Eesti Tehnoloogiaagentuuri projekti

“Mikroprotsessorite isetestimise kontseptsioon ja projekteerimisvahendid (MIKROTEST)”

1. tegevusaasta aruanne

OÜ Artec Design Group
Tallinna Tehnikaülikool

Tallinn
Mai 2002

Projekti põhitäitjad

Jüri Põldre, Artec Design Group, peaspetsialist

Raimund Ubar, TTÜ professor

Jaan Raik, TTÜ vanemteadur

Artur Jutman, TTÜ teadur

Eero Ivask, TTÜ teadur

Annotatsioon

Käesolevas dokumendis on esitatud Eesti Tehnoloogiaagentuuri projekti “Mikroprotsessorite isetestimise kontseptsioon ja projekteerimisvahendid” (MIKROTEST) esimese aasta aruanne. Projekt MIKROTEST käivitus 1. juunil 2001 ning tema kestus on kaks aastat. Rakendusuuringu sisuks on uut tüüpi tarkvara väljatöötamine mikroprotsessor-süsteemide rikete simuleerimiseks, isetestimisarhitektuuride projekteerimiseks ning selle tarkvara rakendamine konkreetse uut tüüpi isetestiva mikroprotsessorsüsteemi projekteerimisel.

Projekti eesmärgiks on

- uut tüüpi spetsiaaltarkvara väljatöötamine digitaalsüsteemide testimise ja diagnostika automatiseerimiseks,
- metoodika loomine digitaalsüsteemide isetestimiskontseptsiooni projekteerimiseks ja
- uut tüüpi kõrgekvaliteedilise isetestiva mikroprotsessorsüsteemi väljatöötamine ning projekteerimine.

Projektis osalevateks partneriteks on Tallinna Tehnikaülikooli Arvutitehnika instituut (TTÜ) ja OÜ Artec Design Group (ADG), kus esimese rolliks on uute diagnostikavahendite loomine ja teise rolliks uut tüüpi krüptograafilise võrguprotsessori projekteerimine ning mõlema partneri ühistööna valmib uus metoodika ning kontseptsioon isetestimise ideoloogia realiseerimiseks digitaalsüsteemides.

Rakendusuuringu oodatavateks tulemusteks on:

- uus tarkvara isetestivate digitaalsüsteemide projekteerimiseks, mis koosneb järgmistest komponentidest: hierarhiline rikete simulaator, hierarhiline testide generaator, isetestimis-arhitektuuride kvaliteedi analüüsi- ja optimeerimisprogrammid
- metoodika väljatöötamine isetestivate mikroprotsessorite projekteerimiseks ja selle rakendamine reaalsel tootel
- uut tüüpi kõrgekvaliteedilise isetestiva võrguprotsessori väljatöötamine ja projekteerimine

Sisukord

1 Üldist projektist

1.1 Projekti sisu ja oodatav lõpptulemus

Käesolevas dokumendis on esitatud Eesti Tehnoloogiaagentuuri projekti “Mikroprotsessorite isetestimise kontseptsioon ja projekteerimisvahendid” (MIKROTEST) esimese aasta aruanne. Projekt MIKROTEST käivitus 1. juunil 2001 ning tema kestus on kaks aastat. Rakendusuuringu sisuks on uut tüüpi tarkvara väljatöötamine mikroprotsessor-süsteemide rikete simuleerimiseks, isetestimisarhitektuuride projekteerimiseks ning selle tarkvara rakendamine konkreetse uut tüüpi isetestiva mikroprotsessorsüsteemi projekteerimisel.

Projekti eesmärgiks on

- uut tüüpi spetsiaaltarkvara väljatöötamine digitaalsüsteemide testimise ja diagnostika automatiseerimiseks,
- metoodika loomine digitaalsüsteemide isetestimiskontseptsiooni projekteerimiseks ja
- uut tüüpi kõrgekvaliteedilise isetestiva mikroprotsessorsüsteemi väljatöötamine ning projekteerimine.

Projektis osalevateks partneriteks on Tallinna Tehnikaülikooli Arvutitehnika instituut (TTÜ) ja OÜ Artec Design Group (ADG), kus esimese rolliks on uute diagnostikavahendite loomine ja teise rolliks uut tüüpi krüptograafilise võrguprotsessori projekteerimine ning mõlema partneri ühistööna valmib uus metoodika ning kontseptsioon isetestimise ideoloogia realiseerimiseks digitaalsüsteemides.

Rakendusuuringu oodatavateks tulemusteks on:

- uus tarkvara isetestivate digitaalsüsteemide projekteerimiseks, mis koosneb järgmistest komponentidest: hierarhiline rikete simulaator, hierarhiline testide generaator, isetestimis-arhitektuuride kvaliteedi analüüsi- ja optimeerimisprogrammid
- metoodika väljatöötamine isetestivate mikroprotsessorite projekteerimiseks ja selle rakendamine reaalsel tootel
- uut tüüpi kõrgekvaliteedilise isetestiva võrguprotsessori väljatöötamine ja projekteerimine

Projekti tulemuste oodatav majanduslik efekt väljendub järgnevas:

- isetestimiskontseptsiooni väljatöötamine ja rakendamine võimaldab vältida välistestri soetamist ning kasutamist, milliste hind tänapäeval ulatub suurusjärku 1 miljon dollarit
- testprogrammide projekteerimise automatiseerimine võimaldab lahendada ülesandeid, mis traditsiooniliselt ühe tüüpilise mikroprotsessori puhul nõudis terve aasta, mõne nädalaga
- väljatöötatava diagnostikatarkvara kasutamine võimaldab automatiseerida testprogrammide genereerimist ja kogu diagnostikakontseptsiooni väljatöötamist, mille tulemusena õnnestub kiirendada testimisega seotud projekteerimise kogutsüklit kuni 50%

1.2 Esimese projektiaasta tulemuste aruande ülesehitus

Käesolev aastaaruanne on üles ehitatud vastavalt esimese aasta peamistele tulemustele, milledeks on planeeritud:

- Krüptograafilise võrguprotsessori prototüüpriistvara loomine programmeeritavate loogikamaatriksite platvormil (vastutav täitja ADG, esitatud peatükis 2).
- Prototüüpkiibi struktuurianalüüs ja isetestimise metodoloogia väljatöötamine (täitja TTÜ, peatükk 3.1).
- Diagnostikavahendite skeemiliidesed projekteerimistarkvaraga (TTÜ, peatükk 3.2).
- Diagnostikaalgoritmide väljatöötamine (TTÜ, peatükk 3.3).

Aruande peatükis 4 on esitatud kokkuvõte. Rakendusuuringu täpne ülesannete nimistu, projekti ajagraafik ning vahetulemuste (kontrollpunktide) kirjeldus vastavalt tööplaanile on esitatud lisas 1. Lisades on ära toodud täiendav tehniline informatsioon.

2 Krüptograafilise võrguprotsessori prototüüpriistvara disain

Peatükk hõlmab projekti raames OÜ Artec Design Group (ADG) poolt läbi viidud ülesandeid krüptograafilise võrguprotsessori BITSie ning tuuma VPNow väljatöötamisel. Võrguprotsessor BITSie on projekteeritud korduvkasutust ja edasiarendust silmaspidades. Turustama hakatakse nii tuuma VPNow tarkvaralise kirjeldusena (soft core) kui ka mikroskeemi BITSie, mida võib vaadelda sisuliselt VPNow ühe realisatsioonina.

Vastavalt tööplaanis esitatud ajagraafikule (vt. Lisa 1) koosnes ADG töö esimesel projekti aastal kolmest ülesandest, mille põhjal on ka käesolev peatükk alampeatükkideks jaotatud.

2.1 Prototüüpkiibi spetsifikatsioon (Ülesanne 1)

2.1.1 Sissejuhatus

Rakendusuuringu raames projekteeritav kiip BITSie on krüptograafiline võrguprotsessor, mis pakub kiiret ja turvalist sidet üle arvutivõrgu. Ta võimaldab TCP/IP protokolli krüpteerimist (TCP/IP on võrguside üldtunnustatud standardprotokoll). BITSie saab ühendada personaalarvuti siinile. Nimetatud lähenemise eeliseks on, et kõik krüpteerimisfunktsioonid viiakse läbi riistvaras ning arvuti mikroprotsessor saab täita samaaegselt muid operatsioone.

Projekteeritavas kiibis on realiseeritud virtuaalne privaatne võrk (Virtual Private Networking ehk VPN). See tugineb IPsec standardile, mis on välja antud organisatsiooni IETF poolt. (IETFi kompetentsi kuulub kõikide internetti puudutavate standardite väljatöötamine). IPsec on ka kohustuslik osa uuest interneti protokolli standardist - ipv6. Järgneva paari aasta jooksul peaks ipv6 asendama järk-järgult seni kehtinud ipv4 standardi ning seda toetavad enamus uutest aplikatsioonidest.

Üheks BITSie poolt pakutavaks uuenduseks on, et ta võimaldab ära kasutada vana võrgu infrastruktuuri üleminekul ipv4 standardilt ipv6le, mis peaks aset leidma järgneva aastakümne jooksul. Kiibi abil saab “tunneldada” uut protokolli eksisteerivate vanade võrgukanalite kaudu. Seega saavad uue interneti protokolli (IP) versiooni võrgud pidada sidet (moodustada virtuaalseid privaatseid võrke), kasutades olemasolevat ipv4 standardile põhinevat infrastruktuuri ning uut ja vanat tüüpi võrgud võivad üleminekuajal koos eksisteerida. Tulevikus, kui infrastruktuur peaks uuenema, on tunneldamine võimalik ära keelata tarkvaralisi vahendeid kasutades. (Tunneldamine lisab vähem kui 0.1 % edastatavate pakettide suurusele).

Nagu eelpool öeldud on võrguprotsessor projekteeritud korduvkasutust ja edasiarendust silmaspidades. Turustama hakatakse nii tuuma VPNow tarkvaralise kirjeldusena (soft core) kui ka mikroskeemi BITSie, mida võib vaadelda sisuliselt VPNow ühe realisatsioonina.

2.1.2 Süsteemi kirjeldus

Krüptograafiline võrguprotsessor BITSie on integreeritud sisend-väljund seade, mida saab kasutada nii eraldiseisva seadmena kui ka kaasprotsessorina. Kiibil on sisemine mikroprotsessor, mis koos perifeermoodulitega moodustavad andmetöötluse tuuma. Tarkvara laetakse kiipi välisest FLASH-mälust. Sealjuures toetatakse nii EPROM (näiteks I^2C) kui ka paralleelseid FLASH seadmeid.

Eraldiseisva seadme rezhiimi puhul on kiip ise kogu andmetöötluse keskpunktiks. Sel juhul jookseb kogu süsteemi tarkvara kiibisisesel RISC mikroprotsessoril. Mälu hulka saab laiendada, kasutades välist SDRAM või SRAM muutmälu. Kõik süsteemi komponendid ühendatakse BITSie’ga kiibi poolt toetatavate standardliideste kaudu. Ülalkirjeldatud juhul on BITSie süsteemi peremeheks (ingl. k. master).

Kaasprotsessori rezhiimis töötades on kiip peamiselt orjaks (slave) süsteemisiinil, kasutades peremehe funktsiooni ainult juhul, kui valitud süsteemisiin seda võimaldab ning andmete

edastamise puhul. Süsteemitarkvara konfigureerib kiibi täitma vastavat funktsiooni ning seadet juhitakse sättides paika vastavad väärtused konfiguratsiooniregistrites.

Süsteemisiiniks võib olla üks paljudest tänapäeval laialt kasutatavatest standardsiinidest. Kõige tuntum neist on PCI siin, mis toetab ka PC kaardi liideseks olevat Cardbus'i. Ülejäänud toetatavad siinid on aadress-andmed-kontroll tüüpi. Selliseid siine kasutatakse sardmikroprotsessorites nagu Intel StrongARM perekond, Hitachi SH4, NEC VR41xx ja üldine 8/16/32-bitine siin.

Seadme põhiliseks funktsiooniks on krüpteerida võrgu andmeid ja konverteerida IP protokolle. Selleks on seadmes olemas sisemine krüptograafiline kiirendi ning seadme andmeosa on optimeeritud pakettide edastuseks. Suuremad TCP/IP võrguproduktide tootjad on kokku leppinud, et võtavad standardid IPsec ja ipv6 oma tulevikutoodete aluseks.

Seadmes kasutatavad Etherneti adapterid on registertasemel kokkusobivad DEC 21143 seadmega. Kui seadet kasutada IPsec rezhiimis jääb andmete ülekandmise mehhanism samaks. Täiendavad parameetrid selleks, et defineerida, kuidas krüpteerida andmeid moodustavad eraldi andmevoo, mida juhib vastav draiver. Selline lähenemine lubab ühendada seadme ükskõik millisesse süsteemi, millel on olemas DEC 21143 tugi. Ainsaks vajaminevaks draiveriks on IPsec draiver, mis juhib võtmehaldust ja süsteemi turvarezhiime.

2.1.3 Süsteemi initsialiseerimine

Seade väljub resetist ja skanneerib ja skanneerib vastavaid staatusbitte, et teha kindlaks kasutatavate liideste tüübid, välismälu olemasolu ja peamine süsteemi rezhiim.

RISC mikroprotsessor käivitub sisemisest koodi püsimälust. Esimene täidetav ülesanne on laiendatud isetest [COM_TEST], mis disainitakse TTÜ poolt käesoleva projekti raames. Selle testi eesmärgiks on kontrollida struktuursete rikete võimalikku esinemist seadmes. Testi esimene samm kasutab liideseid tagasisidestatud rezhiimis, et testandmeid saata ja vastuvõtta. Selle sammu poolt katmata jäänud rikked testitakse isetesti struktuuride poolt. Need muudavad seadme sisuliselt testvektorite generaatoriks/analüsaatoriks. Kui nimetatud test läbi ei lähe, siis loetakse seade rikkis olevaks ning vastav vea kood on võimalik JTAG pordist välja lugeda.

Isetest võimaldab testimise kulusid alandada, kuna testrile investeeritakse palju vähem kui välist testrit kasutades ning testimiseks ei ole tarvis ka nii pikka aega kui seda kulub välise testri puhul. Lisaks sellele on võimalik isetesti rakendada vahetult enne seadme sisselülitamist seadme normaalses töös. Rikete avastamisel saab toetav personal seadme sel juhul korrasoleva vastu vahetada vältides seega võimalikke vigu, mis vastasel korral võiksid välja tulla kriitilistes olukordades.

Järgmiseks operatsiooniks võib olla seadmega ühendatud välismälu mahu ja korrasoleku test. Selleks on olemas standardsed tuntud meetodid.

Järgmine samm sõltub BITSie peamisest süsteemirezhiimist. Kui valitud on eraldiseisva seadme rezhiim, siis seade skanneerib koodilugemisseadmete olemasolu – esiteks järjestik ning seejärel paralleelset FLASH mälu. Juhul kui leitakse vastav signatuur, loetakse kood kiibisisesest muutmälust ja käivitatakse BITSie'l. Juhul kui ei leita ühtegi välist koodisisestus seadet, siseneb BITSie järjestikliidesel terminali rezhiimi.

Kui seadme süsteemirezhiimiks on kaasprotsessori rezhiim, siis BITSie aktiveerib vastava pordi ning hakkab jälgima välist siini. Süsteem peab panema seadme joaks paika jagatud mälu parameetrid ning seejärel seadme aktiveerima. Kui BITSie on aktiveeritud, siis käivitab ta programmi jagatud mälu mälupiirkonnast. See lubab süsteemi konfiguratsiooni draiveriga paika panna. Tarkvara kasutada on seadme sisemine andme- ja käsuvahemälu ning ta jookseb suhteliselt kiiresti. Kõik andmeedastused toimuvad burst rezhiimis.

Seejärel võtab tarkvara juba seadme juhtimise täielikult üle.

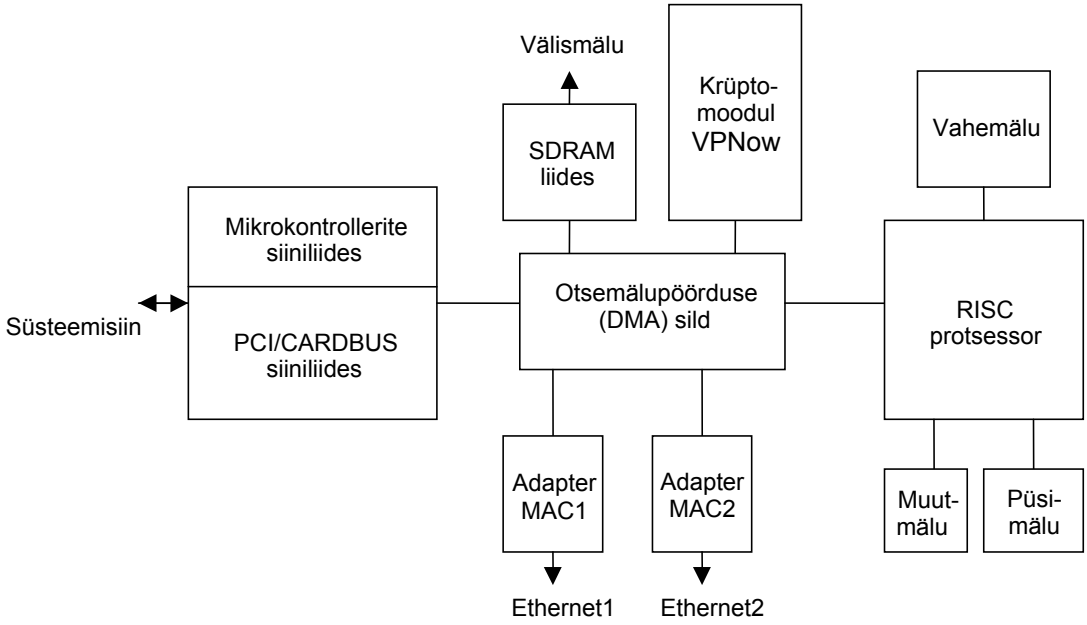
2.1.4 Kiibi üldine struktuur

Krüptograafiline võrguprotsessor BITSie koosneb järgmistest moodulitest:

- Krüptograafiline moodul VPNow
  Kiirendab krüptoarvutusi võimaldades täiskiirusel võrguühendust.
- Otsemälupöördussild
  Organiseerib andmevahetust kiibipealse muutmäluga.
- SDRAM liides
  Garanteerib mälu kättesaadavuse toote kavandatud eluajaks (kolmeks aastaks).
- 2 etherneti adapterit
  Kiibiga võib ühendada kaks erinevat võrku.
- 32-bitine RISC mikroprotsessor
  250 MHz sisseehitatud RISC mikroprotsessor kiibi juhtimiseks.
- Süsteemi siiniliidesed
  Võimalik ühendada mikrokontrollerite ja PCI siinile. Mikrokontrollereid kasutatakse võrguaplikatsioonides nende soodsast hinna/efektiivsuse suhtest tulenevalt üha enam.
- Täiendavad moodulid.

Ülalnimetatud moodulitest on etherneti adapter, PCI siiniliides ja mikroprotsessor teistelt tootjatelt ostetud tuumad. Etherneti adapter ning siiniliides on firmalt InSilicon Corporation ja mikroprotsessor firmalt ARC Iic.

Krüptomoodul VPNow, SDRAM liides ja otsemälupöördussild on projekteeritud ADG poolt. Süsteem koosneb ka mitmetest teistest, väiksematest moodulitest, mille koguosa kiibist on vähem kui 5 %.



Joonis 2.1. Krüptograafiline võrguprotsessor BITSie arhitektuur

2.2 VPNow verifitseerimiskeskkonna spetsifikatsioon (Ülesanne 2)

Käesolev alampeatükk kirjeldab trükkplaati, mis välja töötatud ADG poolt projekteeritud tuumal VPNow põhineva prototüüpkiibi arenduseks ja silumiseks. Nimetatud trükkplaadil on olemas kõik komponendid, mis on vajalikud VPNow kiibi emuleerimiseks prototüüpriistvaral reaalse tootega ligilähedasel töösagedusel. Trükkplaat istub nii PCI kui ISA slotti ning baseerub Xilinxi programmeeritaval loogikamaatriksil.

2.2.1 Sissejuhatus

VPNow prototüüp on keeruline süsteem-kiibil tehnoloogial põhinev mikroskeem. Projekteerimise käigus kirjeldatakse skeemi funktsionaalsust C-taseme mudelist riistvara realisatsioonini välja. Projekteerimise lõppfaasis, kus enamus skeemist on kirjeldatud juba loogikalülituste tasemel, ei ole enam praktiliselt mõistlik sellisel mudelil töötada, sest reaalajas simuleerimine osutub ülejõukäivaks ülesandeks.

Loomulik lahendus probleemist ülesaamiseks on kiirema simulaatori kasutamine. Selleks otstarbeks saab kasutada näiteks programmeeritavaid loogikamaatrikseid. Loogikamaatriksid konfigureeritakse vastavalt projekteeritava kiibi loogikataseme struktuurile ning neil viiakse läbi kiibi prototüüpimine. Programmeeritavate loogikamaatriksite kasutamine võimaldab suurt paindlikkust, kuid võrreldes ränil toodetud kiibiga on nende töökiirus väiksem. Antud projekteerimisetapis on loogikamaatriksite rakendamine siiski sobivaks kompromissiks.

Viimase aastakümne jooksul on programmeeritavad loogikamaatriksid teinud läbi tohutu arenguhüppe sajakonna loogikalülitusega kiipidest kuni tänapäeva miljoni lüliga mikroskeemideni välja. Sellise suurusega kiibid lubavad juba enamuse skeemi funktsionaalsusest ühte maatriksisse suruda. Välisteks komponentideks jäävad seejures digitaal-analoog ja toitenivoo muundurid. Samuti on mõistlik kasutada välist mälu, sest see jätab süsteemi jaoks loogikamaatriksisse rohkem vabu ressursse.

2.2.2 Verifitseerimisplatvormi arhitektuur

Trükkplaadi arhitektuuri haldab ADG poolt välja töötatud ühekiibi PC machZ ning silumisliidesed. Vajaduse korral saab trükkplaadile lisada täiendavaid komponente. Järgnevas on ära toodud nõuded, mis on jaotatud funktsiooni põhjal gruppidesse.

Vpnow tuum:

1. Loogikamaatriks süsteemi emuleerimiseks
   Virtex-II tüüpi FPGA 16 sisemise DLL mooduliga takteerimiseks

2. FPGA konfiguratsiooni PROM
   Säilitab FPGA konfiguratsiooniinfot. Kasutatakse disaini lõppfaasis, esialgne konfiguratsioon loetakse JTAG/serial programmeerimisliidese kaudu.

Vpnow välisliidesed:

1. 10/100 PHY magnetics'iga ja Etherneti soketid
   Vpnow kiibil on MII pordid. Päised Vpnow MII portide ja PHY moodulite vahel. PHY moodulid teisendavad Etherneti füüsikalised signaalid MII siinile.
2. SPI järjestik-PROM Etherneti addressi ja PCI väärtuste salvestamiseks
   Väike mälu (256 baiti) konfiguratsiooni väärtuste salvestamiseks
3. RS232 koos nivooteisendajatega silumiskonsooli jaoks
   Vpnow on silumiseks samuti olemas sisemine konsool.
4. PCI slott ja konnektor
   Konnector hosti süsteemiga. Kaardil asuv konnektor ja slott.
5. CardBus konnektor (pin header)

CardBusi konnektorit kasutatakse selleks, et kasutada seadet PC (LapTop) kaardina. Seega peavad konnektoril olema päised, et ühendada laiendus hosti süsteemile. Kõik lisaväljaviigud PCMCIA CardBus soketist tuleb vedada otse FPGA-sse.

6. GPIO klaviatuuri skaneerimiseks ja toitehalduseks

7. SDRAM DIMM soket
   SDRAM DIMM soket on ühendatud Vpnow-ga 32-bitises rezhiimis.. Vpnow'l on 8 programmeeritavat CS signaali. CS[0]..CS[3] ühendatakse SDRAM DIMM-iga.
8. FLASH laienduskaart (Flash extension card)
   Kaart ühendatakse flash soketiga. Lubab kasutada vpnow-d välisseadmetega. CS[4]..CS[7] ühendatakse laiendusslotiga.
9. DIP lülitid konfigureerimiseks
   8 DIP lülitid, et konfigureerida vpnow alglaadimist.

Silumise võimalused:

1. JTAG konnektor FPGA-le
   JTAG lubab FPGA-d programmeerida ja tema olekut välja lugeda. Samuti saab teda kasutada, et siluda protsessori tuuma tööd.
2. LED displeid (5*2 7-segmendiga displeid)
   Harilikud katood-segmentidega displeid. Vajavad võimendeid.

Täiendavad võimalused:

1. Kaks 40 väljaviiguga testi päist kiireks LVDS standardil põhinevaks ühenduseks
2. USB PHY draiver
   PHY draiver lubab süsteemile lisada USB.

2.2.3 Sünkroniseerimine ja reset

Kaardil on üks peamine reset signaal – selleks on FPGA, mis töötab vpnow tuumana. Madalaktiivne peamine reset genereeritakse kaardi peal oleva lülitiga ja ta on ühendatud FPGA-ga. Ülejäänud tuuma sisenevad ja sealt väljuvad resetid on seotud siinistandarditega. Nendeks on:

1. MII/ Ethernet PHY
   Reset väljaviiku pole defineeritud
2. PCI/ Cardbus
   Välisseadmest vpnow-sse from (PRST väljaviik)
3. SPI Serial PROM
   Vpnow-st serial PROM-i (CS_OE väljaviik)

Taktsignaalid genereeritakse välise kellageneraatori abil ja neid tarbivad kaardil olevad seadmed.

1. Vpnow tuum
   25 MHz taktsignaal, mis korrutatakse sisemiselt vajalikule sagedusele.
2. Ethernet PHY
   25 MHz taktsignaal, mida genereerib PHY 25 MHz kristall.
3. PCI liides
   33 MHz asünkroonne taktsignaal, mida genereerib välisseade.
4. SDRAM liides
   100 MHz taktsignaal, mida genereerib vpnow
5. SPI
   Vpnow genereerib aeglased (< 5 MHz) taktsignaalid sisemiselt.

Kõik ülejäänud taktsignaalid genereerib FPGA kasutades DCM (digital clock manager) mooduleid. Selleks, et minimiseerida taktsignaali hälbeid tuleb taktväljund Etherneti ja SDRAMi jaoks saata taktsisendi kaudu tagasi FPGA DLL-idesse.

2.2.4 Võimsustarve

Võimsuse tarbijateks on kiibil asuvad mikroskeemid. Nendeks on:

1. FPGA

Põhiliseks tarbijaks on kaardil asuv FPGA. Hetkel on kandidaadiks Xilinx Virtex-II seeria 6 miljoni loogikalüliga seade - X2V6000. FPGA on saadaval FF1152, FF1517 ja BF957 korpustes. Valisime BF957 korpuse, sest kasutatavate väljaviikude arv on piisav ja trasseerimise võrk on suurim võimalik. FPGA töötab 1.8 V toitepingega ja väljaviigud 3.3 V pingega. Eeldatavasti tarbib FPGA 3-5 korda rohkem võimsust kui tootena valmiv kiip ränis, seega peaks FPGA-sse minev vool olema kuni 2 A. Väljaviigud tarbivad kuni 0.5A voolu.

2. ETHERNET PHY seadmed

Ethernet PHY seadmed töötavad toitepingel 3.3 V. Võimalikeks kandidaatideks on 78q2120 firmalt TDK, DP83846A firmalt National Semiconductor või am79C874 firmalt AMD. PHY seadmete tootjate nimekiri on laiem. Oluliseks karakteristikuks on 10/100 ethernet ühenduse kiirus ja MII liidese toetus. Kõik eelpool loetletud seadmed töötavad ühisest 3.3 V pingeallikast ning tarbivad kokku 90 – 120 mA. Kõik nad kasutavad 25 töötakti.

3. USB

USB draiver hõlmab vaid ühte füüsilist draiverit. Mikroskeemil PDIUSBP11A firmalt Philips on kõik draiverid ühe USB pordi jaoks ühes mikroskeemis. Ta toetab kiirusi 12 ja 1.5 Mbit/sec ja töötab 3.3 V pingeallikaga. Kiip tarbib 100 mA täiskiirusel töötades. USB eeldab ka täiendavat mikroskeemi siini võimsuse haldamiseks.

4. Staatiline RAM mälu

Staatilist RAM-i kasutatakse, et emuleerida kiibipealset duaalport mälu. Kuna enamik andmeedastusi on BURST tüüpi, siis on ka kasutatav SRAM burst tüüpi. Vpnow tarvis vajaminev suurus on 64 KB (8K * 64 bit). Kaardil kasutatav mälu on SYNCBURST sram firmalt Micron. See 64K*32bit seade (MT58L64L32) töötab 3.3 V pingega ja tarbib 245 mA.

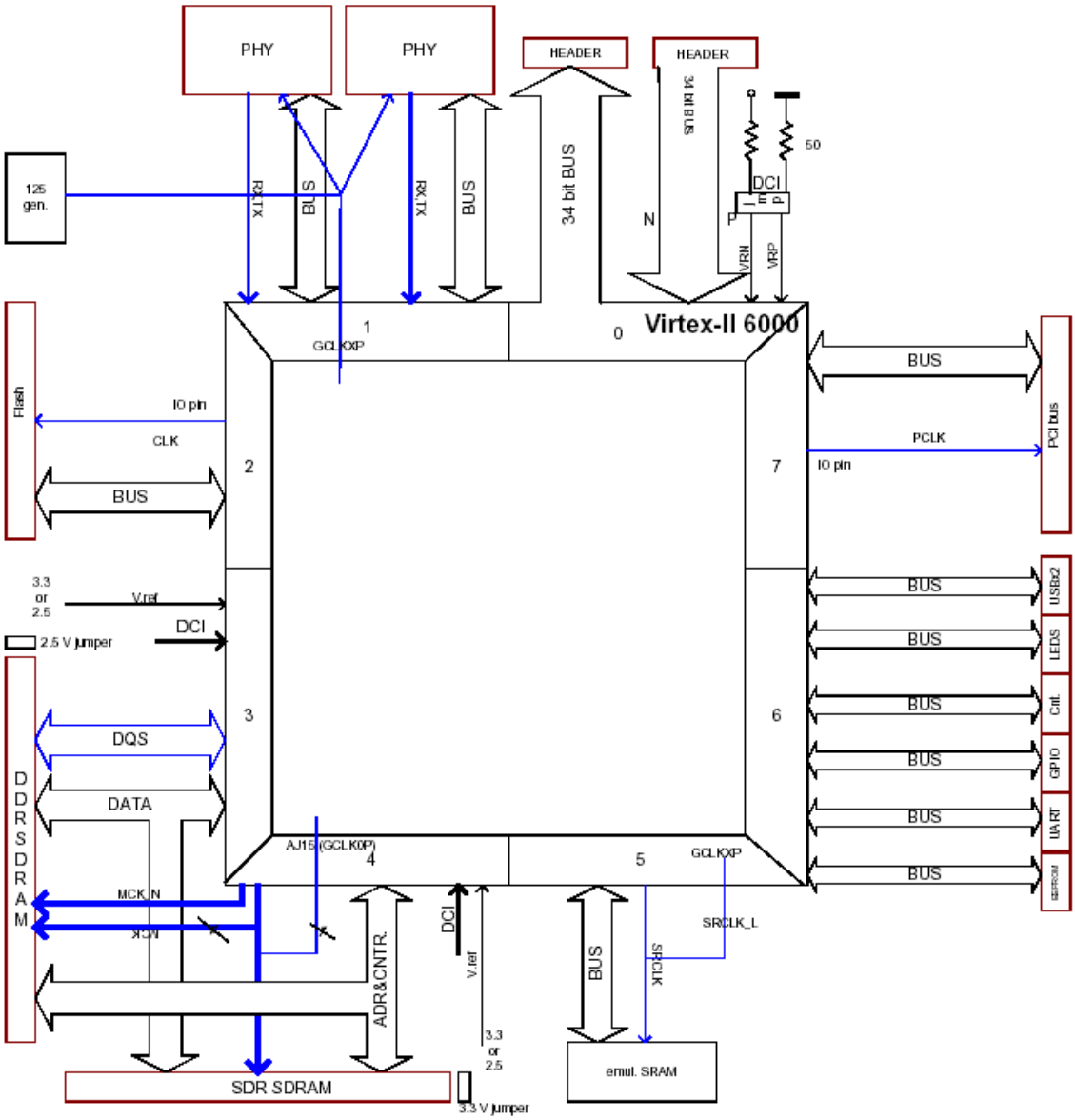
6. SDRAM

SDRAM ühendatakse DIMM soketisse. DIMM töötab 3.3 V pingel ja tarbib voolu kuni 2A.

7. DDR sdram

DDR sdram lisatakse tuleviku laiendusi silmaspidades ning teda käesolevas ei kirjeldata.

Joonisel 2.2 on esitatud VPNow tuuma verifitseerimisplatvormi XILINX Virtex 2 programmeeritava loogikamaatriksi üldine skeem.



Joonis 2.2. VPNow tuuma verifitseerimisplatvormi XILINX Virtex 2 programmeeritav loogikamaatriks

2.3 Tarkvara ja riistvara koosarendus ja test (Ülesanne 3)

Käesolevas alampeatükis on kirjeldatud krüptograafilise võrguprotsessor BITSie süsteemi funktsionaalsuse jagamine riist- ja tarkvara vahel. Tarkvara saab omakorda jagada kahte kategoorjasse: sisemine ja väline tarkvara. Sisemine tarkvara jookseb kiibil oleval RISC protsessoril, samas kui väline tarkvara töötab süsteemil, mis sisaldab võrguprotsessor BITSie't.

2.3.1 Multifunktsionaalne otsemälupöördussild

Multifunktsionaalne otsemälupöördussild (Data Pump) sisaldab mitut otsemälupöörduse (direct memory access e. DMA) moodulit, millel on ühine kiibil asetsev liides. Iga DMA moodul realiseerib kanali, mis on võimeline andmehulki edastama ja vastuvõtma. Andmeid saab vahetada kiibipealse Seadme ja mälukontrolleri vahel. Kanali lõpppunkt ja suund on riistvaras jäigalt fikseeritud, mis tähendab, et iga kanal saab vahetada andmeid ainult oma lõpppunkti ja mälukontrolleri vahel. Kanali lõpppunktideks on järgmised kiibi tuumad:

1) MAC_1_RX,
2) MAC_1_TX,
3) MAC_2_RX,
4) MAC_2_TX,
5) VPNow (krüptomoodul),
6) EXT_MEM,
7) PCI_RXTX

Multifunktsionaalsel DMA sillal on keskne roll Ethernetiga seotud andmeedastusprotsessis. Joonisel 2.3 on ära toodud võrguprotsessor BITSie riistvaraplokkid ning nende omavahelised ühendused.

Kõik joonisel 2.3 näidatud komponendid on riistvaras ühendatud. Kiibisisene RISC protsessor käivitab nn. planeerimistarkvara, mis konfigureerib DMA silla registrid ning kontrollib konfigureerimisprotsessi edukat läbimist. Kuna kirjeldatud moodul kujutab endast lihtsat DMA seadet, siis puudub vajadus kasutada spetsiaalset tarkvara seadme töös.

2.3.2 Tükeldus riist- ja tarkvaraks

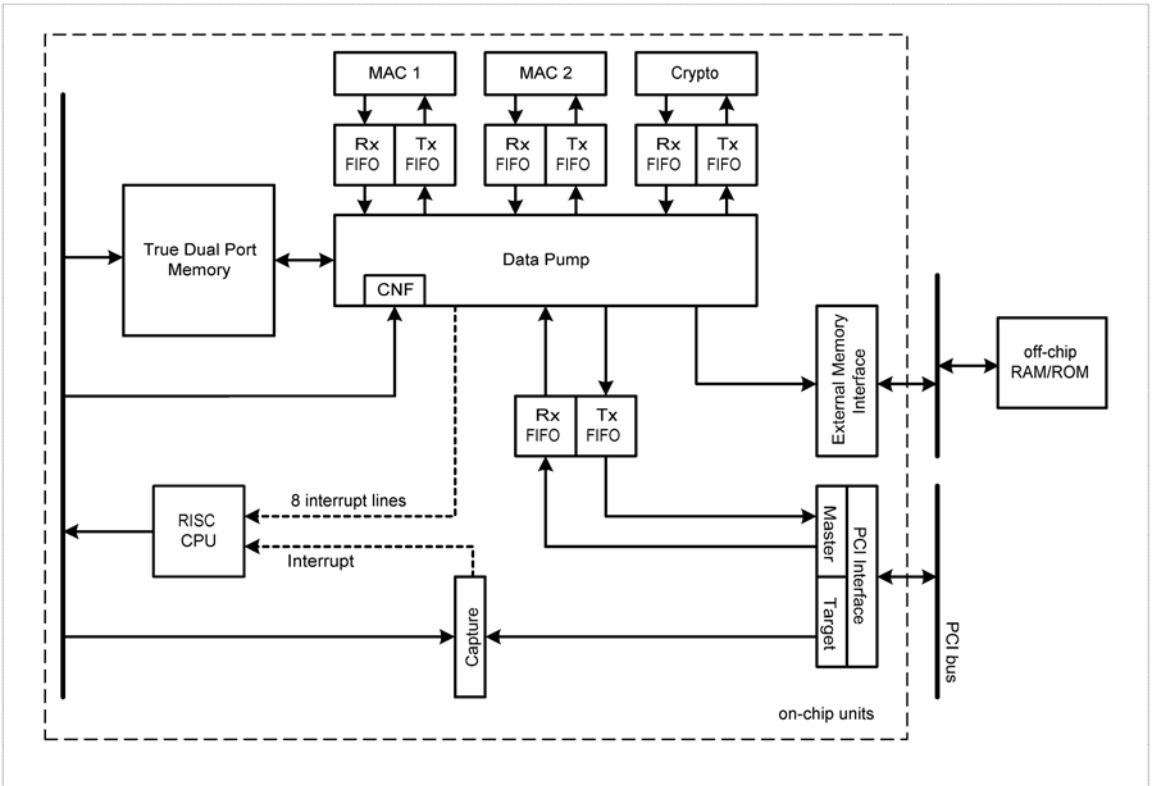
Läbiviidud riist- ja tarkvara koosarenduse käigus jagati multifunktsionaalse DMA silla alamsüsteemi funktsionaalsus järgnevalt:

Riistara funktsioonid:

- Kiire, lõimtöötlusel põhinev, pakettorienteeritud andmeedastus kiibipealse duaalport muutmälu ja kanali lõpppunkti vahel.

Tarkvara funktsioonid:

- Emuleerida Ethernet kontrollerit.
- Realiseerida deskriptorpõhist andmevahetust.

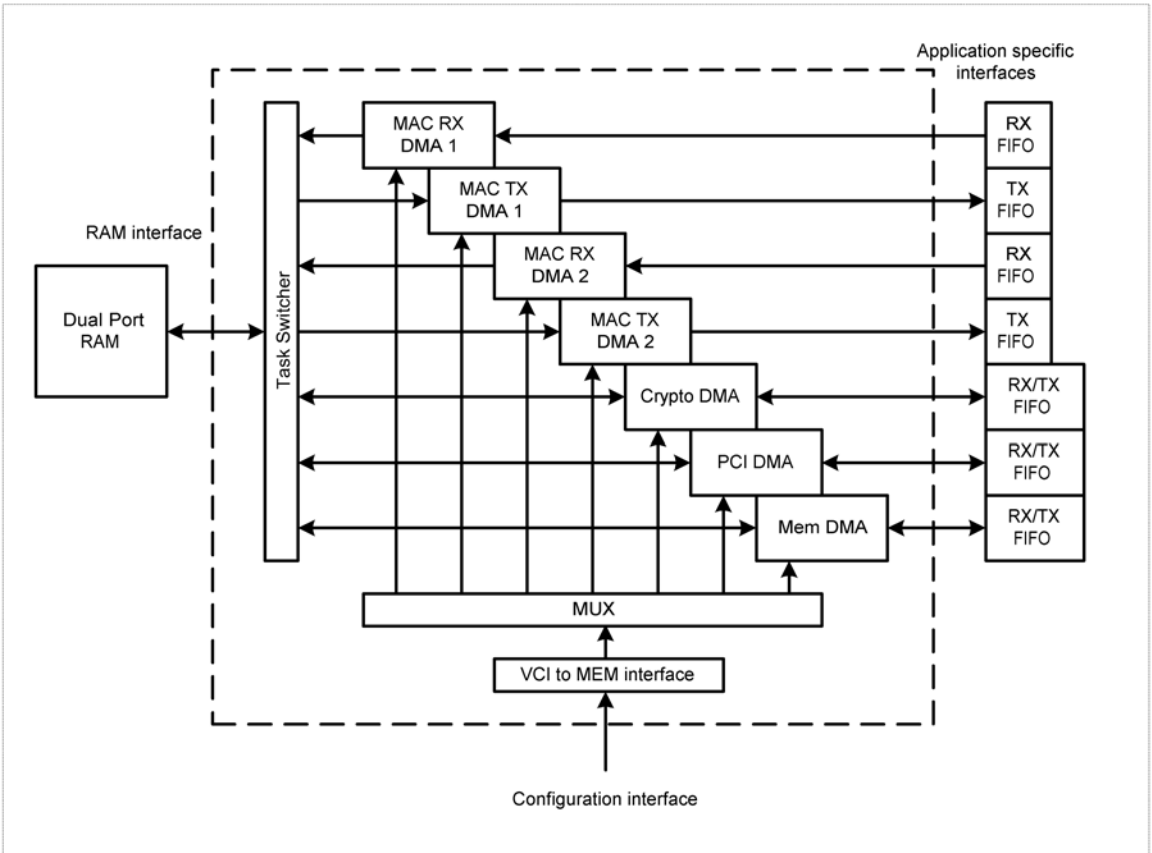


Joonis 2.3. Võrguprotsessor BITSie üldine struktuur

2.3.3 Otsemälupöördussilla sisestruktuur

DMA silla sisestruktuur on esitatud joonisel 2.4. Sild koosneb seitsmest DMA moodulist, ühest muutmälu liidesest (kujutatud vasakul) ja seitsmest spetsiaalsest liidesest sihtpunkt komponentidele. Iga DMA moodul viib läbi lõpppunkti juhtimist ja andmeedastust. Lõpppunkti juhtimine tähendab sisuliselt seda, et DMA moodul saab organiseerida andmevoogu FIFO registrites.

Igal DMA moodulil on neli 32-bitist konfiguratsiooniregistrit. Need registrid on ligipääsetavad otsemälupöördus-mooduli konfigureerimisliidese kaudu. Moodul Task Switcher jagab üht muutmälu liidest DMA moodulite vahel. Jagamine toimub järgnevalt. DMA moodul teatab soovist andmeid edastada. Kui ta saab selleks loa, siis toimub andmeedastus antud DMA mooduli kaudu senikaua, kuni ta võtab oma soovi peale andmeedastust tagasi. Seega tuleb DMA moodulid projekteerida nii, et nad viiksid läbi andmeedastust väikeste pakettidena. Pakettide suurused sõltuvad FIFO registrite sügavusest. Moodulile Task Switcher saavad DMA moodulid edastada soove kolmel tasemel. Järgmisena teenindatav DMA moodul valitakse vastavalt soovi esitamise tasemele ja kanali numbrile. Otsemälupöördussilla andmeosa on sisemiselt 64-bitine.



Joonis 2.4. Otsemälupöördussilla sisestruktuur

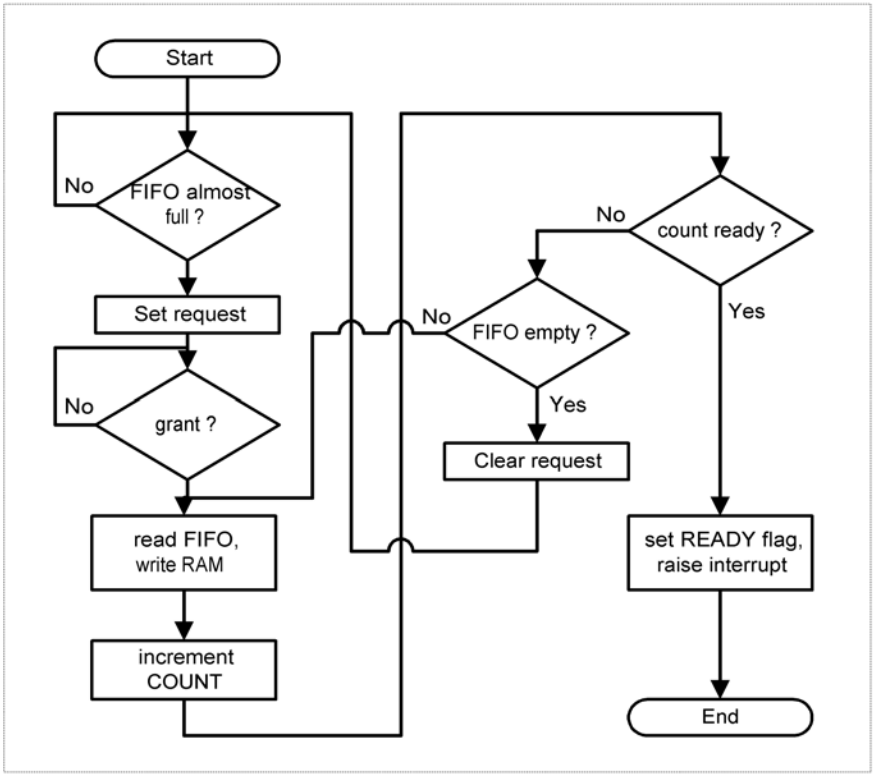
Kuna DMA on võimeline edastama andmeid kiibisisesest mälust kanali lõpppunkti ühe paketi kaupa, siis tuleb kogu andmevahetuse organiseerimine läbi viia mikroprotsessori poolt. See töö sisaldab puhvrite allokeerimist ja DMA moodulite käivitamist. Näiteks selleks, et saata andmepakett hosti mälust kiibi protsessorile tuleb täita järgmised operatsioonid:

1) Eraldada puhver kiibipealses duaalport mälus
2) Programmeerida PCI DMA moodul saatma pakett välismälust kiibipealse mälu puhvrisse
3) Oodata andmeedastuse lõppemist
4) Programmeerida MAC TX DMA plokk saatma pakett kiibisisesest puhvrist protsessorile
5) Oodata andmeedastuse lõppemist
6) Kontrollida staatust

Sarnased operatsioonid tuleb läbi viia kui võtta vastu andmepakett ja saata see hosti mälusse.

Ka shifreerimise ja dekrüptimise operatsioonid on põhimõtteliselt andmeedastuse operatsioonid. Andmete edastamine mingilt kiibil olevalt aadressilt teisele toimub läbi shifreerimisloogika.

Iga DMA moodul jooksutab eraldi mikroprogrammi. Joonisel 2.5 on esitatud lihtsustatud diagramm DMA vastuvõtva kanali mikroprogrammist. Programm käivitub kui protsessor paneb püsti RUN lipu ning lõpeb kui kõik andmebaidid on edastatud.



Joonis 2.5. DMA mooduli mikroprogramm

3 Lahendused süsteem-kiibil skeemide diagnostikaks

Käesolev peatükk hõlmab projekti raames TTÜ poolt välja töötatud lahendusi konkreetsemalt krüptograafilise võrguprotsessori ning laiemalt süsteem-kiibil metodoloogial põhinevate skeemide testiks. Vastavalt tööplaanis esitatud ajagraafikule (vt. Lisa 1) jagunesid esimesel projekti aastal TTÜ ülesanded kolme valdkonda:

1. Prototüüpkiibi struktuurianalüüs ja isetestimise metodoloogia väljatöötamine (Ülesanne 8)

   Nimetatud punkti alamosaks tuleb lugeda ka ülesannet 16 “kommertsiaalsete testilahenduste rakendamine”.

2. Diagnostikavahendite skeemiliidesed projekteerimistarkvaraga

   Riistvarakirjelduskeelte (EDIF, VHDL) kasutatavate alamhulkade defineerimine. Diagnostikavahendite skeemiliideste realiseerimine projekteerimistarkvaraga (Ülesanded 17, 18). Punkt hõlmab ka otsustusdiagrammide (OD) mudelite genereerimist kiibi moodulitele ehk tuumadele (Ülesanne 19).

3. Diagnostikaalgoritmide väljatöötamine

   Algoritmide väljatöötamine hierarhilisel OD mudelil põhinevaks veasimuleerimiseks (Ülesanne 9) ja testigenereerimiseks (Ülesanne 10). Tinglikult võib sellesse valdkonda lugeda ka isetesti emulaatori, kui konkreetse IT-lahenduse, realiseerimise (Ülesanne 11).

Käesolev peatükk on jaotatud alampeatükkideks vastavalt kolmele ülalesitatud valdkonnale ning neis sisalduvatele tööülesannetele.

3.1 Prototüüpkiibi struktuurianalüüs ja testimise metodoloogia

3.1.1 Prototüüpkiibi struktuurianalüüs ja testimise metodoloogia (Ülesanne 8)

Struktuurianalüüsi käigus tutvustas ADG esiteks TTÜ projekti osalisi projekteeritava seadme arhitektuuri ja funktsionaalsusega. (Ülevaade projekteeritavast kiibist on esitatud käesoleva aruande punktides 2.1.2, 2.2 ja 2.3). Nagu juba eelpool öeldud, koosneb krüptograafiline võrguprotsessor BITSie järgmistest moodulitest:

- Krüptograafiline moodul VPNow
  Kiirendab krüptoarvutusi võimaldades täiskiirusel võrguühendust.
- Otsemälupöördussild
  Organiseerib andmevahetust kiibipealse muutmäluga.
- SDRAM liides
  Garanteerib mälu kättesaadavuse toote kavandatud eluajaks (kolmeks aastaks).
- 2 etherneti adapterit
  Kiibiga võib ühendada kaks erinevat võrku.
- 32-bitine RISC mikroprotsessor
  250 MHz sisseehitatud RISC mikroprotsessor kiibi juhtimiseks.
- Süsteemi siiniliidesed
  Võimalik ühendada mikrokontrollerite ja PCI siinile. Mikrokontrollereid kasutatakse võrguaplikatsioonides nende soodsast hinna/efektiivsuse suhtest tulenevalt üha enam.
- Täiendavad moodulid.

Nagu juba kirjeldatud aruande punktis 2.1 on kiibisisese RISC protsessori esimeseks tööks laiendatud isetesti [COM_TEST] käivitamine, mille eesmärgiks on kontrollida struktuursete rikete võimalikku esinemist seadmes. Vastav lahendus on plaanis välja töötada projekti teise aasta jooksul.

Isetesti kasutamine võimaldab kiibi testimise kulusid alandada, kuna testrile investeeritakse palju vähem kui välist testrit kasutades ning testimiseks ei ole tarvis ka nii pikka aega kui seda kulub välise testri puhul. Lisaks sellele on võimalik isetesti rakendada vahetult enne seadme sisselülitamist seadme normaalses töös. Rikete avastamisel saab toetav personal seadme sel juhul korrasoleva vastu vahetada vältides seega võimalikke vigu, mis vastasel korral võiksid esineda kriitilistes olukordades.

Järgnevas on ära toodud üldine strateegia kiibi erinevate tuumade testimiseks.

Kiibisisene RISC protsessor

Mikroprotsessorid on oma funktsioonilt suhteliselt standardsed. Siin on olemas mitmeid standardseid lahendusi, kuidas test läbi viia. Antud projekti raames on plaanis kasutada lähenemist, kus proovitakse läbi kõik mikroprogrammi harud ja andmeosa testitakse juhuslike väärtustega, hinnates samaaegselt saavutatud vigade katet veasimulaatori abil.

Liideste tuumad

Liidesed ühendatakse testi käigus tagasisidestatud rezhiimis, et testandmeid saata ja vastuvõtta. Selle sammu poolt katmata jäänud rikked testitakse isetesti struktuuride poolt. Need muudavad seadme sisuliselt testvektorite generaatoriks/analüsaatoriks.

Mälude test

Üheks operatsiooniks võib olla ka seadmega ühendatud välismälu mahu ja korrasoleku test. Selleks on olemas standardsed tuntud meetodid.

Muud tuumad

ADG poolt väljatöötatavad tuumad - krüptograafiline moodul VPNow, otsemälupöördussild ja SDRAM liides kujutavad endast olulist osa kogu seadme funktsionaalsusest ja ülesehitusest. Krüptograafiline moodul on plaanis testida kasutades selleks globaalset isetesti ning täiendavat isetesti kontrollerit. Selline lahendus on põhjendatud kuna on teada, et krüptomoodulis läbiviidavad operatsioonid oma olemuselt pseudojuhusliku testjadaga hästi testitavad.

Ülejäänud kahe mooduli puhul tuleb valida järgmiste alternatiivide vahel:

- Viia läbi funktsionaalne isetest

Selle lahenduse eeliseks on, et selle puhul ei läheks tarvis täiendavat riistvara, kuid ta võib osutuda mittesobivaks kui genereeritud test osutub liialt pikaks, ei saavuta piisavat katet või väliskeskkonna modelleerimine funktsionaalse testi tuletamiseks on liialt keeruline.

- Realiseerida juurdepääsumehhanism krae ehk wrapper'ina

Lahenduse eeliseks on, et testid võib genereerida tuumale kui eraldiseisvale moodulile. Puuduseks

- Isetesti struktuuride lisamine

Lahendus nõuab täiendavate riistvaraliste struktuuride lisamist ning ei ole otstarbekas juhul kui testitav tuum omab keerulist juhtosa või ta ei ole pseudojuhuslike vektoritega testitav.

- Muuta kõik tuuma registrid skanneeritavateks

Viimane oleks kõige lihtsam lahendus, kuid nõuab palju täiendavat riistvara.

Võimalik on, et realiseerida tuleb kombinatsioon loetletud lahendusvariantidest. Kui testprogramm COM_TEST läbi ei lähe, siis loetakse seade rikkisolevaks ning vastav vea kood on võimalik JTAG pordist välja lugeda.

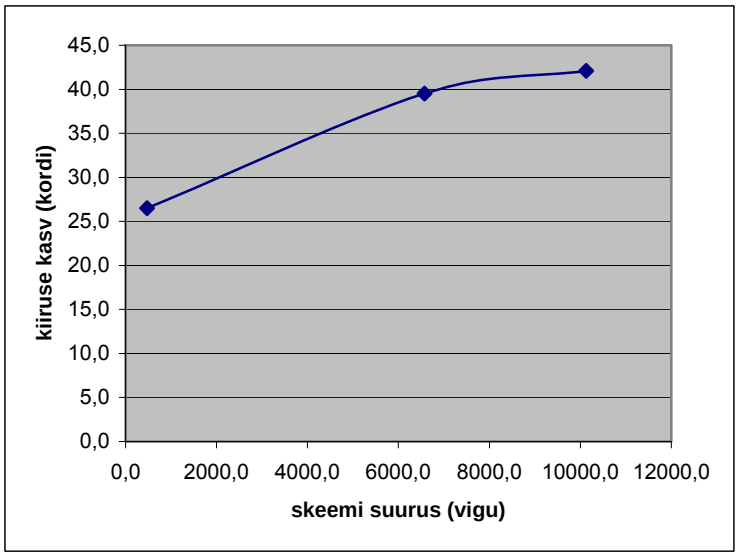
3.1.2 Kommertsiaalsete testilahenduste rakendamine (Ülesanne 16)

Kommertsiaalsest testitarkvarast oli TTÜ jaoks kõige olulisem mõõta kommertsiaalsete testigeneraatorite jõudlust ning võrrelda seda pakutava, otsustusdiagrammidel põhineva lähenemisega. Huvi pakkus eelkõige testigeneraatorid järjestikskeemidele. Nimetatud tooteid pakub hetkel vaid kaks firmat: Mentor Graphics ning Syntest. Esimesele programmile oli meil juurdepääs tänu sidemetele Linköpingi Ülikooliga (Rootsi) ning teine oli installeeritud Stuttgarti Ülikoolis (Saksamaa). Kuna Syntesti pakutav testigeneraator oli Mentor Graphicsi programmist palju võimsam, võtsime selle eksperimentide aluseks.

Eksperimendid on esitatud tabelis 3.1 viidi läbi neljal laialt levinud katseskeemil. Skeemide suurused ulatusid paarisajast loogikalülitusest kuni rohkem kui nelja tuhande lüliga näiteni. Katsetulemused näitasid, et kommertsiaalne tarkvara oli võimeline genereerima teste skeemidele, mis koosnesid kuni 1000-2000 loogikalülitusest suurema täpsusega kui TTÜ prototüüpversioon hierarhilisest testigeneraatorist. Sellest piirist edasi, aga vigade leidmise täpsus langes järult ning suuremate skeemide puhul andis hierarhiline otsustusdiagrammidel põhinev meetod suurema vea katte. Nagu näitab joonis 3.1, oli hierarhiline generaator oluliselt kiirem ning generaatorite kiiruste vahe isegi kasvas skeemi suuruse kasvades.

Tabel 3.1. Kommertsiaalse testitarkvara võrdlus TTÜ diagnostikalahendusega

skeem	loogika-lülisi	vigu mudelis	SYNTEST			Hierarhiline (TTÜ)		
			avastatud vigu	Aeg, s	testi pikkus	avastatud vigu	aeg, s	testi pikkus
gcd	243	474	**428**	92.7	**342**	421	**3.5**	1035
mult8x8	1164	2104	**1524**	2871	1450	1510	**13.3**	3250
risc	2830	6572	6212	1094	**1194**	**6352**	**27.7**	3777
diffeq	4312	10126	9683	1359	**529**	**9714**	**32.3**	6952

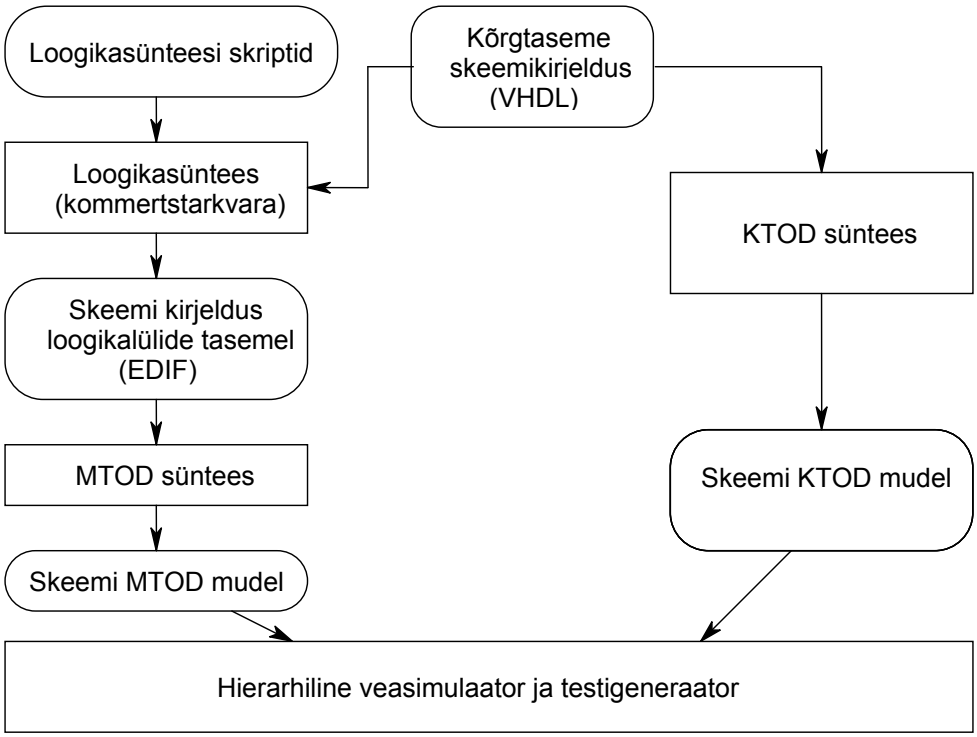


Joonis 3.1. Testigeneraatorite kiiruste vahe sõltuvalt skeemi suurusest

3.2 Diagnostikavahendite skeemiliidesed projekteerimistarkvaraga

3.2.1 Hierarhilise skeemiliidese struktuur

Projekti diagnostikavahendite skeemiliidesed on realiseeritud järgnevalt. Kõrgtaseme VHDL-keelsed skeemikirjeldused konverteeritakse kõrgtaseme OD (KTOD) mudeliks, kasutades selleks spetsiaalselt VHDL liidest. VHDL-kirjeldusel viiakse läbi ka loogikasüntees vastava kommertstarkvara abil. Sünteesi tulemusena saadakse skeemi moodulite loogikalülide taseme kirjeldus, mis konverteeritakse madala taseme OD (MTOD) mudeliks, kasutades selleks otstarbeks loodud EDIF formaadi liidest. Hierarhilise testitarkvara skeemiliideste üldine struktuur on kujutatud joonisel 3.2.



Joonis 3.2. Diagnostikavahendite integreerimine kommertsprojekteerimistarkvaraga

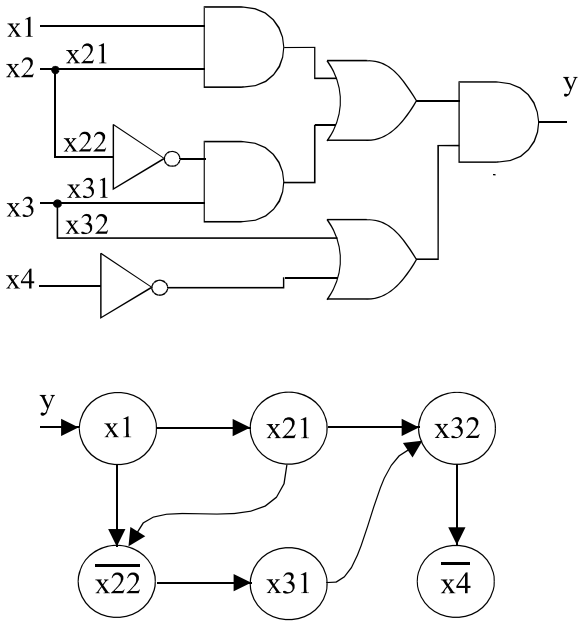
Käesolevas alampeatükis on kirjeldatud EDIF liides, VHDL liides ning kiibi tuumade konverteerimine MTOD mudelile.

3.2.2 Liides EDIF formaadist MTOD mudeli sünteesiks (Ülesanne 17)

Käesoleva projekti raames väljatöötatav testitarkvaras kasutatakse skeemi loogikataseme kujutamiseks madaltaseme otsustusdiagrammide (MTOD) mudelit, mis kujutab endast erijuhtu klassikalisest binaarsete otsustusdiagrammide (BOD) mudelist. MTOD mudelil on traditsioonilise BOD mudeli ees rida eeliseid. Erinevalt BOD mudelist, võimaldab MTOD kirjeldada skeemi struktuuri otseselt mudelis endas: iga mudeli tipp vastab üks-üheselt teatud struktuursele teele skeemis. Lisaks sellele on MTOD mudelit võimalik kiirelt genereerida praktiliselt suvalisele reaalsele loogikaskeemile, sest mudeli genereerimise halvim aja ja mälu nõudlus on lineaarne, samas kui BOD mudeli mälunõudlus kasvab halvimal juhul eksponentsiaalses suhtes skeemi suuruse kasvuga.

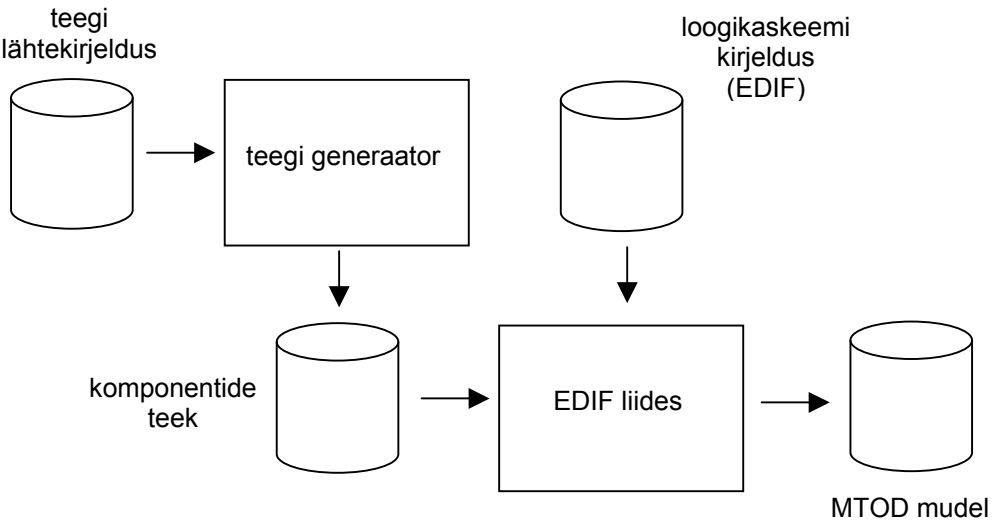
Teatavasti genereeritakse traditsiooniline BOD mudel kasutades Shannoni arendust. MTOD mudeli genereerimisel kasutatakse aga superpositsiooni printsiipi. Just superpositsiooni kasutamine võimaldab MTOD mudeli puhul säilitada skeemi struktuurse informatsiooni. MTOD mudel genereeritakse liikudes skeemi väljunditest sisendite poole ning asendades iga loogikalüli rekursiivselt talle vastava elementaar-BOD mudeliga. Et vältida mudeli plahvatuslikku kasvu, tükeldatakse skeem hargnemisteta puukujulisteks alamskeemideks ning igale sellisele alamskeemile genereeritakse eraldi MTOD. Alamskeemideks tükeldamine tagabki MTOD mudeli genereerimise lineaarses ajas ja ruumis.

Joonisel 3.3 on toodud näide kombinatsioonskeemist ning selle MTOD esitusest. Lihtsuse mõttes on väärtused 0, 1 jäetud graafi kaartele märkimata ja kokkuleppeliselt loetakse paremale viivat kaart 1-kaareks ning alla viivat kaart 0-kaareks. Samuti on jäetud kujutamata terminaaltipud 0 ja 1. Väljumine graafist paremale vastab tulemusele y = 1 ning väljumine alla tulemusele y = 0. Saadud MTOD koosneb kuuest tipust ning igaüks neist kujutab ühte struktuurset teed mudelile vastavas skeemis.



Joonis 3.3. Loogikaskeem ja talle vastav MTOD mudel

TTÜs töötati välja liides madaltaseme otsustusdiagrammide (MTOD) mudeli sünteesiks EDIF 2.0.0 formaadis. Liides toetab laia EDIF formaadi alamhulka, mis võimaldab teisendada enamuse maailma juhtivate CAD süsteemide k.a Synopsys, Mentor Graphics, Cadence, ViewLogic skeeme. Kuna EDIF formaat ei käsitle valitud teegi loogikalülide funktsionaalset informatsiooni vaid vaatleb neid “mustade kastidena”, oli tarvis realiseerida ka loogikalülide teegi generaator. Nimetatud programm võimaldab kasutajal teegi komponentide kirjeldused lihtsas formaadis (vt. lisa 3) ette anda ning EDIF-i komponentide teek automaatselt genereerida. Joonisel 3.4 on kirjeldatud realiseeritud MTOD liidese struktuuri.



Joonis 3.4. Realiseeritud MTOD liidese struktuur

Eraldi ülesandeks kujunes ADG poolt kasutatava reaalse tehnoloogiateegi CORELIB komponentide kirjeldamine. Teek sisaldas 777 komponenti, millest on käesolevaks kirjeldatud 540 funktsionaalsus. Kirjeldamata jäid näiteks lukktrigerid (latches) ja muud komponendid, millele antud projekti raames eeldatavasti rakendus puudub. Vajaduse korral on tulevikus võimalik toetatavate komponentide hulka suurendada.

Lisaks eelpool kirjeldatud EDIF formaadist MTOD mudeli genereerimisele võimaldab realiseeritud liides lugeda EDIF-i asemel ISCAS´89 formaati või seda formaati EDIF-ist genereerida. Tänu sellele on TTÜ testitarkvara võimalik võrrelda suurema osaga olemasolevast akadeemilisest diagnostikatarkvarast. Realiseeritud liidesprogrammide käsuread on esitatud lisas 2.1.

3.2.3 Liides VHDL riistvarakirjelduskeelest KTOD mudeli sünteesiks (Ülesanne 18)

Kasutades riistvara kirjelduskeelt VHDL on digitaalsüsteemi võimalik kirjeldada erinevatel abstraktsioonitasemetel. Käesolevas peatükis vaatleme kõrgtaseme kirjeldusi, mida võib omakorda tinglikult jagada kolmeks tasemeks (alustades abstraktsemast):

1. Käituvuslik tase

   Sellel tasemel on esitatud skeemi funktsionaalsus algoritmina. Puudub detailne info skeemi realisatsiooni kohta. Puudub sünkronisatsoon, s.t. ei ole näidatud sisend-väljund signaalide ajastust.

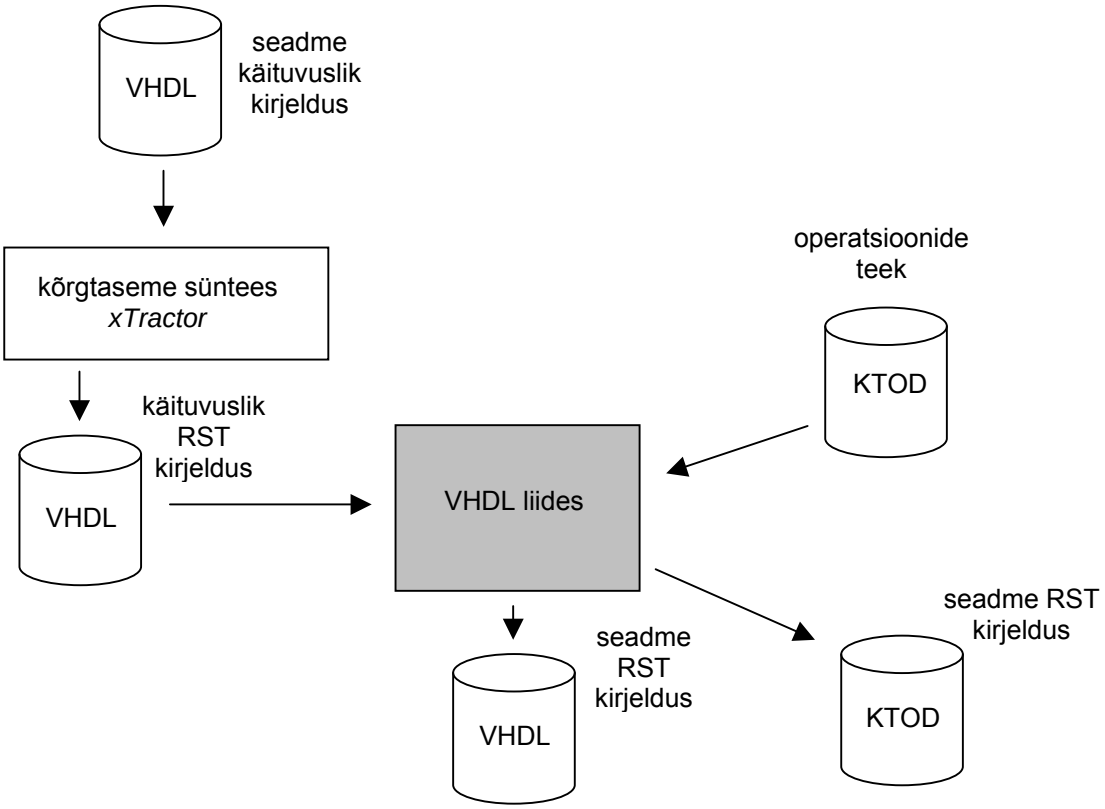
2. Käituvuslik register-siirde tase (RST)

   Esitatud on juhtautomaadi olekud ja signaalid on sünkroniseeritud. Kuid sel tasemel puudub hierarhia, pole teada, millistes moodulites on mingi operatsioon realiseeritud ning pole kirjeldatud multipleksereid.

3. Register-siirde tase

   RST tasemel on seade tükeldatud juht- ning operatsioonautomaadiks. Juhtautomaat on esitatud olekutabelina. Operatsioonautomaat on esitatud struktuurselt, registrite, operaatormoodulite ning multiplekserite võrguna. Võimalik on hierarhiline esitusviis, kus iga operatsioonautomaadi mooduli jaoks on sünteesitud ka vastav loogikataseme kirjeldus.

Tänu oma hierarhilisele iseloomule on just RST tase valitud käesolevas projektis väljatöötatud algoritmide (vt. peatükid 3.3.1 ja 3.3.2) sisendmudeliks. Kuid RST mudeli näol on tegemist praktilises projekteerimistöös harvaesineva kirjeldusviisiga ning seetõttu realiseeriti KTOD liides, mis kasutab sisendina käituvusliku RST taseme VHDL-i. Nimetatud lähenemisel on mitu eelist. Esiteks, käitumuslik RST on projekteerijate poolt laialt kasutatav kirjeldusvorm ning liides sellest võimaldab rakendada projekti raames väljatöötatavat hierarhilist testitarkvara suure hulga reaalsete näidete peal. Teiseks, võimaldab selline liides kasutada stsenaariumi, kus projekteerijal on olemas seadme käituvuslik kirjeldus ning kasutades kõrgtaseme sünteesi tarkvara (näiteks TTÜs varem väja töötatud programmi xTractor) luuakse automaatselt käituvuslik RST. Joonis 3.5 iseloomustab realiseeritud VHDL liidese kasutamist digitaalseadmete kõrgtaseme sünteesi protsessis.

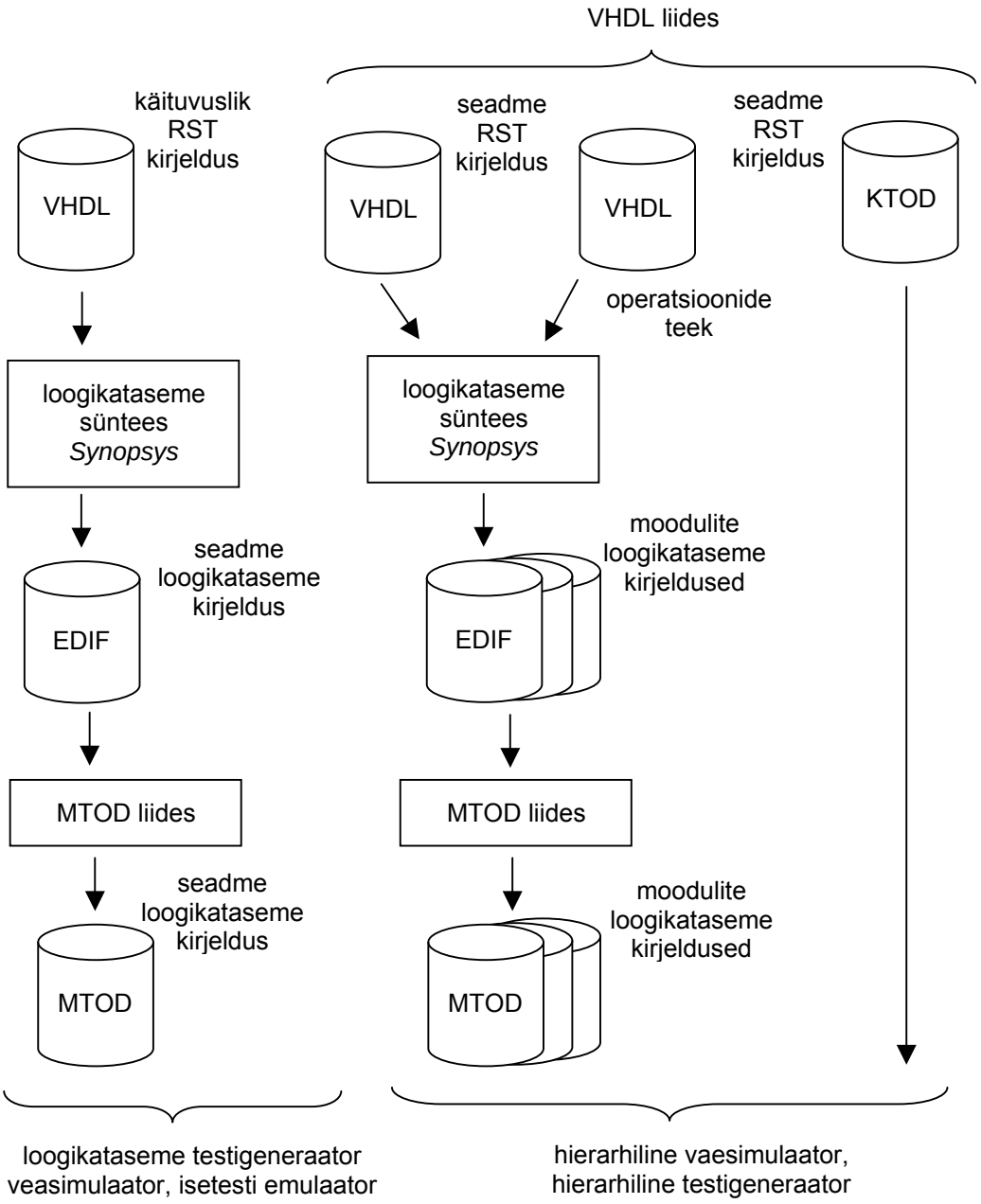


Joonis 3.5. Digitaalskeemide kõrgtaseme süntees ja VHDL-KTOD liides

VHDL liides genereerib seadme RST mudeli VHDL ning KTOD formaadis. Seadme VHDL mudelist ning VHDL liidese operatsioonide teegist sünteesitakse operatsioonautomaadi moodulite loogikataseme kirjeldused. (Käesolevas projektis kasutab TTÜ loogikasünteesiks kommertstarkvara Synopsys). Seejärel konverteeritakse moodulite EDIF kirjeldused vastava liidese abil MTOD mudeliteks. Saadud MTOD mudelid koos seadme KTOD kirjeldusega on sisendiks projekti käigus väljatöötatavatele hierarhilistele diagnostikalahendustele.

Alternatiivseks võimaluseks on kasutada seadme mittehierarhilist kirjeldusviisi loogikalülituste tasemel. Selleks sünteesitakse skeemi käituvuslikust RST kirjeldusest loogikalülituste tase kasutades vastavat sünteesitarkvara. Selline kirjeldus on sisendiks näiteks loogikatasemel töötavale veasimulaatorile, testigeneraatorile ning antud projekti raames väljatöötatud isetesti emulaatorile (vt. peatükk 3.3.3).

Joonisel 3.6 on ära toodud eelpool kirjeldatud sisendmudelite genereerimine projekti diagnostikalahendustele.



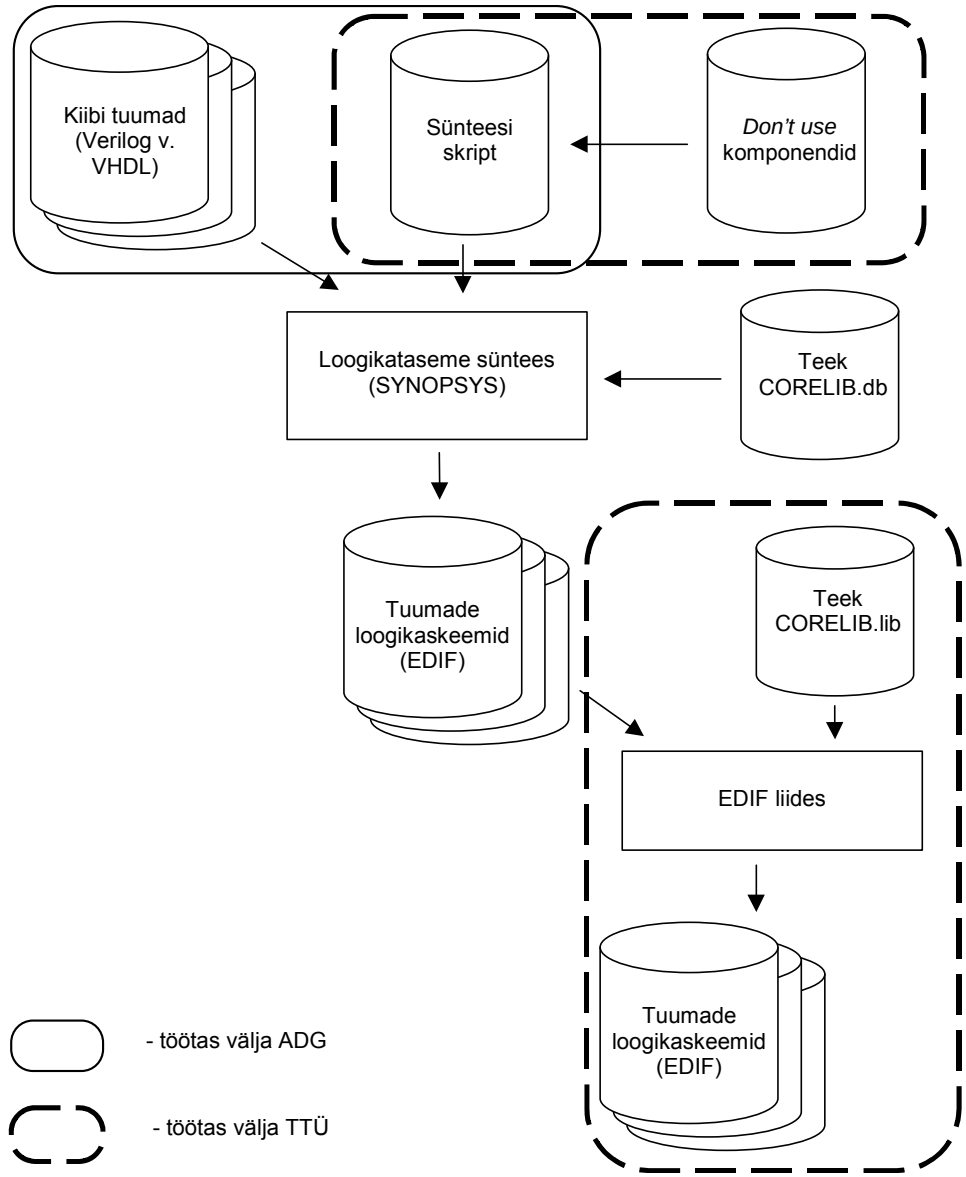
Joonis 3.6. VHDL liidese väljundi kasutamine diagnostikalahenduste poolt

Käesolevas alampeatükis kirjeldatud liidese poolt toetatav VHDL-i alamkuju on defineeritud lisas 4. Liidese programmi käsurida on esitatud lisas 2.2.

3.2.4 MTOD mudeli süntees prototüüpkiibi moodulitele (Ülesanne 19)

Joonisel 3.7 on esitatud skeem kiibi tuumade konverteerimisest madaltaseme otsustusdiagrammide (MTOD) mudelile. MTOD mudeli genereerimine algab loogikasünteesiga, milleks kasutatakse firma SYNOPSYS programmi Design Compiler. Kasutades ADG poolt disainitud sünteesiskripti edasiarendust teisendab loogikataseme süntees tuuma Verilog või VHDL kirjelduse loogikataseme EDIF kirjelduseks. Seejuures loeb sünteesiskript faili komponentide nimistu, mida ei lubata sünteesitavas skeemis kasutada (vt. peatükk 3.2.2). Loogikaskeemist genereerib MTOD liides (vt. 3.2.2) madaltaseme otsustusdiagrammide kirjelduse, kasutades EDIF formaadi parsimiseks programmi raames loodud CORELIB.lib tehnoloogia teegi kirjeldust.

Hetkel on tuumade konverteerimine täielikult automatiseeritud ning verifitseeritud SDRAM kontrolleri tuumal. Konverteerimise tulemusena saadi 2305 graafist ja 12433 tipust koosnev otsustusdiagrammi mudel.



Joonis 3.7. MTOD mudeli süntees prototüüpkiibi moodulitele

3.3 Algoritmid ja tarkvara süsteem-kiibil skeemide testimiseks

Käesolevas alampeatükis vaatleme TTÜ poolt välja töötatud testilahendusi. Peatükid 3.3.1 ja 3.3.2 kirjeldavad uudseid otsustusdiagrammidel töötavaid hierarhilisi algoritme vastavalt veasimuleerimiseks ja testide genereerimiseks. Peatükis 3.3.3 on kirjeldatud isetesti emulaatori realisatsioon.

3.3.1 Hierarhilisel OD mudelil põhinev veasimuleerimisalgoritm (Ülesanne 9)

3.3.1.1 Sissejuhatus

Veasimuleerimine on protsess, mida viiakse läbi mingi testide hulga kvaliteedi mõõtmiseks. Selline protsess kuulub tihtikasutatava alamprotseduurina testigenereerimise ja isetestimise vahendite koosseisu. Võttes arvesse tänapäeva digitaalsüsteemide keerukust, ei ole traditsioonilised, ainult loogikalülituste tasemel töötavad veasimulaatorid enam praktikas rakendatavad oma madala töökiiruse tõttu. Ühe lahendusena, et veasimuleerimisprotsessi kiirendada oleks vajalik kasutusele võtta uudseid, hierarhilisi testide kvaliteedi hindamise meetodeid, milledes kasutatakse nn. “jaga ja valitse” põhimõtet.

Käesoleva projekti tulemusena töötatakse välja uudne hierarhiline veasimulaator, mis baseerub mitmetasemelise otsustusdiagrammi (OD) mudeli kasutamisel. Mitmetasemelised OD mudelid võimaldavad digitaalsüsteemide veaanalüüsi põhjus-tagajärg seoste tõhusat analüüsi. Selline uudne matemaatiline baas hierarhilisele veasimuleerimisele lubab üldistada tuntud madala taseme veaanalüüsi meetodeid kõrgemal abstraktsioonitasemel. Projekti raames loodavat uudset hierarhilist veasimulaatorit on plaanis kasutada nii eraldiseisva programmi kui ka isetesti väljatöötatamise vahendite koostisosana.

Käesolevas aruandes on esitatud hierarhilisel OD mudelil töötava veasimulaatori algoritm [2].

3.3.1.2 Meetodi üldine kirjeldus

Digitaalskeemi vigade analüüs toimub moodul-mooduli kaupa. Igal algoritmi iteratsioonil valitakse sihtmoodul, mida vaadeldakse loogikatasemel, samas kui ülejäänud mooduleid käsitletakse kõrgemal abstraktsioonitasemel. Sihtmooduli simuleerisel, määratakse igal iteratsioonil vead, mis põhjustavad selle mooduli puhul vigast väljundväärtust antud sisendvektori ning seadme oleku juures. Sihtmoodulis avastatavate vigade levitamine toimub kõrgemal tasemel. Seda protsessi iseloomustab järgnev näide.

Näide 1. Joonisel 3.8 on toodud näide hierarhiliselt esitatud digitaalsüsteemist, mis koosneb kolmest moodulist: A,B, ja C. Sihtmooduliks, millel veaanalüüs läbi viiakse on valitud moodul B ning see moodul on esitatud loogikalülituste tasemel. Testvektorite jada simuleeritakse läbi skeemi kõrgtaseme kirjeldusel, vektor-vektori kaupa. Kui jada esimese vektori P puhul jõutakse sihtmoodulini B, siis viiakse selles läbi loogikataseme veaanalüüs. Selle käigus leitakse vigade hulk R, mis põhjustavad mooduli B väljundi muutumist. Iga sellise aktiveeritud vea $r \in R$ jaoks leitakse vastav vigane moodul B väljundvektor $P(r)$. Aktiveeritud vead jaotatakse alamhulkadeks $R_i \subseteq R$, i = 1,…,k nii et iga r $\in R_i$ jaoks on väljundvektor $P(r)$ sama. Veavaba vektor P ja kõik veaga vektorid P_1, …,P_k simuleeritakse läbi teiste seadme moodulite kõrgtasemel. Juhul kui $P_i \neq P$ mõnes seadme jälgitavas väljundis, siis öeldakse, et vead R_i on avastatud. Kord juba avastatud vead võib edasisest analüüsist välja jätta.

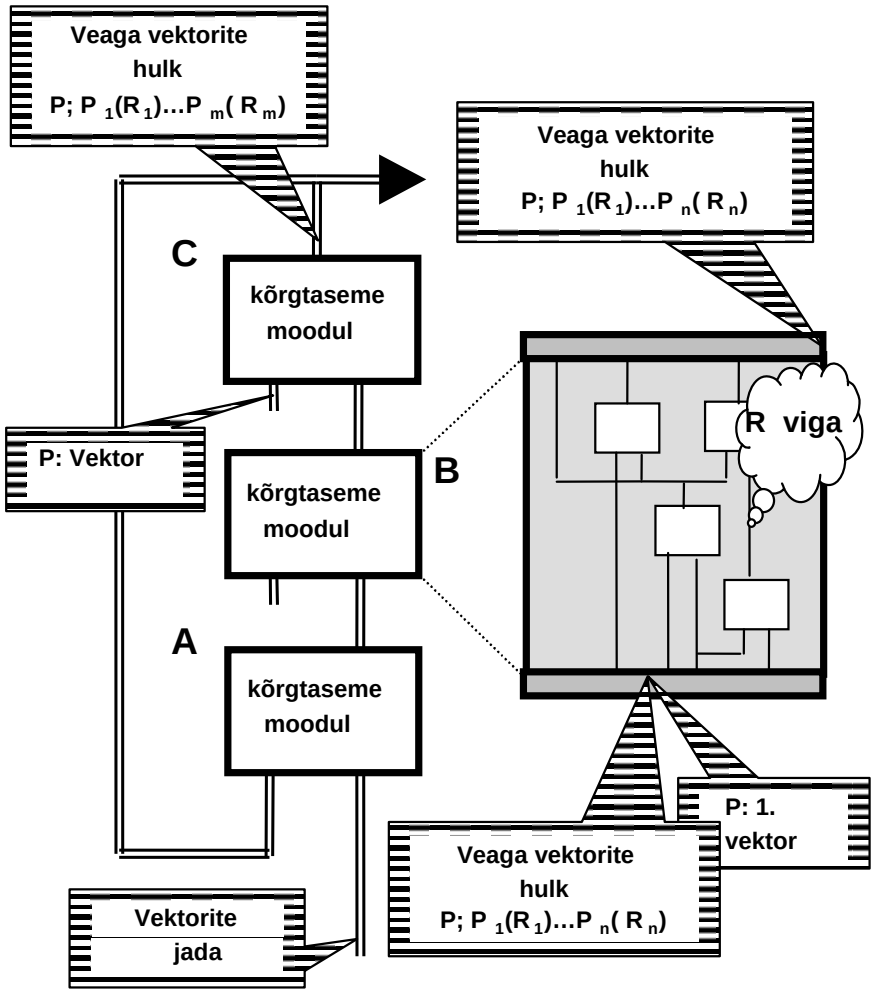
Üldjuhul genereeritakse iga kõrgtaseme mooduli väljundisse andmestruktuur (kompleksvektor) $D = \{P, (P_1,R_1), \ldots, (P_L,R_L)\}$, kus $R_1 \cup R_2 \cup \ldots \cup R_L \subseteq$ R ja L ≤ k (kuna läbi moodulite levimisel võivad veagrupid R_i sumbuda või omavahel liituda!). Kui järgmisel iteratsioonil jõuab sihtmoodulini B kompleksvektor D, siis kõikidel selles sisalduvatel vektoritel $\{P,P_1, \ldots,P_k\}$ tuleb läbi viia veasimuleerimine loogikatasemel. Vektori P jaoks määratakse uued vead, mida P aktiveerib ja kõigi vigade jaoks, mis kuuluvad veagruppidesse R_i tehakse kindlaks, kas nad võimaldavad veavektor P_i levimist uuesti läbi B või mitte. Seejärel luuakse uus andmestruktuur D sihtmooduli B väljundis.

Vaatleme moodulit, mis kujutab endast n argumendiga funktsiooni $y = f(x_1, \ldots x_n)$ ja kompleksvektorite hulka $D_i = \{P_{i,0}, (P_{i,1},R_{i,1}), \ldots, (P_{i,ki},R_{i,ki})\}$, $i = 1,2, \ldots, n$, kus $P_{i,0}$ on veavaba vektor sisendil x_i, ja $P_{i,j}$ on võimalikud veaga vektorid samal sisendil patterns ja vastavad

$$R_i = R_{i,1} \cup R_{i,2} \cup \ldots \cup R_{i,k_i} \subseteq R$$

vigadele $R_{i,j}$, j=1,2,… k_i. Tähistame

vigade hulka, mis on levitatud sisendisse x_i, kus R on kogu vigade hulk vaadeldavas sihtmoodulis.

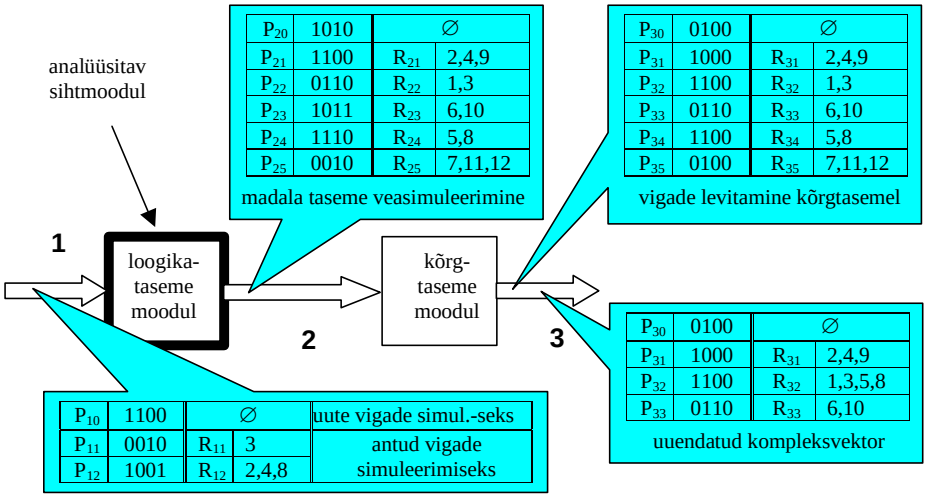


Joonis 3.8. Hierarhiline veasimuleerimine

Protseduur 1. Selleks, et simuleerida kõrgtaseme moodulit kompleksvektoril $D = (D_1,\ldots, D_n)$, tuleb esiteks läbi simuleerida veavaba vektor $(P_{1,0},\ldots,P_{n,0})$ kõrgel tasemel. Seejärel tuleb kõigi moodulini levitatud vigade

$$r \in R' = \bigcup_{i=1,n} \left(\bigcup_{j=1,ki} R_{ij} \right) \subseteq R$$

jaoks kõrgtasemel simuleerida vektorid $(P_1(r),\ldots,P_n(r))$, kus $i = 1, \ldots, n$ $P_i(r) = P_{i,0}$ kui $r \notin R_i$, ja $P_i(r) = P_{ij}$ kui $r \in R_{ij}$.



P_{20}	1010	∅	
P_{21}	1100	R_{21}	2,4,9
P_{22}	0110	R_{22}	1,3
P_{23}	1011	R_{23}	6,10
P_{24}	1110	R_{24}	5,8
P_{25}	0010	R_{25}	7,11,12

P_{30}	0100	∅	
P_{31}	1000	R_{31}	2,4,9
P_{32}	1100	R_{32}	1,3
P_{33}	0110	R_{33}	6,10
P_{34}	1100	R_{34}	5,8
P_{35}	0100	R_{35}	7,11,12

P_{30}	0100	∅	
P_{31}	1000	R_{31}	2,4,9
P_{32}	1100	R_{32}	1,3,5,8
P_{33}	0110	R_{33}	6,10

P_{10}	1100	∅	
P_{11}	0010	R_{11}	3
P_{12}	1001	R_{12}	2,4,8

Joonis 3.9. Vigade levitamine hierarhilises veasimuleerimises

Protseduur 2. Selleks, et simuleerida sihtmoodulit madalal tasemel kompleksvektoril $D = (D_1,\ldots, D_n)$, tuleb esiteks kõrgtasemel määrata mooduli veavaba käitumine vektoril $(P_1,\ldots,P_n)$ ning arvutada madaltaseme veasimuleerimise teel need vead, mis põhjustavad mooduli vale väljundväärtust, arvestades vaid neid vigu, mille mõju ei levinud sihtmoodulini. Seejärel tuleb vaadelda vigu $r \in R$, mis levisid uuesti sihtmoodulini kompleksvektoril D ning vektorid $(P_1(r), \ldots,P_n(r))$ tuleb simuleerida viies sisse vead r.

Näide 2. Näide vigade levitamisest hierarhilises veasimuleerimises on esitatud joonisel 3.9. Oletame, et loogikatasemel esitatud sihtmooduli sisendisse 1 levis kompleksvektor $D_1 = \{P_{10}, (P_{11},R_{11}), (P_{12},R_{12})\} = \{1100, 0010(3), 1001(2,4,8)\}$. S.t. sihtmooduli vead 2,3,4,8 on levinud tagasiside kaudu moodulisse tagasi. Veaanalüüsi tulemusena leiame, et 12 viga avastatakse mooduli väljundis. (Seda veavaba vektori P_{10} = 1100 ja moodulini levitatud veaga kompleksvektorite 0010 and 1001 tõttu). Need vead põhjustavad 5 erinevat vigast mooduli väljundreaktsiooni, mis erinevad oodatud väärtusest 1010. Kõik 6 väljundvektorit simuleeritakse kõrgtasemel järgmisel moodulil. Selle tulemusena saame $P_{30}=P_{35}$, mis tähendab, et vead 7,11 and 12 maskeeruvad – paar (P_{35},R_{35}) heidetakse kõrvale edasisest analüüsist. Kuna $P_{32}=P_{34}$, siis vigu 1,3 ja 5,8 ei saa eristada ning me koondame need ühte veagruppi $R_{32} = \{1,3,5,8\}$.

Käesolevas projektis välja pakutud algoritmi uudsuseks on, et moodulite madaltaseme veaanalüüsiks ning kõrgtaseme vealevitamiseks kasutatakse sama, otsustusdiagrammidel põhinevat matemaatilist mudelit.

3.3.1.3 Otsustusdiagrammide mudel

Otsustusdiagrammid on diskreetsete funktsioonide esitusvorm graafina. Diskreetne funktsioon y = f(x), kus $y=(y_1,\ldots y_n)$ ja $x=(x_1,\ldots x_m)$ on vektorid, on defineeritud lõplikul määramispiirkonnal $X=X_1\times\ldots\times X_m$ ning muutumispiirkonnal $y \in Y = Y_1\times\ldots\times Y_n$. Muutujad võivad olla tõeväärtus, tõeväärtusvektori või täisarvu tüüpi.

Diskreetsete funktsioonide y=f(x) kirjeldamiseks kasutame otsustusdiagrammi G_y. Otsustusdiagramm (OD) on suunatud mittetsükliline märgitud graaf, mida saab esitada nelikuna G=(M,E,X,Γ), kus M on lõplik tippude hulk, E on lõplik karate hulk, X on funktsioonide hulk, mis kirjeldab tippe märkivaid muutujaid ja nende määramispiirkondi ja Γ on funktsioonide hulk kaartel E.

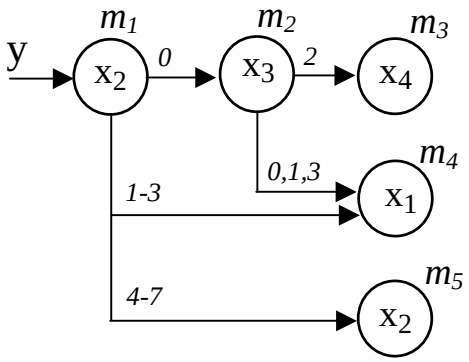
Funktsioon $X(m_i)$ tagastab paari (x_i,X_i), kus x_i on muutuja, mis märgib tippu m_i ja X_i on muutuja x_i määramispiirkond. Iga OD tipp on märgitud muutujaga, kusjuures üks muutuja võib märkida mitmeid erinevaid tippe. Erijuhul võivad OD tipud olla märgitud konstantide ning algebraliste avaldistega.

OD kaar $e\in E$ on järjestatud paar $e=(m_1,m_2)\in E^2$, kus E^2 on kõikvõimalike järjestatud paaride hulk hulgal E. e graafiliseks interpretatsiooniks on suunatud kaar, mis viib tipust m_1 tippu m_2. Öeldakse, et tipp m_1 on tipu m_2 *eellane*, ja m_2 on vastavalt tipu m_1 *järglane*.

Γ on funktsioon hulgast E ning ta kirjeldab kaarte aktiveerimistingimusi simuleerimis-protsessis. Γ(e) väärtus on X_i alamhulk, kus $e=(m_i,m_j)$ ja $X(m_i)=(x_i,X_i)$. Oluline on märkida, et $Pm_i=\{\ \Gamma(e) \mid e=(m_i,m_j)\in E\ \}$ on hulga X_i tükeldus.

OD-l on ainult üks algustipp, s.o. tipp, millel eellased puuduvad. Tippu, millel järglased puuduvad nimetame *terminaaltipuks*.

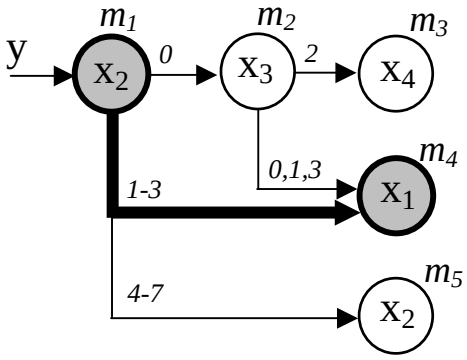
Joonisel 3.10 on toodud järgnevas kirjeldatud OD graafiline esitusviis:



Joonis 3.10. Otsustusdiagrammi graafiline esitus

$G_y=(M,E,X,\Gamma)$,
$M=\{m_1, m_2, m_3, m_4, m_5\}$,
$E=\{e_1, e_2, e_3, e_4, e_5\}$, $e_1=(m_1, m_2)$, $e_2=(m_1, m_4)$, $e_3=(m_1, m_5)$, $e_4=(m_2, m_3)$, $e_5=(m_2, m_4)$,
$X(m_1)=X(m_5)=(x_2,\{0,1,2,\ldots,7\})$, $X(m_2)=(x_3,\{0,1,2,3\})$, $X(m_3)=(x_4,?)$, $X(m_4)=(x_1,?)$,
$\Gamma(e_1)=\{0\}$, $\Gamma(e_2)=\{1,2,3\}$, $\Gamma(e_3)=\{4,5,6,7\}$, $\Gamma(e_4)=\{2\}$, $\Gamma(e_5)=\{0,1,3\}$.

Simuleerimine OD mudelil toimub järgnevalt. Vaadelgem olukorda, kus kõigile tipumuutujatele on omistatud mingi väärtus. Vastavalt nendele väärtustele on aktiveeritud iga (mitte-terminaalse) tipu mingi väljundkaar. Sellised üksteisele järgnevad aktiveeritud kaared moodustavad omakorda *aktiveeritud tee*. Iga muutujaväärtuste kombinatsiooni korral eksisteerib alati aktiveeritud tee, mis viib graafi algustipust mingisse terminaaltippu. Sellist teed nimetatakse graafi *peamiseks aktiveeritud teeks*.



$x_1 = 4$
$x_2 = 2$
$x_3 = ?$
$x_4 = ?$

$y = 4$

Joonis 3.11. Simuleerimine otsustusdiagrammi mudelil

Joonisel 3.11 on näidatud simuleerimisprotsess otsustusdiagrammil jooniselt 3.9. Olgu muutuja x_2 väärtuseks 2. Selle tulemusena aktiveeritakse tee (tähistatud jämedate nooltega) algtipust m_1 terminaaltippu m_4, mida märgib muutuja x_1. Antud näiteks on x_1 väärtuseks 4, seega $y=x_1=4$.

Oluline on märkida, et selline simuleerimine on oma olemuselt sündmuspõhine, sest kõikide tippude läbisimuleerimise asemel vaatleme vaid tippe peamisel aktiveeritud teel (tähistatud halli värviga joonisel 3.11).

Kui kirjeldada süsteemi funktsionaalsust otsustusdiagrammide mudelil, siis üldjuhul kasutatakse otsustusdiagrammide süsteemi. Sel juhul arvutatakse mõnede otsustus-diagrammi tippe märkivate muutujate väärtused teiste süsteemi kuuluvate diagrammide poolt.

3.3.1.4 Meetodi rakendus OD mudelil

Järgnevas kirjeldame, kuidas alampeatükis 3.3.1.2 ära toodud üldist hierarhilise vigade simuleerimise printsiipi rakendada otsustusdiagrammide mudelil.

Protseduur 3. Viiakse läbi veavaba simuleerimine kõrgtaseme otsustusdiagrammide mudelil. (Simuleerimisprotsess on kirjeldatud alampeatükis 3.3.1.3).

Vaatleme nüüd vigade levitamist kõrgemal abstraktsioonitasemel. Koosnegu digitaalsüsteem komponentidest. Tähistame komponentide funktsioonide hulka F, ning seome eri komponentide sisendid väljundid muutujate hulgaga Z. Olgu süsteemi komponent funktsiooniga $z = f(z_1, z_2, \ldots, z_n) = f(Z'),\ Z' \subseteq Z$, mida kirjeldab otsustusdiagramm G_z. Me lähtume faktist, et vead võivad levida kõikidesse muutujatesse $z_i \in Z'$, mis märgivad diagrammi tippe. Igale otsustusdiagrammi tipule m, mida märgib muutuja $z(m)$, vastab kompleksvektor

$$D_{z(m)} = \{P_{z(m),0}, (P_{z(m),1},R_{z(m),1}), \ldots, (P_{z(m),kz},R_{z(m),kz})\}.$$

Sellest tulenevalt on vigade hulk $R_{z(m)} = R_{z(m),1} \cup \ldots \cup R_{z(m),k}$ levitatud tippu m.

Vaadelgem veasimuleerimist otsustusdiagrammile G_z järgmiste protseduuride kogumina.

Protseduur 4. Esiteks viiakse graafis läbi veavaba simuleerimine vastavalt protseduurile 3 ning arvutatakse veavaba väärtus $z=z(m^{T,0})$ jaoks, kus $m^{T,0}$ on terminaaltipp peamisel aktiveeritud teel.

Märgime graafi tippude hulka peamisel aktiveeritud teel algtipust kuni tipuni m (m välja arvatud) tähistusega $M_{FF}(m)$. Olgu R_{FF} (m) vigade hulk, mis on levitatud tippudesse $n \in M_{FF}(m)$. Veavabal aktiveeritud teel olevasse tippu m jõudmise tingimuseks on vigade R_{FF} (m) puudumine. Tähistame $R_{CF}(m)$ abil vigade hulka, mis on kooskõlas aktiveeritud vigase teega algtipust m^0 tippu m. Tippude m jaoks veavabal teel saame $R_{CF}(m) = R - R_{FF}(m)$.

Olgu L otsustusdiagrammi kõigi simuleeritavate vigade hulk. Kõik tipud veavabal teel arvatakse hulka L. Selleks, et nende vigade jaok veasimuleerimist läbi viia, tuleb kasutada protseduuri 4 või 5. Protseduuri rakendamise tulemusena uuendatakse listi L. Veasimuleerimine lõpeb kui list L on saanud tühjaks.

Protseduur 5. Veasimuleerimine terminaaltipule $m^{T,0} \in L$ funktsiooniga $z = z(m^{T,0}) = f(z_1,\ldots, z_p)$ kompleksvektoril $D = (D_1,\ldots, D_p)$,

$D_i = \{P_{i,0}, (P_{i,1},R'_{i,1}), \ldots, (P_{i,ki},R'_{i,ki})\}$, $i = 1,2, \ldots, p$,

kus

$$\forall i,j:\ R'_{i,j} = (R_{i,j} - R_{FF}(m^{T,0})) \cap R_{CF}(m)$$

on ekvivalentne Protseduuriga 1 kõrgtaseme mooduli simuleerimiseks (vt. Alampeatükk 3.3.1.2).

Protseduur 6. Mitteterminaaltipu $m \in L$ veasimuleerimine muutujaga $z(m)$ kompleksvektoril

$D_{z(m)} = \{P_{z(m),0}, (P_{z(m),1},R'_{z(m),1}), \ldots, (P_{z(m),km},R'_{z(m),km})\}$

kus

$$\forall j:\ R'_{z(m),j} = (R_{z(m),j} - R_{FF}(m)) \cap R_{CF}(m),$$

seisneb järgnevas:

a) kui m on veavabal teel ja $R'_{z(m)} = R'_{z(m),1} \cup \ldots \cup R'_{z(m),km} = \varnothing$ siis ühtki tippu listi L ei lisata

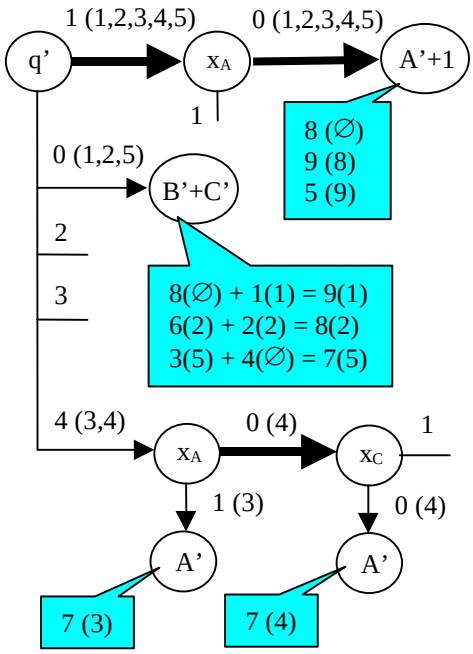
b) kui m ei ole veavabal teel ja $R'_{z(m)} = \varnothing$, siis tipp m^e, kus $e = P_{z(m),0}$, lisatakse listi L; uue tipu m^e jaoks leiame:

$$R_{FF}(m^e) = R_{FF}(m) \cup R_{z(m}e_{)},$$
$$R_{CF}(m^e) = R_{CF}(m),$$

c) kui $R'_{z(m)} \neq \varnothing$, siis kõik tipud m^e, kus $e = P_{z(m),i}$, i: $R'_{z(m),i} \neq \varnothing$, haaratakse listi L; kõigi nende tippude jaoks arvutame

$$R_{CF}(m^e) = R_{CF}(m) \cap R'_{z(m),i},$$

$R_{FF}(m^e) = R_{FF}(m)$.Veasimuleerimise protseduuride 6 ja 7 tulemusena saame kompleksvektori graafi muutujale z: $D_z = \{P_{z,0}, (P_{z,1},R_{z,1}), \ldots, (P_{z,kz},R_{z,kz})\}$. Kõik paarid $(P_{z,i},R_{z,i})$ kus $P_{z,i} = P_{z,0}$ on elimineeritud kuna vead $R_{z,i}$ on selles staadiumis maskeerunud. Kõik paaride grupid $\{(P_{z,i},R_{z,i}), (P_{z,j},R_{z,j})\}$ kus $P_{z,i} = P_{z,j}$ liidetakse üheks paariks $(P_{z,i},R_{z,i})$ kus $R_{z,i} = R_{z,i} \cup R_{z,j}$.



Joonis 3.12. Hierarhiline veasimuleerimine otsustusdiagrammil

Näide 4. Vaatleme otsustusdiagrammi joonisel 3.12 kompleksvektoritel:
D_q = {1, 0 (1,2,5), 4 (3,4)},
D_{xA} = {0, 1 (3,5)},
D_{xC} = {1, 0 (4,6)},
D_A = {7, 3 (4,5,7), 4 (1,3,9), 8 (2,8)},
D_B = {8, 3 (4,5), 4 (3,7), 6 (2,8)},
D_C = {4, 1 (1,3,4), 2 (2,6), 5 (6,7)}.

Joonis kirjeldab otsustusdiagrammi fragmenti – näidatud on vaid need teed, mis veasimuleerimisel läbitakse. Veavaba vektori poolt aktiveeritud kaared on joonisel tähistatud jämeda joonega. Kaared on tähistatud paaridega (*e,R*) kus *e* on kaare aktiveeriv tipumuutuja väärtus ja *R* on alamhulk vigade hulgast $R_{FF}(m)$ järgmise tipu *m* jaoks veavabal teel ning alamhulk vigade hulgast $R_{CF}(m)$ järgmiste tippude *m* jaoks vigastel teedel. Kuna $R_{FF}(x_A)$ = {1,2,3,4,5,6} sisaldab kõiki vigu, mis on levitatud muutujani x_A, siis vigaseid teid ei simuleerita tipust x_A: R'_{xA} = (R_{xA} - R_{FF} (x_A)) = ∅. Kõikidest vigadest, mis levisid muutujani *A'*, simuleeritakse tipus *A'*+1 ainult vead 8 ja 9. Terminaaltipus *B*'+*C*' simuleeritakse vaid vigu 1,2,5, kuna ainult need vead on kooskõlas liikumisega tipust *q'* antud suunas.

Peale kõigi antud kompleksvektori puhul läbitud 4 terminaaltipul läbi viidud vigade simuleerimist arvutame me lõpptulemuse järgnevalt. Viga 2, mis levib tippu *B'+C'* maskeerub kuna väärtus *B'+C'*=8, mis leiti vea 2 jaoks on võrdne veavaba väärtusega tipus *A*'+1. Vead 3, 4 ja 5, mis levivad erinevatesse otsustusdiagrammidese liidetakse ühte gruppi, sest kõigi nende vigade korral saadakse graafimuutujale *A* võrdne vigane väärtus 7. Samuti vead 1 ja 8 liituvad üheks grupiks. Seega on A lõpliku kompleksvektori väärtus: D_A = {8, 5(9), 7 (3,4,5), 9 (1,8)}.

Moodulite vigade simuleerimiseks madalal tasemel kasutatakse MTOD mudelit (vt. peatükk 3.2.2). Vaadelgem veasimuleerimist otsustusdiagrammile G_z, mis kirjeldab binaarset funktsiooni $z = f(z_1, z_2, \ldots, z_p) = f(Z'), Z' \subseteq Z$ etteantud kompleksvektoril $D = \{P_0, (P_1,R_1), \ldots, (P_n,R_n)\}$ järgmiste protseduuride kogumina.

Esiteks simuleeritakse veavaba tee veavaba vektori P_0 jaoks vastavalt protseduurile 4 ja arvutatakse muutuja $z = z(m^{T,0})$ veavaba väärtus, kus $m^{T,0}$ on terminaaltipp, milleni jõutakse tee aktiveerimisel (binaarsete OD-de puhul $z(m^{T,0}) \in \{0,1\}$). Tähistame kõiki tippe, mis asuvad

veavabal aktiveeritud teel, välja arvatud terminaaltipp $m^{T,0}$ tähistusega M_{FF}.

Protseduur 7. Kõigi tippude $m \in M_{FF}$ jaoks viiakse läbi veasimuleerimine: väärtus $z(m)=e$ inverteeritakse ning aktiveeritakse uus tee tipust m vastavalt inverteeritud väärtusele. Kui niiviisi toimides jõuame ikka samasse terminaaltippu $m^{T,0}$, siis viga $z(m)\equiv e$ ei avastata. Vastasel korral loeme vea avastatuks.

Protseduur 8. Järgnevalt simuleeritakse kõigi vigade $r \in R_1 \cup \ldots \cup R_n$ jaoks vigased vektorid P_i , $i = 1,\ldots,n$. Iga viga $r \in R_i$ viiakse kõigepealt skeemi siise ning seejärel viiakse läbi veavaba simuleerimine vektoril R_i. Oletame, et jõuti terminaaltippu $m^{T,r}$. Juhul kui $z(m^{T,r}) \neq z(m^{T,0})$, siis viga r jääb levitatavate vigade listi, vastasel korral aga mitte.

Käesolevas alampeatükis kirjeldatud meetod on plaanis programmina realiseerida projekti teise aasta jooksul.

3.3.2 Hierarhilisel OD mudelil põhinev testigenereerimisalgoritm (Ülesanne 10)

3.3.2.1 Sissejuhatus

Testide genereerimine digitaalseadmetele on äärmiselt keeruline matemaatiline probleem. Kombinatsioonskeemide puhul on testigenereerimise keerukus halvimal juhul eksponent-siaalses sõltuvuses skeemi sisendbittide arvust. Järjestikskeemide (s.t. mäluga skeemide) puhul on tegemist veelgi komplitseerituma ülesandega, mis pole tänaseni praktilist lahendust leidnud.

Tänapäeva digitaalskeemide suure keerukuse tõttu on traditsioonilised loogikalülituste tasemel töötavad testigenereerimismeetodid muutunud praktiliselt kõlbmatuks. Lahendusena on pakutud kõrgtasemel töötavaid testigeneraatoreid, millede puuduseks on aga ebatäpsus, mis on põhjustatud asjaolust, et nad ei arvesta skeemi madala taseme struktuuriga. Kuldseks keskteeks on siin hierarhiliste testigenereerimismeetodite rakendamine.

Hierarhilises testigenereerimises eristatakse alt-üles ja ülalt-alla lähenemist. Alt-üles meetodi puhul genereeritakse kõigepealt testvektorid moodulitele, mis on esitatud madalal tasemel. Seejärel kompileeritakse need testid kõrgtasemel kogu seadme globaalseks testiks. Selline lähenemine ignoreerib nn. täielikkuse probleemi – eelnevalt genereeritud moodulite testvektorid võivad olla vastuolus kitsendustega, mis tulenevad teistest moodulitest või süsteemi struktuurist.

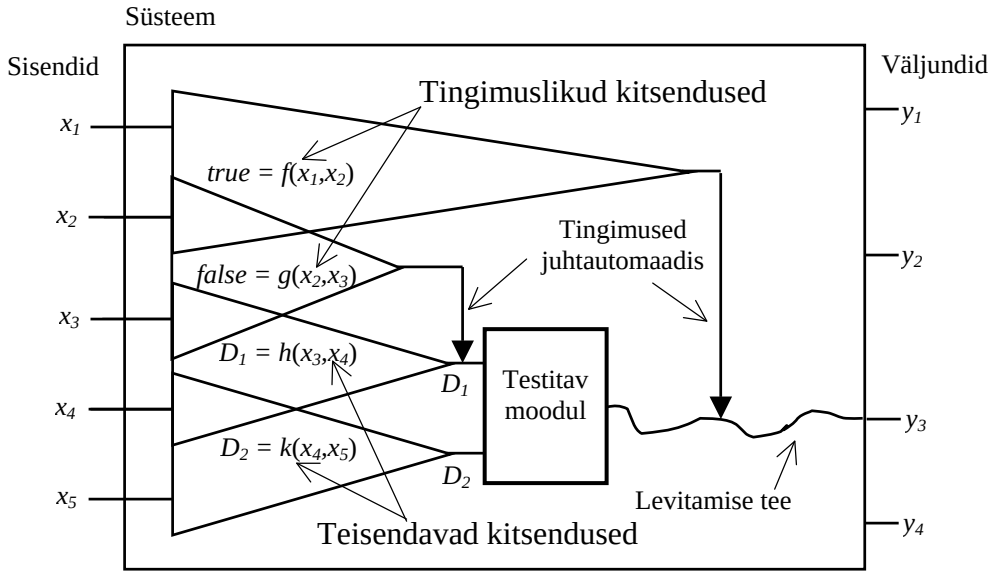
Ülalt-alla lähenemise puhul ekstraheeritakse kõigepealt kõrgtaseme funktsionaalsusest tulenevad kitsendused, mis võetakse seejärel aluseks süsteemi moodulite testimisel madalal tasemel. Sellise lähenemise puuduseks on, et teda ei saa kasutada, kui moodulite madaltaseme kirjeldus pole teada. Käesolevas projektis valisime me aluseks ülalt-alla lähenemise.

3.3.2.2 Vea levitamine ja kitsenduste mõiste

Järgnev algoritm töötab kahetasemelisel otsustusdiagrammide mudelil, kuid tema kirjeldus on arusaadavust silmaspidades esitatud mudelist sõltumatuna. Vea levitamise protseduuri eesmärgiks on aktiviseerida süsteemi juhtautomaadi olekujada, mis levitaks veaefekti süsteemi vaadeldava mooduli väljundist süsteemi globaalsesse väljundisse. Käesolevas algoritmis rakendatakse levitamist mööda ühte teed ning veaefekti levimise üle peetakse arvet, kasutades selleks spetsiaalset viita, mida nimetame *veaviidaks*.

Vea levitamise käigus toimub ka kõrgtaseme tee aktiviseerimise kitsenduste ekstraheerimine. Käesolevas lähenemises eristatakse kaht tüüpi kitsendusi: *tingimuslikud kitsendused* ja *teisendavad kitsendused*. Tingimuslikud on kitsendused, mis esitavad süsteemi juhtautomaadi töös esinevatel hargnemistel tehtud valikuid. Teisendavad kitsendused on esitatud avaldisena, mis kirjeldab andmete muutumist levimisel läbi süsteemi moodulite.

Joonisel 3.13 on toodud lihtsustatud näide eri tüüpi kitsenduste rollist hierarhilises testigenereerimises.



Joonis 3.13. Tingimuslikud ja teisendavad kitsendused hierarhilisel testigenereerimisel

Otsustusdiagrammidel põhinev hierarhiline testigenereerimisalgoritm genereerib testi igale OD mudeli tipule. Mudeli terminaaltippude ja süsteemi registersiirde taseme operaatormoodulite vahel on üks-ühene vastavus. Mitteterminaalsete tippude ja süsteemi moodulite vahel analoogset vastavust aga pole. Sellegipoolest garanteerib OD mudeli tippude 100 % testi kate ka täieliku katte struktuursel tasemel.

Testigenereerimisalgoritmi [3] esimeseks etapiks on testi püstitus. Püstituse käigus aktiveeritakse tee otsustusdiagrammis testitavasse tippu ning seatakse veaviit viitama diagrammi muutujale. Märgitakse ära, et testitava tipu muutuja väärtus tuleb kinnitada (s.t. levitada sisenditeni). Mitteterminaalse tipu testi püstituse täiendavaks tingimuseks on lisaks, et tipu järglasterminaaltipude muutujate väärtused tuleb kinnitada ning üksteisest eristada.

Peale testi püstitust on järgmiseks operatsiooniks veaefekti levitamine testitava mooduli väljundist süsteemi väljundisse. Järgnevas on kirjeldatud vea levitamise algoritm pseudoprogrammina.

Algoritm 1.

Viime läbi testi püstituse moodulile *M*
Loome teisendavad kitsendused mooduli sisenditest $input_i(M)$
Paneme veaviida viitama mooduli väljundile *output(M)*

While veaviit ei viita süsteemi väljundile
  Olgu *a* skeemi signaal, millele viitab veaviit
  Valime signaalile *a* järgneva registri *b*
  Omistame juhtautomaadi juhtsignaalidele väärtused, mis on vajalikud
      andmete levitamiseks signaalist *a* signaali *b*
  Seame veaviida viitama signaalile *b*

  If leidub sobilik juhtautomaadi väljundvektor ***then***
    Valime väljundvektori, mis on kooskõlas tehtud signaaliomistustega
    Loome vastavad tingimuslikud kitsendused
    If *b* on register ***then***
      Liigume järgmisesse takti
    Endif
  Else
    Võtame tagasi viimase valiku
  Endif

Endwhile

3.3.2.3 Väärtuste kinnitamisülesanne

Vealevitamisele järgneb kitsenduste kinnitamisülesanne. Kõik kitsendustes esinevad muutujad tuleb kinnitada kuni vastavates avaldistes saavad olema ainult süsteemi sisendmuutujad. Kinnitamisülesanne liigub süsteemi funktsioneerimise mõttes ajas tagasi, alustades taktist, mil levitamisülesanne töö lõpetas. Kinnitamisprotsessi käigus kitsendusi kirjeldavaid avaldusi uuendatakse, asendades nendes leiduvad muutujad uute avaldiste või muutujatega. Samuti luuakse kinnitamise käigus üldjuhul uusi tingimuslikke kitsendusi. Nimetame kõiki süsteemi signaale, mis vastavad süsteemi sisenditele kinnitatud signaalideks, ülejäänud signaale aga kinnitamata signaalideks. Kitsenduste uuendamist võib vaadelda kui kinnitamata signaalide superponeerimist uute signaalide või nende avaldistega, mis määratakse juhtautomaadi poolt seadmes aktiveeritud poolt.

Peale kinnitamisülesaanet kitsendused lahendatakse. Selleks otstarbeks on olemas standardsed meetodid.

Järgnevas on esitatud kinnitamisülesande algoritm:

Algoritm 2:

While eksisteerib kinnitamata kitsendusi
  If praegune takt on varasem testi püstitusest **then**
    Valime mingi kitsenduse kinnitamata signaali b
    Valime signaalile *b* eelneva registri *a*
    Omistame juhtautomaadi juhtsignaalidele väärtused, mis on vajalikud
        andmete levitamiseks signaalist *b* tagasi signaali *a*
    If leidub sobilik juhtautomaadi väljundvektor **then**
      Valime väljundvektori, mis on kooskõlas tehtud signaaliomistustega
      Loome vastavad tingimuslikud kitsendused
        If b on register **then**
          Liigume eelmisesse takti
        Endif
    Else
        Võtame tagasi viimase valiku
    Endif
  Else
        Liigume eelmisesse takti
  Endif
  Uuendame kõiki aktiivseid kitsendusi
Endwhile
Lahendame kitsendused

3.3.2.4 Algoritmi eelised

Käesolevas alampeatükis kirjeldatud hierarhilisel otsustusdiagrammidel põhineval algoritmi kolm peamist eelist traditsioonilise lähenemisega võrreldes:

1) Universaalsus, üldisus

Erinevalt olemasolevatest meetoditest on käesoleva lähenemise puhul juht- ja operatsioon-automaati käsitletud ühtse mudelina kasutades otsustusdiagramme. Ka kõrgtaseme ja madaltaseme käsitle,ine toimub otsustusdiagrammidel.

2) Lihtsus, kiirus

Kuna levitamine toimub mööda ühte teed ja kuna kitsendustele pole rakendatud läbipaistvuse tingimusi on välja pakutud meetod oluliselt kiirem ning suurema lubatud lahendite hulgaga kui senipakutud lahendused.

3) Parem defektide kate

Varasemas avaldatud hierarhilised algoritmid genereerivad teste vaid süsteemi andmeosa funktsionaalsetele moodulitele. Käesolev algoritm kasutab veamudelit, mis katab 100 % andmeosa riketest, hõlmates nii funktsionaalsete moodulite rikkeid (testi püstitus terminaaltippudele) kui ka registrite ning multiplekserite rikkeid (mitteterminaaltippude test).

3.3.3 Isetesti emulaatori realisatsioon (Ülesanne 11)

Käesolevas alampeatükis kirjeldame isetesti emulaatori tarkvara realisatsiooni. Peamisteks argumentideks kiipi sisseehitatud isetestimise rakendamiseks käesolevas projektis on järgmised.

1) Mõned kiibisisesed moodulid võivad olla välise testri jaoks raskesti ligipääsetavad.

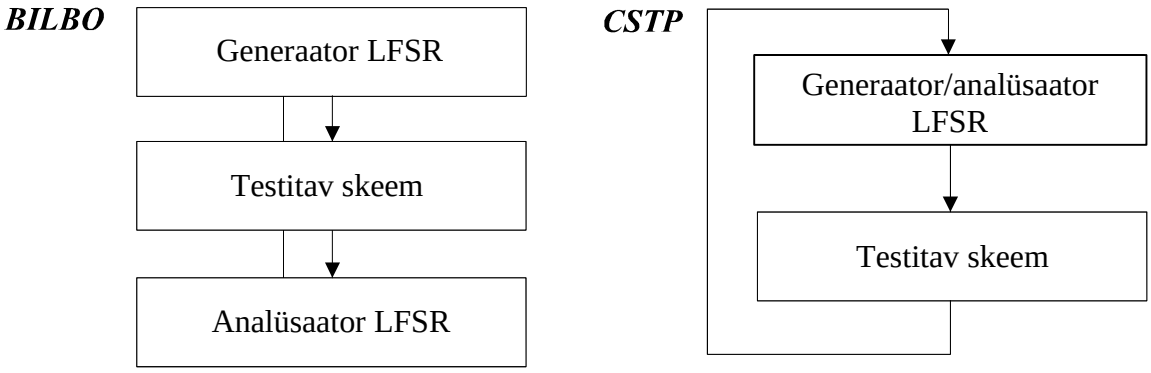
2) Isetestimine lubab testida kiipi tema enda töösagedusel, sest kasutatakse kiibi tehnoloogiat, samas kui välised testseadmed üldjuhul esindavad eilset tehnoloogiat. Kiibi test madalamal sagedusel kui töösagedus võib põhjustada selles esinevate võimalike viitevigade mitteavastamist testi käigus.

3) Kuna isetestimise puhul rakendatakse testi kiibi enda vahendeid kasutades, säästab see ülikallist välise testri mälu.

4) Isetestimine võimaldab kiipi testida tema erinevates tootmis- ja rakendusstaadiumides: tootmisliinil, trükkplaadil ning ka jooksvalt, kiibi töö käigus.

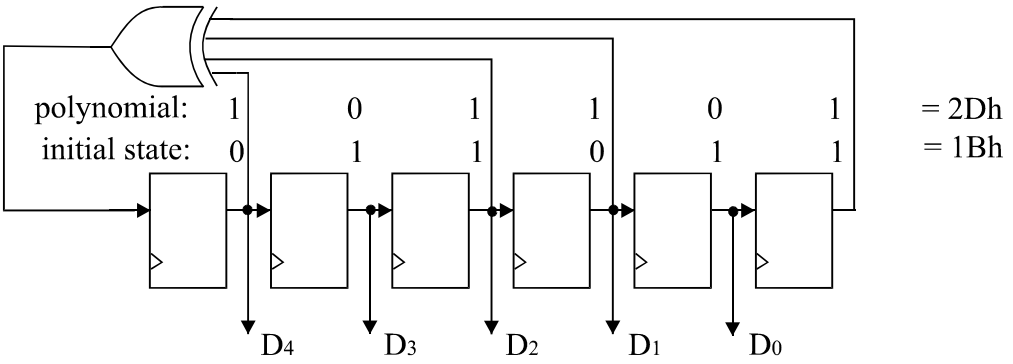
3.3.3.1 Emulaatori poolt toetatavad arhitektuurid

Projekti raames realiseeritud isetesti emulaator toetab kahte tüüpi isetesti arhitektuure: BILBO (Built-In Logic Block Observer) ja CSTP (Circular Self-Test Path). BILBO arhitektuuri puhul on tegemist eraldi vektori generaatori ja tulemuste analüsaatoriga, samas kui CSTP kasutab kombineeritud generaator-analüsaatorit. BILBO eeliseks on, et genereeritavad pseudojuhuslikud vektorid ei sõltu testitava skeemi väljundväärtustest. CSTP pluss on väiksem nõudlus täiendava ränipindala järele kuna kasutatakse vaid üht LFSR (Linear Feedback Shift Register) tüüpi nihkeregistrit. Joonisel 3.14 on näidatud toetatavad isetesti arhitektuurid.



Joon. 3.14. Toetatavad isetesti arhitektuurid

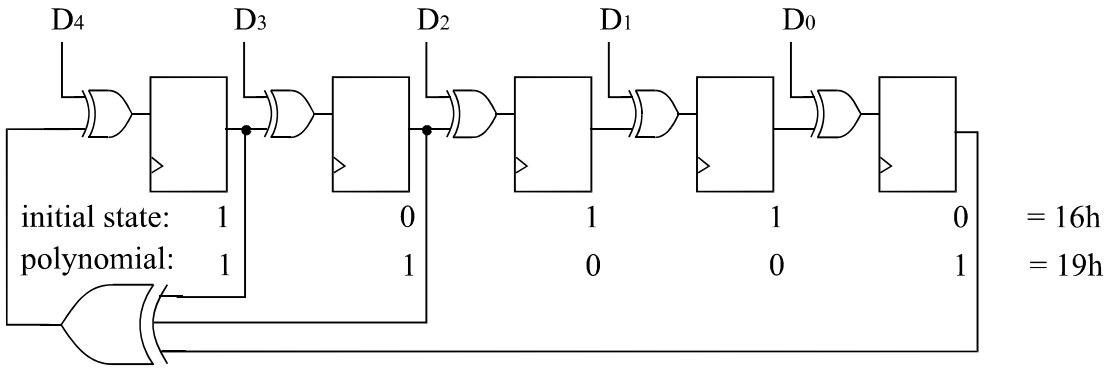
Realiseeritud isetesti emulaator võimaldab kasutajal sisestada tagasiside polünoome LFSR nihkeregistrite jaoks ning samuti nende algolekuid. Spetsifitseerida tuleb ka vastavate registrite siinilaius. Joonisel 3.15 on toodud näide generaator LFSR registrist. Antud näites on generaatori algolekuks 011011 ning tagasiside polünoomiks 101101.



Joonis 3.15 Generaator LFSR register

Näites on LFSR registri siinilaius (6 bitti) suurem kui testitava skeemi sisendite arv (5 sisendit). Sel juhul ühendatakse testitava skeemi sisendid generaatori registri vanemate bittidega (vt. joonis). Generaatori siinilaius peab olema suurem või võrdne skeemi sisendite arvuga.

Joonisel 3.16 on esitatud näide analüsaatori LFSR registrist, mille algolekuks on 10110 ja tagasiside polünoomiks 11001.



Joonis 3.16 Analüsaator LFSR register

Sarnaselt generaatoriga, kui analüsaatori siinilaius on suurem kui skeemi väljundite arv, siis väljundid ühendatakse vaikimisi LFSR registri vanemate bittide poole. Seda saab muuta kasutades käsurea võtmeid (vt. lisa 2). Analüsaatori siinilaius peab alati olema suurem või võrdne skeemi väljundite arvuga.

Realiseeritud isetesti emulaatori detailsem funktsionaalsus käsurea parameetrite näol on ära toodud lisas 2.3.

4 Kokkuvõte ja eesseisvad ülesanded

Käesoleva rakendusuuringu oodatavateks tulemusteks on:

- uus tarkvara isetestivate digitaalsüsteemide projekteerimiseks, mis koosneb järgmistest komponentidest: hierarhiline rikete simulaator, hierarhiline testide generaator, isetestimis-arhitektuuride kvaliteedi analüüsi- ja optimeerimisprogrammid
- metoodika väljatöötamine isetestivate mikroprotsessorite projekteerimiseks ja selle rakendamine reaalsel tootel
- uut tüüpi kõrgekvaliteedilise isetestiva võrguprotsessori väljatöötamine ja projekteerimine

Rakendussuuringu kestuseks on kaks aastat. Esimese projektiaasta jooksul läbi viidud töö on kirjeldatud käesolevas aruandes. See hõlmab järgmisi ülesandeid:

- Krüptograafilise võrguprotsessori prototüüpriistvara loomine programmeeritavate loogikamaatriksite platvormil.
- Prototüüpkiibi struktuurianalüüs ja isetestimise metodoloogia väljatöötamine.
- Diagnostikavahendite skeemiliidesed projekteerimistarkvaraga.
- Hierarhiliste diagnostikaalgoritmide väljatöötamine.
- Isetesti diagnostikalahenduse realisatsioon.

Rakendusuuringu teise aasta põhiülesannetena on plaanis:

- Toota projekteeritav kiip ränis ning viia läbi kiibi test.
- Realiseerida programmid hierarhiliseks veasimuleerimiseks ja testigenereerimiseks
- Töötada välja isetesti struktuurid kiibile

Kasutatud kirjandus

[1] A.Schneider, E.Ivask, P.Miklos, J.Raik, K.H.Diener, R.Ubar, T.Cibáková, E.Gramatová. Internet-based Collaborative Test Generation with MOSCITO. IEEE Proc. of Design Automation and Test in Europe - DATE02, pp. 221-226, Paris, March 4-8, 2002.

[2] R.Ubar, J.Raik, E.Ivask, M.Brik. Multi-Level Fault Simulation of Digital Systems on Decision Diagrams, IEEE Workshop on Electronic Design, Test and Applications - DELTA02, pp.86-91, Christchurch, New Zealand, 29-31 January 2002.

[3] J.Raik. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams, PhD thesis, TTU press, Tallinn 2001.

[4] Xilinx virtex II datasheet, Xilinx Inc.

[5] Top level architecture description. ADG documentation

[6] Verification PCB block description. ADG documentation

[7] Verification PCB schematic. ADG documentation

[8] IETF ipsec standard documents

Lisa 1. Rakendusuuringu etapid ja ülesanded

Rakendusuuringu sisuks on uut tüüpi tarkvara väljatöötamine mikroprotsessorsüsteemide rikete simuleerimiseks, isetestimisarhitektuuride projekteerimiseks ning selle tarkvara rakendamine konkreetse uut tüüpi isetestiva mikroprotsessorsüsteemi projekteerimisel. Rakendusuuringu ülesannete nimistu on esitatud peatükis L1.1. Projekti ajagraafik on ära toodud diagrammina punktis L1.2 ning vahetulemuste (kontrollpunktide) kirjeldus on esitatud punktis L1.3.

L1.1 Ülesannete kirjeldus

AS Artec Design Group ülesanded

Ülesanne 1. Spetsifikatsiooni koostamine
Kestvus: 3 kuud
Määratakse süsteemi nõuded riist- ja tarkvarale ning koostatakse kiibi kõrgtaseme algoritmiline mudel. Koostatakse verifikatsiooniplatvormi spetsifikatsioon.

Ülesanne 2. Verifitseerimiskeskkonna väljatöötamine
Kestvus: 4 kuud
Kiip sünteesitakse prototüüpimiseks programmeeritavate loogikamaatriksite (FPGA) arhitektuuril. Prototüüpimise trükkplaadi tootmine, komponentide paigaldus. Testitakse prototüüp-FPGA funktsionaalsust.

Ülesanne 3. Tarkvara ja riistvara koosarendus ja test
Kestvus: 5 kuud
Tarkvara ja riistvara koosarendus ja test. Projekteerimisvigade parandamine.

Ülesanne 4. Kiibi füüsikalise taseme süntees
Kestvus: 5 kuud
Sünteesitakse transistorite paigutus ränil ja trasseeritakse nendevahelised ühendused. Läbirääkimised ränitootjaga. Valideerimisplatvormi arendus.

Ülesanne 5. Isetestiva kiibi tootmine ränil
Kestvus: 2 kuud
Kiibi kirjeldusandmete ettevalmistamine ja saatmine tehasesse, kiibi tootmine ja eeltestimine tehases. Valideerimisplatvormi koostamine. (Luuakse trükkplaat ning tarkvara toodetud kiibi testimiseks reaalses süsteemis).

Ülesanne 6. Kiibi test süsteemis
Kestvus: 3 kuud
Kiibi testimine süsteemis (arvutivõrk) toetatavatel operatsioonisüsteemidel (Windows, Linux) ja rakendustel (ruuterid, tulemüür jne.). Valideerimisplatvormi tootmine.

Ülesanne 7. Rakendatud testimismeetodite efektiivsuse hindamine
Kestvus: 2 kuud
Hinnang rakendatud testimismeetodite efektiivsusele ja majanduslikule tasuvusele toote arenduses.

Tallinna Tehnikaülikooli ülesanded

Ülesanne 8. Kiibi arhitektuuri analüüs
Kestvus: 3 kuud
Projekteeritava võrguprotsessori arhitektuurianalüüs. Üldise testi strateegia defineerimine kiibi moodulite testiks.

Ülesanne 9. Veasimuleerimisalgoritmi väljatöötamine
Kestvus: 2 kuud
Algoritmi väljatöötamine kiireks hierarhilisel OD mudelil põhinevaks veasimuleerimiseks.

Ülesanne 10. Testigenereerimisalgoritmi väljatöötamine
Kestvus: 4 kuud
Algoritmi väljatöötamine kiireks hierarhilisel OD mudelil põhinevaks testide genereerimiseks.

Ülesanne 11. Isetesti emulaatori väljatöötamine
Kestvus: 3 kuud
Algoritmi väljatöötamine isetesti struktuuride emuleerimiseks. Vastava emulaatori programmeerimine.

Ülesanne 12. Kõrgtaseme simulaatori realisatsioon
Kestvus: 2 kuud
Programmeeritakse kõrgtaseme OD mudelil töötav simulaator.

Ülesanne 13. Madaltaseme veasimulaatori realisatsioon
Kestvus: 3 kuud
Programmeeritakse madaltaseme OD mudelil töötav vigade simulaator. Kahe eelpoolnimetatud simulaatori baasil luuakse hierarhiline keskkond, mis võimaldab kiiret veasimuleerimist keerukatele digitaalsüsteemidele.

Ülesanne 14. Hierarhilise testigeneraatori realisatsioon
Kestvus: 4 kuud
Programmeeritakse testigeneraator keerukate digitaalskeemide hierarhiliseks testiks.

Ülesanne 15. Kiibi testimine projekti lahendustel
Kestvus: 3 kuud
Mõõdetakse isetesti ning projekteerija poolt skeemi funktsionaalsuse kontrolliks ette nähtud vektorite kvaliteeti tootmisdefektide katmisel kasutades projekti raames väljatöötatud veasimuleerimis-tarkvara. Genereerimitakse testid funktsionaalsete vektorite ning isetesti poolt avastamata riketele.

Ülesanne 16. Kommertsiaalse testilahenduste rakendamine
Kestvus: 3 kuud
Hõlmab skanneeritava tee sünteesi kiibi moodulitele kasutades kommertsiaalset testitarkvara. Antakse hinnang saavutatud veakattele, testi kvaliteedile ja pikkusele.

Ülesanne 17. EDIF liidese realisatsioon
Kestvus: 3 kuud
Programmeeritakse liides, mis võimaldab skeemi loogikataseme EDIF kirjeldusi konverteerida madaltaseme otsustusdiagrammide mudeliks.

Ülesanne 18. VHDL liidese realisatsioon
Kestvus: 4 kuud
Programmeeritakse liides, mis võimaldab skeemi kõrgtaseme VHDL kirjeldusi konverteerida otsustusdiagrammide mudeliks.

Ülesanne 19. Kiibi moodulite konverteerimine OD mudelisse
Kestvus: 2 kuud
Genereeritakse kiibi moodulite jaoks vastavad kõrg- ja madaltaseme otsustusdiagrammide mudelid.

Ülesanne 20. Isetestimise struktuuride väljatöötamine kiibi moodulitele
Kestvus: 4 kuud
Kontrolleri projekteerimine kiibi isetesti käivitamiseks ja juhtimiseks. Isetestimise struktuuride projekteerimine kiibi moodulitele vastavalt P1500 standardile.

Ülesanne 21. Testide planeerimine ja paralleliseerimine
Kestvus: 3 kuud
Kiibi testi pikkuse minimiseerimine valides optimaalse testide järjestuse ning rakendades osa teste paralleelselt. Testi kogupikkus on oluline parameeter, kuna kiibi testimise hind tehases (testri aeg) on sellega proportsionaalne.

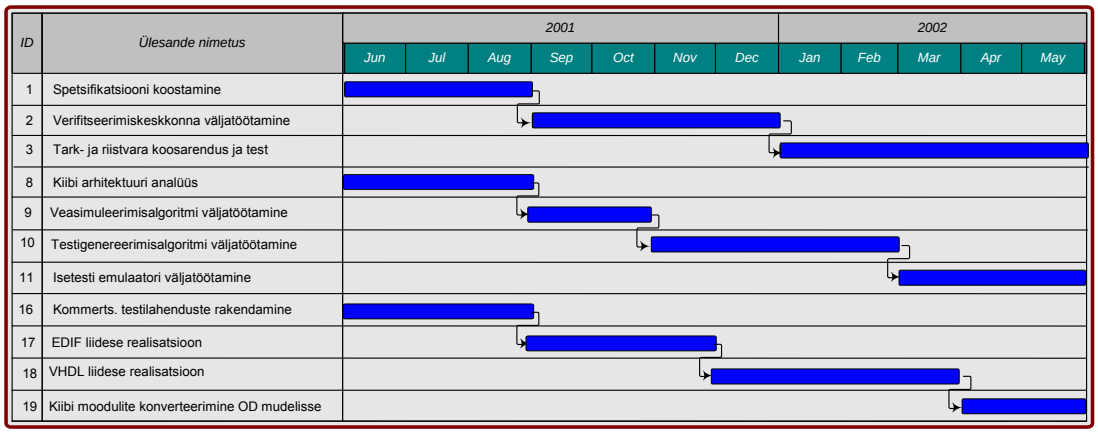
Ülesanne 22. Isetestimise struktuuride ränipindala võrdlus kommertsiaalsete struktuuride (skanneeritav tee) omaga
Kestvus: 2 kuud

Teatavasti sünteesiti süsteemi moodulitele täielik skanneeritav tee (vt. ül. 16). Sellise lahenduse poolt saavutatud parameetreid (täiendav ränipindala, veakate, testi pikkus) võrreldakse projekti raames rakendatuga ning antakse hinnang, kui palju oli projekti lahendus efektiivsem ja ökonoomsem.

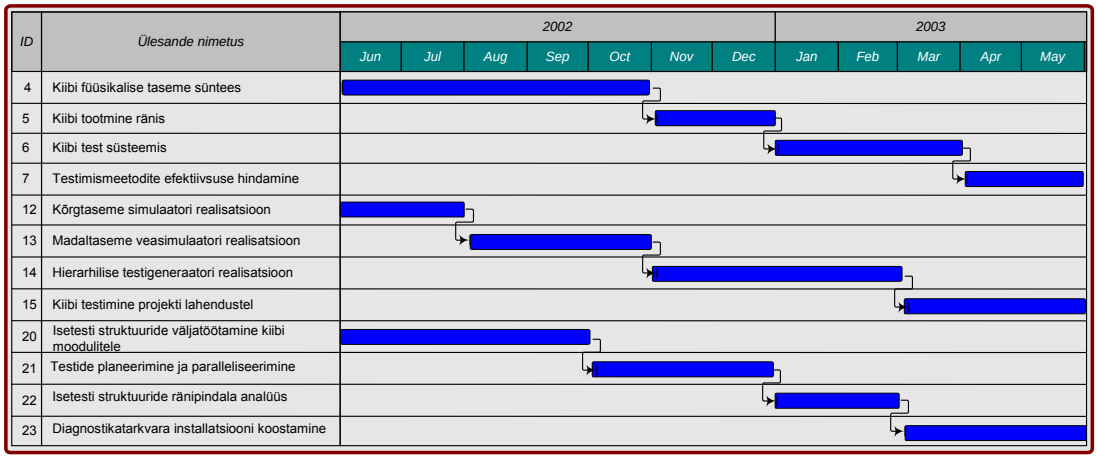
Ülesanne 23. Väljatöötatud diagnostikatarkvara installatsiooni koostamine
Kestvus: 3 kuud
Luuakse tarkvara installatsioon, mis võimaldab projekti raames loodud testimise programme ning dokumentatsiooni pakkuda kommertspaketina või vabavarana.

L1.2 Rakendusuuringu ajagraafik

Joonistel L1.1, L1.2 on välja toodud rakendusuuringu projekti ajagraafik Gantti diagrammina aastate kaupa.



Joon. L1.1. Rakendusuuringu projekti ajagraafik, 1. Aasta



Joon. L1.2. Rakendusuuringu projekti ajagraafik, 2. aasta

L1.3 Kontrollpunktid

Artec Design Group

1) Prototüüp-riistvara ja kiibi verifitseerimiskeskkond

Kirjeldus:
Isetestiva võrguprotsessori spetsifikatsiooni ja verifikatsioonikeskkonna väljatöötamine. Valmib prototüüp-riistvara loogikamaatriksite (FPGA) arhitektuuril.

Edastatav tulemus: prototüüp riistvara, vahearuanne.

Valmimistähtaeg: 1.6.2002

2) Uut tüüpi isetestiva kiibi tootmine väljatöötatud uue tarkvara kasutamisel

Kirjeldus:
Isetesti struktuuride integreerimine kiibi moodulitesse. Isetestiva integraalskeemi tootmine ränis. Hinnang rakendatud testimismeetodite efektiivsusele toote arenduses.

Edastatav tulemus: isetestiv integraalskeem, lõpparuanne

Valmimistähtaeg: 1.6.2003

Tallinna Tehnikaülikool

3) Prototüüpkiibi struktuurianalüüs ja isetestimise metodoloogia; diagnostikavahendite skeemiliidesed projekteerimistarkvaraga; diagnostikaalgoritmide väljatöötamine

Kirjeldus:
Konkreetse mikroprotsessorsüsteemi struktuurianalüüs ja isetestimise metodoloogia ning testimisarhitektuuride väljatöötamine. Riistvarakirjelduskeelte (EDIF, VHDL) projektis kasutatavate alamhulkade spetsifikatsioonide dokumenteerimine. Diagnostikavahendite skeemiliideste loomine projekteerimistarkvaraga. Algoritmide väljatöötamine
- rikete hierarhiliseks simuleerimiseks,
- testide genereerimiseks ja
- isetestimisarhitektuuride kvaliteedi analüüsiks keerulistes digitaalsüsteemides.

Edastatav tulemus: skeemiliideste programmid, vahearuanne

Valmimistähtaeg: 1.6.2002

4) Diagnostikatarkvara ning isetesti rakendus konkreetses tootes

Kirjeldus:
Uus diagnostikatarkvara. Isetesti realiseerimine toodetaval kiibil. Tarkvara ning isetesti jõudluse analüüs.

Edastatav tulemus: tarkvara, isetestistruktuurid, süsteemkiibi testimise metoodika kirjeldus, lõpparuanne

Valmimistähtaeg: 1.6.2003

Lisa 2. Tarkvaralahenduste käsuread

Järgnevas on ära toodud projekti esimese aasta jooksul valminud tarkvaraliste testilahenduste käsuread.

L2.1 MTOD liidese programmid

MTOD liides

programm: `import`

Konverteerib EDIF 2.0.0 loogikataseme skeeme MTOD ja ISCAS'89 formaati.

sisend: EDIF loogikataseme skeemi fail, tehnoloogia teek
väljund: MTOD mudeli fail (.agm) või ISCAS'89 fail (.cir)

süntaks: `import` *[options] <EDIF file> <library file>*
options:

`-iscas89`	Geneeritakse ISCAS'89 formaat.
`-read_iscas89`	Loetakse skeem ISCAS'89 formaadist. (Teeki pole tarvis lisada!)
`-tool` <*application*>	Lubatud *application* väärtused on `orcad` ja `cadence`.
`-clock` <*clock_input*>	*clock_input* taktsisendi nimi.
`-gnd` <*gnd_name*>	Ühendus *gnd_name* on ühendatud loogilise 0-nivooga.
`-vdd` <*vdd_name*>	Ühendus *vdd_name* on ühendatud loogilise 1-nivooga.

MTOD liidese veateated

Vigade korral EDIF kirjelduse ja tehnoloogia teegi parsimisel väljastab liides veateate või hoiatuse, milledega käib kaasas skeemifaili reanumber, kus viga esines. Hierarhiliste skeemide parsimine kutsub esile järgmist tüüpi hoiatusi, mida tuleks aga üldjuhul ignoreerida:

```
Design: COMPONENT_1
Parsing macro

Warning at line 1917: Gate not in library
```

See teade ütleb, et komponent nimega COMPONENT_1 leiti EDIF-i kirjeldusest ning tema jaoks ei leitud tehnoloogia teegist vastet. EDIF-i parser käsitleb sellist komponenti makrokomponendina ning laskub hierarhiatasemel allapoole, et selle loogikataseme struktuuri parsida ja jätkab seejärel oma normaalset tööd.

Tehnoloogia teegi generaator

programm: `libgen`

Genereerib MTOD liidesele tehnoloogia teegi.

sisend: Tehnoloogia teegi lähteformaadis esitatud fail (vt. lisa 3).
väljund: MTOD liidese tehnoloogia teek (.lib)

süntaks: `libgen` <*library source*> <*library file*>

L2.2 KTOD liidese programm

programm: `vhdl2dd`

Teisendab käituvuslikul registersiirde tasemel esitatud VHDL kirjeldused KTOD mudeliks.

sisend: Käituvuslik RST VHDL mudel, OD teek
väljund: Otsustusdiagrammi mudel väljastatakse faili nimega "*vhdl_lähtekirjeldus*.agm"
RST VHDL mudel väljastatakse faili "*vhdl_lähtekirjeldus*_rtl.vhdl"

süntaks:

`vhdl2dd` <*vhdl_lähtekirjeldus*> <*olekumuutuja_nimi*> <*reset_muutuja_nimi*>

<*vhdl_lähtekirjeldus*>	on faili nimi koos laiendiga.
<*olekumuutuja_nimi*>	on juhtautomaadi olekumuutuja nimi.
<*reset_muutuja_nimi*>	on reset signaali kirjeldava muutuja nimi.

L2.3 Isetesti emulaator

programm: `bist`

Isetesti arhitektuuride BILBO ja CSTP emulaator.

sisend: MTOD mudeli fail (.agm)
väljund: testvektorite fail (.tst)

süntaks:

`bist –gpoly` <*generator_poly*> `-ginit` <*generator_init*>
[`–apoly` <*analyzer_poly*> `-ainit` <*analyzer_init*>] [*options*] <*design*>

generator_poly: Generaatorregistri tagasiside polünoom binaarkujul.

generator_init: Generaatorregistri algväärtus binaarkujul.

.

analyzer_poly: Analüsaatorregistri tagasiside polünoom binaarkujul.

analyzer_init: Analüsaatorregistri algväärtus binaarkujul.

design: Skeemifaili nimi ilma .agm laiendita.

options:

`-aliasing`	Emuleerime arvestades vigade maskeerumist analüsaatorregistris.
`-simul < bilbo | cstp >`	Valime BILBO või CSTP arhitektuuri vahel.
`-count` <*cycles*>	Genereeritava testi pikkus taktides. Vaikimisi 1000.
`-optimize`	Jätame ära testi viimased vektorid, mis lisavigu ei avasta.
`-lsb`	Seome skeemi väljundid analüsaatorregistri nooremate bittidega.

Lisa 3. Loogikalülide teegi lähteformaat

Loogikalülide teegi lähteformaati kasutatakse tehnoloogiateekide genereerimisel madaltaseme otsustusdiagrammide (MTOD) mudeli liidesele. Teegi lähteformaat on kirjeldatud järgnevas.

`GATE` <*cellName*> <*BooleanExpression*> `;`

kus *cellName* on teegi komonendi nimetus ja
BooleanExpression võib esitada kui:

<*outputPin*> = <*BooleanFunction*>

kus *BooleanFunction* on omakorda teegi komonendi poolt realiseeritud Boole'i funktsioon komponendi sisenditest.

Kasutada võib järgmisi Boole'i operatsioone (prioriteetsuse järjekorras):

( ) sulud (prioriteet),

! inversioon,

* konjunktsioon,

+ disjunktsioon.

Kui *BooleanFunction* on `CONST0` või `CONST1`, siis vastab teegi komponent vastavalt loogilisele null või üks nivoole.

Näide:

Järgnevas on toodud näide teegi lähtekirjeldusest komponendile `my_nand2`. See kujutab endast kahe sisendiga JA-EI lüli väljundiga y ja sisenditega a ja b:

```
GATE my_nand2 y=!(a*b);
```

Lisa 4. KTOD liidese poolt toetatav VHDL alamkuju

Käitumuslik RST kirjeldab digitaalseadme juhtosa ning andmeosa ühise kirjeldusena. Juhtosa on esitatud ilmutatud kujul s.t. kirjeldus sisaldab juhtautomaadi olekuid ja igas olekus on ära kirjeldatud võimalikud sooritatavad andmeosa registersiirete taseme operatsioonid.

L4.1 VHDL -i alamhulga kirjeldus

Translaatori 'front end', s.t. programmiosa, mis teostab lähteteksti parsimise, teksti analüüsi ning sümboltabeli genereerimise, on üldiselt orienteeritud VHDL '93 standardi toetamisele. Järgnevalt on käsitletud VHDL konstruktsioone, mida otsustusdiagrammide sünteesil (nn. translaatori 'back end') toetatakse:

- Andmetüübid
  - Bit
  - Bitvector
  - Integer
- Operaatorid
- VHDL struktuur
  - Entity
    - Port deklaratsioon
  - Architecture (ainult üks arhitektuur disaini kohta)
    - Käitumuslik kirjeldusstiil
      - Process lause
        - If/then/else/end if
        - Case
        - Next/exit laused
        - Variable (muutujad)
        - Signal (signaalid)
        - Wait (protsess ilma tundlikuse nimistuta)
        - Protseduuri ja funktsiooni väljakutsed
      - Funktsioonid ja protseduurid

L4.2 Otsustusdiagrammide sünteesi etapid

1) VHDL-i lähteteksti teisendamine abstraktse süntaksi puu (Abstract Syntax Tree e. AST) kujule

2) Muutujate info kogumine sümboltabelisse lähteteksti skaneerimise käigus

3) VHDL AST-i teisendamine juht- ja andmevoo graafiks (Control Data Flow Graph e. CDFG)

L4.2.1 Digitaalseadme andmeosa süntees:

1) Andmevoo graafi (DFG) ekstraheerimine.
   Andmevoo graafis paiknevad ainult operatsioonid (täitmise järjekorras).

2) Ressursside eraldamise ja sidumine.
   Ressursideks on registrid, funktsionaalsed elemendid (n. summaator) ja multiplekserid.

Iga muutuja jaoks tuleb eraldada register (s.t. muutuja seotakse konkreetse registriga). Ühte registrit saaks põhimõtteliselt kasutada mitme muutuja jaoks kui neid muutujaid kasutatakse erinevates juhtautomaadi olekutes s.t. muutujate eluead ei kattu. See eeldab et eelnevalt on teostatud muutujate eluea analüüs. See ei kuulu, aga käesolev liidese ülesannete hulka.

Igale operatsioonile leitakse funktsionaalselt sobiv riistvaraelement (n. liitmise operatsiooni korral summaator). Seejuures jälgib antud liides põhimõtet, et kõikidele operatsioonidele tuleb omistada erinevad (omaette) funktsionaalsed elemendid.

Lõpuks viiakse läbi andmete kommuteerimise elementide (multiplekserite) eraldamine ja sidumine registrite ning funktsionaalsete elementidega.

L4.2.2 Väljundi genereerimine otsustusdiagrammide formaadis.

- Sisendite, konstantide, operatsioonide info väljastatakse teksti kujul otsustusdiagrammide formaadis sümboltabelisse kogutud info baasil.
- Iga registri jaoks genereeritakse eraldi otsustusdiagramm
- Iga väljundi jaoks tuleb genereerida graaf kui väljund on multipleksitud s.t. talle omistatakse väärtusi erinevatest allikatest
- Digitaalseadme juhtosa (e. olekuautomaadi) graafi väljastamine

L4.3 Väljundi genereerimine RST VHDL formaadis.

RST VHDL genereeritakse juht- ja andmevoo graafide baasil. Graafi tippude ja VHDL-i lausete (operatsioonide) vahel on üks-ühene vastavus.

Väljastatav VHDL-kirjeldus on madalama abstraktsioonitasemega võrreldes käituvusliku registersiirde taseme kirjeldusega. Digitaalseadme juht- ja operatsioonosad on eristatud, iga operatsioon VHDL-i algtekstis on asendatud vastava funktsionaalse elemendiga. Andmete kommuteerimiseks on juurde toodud multiplekserid.

RST VHDL kirjeldust saab kasutada loogikataseme sünteesipaketiga “Synopsys”, mille tagajärjel tekib iga struktuuriploki (mooduli) jaoks loogikataseme skeem. Saadud loogikataseme kirjeldustest genereeritud MTOD mudeleid kasutatakse hierarhilise simuleerimise ja testigenereerimise programmides madaltaseme mudelina.