KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Rolandas Narvilas

Funkcinių testų skaitmeniniams įrenginiams projektavimas ir analizė

Magistro darbas

Vadovas:

Prof. K. Motiejūnas

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Rolandas Narvilas

Funkcinių testų skaitmeniniams įrenginiams projektavimas ir analizė

Magistro darbas

Recenzentas:		**Vadovas:**	
2010-05-31	**Doc. A. Lenkevičius**	**2010-05-31**	**prof. K. Motiejūnas**

Atliko:
2010-05-31 **IFM-9/2 gr. stud.**
Rolandas Narvilas

Kaunas, 2011

Turinys

SUMMARY

Project objective – to develop a system, which generates functional tests for non-scan synchronous sequential circuits based on functional delay models. During project execution, the analysis of design and technology solutions was performed. The architecture of the developed software is based on the requirement to be able to use the models of the benchmark circuits that are written in C programming language. Analysis of the effectiveness of the model file integration, possibilities of improving random test sequence generation and the influence of distribution of „1“ in randomly generated test patterns was performed. The results of the analysis were:

- Type of the model file integration has little effect when using large circuit models.
- The implementation of semi deterministic algorithms showed that the optimisation of separate steps by construction of test subsequences doesn't improve the final outcome.
- The distribution of „1“ in randomly generated test patterns has effect on the fault coverage and can be used to improve test generation process.

ĮVADAS

Puslaidininkiai prietaisai tampa vis sudėtingesni atsižvelgiant į tranzistorių kiekį juose, dažnį ir integracijos laipsnį. Defektų spektras pasipildė naujais defektais, kurie ne visada yra aptinkami tradiciniais statiniais testais, dar vadinamais fiksuotos reikšmės gedimų testais.

Testų generavimas yra kuriamas dviem kryptimis. Paprastai, testas schemai yra generuojamas struktūriniame lygyje. Šiuo atveju, problema yra generacijos laikas, todėl kad jis tiesiogiai įtakoja produkto pateikimo rinkai laiką. Funkcinis testas paprastai yra žymiai didesnis už testą, sukurtą pagal schemos realizaciją, tam, kad užtikrintų gerą gedimų padengimą dideliam realizacijų kiekiui. Kai schemai vietoj aukšto abstrakcijos lygio modelio pritaikoma tam tikra realizacija, atsižvelgiant į šią realizaciją galima atlikti funkcinių testų minimizavimą, kad būtų pašalinti testiniai atvejai, kurie neaptinka šios realizacijos gedimų.

Projekto tikslas – sukurti sistemą, skirtą schemų testinių atvejų atrinkimui naudojant „juodos dėžės“ modelius ir jiems pritaikytus gedimų modelius.

Atsižvelgiant į tai, jog sistema darbui naudos C kalba aprašytas schemų modelius, jai kurti buvo pasirinktas įrankis Microsoft Visual Studio 2010.

1. ANALITINĖ DALIS

Šiame skyriuje apibūdinami tyrimo tikslai, apibendrinama atlikta literatūros analizė, pasirengiama projekto reikalavimų specifikavimui, projektavimui, susipažįstama su užsakymo taikymo sritimi ir pasauliniais pasiekimais taikomojoje srityje.

Raktiniai žodžiai: schemų testavimas, gedimų modeliai, funkcinių testų sudarymas, perdavimų matricos gedimų modelis, vėlinimo gedimų modeliai, „juodos dėžės“ gedimų modeliai.

Keywords: circuit testing, fault models, functional test generation, input-output pin pair fault model, delay fault models, black-box fault models.

1.1. Literatūros apžvalga

1.1.1. Temos aktualumas ir tikslingumas

Testavimas sudaro didelę elektroninės schemos projektavimo ir gamybos išlaidų dalį. Į rinką išleistas produktas su defektais tik dar labiau padidina išlaidas. Nuolat augant projektų apimčiai problema vis didėja, nes testavimui reikalingi vis didesni techniniai resursai. Iki šiol plačiai naudojamas konstantinių gedimų modelis (SSF) tampa nepakankamai efektyvus, nes didėjant integracijos laipsniui kai kurie fiziniai defektai pasireiškia kaip vėlinimo gedimai.

Vienas iš galimų sprendimų, siekiant sumažinti elektroninių schemų projektavimo išlaidas, yra atlikti kaip įmanoma didesnę dalį darbų su aukštesnio abstrakcijos lygio modeliais. Šių modelių apimtis žymiai mažesnė, todėl jų modeliavimas paprastesnis, reikalaujantis mažiau resursų ir pinigų. Tačiau nėra visuotinai pripažintų gedimų modelių aukšto abstrakcijos lygio aprašams. Viso projektavimo proceso metu naudojamos įvairios kalbos ir programinės priemonės modelių aprašymui. Tai apsunkina į šių aprašų tekstą orientuotų gedimų modelių sudarymą. Siūlomas sprendimas – naudoti „juodos dėžės“ modelius ir jiems pritaikytus gedimų modelius testų sudarymui.

Tyrimo tikslas yra išnagrinėti testų „juodos dėžės“ modeliams sudarymo galimybes. Tam tikslui turi būti sudaryti skaitmeninių įrenginių testavimo metodai, kurie leistų sudarinėti kokybiškus testus, kai nežinoma įrenginių struktūra, o žinomas tik schemos veikimo algoritmas (funkcinis modelis).

1.2. Testų tipai

Sąvokos

Testavimas – elektroninių schemų projektavimo proceso dalis. Jo tikslas surasti klaidas – projektavimo klaidas, gamybos defektus ir eksploatacijos metu atsiradusius gedimus, patvirtinti teisingą schemos veikimą. Aptikus klaidingą schemos veikimą, antras testavimo tikslas – lokalizuoti problemą [1]. Norint lokalizuoti gedimą, reikalingos žinios apie schemos realizaciją.

Verifikavimas – projektavimo proceso operacija, kurios tikslas patikrinti, ar nebuvo nukrypta nuo specifikacijos. Kitaip sakant, šio žingsnio metu tikrinama, ar skirtingo abstrakcijos lygio modeliai aprašo tą patį objektą. Verifikavimas vykdomas anksčiau nei fiziškai pagaminama schema, naudojant įvairaus abstrakcijos lygio modelius. Verifikavimas gali būti statinis arba dinaminis. Statinis verifikavimas atliekamas analizuojant modelio tekstą ar struktūrą. Dinaminio verifikavimo atveju apskaičiuojamos modelio reakcijos į testus (*test cases*).

Validavimas – projekto tinkamumo įvertinimas. Šio žingsnio metu tikrinama, ar kuriamas produktas veikia taip, kaip iš jo tikimasi. Tuo tikslu analizuojamas prototipo veikimas tipinėmis (ar atlieka funkcijas?) ir netipinėmis (ar įmanoma sugadinti) sąlygomis.

Gedimų modelis – prielaidų, kaip veiks schema su defektu, aprašymas [2]. Gedimo modelio tikslas – modeliuoti kuo daugiau fizinių defektų kuo aukštesniame abstrakcijos lygyje. Taip sumažinama defektų, kurie turi būti įvertinti testo sudarymo metu, aibė, be to, supaprastėja įrenginių, kuriems sudaromi testai, modeliai. Plačiausiai naudojami loginių elementų lygio gedimų modeliai. Tranzistorių lygio gedimų modeliai daugumai realių schemų reikalauja per didelių skaičiavimo pajėgumų, o RTL lygio modeliai per daug abstraktūs ir tiksliai neatvaizduoja realių gedimų [3].

Testas – į schemą paduodama įėjimų reikšmių seka, kurios rezultatai teisingai veikiančiai schemai iš anksto žinomi. Testo paskirtis – aptikti gedimus ar defektus schemoje.

Išsamus testas – visa galimų schemos įėjimų aibė. Šis testas pritaikomas tik palyginti nedidelėms schemoms, nes augant schemos sudėtingumui, o tuo pačiu ir įėjimų skaičiui, šio testo ilgis 2^n , čia n – įėjimų skaičius. Dėl šios priežasties didelių schemų testavimui naudojami trumpesni testai, kurie sudaromi:

rankiniu būdu, šiuo atveju testo kūrėjas turi gerai žinoti schemos realizaciją;

atsitiktinai parenkant rinkinius;

naudojant ATPG (automatinių testų generatorių).

Visi šie metodai stipriai sutrumpina testą, tuo pačiu išlaikydami aukštą testo pilnumą. Testo kokybę nusako **testo pilnumas** – testo aptinkamų gedimų dalis visoje galimų schemos gedimų aibėje: $q = \frac{n_d}{n_a}$ (1)

čia *q* – testo pilnumas,

nd – testo aptinkamų gedimų skaičius,

na – visi galimi schemos gedimai.

Scan-testas

Automatinis testų sudarymas schemoms su atmintimi – žymiai sudėtingesnis uždavinys nei testų generavimas kombinacinėms schemoms. Problema yra tai, kad norint pasiekti tam tikrą schemos būseną, gali reikėti labai didelio kiekio nustatančių rinkinių, pavyzdžiui, 32 bitų skaitliuko persipildymo signalo patikrinimui reikia 2^{32} rinkinių. Kita galima problema – jei projektuotojas nenumatė nustatymo į pradinę padėtį, gali reikėti specialiai sudarinėti testinę seką, kuri nustatytų schemą į žinomą padėtį.

Siekiant išvengti tokių problemų, dažnai schema projektuojama taip, kad būtų galima taikyti struktūrinį testavimą (*structured test approach*) [4]. Ši strategija dažnai vadinama „*design for test*“ (DFT), nors šis terminas apibūdina platesnę sritį.

Struktūrinio testavimo atveju, kiekvienas atminties elementas pakeičiamas skenavimo trigeriu. Metodo trūkumas tas, kad gaunamas didesnis schemos plotas bei vėlinimas, tačiau praktikoje šiuos trūkumus atsveria pasiekiamas aukštas schemos stebimumas ir valdomumas. Kombinacinei logikai tarp skenavimo trigerių testai gali būti sudaromi naudojant paprastesnius ATPG įrankius.

Dalinio skenavimo atveju, skenavimo trigeriais pakeičiama tik dalis atminties elementų. Taip pat yra IBM sukurta skenavimo modifikacija lygiui jautrus skenavimas (LSSD – level-sensitive scan design), kurioje skenavimo elementai valdomi skirtingais sinchrosignalais [5, 6]. Scan testavime naudojamas nuoseklus interfeisas, todėl išauga laiko sąnaudos, reikalingos testo įvykdymui. Tai riboja scan testavimo panaudojimą, kai gaminama daug schemų.

BIST

BIST – (*Built-in Self-test*) – metodai, kuriuos naudojant galima papildyti schemą logika, skirta savęs testavimui. Bendras principas – sudaryti testinius rinkinius, paduoti juos į schemos logiką ir analizuoti gautus atsakymus.

Paprasčiausias BIST variantas – LFSR (*linear feedback shift register*) registro pagalba generuojama pseudoatsitiktinė signalų seka, kuri paduodama į testuojamą logiką, o gauti atsakymai į SISR (*serial input signature register*) registrą. SISR registro išėjimuose gaunamas „parašas“, kuris su didele tikimybe bus skirtingas gerai schemai ir schemai su gedimais. Schemos parašas gali būti tikrinamas tiek schemos viduje, tiek ir išvedamas ir analizuojamas išorėje. „Parašų analizės“ metodas buvo sukurtas ir naudojamas aštuntame dešimtmetyje Hewlet-Packard [5].

Yra tikimybė, kad kelios sekos duos tą patį parašą, t. y. kad kai kurie gedimai gali būti nepastebimi naudojant aukščiau aprašytą BIST metodą. Gedimo maskavimo tikimybė gali būti apskaičiuota pagal formulę(2):

$$p = \frac{2^{L-R} - 1}{2^{L} - 1} \qquad (2)$$

Čia: L – testinės sekos ilgis; R – parašo registro ilgis, p – gedimo maskavimo tikimybė. Kai L>>R, gedimo tikimybė p≈2-R.

Šios formulės aprašo gedimų padengimą, bet ne defektų padengimą. Nėra paprasto metodo, kaip šias dvi sąvokas surišti tarpusavyje, nes laikoma, kad visų klaidingų sekų gavimo tikimybė vienoda, bet realiai dėl defektų dažniau atsiranda kai kurios sekos (pvz., visi nuliai). Praktikoje dažniausiai aukštas gedimų padengimas duoda ir aukštą defektų padengimą.

SISR registro idėja gali būti išplėsta, kad būtų galima iškart analizuoti keletą išėjimų, taip gaunant MISR (*multiple-input signature register*) registrą. Papildomos logikos pagalba, MISR registrai gali būti perkonfigūruojami, kad veiktų kaip LFSR arba parašų analizės registrai, tai vadinama BILBO (*built-in logic block observer*). Įdedant testuojamą logiką į grįžtamo ryšio grandines, galima gauti ciklines BIST struktūras, viena iš tokių struktūrų – CSTP (*circular self-test path*).

BIST logika gali būti apjungiama su perimetro skenavimo logika, taip gaunant scanBIST.

1.2.1. Atsitiktinio testavimo metodas

Naudojant atsitiktinio testavimo metodą (*random testing*) į schemą paduodami atsitiktinai sugeneruoti rinkiniai [6]. Šis testų generavimo metodas duoda pakankamai gerus rezultatus didelėms schemoms, kurioms nepritaikomas išsamus testavimas. Priklausomai nuo naudojamos projektavimo technologijos, gali būti naudojamos atsitiktinio generavimo modifikacijos su

nevienodu 1/0 pasiskirstymu įėjimuose. Buvo įrodyta, kad tokiu būdu pasiekiamas didesnis testo pilnumas [7].

Jei schemos realizacija loginių elementų lygyje nežinoma (pvz., kai naudojamos intelektualinės nuosavybės teisėmis apsaugotos posistemės), testai turi būti sudaromi pagal funkcinio lygio schemos modelį. Funkciniame lygyje sudarytas testas nepriklauso nuo konkrečios realizacijos, todėl toks testas gali būti sudaromas ankstyvuose projekto etapuose [8]. Funkcinis automatinis testų sudarymas taip pat gali būti naudojamas siekiant numatyti galimas testuojamumo problemas prieš pasirenkant realizaciją. Kitas funkcinių ATPG metodų privalumas – mažesnė gedimų aibė nei struktūriniuose ATPG. Funkcinio ATPG atveju gedimų skaičius proporcingas įėjimų ir išėjimų sandaugai [9]. Yra pasiūlyta keletas gedimų modelių [10], [11], [12]. Underwood'o pasiūlytas modelis leidžia sudaryti naudojimui tinkamo ilgio testines sekas, tačiau pasiekiamas žemas testo pilnumas. Pomeranz'o ir Reddy [12] pasiūlytas metodas padengia visus galimus kelius schemos viduje, tačiau gaunamos per ilgos testinės sekos. Kompromisinis variantas [10] apjungia abiejų metodų savybes, sumažindamas testinės sekos ilgį, tačiau aptinkama šiek tiek mažiau gedimų.

1.2.2. Vėlinimo gedimų modeliai

Vėlinimo gedimų testavimo tikslas – nustatyti su laikinais signalų parametrais susijusius defektus, dėl kurių prastėja integrinės schemos charakteristikos bei įsitikinti, ar schema teisinga.

Išskiriami keturi vėlinimo gedimų modeliai [13]:

1. perjungimo vėlinimo gedimų modelis (*transition fault delay*);
2. ventilių vėlinimo gedimų modelis (*gate delay fault model*);
3. kelio vėlinimo gedimų modelis (*path delay fault model*);
4. segmento vėlinimo gedimų modelis (*segment delay fault model*).

Perjungimo vėlinimo gedimų modelis kiekvienam keliui schemos viduje priskiria du gedimus: signalo kilimo vėlavimo (*slow-to-rise*) gedimą ir signalo kritimo vėlavimo (*slow-to-fall*) gedimą. Šitie gedimai veikia kaip laikini fiksuotos reikšmės gedimai. Perdavimo gedimui patikrinti reikia dviejų rinkinių: pirmas (nustatymo) rinkinys nustato reikalingą būseną, o antras (perdavimo) rinkinys sukelia norimo signalo pasikeitimo perdavimą į išėjimus. Perdavimo gedimų modelis atvaizduoja defektus, kurie sukelia pakankamai didelį vėlinimą, kad būtų pastebimas loginis gedimas išėjime. Pagrindinis modelio trūkumas tas, kad sunku nustatyti minimalaus pastebimo vėlinimo gedimo dydį. Kitas trūkumas – šis modelis neįvertina, kad keli nedideli atskirų elementų vėlinimai gali sukelti didelį vėlinimą ilgame kelyje.

Ventilių vėlinimo gedimų modelis – konkretesnis perdavimo gedimų modelio variantas, nes čia įvertinami vėlinimų dydžiai. Laikoma, kad žinomos atskirų loginių elementų vėlinimų ribos. Taip pat žinomos tikėtinų vėlinimo gedimų charakteristikos. Gedimas – papildomas tam tikro dydžio kilimo ar kritimo vėlinimas loginio elemento išėjime.

Kelio vėlinimo gedimų modelyje laikoma, kad kiekvienas kelias, kurio vėlinimas didesnis už sinchrosignalo periodą yra gedimas. Kiekvienam keliui, jungiančiam įėjimą su išėjimu, priskiriami du vėlinimo gedimų keliai. Kilimo gedimo keliu signalas perduodamas, kai įėjimo reikšmė keičiasi iš loginio „0“ į „1“, o kritimo gedimo keliu – kai reikšmė keičiasi iš „1“ į „0“.

Signalui praeinant per inverterį, naudojamas kelias atitinkamai pasikeičia į priešingą. Kelio vėlinimo gedimas atvaizduojamas kaip kylančių ir krentančių perėjimų seka nuo įėjimo į išėjimą. Defektų suradimui reikalingos testinių rinkinių poros. Pirmas rinkinys nustato visus kelio elementus į norimą būseną, o antras sukelia signalo pasikeitimus. Šio metodo trūkumas – labai didelė kelių aibė.

Segmento vėlinimo gedimų modelis – ventilių vėlinimo gedimų modelio išplėtimas, kuris modeliuoja vėlinimo gedimus keliuose nuosekliose atkarpose. Atkarpų skaičius pasirenkamas pagal gamybos proceso charakteristikas. Pasirinkus atkarpų skaičių L, gedimų aibę sudaro visi L ilgio kelių segmentai ir visi keliai trumpesni nei L atkarpų keliai. Laikoma, kad kiekvienas kelias, į kurį įeina segmentas su gedimu, turi didesnį nei leidžiama vėlinimą.

1.2.3. „Juodos dėžės“ gedimų modeliai

„Juodos dėžės“ modeliai aprašo sistemos veikimą, neatskleisdami vidinės realizacijos detalių, todėl šių modelių pagalba gali būti matomos tik įėjimų-išėjimų priklausomybės [14]. Gedimų modeliai, vertinantys tik įėjimų-išėjimų sąryšius („juodos dėžės“ gedimų modeliai) yra labai abstraktūs lyginant juos su kitais gedimų modeliais.

1. perdavimų matricos arba įėjimų-išėjimų porų gedimų modelis;
2. trimatės perdavimų matricos arba įėjimo-įėjimo-išėjimo trejeto gedimų modelis;
3. funkcijos termų aktyvavimo gedimų modelis.

Perdavimų matricos gedimų modelis (input-output pin pair) – skirtas „juodos dėžės“ testavimui. Šis modelis analizuoja įėjimų pokyčių įtaką išėjimams ir nesigilina į vidinę schemos

struktūrą. Modelis nepriklausomas nuo testuojamos schemos aprašymo kalbos, vienintelis reikalavimas – kad būtų galima modeliuoti schemos veikimą.

Perdavimų matricos gedimų modelis remiasi prielaida, kad elektroninėje schemoje yra elektriniai keliai nuo kiekvieno įėjimo į vieną ar daugiau išėjimų. Tikslas yra surasti rinkinius, kurie tikrintų šiuos kelius.

Trimatės perdavimų matricos gedimų modelis (input-output pin triplet) panašus į paprastą perdavimų matricos gedimų modelį, tačiau naudojama didesnė gedimų aibė. Įvertinamos priklausomybės trejetuose, sudarytuose iš dviejų įėjimų ir vieno išėjimo.

Funkcijos termų aktyvavimo gedimų modelis (activating function tems) išnaudoja loginių funkcijų savybes. Šiuo atveju ieškoma tiesioginių ir atvirkštinių išėjimo funkcijų termų. Nustatomi termai nevisai tiksliai atitinka realią išėjimo loginę funkciją.

1.3. **Analizės išvados**

- Skaitmeninių įrenginių testavimas – problematiškas procesas. Vystantis technologijoms, didėjant schemų apimčiai, tradiciniai testų sudarymo metodai tampa vis mažiau efektyvūs.
- Testuojant didelės apimties schemas reikalingi labai dideli testavimo įrenginių resursai, todėl pilnas testavimas neįmanomas. Testavimui atlikti naudojami tam tikri testų sudarymo metodai.
- „Juodos dėžės“ testavimui skirti metodai remiasi aukšto abstrakcijos lygio schemų modeliais. Naudojantis šiais metodais įmanoma testuoti ir didelės apimties schemas.

2. PROJEKTINĖ DALIS

Šis skyrius skirtas pateikti išsamią informaciją apie kuriamos sistemos architektūrinius sprendimus. Skyriuje pateikiami keletas skirtingų architektūrinių vaizdų, kurių kiekvienas parodo tam tikrą specifinę sistemos architektūros informaciją.

2.1. Projekto tikslai

Projekto tikslas – sukurti programą, skirtą schemų testavimui naudojant „juodos dėžės“ modelius ir jiems pritaikytus gedimų modelius. Sistemos darbo rezultatai labai priklausys nuo sistemos kodo efektyvumo ir kompiuterio procesoriaus charakteristikų. Užduočių vykdymo laikas gali siekti savaites ar net mėnesius, todėl sistema turi pasižymėti stabilumu.

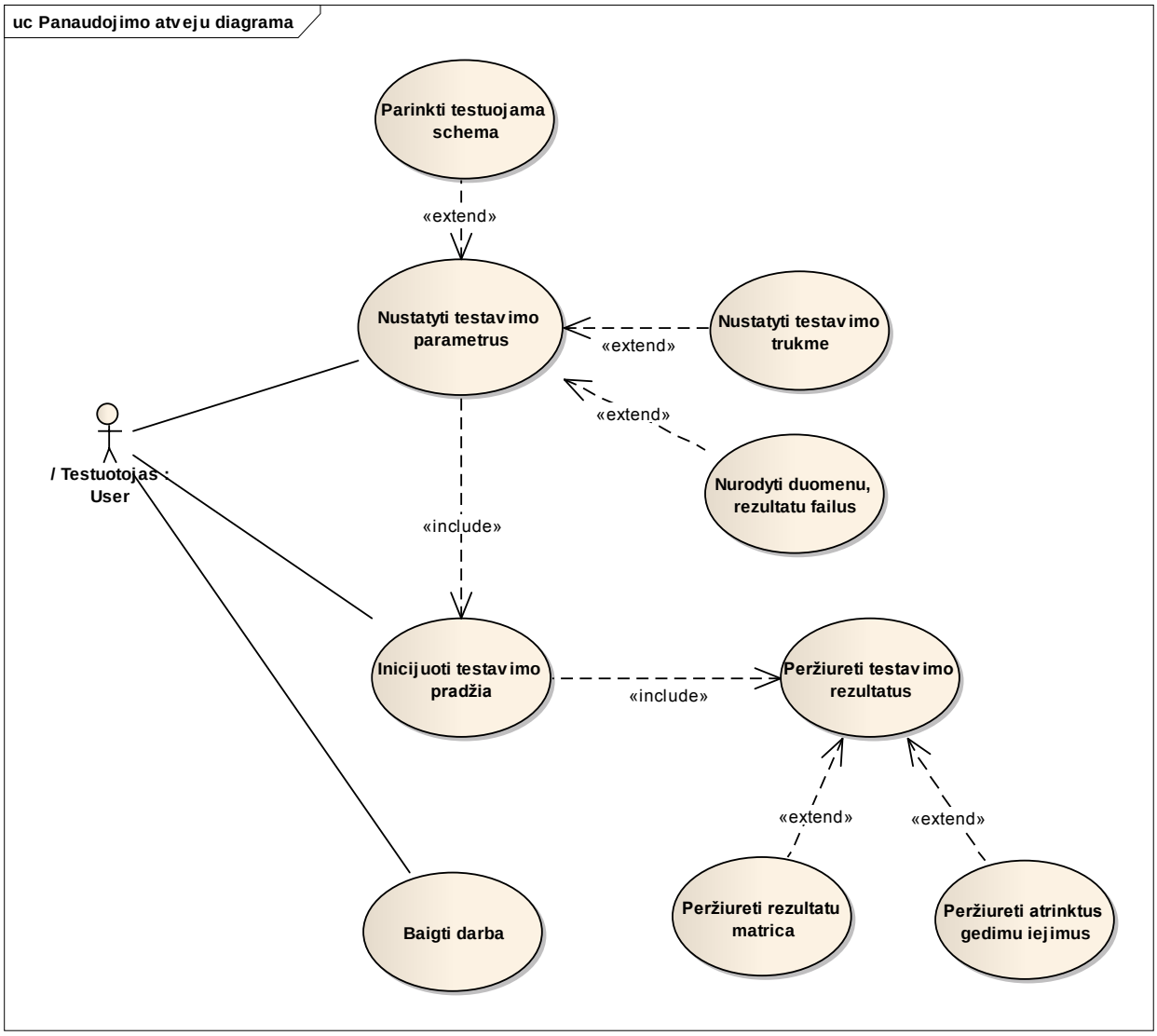
2.2. Priimti techniniai sprendimai

Sistemos duomenų failai pateikiami *.h formatu. Duomenų failų turinys yra programavimo kalba C aprašytos funkcijos.

Atsižvelgiant į tai, jog sistema darbui naudos C kalba aprašytas funkcijas, jai kurti buvo pasirinktas įrankis Microsoft Visual Studio 2010.

Diegimo aplinka: Sistema bus skirta Microsoft Windows Xp/Vista/7 operacinėms sistemoms.

2.3. **Panaudojimo atvejai**



1 pav. Panaudojimo atvejų diagrama

Lentelė 1. PA1 aprašymas

1. PANAUDOJIMO ATVEJIS: Parinkti testuojamą schemą **Vartotojas/Aktorius:** Testuotojas **Aprašas:** Procesas, kurio metu testuotojas parenka testuojamą schemą savo nuožiūra. **Prieš sąlyga:** Įjungta sistema **Sužadinimo sąlyga:** Poreikis ištestuoti tam tikrą schemą. **Po-sąlyga:** Schemos failas sėkmingai parenkamas ir paruošiamas darbui.

Lentelė 2. PA2 aprašymas

2. PANAUDOJIMO ATVEJIS: Nurodyti duomenų, rezultatų failus **Vartotojas/Aktorius:**Testuotojas **Aprašas:** Procesas, kurio metu testuotojas parenka duomenų ir rezultatų failus savo nuožiūra. **Prieš sąlyga:** Įjungta sistema **Sužadinimo sąlyga:** Poreikis atnaujinti jau esamus ar išsaugoti būsimus rezultatus. **Po-sąlyga:** Duomenų ir rezultatų failai sėkmingai parinki ir paruošti pildymui/atnaujinimui.

Lentelė 3. PA3 aprašymas

3. PANAUDOJIMO ATVEJIS: Nustatyti testavimo trukmę **Vartotojas/Aktorius:**Testuotojas **Aprašas:** Procesas, kurio metu testuotojas nurodo norimą testavimo trkmę. **Prieš sąlyga:** Įjungta sistema **Sužadinimo sąlyga:** Poreikis nustatyti testavimo apimtį. **Po-sąlyga:** Nustatoma testavimo trukmė

Lentelė 4. PA4 aprašymas

4. PANAUDOJIMO ATVEJIS: Inicijuoti testavimo pradžią **Vartotojas/Aktorius:**Testuotojas **Aprašas:** veiksmas, kurio metu testuotojas inicijuoja testavimo pradžią naudojant nustatytus parametrus. **Prieš sąlyga:** Nustatyti visi testavimo parametrai **Sužadinimo sąlyga:** Spaudžiamas testavimo pradžios mygtukas **Po-sąlyga:** Testavimas vykdomas naudojant nustatytus parametrus, nustatytą laiką.

Lentelė 5. PA5 aprašymas

5. PANAUDOJIMO ATVEJIS: Peržiūrėti rezultatų matricą **Vartotojas/Aktorius:**Testuotojas **Aprašas:** testavimo metu suformuotos/atnaujintos matricos peržiūrėjimas **Prieš sąlyga:** Įvykdytas testavimas **Sužadinimo sąlyga:** Poreikis peržiūrėti rezultatų matricą. **Po-sąlyga:** Atvaizduojama rezultatų matrica.

Lentelė 6. PA6 aprašymas

6. PANAUDOJIMO ATVEJIS: peržiūrėti atrinktus gedimų įėjimus **Vartotojas/Aktorius:**Testuotojas **Aprašas:** Atrinktų gedimų įėjimų peržiūrėjimas **Prieš sąlyga:** Įvykdytas testavimas **Sužadinimo sąlyga:** Poreikis peržiūrėti atrinktus rezultatus. **Po-sąlyga:** Atvaizduojami atrinkti rezultatai.

Lentelė 7. PA7 aprašymas

7. PANAUDOJIMO ATVEJIS: Baigti darbą **Vartotojas/Aktorius:**Testuotojas **Aprašas:** Procesas, kurio metu užbaigiamas sistemos darbas **Prieš sąlyga:** Šiuo metu nevykdomas testavimas. **Sužadinimo sąlyga:** Poreikis baigti sistemos darbą. **Po-sąlyga:** Sistema sėkmingai išjungiama.

2.4. **Funkciniai reikalavimai**

Pateikiami sistemos funkciniai reikalavimai:

Lentelė 8. FR1 aprašymas

Reikalavimas: 1	**Reikalavimo tipas:**		**Panaudojimo atvejis: 1**
Aprašymas	Norimo schemos failo parinkimas.		
Pagrindimas	• Turi būti galimybė naudoti ir naujai gautus schemų failus, todėl nepakanka integruoti vien esamus.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė naudoti bet kurį schemos failą.		
Užsakovo patenkinimas	3	**Užsakovo nepatenkinimas**	4
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 9. FR2 aprašymas

Reikalavimas: 2	**Reikalavimo tipas:**		**Panaudojimo atvejis:2**
Aprašymas	Duomenų, rezultatų failų nurodymas.		
Pagrindimas	• Turi būti galimybė nurodyti duomenų, rezultatų failus.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė nurodyti duomenų, rezultatų failus.		
Užsakovo patenkinimas	3	**Užsakovo nepatenkinimas**	5
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 10. FR3 aprašymas

Reikalavimas: 3	**Reikalavimo tipas:**		**Panaudojimo atvejis: 3**
Aprašymas	Testavimo trukmės nurodymas sekundėmis.		
Pagrindimas	• Reikalinga galimybė suplanuoti testo trukmę, apimtį		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė nustatyti testo trukmę.		
Užsakovo patenkinimas	4	**Užsakovo nepatenkinimas**	5
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 11. FR4 aprašymas

Reikalavimas: 4	**Reikalavimo tipas:**		**Panaudojimo atvejis: 4**
Aprašymas	Testavimo pradžios inicijavimas.		
Pagrindimas	• Vartotojas turi pats nuspręsti kada visi norimi nustatymai yra įvesti ir pradėti testavimą.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė inicijuoti testavimo pradžią.		
Užsakovo patenkinimas	2	**Užsakovo nepatenkinimas**	5
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 12. FR5 aprašymas

Reikalavimas: 5	**Reikalavimo tipas:**		**Panaudojimo atvejis: 5**
Aprašymas	Rezultatų matricos peržiūrėjimas.		
Pagrindimas	• Vartotojo patogumui, turi būti galimybė peržiūrėti rezultatų matricą pačioje sistemoje.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė peržiūrėti rezultatų matricą.		
Užsakovo patenkinimas	2	**Užsakovo nepatenkinimas**	3
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 13. FR6 aprašymas

Reikalavimas: 6	**Reikalavimo tipas:**		**Panaudojimo atvejis: 6**
Aprašymas	Atrinktų gedimų įėjimų peržiūrėjimas.		
Pagrindimas	• Vartotojo patogumui, turi būti galimybė peržiūrėti atrinktų įėjimų aibę.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė peržiūrėti atrinktus įėjimus.		
Užsakovo patenkinimas	1	**Užsakovo nepatenkinimas**	1
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

Lentelė 14. FR7 aprašymas

Reikalavimas: 7	**Reikalavimo tipas:**		**Panaudojimo atvejis: 7**
Aprašymas	Programos darbo baigimas.		
Pagrindimas	• Vartotojas turi galėti baigti programos darbą.		
Šaltinis	Užsakovas		
Tinkamumo kriterijus	Galimybė baigti programos darbą.		
Užsakovo patenkinimas	1	**Užsakovo nepatenkinimas**	5
Priklausomybės	Nėra	**Konfliktai**	Nėra
Papildoma medžiaga	Nėra		
Istorija	Užregistruotas: 2010-03-27		

2.5. Nefunkciniai reikalavimai

Reikalavimai sistemos išvaizdai (Look and feel)

Sistemos išvaizdai yra keliami tokie reikalavimai:

Aiški vartotojo sąsaja;

Patogumas kartoti panašius veiksmus.

Reikalavimai panaudojamumui (Usability)

Sistemos panaudojamumui yra keliami tokie reikalavimai:

Paprastas sistemos panaudojimas, nereikalaujantis jokių specialių papildomų apmokymų;

Reikalavimai vykdymo charakteristikoms (Performance)

Sistemos vykdymo charakteristikoms yra keliami tokie reikalavimai:

Pakankamai greitas schemų failų atidarymas.

Kiek įmanoma optimizuotas testavimo algoritmas, efektyviai išnaudojantis kompiuterio resursus.

Reikalavimai veikimo sąlygoms (Operational)

Sistema gali veikti tiek stacionarioje, tiek mobilioje darbo vietoje, svarbu tik tai, kad darbo vietos kompiuteris atitiktų šio dokumento 4.5. skyriuje aprašytus reikalavimus.

Reikalavimai sistemos priežiūrai (Maintainability and portability)

Sistemos priežiūrai yra keliami tokie reikalavimai:

Sistema turi būti suprojektuota objektiškai su galimybe ateityje lengvai atlikti jos plėtimo bei tobulinimo darbus pasikeitus organizacijos veiklos taisyklėms arba poreikiams.

Reikalavimai saugumui (Security)

Sistemai nekeliami jokie saugumo reikalavimai.

Kultūriniai-politiniai reikalavimai

Sistemai nėra keliami jokie kultūriniai arba politiniai reikalavimai.

Teisiniai reikalavimai

Sistema turi veikti pagal šalies, kurioje ji įdiegta, įstatymus.

2.6. Sistemos architektūra

Sistemos architektūra pateikiama tokiomis priemonėmis:

Statinis sistemos vaizdas

Sistemos skaidymo į paketus diagrama

Sistemos klasių diagramos

Klasių, sujungtų į paketus, diagramos

Dinaminis sistemos vaizdas

Būsenų diagramos

Sekų diagrama

Veiklos diagramos

Bendradarbiavimo diagramos

Išdėstymo vaizdas

Sistemos komponentų išdėstymo diagrama

2.6.1. Architektūros tikslai ir apribojimai

Sistemos architektūrai keliami tokie tikslai ir apribojimai:

Sistema turi būti suprojektuota objektiškai su galimybe ateityje lengvai atlikti jos plėtimo bei tobulinimo darbus.

Kiek įmanoma optimizuotas testinių atvejų atrinkimo algoritmas, efektyviai išnaudojantis kompiuterio resursus.

Sistemos duomenų failai pateikiami *.h formatu. Duomenų failų turinys yra programavimo kalba C aprašytos funkcijos.

Sistema funkcionuoja lokaliai.

Sistema gali veikti tiek stacionarioje, tiek mobilioje darbo vietoje.

2.6.2. Sistemos statinis vaizdas

Sistemos skaidymas į paketus

Sistema yra išskaidyta į keturis pagrindinius paketus:

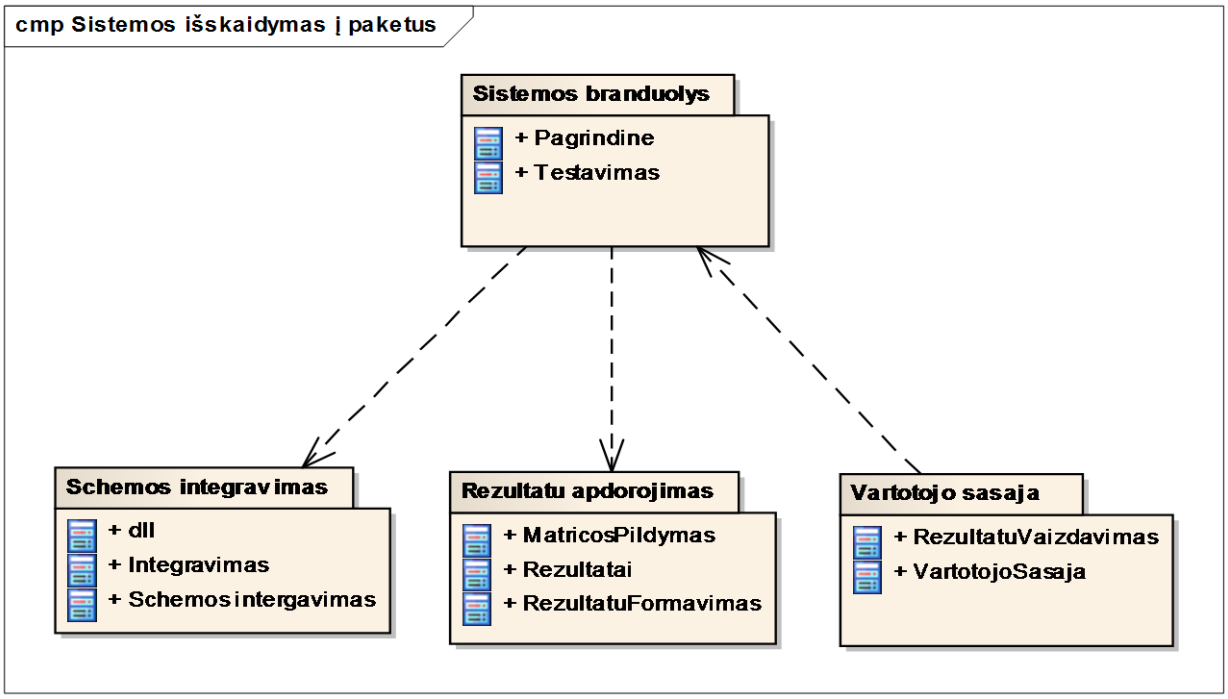
Sistemos branduolys. Paketas, kuriame saugomos sisteminės klasės, būtinos sistemos darbui apjungti.

Vartotojo sąsaja. Paketo paskirtis – testavimo rezultatų atvaizdavimas ir nustatymų keitimas.

Schemos integravimas. Paketas skirtas schemos failo naudojimo paruošimui integruojant jį į sistemą.

Rezultatų apdorojimas. Paketas skirtas duomenų naudojant schemos failą apdorojimui ir rezultatų suformavimui.

Žemiau pateikiama schema, pavaizduojanti sistemos struktūros išskaidymą į paketus aukščiausiame lygyje.



2 pav. Sistemos išskaidymas į paketus aukščiausiame lygyje

Paketų detalizavimas

Paketas “Schemos integravimas”

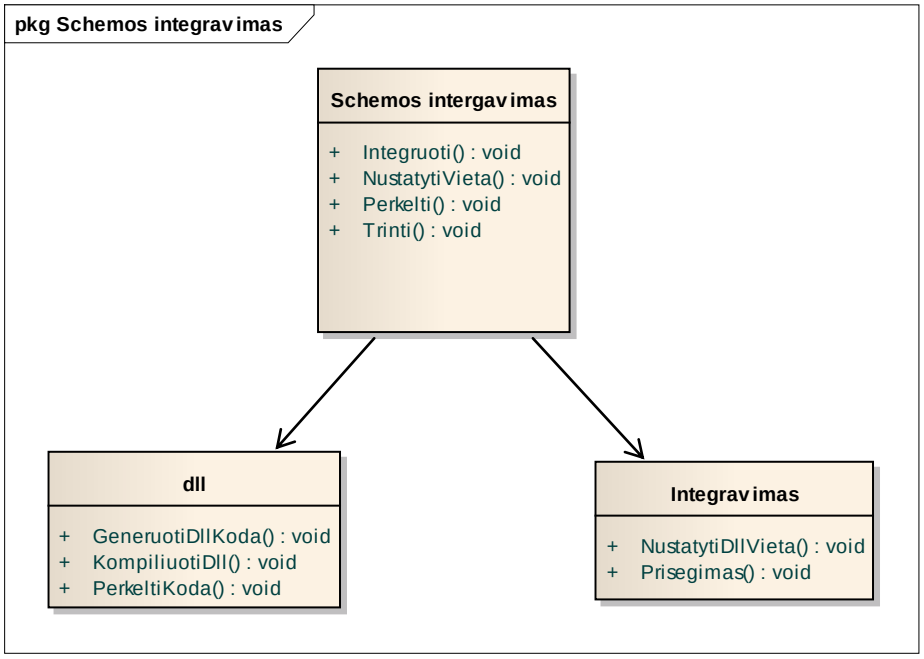
Paketas skirtas schemos failo naudojimo paruošimui integruojant jį į sistemą.

Žemiau trumpai pateikiama paketo klasių paskirtis.

Klasė “dll”. Klasės paskirtis yra paversti schemos .h failą .dll failu tam, kad jį galėtų naudoti sistema.

Klasė “Integravimas”. Šios klasės paskirtis yra naujai sukurto .dll failo integravimas į sistemą.

Klasė “Schemos integravimas” skirta apjungti klasėms “dll” ir “Integravimas”. Nustato sistemos buvimo vietą ir atitinkamai perkelia .dll failą.



3 pav. Paketo „Schemos integravimas“ klasių diagrama

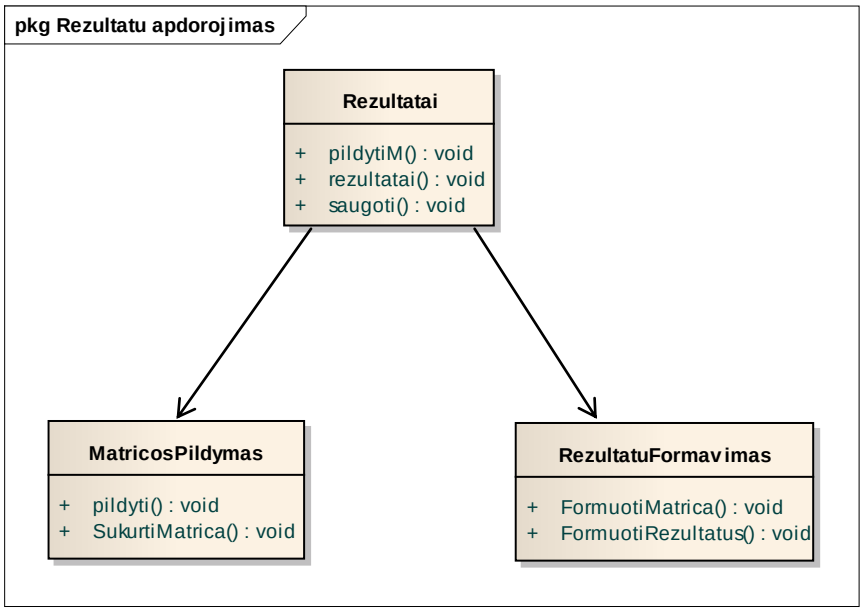
Paketas “Rezultatų apdorojimas”

Paketas skirtas testų, naudojant schemos failą rezultatų formavimui, saugojimui ir paruošimui atvaizduoti.

Klasė “MatricosPildymas”. Klasė skirta rezultatų matricos sukūrimui ir užpildymui.

Klasė “Rezultatai”. Skirta gautų rezultatų saugojimui ir jų kitų dviejų klasių apjungimui.

Klasė “RezultatuFormavimas” skirta paruošti rezultatus atvaizdavimui.



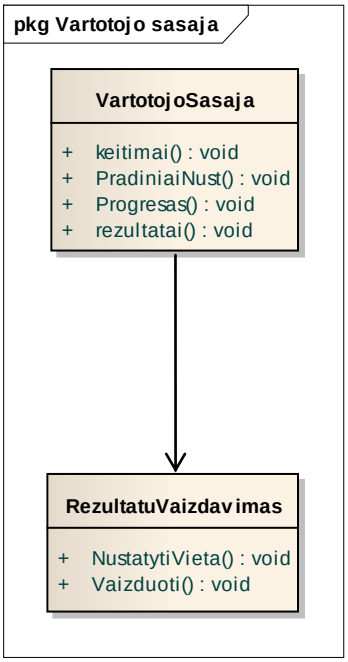
4 pav. Paketo „Rezultatų apdorojimas“ klasių diagrama

Paketas “Vartotojo sąsaja”

Paketas vartotojo nustatymų keitimui, progreso ir rezultatų atvaizdavimui.

Klasė “VartotojoSasaja”. Klasė skirta vartotojo nustatymų keitimui ir “RezultatuVaizdavimas” klasės naudojimui.

Klasė “RezultatuVaizdavimas”. Skirta gautų rezultatų atvaizdavimui.



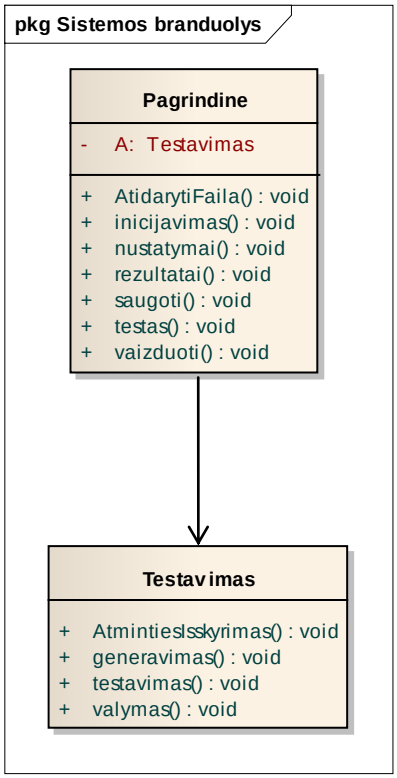
5 pav. Paketo „Vartotojo sąsaja“ klasių diagrama

Paketas “Sistemos branduolys”

Paketas skirtas visų sistemos veiksmų iniciavimui, valdymui ir testavimo veiksmų atlikimui.

Klasė “Pagrindine”. Skirta kitų “Sistemos branduolys” klasių ir sistemos paketų apjungimui.

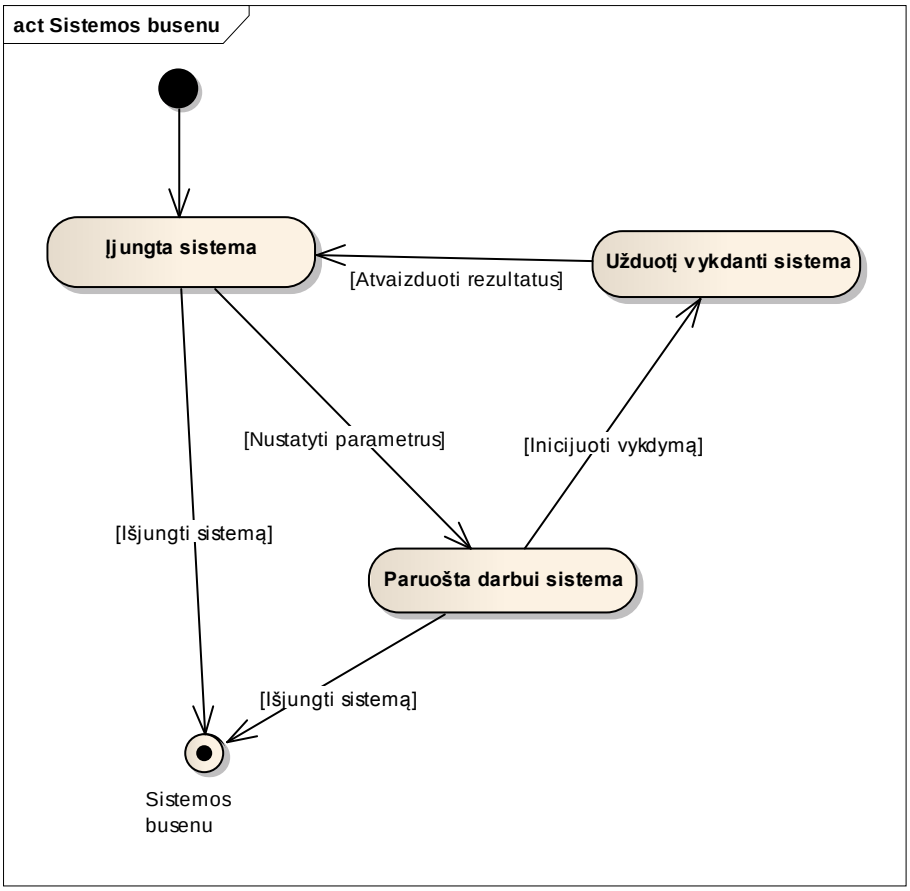
Klasė “Testavimas” skirta atlikti testavimui pagal nustatytus parametrus, įėjimų sekų generavimui.



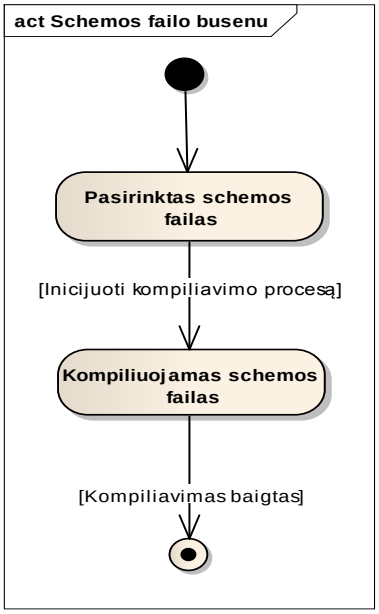
6 pav. Paketo „Sistemos branduolys“ klasių diagrama

2.7. Sistemos dinaminis vaizdas

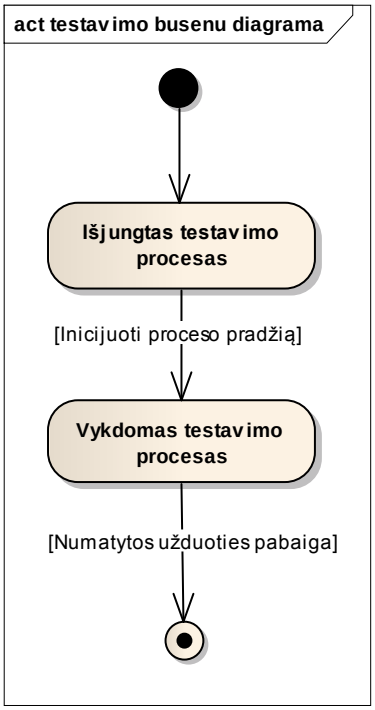
2.7.1. Būsenų diagramos



7 pav. Sistemos būsenų diagrama

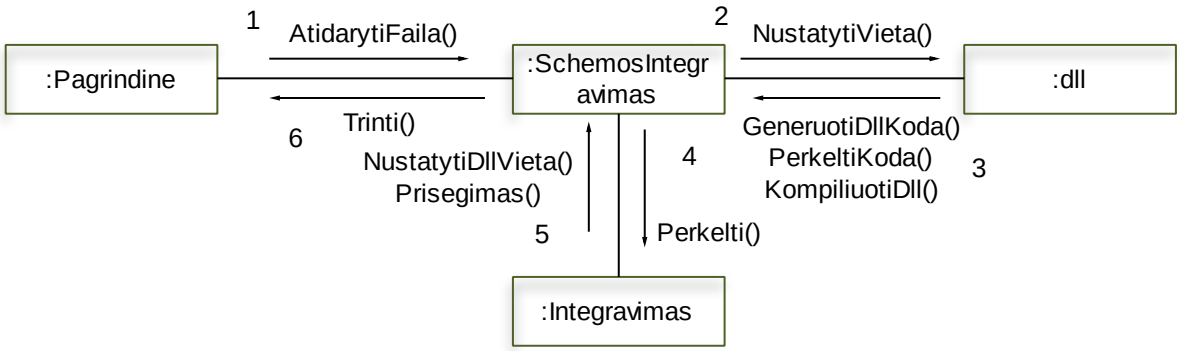


8 pav. Schemos modelių failo būsenų diagrama

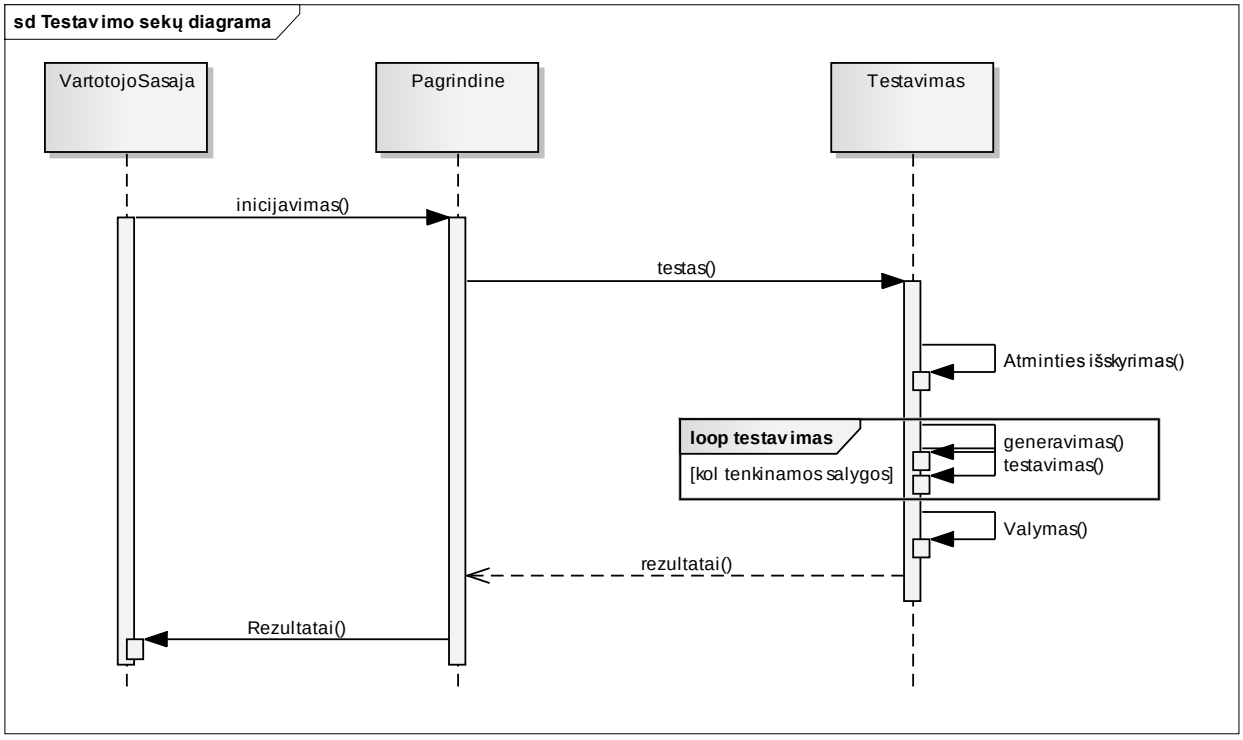


9 pav. Testavimo būsenų diagrama

2.7.2. Sąveikos diagramos

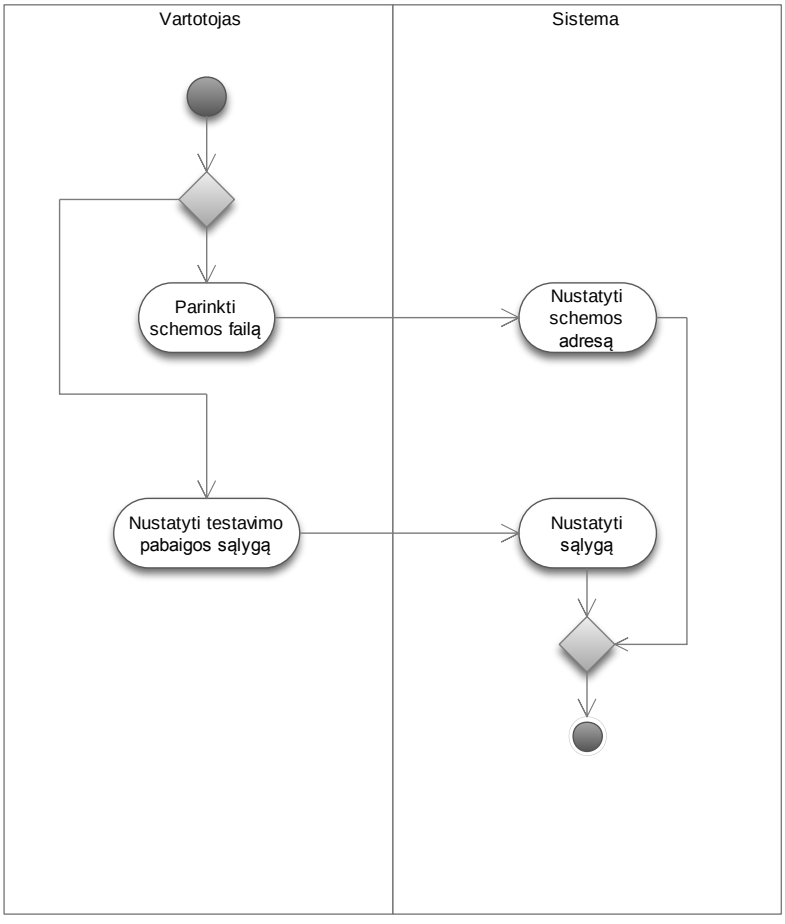


10 pav. Schemos failo integravimo bendradarbiavimo diagrama

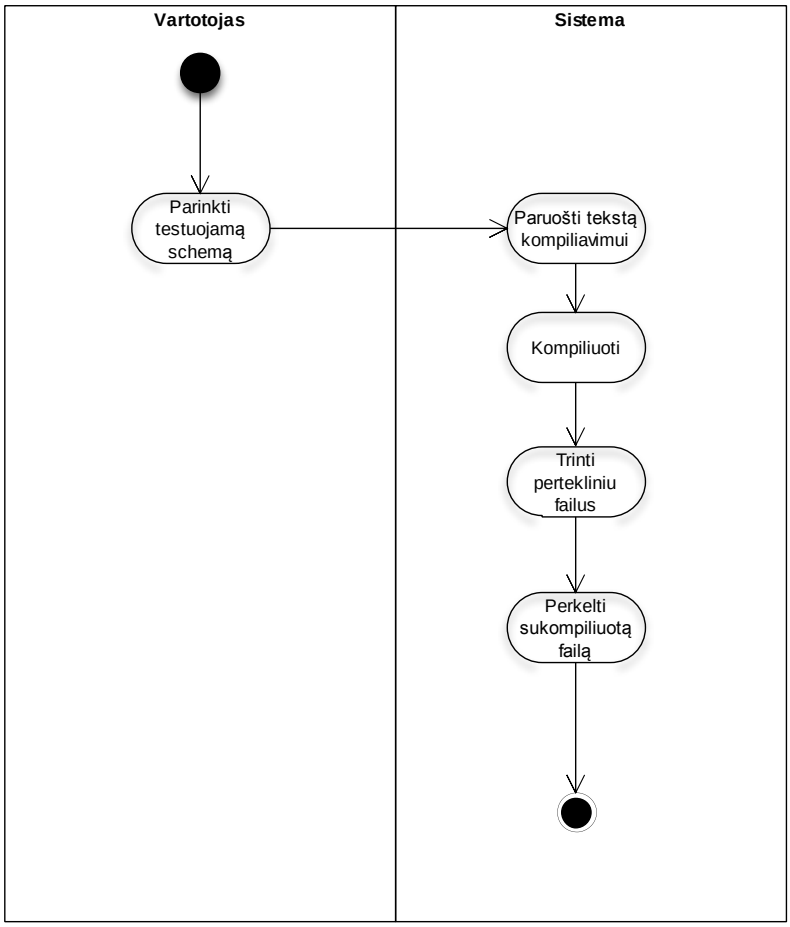


11 pav. Testų sudarymo sekų diagrama

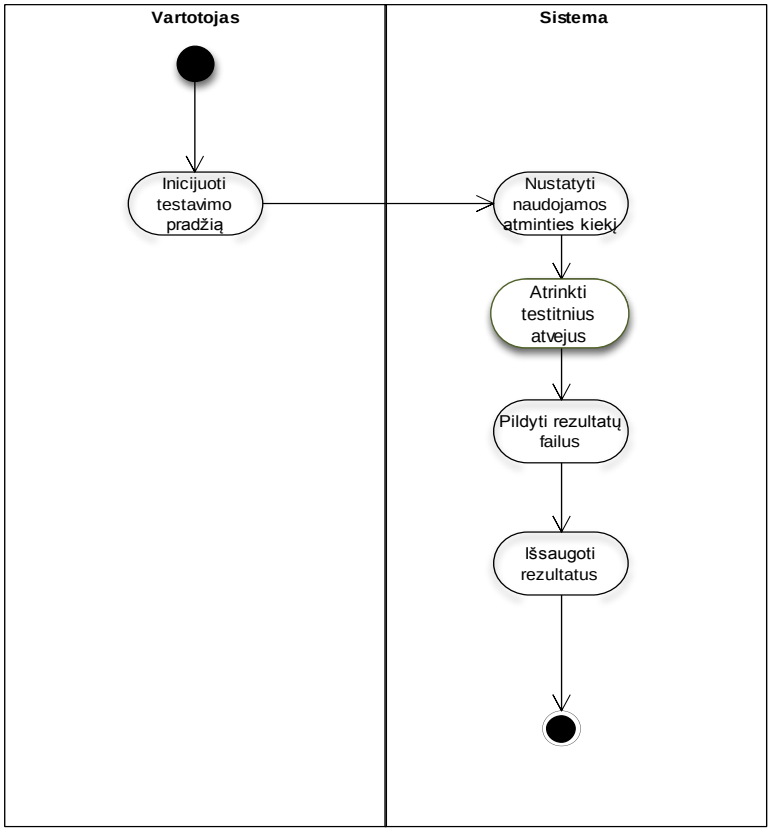
2.7.3. Veiklos diagramos



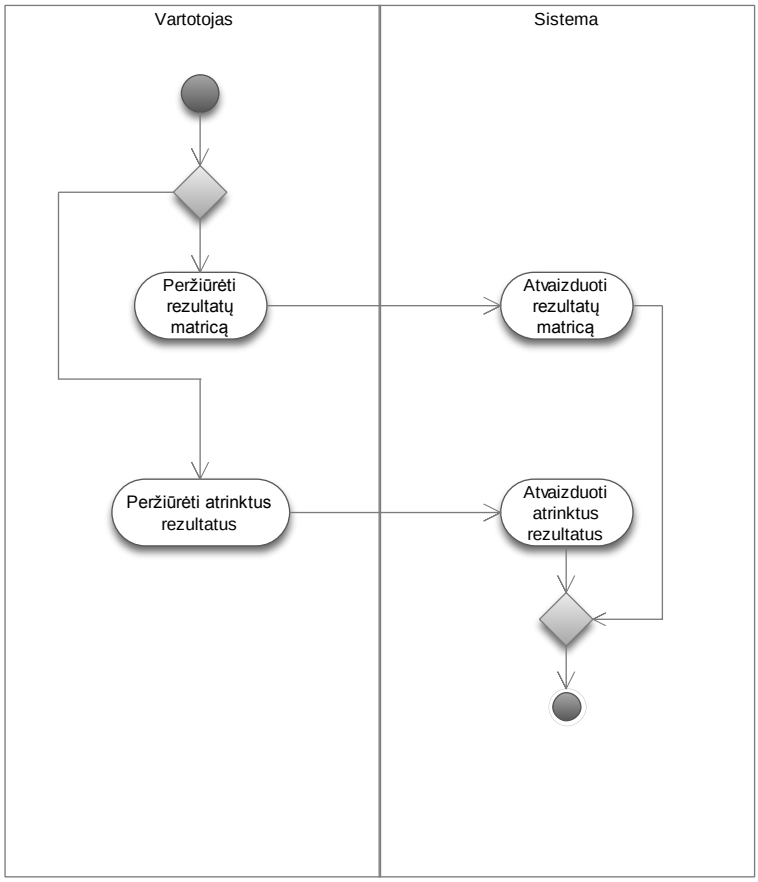
12 pav. Nustatymų keitimo veiklos diagrama



13 pav. Schemos failo integravimo veiklos diagrama



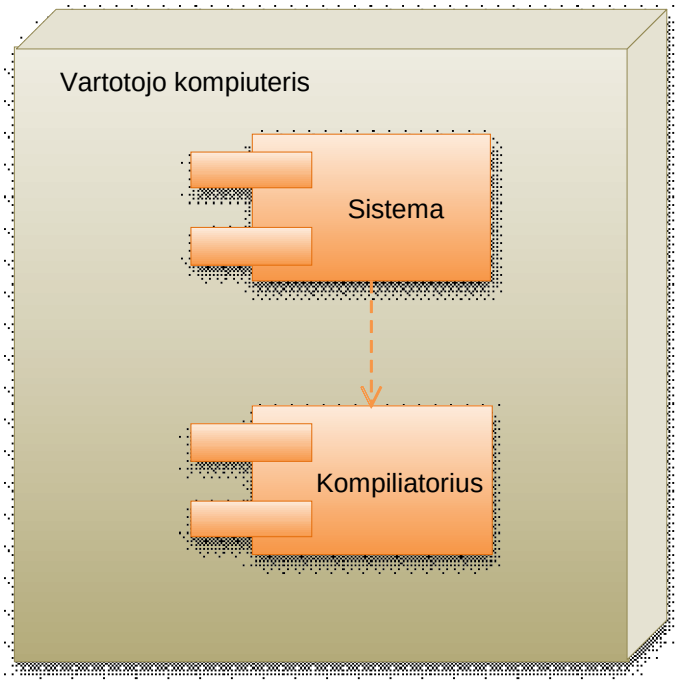
14 pav. Testų sudarymo proceso veiklos diagrama



15 pav. Rezultatų vaizdavimo veiklos diagrama

2.8. **Išdėstymo vaizdas**

Žemiau pateikta sistemos išdėstymo schema. Sistema funkcionuoja lokaliai, naudodama integruotą kompiliatorių.



16 pav. Sistemos išdėstymo vaizdas

Sistema

Tai pagrindinė kuriamos sistemos dalis, turinti savo vartotojo sąsają ir atliekanti veiksmus savarankiškai.

Kompiliatorius

Tai integruota pagalbinė sistema, naudojama schemų failų integravimui. Šiam tikslui naudojamas nemokamas Digital Mars [15] kompiliatorius. Kompiliatorius pateikiamas kaip sistemos komponentas ir nereikalauja papildomo diegimo ar paleidimo.

Kokybė

Sistema yra lengvai diegiama ir funkcionavimui nereikalauja jokių papildomų diegimo veiksmų.

Objektiškai orientuota sistemos struktūra leidžia lengviau pakartotinai panaudoti tam tikras klases ir įtraukti į sistemą naujus modulius.

Pasirinktas kompiliatorius Digital Mars yra nemokamas ir nereikalauja papildomo diegimo.

Dėl paprasto įdiegimo sistema yra lengvai pernešama ir gali funkcionuoti tiek stacionarioje, tiek mobilioje darbo vietoje.

2.9. **Išvados**

Šiame skyriuje pateikiami kuriamos schemų funkcinių testų atrinkimo sistemos reikalavimai, tikslai, architektūros specifikacija. Sistema atvaizduojama tiek statiškai, tiek dinamiškai, naudojant klasių, būsenų, bendradarbiavimo ir sekos diagramas.

Specifikacijos metu išskirti septyni sistemos panaudojimo atvejai. Sistema padalinta į keturis pagrindinius paketus: branduolį, vartotojo sąsają, schemos integravimą ir rezultatų apdorojimą.

Suprojektuota sistema yra lengvai plečiama, taip pat šiek tiek modifikavus sistemos komponentus, juos galima pritaikyti ir kitiems programų kūrimo įrankiams, naudojantiems c++ kalbą.

Dėl pasirinktos architektūros sistema yra lengvai įdiegiama ir pernešama, taip pat labai sumažėja sistemos ir techninės/programinės įrangos nesuderinamumo galimybė.

3. EKSPERIMENTINĖ IR TIRIAMOJI DALIS

Šiame skyriuje bus lyginamas sistemos darbo našumas, naudojant skirtingai integruotus schemų modelių failus ir tiriamos testų generavimo metodo neturinčioms skleidimo registro sinchroninėms nuosekliosioms schemoms patobulinimo galimybės. Taip pat tiriama ir procentinio „1“ pasiskirstymo įtaka atsitiktinai generuojamiems testiniams atvejams.

3.1. Schemų failų integracijos efektyvumo tyrimas

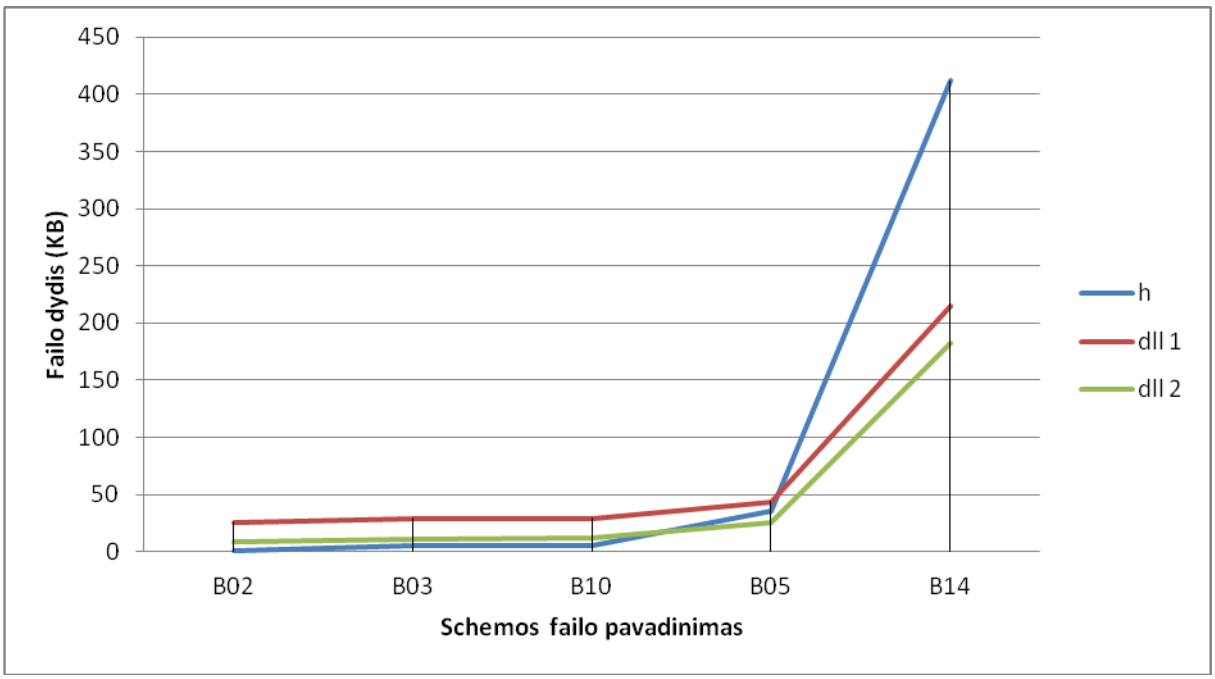
Testuojamų schemų modelių failai sistemai pateikiami kaip *.h tipo failai, parašyti C kalba. Tiesioginė failų integracija nenaudojama, nes ateityje gali atsirasti naujų schemų failų, kurie negalės būti integruoti neperkompiliuojant visos sistemos. Šiuo metu sistema darbui schemų failus integruoja juos konvertuodama į *.dll tipo failus. Tam naudojamas atskiras kompiliatorius Digital Mars. Schemos failo integraciją gali atlikti ir pats vartotojas. Palyginimui taip pat naudojame *.dll failus, sukompiliuotus su Visual Studio, tuo pačiu kompiliatoriumi, kuriuo yra sukompiliuota pati sistema.

Bandysime ištirti kreipinių laikų skirtumus, naudojant .dll ir integruotų .h schemų modelių failus ir įvertinti jų įtaką realiam sistemos darbui su failais.

Lentelė 15. Schemų failai naudojami tyrimui

Schemos failo pavadinimas	B02	B03	B10	B05	B14
*.h failo dydis (bytes)	956	4,420	5,774	36,798	421,080
*.dll 1 failo dydis (bytes)	26,140	28,700	29,212	44,060	219,676
*.dll 2 failo dydis (bytes)	8,704	10,752	11,776	26,624	186,368

Lentelėje 15 pateikiami visų tipų naudojamų schemų modelių failų dydžiai. *.dll 1 yra schemos modelių failai, sukompiliuoti naudojant Digital Mars kompiliatorių, *.dll 2 yra failai, sukompiliuoti naudojant Visual Studio kompiliatorių.



17 pav. Schemos modelio failo dydžio priklausomybė nuo tipo

Iš grafiko 17 pav. matome, jog kodo optimizavimas kompiliuojant *.dll tipo failus yra nežymus mažos apimties failams, tačiau kompiliuojant didesnius failus, matomas ženklus failų dydžių skirtumas. Taip pat matomas ir skirtumas tarp Digital Mars ir Visual Studio kompiliatorių. Iš jo galime numatyti, jog sistemos darbas naudojant su Digital Mars kompiliatoriumi sukompiliuotus failus (dll 1) bus lėtesnis nei naudojant su Visual Studio sukompiliuotus failus (dll2).

Tyrimui naudojamo kompiuterio parametrai: Procesorius: Athlon x2 3,2 GHz, Operatyvioji atmintis: 4 GB RAM, Operacinė sistema: Windows 7 64 bit.

Tyrimui sukurta atskira mini sistema, kuri nurodytą kiekį kartų kreipiasi į integruotą schemos modelio failą ir skaičiuoja tam sugaištą laiką. Sistema turi 16 lentelėje išvardytus schemų modelių failus, integruotus visais būdais.

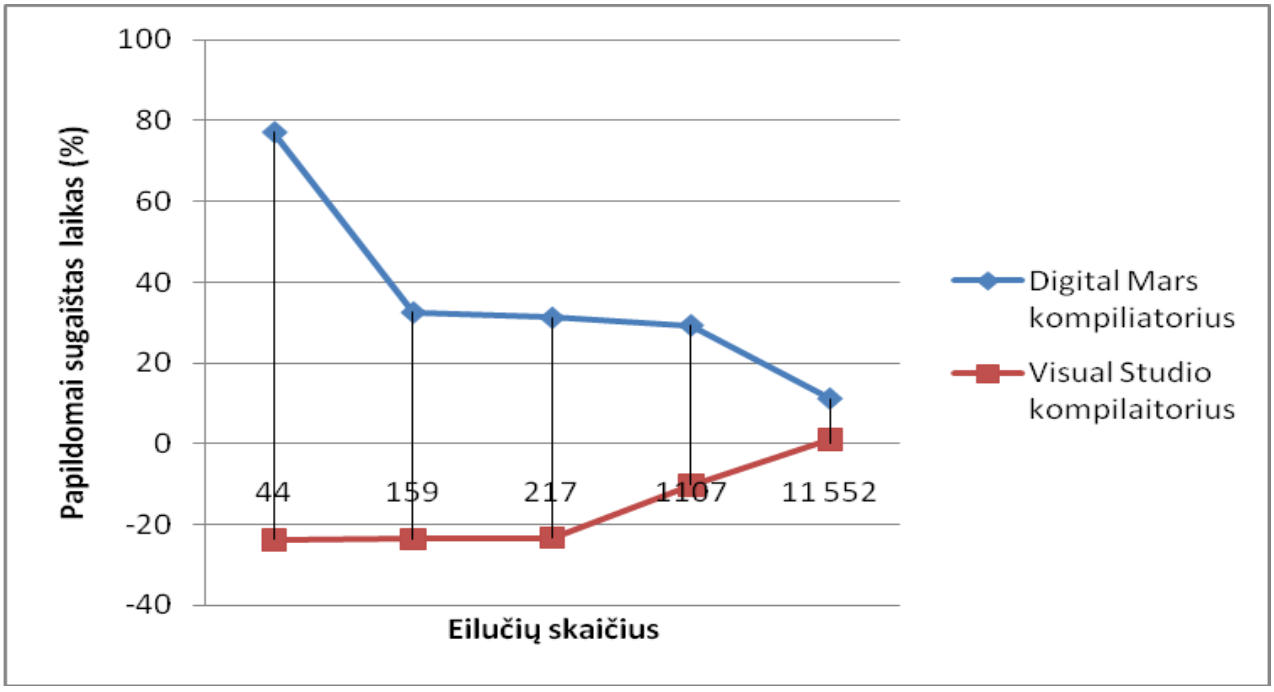
Kreipiniams sugaištas laikas skaičiuojamas pagal procesoriaus taktų kiekį naudojant formulę: $\frac{k_{viso}}{k_{daznis}}$.

Sistemai vykdant darbą skaičiuojamas procesoriaus taktų kiekis, jai darbą baigus, skaičiavimas sustabdomas, ir gautas taktų kiekis k_{viso} padalijamas iš procesoriaus dažnio k_{daznis}. Taip gaunamas tikslus sugaištas darbo laikas. Kadangi kreipiniams į failus naudojame tik vieną giją, galime teigti, jog esant minimaliam kompiuterio apkrovimui vienas procesoriaus branduolys yra pilnai išnaudojamas sistemos darbui, arba kad tikrasis procesoriaus darbo laikas būtų apytikriai tiek kartų mažesnis, kiek tame kompiuteryje procesorius turi branduolių.

Atliekant testą, į kiekvieną schemos failą kreipiamasi po 1 000 000 kartų visais integracijos būdais.

Lentelė 16. Eksperimento rezultatai

Schemos failo pavadinimas	B02	B03	B10	B05	B14
Kreipinių trukmė į *.h failą (s)	0.0767	0.9858	1.1595	11.2009	850.308
Kreipinių trukmė į *.dll failą(s) 1	0.1358	1.3066	1.5213	14.4884	945.961
Kreipinių trukmė į *.dll failą(s) 2	0.0585	0.7534	0.8895	10.0558	859.4206



18 pav. Kreipinių trukmės skirtumo procentinė priklausomybė nuo eilučių skaičiaus

Iš grafiko 18 pav. matome, jog dirbant su mažais failais sistemos darbas naudojant *.dll failus, sukompiliuotus su Visual Studio kompiliatoriumi yra našesnis. Taip yra dėl to, kad kompiliuojant *.dll failus kompiliatorius optimizuoja kodą ir jame esantys metodai tampa efektyvesni. Be to, kompiliuojant *.h tipo failą kartu su likusiu sistemos kodu, viskas sukompiliuojama į CIL (Common Intermediate Language) tam, jog sistema palaikytų .NET framework, o įjungus sistemą CIL kodas dar kartą kompiliuojamas į operacinei sistemai suprantamą kodą. Kompiliuojant *.dll failus tai nėra daroma, todėl šiuo atžvilgiu *.dll failų našumas yra šiek tiek didesnis (18 pav.). Tačiau didėjant schemos modelio failo dydžiui, pastebimas vis lėtesnis sistemos darbas (lyginant su *.h failais), taip vyksta dėl lėtesnio kreipimosi į *.dll failus (eksperimento atveju, didėjant failų dydžiui, didėja ir kreipinių dydis). Pagal darbo su didžiausia schema rezultatus matome, jog dar padidėjus schemos modelio failo apimčiai darbas su *.h tipo modelių failais tampa spartesnis, kitaip sakant, persveria optimizuoto kodo *.dll failuose efektyvumo pranašumą. Taip pat pastebimas Digital Mars ir Visual Studio

kompiliatorių efektyvumo skirtumas. Tačiau didėjant schemų apimčiai šis skirtumas vis mažėja, nes didėja papildomai užgaištas laikas kreipiniams į *.dll failus.

Papildomai užgaištas laikas (18 pav.) skaičiuojamas pagal formulę: $\left(\frac{D}{H} \cdot 100\right) - 100$, kur D yra kreipiniams į *.dll failą sugaištas laikas, H yra kreipiniams į *.h failą sugaištas laikas. Taip gaunamas laikas (%), kuris yra papildomai sugaištamas naudojant *.dll integracijos tipą, lyginant su *.h integracijos tipu.

Atlikus tyrimą matome (18 pav.), jog integracijos tipas labiausiai įtakoja tik darbą su nedidelėmis schemomis, kurių bendri darbo laikai nėra dideli ir šiuo atžvilgiu gautas skirtumas yra gana neaktualus. Integracijos tipo pakeitimas galėtų šiek tiek pagerinti darbą su didelėmis schemomis, kurių darbo laikai yra gan dideli, tačiau atsižvelgiant į tyrimo rezultatus (16 lentelė) galima teigti, jog didėjant schemos failams ir darbo apimčiai, integracijos tipas vis mažiau įtakoja bendrą darbo trukmę ir menkai padidintų sistemos darbo efektyvumą naudojant tiesiogiai integruotus *.h failus. Sistemai dirbant normaliu režimu, kreipimasis į failus yra tik dalis viso sistemos darbo, todėl procentinės sistemos darbo skirtumų išraiškos sistemai funkcionuojant normaliai būtų ženkliai mažesnės.

3.2. **Testų generavimo sinchroninėms nuosekliosioms schemoms tyrimas**

Puslaidininkiai prietaisai tampa vis sudėtingesni atsižvelgiant į tranzistorių kiekį juose, dažnį ir integracijos laipsnį. Nanometrinių procesų taikymas sąlygoja naujų defektų, kurie įtakoja signalų laikus, klasių atsiradimą. Defektų spektras pasipildė naujais defektais, kurie ne visada yra aptinkami tradiciniais statiniais testais, dar vadinamais fiksuotos reikšmės gedimų testais.

Testų generavimas yra kuriamas dviem kryptimis. Paprastai testas schemai yra generuojamas struktūriniame lygyje. Šiuo atveju, problema yra generacijos laikas, nes jis tiesiogiai įtakoja produkto pateikimo rinkai laiką. Ši užduotis yra gana sudėtinga, ypač nuosekliosioms schemoms, todėl kuriant tokias schemas yra taikoma lengvai testuojamo projektavimo metodika (DFT). Tai padeda sumažinti testų kūrimo kainą. Bet šis toks projektavimas įgalina sinchronines nuosekliąsias schemas nustatyti į būsenas, kurių schema negali įgyti normaliai funkcionuodama. To pasėkoje, šis projektavimas įgalina naudoti schemų testavimui rinkinius, kurie yra nepritaikomi per funkcinį schemos darbą. Tai veda į bereikalingą produkcijos praradimą. Kita DFT metodo blogybė yra ta, kad padidėja schemos vėlinimas.

Kita svarbi testų generavimo kryptis yra aukšto abstrakcijos lygio funkcinių testų kūrimas. Pradiniais kūrimo etapais struktūrinė dizaino realizacija yra nežinoma, todėl testų

generavimo užduotis yra dar sudėtingesnė, nes testas turi būti generuojamas visiems realizacijos atvejams. Tačiau testo kūrimas gali būti atliekamas paraleliai su kitomis kūrimo stadijomis. Šiuo atveju, testų generavimo laikas nėra kritiškai svarbus. Per kūrimo procesą pagal specifikaciją yra sukuriamas programinis schemos prototipas. Programinis schemos prototipas simuliuoja schemos funkcijas, pagal duotas įėjimų reikšmes gali gauti išėjimų reikšmes. Funkcinis testas gali būti generuojamas remiantis schemos programiniu prototipu. Funkcinis testas taip pat yra vertingas intelektualios nuosavybės (IP) komponentų testavimui, kurių realizacija yra nežinoma schemos kūrėjams.

Funkcinis testas paprastai yra žymiai didesnis už testą, sukurtą pagal schemos realizaciją, siekiant užtikrinti gerą gedimų padengimą dideliam realizacijų kiekiui. Kai schemai vietoj aukšto abstrakcijos lygio modelio pritaikoma tam tikra realizacija, atsižvelgiant į šią realizaciją galima atlikti funkcinių testų minimizavimą, kad būtų pašalinti testiniai atvejai, kurie neaptinka šios realizacijos gedimų. Vėliau gali būti suformuojamas neaptiktų gedimų sąrašas ir panaudojant deterministinius metodus įmanoma aptikti šiame sąraše esančius gedimus. Funkcinio testo pritaikymas tam tikrai realizacijai yra žymiai paprastesnė užduotis nei testų generavimas nuo pradžių. Adaptacijos procesas ilgai netrunka ir menkai įtakoja bendrą kūrimo laiką. Tai yra didelis funkcinių testų pranašumas. Jei aukšto lygmens aprašas yra persintezuojamas, funkcinis testas išlieka toks pat. Jis tik turi būti pritaikytas naujai realizacijai.

Atsitiktinės testų sekos gali būti naudojamos schemų testavimui gamybos metu ir modeliavimu paremto projektavimo verifikavimui [16-19]. [18, 19] atlikti tyrimai rodo, jog funkciniai testai, kurie yra sukurti naudojant atsitiktinį testų generavimą, pasiekia didesnį gedimų padengimo laipsnį nei testai, kurie yra sukurti naudojant deterministinius ATPG įrankius. Bandysime ištirti atsitiktinių testų, skirtų nuoseklių sinchroninių schemų testavimui, generavimo patobulinimo galimybes.

3.3. **Susieti darbai**

Nuosekli schema yra sudaryta iš dviejų dalių: kombinatorinės logikos ir trigerių, kurie yra valdomi sinchro signalu. Tik pirminiai schemos įėjimai (PI) yra kontroliuojami, ir pirminiai išėjimai (PO) yra matomi. Tam, kad būtų atliktas vėlinimo gedimų testavimas neturinčioms skleidimo registro nuosekliosioms schemoms, vykdomi tokie žingsniai: (a) schema nustatoma į žinomą būseną, (b) gedimų aktyvacija, kad būtų stimuliuojamas testuojamas gedimas (c) gedimo efekto sklidimas į pirminį išėjimą (PO). Gali prireikti kelių įėjimo vektorių schemos inicijavimui ir gedimo efektų perdavimui į pirminį išėjimą (PO) [20].

Atsitiktinių testų sekų taikymas neturinčių skleidimo registro nuosekliųjų schemų testavimui aprašomas [16, 17]. [16] parodyta, jog atsitiktinės pirminių įėjimų sekos pasiekia tik žemą sinchroninių nuosekliųjų schemų gedimų padengimą, todėl, kad atsitiktiniai generuojami testai pakartotinai priskiria tuos pačius dydžius būsenų kintamųjų poaibiams. Siekiant pagerinti gedimų padengiamumą, [16] yra apibūdinta procedūra, modifikuojanti atsitiktinę pirminių įėjimų seką tam, kad būtų pašalintas įėjimų vektorių, kurie sinchronizuoja būsenų kintamųjų poaibius, atsiradimas. Yra pademonstruota, jog ši procedūra turi didelę įtaką gedimų padengimui, naudojant atsitiktines pirminių įėjimų sekas.

Tyrimas [17] rodo, jog pakankamai ilgos atsitiktinės testų sekos pasiekia geresnį gedimų padengimą nei testai, atliekami deterministiniais ATPG įrankiais. [17] šaltinis aprašo ilgų testų sekų dalinimą į mažesnes sekas. Šio metodo taikymas padidina pradinės atisitiktinai sugeneruotos testų sekos gedimų padengimą ir sumažina testo ilgį, pašalinant mažas sekas, kurios neatranda naujų gedimų.

[21] šaltinyje aprašomame metode įėjimų vektoriai, įeinantys į testavimo seką, yra nustatomi iš anksto. Testinės sekos generavimo metu iš anksto apskaičiuojamas tokių įėjimo vektorių rinkinys, kad gauta testinė seka turėtų kuo didesnį gedimų padengimą. Testo generavimo procesas tuomet ieško tam tikrų vektorių tenkinančių sąlygas, o ne generuoja atsitiktinį įėjimą, kuris yra ribotas tik schemos įėjimo ilgiu. Šis metodas pagerina tik konstantinių gedimų radimą.

Vėlinimo gedimų testai, skirti neturinčioms skleidimo registro sinchroninėms nuosekliosioms schemoms gali būti kuriami ir funkciniame lygyje, naudojant programinį prototipo modelį [18, 19, 22].

Funkciniais vėlinimo modeliais paremtas testų generavimas neturinčioms skleidimo registro sinchroninėms nuosekliosioms schemoms yra aprašomas [18] šaltinyje. Neturinti skleidimo registro nuoseklioji schema yra traktuojama kaip modelis, susidedantis iš k kombinatorinės logikos schemos kopijų. K dydis apibūdina sinchro sekos ilgį. Šis modelis yra ypač naudingas, kai perjungimo vėlinimo gedimams aptikti reikalingos ilgos testų sekos. [19] šaltinyje yra funkciniame lygmenyje generuojami testai, kurie skirti aptikti ventilinio lygio perjungimo vėlinimo gedimams. Remiantis eksperimentų rezultatais yra sukurtas karkasas, skirtas neturinčių skleidimo registro nuosekliųjų schemų testų generavimui. [18, 19] šaltiniuose yra pateikiami eksperimentiniai rezultatai, skirti parodyti vėlinimo testų, sukurtų funkciniame lygyje ir naudojančių funkcinius gedimų modelius, pranašumą perjungimo vėlinimo testams, sudaromiems tranzistorių lygyje, naudojant deterministinį testų generatorių.

Kang et al. [22] pasiūlė įėjimų/išėjimų persiuntimo (TRIO) gedimų modelį, skirtą funkcinių testų atrinkimui registrų perdavimo lygmenyje (RTL). Jis yra kuriamas atsižvelgiant į

modelio pirminius įėjimus, pirminius išėjimus ir būsenos kintamuosius. Tačiau šis modelis yra apytikslis, nes jis nesąlygoja signalo perdavimo iki pat pirminių išėjimų.

Sinchroninėms nuosekliosioms schemoms svarbi problema yra jų inicijavimas, nes tam, kad jos veiktų teisingai, schemos turi pradėti darbą nuo žinomos pradinės būsenos. Ši sąlyga taip pat galioja ir schemų testų generavimui [23]. Šiame darbe neskiriame dėmesio inicijavimui, tai yra, laikome, jog schemos turi *reset* įėjimą.

Toliau darbe naudosime terminus „funkcinis vėlinimo gedimas“ ir „segmentas“, kurie atitinkamai aprašomi šaltiniuose [19] ir [17].

Apibrėžimas 1. Funkcinis vėlinimo gedimas yra struktūrizuotas sąrašas, (I, O, tI, tO), kur I yra pirminis įėjimas x_i (i=1, …, n) arba prieš tai buvusi atminties elemento būsena q_l (l=1, …, v), O yra pirminis išėjimas y_j (i=1, …, m) arba po to esanti atminties elemento būsena p_k (l=1, …, v), tI yra signalo kilimas arba kritimas įėjime, tO yra signalo kilimas arba kritimas išėjime. [19]

Apibrėžimas 2. Segmentas yra įėjimų seka, prasidedanti schemos inicijavimu. Segmentas yra sudarytas iš dviejų tipų įėjimų sekų: pirmoji dalis yra schemos inicijavimo įėjimų rinkiniai, kurie nustato schemą į žinomą būseną, antroji dalis yra testinių atvejų rinkiniai. [17]

Testinių atvejų įėjimų rinkinių ilgis apsprendžia segmento ilgį.

3.3.1. Atsitiktinio generavimo patobulinimas

Tirsime „1“ ir „0“ procentinio pasiskirstymo generuojamoje atsitiktinių testinių atvejų sekoje įtaką testo kokybei. Atsitiktinai generuojame visą segmentą S, tada modeliuodami gedimus tikriname, kokius funkcinius vėlinimo gedimus S aptinka. Jei S aptinka nors vieną dar neaptiktą gedimą, jis yra įrašomas į galutinį testą.

Eksperimentams buvo pasirinkti ITC’99 neturinčių skleidimo registro nuoseklių sinchroninių schemų etaloniniai modeliai. Tirsime funkcinius vėlinimo gedimus. Etaloniniai schemų modeliai yra parašyti C programavimo kalba. Tam tikrų schemų segmentų ilgiai yra paimti iš [17].

17 lentelėje yra atvaizduoti apibendrinti „0“ ir „1“ paskirstymo eksperimentų rezultatai. Kiekvienai schemai (Schema) yra pateikiamas aptiktų funkcinių vėlinimo gedimų skaičius ir (VK sk.) ir sekos ilgis (Ilgis). „1“ pasiskirstymas yra pateikiamas procentais. Užrašai „70%“ (60% ir t.t.) reiškia, jog vidutiniškai tiek procentų „1“ yra kiekviename atsitiktinai sugeneruotame testiniame rinkinyje.

Lentelė 17. „1“ pasiskirstymo įtaka testo kokybei

Schema	70%		60%		50%		40%		30%	
	VK sk.	Ilgis	VK sk.	Ilgis	VK sk.	Ilgis	VK sk.	Ilgis	VK sk.	Ilgis
b04	1797	4410	1797	3755	1797	3753	1797	3990	1795	4663
b06	89	85	89	64	89	75	89	73	89	96
b07	69	433	69	617	69	541	70	624	70	629
b08	206	1475	206	893	206	931	206	1383	206	1449
b09	526	2283	526	2393	526	2927	526	3569	526	4790
b10	292	654	292	627	292	654	291	830	290	1013
b11	957	38264	957	23979	957	36268	957	21742	956	28871
b12	272	14280	293	11376	295	7360	297	3719	310 637*	6595 32636*
b13	353	11517	352	12098	353	11906	353	11350	353	16941
Vidurkis	526	7736	529	5463	529	6564	529	4491	530	6013
b14	19739	353419	19244	362841	19120	364663	20184	367587	26613 33224 **	405283 450066 **
b15	16602	23102	14665	20487	13402	17489	12805	16735	10816	13486
Vidurkis b14, b15	18171	188261	16955	191664	16261	191076	16495	192161	18715	209385
Bendras vidurkis	3718	40902	3499	39921	3373	40597	3416	39237	3820	43983

*- pasiskirstymas 5%; ** - pasiskirstymas 20%

Kad būtų gauti patikimesni rezultatai, kiekvienam atvejui buvo atliekami du testų generavimai. Vienetų pasiskirstymas buvo tiriamas tarp 30%-70%. 18 lentelėje yra pateikiami rezultatų vidurkiai. Atskiroms schemoms testų generavimo laikai yra vienodi kiekvienam „1“ pasiskirstymui. Testo kokybę apsprendžia du faktoriai – rastų gedimų kiekis ir testo ilgis. Mes laikome geresniu testu tą testą, kuris randa daugiau gedimų. Jei yra du ar daugiau testų, kurie randa vienodą skaičių gedimų, geriausias yra trumpiausias. 18 lentelėje yra pažymėti geriausi rezultatai. Be to, mes darome prielaidą, jog testų generavimo procesas prisisotina (visi arba beveik visi gedimai yra aptikti), kai iš viso sugeneruotų ir išanalizuotų segmentų skaičius yra didesnis už paskutinio atrinkto segmento eilės numerį, padaugintą iš koeficiento K.

Išanalizuokime 17 lentelėje pateiktus rezultatus. Testų generavimo procesas schemoms b04-b13 prisisotina labai greitai. Mažiausias K dydis buvo 6, tačiau [19] šaltinyje teigiama, jog netgi kai K yra 2, gaunami patenkinamos kokybės testai. Palyginę kiekvienos schemos rezultatus matome, jog testų kokybė yra mažai priklausoma nuo „1“ pasiskirstymo. Todėl galime daryti išvadą, jei testų generavimo procesas prisisotina greitai, tada „1“ ir „0“ pasiskirstymas atsitiktinai generuojamuose įėjimų rinkiniuose nėra svarbus rodiklis. Schemoms b14 ir b15 yra kitaip. Didžiausias gautas k dydis buvo 1.12, tai reiškia, jog testų generavimo procesas buvo toli gražu neprisisotinęs. Matome, jog šiuo atveju, „1“ pasiskirstymas yra svarbus faktorius testo kokybei. Aptiktų funkcinių vėlinimo gedimų kiekis labai skiriasi kiekvienam „1“

pasiskirstymo atvejui. Schemai b14, didžiausias aptiktų gedimų kiekis yra 26613, kai „1“ pasiskirstymas 30%, o mažiausias aptiktų gedimų kiekis yra 19120, kai „1“ pasiskirstymas 50%. Aptiktų gedimų kiekių skirtumai yra panašūs ir schemai b15 (didžiausias kiekis 16602, kai „1“ pasiskirstymas 70% ir mažiausias 10816, kai „1“ pasiskirstymas 30%). Remiantis šiais pastebėjimais siūlome tokį testų generavimo karkasą:

1. Pradžioje atlikti daug trumpų testų, su įvairiais „1“ pasiskirstymais.
2. Atrinkti geriausią „1“ pasiskirstymą, pagal didžiausią rastų gedimų kiekį.
3. Generuoti funkcinio vėlinimo testus su pasirinktu „1“ pasiskirstymu, iki kol prisisotins testų generacijos procesas.

Toliau, segmentų generavimui pritaikėme du pusiau deterministinius algoritmus. Juos vadiname pusiau deterministiniais, nes visi testiniai atvejai yra sugeneruoti atsitiktinai, tačiau iš tam tikro skaičiaus M galimų testinių atvejų kandidatų atrenkamas tas, kuris turi didžiausią svorį. Testinio atvejo svoris skaičiuojamas taip: $W=N_{det}*c1 + (N_{act} + N_{prop})*c2$, kur N_{det} yra naujų (dar neaptiktų prieš tai buvusiuose vidinėse sekose) funkcinių vėlinimo gedimų skaičius; N_{act} yra naujų vėlinimo gedimų, kurie yra aktyvuojami tolimesniam testavimui; N_{prop} yra naujų funkcinių gedimų, kurie toliau sklinda, skaičius; c1 ir c2 yra laisvai pasirenkami koeficientai.

Paveiksliuke 19 pristatome pirmąjį algoritmą, kur p_{pask} yra paskutinis į testinę seką įtrauktas rinkinys.

For i=1 to M do;
    Generuoti atsitiktinį testinį atvejį p_i
    P_{pask} ir p_i atlikti gedimų aktyvaciją ir nustatyti dydžius N_{det} ir N_{act};
    p_i atlikti gedimų sklidimo tikrinimą, ir nustatyti dydžius N_{prop} ir $N1_{det}$;
    $N_{det}= N_{det} + N1_{det}$
    Apskaičiuoti svorį W_i, p_i elementui ($W_i=N_{det}*c1 + (N_{act} + N_{prop})*c2$);
End for;
Į testinių atvejų seką įtraukti testinį atvejį p_i kuris turi didžiausią svorį W_i;
p_{last} ir išrinktam p_i elementams atlikti gedimų aktyvaciją;
Pridėti naujai rastus gedimus prie gedimų sąrašo;
Atnaujinti gedimų, kuriems reikia tikrinti sklidimą, sąrašą.

19 pav. Algoritmo 1 apibūdinimas

Antrasis algoritmas pateikiamas paveiksliuke 23.

For i=1 to M do;
    Generuoti atsitiktinį testinį atvejį p_i
    p_i atlikti gedimų sklidimo tikrinimą, ir nustatyti dydžius N_{prop} ir $N1_{det}$;

Apskaičiuoti svorį W_i, p_i elementui (W_i=$N1_{det}$*c1 + N_{prop}*c2);

End for;

Į testinių atvejų seką įtraukti testinį atvejį p_i kuris turi didžiausią svorį W_i;

p_{last} ir išrinktam p_i elementams atlikti gedimų aktyvaciją;

Pridėti naujai rastus gedimus prie gedimų sąrašo;

Atnaujinti gedimų, kuriems reikia tikrinti sklidimą, sąrašą.

20 pav. Algoritmo 2 apibūdinimas

Pagrindinis skirtumas tarp šių dviejų algoritmų yra tas, jog Algoritmas 1 testo sudarymo fazėje didesnį dėmesį skiria gedimų aktyvacijai, o Algoritmas 2 koncentruojasi ties gedimų sklidimo faze. Abiejų algoritmų taikymo funkcinių vėlinimo gedimų testų generavimui rezultatai yra pateikti 18 ir 19 lentelėse.

Lentelė 18. Algoritmo 1 taikymo rezultatai

Schema	Geriausi dydžiai ir lentelės 1			Algoritmas 1						
				M	c1=100; c2=1		c1=1; c2=100		c1=1; c2=1	
	VK sk.	Ilgis	%		VK sk.	Ilgis	VK sk.	Ilgis	VK sk.	Ilgis
b04	1797	3753	50%	10	1797	3788	1797	3584	1797	3790
b06	89	64	60%	7	89	81	89	75	89	83
b07	70	624	40%	3	69	538	69.5	605	69.5	569
b08	206	893	60%	10	206	825	206	1151	206	1164
b09	526	2283	70%	3	526	3240	526	3098	526	2946
b10	292	627	60%	10	292	616	292	647	291.5	635
b11	957	21742	40%	10	957	38153	957	47316	957	41195
b12	637	32636	5%	10	608	28957	601	25018	605	27518
b13	353	11350	40%	10	353	12561	353	15396	353	12554
Vidurkis	547	8219	-	-	544	9862	543	10766	544	10050
b14	33224	450066	20%	30	28900	150951	32909	428715	27944	119628
b15	16602	23102	70%	30	15917	22502	15949	22566	15825	22951
Vidurkis b14, b15	24913	236584	-	-	22408.5	86726.5	24429	225640.5	21884.5	71289.5
Bendras vidurkis	4978	49740	-	-	4519	23837	4886	49834	4424	21185

Lentelė 19. Algoritmo 2 taikymo rezultatai

Schema	Geriausi dydžiai ir lentelės 1			Algoritmas 2						
				M	c1=100; c2=1		c1=1; c2=100		c1=1; c2=1	
	VK sk.	Ilgis	%		VK sk.	Ilgis	VK sk.	Ilgis	VK sk.	Ilgis
b04	1797	3753	50%	10	1797	3807	1797	3715	1797	3645
b06	89	64	60%	7	89	77	89	60	89	76
b07	70	624	40%	3	70	742	69	693	69	490
b08	206	893	60%	10	206	1179	206	1156	206	1407
b09	526	2283	70%	3	526	3020	526	2973	526	2863
b10	292	627	60%	10	292	658	291.5	667	291.5	630
b11	957	21742	40%	10	957	40154	957	42777	957	36478
b12	637	32636	5%	10	641	36452	643	35917	641	34949
b13	353	11350	40%	10	353	13215	353	15396	353	16929
Vidurkis	547	8219	-	-	548	11034	548	11484	548	10830
b14	33224	450066	20%	30	32525	385010	32909	428715	32763	413520
b15	16602	23102	70%	30	15248	23317	15761	23463	16507	21902

Vidurkis b14, b15	24913	236584	-	-	23886.5	204163.5	24335	226089	24635	217711
Bendras vidurkis	4978	49740	-	-	4791	46148	4873	50503	4927	48444

Palyginimui taip pat pateikiame geriausius rezultatus iš 17 lentelės (2-4 stulpeliai), kur naudotas „1“ pasiskirstymo procentas pateiktas 4 stulpelyje. Kiekvienu atveju buvo atlikti du atskiri testų generavimai. Abu algoritmai yra taikomi naudojant stulpelyje 4 pateiktą „1“ pasiskirstymą. Testų generavimas atitinkamoms schemoms užtruko tokį pat laiką, kaip ir testai, pateikti 17 lentelėje. Geriausius rezultatus davę testai yra pažymėti.

Taikėme Algoritmus 1 ir 2 naudodami įvairius koeficientų c1 ir c2 dydžius, bandydami koncentruotis ties naujų gedimų radimu (c1 = 100) arba teikdami pirmenybę gedimų sklidimo tikrinimui (c2 = 100); trečiuoju atveju abu koeficientai buvo lygūs. Buvo tikimasi, jog šių algoritmų taikymas schemose, kurių testų generavimas sunkiai prisisotina, duos aukštesnius ar netgi žymiai aukštesnius gedimų padengimus; schemose, kurių testų generavimas prisisotina greit, buvo tikimasi mažesnių testų ilgių. Tačiau lūkesčiai nepasiteisino. Palyginę pusiau deterministinių testų generavimo rezultatus su visiškai atsitiktiniu generavimu, matome, jog atsitiktinio testų generavimo atveju 6 schemoms buvo pasiekti geriausi rezultatai (iš viso buvo tiriama 11 schemų). Rezultatų vidurkiai taip pat rodo atsitiktinio generavimo pranašumą. Buvo galima galvoti, jog galbūt M dydžiai buvo per maži, tačiau gauti schemų b07, b09 (abi turi po vieną pirminį įėjimą) ir b06 (turi 2 pirminius įėjimus), kurioms buvo naudojamas sąlyginai didelis galimų kandidatų sąrašas, rezultatai, panaikina šią prielaidą. Galime daryti išvadą, jog generuojant testinius segmentus, atskirų žingsnių optimizacija nepagerina galutinio rezultato.

3.4. **Tyrimo išvados**

1. Integracijos tipas labiausiai įtakoja tik darbą su nedidelėmis schemomis.
2. Didėjant schemos failams ir darbo apimčiai, integracijos tipas vis mažiau įtakoja bendrą darbo trukmę ir menkai padidintų sistemos darbo efektyvumą naudojant tiesiogiai integruotus *.h failus.
3. Tyrimai rodo, jog funkciniai testai, gaunami atsitiktiniu testų generavimo metodu duoda gerus gedimų padengimo rezultatus.
4. Remiantis procentiniu „1“ pasiskirstymu atsitiktinai generuojamiems testiniams atvejams sudarytas testų generavimo karkasas.
5. Pusiau deterministinių algoritmų pritaikymas parodė, jog atskirų žingsnių optimizacija sudarant testinius segmentus nepagerina galutinio rezultato.

IŠVADOS

1. Projektuojant schemų testus didelės apimties schemoms reikalingi labai dideli kompiuterinės įrangos resursai, todėl pilnas testavimas neįmanomas. Skaitmeninių įrenginių testavimas – problematiškas procesas. Vystantis technologijoms, didėjant schemų apimčiai, tradiciniai testų sudarymo metodai tampa vis mažiau efektyvūs.
2. Remiantis funkciniais reikalavimais sukurta sistemos architektūra, pagal ją realizuota testų generavimo sinchroninėms nuosekliosioms schemoms programinė įranga.
3. Sukurta sistema padėjo ištirti atsitiktinio testinių atvejų generavimo patobulinimo galimybes ir procentinio „1“ pasiskirstymo įtaką sugeneruotų testų pilnumui ir efektyvumui.
4. Remiantis procentinio „1“ pasiskirstymo atsitiktinai generuojamiems testiniams atvejams tyrimais buvo pasiūlyta efektyvaus testų generavimo metodika.
5. Pusiau deterministinių algoritmų pritaikymas parodė, jog atskirų žingsnių optimizacija sudarant segmentą nepagerina galutinio rezultato.
6. Didėjant schemos failams ir darbo apimčiai, integracijos tipas vis mažiau įtakoja bendrą darbo trukmę ir menkai padidintų sistemos darbo efektyvumą naudojant tiesiogiai integruotus *.h failus.

LITERATŪRA

[1] M. Abramovici, M. A. Breuer, A. D. Friedman, Digital Systems Testing and Testable Design, IEEE Press, 1990

[2] R. Garcia, Rethink fault models for submicron-IC test, pp.35, 2001

[3] M. J.S. Smith, Application-Specific Integrated Circuits, Addison-Wesley, 2000

[4] K. Zarrineh, S.J. Upadhyaya, V. Chickermane, System-on-chip testability using LSSD scan structures, pp.83 - 97, 2001

[5] R. David, Random Testing of Digital Circuits: Theory and Applications, Marcel Dekker, Inc., 1998

[6] V. Jusas, K. Paulikas, R. Seinauskas, Procedures for Selection of Validation Vectors on the Algorithm Level, pp.90-95, 2001

[7] M. Michael, S. Tragoudas, ATPG Tools for Delay Faults at the Functional Level, pp.33-57, 2002

[8] Pomeranz, S. M. Reddy., Functional Test Generation for Delay Faults in Combinational Circuits, pp.687-694, 1995

[9] B. Underwood,W. O. Law, S. Kang, H. Konuk, Fastpath: A Path-Delay Test Generator for Standard Scan Designs, pp.154-163, 1994

[10] I. Pomeranz and S.M. Reddy, On Testing Delay Faults in Macro-based Combinational Circuits, pp.332–339, 1994

[11] D. Van Tassel, Program Style, Design, Efficiency, Debugging, and Testing, Prentice Hall, 1974

[12] [Synopsys, Online Documentation, 2003.06

[13] A. Hartman, Software and Hardware Testing UsingCombinatorial Covering Suites, pp., 2003

[14] D. T. Tang, C. L. Chen, Iterative exhaustive pattern generation for logictesting, pp.212-219, 1984

[15] Digital Mars kompiliatorius, [žiūrėta 2010 balandžio 30], prieiga internete: http://www.digitalmars.com/

[16] Pomeranz, S.M. Reddy. Primary input vectors to avoid in random test sequences for synchronous sequential circuits. *IEEE transactions on computer-aided design of integrated circuits and systems*, 27(1), 2008, 193-197.

[17] E. Bareiša, V. Jusas, K. Motiejūnas, R. Šeinauskas. The non-scan delay test enrichment based on random generated long test sequences. *Information Technology and Control*, 39(4), 2010, 251-256.

[18] E. Bareiša, V. Jusas, L. Motiejūnas, R. Šeinauskas. Generating Functional Delay Fault Tests for Non-Scan Sequential Circuits. *Information Technology and Control*, 39(2), 2010, 100-107.

[19] E. Bareiša, V. Jusas, K. Motiejūnas, R. Šeinauskas. On delay test generation for non-scan sequential circuits at functional level. *Elektronika ir elektrotechnika – Electronics and electrical engineering*, 3(109), 2011, 67-70.

[20] P. Pant, A. Chatterjee. Path-delay fault diagnosis in non-scan sequential circuits with at-speed test application. *Proceedings of the International Test Conference ITC 2000*, 245-252.

[21] Pomeranz, S.M. Reddy. TOV: sequential test generation by ordering of test vectors. *IEEE transactions on computer-aided design of integrated circuits and systems*, 29(3), 2010, 454-465.

[22] J. Kang, S. C. Seth, V. Gangaram. Efficient RTL Coverage Metric for Functional Test Selection. *Proceedings of the 25th IEEE VLSI Test Symposium*, 2007, 318-324.

[23] K. Morkūnas, R. Šeinauskas. Circuit Reset Sequences based on Software Prototypes. *Elektronika ir elektrotechnika – Electronics and electrical engineering*, 7(103), 2010, 71-76.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

CAD (*Computer – Aided Design*) Kompiuterinis projektavimas

BSDL - Perimetro skenavimo aprašymo kalba

CPU (central processing unit) – kompiuterio procesorius

FPGA (field-programmable gate array) – integruota schema, kurią galima konfigūruoti po pagaminimo

Quad Core – procesoriaus sistema susidedanti iš 4 atskirų branduolių.

Intel Core i7 – sekančios kartos procesorius, technologija pranokstantis Quad Core

Microsoft Windows – kompanijos Microsoft kuriamų operacinių sistemų serija.

SSF (*single stuck-at fault)* - fiksuotos reikšmės gedimų modelis

RTL (Register Transfer Level) – registrų transformavimo lygis

ATPG (*Automatic Test Pattern Generator)* – automatinis testinių rinkinių generatorius

DFT (*design for test) – kūrimas testavimui (schemų)*

LSSD (level-sensitive scan design) - lygiui jautrus skenavimas

BIST - (*Built-in Self-test*) – integruotas savitestavimas

LFSR (*linear feedback shift register*) - registro pagalba generuojama pseudoatsitiktinė signalų seka, kuri paduodama į testuojamą logiką.

SISR (*serial input signature register*) – įėjimų parašų registras

MISR (*multiple-input signature register*) – kelių įėjimų parašų registras

BILBO (*built-in logic block observer*) – integruoto loginio bloko jutiklis

CSTP (*circular self-test path*) – ciklinis savitesto kelias

DFT (*design for testability*) – lengvai testuojamas projektavimas

RTL (*Register transfer leve*l) Registrų perdavimo lygis

Kompiliatorius – programinė įranga, kuri sugeba tam tikra programavimo kalba parašytą programą transliuoti į ekvivalenčią programą kita kalba

.NET Framework - programinis karkasas, kuris veikia windows aplinkoje. Jis turi didelę biblioteką ir palaiko kelias programavimo kalbas.

Digital Mars – sistemos darbui naudojamas kompiliatorius

UML – unifikuota modeliavimo kalba

PA – panaudojimo atvejis(-ai)