TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAF40LT

Martin Väljaots 134229

ARITMEETIKA-LOOGIKASEADME ISETESTIMINE

Bakalaureusetöö

Juhendaja: Raimund-Johannes Ubar
Tehnikadoktor

Professor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Martin Väljaots

23.05.2016

Annotatsioon

Töö eesmärgiks oli uurida isetestimise efektiivsust digitaalskeemides või -süsteemides, kasutades testidena süsteemi enda töös kasutatavaid funktsionaalseid signaale (käske, juhtsignaale ja andmeid ehk protsesside vahetulemusi), mis tekivad normaalsetes töörežiimides.

Uurimise eesmärgil projekteeriti täisarvude korrutamisseade, töötati välja programm seadme töö modelleerimiseks, uuriti diagnostikatööriistade abil seadme üldist testitavust ehk 100%-ilist rikete katte saavutatavust, süsteemi töö modelleerimise teel tehti kindlaks seadme töö käigus tekkivad signaalid nende hilisema kasutamise eesmärgil testimiseks ning viidi läbi süsteemi isetestimise seansi simulatsioon isetestimise kvaliteedi määramiseks. Samuti uuriti seadme testitavuse parandamise võimalusi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 6 peatükki, 10 joonist, 5 tabelit.

Abstract

Self testing of arithmetic logic unit

The purpose of this thesis was to study the efficiency of self testing in digital schematics or systems, using functional signals used in the system's operation (commands, control signals and data or the interim results of processes) that are generated during normal work modes.

For the purpose of this study an integer multiplier device was designed, a program for modelling the device's operation was developed, the device's overall testability or the attainability of 100% fault coverage was studied using diagnostics tools, the signals generated during the device's operation were identified and stored for later use for testing and a simulation of the system's self testing session was carried out to determine the quality of self testing. The possibilities of improving the testability of the device were also explored.

Functional self testing was proven to be fairly efficient through using the compilation of test vectors generated with the developed program as a test for the device, however, due to the small size of the schematic that was used to imitate functional self testing in this study, the applied testing software Turbo Tester turned out to be a much faster and more effective method for testing said schematic. On the other hand, it must be said that even though the results from the functional self testing session were topped by the results from tests made using Turbo Tester's built-in test generation methods, that functional self testing removes the need for both external testing devices and software, the use of which can become more costly than spending the time needed for functional self testing.

One can speculate that the efficiency of functional self testing will increase in far larger schematics and systems than the example used in this study, which would take multiple hours to test using external testing equipment.

The thesis is in Estonian and contains 28 pages of text, 6 chapters, 10 figures, 5 tables.

Lühendite ja mõistete sõnastik

Aborted rike	Rike, mille testimiseks sobilike testvektorite otsimine katkestati ette antud otsinguteressursi kitsenduste tõttu
Astah Community	Tarkvara UML diagrammide modelleerimiseks
Backtracks	Ette antud lubatud otsingute ressurss (konfliktide lahendamiskatsete arv)
BCU	*BIST Control Unit*, *BIST*-i juhtseade
BIST	*Built-In Self-Test*, sisseehitatud isetest
Bitt	Informatsiooniühik, millel on kaks võimalikku väärtust. Enamasti omistatakse nendeks väärtusteks 0 ja 1. Ühele järgule kahendarvus vastab üks bitt.
Cadence	Disaini graafiline sisestusprogramm (editor)
Counter	Loendur
CUT	*Circuitry Under Test*, testitav skeem/skeemiosa
Funktsioon	Programmi protseduur, mis tagastab väärtuse
Meetod	Programmi protseduur, mis ei tagasta väärtust
Multiplicand	Korrutamistehtes korrutatav tegur
Multiplier	Korrutamistehtes korrutav tegur
Nihe vasakule	Kahendarvu bittide väärtuste nihutamine vanematesse järkudesse
On-line	Jooksev testide genereerimine
PadLeft(täisarv, sümbol)	Meetod Visual Basic .NET-is, mis lisab stringile vasakult etteantud sümbolit, kuni stringis saavutatakse täisarvuga määratud pikkus
Poolsummaator	Summaator, milles ei arvestata ülekannet nooremast järgust
Rikete liiasus	Testitamatute rikete esinemine skeemis/süsteemis
String	Sümbolite jada
StringBuilder	Klass Visual Basic .NET-is stringide ehitamiseks
StrReverse(string)	Meetod Visual Basic .NET-is, mis pöörab stringi tagurpidi
Testeksperiment	Testi sisestamine testitavasse süsteemi ning testi tulemuste salvestamine
Testgeneraator	Seade, mis koostab süsteemi testimiseks testvektoreid

Testitamatu rike	Rike, mille tuvastamiseks ei leidu ühtki testvektorit
Testvektor	Signaalide kogumik, mis sisestatakse testitavasse süsteemi
ToString	Meetod Visual Basic .NET-is, mis teisendab *StringBuilder*'i sisu stringiks
TPG	*Test Pattern Generator,* testvektorite generaator
TRA	*Test Response Analyzer*, testitulemuse analüsaator
Turbo Tester	Digitaalsüsteemide testimistarkvara
Täissummaator	Summaator, milles arvestatakse ülekannet nooremast järgust
Visual Basic .NET	Objekt-orienteeritud programmeerimiskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

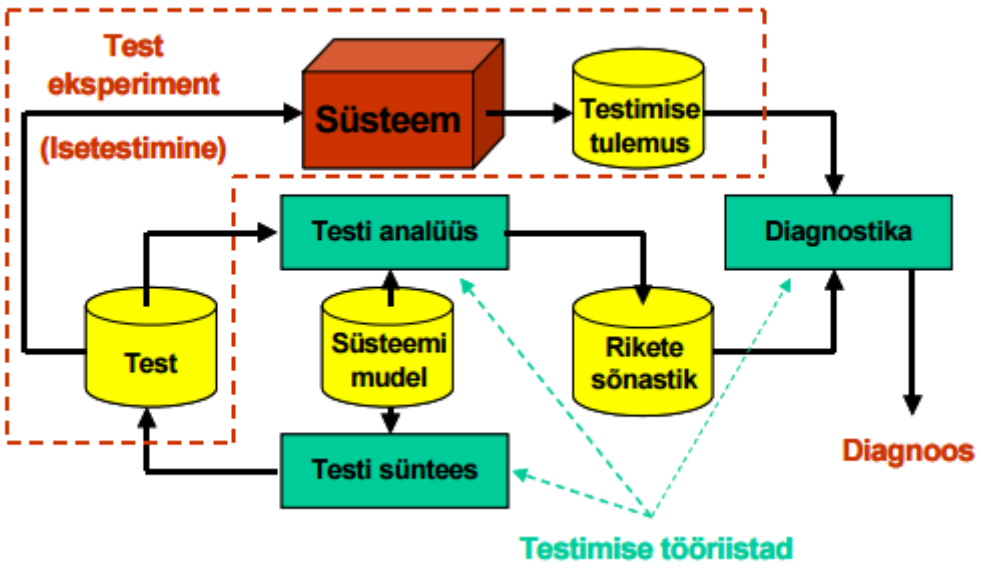
Mida kaugemale areneb tehnoloogia, seda keerulisemaks muutuvad digitaalsüsteemid, mille kvaliteedi tagamine testimisseadmete ja testimistarkvaraga tähendab tänu skeemide elementide ning võimalike vigade arvu suurenemisele järjest mahukamat ning aeganõudvamat testimisprotsessi. Ühe lahendusena sellele probleemile on esile kerkinud isetestivad digitaalsüsteemid, mis elimineerivad vajaduse välise testimissüsteemi järgi, kasutades testimiseks erilist sisseehitatud riistvara [1]. Isetestimise edasijõudnum variant on funktsionaalne isetest, milles kasutatakse testvektoritena süsteemi töö käigus tekkivaid vahetulemusi, eemaldades seeläbi isetestivast süsteemist testvektoreid genereeriva osa.

Käesolevas töös simuleerin isetestivat süsteemi, mis arvutab etteantud algoperandidega a ja b aritmeetikatehet, kus nii a kui b on 8-bitised binaararvud, kasutades programmi, mis arvutab sama aritmeetikatehet ning väljastab vahetulemustena iga liitmistehte operandid, ja kasutades neid vahetulemusi testvektoritena testimistarkvara Turbo Tester kaudu summaatori aritmeetika-loogikaskeemi testimiseks. Võrdlen selle testi tulemusi sellise kombinatsioonskeemil sooritatud testi tulemustega, mille testvektorid koosnevad programmi algselt sisestatavatest a ja b väärtustest, ning annan ühtlasi hinnangu isetestimise kvaliteedile. Samuti võrdlen mõlemal skeemil Turbo Tester tarkvaras meetoditega *Generate*, *Genetic* ja *Random* sooritatud testide tulemusi.

Teine peatükk annab kompaktse ülevaate digitaalsüsteemide testimise ja diagnostika põhiprotsessidest ning kirjeldab lühidalt süsteemi testimist nii välise testimisseadme kui ka sisseehitatud erivahenditega. Kolmandas peatükis on kirjeldatud aritmeetikatehet realiseerivat algoritmi ning neljandas peatükis on kirjeldatud vastava algoritmi põhjal koostatud programmi. Viiendas peatükis on toodud isetestimist imiteeriva summaatori ning etteantud aritmeetikatehet sooritava ALU skeemidel sooritatud testide tulemused ning nende põhjal tehtud järeldused.

2 Lühiülevaade digitaalsüsteemide testimisest

„Digitaalsüsteemide tehnilise diagnostika eesmärk on testimise abil kindlaks teha, kas süsteem on töökorras ja kui ta ei ole töökorras, siis lokaliseerida rikete asukoht nende kõrvaldamiseks.“ [2] Digitaalsüsteemide diagnostika olulisemaid objekte ja protseduure illustreerib joonis 1.



Joonis 1. Testimise tööriistad [2].

Joonisel 1 kujutatavatest objektidest on keskseks objektiks tehniline süsteem, mis kuulub testimisele ja vastava testimise tulemused diagnoosimisele. Testeksperiment viiakse süsteemis läbi kas spetsiaalse välise testimisseadme (automaatse testri) abil või süsteemi enda ressursside poolt ehk isetestimise kaudu [2].

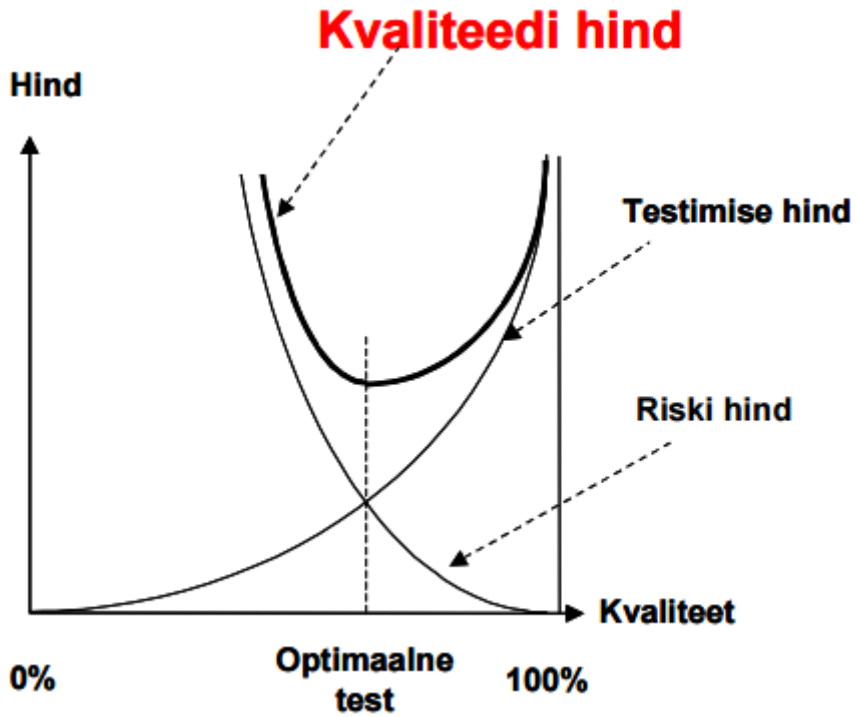
Mõlemal testimise viisil on vajalik süsteemile rakendada testi ehk testvektorite jada, mis sõltuvalt testeksperimendi läbiviimise viisist on kas eelnevalt koostatud (sünteesitud) ja hoitakse testri mälus või genereeritakse jooksvalt (ehk *on-line*) [2]. Testi kvaliteet on avastatavate rikete protsent kõigi võimalike rikete suhtes, mille kindlaks tegemisel kasutatakse erilisi tarkvaralisi testi analüsaatoreid ehk rikete simulaatoreid.

2.1 Välise testmisseadme abil testimine

Automaatse testri abil testide läbiviimisel kasutatakse eelnevalt sünteesitud testvektorite jada, mida hoitakse testri mälus. Testide sünteesiks rakendatakse väga keerulisi tarkvaratööriistu – testide generaatoreid [2].

Testigeneraator kujutab endast programmi, mis analüüsib tervet elektronskeemi mudelit, määrab kindlaks kõik skeemis esineda võivad rikked (lühised ning katkestused elementide sees ja ühenduste vahel) ning otsib ja püüab leida iga rikke jaoks signaalide kogumi, mille rakendamisel skeemi sisenditele avalduks vastav rike, kui see skeemis esineb [2].

Selline testide genereerimine võtab [2] hinnangute alusel mõne tuhande elemendiga skeemi ja parimate generaatorite puhul aega mõni minut, kuid mõnekümne tuhande elemendi korral võib testide genereerimine aega võtta juba tunde ning veelgi suuremate skeemide puhul suureneb kuluv aeg eksponentsiaalselt. Seega tekib testigeneraatorite kasutamisel probleem, kus mingist hetkest alates on testide genereerimine 100% ulatuses ajaliselt ja finantsiaalselt liiga kulukas, kuid teiselt poolt on vähese testimise juures suurem vigade esinemise risk ning kõrgem rikete esinemisel nende parandamise maksumus. Testimise ja riski maksumuse vahekorda illustreerib joonis 2.



Joonis 2. Testimise ja riski maksumuse vahekord [2].

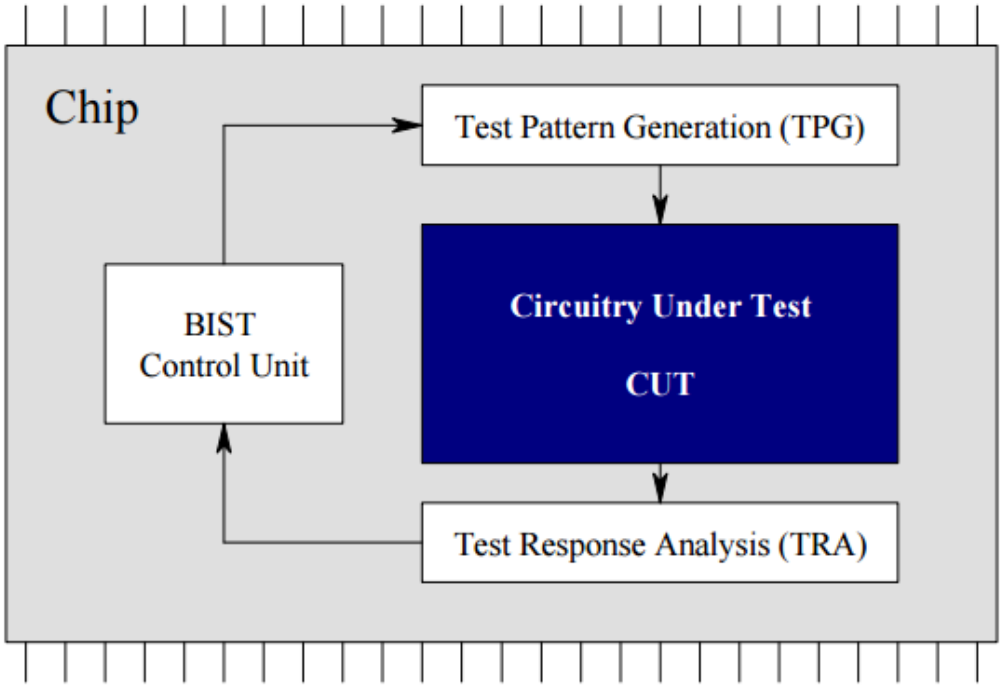
2.2 Isetestimine

Isetestimiseks nimetatakse testimismeetodit, mille käigus viiakse testeksperiment läbi testimisele ja diagnoosimisele kuuluvas tehnilises süsteemis sellesse süsteemi sisse ehitatud erivahendite poolt [2]. Nende erivahendite sisseehitamise eesmärgiks on elimineerida vajadus väliste testimisseadmete järele.

Sisseehitatud isetest pakub lahendust väliste testimisseadmete kasutamisel järjest kõrgemale testide genereerimise maksumusele, skeemide kasvavast suurusest tulenevale testandmete hulga suurenemisele ning suurenevale erinevusele automaattestrite kiiruse ja testitava elektronskeemi sisemise sageduse vahel [1].

Tüüpilise sisseehitatud isetesti ehk BIST-i ehitus on toodud joonisel 3 [1]. BIST koosneb enamasti testvektorite generaatorist (*TPG – Test Pattern Generator*), testi tulemuse analüsaatorist (*TRA – Test Response Analysis*) ja isetesti juhtseadmest (*BCU – BIST Control Unit*) [1]. Testvektorite generaator rakendab testvektoreid testitavale elektronskeemile (*CUT – Circuitry Under Test*).

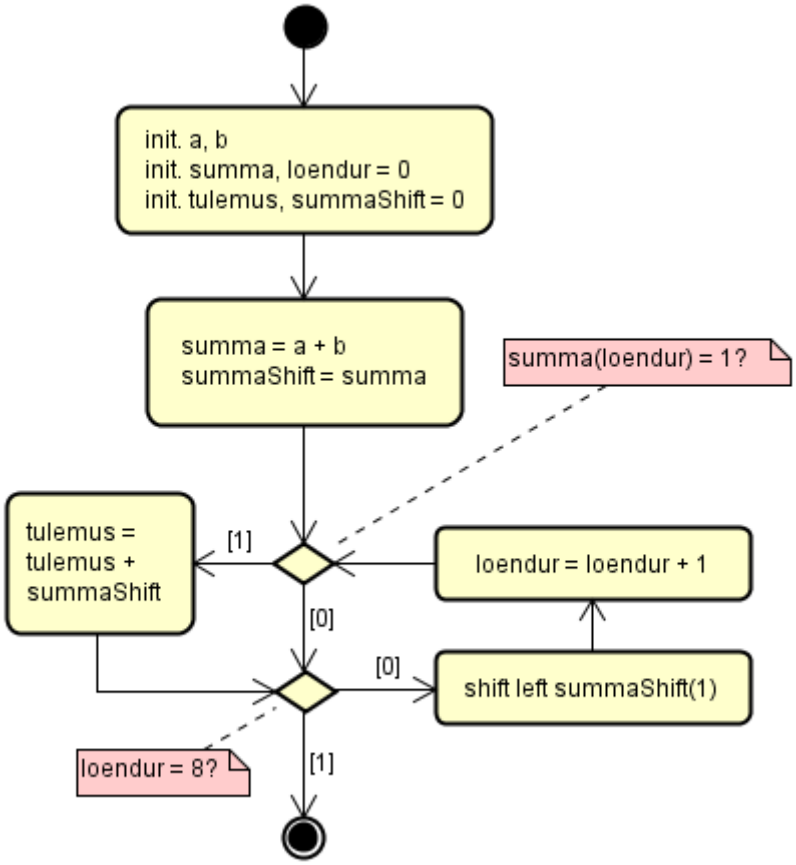
Funktsionaalse BIST-i korral ei kasutata testvektorite saamiseks erilist testigeneraatorit, selle asemel kasutatakse testvektoritena süsteemi enda protsesside vahetulemusi.



Joonis 3. Tüüpiline BIST arhitektuur [1].

3 Aritmeetikatehet realiseeriv algoritm

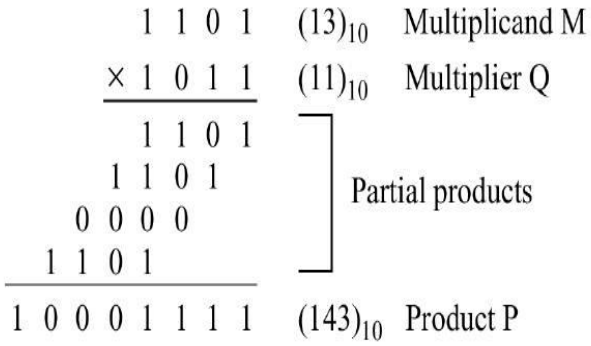
Antud töös käsitletavat aritmeetikatehet $(a+b)^2$ realiseeriv algoritm on esitatud joonisel 4. Algoritmi kujutamiseks on kasutatud programmi Astah Community.



Joonis 4. Aritmeetikatehet $(a+b)^2$ realiseeriv algoritm.

Algoritmi alguses initsialiseeritakse sisestatavad a ja b väärtused ning kõik algoritmis kasutatavad muutujad algväärtustatakse nulliks. Esimese tehtena antakse summa väärtuseks a ja b liitmise tulemus. Algoritmis on summa ruudu arvutamise osa üles ehitatud viisil, kus a ja b summa bittide kontrollimisega otsustatakse, kas tulemust – muutuja tulemus all on mõeldud korrutamise ehk summa ruudu tulemust – suurendatakse või mitte, kuid samuti on vajalik muutuja, mis nihkub vastavalt korrutamise algoritmile (näidis joonisel 5) ning mille võrra vastaval tingimusel tulemust suurendatakse. Selleks muutujaks on antud algoritmis muutuja summaShift, mille väärtuseks seatakse eespool leitud summa väärtus.

Järgnev osa algoritmist tegeleb summa ruudu ehk summa iseendaga korrutise väärtuse arvutamisega. Korrutamine on realiseeritud „nihuta-ja-liida“ põhimõttel, nagu on näidatud joonisel 5. Vastavalt korrutava operandi (joonisel 5 *Multiplier Q*) bittide väärtustele arvestatakse, millistes järkudes toimub korrutatava operandi (joonisel 5 *Multiplicand M*) liitmine tulemusele, ning iga järgu kohta toimub üks nihe vasakule. Töös käsitletavas algoritmis on mõlemad korrutise tegurid – nii korrutatav kui korrutaja – sama väärtusega, seega toimub bittide kontroll operandide a ja b summa põhjal.



Joonis 5. Näide binaararvude „nihuta-ja-liida“ korrutamisest [3].

Et määrata, kas tulemust suurendatakse loenduri väärtusele vastava summa biti arvelt, kontrollitakse algoritmis selle biti väärtust. Tingimuse osa „summa(loendur)“ all on mõeldud summa loenduri väärtusega indekseeritud bitti. Näiteks kui loenduri väärtus on 0, kontrollitakse summa nullindat bitti, arvestades seejuures, et nullis bitt on kõige madalam järk, järgmine kõrgem järk esimene, järgmine pärast seda teine ja nii edasi. Summa vastava biti väärtuse 1 korral liidetakse tulemusele juurde sellel hetkel muutuja summaShift väärtus, vastasel juhul jäetakse tulemuse väärtus samaks.

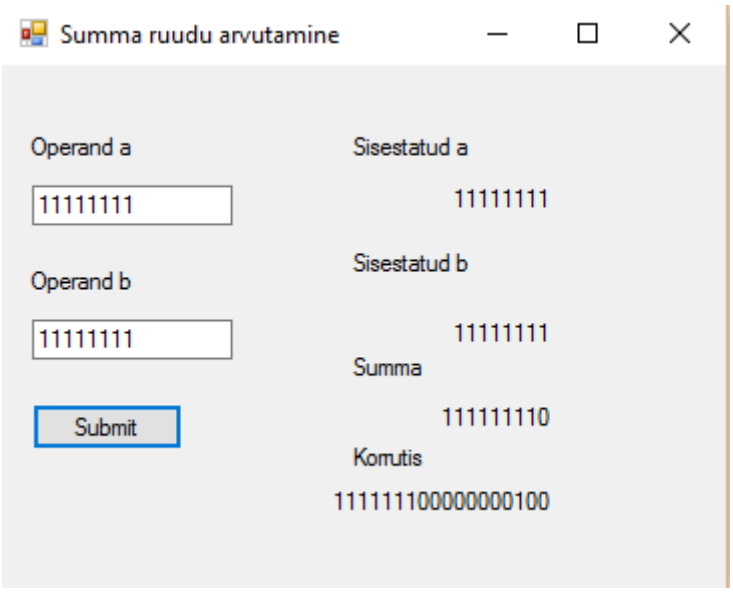
Seejärel kontrollitakse loenduri väärtust. Algoritm jätkab oma tegevust niikaua, kui loenduri väärtus pole jõudnud kaheksani, ehk ei ole kontrollitud kõiki summa võimalikke bitte. Tingimusel, et loenduri väärtus ei ole kaheksa, nihutatakse muutuja summaShift väärtust ühe biti võrra vasakule ning suurendatakse loenduri väärtust järgmise liitmistehte jaoks. Kui loenduri väärtuse kontrolli ajaks on see jõudnud kaheksani, siis kui arvestada, et 8-bitiste kahendarvude liitmise maksimaalne tulemus on koos ülekandebitiga 9-järguline ning järkude loendamine algab nullist, on algoritm aritmeetikatehtele $(a+b)^2$ jõudnud lõpuni.

4 Algoritmi imiteeriv programm

Algoritmi imiteeriv programm on realiseeritud programmeerimiskeeles Visual Basic .NET, raamistikus .NET 4.0, kasutades arenduskeskkonda Microsoft Visual Studio 2015 Community. Programmikood on esitatud täispikkuses lisas 1.

Programmi ülesandeks on lisaks algoritmi järgi aritmeetikatehte teostamisele luua ka tekstifaili vahetulemuste tabel, milles on välja toodud iga aritmeetikatehte läbimisel liitmistehte mõlemad sisendid, alustades algsisenditest a ja b ning lõpetades viimase korrutamise osaks oleva liitmistehte sisenditega. Sisendite vahele jäetakse parema loetavuse eesmärgil tühimik järgmise tabulatsioonimärgini. Samuti jäetakse parema loetavuse eesmärgil tühi rida iga uue tabeli ette, mis tekib a ja b sisestamisel.

Algsisendite a ja b sisestamine käib programmi käivitamisel avanevas aknas. Pärast a ja b sisestamist nupu „Submit“ vajutamisel kuvatakse aknas sisestatud operandide väärtused, operandide summa väärtus ning summa ruudu väärtus. Näide programmi aknast, kuhu on sisestatud nii a kui b väärtuseks kahendarv 11111111, mis ühtlasi annab maksimaalse summa ning summa ruudu väärtuse, on toodud joonisel 6. Selliste väärtuste sisestamisel tekkiv tabel koos esimese tühja reaga on välja toodud tabelis 1.



Joonis 6. Ekraanipilt aritmeetikatehet läbiviiva programmi aknast.

Tabel 1. Sisendite a ja b väärtusteks kahendarvu 11111111 sisestamisel tekkiv tabel.

11111111000000000	11111111000000000
00000000000000000	00000000000000000
00000000000000000	00111111110000000
00111111110000000	00011111111000000
00101111110100000	00001111111100000
00100111110110000	00000111111110000
00100011110111000	00000011111111000
00100001110111100	00000001111111100
00100000110111110	00000000111111110
00100000010111111	00000000011111111

Kõik väljastatavad vahetulemused on 17-kohalised, kuna nii mitu sisendit on kummagi operandi jaoks viiendas peatükis testitaval summaatoril, samuti on vahetulemuste järgud vasakult paremale järjekorras noorimast vanimani, mis teeb lihtsamaks tulemuste testvektoritena Turbo Testrisse sisestamise. Summaatorisse ei ole operandide sisestamisel vaja sisestada vahetulemuste 18. järku ehk ülekandebitti, kuna ka kõige suuremate võimalike algoperandidega muutub ülekandebitt üheks alles viimase tehte tulemuses, mida enam summaatorisse ei sisestata ning seega ka vahetulemuste faili ei lisata.

Pärast tabeli esimest operande sisaldavat rida, kus on algsisendid a ja b, on vasakus tulbas esitatud tulemuse praegune väärtus ning paremas tulbas tulemuse praegusele väärtusele liidetav nihutatava muutuja praegune väärtus. Nihutatava muutuja väärtuseks sisestatakse kõik nullid, kui vastavas tehtes ei toimu nihutatava muutuja liitmist tulemusele.

4.1 Programmi meetodite ning funktsioonide kirjeldus

Programmi keskseks meetodiks on meetod btnSubmit_Click, mille poole pöördutakse programmi käivitamisel avanevas aknas nupule *Submit* vajutamisel ning mis pöördub teistesse kasutatavatesse funktsioonidesse ja meetoditesse. Sisestatud operandid a ja b salvestatakse vastavalt stringidesse strA ning strB, mis programmi aknas kergesti arusaadavaks kuvamiseks lühendatakse esmalt funktsiooniga arvuLyhendamine vanima 1-e sisaldava järguni, seejärel täidetakse ühtlase pikkuse saamiseks meetodiga *PadLeft*(8, „0“) [4] vasakult 0-idega nii palju, kui tühja ruumi on, juhul, kui vastav string ei oma 1-e kõige vanemas ehk 8. järgus.

Meetodis kontrollitakse ka, et a või b poleks sisestamata jäänud, vastasel juhul ilmub ekraanile veateade. Kui a ja b sisestamisel vigu ei leidu, sisestatakse tekstifaili „testvektorid.txt“ tühi rida ning selle alla meetodiga vaheTulemusteFailiGenereerimine a ja b väärtused esimeste liidetavate operandidena, ning pöördutakse a ja b summa saamiseks funktsiooni liitmine poole. Funktsioon liitmine tagastab summa tulemuse, mis seatakse stringi strC väärtuseks. Ka strC lühendatakse programmi aknas kuvamiseks vasakult vanima 1-e sisaldava järguni, kuid funktsiooni korrutamine jaoks täidetakse selle tühjad kohad vasakult meetodiga *PadLeft*(9, „0“) 0-idega, kui tulemuse vanimas ehk 9. järgus (ülekandebitt) ei ole juba 1-e. Täidetud string antakse sisendiks funktsioonile korrutamine, mille tulemus lühendatakse samuti aknas kuvamiseks vasakult vanima 1-e sisaldava järguni.

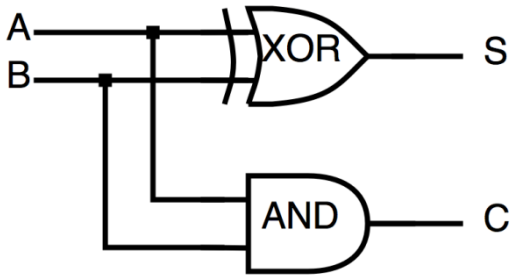
Üks erinevus programmi ning algoritmi vahel on see, et programmis on kõige noorem järk kõige kõrgema järjekorraindeksiga, kuna Visual Basic .NET-is (ning ka enamikes teistes programmeerimiskeeltes) loetakse stringides sümbolite kohti vasakult paremale alates nullist – algoritmis on aga noorim järk kõige väiksema järjekorraindeksiga, kuna algoritmis loetakse järke paremalt vasakule. Programmis on siiski järjestus tehtud selliselt, et nooremad järgud algavad paremalt.

Funktsiooni arvuLyhendamine sisendiks on lühendamisele kuuluva arvu string strArv. Niikaua, kui stringi vasakult esimesel kohal asub 0 ning stringi pikkus on ühest suurem, eemaldatakse stringi vasakult esimesel kohal asuv sümbol. Funktsioon tagastab vastava lühendatud stringi.

Meetodi vaheTulemusteFailiGenereerimine sisenditeks on antud ajahetkel liitmistehte sisenditeks olevad tabelisse lisatavad operandid. Mõlemate operandide stringid täidetakse vastavalt vajadusele meetodiga *PadLeft*(17, „0“) vasakult nullidega, et saavutada 17-kohalised stringid, mis kirjutatakse meetodiga *StrReverse*(string) [5] ümber pööratuna tühimikku vahele jättes tekstifaili „testvektorid.txt“ viimasele reale.

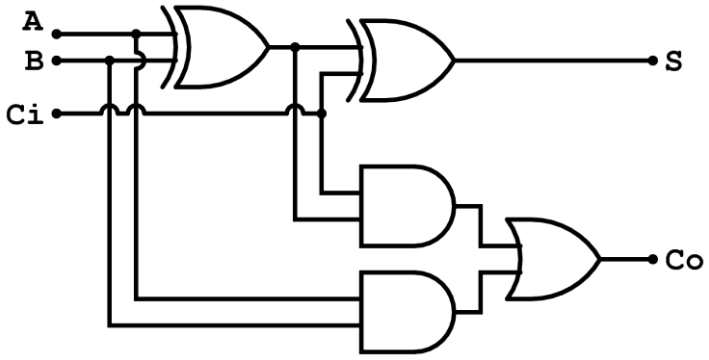
Funktsiooni liitmine sisenditeks on liitmistehtele kuuluvad stringidena kujutletavad operandid strLiidetav1 ja strLiidetav2 ning tulemuse stringi ilma ülekandebitita maksimaalne pikkus intLength. Funktsioonis kasutatakse klassi *StringBuilder* [6] summa tulemuse ning ülekande stringide ehitamiseks, kuna Visual Basic .NET-is puudub võimalus stringi üksikuid sümboleid muuta.

Noorima järgu puhul on liitmine realiseeritud poolsummaatori loogikaskeemi alusel, kuna pole vaja arvestada ülekandega nooremast järgust. Näide poolsummaatori loogikaskeemist on joonisel 7.



Joonis 7. Poolsummaatori loogikaskeem [7].

Vanemate järkude liitmine on realiseeritud täissummaatori loogikaskeemi alusel. Näide täissummaatori loogikaskeemist on joonisel 8.



Joonis 8. Täissummaatori loogikaskeem [7].

Summa tulemuse ehitamiseks kasutatav *StringBuilder* konverteeritakse stringiks strSumma meetodiga *ToString* [6]. Enne tulemuse lõplikku väljastamist kontrollitakse ülekannet vanimast järgust. Ülekande eksisteerimisel lisatakse see lõpptulemusele, mis väljastatakse stringina strSumma.

Funktsiooni korrutamine sisendiks on string strC, milles on salvestatud algoperandidega a ja b sooritatud liitmistehte tulemus. Alguses defineeritakse uus *StringBuilder* klassi muutuja tulemuseEhitaja ning stringid strTulemus ja strTyhi, milles täidetakse 17 kohta nullidega. Muutuja strSummaShift on nihutatav muutuja, mille saamiseks lühendatakse muutuja strC vasakult esimese vanima järguni, mis sisaldab 1-e.

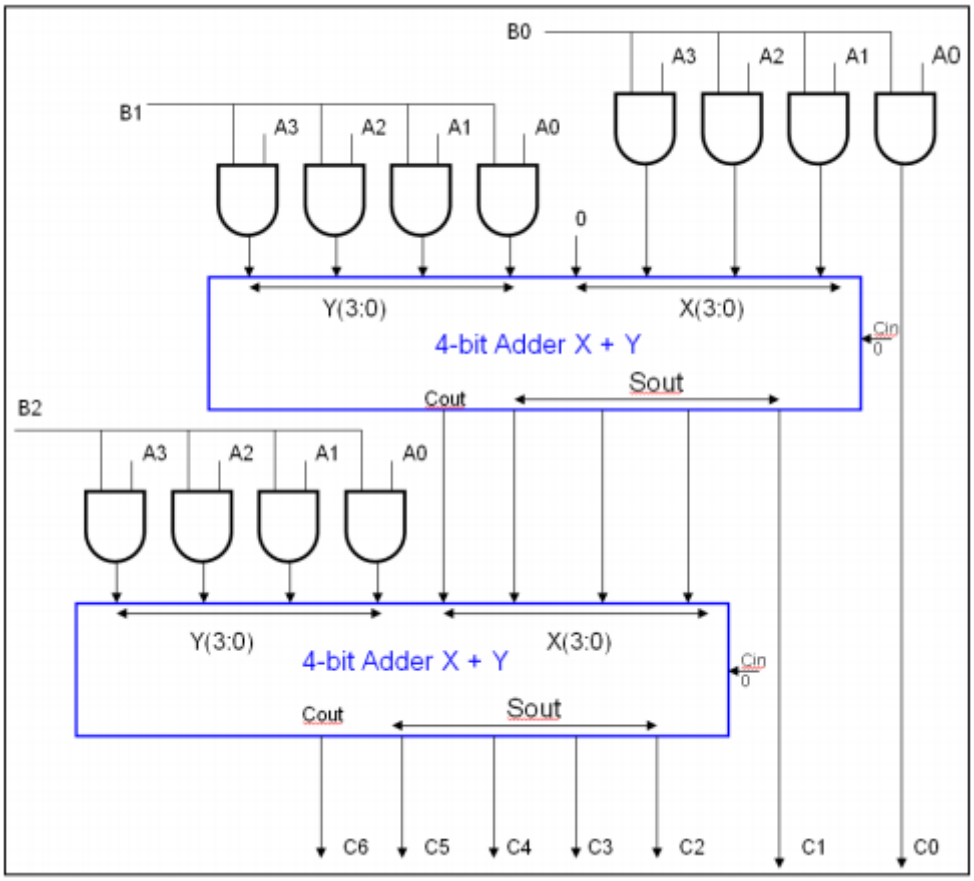
Funktsiooni sees on tsükkel, mille loendur käib kaheksast nullini ning milles kontrollitakse vastavalt loenduri hetkeväärtusele üht kindlat bitti a ja b liitmise tulemuses. Kui vastav bitt on 1, pöördutakse funktsiooni liitmine poole, andes sisenditeks strTulemus, strSummaShift, ja sisendiks intLength 17, ning meetodi vaheTulemusteFailiGenereerimine poole, mille sisenditeks antakse strTulemus ning strSummaShift. Kui vastav bitt on 0, tähendab, et ei toimu nihutatava muutuja liitmist tulemusele ning pöördutakse ainult meetodi vaheTulemusteFailiGenereerimine poole, andes selle sisenditeks strTulemus ning muutuja strSummaShift asemel strTyhi. Mõlemal juhul nihutatakse muutujat strSummaShift vasakule, lisades stringile lõppu 0-väärtusega biti, nihutades seeläbi teisi bitte vanematesse järkudesse. Korrutamistehte lõpul tagastab funktsioon korrutamistehte tulemuse stringina strTulemus.

Kui korrutamistehe on sooritatud, väljastatakse programmi aknasse korrutamistehte tulemus ning on võimalik sisestada järgmised algoperandid.

5 Aritmeetika-loogikaseadmete testimine

Nii summaatori kui ka summa ruutu arvutava ALU skeemid on disainitud programmiga Cadence. Testeksperimentide läbiviimiseks on kasutatud programmi Turbo Tester.

Summaatori loogikaskeem koosneb ühest poolsummaatorist (joonis 7) ning 16-st täissummaatorist (joonis 8). Summa ruutu arvutava ALU loogikaskeem (edaspidi ka kombinatsioonskeem) koosneb suurest algoperandide liitmistehet sooritavast summaatorist, mis koosneb ühest poolsummaatorist ja 7-st täissummaatorist, ning korrutamistehte realiseerimiseks 8-st samasugusest, kuid ühe täissummaatori võrra suuremast suurest summaatorist, mis on omavahel ühendatud vastavalt joonisel 9 toodud näitele, et imiteerida korrutamistehtel sooritatavat nihet.



Joonis 9. Kombinatsioonskeem 4-bitise kahendarvu korrutamiseks 3-bitise kahendarvuga [8].

Esmalt sooritasin summaatorskeemil testid Turbo Testri meetoditega *Generate*, *Genetic* ja *Random*. Vastavate testide tulemused on toodud tabelis 2.

Tabel 2. Summaatorskeemil *Generate*, *Genetic* ja *Random* meetoditega tehtud testide tulemused.

Meetod	**Tuvastatud rikete arv kõigi rikete kohta**	**Rikete katteprotsent**	**Testvektorite arv**
Generate	464/464	100%	18
Genetic	464/464	100%	12
Random	464/464	100%	20

Seejärel sooritasin summaatorskeemil meetodiga *Analyze* üheksa testi, mille jaoks kasutasin programmi suvaliste algoperandidega loodud testvektorite faili. Iga testi kohta lisasin juurde ühe komplekti vahetulemusi – algoperandid ning üheksa liitmistehte sisendoperandid, kokku kümme vektorit iga komplekti kohta. Nende testide tulemused on toodud tabelis 3.

Tabel 3. Summaatorskeemil programmi poolt koostatud testvektorite failiga sooritatud testide tulemused.

Testi number	**Tuvastatud rikete arv kõigi rikete kohta**	**Rikete katteprotsent**	**Testvektorite arv**
1	284/464	61,21%	10
2	362/464	78,02%	20
3	362/464	78,02%	30
4	387/464	83,41%	40
5	408/464	87,93%	50
6	426/464	91,81%	60
7	441/464	95,04%	70
8	441/464	95,04%	80
9	449/464	96,77%	90

Summa ruutu arvutava ALU kombinatsioonskeemi joonis on toodud lisas 2. Kombinatsioonskeemi testimisel tegin lisaks *Generate*, *Genetic* ja *Random* testidele ka kaks testi *Generate* meetodiga, kus muutsin *backtracks*'ide suurust 10000 ja 100000 peale. Kombinatsioonskeemiga tehtud testide tulemused on toodud tabelis 4. *Genetic* ja *Random* meetodite puhul pole välja toodud rikete efektiivsust ning testitamatute rikete – rikked, mille olemasolu kontrollimiseks ei leidu testvektoreid – ega *aborted* rikete –

rikked, mille testimiseks sobilike testvektorite otsimine katkestati ette antud otsinguteressursi kitsenduste tõttu – arvu, kuna need meetodid ei tagasta selliseid andmeid.

Tabel 4. Summa ruudu aritmeetikatehet realiseerival kombinatsioonskeemil meetoditega *Generate*, *Genetic* ja *Random* tehtud testide tulemused.

Meetod	**Testitud rikete arv kõigi rikete kohta**	**Testitamatute rikete arv/*Aborted* rikete arv**	**Rikete katteprotsent**	**Rikete efektiivsus**	**Testvektorite arv**
Generate	2017/2158	5/136	93,47%	93,68%	28
Genetic	2086/2158	-	96, 66%	-	24
Random	2086/2158	-	96,66%	-	45
Generate (10000 *backtracks*)	2086/2158	44/28	96,66%	98,68%	41
Generate (100000 *backtracks*)	2086/2158	72/0	96,66%	100%	41

Võrdlusmomendi tekitamiseks isetestimist imiteeriva summaatori skeemi ning mitte-isetestiva kombinatsioonskeemi vahel tekitasin meetodiga *Analyze* sarnaselt summaatorskeemi testidele kombinatsioonskeemile testid, mis koosnesid programmi koostatud testvektorite faili vektorite komplektide esimestest kirjetest ehk algsisenditest a ja b, kuna kombinatsioonskeemis ei toimu skeemi mitmekordset läbimist nagu isetestimist imiteerivas summaatori skeemis.

Kuna testvektorite faili kirjutatakse a ja b summaatori skeemi jaoks 17-kohaliselt, eemaldasin kombinatsioonskeemis läbi viidavate testide jaoks vektoritelt 9 vanemat järku, mis algsisendite puhul on tühjad. Nende testide tulemused on välja toodud tabelis 5.

Tabel 5. Summa ruudu aritmeetikatehet realiseerival kombinatsioonskeemil testvektorite failiga tehtud testide tulemused.

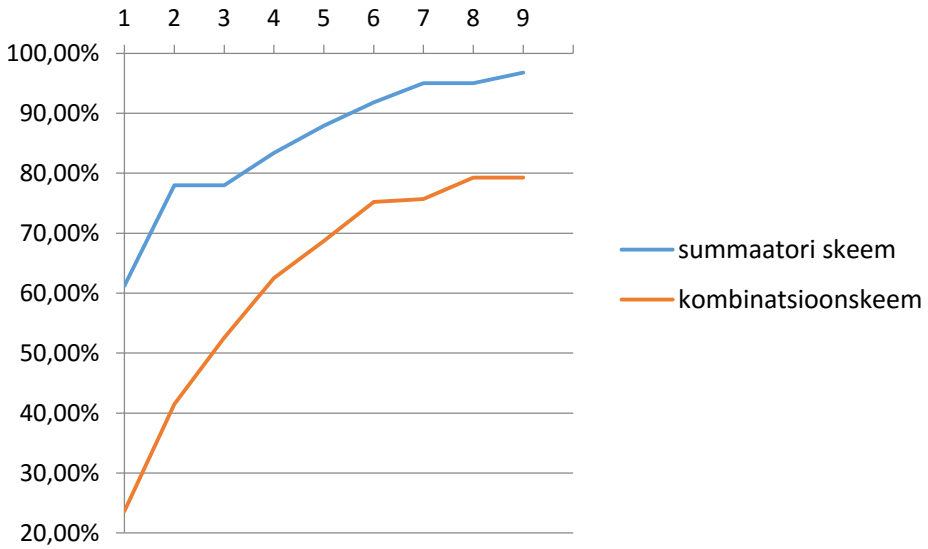
Testvektorite arv	**Testitud rikete arv kõigi rikete kohta**	**Vigade katteprotsent**
1	510/2158	23,63%
2	895/2158	41,47%
3	1135/2158	52,59%
4	1349/2158	62,51%
5	1483/2158	68,72%
6	1623/2158	75,21%
7	1634/2158	75,72%
8	1711/2158	79,29%
9	1711/2158	79,29%

5.1 Testide analüüs ja järeldused

Turbo Testri meetoditega *Generate*, *Genetic* ja *Random* tehtud testide tulemustest saab järeldada, et summaatori skeem on täielikult testitav, samas kui kombinatsioonskeem on 96,66% ulatuses testitav. Kombinatsioonskeemis esineb rikete liiasus – skeemis esineb rikkeid, mille testimiseks ei leidu mitte ühtki testvektorit [2]. Selliseid rikkeid on kombinatsioonskeemis 72 tükki.

Summaatori skeem osutus täielikult testitavaks, seega pole selle testitavuse parandamine tarvilik. Kombinatsioonskeemi testitavuse parandamiseks võib tekitada skeemis lisaväljundeid kohtadesse, kust riket pole võimalik edasi tavalistesse väljunditesse levitada, tehes seeläbi võimalikuks rikke tuvastamise. Samas tekitavad lisaväljundid omakorda juurde potensiaalseid rikkeid, mida testima peab. Kombinatsioonskeemi võib ka lihtsustada Boole'i diferentsiaalvõrrandite abil [2], kuid see on sellise suurusega skeemi korral üsna keeruline.

Graafik testvektorite failiga tehtud testide rikete katteprotsentidega on toodud joonisel 10.



Joonis 10. Skeemidel testvektorite failiga sooritatud testide rikete katteprotsent testi kohta.

Võrreldes kombinatsioonskeemil testvektorite failiga tehtud testide tulemustega on funktsionaalset isetestimist imiteeriva summaatori skeemil tehtud testide tulemused märgatavalt paremad, saavutades viimase testiga vigade katteprotsendiks 96,77%, samas kui kombinatsioonskeemil saavutati viimase testiga katteprotsendiks 79,29%.

Hindan summaatori skeemil tehtud testide tulemused heaks, kuna skeem on täielikult testitav ning ettantud testvektoritega saavutati peaaegu maksimaalne võimalik rikete katteprotsent, mida ilmselt oleks saanud üsna lihtsasti parandada – arvan, et lisades veel 9 erinevat algoperandide paari, oleks olnud täiesti võimalik saavutada 100%-iline rikete kate. Mõlemas skeemis saavutati parim vastava skeemi rikete katteprotsent vähimate testvektorite arvuga meetodiga *Genetic* – summaatori skeemis 12 testvektoriga ning kombinatsioonskeemis 24 testvektoriga.

Üldplaanis võib testide tulemuste põhjal järeldada, et funktsionaalne isetest on üsnagi efektiivne digitaalsüsteemide testimise meetod, kuid antud skeemide puhul andsid testimistarkvaraga Turbo Tester sooritatud testid siiski parema tulemuse. Samas tuleb arvestada, et siinkohal 90 testi isetesti puhul on küll ajaliselt pikem kui Turbo Testriga läbi viidud testid, kuid samas ei ole seejuures vaja väliseid testseadmeid, mis on sageli väga kulukad – kuigi Turbo Tester on näiteks vabavara – ega testprotseduuri eraldi realiseerimist, samuti pole võrreldes tavalise sisseehitatud isetestiga vajalik testgeneraatori realiseerimine seadmes.

Tulemuste vaatlemisel tuleb arvesse võtta teises peatükis välja toodud hinnanguid loogikaskeemi suurusele vastavale testvektorite genereerimise ajale. Mõlemad skeemid sisaldavad digitaalsüsteemide maailma mõistes väga väikest arvu elemente – summaatori skeemi elementide arv jääb saja piiridesse ja kombinatsioonskeemi elementide arv jääb alla 500 elemendi – ning Turbo Testriga testimise tulemused saab kätte silmapilkselt, välja arvatud kombinatsioonskeemi testimisel meetodi *Generate backtracks*'ide arvu suurendamisel, kus testimine võttis mõlemal juhul siiski aega alla 30 sekundi.

Mõlema testitud seadmega esineb ilmselt olukord, kus Turbo Tester suudab testi läbi viia kiiremini, kui seadmetesse jõutakse esimesed algoperandid sisestada. Seega arvan, et funktsionaalse isetesti tõeline efektiivsus avaldub suuremates, mõnekümne tuhande elemendiga skeemides, mille välise testseadmega testimiseks kuluks tunde.

6 Kokkuvõte

Funktsionaalset isetesti imiteeriva summaatori ALU skeem osutus täielikult testitavaks, samas kui kombinatsioonskeem osutus testitavaks 96,66% ulatuses. Kombinatsioonskeemis esines rikete liiasus – skeemis esines 72 testitamatut riket. Parimaks Turbo Testri meetodiks osutus mõlema skeemi puhul meetod *Genetic*, saavutades summaatori skeemil maksimaalse rikete katte 12 testvektoriga ning kombinatsioonskeemil testitavuse ulatuses maksimaalse rikete katte 24 testvektoriga.

Aritmeetikatehet läbi viivast programmist saadud vahetulemuste failiga testimisel saavutati summaatori skeemil kõige suurema testvektorite arvu juures rikete katteprotsendiks 96,77%. Tulemus on vägagi hea, kui arvestada, et tegelikus funktsionaalselt isetestivas süsteemis tuleb selle tulemuse saamiseks sisestada kõigest 9 algoperandide paari, kasutamata seejuures ei väliseid testseadmeid ega sisseehitatud testgeneraatorit. Funktsionaalse isetesti juures iga algoperandide paari kohta 9 sisestatavat lisatestvektorit on suureks panuseks summaatori skeemis saavutatud kõrgemale rikete katteprotsendile.

Funktsionaalne isetest on küll üsna efektiivne, kuid funktsionaalset isetesti imiteeriva skeemi väikse suuruse tõttu on testimistarkvara Turbo Tester siiski tunduvalt kiirem ning efektiivsem viis antud skeemide testimiseks. See asjaolu ei pruugi tõsi olla muude testimisseadmete ning -tarkvara korral, mille kasutamine võib osutuda kulukamaks, kui funktsionaalseks isetestiks kuluva aja kulutamine. Usun, et funktsionaalse isetesti efektiivsus kasvab märgatavalt suuremates skeemides ja süsteemides, mille testimiseks väliste testseadmetega kuluks mitmeid tunde.

Kasutatud kirjandus

[1] G. Jervan, „High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems,“ Linköping Studies in Science and Technology, Linköpings, 2002.

[2] R. Ubar, Digitaalsüsteemide diagnostika. I, Diagnostiline modelleerimine, Tallinn: Tallinna Tehnikaülikool, 2004.

[3] „binary number system,“ [Võrgumaterjal]. Available: http://www.kprblog.in/cse/binary-number-system/. [Kasutatud 15 Mai 2016].

[4] Microsoft, „String.PadLeft Method(System),“ [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/system.string.padleft(v=vs.110).aspx. [Kasutatud 21 Mai 2016].

[5] Microsoft, „Strings.StrReverse Method (String) (Microsoft.VisualBasic),“ [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.strreverse(v=vs.110).aspx. [Kasutatud 21 Mai 2016].

[6] Microsoft, „StringBuilder Class (System.Text),“ [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/system.text.stringbuilder(v=VS.100).aspx. [Kasutatud 21 Mai 2016].

[7] S. Tarfe, „Adder Circuit By Sahil Tarfe - Alchetron,“ [Võrgumaterjal]. Available: http://alchetron.com/Adder-Circuit-732-W. [Kasutatud 18 Mai 2016].

[8] University of Pennsylvania, Department of Electrical and Systems Engineering, „Lab4_CombMult_updated15,“ 4 Märts 2015. [Võrgumaterjal]. Available: http://www.seas.upenn.edu/~ese171/labs12/Lab4_CombMult.pdf. [Kasutatud 20 Mai 2016].

Lisa 1 – Algoritmi realiseeriv programmikood täispikkuses

```
Public Class formSummaRuut
    Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles
btnSubmit.Click
        Dim strA As String
        Dim strB As String
        Dim strC As String
        Dim strD As String
        If txtA.Text.Length = 0 Then
            MsgBox("Sisestage operand a!")
            Return
        ElseIf txtB.Text.Length = 0 Then
            MsgBox("Sisestage operand b!")
            Return
        End If
        strA = txtA.Text
        lblSisestatudA.Text = arvuLyhendamine(strA)
        strA = strA.PadLeft(8, "0")
        strB = txtB.Text
        lblSisestatudB.Text = arvuLyhendamine(strB)
        strB = strB.PadLeft(8, "0")
        My.Computer.FileSystem.WriteAllText("C:\testvektorid.txt", vbCrLf,
True)
        vaheTulemusteFailiGenereerimine(strA, strB)
        strC = liitmine(strA, strB, 8)
        lblSumma.Text = arvuLyhendamine(strC)
        strC = strC.PadLeft(9, "0")
        strD = korrutamine(strC)
        lblKorrutis.Text = arvuLyhendamine(strD)
    End Sub

    Private Function liitmine(ByVal strLiidetav1 As String, ByVal
strLiidetav2 As String, ByVal intLength As Integer)
        Dim summaEhitaja As New Text.StringBuilder()
        summaEhitaja.Length = 0
        Dim strSumma As String = StrDup(intLength, "0")
        Dim strTyhi As String = StrDup(intLength, "0")
        Dim strCarry As New Text.StringBuilder()
        strCarry.Length = 0
        strCarry.Insert(0, strTyhi)
        summaEhitaja.Insert(0, strTyhi)
        summaEhitaja(summaEhitaja.Length - 1) =
CStr(Val(strLiidetav1(strLiidetav1.Length - 1)) Xor
Val(strLiidetav2(strLiidetav2.Length - 1)))
```

```
        strCarry(strCarry.Length - 1) =
CStr(Val(strLiidetav1(strLiidetav1.Length - 1)) And
Val(strLiidetav2(strLiidetav2.Length - 1)))
        For i As Integer = 0 To (strLiidetav2.Length - 2)
            summaEhitaja(summaEhitaja.Length - 2 - i) =
CStr(Val(strLiidetav1(strLiidetav1.Length - 2 - i)) _
                Xor Val(strLiidetav2(strLiidetav2.Length - 2 - i)) Xor
Val(strCarry(strCarry.Length - 1 - i)))
            strCarry(strCarry.Length - 2 - i) =
CStr(((Val(strLiidetav1(strLiidetav1.Length - 2 - i)) _
                Xor Val(strLiidetav2(strLiidetav2.Length - 2 - i))) And
Val(strCarry(strCarry.Length - 1 - i))) _
                Or (Val(strLiidetav1(strLiidetav1.Length - 2 - i)) And
Val(strLiidetav2(strLiidetav2.Length - 2 - i))))
        Next
        strSumma = summaEhitaja.ToString
        If summaEhitaja.Length - strLiidetav2.Length > 0 Then
            If Val(strCarry(strCarry.Length - strLiidetav2.Length)) = 1 Then
                summaEhitaja(summaEhitaja.Length - strLiidetav2.Length - 1) =
CStr(1)
                strSumma = summaEhitaja.ToString
            End If
        ElseIf summaEhitaja.Length - strLiidetav2.Length = 0 Then
            If Val(strCarry(0)) = 1 Then
                strSumma = "1" & strSumma
            End If
        End If
        Return strSumma
    End Function

    Private Function korrutamine(ByVal strC As String)
        Dim tulemuseEhitaja As New Text.StringBuilder()
        tulemuseEhitaja.Length = 0
        Dim strTulemus As String = StrDup(17, "0")
        Dim strTyhi As String = StrDup(17, "0")
        Dim strSummaShift As String
        tulemuseEhitaja.Insert(0, strTyhi)
        strSummaShift = arvuLyhendamine(strC)
        For counter As Integer = 8 To 0 Step -1
            If Val(strC(counter)) = 1 Then
                vaheTulemusteFailiGenereerimine(strTulemus, strSummaShift)
                strTulemus = liitmine(strTulemus, strSummaShift, 17)
            Else
                vaheTulemusteFailiGenereerimine(strTulemus, strTyhi)
            End If
            strSummaShift = strSummaShift & "0"
        Next
        Return strTulemus
    End Function

    Private Function arvuLyhendamine(ByVal strArv As String)
        While Val(strArv(0)) = 0 And strArv.Length > 1
```

```
            strArv = strArv.Remove(0, 1)
        End While
        Return strArv
    End Function

    Private Sub vaheTulemusteFailiGenereerimine(ByVal strOperand1 As String,
ByVal strOperand2 As String)
        strOperand1 = strOperand1.PadLeft(17, "0")
        strOperand2 = strOperand2.PadLeft(17, "0")
        strOperand1 = StrReverse(strOperand1)
        strOperand2 = StrReverse(strOperand2)
        Dim strVektor As String = strOperand1 & vbTab & strOperand2 & vbCrLf
        My.Computer.FileSystem.WriteAllText("C:\testvektorid.txt", strVektor,
True)
    End Sub
End Class
```

Lisa 2 – Summa ruutu arvutava ALU kombinatsioonskeem