  
kombinatsioonskeem 11  
komponentide teek 113  
komponendi lähiümbrus 114  
konkatenatsioon 56  
käskude dekodeerimine 122  
käsuformaad (käsuvorming) 50, 77, 120

käsuformaadi väli 50  
käsusüsteemi tasand 77  
käsutsükkel 77

## L

lauseanalüüs 124  
liitfunktsioon 25  
liitfunktsiooni osatuletis 25  
liitkonstant 57  
loendur 13  
loogikatingimus (lipp) 50, 124  
lokaalne testvektor 127  
lokaalne sisendvektor 46  
lubatud rikete kordsus 101  
lubatud rikete klass 107  
lõplik automaat 130  
läbipaistvuse tingimus 66  
läte (source) 89  
lävipinge 93, 93  
lühisriike 10, 87, 114  
    JAA-tüüpi lühis (wired-AND short) 88  
    kombinatoorne lühisriike 89, 89  
    0-domineerimisega lühis 88  
    sekventsiaalne lühisriike 88, 88, 88  
    ventiilide vaheline (inter-gate short) 88  
    ventiilisisene (intra-gate short) 88  
    VÕI-tüüpi lühis (wired-OR short) 88  
    1-domineerimisega lühis 88

## M

meetod  
    hierarhiline 123  
    “jaga ja valitse” 15  
    rikete defineerimise meetod 117  
    superpositsiooni meetod 39, 42, 52  
mehhanism 121  
mikrokäsk 63, 122  
mikrooperatsioon (mikrotehe) 68, 122  
mikroprogramm 68  
modelleerimine  
    defektipõhine 12, 82

- diagnostiline 15, 117
- hierarhiline 12
- protseduurne 77
- rikete modelleerimine 80, 104
- Moore'i automaat 56
- udel
  - funktsionaalne 77
  - funktsionaalne rikkemudel 83, 104, 108, 113, 116, 124
  - lõpliku automaadi mudel 130
  - mudeli häirimine (perturbation) 119
  - struktuurne 77
  - protseduurne 77
  - rikete mudel 80, 82
- must kast 39
- muutuja
  - aadressimuutuja 70
  - Boole'i muutuja (binaarmuutuja) 50
  - defektimuutuja 115
  - juhtmuutuja 50
  - käsumuutuja 75
  - liitmuutuja 56
  - seademuutuja 57
  - tipumuutuja 37, 49
  - tsüklimuutuja 77
  - täisarvuline 50
- mäludekooder 123
- mälumassiiv 122
- märgend 124
- määramatu signaal 109

**N**

- neel (drain) 89

**O**

- olekute ning siirete tabel 130
- operatsioon (tehe) 124
- operatsiooniautomaat 52
- operatsioonielement 52
- operatsiooni kood 50, 77
- osaline käskude täitmine 122
- otsustusdiagramm 49



## **P**

parasiitmahtuvus 93  
pinge jagunemine 109  
protessor 50  
puukujuline makro 98  
põhimik (substrate) 89

## **R**

registersiirete tasand 63  
registrite dekodeerimine 121  
riike 9, 82  
    adresseerimise riike 120, 121  
    katastrofaalne 92  
    katkestused 91  
    katkestused transistorskeemis 110  
    kombinatoorsed lühisrikked 89, 89  
    kordsed rikked 11, 23, 87, 101  
    loogiline 11  
    lühisriike 10, 87, 114  
    lühised transistorskeemides 89, 108  
    mitmekordsed rikked 11, 23, 87  
    mittetestitav 94  
    liiane 94  
    permanentne (püsiv) 85  
    sekventsiaalsed lühisrikked 88, 88, 88  
    signaalitee viiteriike 93  
    struktuursed rikked 45, 83  
    takistuslikud lühisrikked 92  
    testitav riike 94  
    ventiili viiteriike 93  
    viiterikked 92  
rikete  
    avastamine 99  
    defineerimise meetod 117  
    diagnoos 35  
    dominantsus 99  
    ekvivalentsus 97  
    eristatavus 97  
    funktsionaalne ekvivalentsus 97  
    hierarhiline esitus 117  
    kollaps 97, 100

- kollapsi reeglid 100
- kollapsi üldreegel 101
- liiasus 94
- lokaliseerimine 8
- maskeerumine 22, 101
- modelleerimine 80, 104
- nimistu 119
- omadused 94
- ringmaskeeruvus 103
- simulaator 9
- simuleerimine 35
- sõnastik 9
- tõenäosus 12
- vektor 35
- rikke levitamine 82
- rikkemudel
  - ajutise (*transient*) rikke mudel 85
  - funktsionaalne rikkemudel 83, 104, 108, 113, 116, 124, 124
  - kaudne rikkemudel 83, 120
  - konstantse rikke mudel 80, 85, 94, 132
  - kõrgtasandi rikkemudelid 119
  - kõrgtasandi rikkemudel otsustusdiagrammidel 126
  - mikroprotsessori funktsionaalne rikete mudel 120, 131
  - mäluseadmete rikete mudelid 122
  - otsene rikkemudel 83, 120
  - otsustusdiagrammi tipurikke mudel 132
  - registersiirete rikkemudel 124, 128
  - rikkemudelite vastavus 128
  - universaalne rikkemudel 126
  - universaalse rikkemudeli definitsioon 127
  - üksiku konstantse rikke mudel 85

## S

- signaalitee 44, 97, 98
- sihtregister 124
- siirdeaadress 127
- simuleerimine
  - rikete simuleerimine 35
  - sümbolsimuleerimine 49, 57, 72
  - sümbolsimuleerimise puu 51
  - tsükli põhine simuleerimine 63

- sisetakistus 109
- skeem
  - järjestikskeem 11, 27
  - kombinatsiooniskeem 11
  - liiane 94
  - loogikaskeem 12
  - makro 41, 44
  - puukujuline alamskeem 41, 44, 98
  - transistorskeem 12, 108, 108
- summaator 10
  - järjestikülekandega 11
- superpositsioon 26
- sümbol  $D$  30
- sümboolne manipuleerimine 43
- sümbolsimuleerimine 49, 57, 72
- sümbolsimuleerimise puu 51
- sümbolväärtus 72
- süsteemi
  - protseduurne mudel 49
  - analüüs 12
  - usaldusväärsus 10
  - modelleerimine 12
  - registersiirete tasand 63
  - sümbolsimuleerimine 49
  - tööviis (töörežiim) 50
  - ventiiltasand 39

## **T**

- tehe (operatsioon) 124
- tehniline diagnostika 6, 8
- tehniline süsteem 8
- tehnoloogia 122, 128
- tehnoloogiline realisatsioon 131
- test 8, 9, 14
  - defektitest 107
  - KR-test 107
  - testi kvaliteet 9, 10, 80, 107
  - testide analüüs 9
  - testide generaator 9
  - testide genereerimine 35, 45
  - testide süntees 9

- täielik rikkeid lokaliseeriv test 97
- täielik test 103
- testeksperiment 8
- tester 8
- testimine 8, 10, 14
  - funktsionaalne testimine 11
  - on-line* testimine 9
  - riketepõhine testimine 11
  - struktuurne testimine 11
  - testimisaeg 11
  - testimise keerukus 11
  - testimise risk 9
  - testimise tööriistad 8
- testjada 94, 116
- testprogramm 123
- testvektor 8
- tipuavaldis 127
- transparentsus 66
- töörežiim (tööviis) 127

## U

- ubikvism 6
- universaalne liides 104

## V

- viga 81, 121
  - käitumisviga 122
  - veasignaali 22, 112
  - vigade maskeerumine 22
- vektorotsustusdiagramm 56
- ventiilide meri 13
- ventiilitasand 39
- VHDL- kirjeldus 57
- VHDL- protsess 57

## Ü

- üldistatud parameetiline funktsioon 105, 114, 116