KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Vytautas Jonutis

Mindaugas Jaraminas

Vaizdų atpažinimo sistemos projektavimas ir tyrimas.

MAGISTRO BAIGIAMASIS DARBAS

Darbo vadovas: prof. E. Bareiša

Kaunas 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Vaizdų atpažinimo sistemos projektavimas ir tyrimas.

MAGISTRO BAIGIAMASIS DARBAS

Recenzentas:
doc. dr. A. Riškus
2008 m. gegužė 22 d.

Vadovas:
prof. E. Bareiša
2008 m. gegužė 22 d.

Atliko:
IFM-2/5 grupės studentai
Vytautas Jonutis
Mindaugas Jaraminas
2008 m. gegužė 22 d.

Kaunas 2008

Turinys

Summary

In this work we analyzing video preprocessing system model. Primary model specifications are described in functional level. It is hard to decide what system architecture should be, so we used *SystemC TLM* modeling language, because it gives us easier way to change system architecture

Using *SystemC* transaction level modeling (*TLM*) the functional primary specification are transformed from functional model to system level. To get synthesizable model we use primary specification and *TLM* model. We solve many system architecture problems while we where working on primary model transformation to high abstraction system. Transforming high abstraction level model to *SystemC* synthesizable code we solve variables selection problems and algorithms conversation problem.

1 Įvadas

1.1 Dokumento paskirtis

Šis dokumentas yra vienlusčių sistemų baigiamasis darbas. Tyrimo objektas yra vaizdo apdorojimo sistema, kurioje naudojamas trimatis skaneris.
Darbe yra analizuojami skirtingo abstrakcijos lygio modeliai ir nagrinėjamas modelio transformavimo kelias į sisteminį lygį.
Aprašoma analizė TLM ir RTL modeliais.

- ✓ Abu esame atsakingi už pradinio funkcinio modelio parinkimą, algoritmų analizę.
- ✓ Mindaugas yra atsakingas už modelio aprašymą aukštame abstrakcijos lygyje, bei sistemos architektūros parinkimą.
- ✓ Vytautas analizuoja aukšto abstrakcijos lygio modelio transformavimą į sintezuojamą modelį.
- ✓ Rezultatuose pateikiami gauti modeliavimo ir sintezės rezultatai.

1.2 Santrauka

Darbe analizuojamas vaizdo apdorojimo sistemos modelis, kuris yra modeliuojamas. Pradinė modelio specifikacija yra aprašoma funkciniame lygyje. Modelio architektūrai modeliuoti mes naudojame transakcijų lygio SystemC, naudodamiesi ja mes galime greitai ir patogiai nustatyti, kokia turėtų būti modeliuojamos sistemos architektūra. Funkcinis modelis yra transformuojamas į sisteminį lygį naudojantis SystemC transakcijų modeliavimo kalba. Naudojantis pradine specifikacija ir TLM modeliu pereiname prie sintezuojamo aprašo. Transformuodami pradinį modelį aukštame abstrakcijos lygyje, mes sprendžiame sistemos architektūros problemą. Transformuodami aukšto lygio modelį į SystemC sintezuojamą aprašą, mes sprendžiame kintamųjų ir algoritmų transformavimo problemas.

2 Analitinė dalis

2.1 Vaizdų apdorojimo sistemos analitinė dalis

Tyrimo objektas yra vaizdų apdorojančios sistemos *SystemC* modelio analizė. Pagrindinės problemos kyla sprendžiant pradinės specifikacijos transformavimą į *SystemC* modelius. Problemų sprendimui naudojame transakcijų lygio *SystemC*.

Pradinis analizuojamas modelis yra parašytas naudojantis C++ programavimo kalba. Šis C++ modelis naudoja vaizdų apdorojimo specifinius algoritmus, kuriuos sugalvojome naudodamiesi literatūra apie realių objektų perkėlimą į trimatę erdvę.

Vaizdų apdorojimo algoritmai naudojami įvairioms problemoms spręsti, kaip:

- Vaizdų pagerinimas medicinoje.
- Skaitmeninių nuotraukų apdorojimui.
- Kuriant protingas sistemas (robotai, veido atpažinimas, objektų atpažinimo sistemos)

2.1.1 Trimačių skenerių apžvalga

Trumpai skirstant trimačius skenerius galimas jų rūšiavimas pagal trimačių sistemų apžvalgą ir pagal jų pritaikymą. Pasaulyje pirmosios bangos šio proceso prasidėjo apytiksliai prieš 20 metų, o panašiai prieš 10 metų sritis plačiai ištobulėjo ir buvo daugelyje gyvenimo atvejų pritaikytos.

Nagrinėjant trimačius skenerius pagal sistemų apžvalgą reiktų paminėti, kad trimačių objektų kompiuteriniai modeliai vis plačiau taikomi įvairiausiose gyvenimo ir pramonės srityse. Tai: multiplikacija, archeologija, architektūra, dantų protezavimas, mokymas, rūbų ir aprangos kūrimas, avalynės gamyba, kriminologija, žaidimai, pramoninis dizainas, gamyba, medicina, kino pramonė, muziejai, esamų objektų planų ir projektų paruošimas, greitas prototipų gaminimas, atvirkštinis konstravimas, skulptūrų kūrimas, žaislų pramonė, formų gamyba, medienos pramonė, internetinių puslapių kūrimas ir kiti.

Šiuo metu daugiau ar mažiau naudojami šie trimačių objektų skanavimo metodai ir įranga:

- Lazeriniai skeneriai (Laser Scanners);
- Skeneriai su rastriniais šviesos šaltiniais (Structured Light Scanners);

- Lazeriniai radariniai skeneriai (Laser Radar Scanners);
- Lazerinio tolimačio skeneriai (Time of Flight Laser Scanners);
- Holografiniai skeneriai (Holography);
- Profilio skeneriai (Profilers)
- Stereoskopiniai skeneriai (Dense Stereo Matching)
- Fotogrammetriniai skeneriai (Close-Range Photogrammetry)

Nagrinėjant trimačius skenerius pagal jų pritaikymą galime apibrėžti galimas sistemos pritaikymo sritis:

1. Vizualinės kokybės kontrolės automatizavimas pramonėje.
2. Robotų valdymas pagal trimačius vaizdus.
3. Detalių rūšiavimo įrenginiai pramonėje.
4. Medienos pramonėje rąstų išpjovimo nustatymo sistemose.
5. Vaisių ir daržovių rūšiavimo įrenginiai.
6. Gėlių rūšiavimo automatai.
7. Automobilių kėbulų geometrijos matavimo įranga.
8. Trimačių vaizdų naudojimas ortopedijos pramonėje.
9. Trimačių vaizdų naudojimas avalynės gamybos pramonėje.
10. Vaikų dantų lanko parametrų matavimas pagal gipsines burnos ertmės kopijas.
11. Trąšų granulių ar kitų birių produktų dydžių ir svorių statistinė analizė gamybos procese.
12. Veidų identifikavimo ir klasifikavimo įranga ir sistemos.

Šioje ataskaitoje bandysime apžvelgti, kas yra pasiekta kiekvienoje iš šių sričių ir kokios tolimesnės trimačių objektų identifikavimo sistemos, ar jos gamybos metu sukurtų programinės ar aparatūrinės įrangos pritaikymo galimybes. Taip pat apžvelgsime keletą veikiančių produktų ir juos šiek tiek panagrinėsime.

2.1.2 3D skanerių sistemų apžvalga

Lazeriniai skeneriai

Lazeriniai skeneriai veikia trianguliacijos principu. Lazerio spindulys (atskiras taškas, linija ar keletas linijų) projektuojamas ant skanuojamo objekto. Video kamera, esanti šone nuo lazerio spindulio šaltinio mato lazerio spindulio vaizdą ant tiriamo objekto. Spindulio taškai ant objekto, esančio toliau, matomi kameroje paslinkti atžvilgiu taškų, esančių ant arčiau esančių objektų.

Pagrindiniai lazerinio skanavimo privalumai yra tai, kad šis metodas pakankamai greitas, nekontaktinis ir tikslus. Labai trapūs objektai gali būti skanuojami nesijaudinant dėl jų pažeidimo. Lazeriniai skeneriai pakankamai greitai išmatuoja didelius taškų skaičius.

Šiuo metodu gali būti matuojamos tik tos objekto dalys, kurias kerta lazerio spindulys, todėl norint gauti viso objekto modelį reikia skanuoti iš kelių pozicijų. Kartais lazeriniai skeneriai naudojami kartu su sukamaisiais stalais, kas leidžia tiksliai slinkti objektą vietoje perstatomo skenerio. Rankiniai lazeriniai skeneriai laisvai judinami erdvėje rankomis, tačiau šiuo atveju jų vietai nustatyti naudojama kitu principu veikianti padėties nustatymo sistema (pav. indukcinė).

Didžiausios problemos kyla lazeriniams skeneriams susidūrus su labai spindžiais, šviečiančiais ar skaidriais paviršiais. Jie taip pat nekaip veikia ir skanuojant labai tamsius objektus. Šiuo atveju gali tekti tiriamą objektą nudažyti šviesiai arba padengti jį baltais milteliais, aprašoma [15].

Skeneriai su rastriniais šviesos šaltiniais

Skeneriai su rastriniais šviesos šaltiniais projektuoja ant tiriamo objekto žinomos struktūros ir piešinio šviesos rastrą. Video kamera, esanti šone nuo šviesos šaltinio mato šviesos rastro vaizdą, kuris iškraipomas priklausomai nuo tiriamo objekto paviršiaus reljefo. Čia gali būti naudojami patys įvairiausi šviesos rastrai: taškiniai, linijiniai, tinkleliniai, apskritiminiai, sinusoidiniai ir kiti, tiek juodai balti tiek ir spalvoti. Toliau objekto taškų erdvinėms koordinatėms išskaičiuoti naudojama trianguliacija.

Skeneriai su rastriniais šviesos šaltiniais iškart nuskanuoja visą vaizdo kameros matomą objekto paviršių. Norint gauti visą trimatį objekto modelį gali tekti pakartoti

skanavimą iš kelių erdvės taškų. Šiuo atveju kaip šviesos šaltinis naudojama halogeninė arba analogiška baltos šviesos lempa, todėl čia nekyla tų pavojingumo problemų, kurios žinomos naudojant lazerius.

Lazeriniai radariniai skeneriai

Lazerinio radarinio skenerio pagrindas - tai plačiajuostis dažniu moduliuojamas infraraudonųjų spektre dirbantis lazeris (100GHz moduliacija), kuris skleidžia stiprų, bet akims nepavojingą signalą. Duomenys greičio ir atstumo matavimui gaunami lyginant gaunamo signalo kylančio ir krentančio dažnio kitimo reikšmių sulyginimo metu. Signalo apdorojimo metu išskiriami interferenciniai dažniai yra tiesiogiai proporcingi atstumui iki objekto.

Šio metodo privalumas didelis vieno taško atstumo nustatymo tikslumas, tačiau erdvinių koordinačių nustatymo tikslumas remiasi mechaninės skanavimo sistemos tikslumu. Taip pat nuo mechanikos galimybių priklauso ir skanavimo greitis.

Lazerinio tolimačio skeneriai

Lazeriniai tolimačio principu veikiantys skeneriai tiesiog siunčia į objektą šviesos impulsą ir matuoja laiką , po kurio šis impulsas grįžta. Žinodami, kad šviesos greitis yra pastovus ir žinomas, galime suskaičiuoti atstumą iki objekto matuojamo taško. Skenerio varikliai kraipo spindulį per visą skanavimo zoną. Trimatės objekto paviršiaus taškų koordinatės išskaičiuojamos kaip lazerio galvutės horizontalių ir vertikalių padėčių kampų ir išmatuotų atstumų visuma.

Tokių skenerių tikslumas ir skiriamoji geba gana ribota, nes tai apsprendžia mechanika ir dažniausiai neviršija +/- 6 mm. Tačiau šių skenerių privalumas dideli matuojamo objekto gabaritai ir atstumai iki jų.

Holografiniai skeneriai

Naudoja lazerius ir holografinį principą, kad nustatyti atstumą ir paviršiaus reljefiškumą. Yra keli metodai. Dažniausiai naudojami paviršių kontrolei, bet gali būti ir trimačių objektų skeneriai.

Vienas iš tokių naudoja lazerius su valdomu bangos ilgiu ir fazės poslinkiu. Gali matuoti 12x12x16" darbinėje zonoje. Matavimas užima 2-3 minutes. Gerai veikia esant poliruotiems paviršiams. Tai aukštų technologijų įrenginiai ir pakankamai brangūs.

Profilio skeneriai

Profilio skeneriai naudoja technologiją panašią į naudojamą kine arba televizijoje, kai aktorius filmuojamas prie vienspalvio ekrano (dažniausiai žalio arba mėlyno) ir tuo būdu po to lengvai “iškerpamas fonas“ ir įdedama reikiama scena.

Profilio skeneriai dažniausiai turi kompiuterio valdomą sukamąjį stalą, pastatytą prieš vienspalvį ekraną. Skanuojamas objektas padedamas ant sukamojo stalo. Kiekvienoje posūkio padėtyje video kamera nuskaito objekto vaizdą. Skeneris išskaičiuoja objekto profilį, išskirdamas perėjimo vietą tarp fono spalvos ir bet kurios kitos. Turėdami profilio linijos koordinates ir žinodami stalo posūkio padėtis galime suskaičiuoti ir viso paviršiaus koordinates. Panaudojus kadrų nefonines spalvas, galima sukurti ir objekto spalvotas tekstūras.

Šie skeneriai yra pakankamai paprasti ir pigūs, tačiau turi vieną esminį trūkumą. Jie visai negali atpažinti objekto įdubusių vietų.

Stereoskopiniai skeneriai

Stereoskopiniai skeneriai naudoja stereoskopinio matymo principą. Techninių sistemų atveju naudojami du vaizdai, kurie vienas kito atžvilgiu skirtingais kampais mato tiriamą objektą.

Trimatei informacijai gauti naudojami specialūs vaizdų atpažinimo algoritmai, todėl šis metodas gana ribotai taikomas praktikoje, o daugiau yra mokslinių ir eksperimentinių tyrinėjimų stadijoje. Šiuo metu šis metodas kartais naudojamas veidų identifikavimo sistemose, nenaudojant pilno trimačio modelio sukūrimo.

Panašiais principais veikia ir fotogrametrinis skeneris, tik čia naudojama daugiau vaizdų įvairiais rakursais.

Fotogrametriniai skeneriai

Fotogrametrinė technika leidžia jums iš dvimačių objekto nuotraukų gauti trimatį objekto modelį. Naudojant žinomų charakteristikų (objektyvo fokuso atstumas, matricos didis ir pikselių skaičius) skaitmeninę kamerą, jums reikės mažiausiai dviejų objekto nuotraukų. Jei jūs galėsite identifikuoti tuos pačius tris taškus ant vaizdų ir žinosite, kurios nors objekto dalies tikslius išmatavimus bus galima suskaičiuoti ir kitų objekto dalių erdvines koordinates.

Šis metodas labai tinka, kai yra ribotas laikas modeliuojamo objekto tyrinėjimams. Vaizdus galima gauti labai greitai. Mokestis už tai ilgas vaizdų apdorojimas ir neišvengiamas rankinis darbas. Jei kuriamo modelio tikslumui gauti reikia pakankamai daug erdvinių taškų darbas bus ilgas ir varginantis.

Šio metodo tikslumą ir gauto modelio kokybę apsprendžia visa eilė veiksnių: tai ir kameros objektyvo kokybė ir video matricos skiriamoji geba, kameros kalibravimo tikslumas ir naudojamų vaizdų skaičius.

Nenagrinėtos tokios įrangos rūšys: kaip akustiniai (Acoustic Position Trackers), lazeriniai (Laser Trackers), magnetiniai (Magnetic Position Trackers) ar optiniai (Optical Position Trackers) padėties lokatoriai ir mechaniniai trimačių objektų skeneriai kaip koordinatinės matavimo mašinos (Coordinate Measurement Machines) ir kontaktiniai zondai (Touch Probes).

Padėties lokatoriai skirti ne trimačio objekto skanavimui ir jo modelio sudarymui, o objekto judėjimo trimatėje erdvėje sekimui.

Mechaniniai trimačių objektų skeneriai pagal savo veikimą ir charakteristikas visai netinka mūsų projekte keliamiems tikslams realizuoti.

2.1.3 3D skanerių pritaikymo sritys

Vizualinės kokybės kontrolės automatizavimas pramonėje

Pramonėje gaminami produktai dažniausiai yra trimačiai. Taikant dvimates kompiuterinės regos sistemas kokybės kontrolei, reikia tiksliai pozicijai nustatyti kontroliuojamus objektus, nes tik tokiu būdu galima įvertinti atitinkamus geometrinius parametrus. Trimačių vaizdų identifikavimo sistema įgalina tiksliai nustatyti objektų geometrinius parametrus, kai tiriamų objektų koordinatės nėra griežtai ir tiksliai žinomos.

Šio projekto vykdymo metu, buvo atlikti keli bandymai panaudoti sistemą elektrinių jungčių kokybės kontrolei jų gamybos ir montavimo metu. Bandymai parodė tokios kontrolės galimybę, tačiau tikslesniems rezultatams gauti reikia pagaminti šiek tiek kitokią, mažiems objektams skenuoti skirtą aparatūrą.

2.1.4 Robotų valdymas pagal trimačius vaizdus

Pramoniniai robotai valdomi pagal griežtai nustatomas koordinates. Trimačių vaizdų identifikavimo sistema galėtų suteiktų galimybę robotui orientuotis kintančioje realioje erdvėje. Tai suteiktų galimybes daug plačiau taikyti robotus pramonėje.

Šiame etape buvo atlikta analizė ir patyrinėta galimybė pritaikyti kuriamą trimačių objektų identifikavimo sistemą, robotų ar intelektualių vežimėlių orientacijai erdvėje. Kaip parodė analizė, šioje srityje šiuo metu perspektyvesnė yra ultragarsinė arba lazerinė lokacija.

Detalių rūšiavimo įrenginiai pramonėje

Detalių rūšiavimo įrenginiai daugiausia naudoja dvimates kompiuterinės regos sistemas. Trimačių vaizdų identifikavimo sistema suteikia galimybę paspartinti rūšiavimo procesą ir palengvinti rūšiavimo įrenginio mechaninę dalį, nes nereikia tiksliai žinoti detalių pozicijos.

Šioje srityje kaip tik ir buvo pirmas konkretus trimatės identifikavimo sistemos taikymas. Panaudojant įrangą, algoritmus ir programas sukurtas projekto vykdymo metu, buvo atlikti eksperimentai su saldainių identifikavimu, kas leistų imtis projektuoti ir gaminti saldainių pakavimo liniją su joje instaliuota saldainių gabaritų ir defektų identifikavimo įranga. Ši įranga veikia panaudojant kombinuotą trimatės ir dvimatės kompiuterinės regos sistemą.

Šis įrenginys brokuoja netinkamų gabaritų ir pažeistus saldainius slenkančius konvejeriu. Tarkim saldainių srautas 1800 saldainių per minutę. Tokiu būdu jie visi skaidomi į 10 eilių ir kiekvienas 5 eiles analizuoja atskiros sistemos su dviem lazeriniais skeneriais ir dar dviem kampu žiūrinčiom vaizdo kamerom, kurios identifikuoja saldainių šoninių sienelių defektus.

Specialiai šiai sistemai adaptuota programinė įranga išanalizuotu vaizdus, atstatytu modelį ir skersinius pjūvius, bei priimtų sprendimą, apie saldainio tinkamumą per maždaug 20ms laiko intervalą.

Pagal šios sistemos komandas netinkami saldainiai nupučiami į broko konvejerį.

Tokios sistemos ar jų grupės gali būti taikomos tiek pramonėje, kuri gamina produktus panašius į saldainius, tiek ir kitokius objektus. Šios sistemos specifika yra didelis greitis, todėl naudojamas supaprastintas trimatis identifikavimas. Lėčiau konvejeriu judantiems objektams ši sistema dar perspektyvesnė, nes galime žymiai tiksliau skenuoti tiriamus objektus ir identifikuoti sudėtingesnius defektus.

Vaisių ir daržovių rūšiavimo įrenginiai

Šie įrenginiai tai tam tikras atskiras "detalių rūšiavimo įrangos" variantas. Šie objektai turi tam tikrą specifiką: dažniausiai juos būtina skenuoti iš visų pusių ir būtina teisingo jų identifikavimo sąlyga yra spalvinės informacijos turėjimas. Tai susiję su jų defektų buvimu bet kurioje vietoje ir dažnu atveju, kai ne tik geometrijos ar reljefo pokyčiai, bet ir spalviniai pokyčiai apsprendžia tiriamo objekto kokybę.

Trimačių vaizdų naudojimas ortopedijos pramonėje

Žmogaus pažeistų galūnių trimatis skenavimas reikalingas gaminant ortopedinius gaminius. Preliminariai buvo aptartas tokios sistemos poreikis su specialistais dirbančiais protezų gamybos srityje. Tikslus galūnės, prie kurios tvirtinamas protezas kompiuterinis modelis ypač reikalingas, kai protezas gaminamas toli nuo jo vartotojo gyvenamos vietos ir naudojant modernias kompiuterių valdomas protezo gamybos technologijas. Vis labiau plintant ir pingant šioms technologijoms, skenavimo poreikis įrengimams taip pat augs.

Trimačių vaizdų naudojimas avalynės gamybos pramonėje

Avalynės, o ypač ortopedinės, gamintojai buvo vieni iš pirmųjų, kurie domėjosi trimačių kojos modelių atkūrimo galimybe. Domintis pramones poreikiais, paaiškėjo, kad juos labiausiai domina trimačių kojos modelių panaudojimas automatizuotai avalynės gamybos projektavimui ir paruošimui. Tai labai specifinė šaka reikalaujanti didelės patirties šioje srityje ir net specialistams sunkiai suprantama.

Toliau domintis šia sritimi, paaiškėjo, kad be ortopedinės avalynės, pasaulyje plinta prekybos avalyne variantas, kai batas parenkamas tiksliai klientui pagal kompiuterinį jo kojos modelį. Šiam metodui išplatinti reikia daug ir pigių kojų skenavimo įrenginių. Todėl veikla tokio įrenginio kūrimo kryptimi yra visai galima ir puikiai leistų panaudoti žinias.

Vaikų dantų lanko parametrų matavimas

Šie matavimai atliekami naudojant burnos ertmės gipsines kopijas ir skirti įvertinti vaikų dantų ir žandikaulio vystymąsi bei diagnozuoti vystymosi anomalijas.

Trąšų granulių ar kitų birių produktų dydžių ir svorių statistinė analizė gamybos procese

Granuliuotų trąšų ar kitų birių produktų gamybos procese labai svarbu pastoviai kontroliuoti dalelių dydžių procentinę sudėtį sraute. Nukrypimai nuo reikiamo dydžio dažniausiai reiškia technologinius nukrypimus ir reikalauja įsikišimo. Šiuo metu šie parametrai dažniausiai kontroliuojami rankiniu būdu sijojant medžiagas per skirtingo tankumo sietus ir po to sveriant.

Tačiau norint suskaičiuoti medžiagos svorio pasiskirstymą, jau reikia pamatuoti erdvinį dalelių dydį.

Veidų identifikavimo ir klasifikavimo įranga ir sistemos

Paskutiniu metu apsaugos sistemose ir identifikavimo ir riboto darbuotojų įleidimo sistemose pradėtas plačiai taikyti žmonių identifikavimas pagal trimačius veido modelius.

Šios sistemos yra kelių sudėtingumo lygių, pradedant nuo dvimačių kompiuterinės regos sistemų iki su sudėtingais kūno skeneriais sujungtų sistemų. Atlikta analizė parodė, kad nesudėtingas, bet patikimas veido skeneris žmonių identifikavimui turėtų paklausą.

Šiai kompiuterinės regos sistemų grupei smarkiai šiuo metu vystantis labai tikėtina, kad atsiras ir daugiau trimačių objektų identifikavimo sistemų pritaikymo sričių, kur sėkmingai bus panaudotos įgytos žinios.

2.1.5 Pramoninių 3D sistemų analizė

Vienas iš didžiausių 3D skanerių gamintojų Amerikoje yra „Laser Design Inc“, kuri yra viena iš pirmaujančių bendrovių pasaulyje. „Laser Design Inc“ gamina lazerinius 3D skanerius jau nuo 1987 m. Gaminami bendros paskirties 3D skaneriai yra montuojami ant aukšto tikslumo CMM (Koordinačių matavimo mašina), kurie suteikia sistemai didelį tikslumą. Įmonė taip pat gamina specifiniams tikslams pritaikytus skanerius, kaip batų arba protezų skanavimas, odontologijos, klausimosi apartų, juvelyrinių dirbinių, geležinkelių bėgių, mobiliųjų telefonų, ir t.t. aprašoma [14].

Įmonė ne tik gamina įrenginius, bet ir teikia 3D skanavimo paslaugas.

Šiuo metu įmonėje gaminami ir pardavinėjami šie bendros paskirties skaneriai.

Pav. nr. 1 *Greitas ir tikslaus 3D skaneris, naudojantis CMM. Skaneris yra patogus* naudojimui siūlo vartotojams išskirtines galimybes skenuojant sudėtingos formos objektus. Naudodami įmonės patentuotas technologijas įrenginys skanuoja objektą iš visų orientacijų, atlikus darbus įrenginys konvertuoja koordinates į vieningą koordinačių sistemą. Operatoriai gali greitai, bei patogiai konvertuoti įvairaus dydžio bei sudėtingumo objektus, ypač tokius su sudėtinga geometrija.

Pav. nr. :2

Tai 3d skaneris su integruota septynių ašių ranka Ultra prieinama, lengvai naudojama su integruota ranka yra panaudotina visų tipų detalių skanavimui. Be to įrenginiai yra pateikiami su programine įranga, kurios pagalba nuskanuoto objekto duomenis galima perkelti į kitą programinę įrangą.

Pateikiama ir specifiniams poreikiams naudojama aparatūrinė bei programinė įranga pvz. batų skanavimo sistema.

Desktop 3D scanner for automatic scanning of shoe soles and other medium-sized parts.

Pav. nr. 3

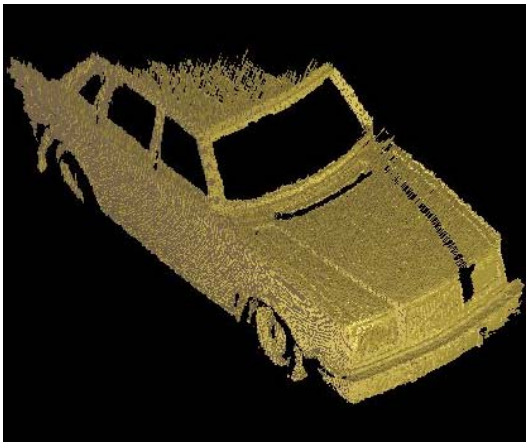
Pav. nr. 4

Išskirtinai sukurta batus kuriančiam institutui, atlikti automatinio skanavimo funkcija. Ši produkto linija yra kompaktiškai pastatoma ant stalo. Sistema yra sukurta tiksliam automatinam skanavimui vidutinio dydžio produktams, tokiems kaip batai - puspadžiai.

Sistema yra paremta besisukančiu padu, bei 200 mm pločio lazeriu. Sistema yra skirta didelio tikslumo objektų skanavimui. Su įrenginiu pateikiama ir programinė įranga „Surveyor Scan Control (SSC)“ apdoroja gautus duomenis iš skirtingų orientacijų perkelia į trimatį taškų debesėlį esantį įprastoje koordinačių sistemoje. Visas procesas užtrunka tik keletą minučių ir yra labai tikslus. Galima netgi apversti objektą ir skanuoti jį antrą kartą ir gautus du duomenų masyvus automatiškai sujungti į vieną objektą.

Taip pat galime rasti pačių paprasčiausių įrenginių naudojančių pigius įrenginius ir nemokamas programas. Tokių prietaisų veikimo principai gali būti labai panašūs, tik jų tikslumas gali būti labai mažas. Tokie įrenginiai panaudojami smulkių objektų perkėlimui į kompiuterį moksliniais tikslais.

Pav. nr. 5



Pav. nr. 6

2.2 C ++ kalbos analizė

C – efektinga programavimo kalba. Ji leidžia geriausiai išnaudoti kompiuterinius resursus. C kalba parašytos programos yra kompaktiškos ir greitai vykdomos. C – mobilioji programavimo kalba. Tai reiškia, jei programa parašyta šia kalba, ji gali būti lengvai, su nedideliais pataisymais arba visai be jų, perkeliama į kitas skaičiavimo sistemas, pvz.: iš *IBM* kompiuterio perkelti programos veikimą į *UNIX*.

C – galinga ir lanksti programavimo kalba. Didelė dalis galingos *UNIX* ir *Windows* operacinės sistemos parašyta C kalba. C kalba parašytos programos naudojamos fizikiniams ir techniniams uždaviniams spręsti, taip pat naudojamos ir animacijai kurti.

C – turi galimybę panaudoti eilę valdančiųjų konstrukcijų, kurios paprastai asocijuojasi su asembleriu.

Programavimo procesas

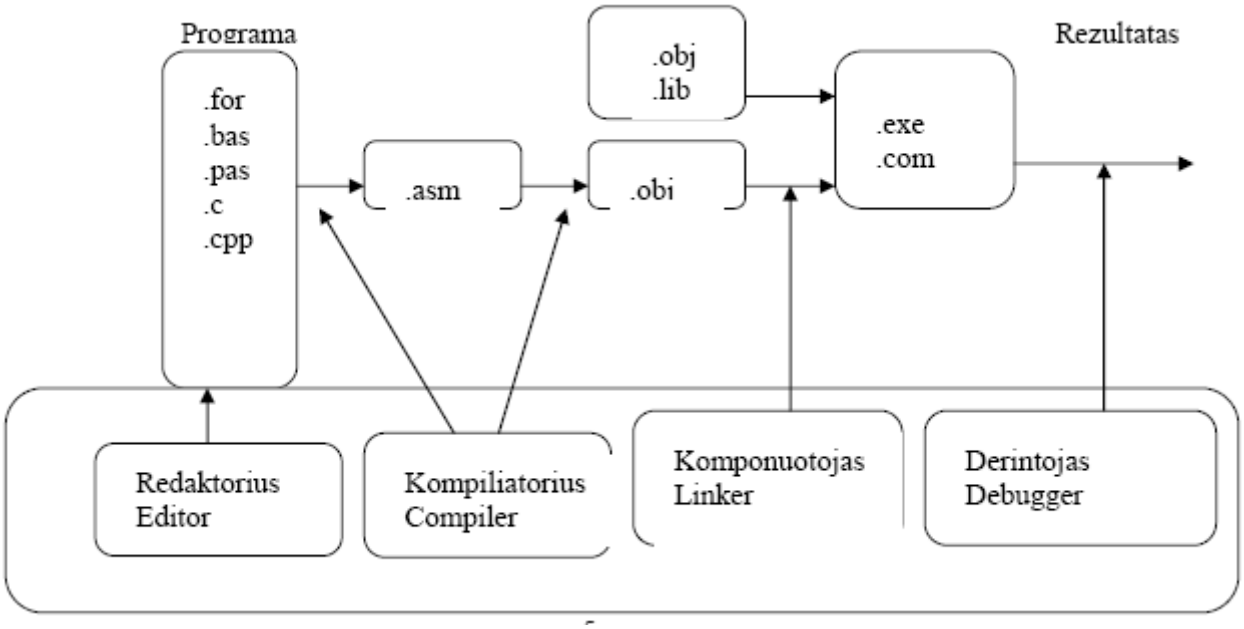
Programos rašymas yra gana ilgas procesas, kuris schematiškai yra pavaizduotas 7 pav. Pirma, pasinaudojus turimu redaktoriumi, rašomas programos tekstas. Čia svarbu neapsirikti renkant programos tekstą ir parinkti bylos vardą, kurioje bus programa. Programos teksto byla *.cpp arba *.c (pirmasis C++ kalba, antrasis – C).

Toliau parašytą programą kompiliuojame, t.y. mūsų parašytą programą perrašome kompiuterio kalba ir gauname objektinę (*.obj) bylą. Tai gali būti tik gabaliukas programos.

Galiausiai, visus programų gabaliukus sujungia į vieną *.exe bylą kita programa, vadinama komponuotuoju (linker). Ši programos dalis kartu įtraukia ir bibliotekines funkcijas. Tačiau gautoji programa yra su klaidomis, todėl ją reikia derinti ir taisyti. Tai atliekama derintojo (Debugger) programos pagalba.

Programuojant *C++* programavimo kalba pradinę specifikaciją yra geras pasirinkimas, nes *SystemC* yra *C++* sistemų modeliavimo biblioteka.

Modeliuojant sistemą aukštame abstrakcijos lygyje nesunkiai galime transformuoti pradinį *C++* programos kodą į *SystemC* transakcijų lygio modelį



Pav. nr. 7 Borland kompiliatorius

2.3 SystemC analizė

SystemC tai aukšto abstrakcijos lygio kalba skirta aparatūros aprašymui ir galinti imituoti aparatūros veikimą tam tikrais laiko intervalais.

SystemC skirstoma i 2 atskiras sritis:

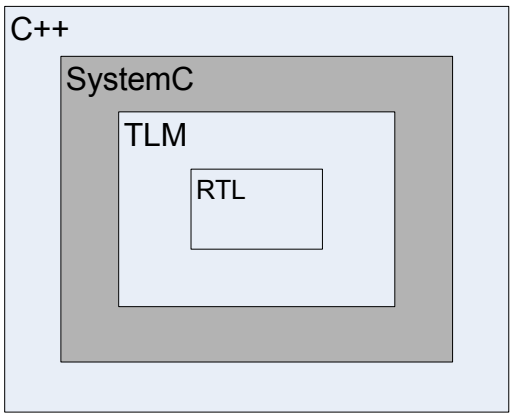
- ✓ *TLM* kalba (skirta modeliavimui)
- ✓ *RTL* kalba (skirta sintezavimui)

2.3.1 SystemC (TLM) transakcijų lygio modeliavimas

Transakcijų lygmens modeliavimas (*TLM*) yra aukšto lygio metodas skaitmeninių sistemų modeliavimui, kur tarp-modulinės komunikacijos detalės yra atskirtos nuo funkcinių dalių išpildymo ar komunikacijos architektūros. Komunikacijos mechanizmai, pvz., magistralės (duomenų mainai) ar *FIFO*, yra modeliuojami kaip kanalai ir yra sujungiami į modulius kaip *SystemC* sąsajos klasės. Transakcijų užklausos vyksta, kviesdamos sąsajos funkcijas tų kanalo modelių, kurie enkapsuliuoja žemo lygio informacijos keitimo detales.. TLM lygmenyje, pabrėžiamas duomenų perkėlimo funkcionalumas - kokie duomenys yra iš kur persiųsti ir į kur - mažai kreipiant dėmesį į jų žemiausio lygio išpildymą ar naudojamą tam protokolą. Šis metodas sisteminio lygmens projektuotojui leidžia išbandyti skirtingas sistemos modifikacijas, pavyzdžiui, skirtingą magistralių architektūrą, kai nebereikia iš naujo koduoti modelių, kurie bendrauja su magistralėmis, nes komunikuojama per bendrą sąsają.

TLM labiausiai tinka tarprocesiniui ir komunikavimo modeliavimui, bet ne algoritmų kūrimui, kurių pagrindu patys procesai veikią.

Sistemos elgsenos modelyje atsiranda subendrinti procesai, kurių vieni kuria duomenis, kiti juos naudoja, dar kiti inicijuoja įvairius ryšius arba pasyviai atsakinėja į kitų procesų užklausas.

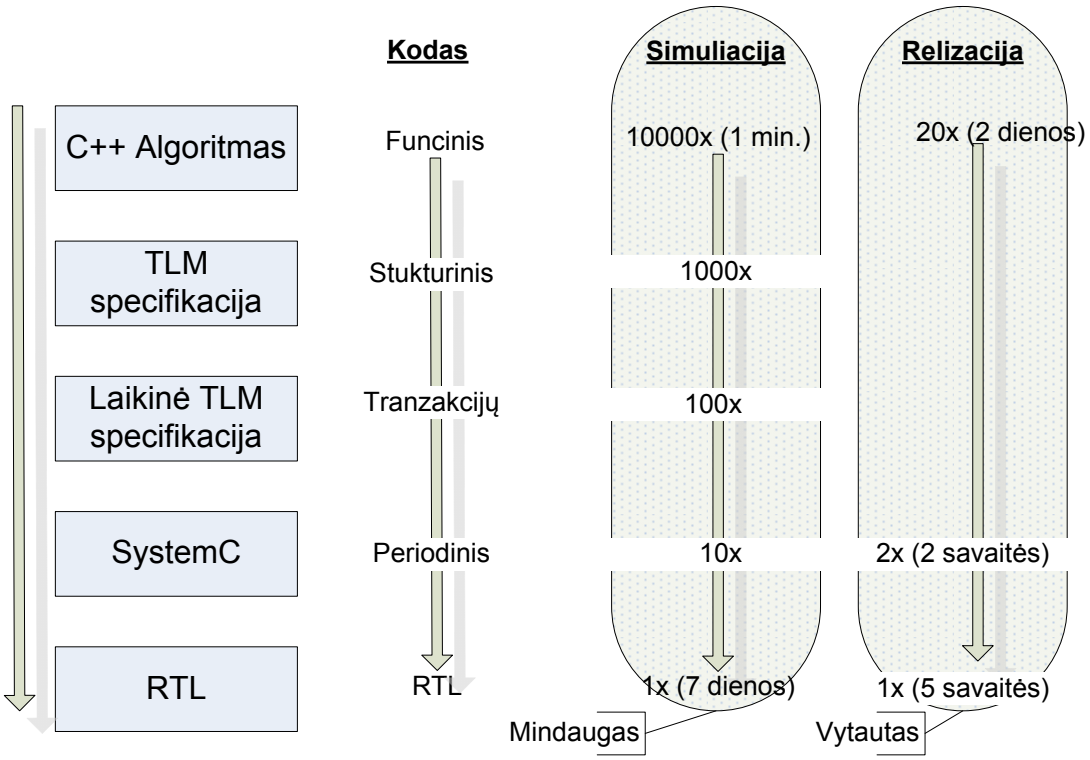


Pav. Nr. 8 C++/SystemC/TLM/RTL (abstrakcija)

Transakcijų lygio *SystemC* (*TLM*) ir *SystemC* yra paremta *C++* programavimo kalba. (pav. nr. 8)

Modeliuojant sistemą yra svarbu sutrumpinti sistemos kūrimo laiką, panaudoti jau rinkoje esančius komponentus (pakartotinis panaudojimas jau esamu sukurtų komponentu ir, arba pridedant išorinius procesorius.)

Kadangi *TLM* aprašymas yra aukštesnio abstrakcijos lygio, tai padeda išvengti daugybės klaidų, kurios būdingos *RTL* lygmeniu. *RTL* lygyje analizuojama kiekvienas atminties bitas, aprašoma [2]



Pav. Nr. 9 modeliavimo etapai, simuliacijos ir realizacijos sudėtingumo didėjimas**,** aprašoma [9].

Diagramoje yra pateikiama sistemos funkcinio modelio transformavimo žingsniai (kokios priemonės yra naudojamos skirtinguose žingsniuose). Yra pateikiama kaip didėja:

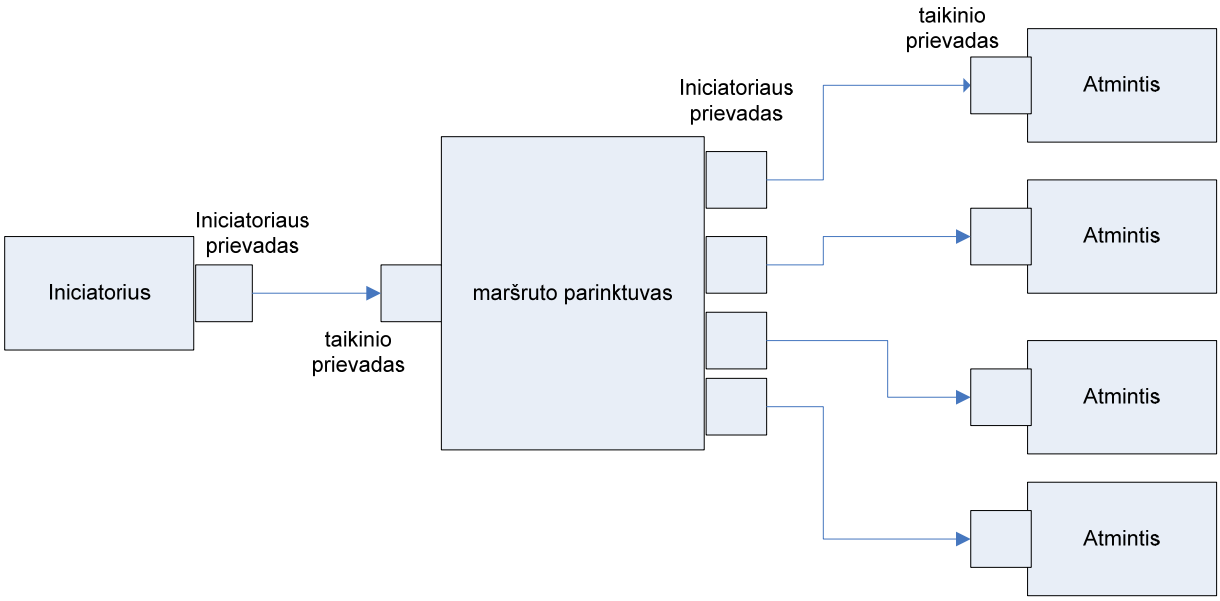
- Abstrakcijos lygis.
- Sudėtingumo lygis.
- Realizacijos laikas.
- Modeliavimo laikas.

2.3.2 *TLM* kalba

TLM pirma versija jau yra *OSCI* standartas. Projekte naudojama *TLM* antroji versija šiuo metu nėra paskelbta kaip *OSCI* standartas, tikimasi, kad po viešo aptarimo antroji versija taip pat taps *OSCI* standartu. Viešas aptarimas vyks birželio mėnesį.

Problemos iškylančios dėl sistemos neišbaigtumo:

- Tad dokumentacija nėra labai geros kokybės.
- Nėra daug straipsnių ir informacijos.
- Mažokai pavyzdžių.



Pav. nr. 10 TLM pavyzdys, aprašoma [4]

TLM suderinamumas

TLM standartas orientuojasi į *SoC* sistemų suderinamumą (daugelio gamintojų patiekiamų sistemų suderinamumą), kurios yra pagristos atminties elementais pririštais prie magistralių. Taip pat yra pateikiamas bazinių sistemų pririšimas prie geležies, be kurio negalimas sistemų suderinamas.

TLM sistemos kūrėjai yra susifokusavę į šiuos sistemų modelius.

- Modeliavimas netikslaus laiko sistemų pagrįstas atminties pririštos prie magistralių modeliavimu (*Modeling of loosely-timed systems based on memory-mapped busies*)
- Modeliavimas apytikrio laiko sistemų pagrįstas atminties pririštos prie magistralių modeliavimu (*Modeling of approximately-timed systems based on memory-mapped busies*) aprašoma [5]

Kadangi yra labai daug atminties tipų pririštų prie magistralių, tai TLM modelyje yra problematiška juos visus įvertinti (reikalinga naudoti parametrus ar atributus).

- Bendrinė TLM specifikacija (modelis) turi būti sudaryta iš vartotojo apibrėžtų šablonu.
- Žemo lygio kintamieji yra rekomenduotini vartotojams naudojantiems šablonus.

2.3.4 Transakcijų lygio modeliavimo sritys ir pritaikymo galimybės

Modelių naudojimas, SW/HW integracija

Naudojantis modeliais galime:

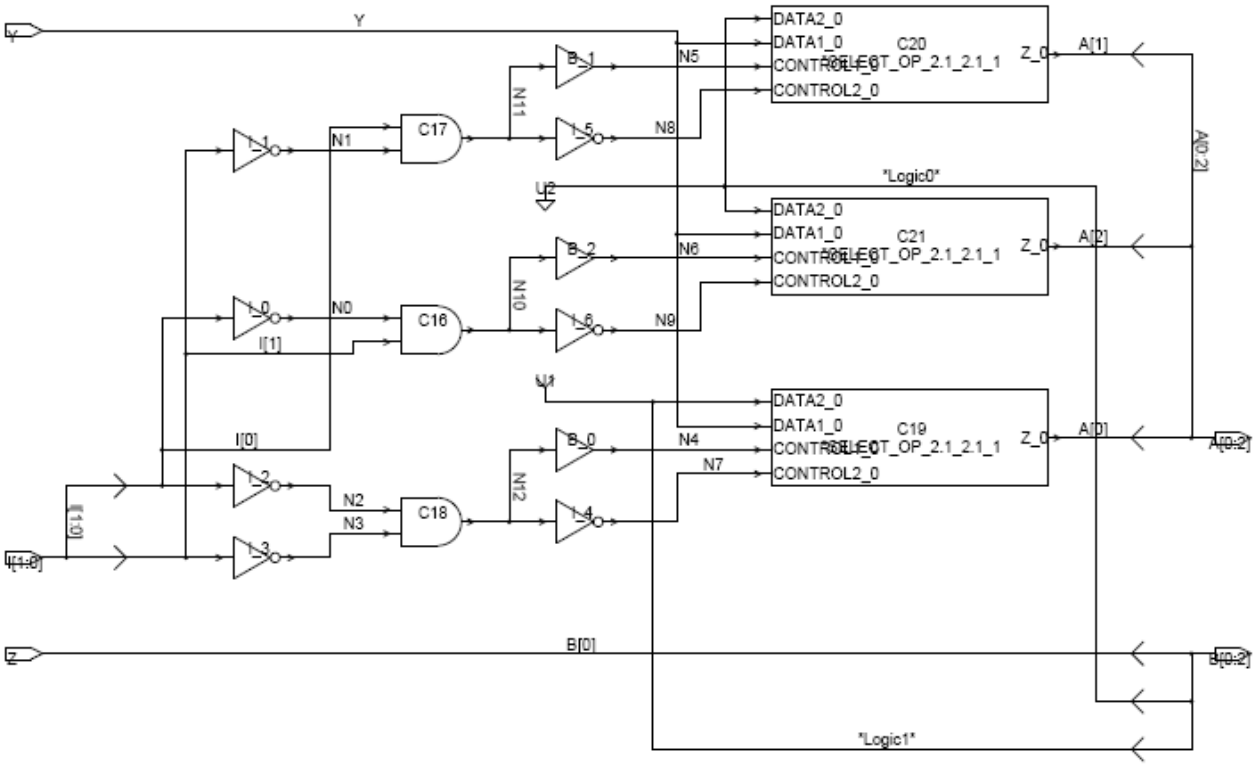
- Programinės įrangos kūrimas naudojantis TLM modeliais (operacinės sistemos tvarkyklių kūrimas, aplikacijų rašymas naudojantis valdyti ar gauti duomenis iš aparatūros).
- Prie aparatūrinių modelių (parašytu TLM) galime prijungti derinimo sistemas.
- Naudodami „loosely timed“ modelius, galime kurti tikras programas (programos kurios išnaudos aparatūrą)
- Programose galime įvertinti bitų eiliškumą. (realios programinės įrangos kūrimas ir testavimas)

Modelių našumo įvertinimas.

- Modelių patikros frazėse galime įvertinti integruotos programinės įrangos teisingumą, išmatuoti tikėtiną integruotos programinės įrangos greitį.
- Kurti skirtingus integruotos programinės įrangos profilius, optimizaciją.
- Greitas modelių modifikavimas.

Aparatūros architektūros analizė.

- Anksti galime specifikuoti aparatūrą, nustatyti reikalingus resursų modelius. (*TLM* fokusuojasi daugiau į atminties posistemes ir aukšto dažnio spartintuvus)
- Pakankamai tikslus aparatūros našumo nustatymas.
- Sistemos didelis lankstumas, didelės konfigūravimo galimybės, aukštas modelių efektyvumas.
- Galime naudoti tikrą programinę įrangą patikrinti modelio funkcionalumą, aprašoma [6]



Pav. nr. 11 RTL sistemos pavyzdys

Modeliuojant sistemas *RTL* lygyje naudojami signalai, ventiliai procesai ir registrai. Kad patikrintume visus sumodeliuotus modelius reikia daugiau laiko, nes reikia patikrinti kiekvieną signalą, jo sujungimą. Naudojami algoritmai yra sudėtingesni ir ,žinoma, sistemos modeliavimas yra žymiai sudėtingesnis nei transakcijų lygyje.

Naudodamiesi *SystemC* transakcijų lygio modeliavimu, mes mėginsime sumodeliuoti savo pasirinktą modelį. Toliau pateikiama panašių modelių analizė naudojantis *TLM*.

Pateiktajame straipsnyje analizuojama *SystemC TLM* pritaikymo galimybė.

Analizuota sistema yra realaus laiko įterptinė sistema, kuri yra labai panaši i mūsų analizuojamą sistemą. Patiekiami algoritmai ir rezultatai. Naudojantis *TLM* pavyko įgyvendinti savo tikslus t.y. sumažinti sistemos kūrimo laiką, gauti sistemos techninius duomenis, greitai pritaikyti kelis algoritmus (greitai modifikuoti sistemą).

Šiame straipsnyje analizuojama daugiaprocesorinė vienlustė sistema, kuri aprašoma ir modeliuojama naudojantis *SystemC* bei transakcijų lygio modeliavimo biblioteka.

Straipsnyje analizuojama didelės sistemos modeliavimo problemos sistemos modelio parinkimo *TLM* modeliavimo koncepcija. Pateikiama analizė skirta specifinių algoritmų vaizdų apdorojimo sistemoms kurti.

2.3.5 RTL sintezės taisyklės

Sintezė - objekto aprašo transformavimas į *bit vektorius*. *Bit vektoriai* yra apibrėžiami skilčių skaičiumi (duomenų pločiu). Skilčių skaičius nusako maksimalią *bit vektoriaus* kintamojo reikšmę. Kaip pavyzdį galime panagrinėti *Boolean* tipo kintamąjį, turintį tik vieną bitą. Jis gali būti analogiški aprašomas kaip *sc_uint<1>* arba *sc_big_int<1>*; Skilčių skaičius paprastai turi būti nurodomas pradžioje, bet naudojant klasių tipą, tai gali būti nutylėta ir sintezės metu automatiškai apskaičiuota. Taip pat reikia skirti realius objektus ir menamus. Paprastai su duomenų įėjimo vektoriumi nevykdomi jokie veiksmai tik priskirimas pabaigoje. Visi skaičiavimai būna vykdomi papildomuose *bit vektoriaus* kintamuosiuose (dažniausiai vadinami *temp*).

Aparatūros realizacija su skaičiavimais remiasi postūmiais. *RTL* naudoja postūmių procesus operacijų vykdymams. Tai galima aprašyt naudojantis SC_METHOD procesą su jautrumo sąrašu paimant tik teigiamus sinchronizacijos signalo įėjimo frontus. Taip pat įmanoma naudoti SC_CTHREAD procesą ir elgsena nuo to nepasikeis. Kaip bebūtų SC_CTHREAD procesas yra mažiau efektyvus ir modeliavimas dėl to vyks lėčiau. Sinchronizacijos signalas turėdamas teigiamą frontą tikrina, ar signalas pasikeitė, ir tikrina, ar jo reikšmė lygi vienetui. Tada pagal jį paprastai priskiriamos duomenų įėjimo reikšmės. Pabaigoje proceso postūmio darbalaukis yra priskiriamas išėjimo signalams.

Sudėtingesni įtaisai yra aprašomi būsenų automatais. Būsenų automatas startuoja visada nuo pirminės būsenos. Vėliau pagal įėjimų reikšmes būsenos keičiasi. Būsenų automato realizacija apsprendžiama vienu-dviem SC_METHOD procesais. SC_METHOD procesai yra patys efektingiausi ir juos rekomenduojama naudoti visur.

Sintezuojant reikia atkreipti dėmesį į sintezės taisykles

Sintezės taisyklės

Moduliai gali palaikyti

- Pavienius arba lygiagrečius procesus aprašant kombinuotą arba nuosekliąją logiką.
- Lygiagrečius modulius aprašant hierarchijas
- Funkcijų kintamuosius

Procesai

- Tik SC_METHOD yra palaikomas RTL sintezės (naudojant *Synopsio SystemC* kompiliatorių)
- SC_CTHREAD tik palaikomas elgsenos sintezėje (pagal *Synopsio SystemC* kompiliatorių).

Procesas

- SC_METHOD

Jautrumo sąrašas

- Visi signalai, kurių procesas yra jautrus, turi būti surašyti jautrumo sąraše.

Pilnas aprašymas

- Visos kombinacijos įėjimų reikšmių turi būti padengtos

Kintamieji

- Gali būti prieinami SC_METHOD procesams.

Duomenų tipai

- Sintezę palaiko ne visus duomenų tipus (žiūrėti Lentelė. 1)

Duomenų tipai	Skilčių skaičius	Palaikymas sintezei
Sc_bit: gimininga c++ bool	2	Rekomenduojamas
Sc_bv	2	Rekomenduojamas
Sc_logic and sc_lv<n>	4	Tik kai reikia
Sc_bv<n>:sc_uint<n>	1-64	Rekomenduojamas
Sc_uint<n>:max 64 bitai su ženklu	1-64	Rekomenduojamas
Sc_int<n>:max 64 bitai be ženklo	1-64	Rekomenduojamas
Sc_bigint<n> and sc_biguint<n>	>64	Tik kai reikia

Lentelė. 1 Duomenų tipai

Pagrindiniai *SystemC* projektavimo blokai

Registrai (DFF) yra vienas iš pagrindinių RTL projektavimo blokų. Praktikoje dažniau naudojami (D flip flops) registras su asinchroniniu *reset* signalu yra vienas iš bazinių RTL projektavimo blokų. Panaudojant vieną kartą *reset* signalą projekto eigoje projektuotojai gali nustatyti registrą (flip flops) į pradinę žinomą būseną.

`Poslinkiai`

```
//DFFa.h
#include „systemc.h“

SC_MODULE(dffa)
{
  Sc_in<bool> clock;
  Sc_in<bool> reset;
  Sc_in<bool> din;
  Sc_out<bool> dout;
  Void do_ffa()
{
  If (reset)   Dout = false;
 Else if (clock.ecent()) dout=din;
}
SC_CTOR(dffa)
{ SC_METHOD(do_ffa);
  Sensitive (reset);
  Sensitive_pos (clock);
}
};
```

Postūmių registras *(shifter)* -tai daug sudėtingesnis įrenginys galintis ne tik nustatyti į pradinę, bet ir į bet kurią būseną. Paprastai turi duomenų įėjimo *(data), reset*, ir duomenų užkrovimo (*load)* galimybes.

Apačioje pateikiame postūmio registro pavyzdį

```
#include “systemc.h
SC_MODULE(shift)
{ sc_in<sc_bv<8> > din;
  sc_in<bool> clk;
  sc_in<bool> load;
sc_in<bool> LR;
sc_out<sc_bv<8> > dout;
sc_bv<8> shiftval;
void shifty();
SC_CTOR(shift);
{ SC_METHOD(shifty);
  sensitive_pos (clk);
 }
};
```

```
//shift.cc
#include "shift.h"
void shift::shifty()
{ if (load)
     shiftval = din;
   else if (LR)
   {  shiftval.range(0,6) = shiftval.range(1,7);
       shiftval[7] = '0';
   }
  else if (!LR)
   {  shiftval.range(1,7) = shiftval.range(0,6);
       shiftval[0] = '0';
   }
dout = shiftval;
}
```

Būsenų automatas

Būsenų automatas naudojamas aprašyti sudėtingiems įtaisams. Būsenų automatas visada startuoja į pradinę būseną. Vėliau naudodamasis įėjimo reikšme būsenos keičiasi. Kaip pavyzdys būsenos gali būti kartojimo, įrašymo, ištrynimo ir t.t.

Apačioje pateiktas pavyzdys

```
// busenų automatas
SC_MODULE(automatas)
{ sc_in<bool>clk;
  sc_in_<char> key;
  ....
 sc_out<sc_logic> operation;
 sc_out<sc_logic>address;
void getnextst();
void setst();
SC_CTOR (automatas)
{ SC_METHOD(getnextst);
   sensitive << key << curent_state;
   SC_METHOD(setst)
   sensitive_pos (clk);
  }
};
```

Atminties elementas

Atminties modelis yra skirtas talpinti ir dirbti su didesniais duomenų kiekiais masyvais ir matricomis. Atmintis blokas susideda iš būsenos aktyvavimo signalo (*RnW*- rašyti ar skaityti), turi vieną įėjimą-išėjimą duomenims (*data*) ir kintamąjį (*address*), kuris nurodo iš kurios atminties vietos imame, ar į kurią rašome informaciją.

Apačioje pateikiamas atminties elemento pavyzdys:

```
//ram.h
#include „systemc.h“
SC_MODULE(ram)
{ sc_in<sc_int<8> > address;
  sc_in<bool> RnW;
  sc_inout_rv<16> data;
 void read_data();
 void write_data();
 sc_lv<16> ram_data[256];

SC_CTOR(ram)
{ SC_METHOD(read_data);
  sensitive << address << RnW;
  SC_METHOD(write_data);
  sensitive << addres<< RnW << data;
 }
};
// ram.cc
#include ram.h
void ram::read_data()
{
   If(!RnW)
     data = ram_data[address.read()];
   Else
     data = „zzzzzzzzzzz“;
}
void ram::write_data()
{
   If(RnW)
     ram_data[address.read()] = data;
}
```

***SystemC* kalbos architektūra**

Paveikslėlis pav. nr. 12 apibendrina *SystemC 2,0* kalbos architektūrą. Čia yra keletas svarbių apibrėžimų šioje diagramoje.

SystemC yra sukurta *C++* pagrindu

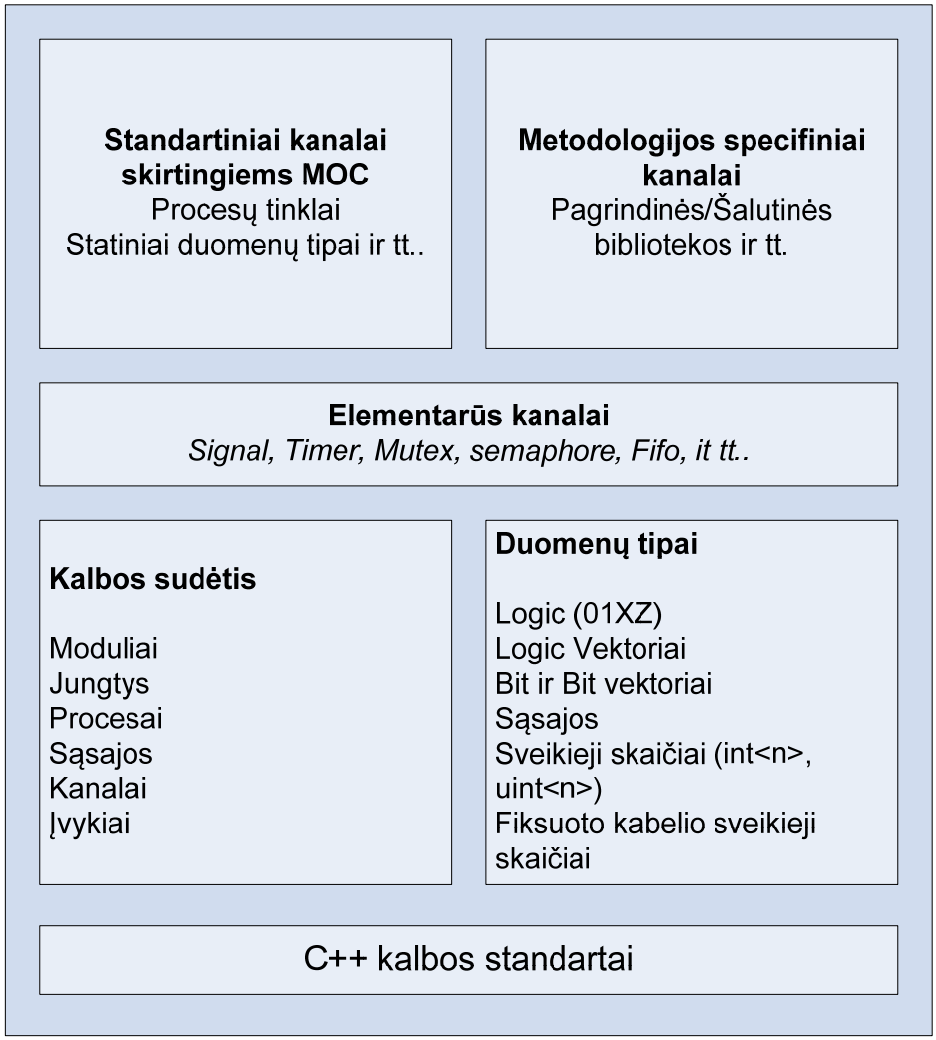
Viršutiniai diagramų sluoksniai yra pastatyti remiantis žemesniais sluoksniais

SystemC kalba apibendrina tik minimalias modeliavimo konstrukcijas struktūriniu aprašymu, vienalaikiškumu, komunikavimu ir sinteze.

Duomenų tipai sintezes atveju yra skirtingi nuo kalboje palaikomu duomenų tipų, bet vartotojo susikurti duomenų tipai yra pilnai palaikomi.

Bendrai naudojama bendravimo mechanizmai, tokie kaip signalai, *fifos* ir (*MOCs*) gali būti sukurti aukščiausiame modulio lygyje.

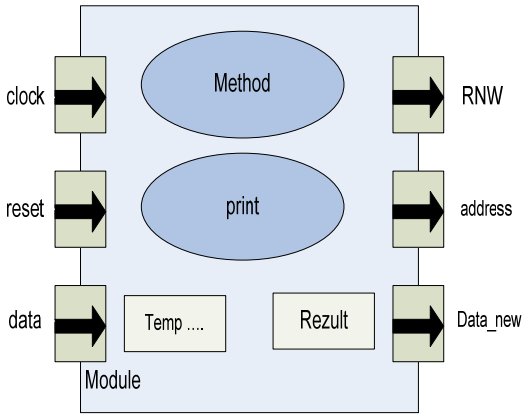
Yra galimybė žemesnius diagramų sluoksnius naudoti ir be aukštesnių sluoksnių.



Pav. Nr. 12 – *SystemC* 2,0 kalbos architektūra.

Mūsų nagrinėjama sistema atrodo taip, kaip pavaizduota paveikslėlyje pav. 13

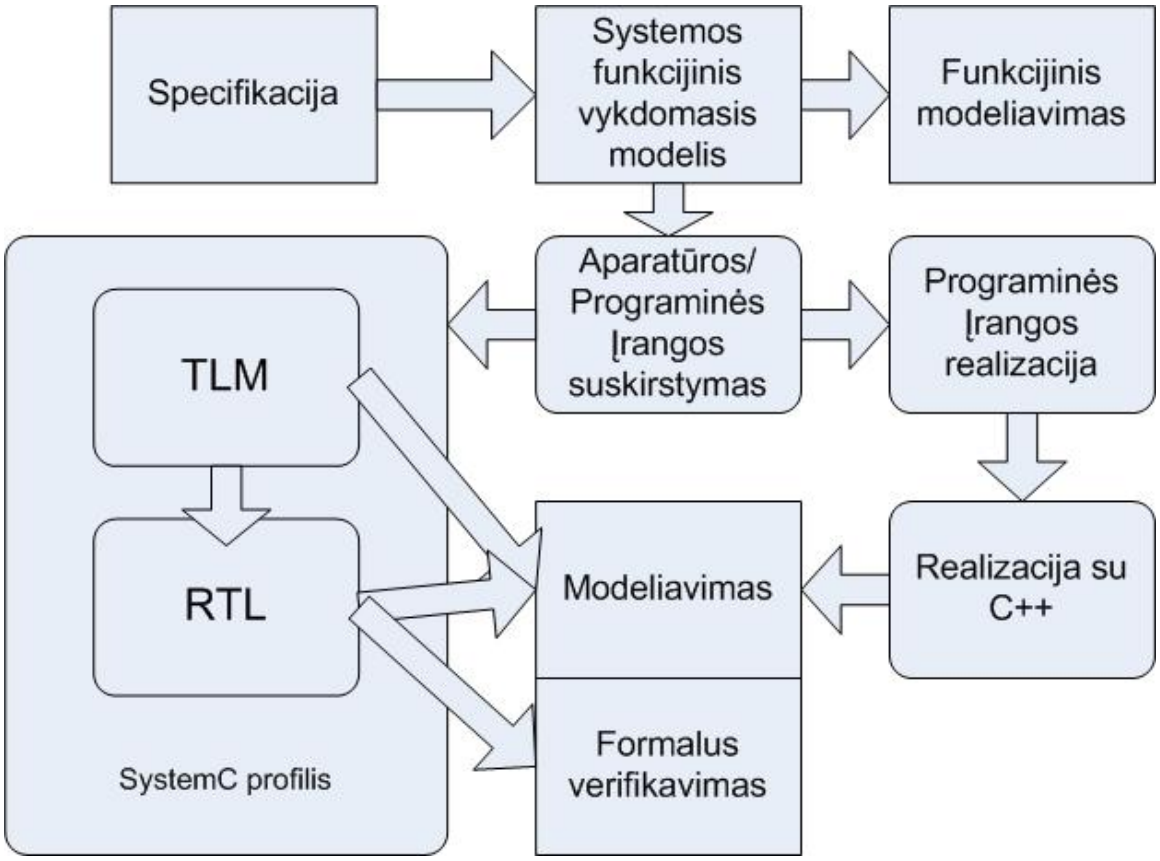
```
SC_MODULE(MODULE)
{ sc_in_clk clock;
   sc_in_clk reset;
   sc_in <sc_uint<9> > data;

   //ram kintamiji
   sc_out<bool> RnW_s;
   sc_out<sc_uint<10> > address_s;
   sc_out<sc_uint<9> > data_new;

   sc_uint<3> STATE;
  //papildomu kintamuju eile
  sc_uint<n> temp;
  void mudule();
  void print();
  SC_CTOR(MODULE)
  {
     SC_METHOD(module);
     sensitive << clock.pos() <<
reset.pos();
     SC_METHOD(print);
     sensitive << clock.pos();
  }
};
```



Pav. nr. 13

Turime 3 įėjimo signalus *clock*, *reset*, ir data ir tris išėjimo signalus *RnW*, *address*, *data_new* tai yra mūsų objektai. Taip pat turime tam tikrą skaičių papildomų *temp* signalų, kurie yra naudojami tik modulio viduje.

2.4 Projektavimo įrankiai



Pav. nr. 14 -vienlustės sistemos projektavimo diagrama (*UML-SoC design flow)*

Projektavimo procesą pradedam nuo specifikacijos, sudarytos pagal sistemos reikalavimus. Tai dažniausiai būna funkcinis sistemos modelis aprašytas natūralia kalba.

Sistemos funkcinis vykdomasis modelis(arba algoritmo specifikacija) yra modelis pagal reikalavimus fiksuojantis sistemos elgseną. Tai dažniausiai yra daroma naudojant programinės įrangos programavimo kalbas (C/C++) arba modeliavimo įrankius tokius, kaip *Simulink*. Toks funkcinis lygis apibrėžia pilną sistemos modelį įskaitant tiek aparatūros, tiek programinės įrangos dalis, kurie yra verifikuojami su aukštu abstrakcijos lygiu.

Esminis pasirinkimas - sudalinimas sistemos į aparatūros ir programinės įrangos dalis, kas apsprendžia galutinį skirtingų dalių likimą ir esamos sistemos galimybes.

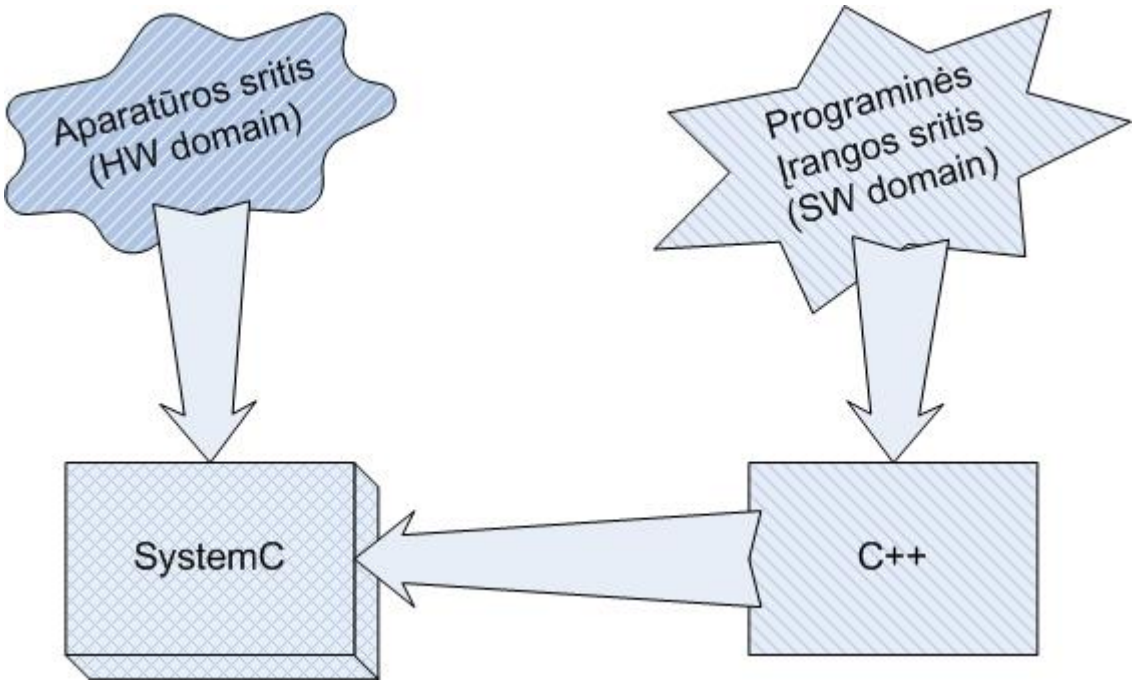
Aparatūros dalis pradžioje apibrėžiama aparatūros aprašymo kalba RTL lygyje.

RTL aprašas automatiškai gali būti transliuojamas į pasirinktą technologinę platformą. Transliavimo metu yra naudojama loginė sintezė. Loginės sintezės rezultatas yra struktūrinis aprašas (*netlist.)* Po to atliekamas fizinis projektavimas. Fizinio projektavimo metu struktūrinis aprašas (*netlist*) papildomas informacija apie automatinį išdėstymą ir sujungimų transliavimą.

UML integracija į projektą leidžia pereiti į kitą projekto abstrakcijos lygį. UML panaudojimas projektuose: vizualizavimas, pernaudojimas, integracija, dokumentacija, modelio analizė ir automatinis generavimas *SystemC* kodo. Tranzakcijos lygis pasižymi tuo, kad palaiko tiek programines, tiek aparatūros dalis.

Programinė dalis gali buti sumodeliuota ir duoti laiko diagramas, pagal kurias matome elgsenos modelį. Tam panaudojame *testbench* modelį.

Tuo tarpu aparatūros modelis mums leidžia dar ir verifikuoti projektą ir sumodeliuoti lustą.



Pav. nr. 15 ryšiai tarp modeliavimo kalbų (*Relations of design languages*)

15 paveikslėlyje parodytas ryšys tarp programinės įrangos ir aparatūros

Kaip matome *SystemC* apjungia tiek programinę, tiek aparatūros dalį.

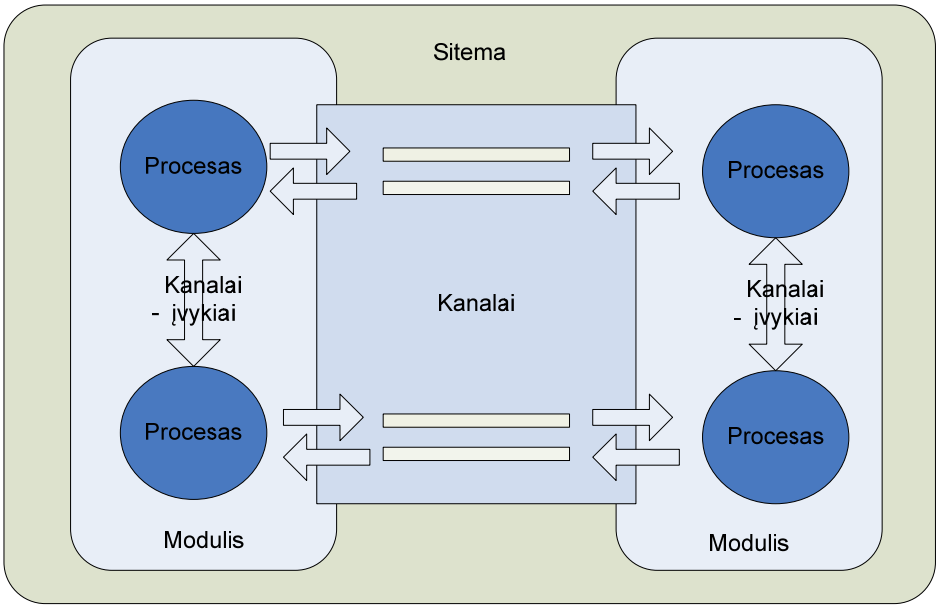
2.5 Projektavimo metodika

SystemC kalbos analizė

SystemC - tai aukšto abstrakcijos lygio kalba, galinti imituoti aparatūros veikimą tam tikrais laiko intervalais. Kanalai yra viena yra iš priemonių aprašyti sistemos veikimą.

Kanalai leidžia patogiai tvarkyti komunikacijų sistemas. Jie gali modeliuoti paprastas komunikacijas, arba pristatyti kompleksines schemas, kaip specialias magistrales. Kanalai yra specifikuojami pagal sąsają, kuri palaiko juos.

- Procesai bendrauja tarpusavyje per kanalus.
- Prisijungimas prie kanalų per sąsajas.
- Sąsajos gali būti skirtingos.
- Moduliai yra sujungti su kanalais jungtimis.
- Jungtys yra objektai modeliuojantys procesų prieigą prie kanalų.
- Jungtys gali būti sujungtos arba virtualiais kanalais (*channel*), arba sąsajom (*interface*).



Pav. nr. 16 Hierarchinė struktūra sistemos projektavimo

SystemC aplinkoje procesų pagalba aprašomas funkcionavimo mechanizmas. Procesai yra vykdomi lygiagrečiai, tokiu būdu imituojamas realios aparatūros funkcionavimas.

Modeliuojant pasikeitimus *SystemC* naudoja procesų koncepciją paaiškinta pav. nr.16. Kaip ir kitos aparatūros aprašymo kalbos (*Verilog*, *VHDL*,…) *SystemC* modeliavimas vyksta branduolyje, kuris keičiasi esant skirtingiems konkuruojantiems procesams. Modeliavimo programa (modelis) pati nustato šiuos pasikeitimus. Jei įvykis įvyksta, tai sistema aktyvuoja tik tuos procesus, į kurių jautrumo sąrašą yra įtraukti minėti įvykiai.

2.5.1 *SystemC* komponentai

Duomenų tipai

Be *C++* savųjų duomenų tipu, *SystemC* dar palaiko aparatūrą apibrėžiančius duomenų tipus. Sprendimai dėl specialių duomenų tipų yra apibrėžiami duomenų pločiu, tikslumu ir operacija. Pasirinkimas atitinkamų duomenų yra svarbus, kadangi tai daro didele įtaką sintezavimo rezultatams.

	Duomenų tipai
Sąrašas bendrųjų duomenų tipu tarp *C++* ir *SystemC*	Int Real, Char Bool Float Double
Sąrašas aparatūrą apribojančių duomenų tipų	• sc_bit Loginis bitas (0 arba 1) • sc_logic Daugiareikšmis duomenų tipas (0 1 X Z) • sc_int<n> Sveikasis skaičius su fiksuotu žodžio ilgiu • sc_bv<n> Loginis ilgio n Bit - vektorius • sc_lv<n> Daugiareikšmis n ilgio Bit - vektorius

Lentele Nr. 2

2.5.2 Moduliai ir sąsajos

SystemC naudoja *SC_MODULE* makro aprašyti modulių klases.

```
SC_MODULE(module name )
{ modulio kūnas
}
SC_MODULE macro:

#define SC_MODULE(module name ) struct module name : public sc_module
```

Modulio kūnas susideda iš pasikartojančių elementų: kanalų , duomenų tipų, modulių objektų, sąsajų, konstruktorių, destruktorių, procesų ir pagalbinių funkcijų.

Sąsajos yra įgaliojimų objektai, kurie susieti kanalais tiesiai su sąsaja. Moduliai komunikuoja su jų jungtimis.

Jungtys aprašytos su skaitymo rašymo sąsaja:

```
Sc_port<sc_signal_in_out_if <int>> rw_port ;
//tas pats su  numatyta suntakse
sc_out<int> rw_port ;
```

2.5.3 Įvykiai, jautrumo sąrašas, pranešimai

Įvykis yra veiksmas inicijuojamas įrenginio arba vartotojo. *SystemC* įvykiai yra pagrindinis įrankis, sinchronizuoti skirtingus procesus. *SystemC* naudoja *sc_event* klases, kad modeliuotų įvykius. Įvykis yra elemento raktas į stimuliatoriaus darbo pasikeitimus. Įvykis neturi laiko tėkmės. Su įvykiais jūs galite atlikti tik du veiksmus: ***wait*** arba ***fire***.
Procesai yra aprašomi jautrumo sąrašu su vienu ar daugiau įvykių. Jei įvykis įvyksta, tai prasideda tą įvykį turinčių (pagal jautrumo sąrašą) procesų vykdymas.

Procesai

Modeliavimo programa negali pilnai imituoti tikro funkcionavimo. Yra tam tikras modeliavimo laiko diskretas, pagal kurį modeliavimo programa modeliuoja sistemos būsenas tik tais fiksuotais laiko intervalais.

Kiekvienas procesas apibrėžia įvykdymus mažomis programos fragmento porcijomis ir taip leidžia kontroliuoti kitus procesus.

SC_THREAD: Gija aktyvuojama modeliavimo procesuose tik vieną kartą. Po to, kai gija startuoja, prasideda vykdymas. Vykdymą galima pristabdyt su *wait* metodu.

SC_METHOD: skirtingai nei SC_THREAD negali savęs pristabdyti, jis įvykdomas iki galo. SC_METHOD yra konkretesnis nei SC_THREAD, todėl yra greičiau modeliuojamas.

Kanalai ir sąsajos

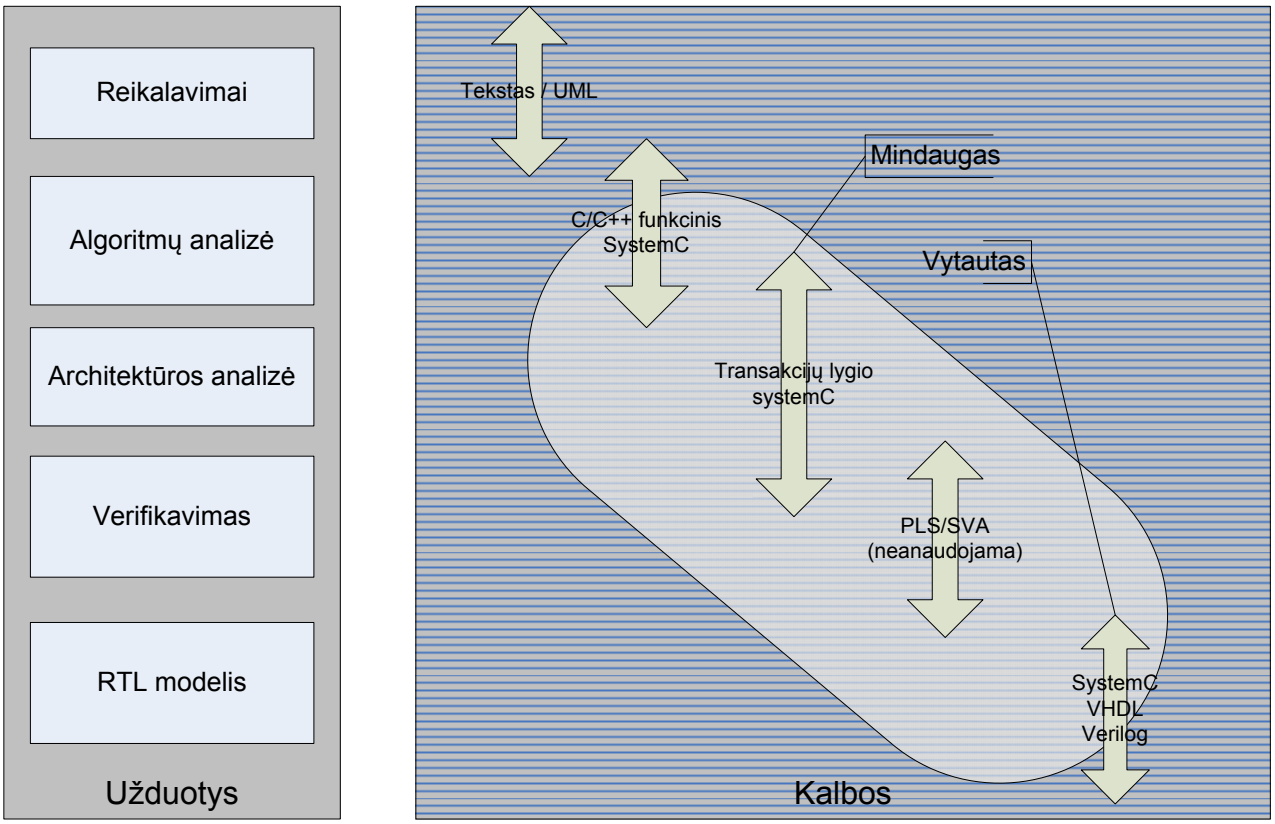
Signalai yra pakankama priemonė aprašyti sąsajas aparatūros procesuose, bet to nepakanka aprašyti visą sistemą. Kanaluose yra metodai skirti sąsajų aprašymui.
Komunikacija yra galima tarp modulių ir modulių viduje. Modulį gali sudaryti keli procesai.
Yra du skirtingi kanalų tipai - primityvus ir hierarchinis.
Primityvūs kanalai negali susidėti iš procesų arba tiesiogiai komunikuoti su primityviais kanalais.

	RTL	Funkcinis (TML)	Algoritminiai
• sc_signal	✔	✔	✔
• sc_fifo		✔	✔
• sc_mutex		✔	✔
• sc_semaphore		✔	✔
• sc_buffer	✔	✔	✔
C++ bendri			✔

Lentele nr. 3

Hierarchiniai kanalai: jie gali sudaryti labai sudėtingas komunikacijas, kadangi jie yra moduliai ir jie neturi aukščiau paminėtų apribojimų. Su hierarchiniais kanalais yra įmanoma modeliuoti abstrakčias komunikacines struktūras.

2.5.5 Aparatūros aprašymo kalbų vieta projektavimo procese



Pav. nr. 17 Modelio kūrimo kalbos ir užduotys.

Modeliuojant sistemas reikalingi įvairūs įrankiai priklausomai nuo projektavimo stadijos. Projektavimo reikalavimams mes surinkome apklausos būdu. Surinktus reikalavimus mes užrašėme ant popieriaus. Darbe neanalizuojamos reikalavimų surinkimo problemos.

Daugiau dėmesio yra skirta darbo algoritmų analizei, kaip juos perkelti į aparatūrą. Kad tai atliktume, mums teko apsispręsti, kokias kalbas naudosime. Paveikslėlyje nr. 17 pateikiama naudojamos kalbos.

Algoritmų analizei naudojome *C++* programavimo kalbą.

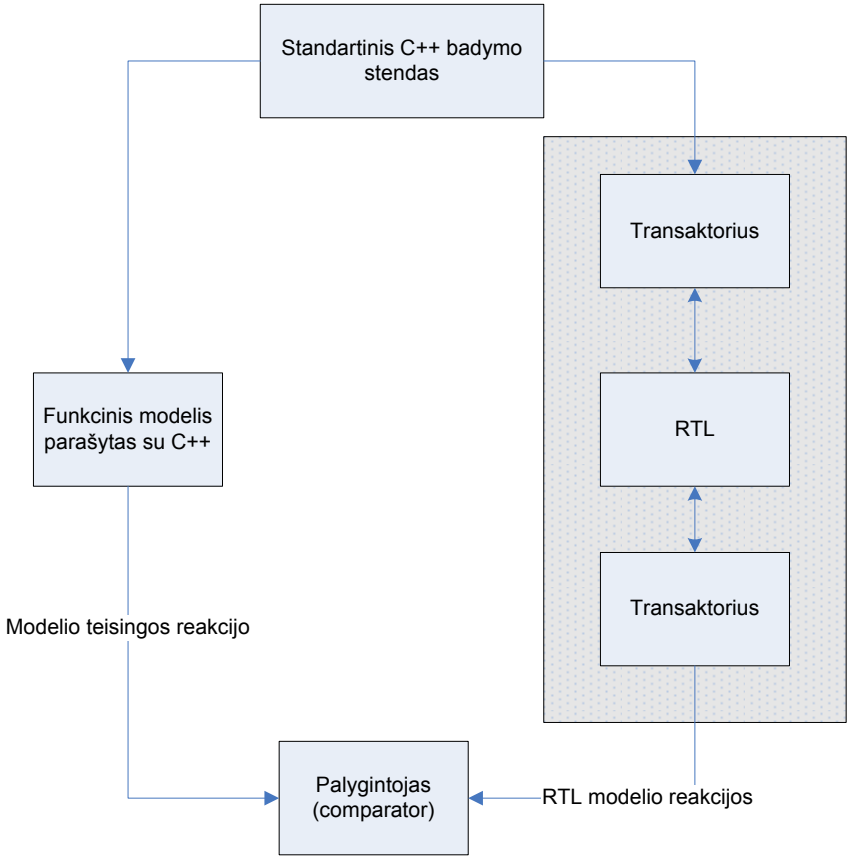
Architektūros analizei naudojame transakcijų lygio *SystemC*.

Magistriniame darbe mes daugiau kreipiame dėmesį į sistemų modeliavimą ir sintezavimą. Todėl magistriniame darbe neanalizuojame sistemos verifikavimo problemos, bet darbe yra atliekamas gautų rezultatų palygimas su funkcinio modelio rezultatais, aprašoma [8]

Mindaugas analizuoja *TLM* modeliavimą (*C++* kodo transformavimas į *SystemC TLM* aplinką).

Vytautas analizuoja modelio sintezavimą (*TLM* transformavimas į sintezuojamą *RTL* aprašą).

2.5.6 *TLM* taikymas automatinam verifikavimui.



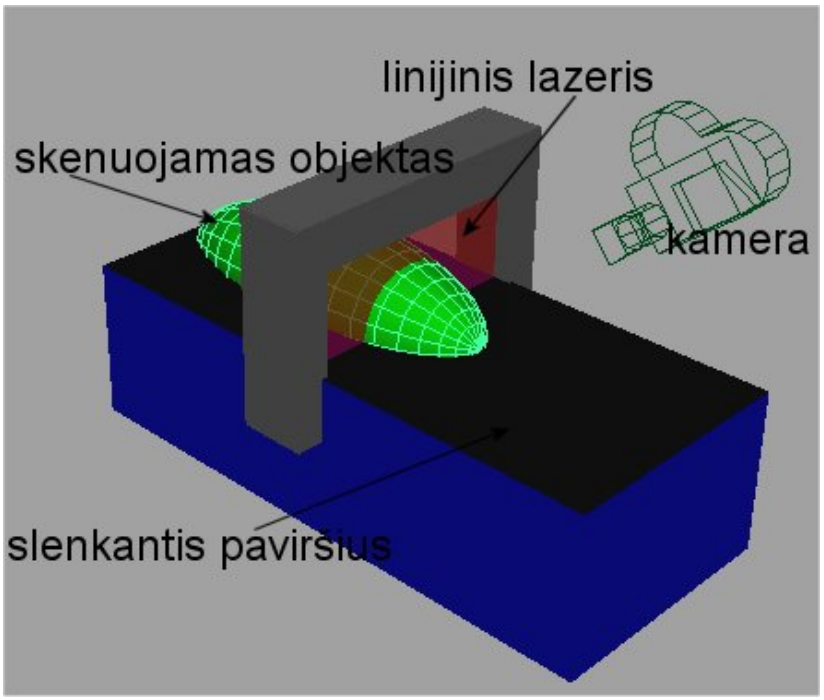
Pav. nr. 18 TLM panaudojimas automatinam sistemos verifikavimas**,** aprašoma [10].

Testų rinkinius, kurie buvo parašyti tikrinti *C++* modeliu, gali būti panaudoti skirtingiems abstrakcijos lygiams tikrinti. Tam tikslui mes galime pasitelkti tranzaktorius. Tranzaktorius - tai toks modulis ar sąsaja, kurios pagalba galime prijungti skirtingo abstrakcijos lygio modelius. Prie tranzaktorių galime jungti susintezuotus modelius. Taip sutaupome laiko modelių patikrinimui, aprašoma [7].

2.6 Funkciniai reikalavimai

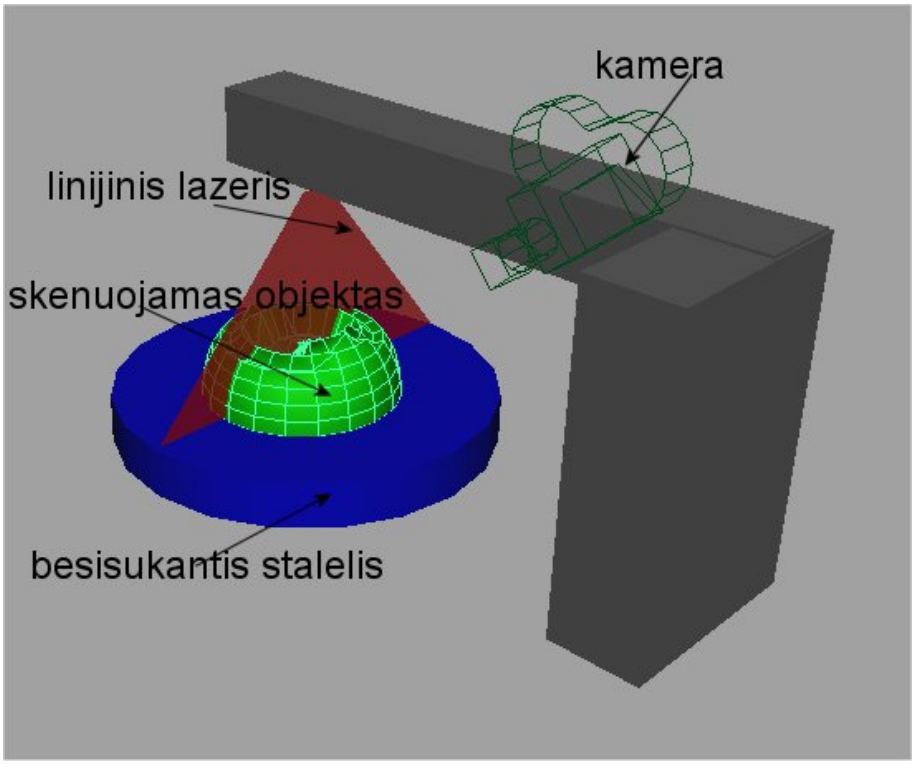
Mūsų pasirinkta sritis yra lazeriniai 3d skaneriai. Atitinkamai mes juos dar suskirstom į 2 dalis pagal skenavimo pobūdį: poslinkio ir pasisukimo kampo.

Poslinkio atveju daiktas yra padedamas ant judančio takelio ir slenkant pro lazerį atitinkamai fiksuojami pjūviai. Tokiu būdu objektas yra tarsi pjaustomas kaip duona. Vėliau atvaizduojant išlaikomi tie patys atstumai ir daiktas sudedamas tarsi supjaustytos duonos riekės, išrikiuojamos ta pačia tvarka, kaip kad buvo pjaustomos (Pav.:19).



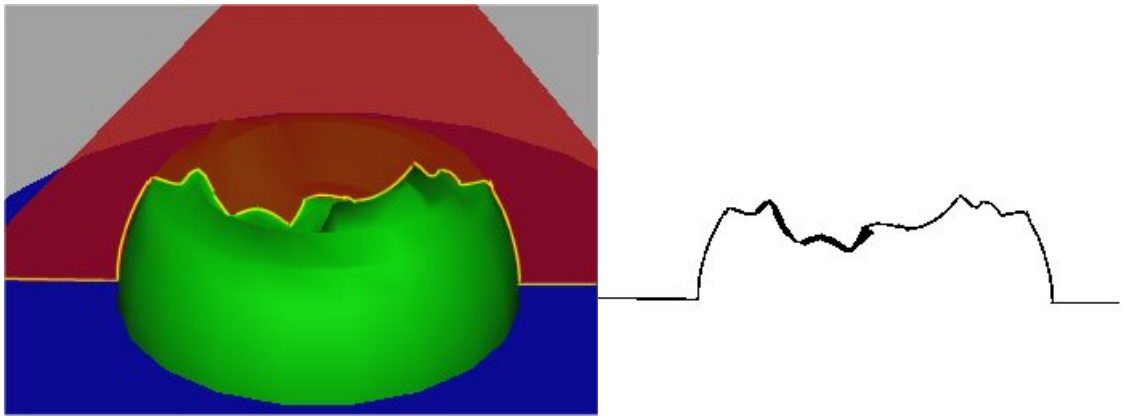
Pav. nr. 19

Pasisukimo atveju daiktas yra padedamas ant besisukančio stalelio ir įtvirtinamas, kad nepasislinktų nuo esamos padėties. Taip gautas linijų masyvas yra supjaustomas tarsi tortas, vėliau atvaizduojant linijos sudedamos tokia pat tvarka(pav.:20).



Pav. nr.20

Kaip tai veikia: lazeris apšviesdamas paviršių ant paviršiaus palieka ryškų pėdsaką. Dėl to naudojant kamerą ir paėmus didelį kontrastą gauname vien tik lazerio apšviestų vietų liniją. Naudojamos kameros rezoliucija yra 640X480 (pav.:21 a ir b).



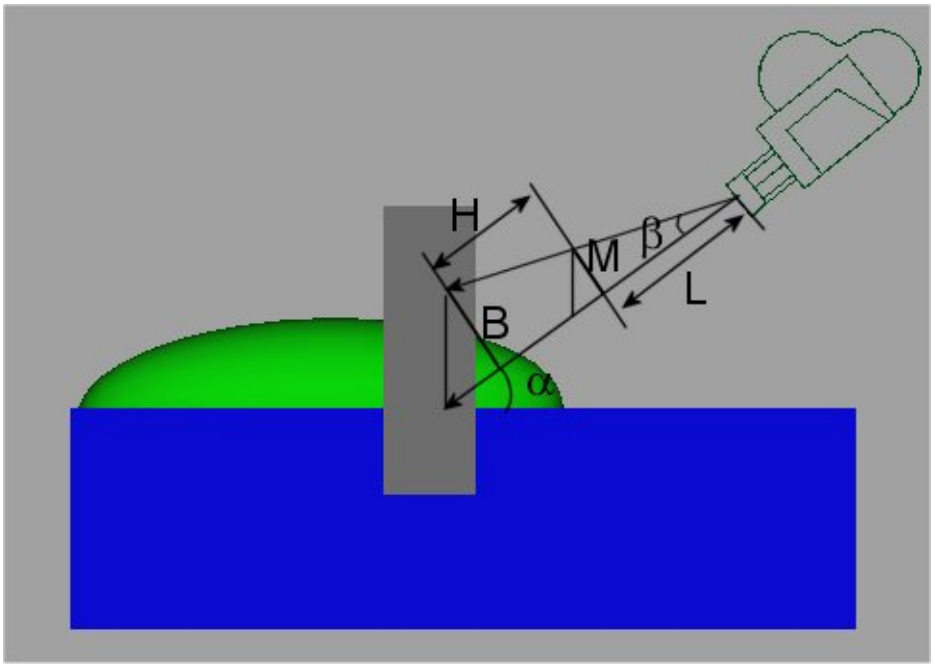
a b

Pav. nr. 21

Gauta kreivė dar nėra tiksli ir tikslumo dėlei skaičiuojame kiekvienam X yra perskaičiuojama reikšmė pagal gausą. Tokiu atveju vienai kreivei aprašyti gauname 640 taškų. Paprastumo dėlei koordinatės X nesaugojame, nes pagal šį principą tinklelis yra griežtai suskirstytas ir turi tik 640 skirtingų X reikšmių [0..639]. Taigi saugome vien tik aukščius Y

taip gauname Y[0..639]. Kadangi 640 taškų nėra labai daug atvaizduoti glotniai kreivei, kreivės glotninimui planuojame naudoti splainus.

Įvertinant kameros kampą, bei kameros didinimo bei mažinimo parametrus (perspektyvą) kamerą judinant ,gauname transformuotą vaizdą, kuris priklauso nuo 2 kampų α ir β (pav.:21).



Pav. nr. 22

Kampas α parodo, kokiu kampu yra pasukta kamera, o kampas β apsprendžia kameros mastelio nuo atstumo problemą. Kampo α keitimas kameroje duoda akivaizdžius pasikeitimus, tai matyti paveikslėliuose (pav.: 23 a ir b).

a b

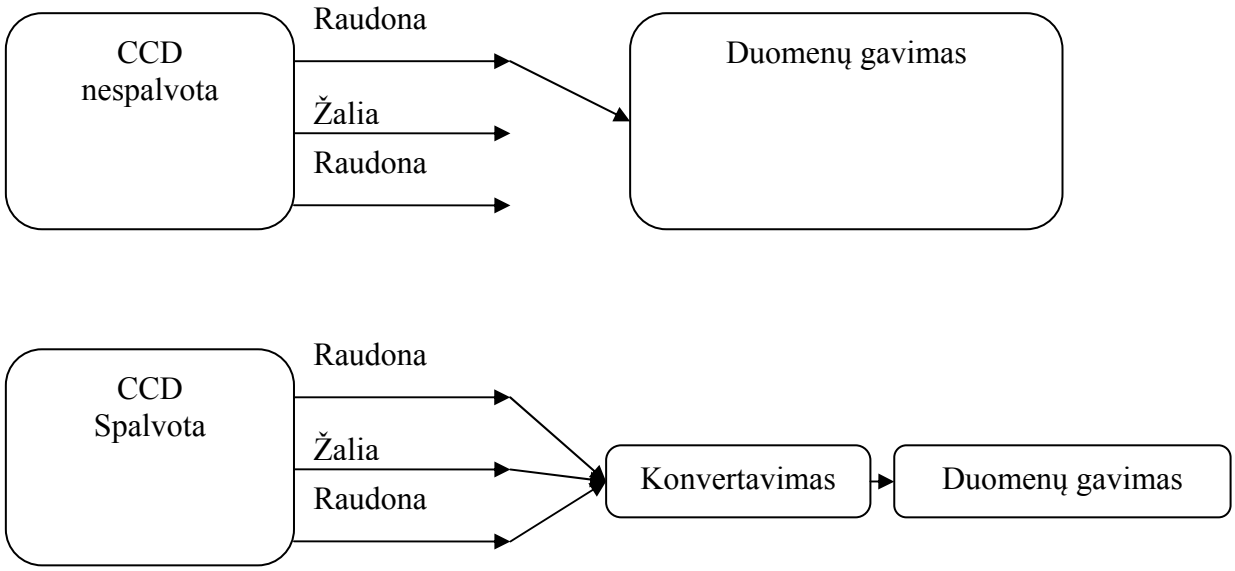
Pav. nr. 23

Kampas β yra reikalingas tik tuo atveju, jei norėsime susieti gautus išmatavimus su realiais, kitu atveju mes jį galime ignoruoti ir laikyti, kad mastelis mums visai nesvarbus (pav. 23 a ir b).

a b

Pav. nr. 24

Modeliavimui reikalingi specialiai paruošti duomenų failai. Jie turi būti surašyti tekstiniame formate. Duomenų failai turi saugoti nespalvotą vaizdą. Nespalvoto vaizdo gavimo algoritmas pateikimas pav. nr. 25



Pav. nr. 25 Spalvoto vaizdo konvertavimo į nespalvotą algoritmas.

Paveiksliuke 25 Pateiktas duomenų konvertavimo įrenginys skaičiuoja duomenis naudodamasis formulė (1) Naudojami koeficientai yra spalvos pikselių reikšmės, raudona, žaliai ir mėlyna, jos skaičiuojamas vidurkis, taip gaunamas nespalvotas vaizdas.

$$F_{duomemys}(R,G,B) = \frac{R+G+B}{3} \qquad \text{formulė (1)}$$

Pav. nr. 26 a b

Paveiksliuke 26 pateikiami konvertavimo iš spalvoto vaizdo į juodai baltą, a pateikiamas spalvotas, b juodai baltas.

2.8 Nefunkciniai reikalavimai

Užtikrinti tikslesniems rezultatams abiem atvejais reikalaujama, kad daiktas tiek slinkimo atveju, tiek sukimosi, būtų gerai prigludęs prie paviršiaus, tobuliausiu atveju netgi priklijuotas, ar prilipintas. Nesant tam net ir silpnos vibracijos įtakoja duomenų iškraipymą ir gaunami rezultatai nėra tikslūs. Mūsų kuriamas įrenginys kol kas nėra labai patogus. Įtakos tam turi ir tai, kad iki šiol dar nežinomas galutinio produkto dizainas. Kol kas įrenginiai naudoja pakankamai daug vietos atsižvelgiant į tai, kokio dydžio objektus skenuojame. Mat bent jau besisukančio apie ašį skenerio ilgis šio metu siekia apie metrą. Tuo tarpu skenuojami objektai yra tik apie ~7 cm^3 tūrio. Taigi logiška būtų , jeigu baigtas produktas užimtų ne daugiau ~20 cm^3 tūrio. Skenuojami objektai taip pat turi būti lazerį atspindinčios spalvos. Kol kas skenuojamų objektų greitis yra iki 1 min. Norint naudoti mūsų produktą, kompiuteris turi turėti grafinę plokštę su 128 MB atminties, pagrindinis procesorius turėtų būti „Pentium 4“ klasės arba geresnis, arba AMD „Athlon“ 2,2+ klasės procesorius arba geresnis, turėti nemažiau kaip 512 MB operatyviosios atminties, 40 GB kietasis diskas arba talpesni modeliai. Sistemos modeliu reikalaujama „Windows“ tipo operacinė sistema, rekomenduojama „Windows XP SP1“ arba naujesnės jos versijos.

Saugumo aspektu tai visai nėra pavojinga, vienintelis kiek nors kenkiantis įrenginys yra lazeris, dirbant su lazerinėmis sistemomis yra taikomos joms skirtos saugumo priemonės,

tai yra specialūs akiniai. Kontrolės reikalavimai yra tik tokie, kad pradžioje patartina gerai susireguliuoti kamerą ir lazerį. Mat pasisukimo atveju per sukimosi centrą būtinai turi eiti lazerio linija. Taip pat lazerio linija turi būti statmena staliukui.

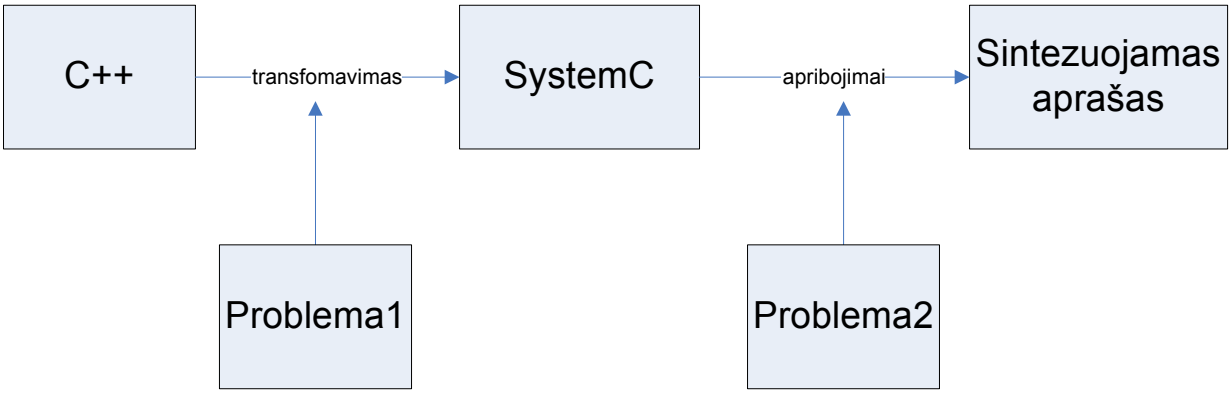
Sistemos modelis yra parašytas naudojantis c++ programavimo kalba. Šiuo metu jis yra pririštas prie Windows operacinės sistemos. Bet kodas yra universalus naudojantis bet kurios kitos operacinės sistemos, kurioje yra bent vienas c++ kompiliatorius. Sunkumai, kuriais galime susidurti yra perkeliamas vartotojo sąsajos ir naudojamos SDK (Programinės įrangos kūrimo įrankiu) perkėlimas į kitas operacines sistemas. Taip pat nerandant būdų arba galimybių perkeliant naudojamas SDK į kitas sistemas gali tekti naudotis kitomis egzistuojančiomis sistemomis, bet tai galbūt gali būti atlikti žymiai lengviau negu iš pirmo žvilgsnio atrodo.

Sistemai taikomi reikalavimai, kad ji veiktų ant Windows tipo sistemų, todėl šiuo metu nėra rūpinimąsi, kad sistema veiktų ir ant kitų operacinių sistemų.

SystemC modeliams reikalingas kompiuteris, kuris turi: „Pentium“ 4 tipo procesorių 512 MB darbinės atminties, 40 GB kietasis diskas, Windows arba Linux šeimos operacinę sistemą.

3 Projektinė dalis

3.1 Bendros perėjimo tarp C++ ir SystemC (TLM ir RTL) taisyklės



Pav. nr. 27 Perėjimo problemos

Perėjimas nuo C++ funkcinio modelio į *SystemC* modelį („Problema1") sprendžia Mindaugas. Modelio transformavimas į sintezuojamą modelį („Problema2") analizuoja Vytautas.

Bendros perėjimo tarp C++ ir *SystemC (TLM ir RTL)* taisyklės

Pradinėse specifikacijose mes turime funkcinį sistemos modelį, kurį turime transformuoti į sisteminį aprašą. Naudojantis transakcijų lygio *SystemC*, mes turime nuspręsti, kokia bus sistemos struktūra.

Pagrindinė problema yra pasirenkant, kuriuos kintamuosius kelsime, į kokius blokus t.y. naudojamus masyvus kelsime į atminties elementus, ar juos paliksime moduliuose ir naudosime kaip registrus.

Visi atminties elementai turi būti pririšti prie magistralių, todėl reikia pasirinkti:

- Kiek naudosime magistralių.
- Kiek sistema turės atminties.
- Kiek procesų sistemoje bus.
- Kokie bus procesai.
- Kaip mes juos valdysime.

RTL ir *TLM* aprašai skiriasi iš esmės. Todėl procesų valdymo sistema negali vienareikšmiškai sutapti su *RTL* aprašuose naudojamais metodais, bet patį principą kaip mes valdysime procesus, mes galime perkelti.

Keisdami *TLM* modelyje architektūrą mes sutaupome laiko, nes perkoduoti žemo lygio modelius užtrunka ilgiau, todėl modelių architektūrą mes turėtume keisti pradinėse projekto stadijose.

Funkcinis C++ aprašas	**TLM modelio aprašas**
#include <iostream> int kintamasis1 void funkcija1() { ….. } void funkcija2() { ….. } int main () { funkcija1(); funkcija2(); }	struct Initiator: sc_module, tlm::tlm_bw_b_transport_if{ tlm::tlm_b_initiator_socket<> socket; SC_CTOR(Initiator) : socket("socket"), dmi_ptr_valid(false) // DMI sąsaja { socket.bind(*this); SC_THREAD(process); } virtual void b_transport(tlm::tlm_generic_payload& trans) { tlm::tlm_command cmd = trans.get_command(); sc_dt::uint64 adr = trans.get_address() ; unsigned char* ptr = trans.get_data_ptr(); unsigned int len = trans.get_data_length(); bool* byt = trans.get_byte_enable_ptr(); unsigned int wid = trans.get_streaming_width(); .... } }

Lentele. nr. 4 C++ ir TLM kodo pavyzdžiai.

Transformuojant modelius skaičiavimo algoritmai nesikeičia, keičiasi su duomenų nuskaitymo įrašymu susijusios funkcijos („*wait*“, duomenų nuskaitymo įrašymui ir nuskaitymui naudojama *DMI* sąsaja).

Funkcinis C++ aprašas	TLM modelio aprašas
if(linija[i]!= 480){ Yp = linija[i]; if(i>1) DY = linija[i]- linija[i-1]; }	if(*(reinterpret_cast<int*>(&dmi_data.dmi_ptr[(k- >getMaxAdrees() + i) * 4]))!= 480){ Yp = *(reinterpret_cast<int*>(&dmi_data.dmi_ptr[(k- >getMaxAdrees() + i) * 4])); wait(dmi_data.read_latency); wait(dmi_data.read_latency); if(i>0) DY = *(reinterpret_cast<int*>(&dmi_data.dmi_ptr[(k- >getMaxAdrees() + i) * 4])) - *(reinterpret_cast<int*>(&dmi_data.dmi_ptr[(k- >getMaxAdrees() + i -1) * 4])); wait(dmi_data.read_latency); wait(dmi_data.read_latency); }

Lentele. nr. 5 *C++* ir *TLM* kodo pavyzdžiai.

Pagrindiniai skirtumai naudojantis *TLM*

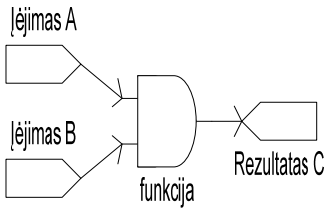
Funkcinis C++ aprašas	TLM modelio aprašas
int masyvas[10][10];	class Memory: sc_module, tlm::tlm_fw_b_transport_if<>{ public: DMI mode tlm::tlm_b_target_socket<> socket; enum { SIZE = 100 }; const sc_time LATENCY; int mem[SIZE]; SC_HAS_PROCESS(Memory); Memory(sc_core::sc_module_name name); ~Memory(); virtual void b_transport(tlm::tlm_generic_payload& trans); virtual bool get_direct_mem_ptr(const sc_dt::uint64& address, tlm::tlm_dmi_mode& dmi_mode, tlm::tlm_dmi& dmi_data); void invalidation_process();

	virtual unsigned int transport_dbg(tlm::tlm_debug_payload& dbg); };

Lentele. nr. 6 *C++* ir *TLM* atminties pavyzdžiai.

Pagrindinis skirtumas rašant paprastą algoritmą su *TLM* yra naudojami jau prie aparatūros pririšti moduliai. Šiuo atveju lentelėje nr. 6 pateikiamas paprasto masyvo transformavimas į TLM atminties modulį. Visas komunikavimas vyksta per tam skirtus modulius „*socket*“ arba per specifines sąsajas (DMI). Nepanaudodami šių modulių ir sąsajų, mes tiesiog modeliuojame paprastą modelį, kuris neduoda mums jokios informacijos ir yra nenaudingas tolimesniuose modeliavimo žingsniuose.

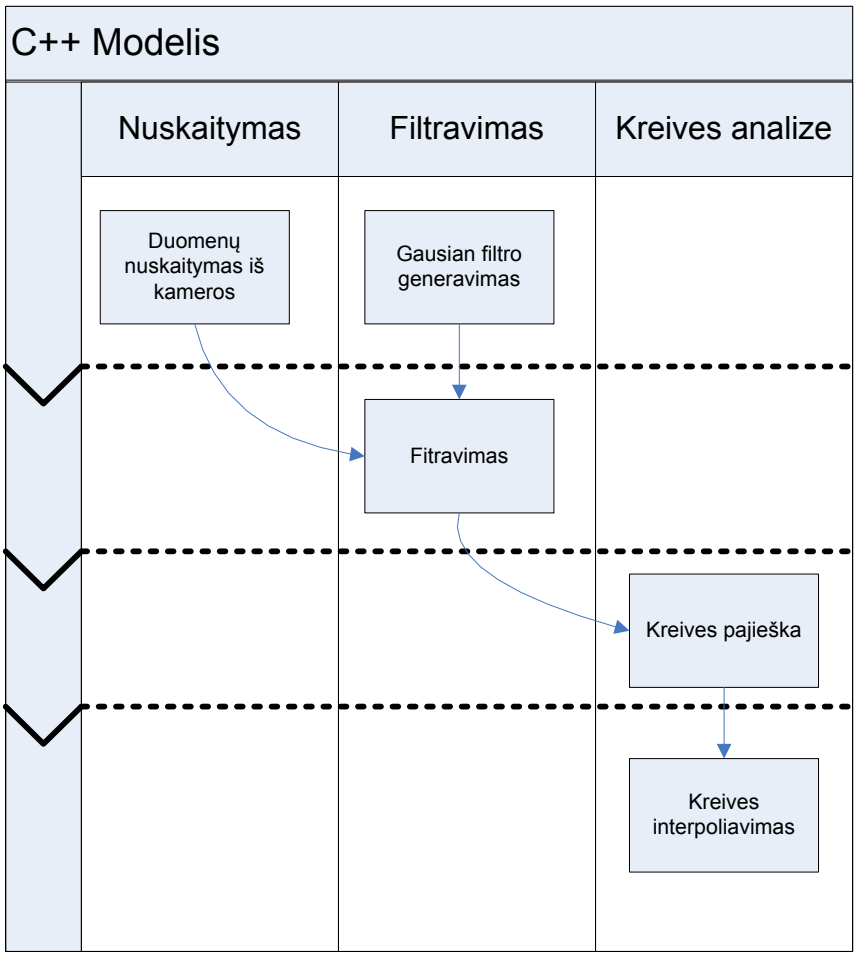
Apytikrio laiko (*approximately-timed*) modeliavimas atliekamas naudojantis „*wait*“ sakiniais. Šie sakiniai yra konfigūruojami pagal pasirinktą architektūrą. aprašoma [8]

	C++	***SystemC***
	bool paprastas_pavizdys (bool A, bool B) { bool C; C = A + B Return C; }	SC_MODULE(paprastas_pavizdys) { sc_in <bool> A; sc_in <bool> B; sc_out <bool> C; SC_CTOR (paprastas_pavizdys) { SC_MODULE(process); Sensitive << A<<B; } Void process() { C = A.read()+B.read(); }
C++ metodai (void, bool, int, float ir kiti) transformuojami į SystemC modulius C++ atsinešami duomenys (A ir B) atitinka SystemC modulio įėjimus (sc_in) C++ metodai būna 3 rūšių : funkciniai (grąžinantys vieną reikšmę (bool, in, float)) nefunkciniai negrąžinantys reikšmių		

C++ funkcinis metodas pereinant į SystemC tampa kaip išėjimu (sc_out)		
SC_MODULE apibendrina tam tikrą modulių SC_METHOD dalį turintį savus įėjimus ir išėjimus SystemC SC_METHOD yra panašu į SC_MODULE, bet negalintis savyje turėti kitų metodų.		
SC_CTOR - tai konstruktorius, kuriame sudedami metodai (SC_METHOD) gijos (SC_THREAD) ir procesai (SC_PROCESS) taip pat prie kiekvieno iš jų yra parašomas jautrumo sąrašas, kuris juos aktyvuoja.		
C++ sudėtingesniems algoritmams prieš transformuojant į SystemC reikia sukurti būsenų diagramas, kurios pilnai atitiktų sistemos funkcionavimą.		
Duomenų tipai	Float, double	Nepalaikomas dėl to reikia keisti į uint <n>
	Int	Int tipas palaikomas, bet rekomenduojama apsiskaičiuot maksimalią reikšmę ir naudoti uint<n> tipą
	bool	Palaikomas, bet jei tai įėjimas kartais geriau naudoti sc_in_clk

Lentele. Nr. 7

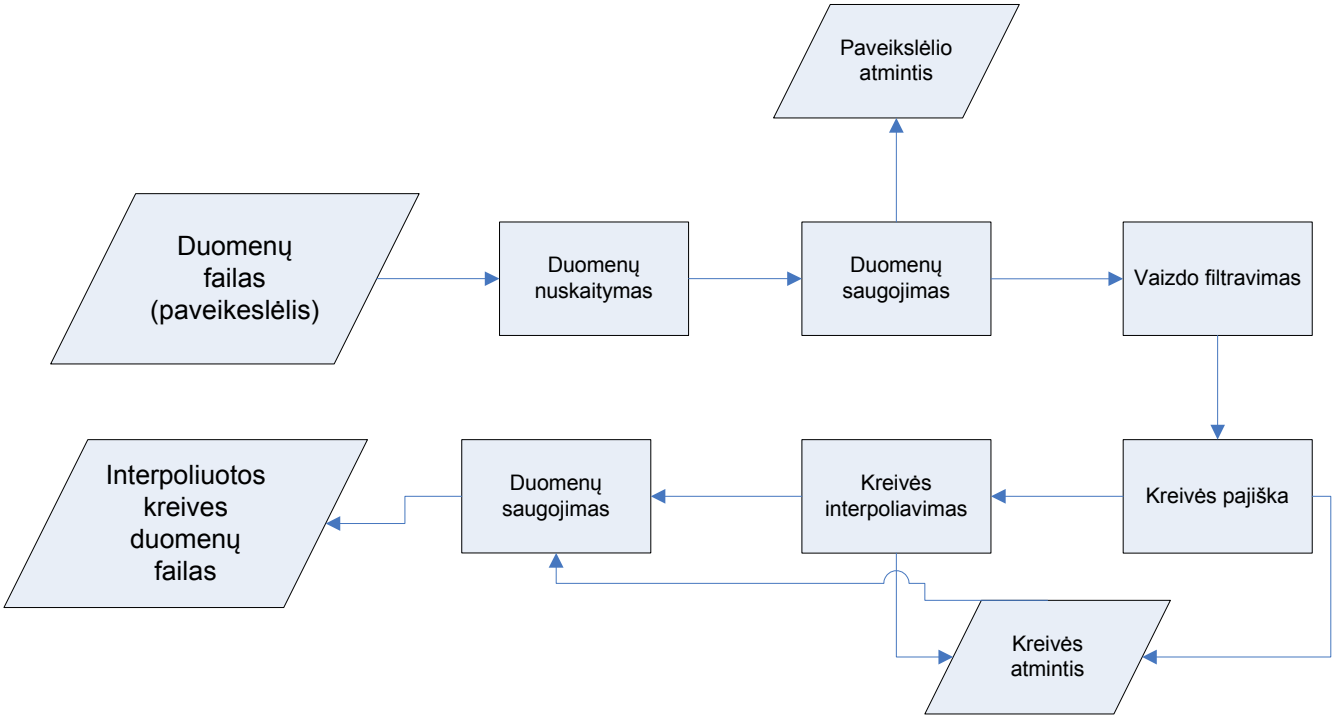
3.2 C++ modelio transformacija į sisteminį modelį



Pav. nr. 28 C++ modelio laiko diagrama.

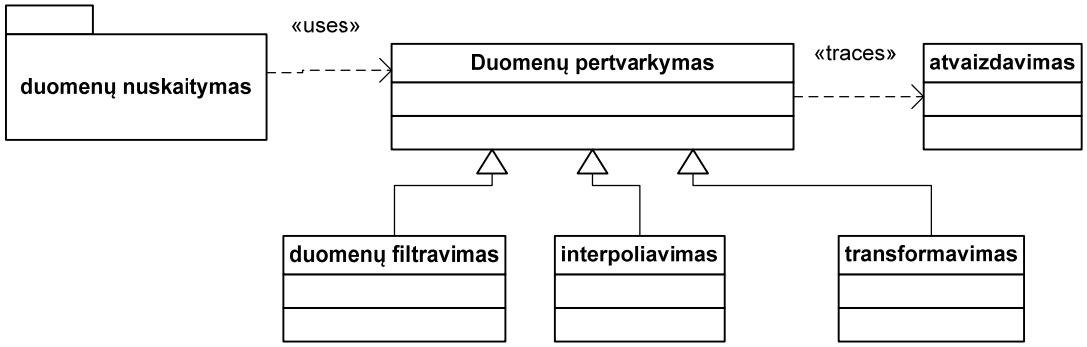
Analizuojama modelio laiko diagrama. Pradiniuose laiko momentuose vyksta du procesai t.y. „Duomenų nuskaitymas“ ir „Gausian“ filtravimo masyvo generavimas“. „Gausian“ filtras yra generuojamas tik vieną kartą, jei mes atliekame antrą iteraciją jo perskaičiuoti nereikia. Taip pat „Gausian“ filtras gali būti užkraunamas iš failo, tokiu atveju jo generuoti nereikia.

Kai baigiasi paveikslėlio užkrovimas, mes galime pradėti filtravimą. Tol kol filtruojame paveikslėlį, negali būti atliekami kiti veiksmai. Turint nufiltruotą paveikslėlį, galime atlikti kreivės paiešką. Kai gauname kreivę, mes ją interpoliuojame. Algoritmo pabaiga. Galime pradėti sekančią iteraciją.



Pav. nr. 29 Sistemos modelio duomenų srauto diagrama.

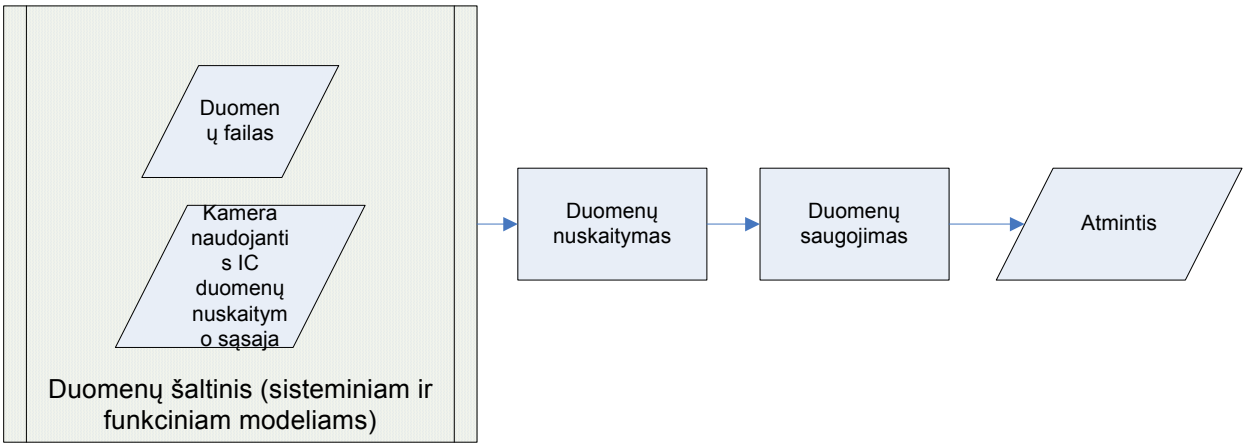
Analizuojama modelio duomenų srautų diagrama. Kad pradėtų veikti algoritmas, mums reikalinga turėti duomenis. Taigi mes nusiskaitome iš duomenų failo arba kameros. Toliau vyksta duomenų išsaugojimas atmintyje. Kai turime duomenis, galime juos analizuoti tai yra juos filtruoti. Turėdami nufiltruotus duomenis, mes pradedame kreivės paiešką. Gauta kreivę saugome atmintyje. Iš gautų kreivės duomenų, mes skaičiuojame interpoliuotą kreivę. Gautą kreivę mes išsaugome duomenų faile, kad vėliau galėtume vaizduoti ekrane.



Pav. nr. 30 Mišri diagrama (**Modelio klasių diagrama**) aprašoma [3].

3.3 Atskirų modelio dalių diagramos

Algoritmų aprašymai ir diagramos



Pav. nr. 31 Duomenų nuskaitymo duomenų srauto diagrama.

Duomenų nuskaitymo diagrama pateikia informaciją apie duomenų judėjimą nuskaitant duomenis. Priklausomai nuo modeliavimo stadijos (gali būti funkcinis modeliavimas arba sisteminis) mes pasirenkame duomenų šaltinį, iš kurio gausime duomenis. Kadangi naudojamos IC duomenų gavimo klasės yra palaikomos tik „*Windows*“ operacinėje sistemoje, tai modeliuojant sisteminį modelį ant „*Linux*“ sistemos, yra būtinas kitas duomenų įvedimo būdas.

Gauti duomenys yra išsaugomi atmintyje.

Filtravimas

Kaip pagrindinis filtravimas darbe yra naudojamas „Gausianinis“ filtras. Ref. {1} „Gausianinis ‘‘ filtras gali panaikinti triukšmą paveikslėlyje, taip gaunamas išlietas paveikslėlis. Tai yra pasiekiama manipuliuojant pikselių reikšmėmis naudojantis kauke. Pavyzdyje yra pateikiama kaukė naudojanti 7x7 dydžio matricą. Kaukės koeficientai yra apskaičiuojami naudojantis formule (2)

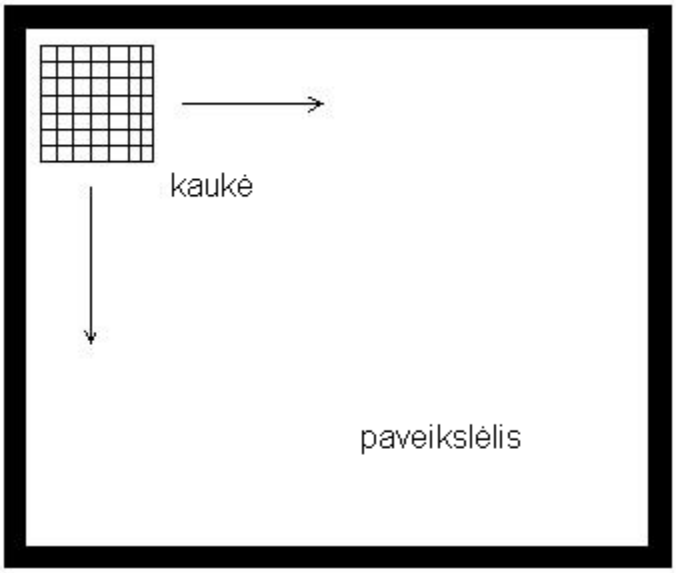
$$h(x,y) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{\frac{-(x^2+y^2)}{2 \cdot \sigma^2}}$$ formulė (2)

Formulėje h(x, y) nustato matricos reikšmes matricoje (kaukėje) . Sigma yra konstanta (pavyzdyje yra naudojama reikšmė lygi 1). Gauta matrica pateikiama paveikslėlyje 16 su normalizuotomis reikšmėmis iki 1.

y \ x	-3	-2	-1	0	1	2	3
3	0.000	0.001	0.006	0.011	0.006	0.001	0.000
2	0.001	0.018	0.082	0.135	0.082	0.018	0.001
1	0.006	0.082	0.367	0.606	0.367	0.082	0.006
0	0.011	0.135	0.606	1.000	0.606	0.135	0.011
-1	0.006	0.082	0.367	0.606	0.367	0.082	0.006
-2	0.001	0.018	0.082	0.135	0.082	0.018	0.001
-3	0.000	0.001	0.006	0.017	0.006	0.001	0.000

Pav. nr. 32

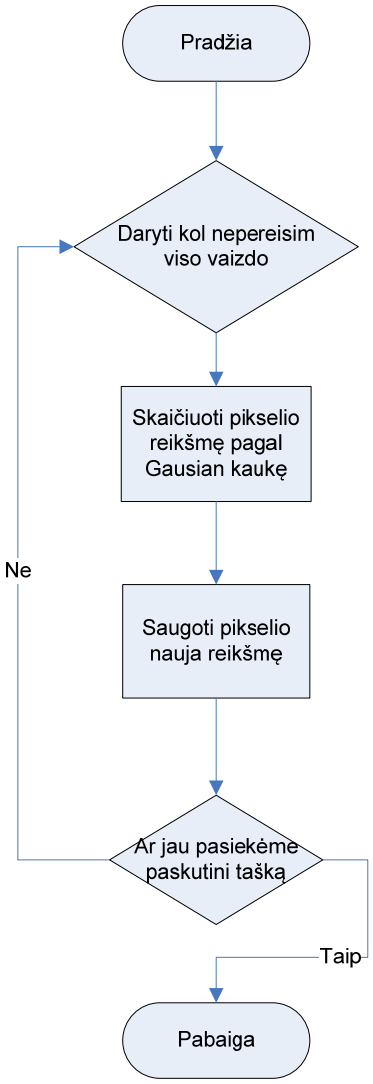
Paveikslėlyje naudojama matrica yra naudojama sugeneruoti naują paveiksliuką. Ant kiekvieno paveikslėlio taško arba pikselio yra uždedama išcentruota matrica tai yra x= 0, ir y = 0. Visi 48 pikseliai, kurie patenka į matricos apglėbiamą erdvę yra sudauginami su matricos koeficientais ir visos reikšmės yra sudedamos. Gauta reikšmė yra vidurkinama ir gaunamas naujo paveiksliuko pikselis. Viso aprašyto proceso iliustracija yra pateikiama 17 paveikslėlyje.



Pav.nr. 34.

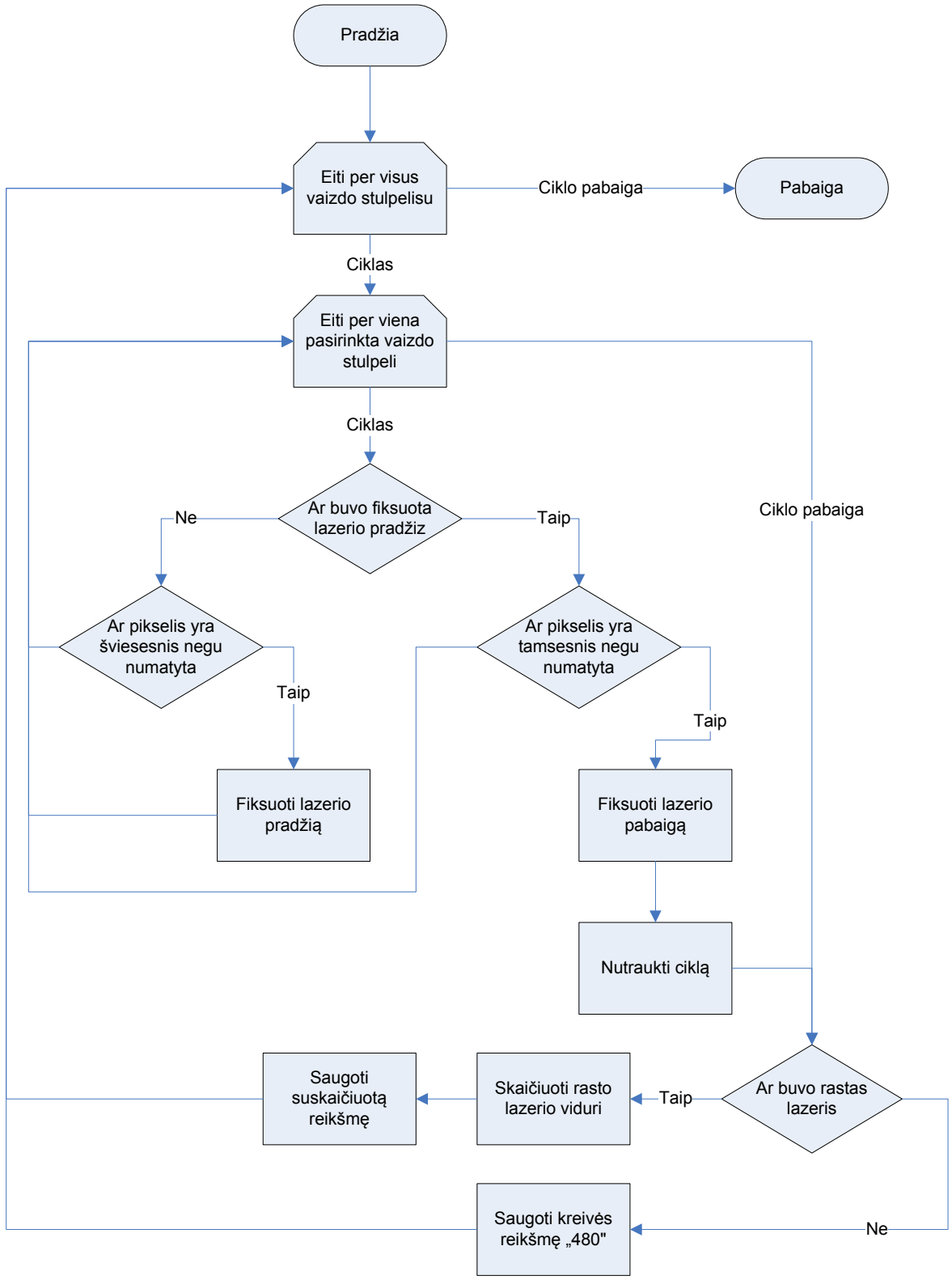
Gauti rezultatai pateikiami paveikslėlyje 35.

Pav. nr. 35 a b prieš filtravimą b) po filtravimo.



Pav. nr. 36 Filtravimo algoritmo loginė diagrama.

Kreivės radimas.

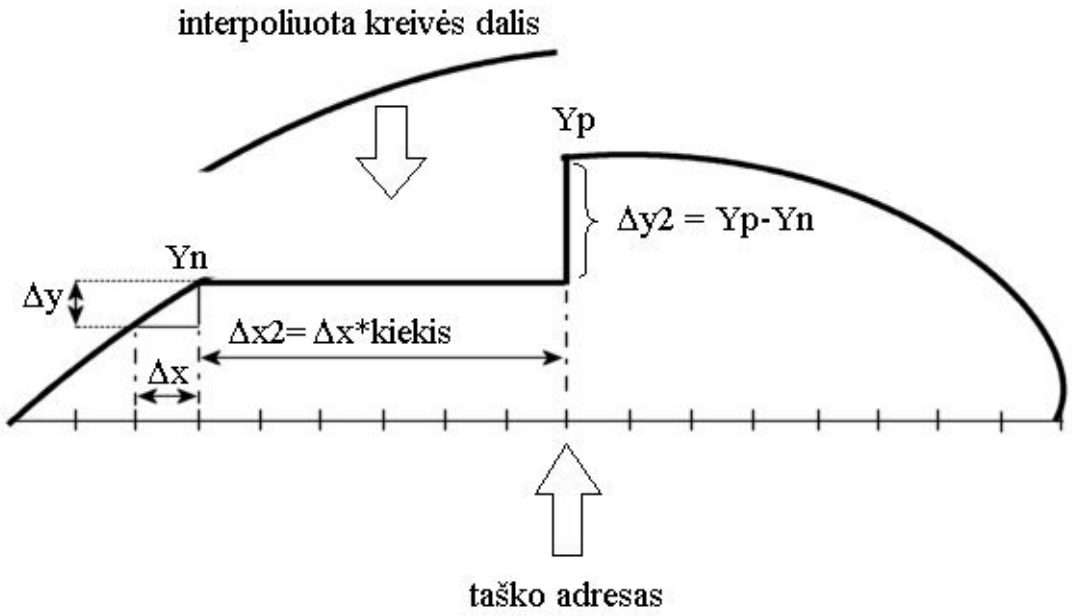


Pav. nr. 37 Kreivės radimo algoritmas.

Kreivės radimo algoritmas yra pateiktas naudojantis blokinėmis loginėmis schemomis.

Algoritmas analizuoja vaizdo stulpelius. Eidamas per vaizdo stulpelius jis ieško lazerio pradžios taško. Radęs pradžios tašką jį išsisaugo atmintyje. Po pirmo taško suradimo yra ieškomas lazerio pabaigos taškas. Jei buvo nerastas lazerio pradžios taškas ir mes perėjome per visą stulpelį, tai į atmintį įrašomas paveikslėlio paskutinio taško indeksas t.y. įrašoma reikšme lygi 480. Jei buvo rastas lazeris, tai į atmintį įrašomas lazerio vidurio taškas. Jis apskaičiuojamas atimant lazerio pabaigos taško indeksą iš lazerio pradžios indekso, tada gauname lazerio plotį. Gautą lazerio plotį padaliname iš dviejų ir gauta reikšmė atitinka pusei lazerio pločio. Pridedant pusę lazerio pločio prie lazerio pradžios indekso gauname indeksą, kuris parodo, kuriame taške yra lazerio vidurys. Taip einame per visus paveikslėlio stulpelius, kol pasiekiame paveikslėlio pabaigą.

Interpoliavimo algoritmo aiškinimo schema



Pav.nr 38

skaičiavimo formulė sudaryta naudojant pav.38, aprašoma [1]

kintamųjų aprašymas

Δx -x pokytis tarp 2 taškų (mūsų atveju visada pastovus dydis)

Δy -y pokytis tarp dviejų taškų

$\Delta x2$ - x pokytis, kol nėra reikšmių

$\Delta y2$ - y pokytis, kol nėra reikšmių

kiekis- skaičius taškų, kurie neturi reikšmės

k1,k2 - krypties koeficientai

naudojamos formulės

$$k1 = \frac{\Delta x}{\Delta y} \quad (1)$$

$$k2 = \frac{\Delta x2}{\Delta y2} \quad (2)$$

$$\Delta y_2 = y_n - y_p \quad (3)$$

$$kx = \frac{k1 - k2}{2} \quad (4)$$

$$\Delta y_n = \Delta y_2 + kx \quad (5)$$

$$y_{p+1} = y_p + \Delta y_n \quad (6)$$

Algoritmo veikimas

Algoritmo idėja tame, kad kol nėra reikšmių, paimamas prieš tai buvęs krypties koeficientas (pagal 1 formulę) ir naujas krypties koeficientas, kuris gaunamas tarp paskutinio turimo duomens (y_p) ir pirmo vėl esamo (y_n) (kaip parodyta 2 formulėje).

Šiuo atveju $\Delta x2$ galime apskaičiuoti įsivedant papildomą kintamąjį (kiekis), kuris nusako kiek taškų trūksta, tai apskaičiuojama pagal 3 formulę.

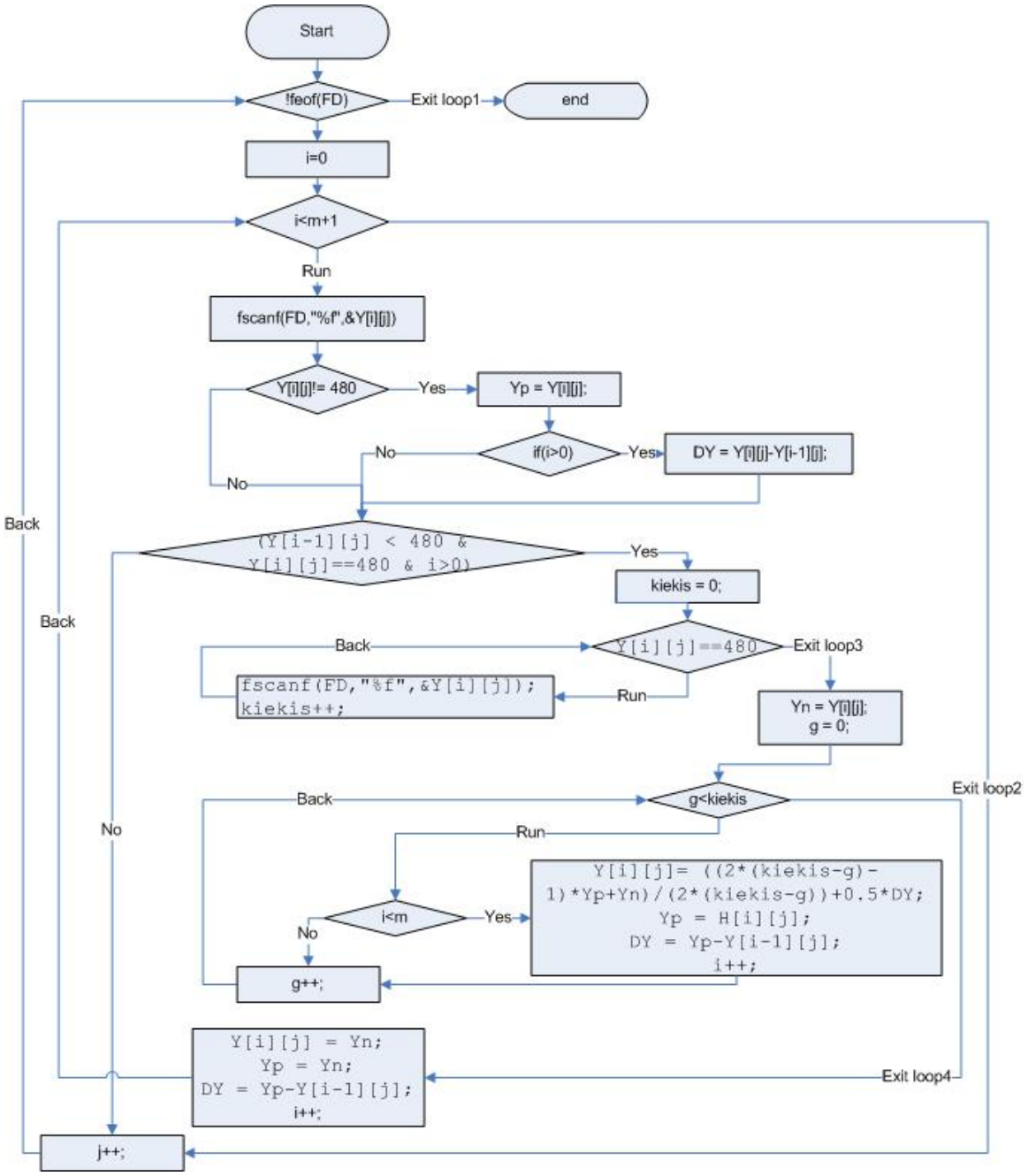
Skaičiavimo esmė tokia: skaičiuojame naują krypties koeficientą kiekvienam naujam taškui, kuris apskaičiuojamas pagal 4 formulę.

Toliau skaičiuojame naują y pokytį, kuris apskaičiuojamas pagal 5 formulę, taigi nauja taško reikšmė gaunama iš 6 formulės.

Suoptimizavus, matematinė išraiška gaunama tokia:

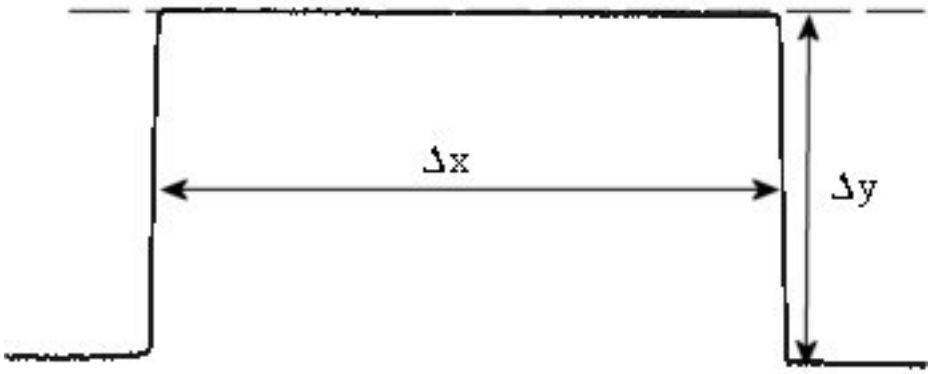
$$y[i][j] = \frac{(2 \cdot kiekis - 1) \cdot y_p + y_n}{2 \cdot kiekis} + 0.5 \cdot \Delta y \quad (7)$$

toliau matyti pav.39 algoritmas

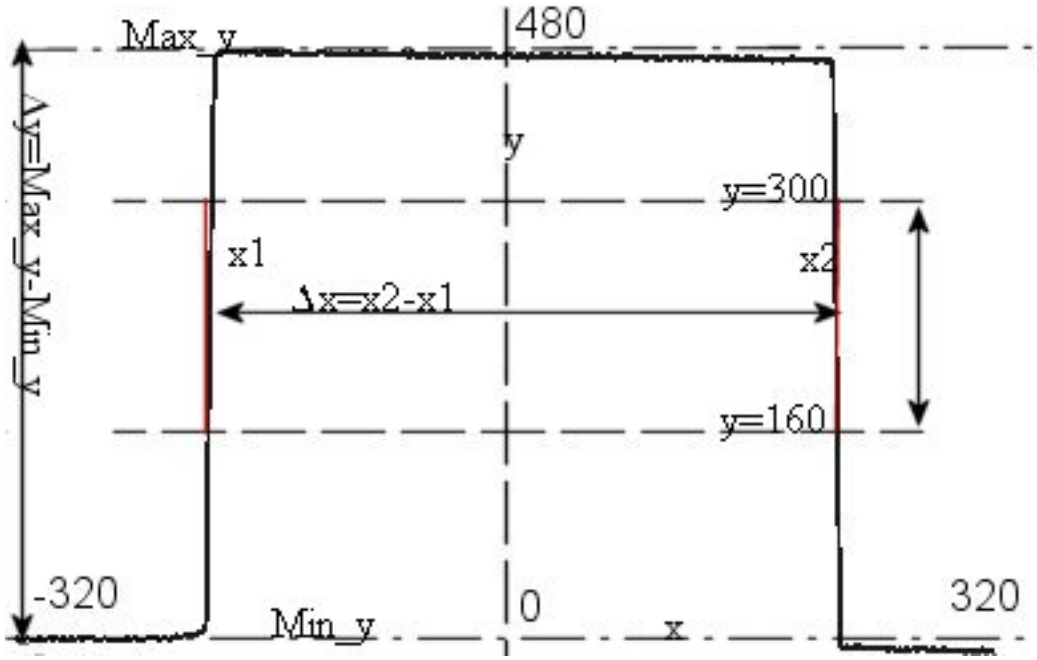


Pav. nr. 39 Interpoliacijos algoritmo loginė schema C++ pradinės specifikacijos modelis

Transformavimo algoritmo aiškinimo schema



Pav. Nr. 40 gautas vaizdas tiesiai i kamerą



Pav.Nr. 41 transformavimo algoritmo schema

Transformavimo koeficiento skaičiavimui naudojame kubą.

Reikalingas tik vienas taisyklingas kubo pjūvis, kuris mums bus reikalingas.

Taip gauname pjūvį, kuris yra transformuotas dėl kameros pozicijos (pav.x1).

Mūsų uždavinys gauti (pav.x2) parodytą vaizdą ir suskaičiuot koeficientą, kuris įvertintų kameros kampą.

Norėdami apskaičiuot šį koeficientą, turime gauti kai kuriuos duomenis: minimalią (Min_y) ir maksimalią (Max_y) y reikšmę ir bei x1 ir x2 reikšmę, kurios apsprendžia kubo šoninių sienelių poziciją X ašyje.

Žinodami tikrą nuskenuoto daikto formą, galime atkurti jį. Šiuo atveju skenuojamas kubas ir dėl to turėtumėm gauti kvadratą. Nuskenavę gauname stačiakampį gretasienį, kuris turi būti transformuotas į kubą. Skaičiuojame Δx ir Δy pokyčius, kurie nusako kraštinių ilgius. Kvadrato atveju $\Delta x = \Delta y$, taigi jei ši sąlyga yra netenkinama reikia skaičiuot nukrypimo koeficientą k.

Jam apskaičiuot naudojamos formulės

$$\Delta x = x2 - x1; \qquad (8)$$

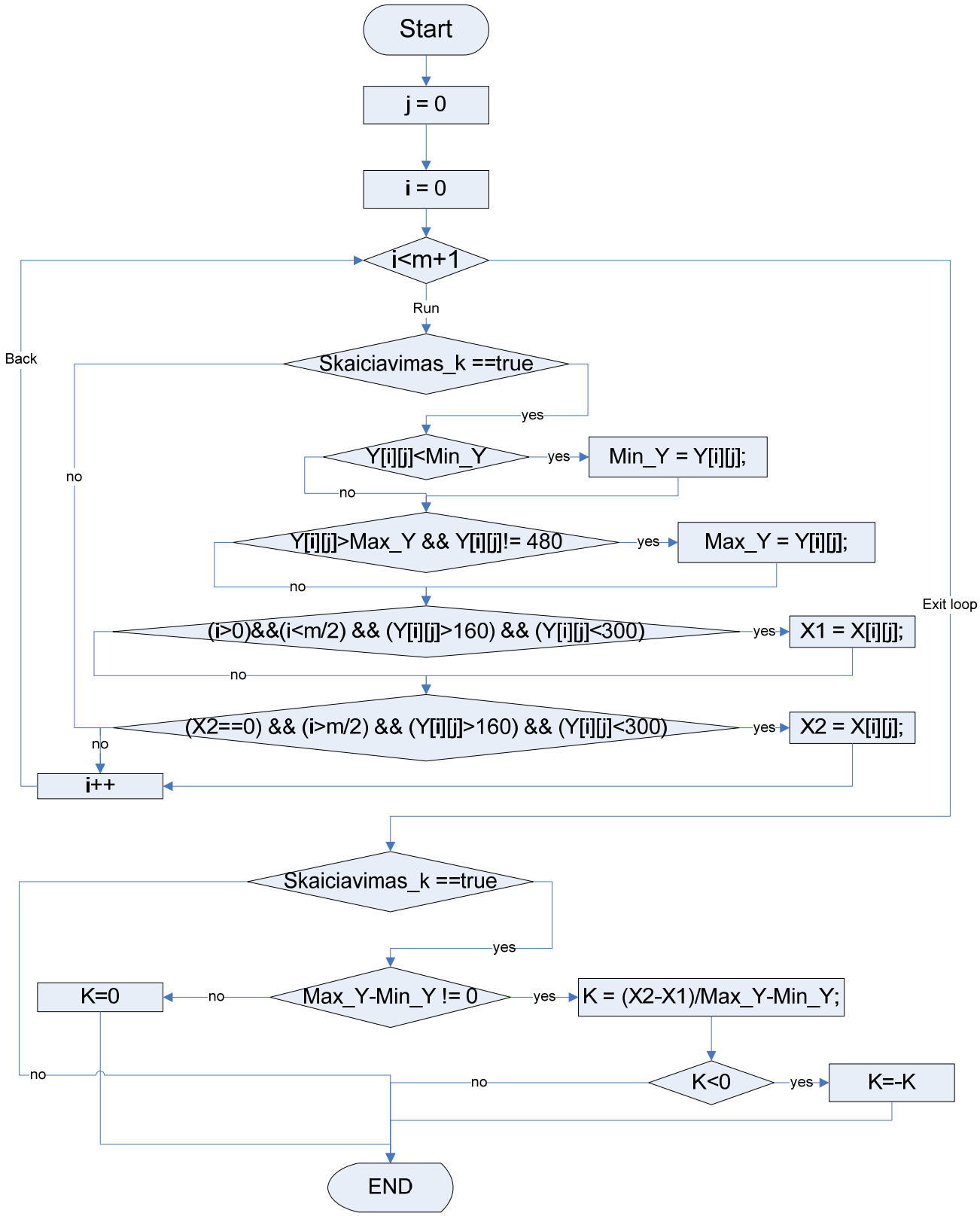
$$\Delta y = Max\_y - Min\_y; \qquad (9)$$

$$k = \frac{\Delta x}{\Delta y} = \frac{x2 - x1}{Max\_y - Min\_y}; \qquad (10)$$

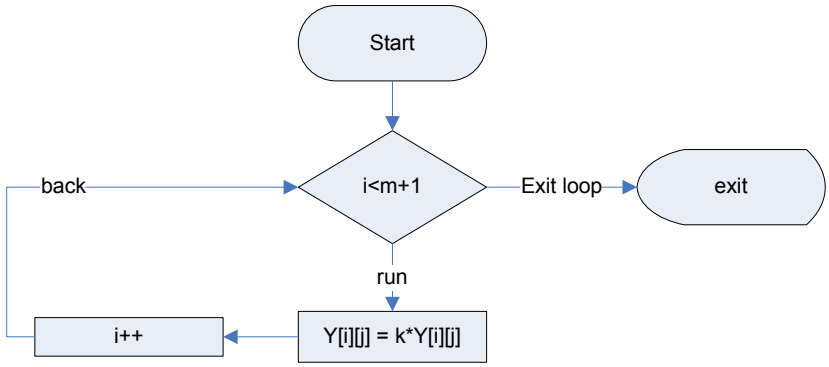
Turint koeficientą k vėliau reikia transformuoti visas Y koordinatės reikšmes.

Tai padaroma esamą reikšmę dauginant iš nukrypimo koeficiento.

$y_{[i][j]} = k * y_{[i][j]};$

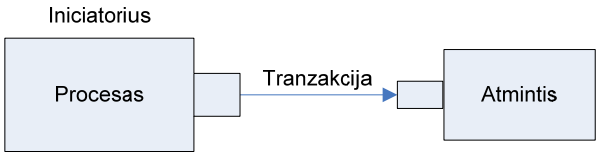


Pav. nr. 42 transformavimo koeficiento skaičiavimo algoritmas

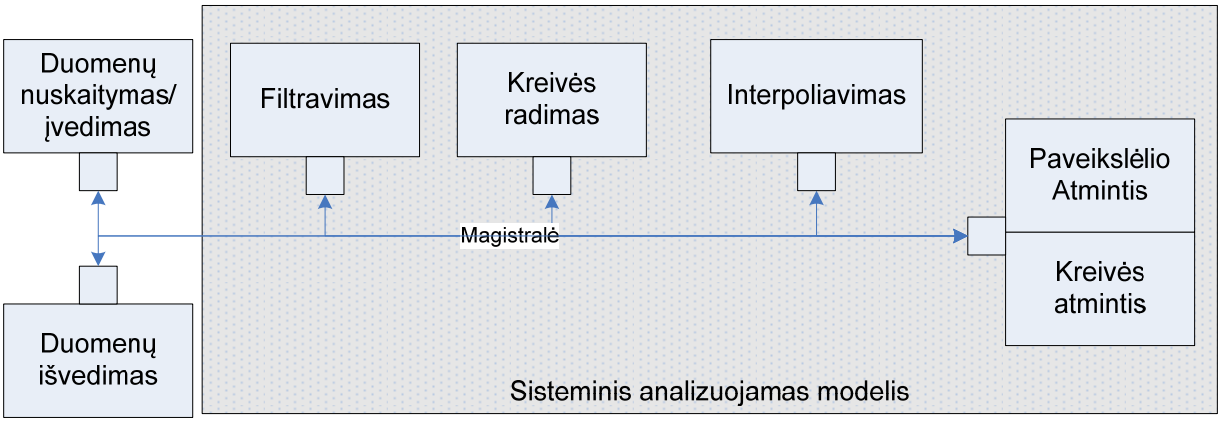


Pav. nr. 43 transformacijos algoritmas

Sisteminiai modeliai



Pav. nr. 44 TLM sistemos s blokinė schema

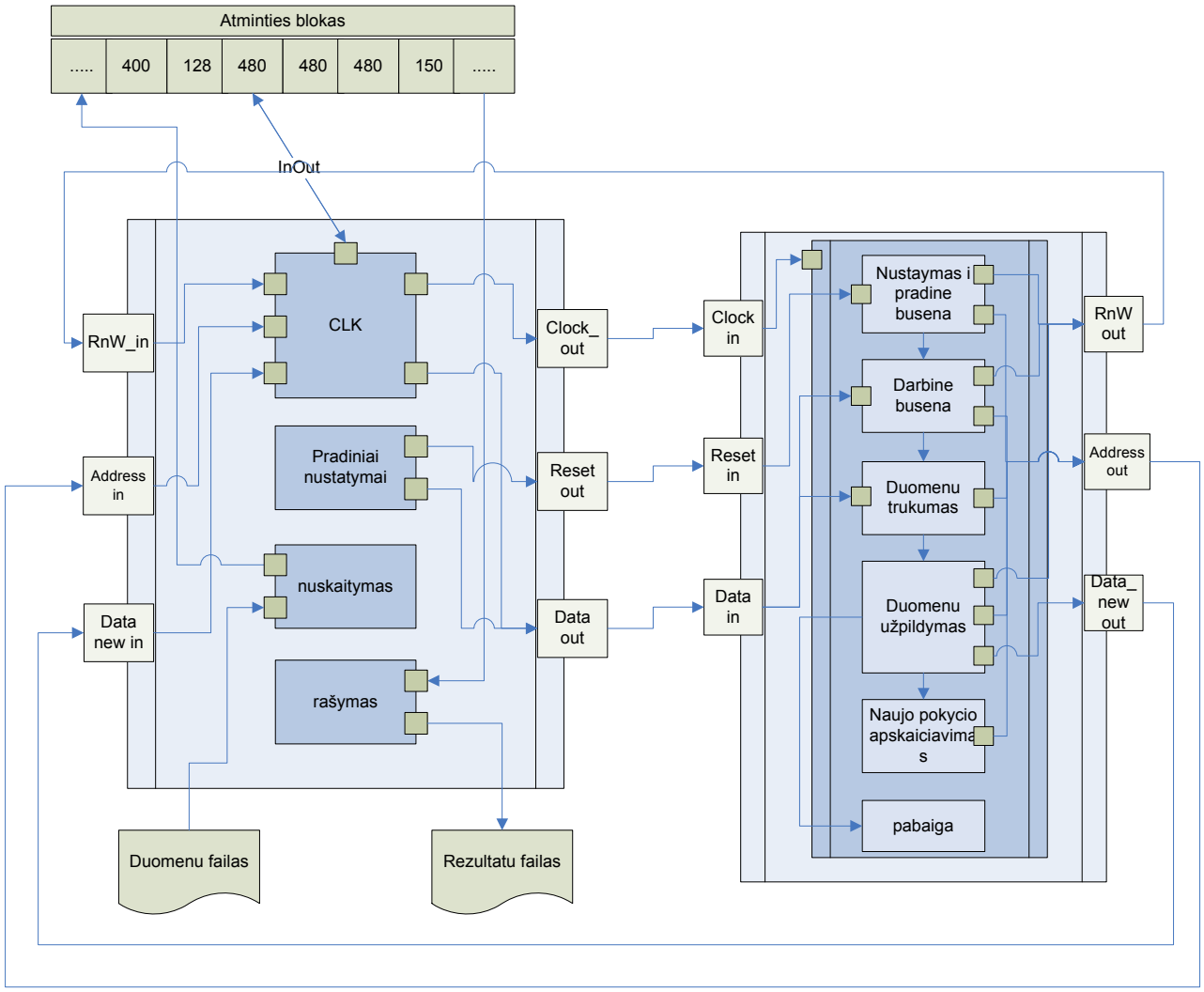


Pav. nr. 45 *TLM* modelio procesų diagrama.

Pateikiamas *TLM* sisteminis modelis. Filtravimas, kreivės radimas, interpoliavimas yra prijungti prie sistemos magistralių. Visi anksčiau išvardinti moduliai ir atminties blokai sudaro sistemą. Duomenų įvedimas ir nuskaitymas yra pagalbiniai blokai, kurie naudojami testuojant sistemą. Duomenų išvedimas naudojamas duomenų išvedimui į failą. Filtravimo, kreives radimo bei interpoliacijos algoritmai yra buvo pateikti anksčiau ataskaitoje. Naudojamas duomenų blokas yra vientisas ilgas masyvas, kuris yra perskaičiuojamas naudojant pagalbinę klasę. Paveikslėlis yra saugomas nuo nulinio iki 307200 elemento (t.y.

paveikslėlio dydis 640 padaugintas iš 480 lygu 307200). Nuo 307200 iki 307840 elemento yra saugomi kreivės duomenys.

Paveikslėlio matricos indekso transformavimas į masyvo indeksą atliekamas pagal formulę. Masyvo indeksas lygu stulpelis padaugintas iš eilutes maksimalios reikšmės ir pridėta eilutė.



Pav. Nr. 46 *RTL* Procesų diagrama

SystemC RTL aprašas susideda iš *bandymo stendo(testbech)* ir interpoliacijos algoritmo modelių. Pav.x parodyta procesų bendravimo schema. *Bandymo stendas* ir interpoliacijos algoritmas vyksta lygiagrečiai.

Bandymo stendas pradedamas vykdyti nuo pradinių nustatymų. Pradinių nustatymų metode yra įrašomas *reset signalas,* taip pat ten gali būti įrašoma nauja duomenų reikšmė rankiniu būdu. Toliau seka duomenų nuskaitoma iš failo. Nuskaitomi duomenys yra surašomi į

atminties bloką. Iš atminties bloko galima nuskaityti ir pakeisti saugomus duomenis. Šiuo atveju tai atlieka *CLK* procesas. Į *CLK* procesą ateina signalai:

RnW -(skaitymas =1 ar rašymas =0)

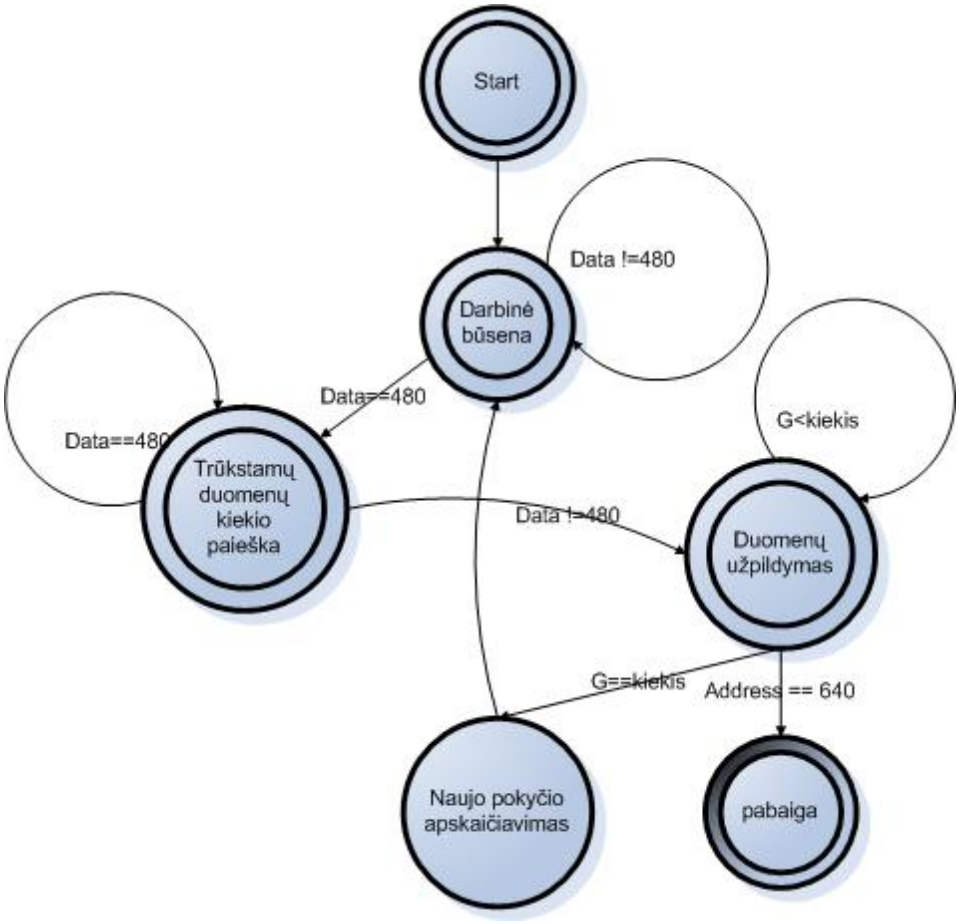
Address – adresas arba indeksas į atitinkamą atminties vietą

Data_new – nauja duomenų reikšmė, kuri turi būti įrašyta į atmintį

Clk procesas taip pat generuoja sinchronizacijos signalą.

Rašymas į failą vyksta po to, kai interpoliacijos algoritmas baigia darbą.

Interpoliacijos algoritmą *RTL* lygyje geriausiai galima įsivaizduoti kaip būsenų automatą. Tam reikia gana gerai išmanyti patį algoritmą ir sugebėti jį sudalinti į atskiras būsenas. Šiuo atveju gautas automatas atrodo taip kaip parodyta pav. 47



Pav. Nr. 47 *RTL* automato būsenų diagrama

- Automatas pradeda veikti, kai *reset* reikšmė lygi 0.
- darbinėje būsenoje automatas tikrina, ar atėję duomenys yra teisingi, ir jei jie teisingi, tada skaičiuojamas pokytis tarp esamos reikšmės ir prieš tai buvusios.

- Atėjus duomenų reikšmei 480 automatas peršoka į kitą būseną ir skaičiuoja kiekį trūkstamų duomenų. Suskaičiavus visus trūkstamus duomenis automatas pereina į duomenų užpildymo būseną.
- Duomenų užpildymo būsenoje yra perskaičiuojamos trūkstamos reikšmės pagal sugalvotą interpoliacijos algoritmą. Kai visos trūkstamos duomenų reikšmės yra užpildytos, automatas peršoka į naujo pokyčio apskaičiavimo būseną. Tačiau jei adreso skiltis pasiekia 640 reikšmę, automatas baigia darbą ir peršoka į pabaigos būseną.
- Naujo pokyčio apskaičiavimo būsena yra skaičiuojama tik vieną kartą, todėl po jos iškart seka darbinė būsena.
- Pabaigos būsenoje automatas nieko nedaro tik išlaiko esamą būseną

4.Eksperimentinė dalis

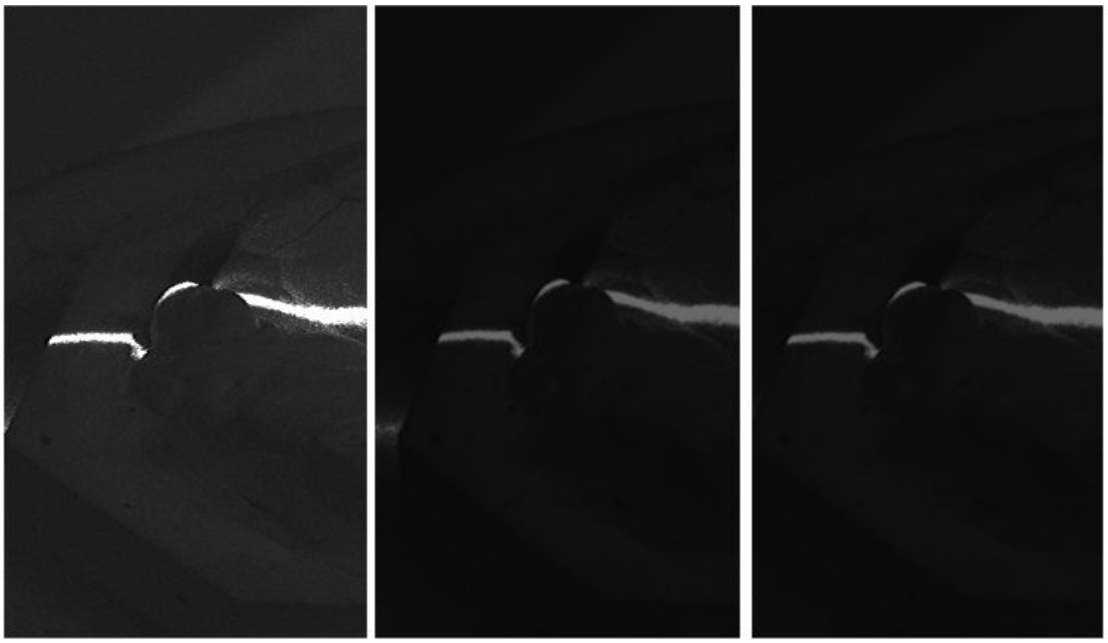
4.1 Eksperimentų rezultatai

Šiame skyrelyje yra pateikti:

- Testinio paveikslėlio filtravimo rezultatai
- Kreivės radimo iš testinio paveikslėlio rezultatai
- Kreivės interpoliacijos rezultatai

Šių rezultatų gavimui buvo panaudota modeliuojama pradine specifikacija C++ kalboje.

4.1.1 Testinio vaizdo filtravimo rezultatai



a)Nefiltruotas paveikslėlis b) C++ filtruotas pav. c) TLM modelio filtruotas pav.

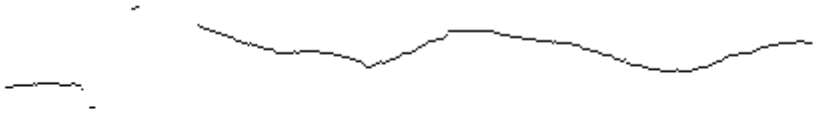
Pav. nr. 48 paveikslėlio filtravimo rezultatai.

Paveikslėlyje numeris 48 yra pateikiama duomenų pavyzdžiai. Pav. 48 a) yra nefiltruotas vaizdas gautas iš skaitmeninės kameros (*CCD* kameros), kaip matome b ir c paveikslėliai yra identiški.

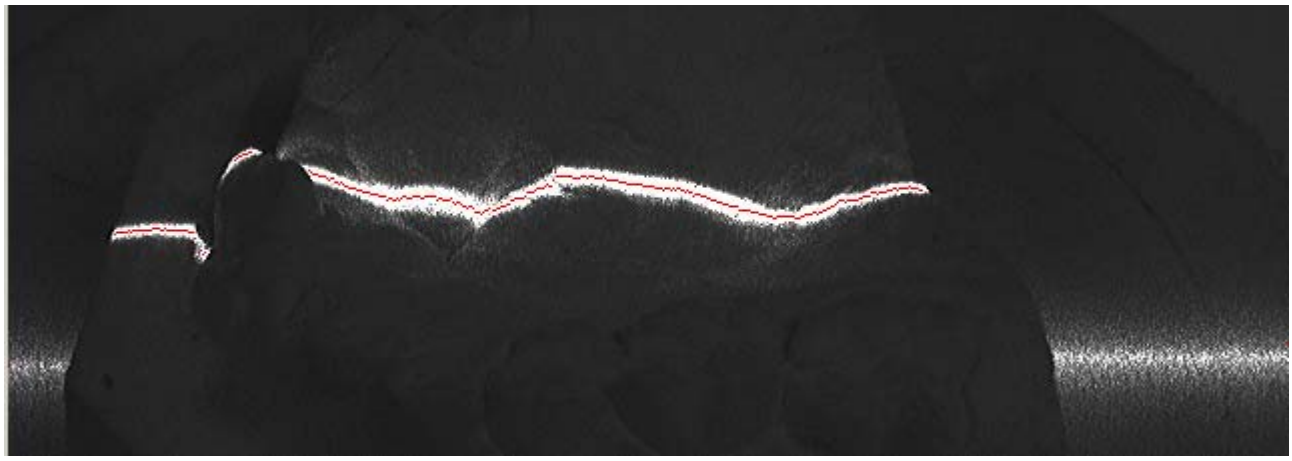
4.1.2 Kreivės radimo algoritmo rezultatai

Mes kreivę ieškome iš paveikslėlio, taigi pradiniai duomenys yra paveikslėlis pateikiamas prieduose.

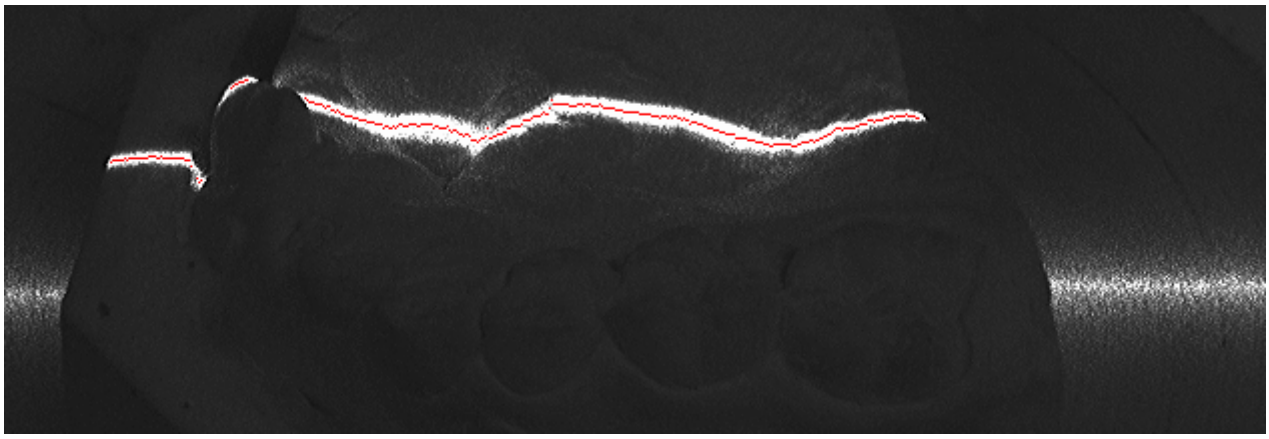
a)



b)



c)

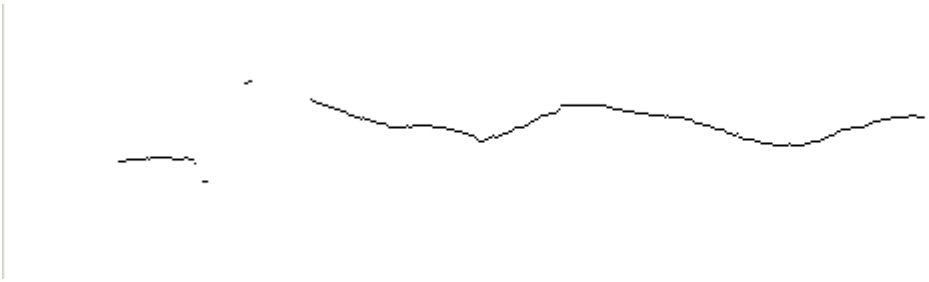


Pav. nr. 49 Gauta kreivė, a) kreivė gauta iš C++ modelio, b) C++ modelio kreivė pavaizduota ant paveiksliuko, c) kreive gauta iš *TLM* modelio (pavaizduota ant paveikslėlio).

4.1.3 Interpoliacijos algoritmo rezultatai

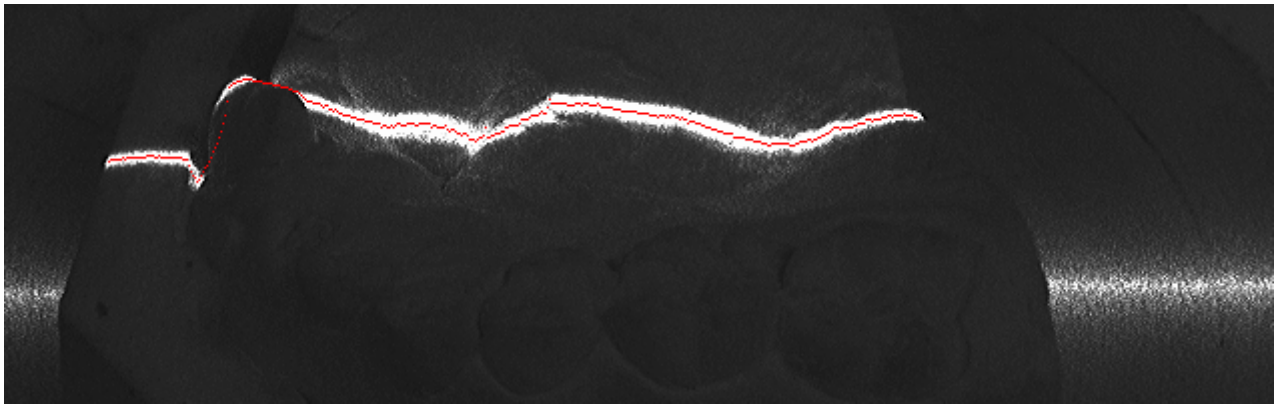
Pradiniai duomenys

a)

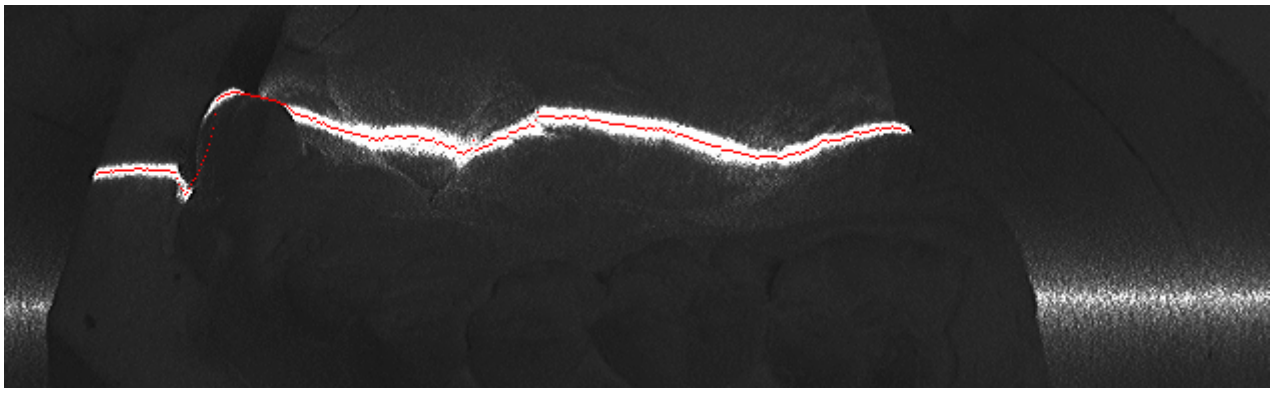


b)

c)



d)

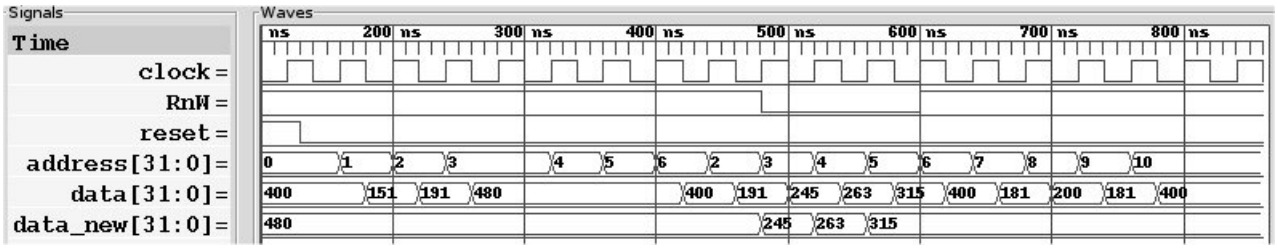


Pav.nr. 50 a) pradiniai duomenys, b) C++ modelio rezultatai, c) C++ modelio rezultatai perkelti ant paveiksliuko, d) *TLM* modelio rezultatai pavaizduoti ant paveiksliuko.

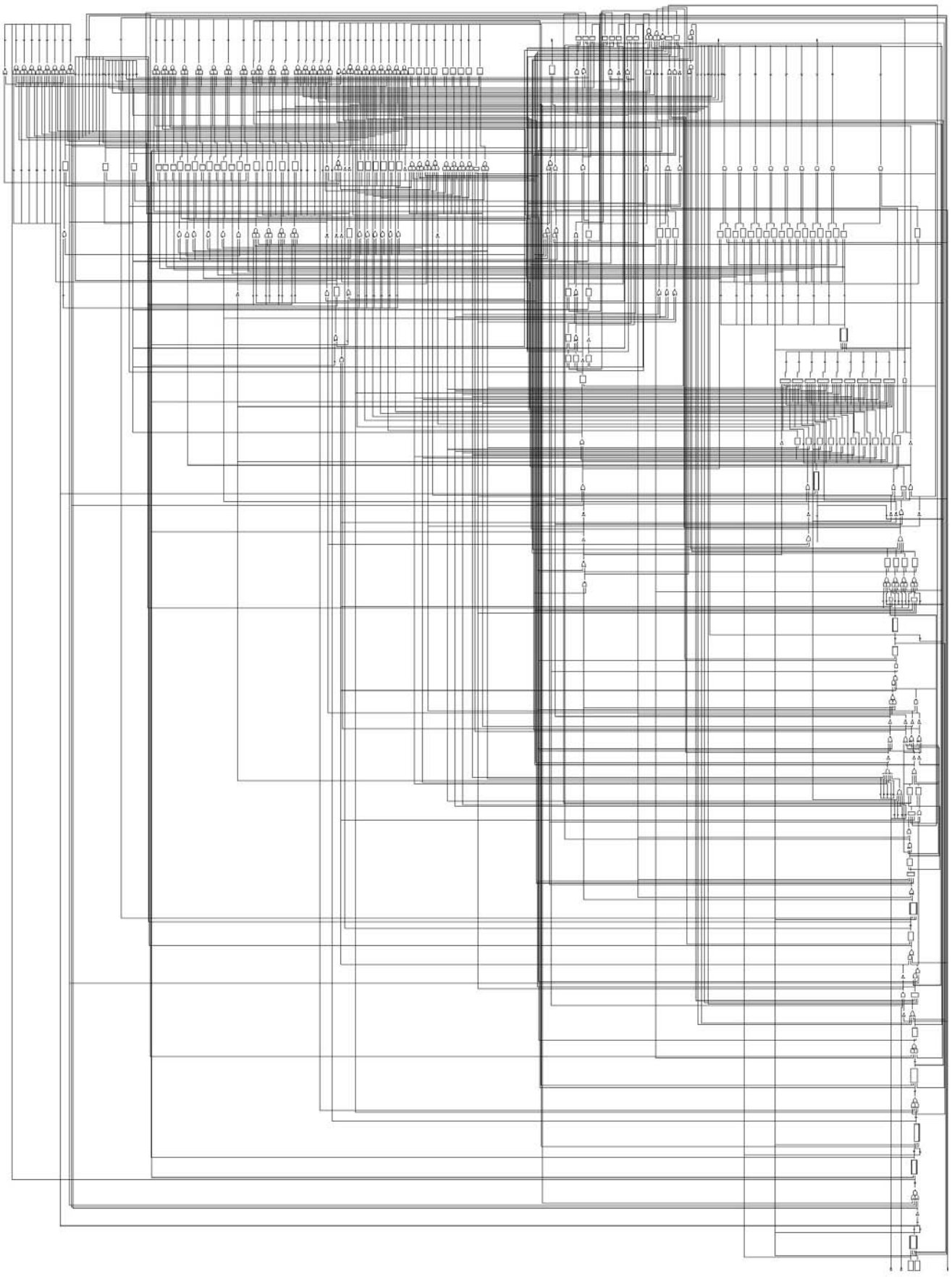
4.1.4 Sintezės rezultatai

	class	Cba_core	GSCLib
Įėjimų skaičius	31	31	31
Tinklų skaičius	890	350	761
Ventilių skaičius	870	281	666
Skirtingų ventilių skaičius	30	53	24
Bendras ventilių plotas	1741.000000	1390.969971	
Apjungtas plotas	1351.000000	1168.45786	
Neapjungtas plotas	390.000000	222.509933	
energija	96.2680 uW	41.6679 mW	26.6964 mW
Trigerių skaičius	77	77	77
Duomenų atvykimo laikas	0,97 r	0,86 r	0,21 r

Lentelė nr.**8**

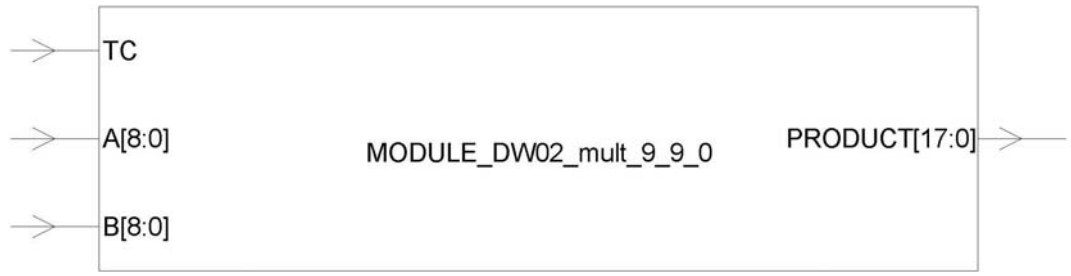


Pav. nr. 51 Modeliavimo rezultatai *RTL*

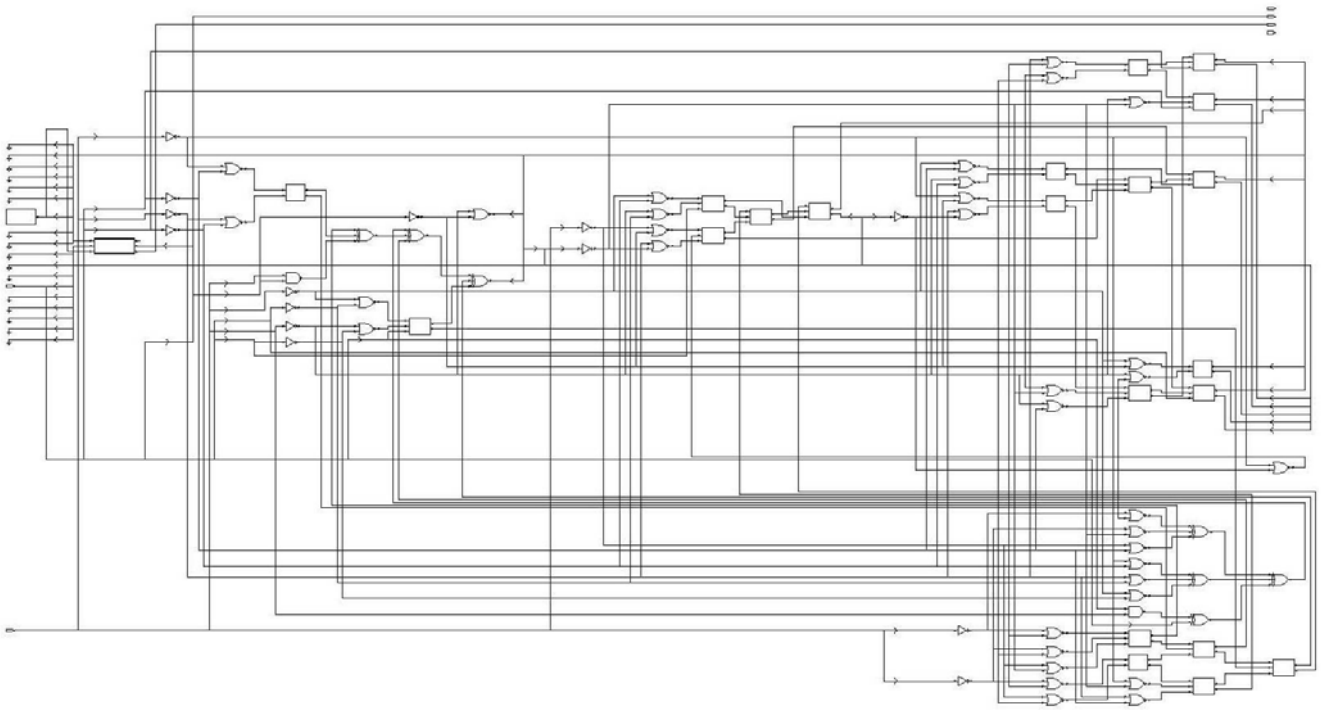


Pav. nr. 52 sintezuotas *SystemC* interpoliacijos algoritmas

Sintezuoto modelio atskirų blokų dalys

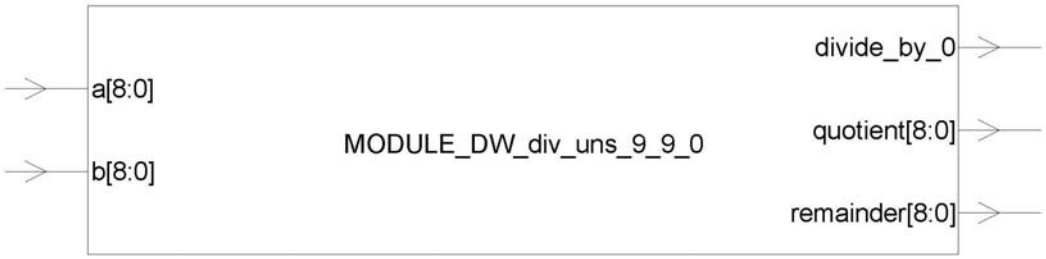


a) Schemos simbolis

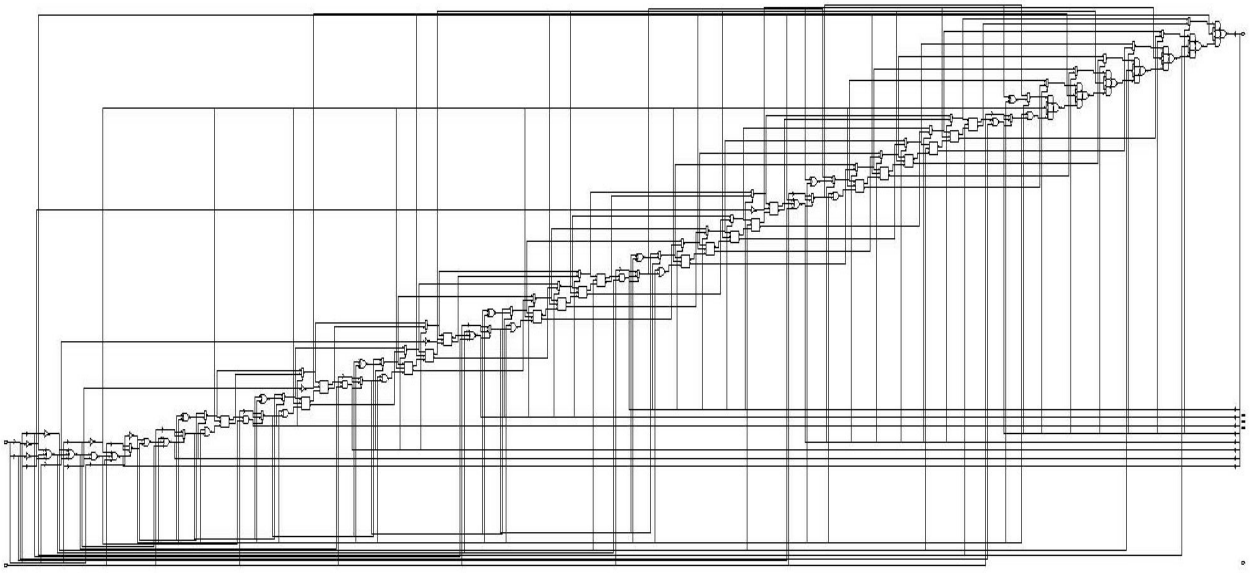


b) Sintezuota schema

Pav. nr. 53 sintezuoto modelio atskira dalis

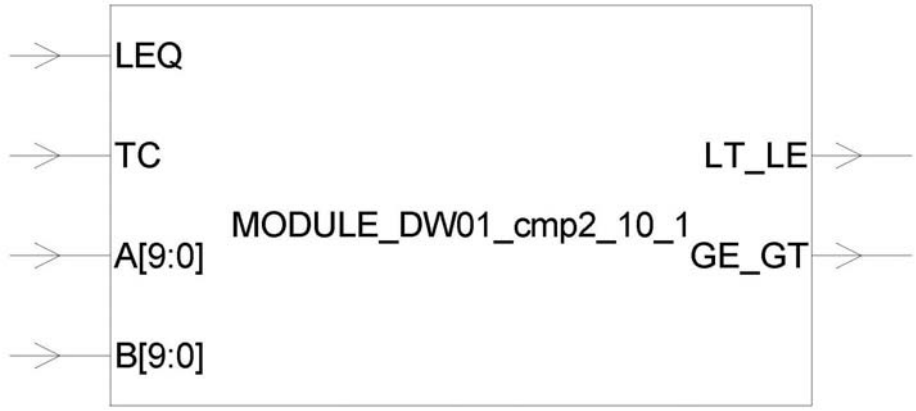


a) Schemos simbolis

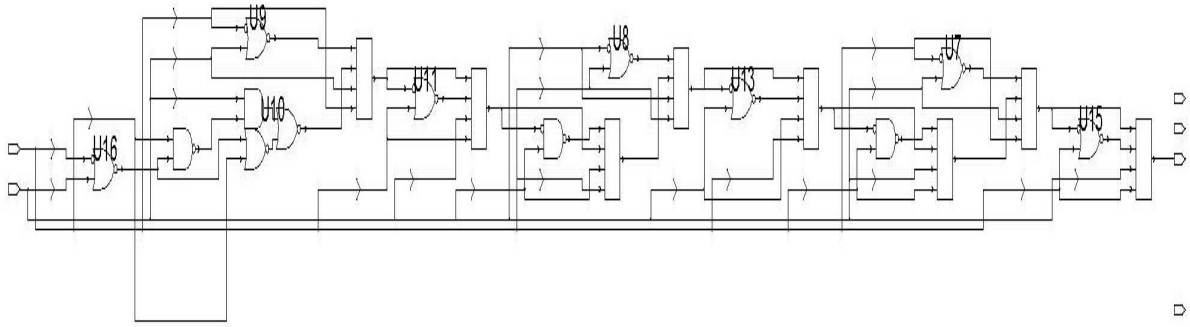


b) Sintezuota schema

Pav. nr. 54 sintezuoto modelio atskira dalis

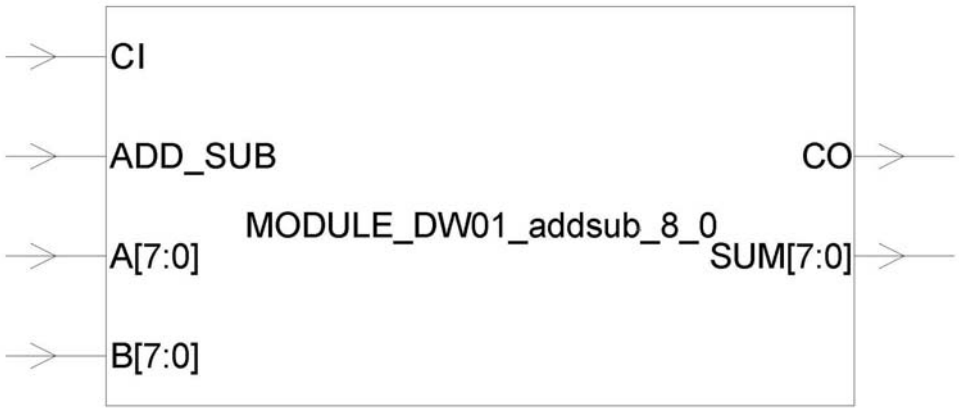


a) Schemos simbolis

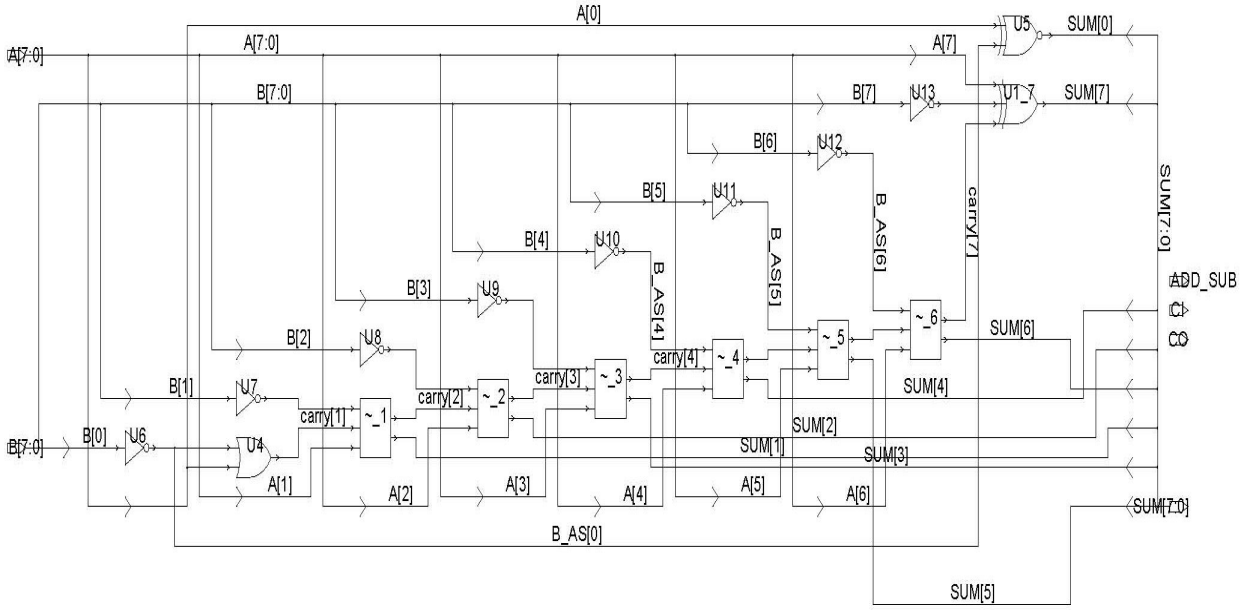


b) Sintezuota schema

Pav. nr. 55 sintezuoto modelio atskira dalis

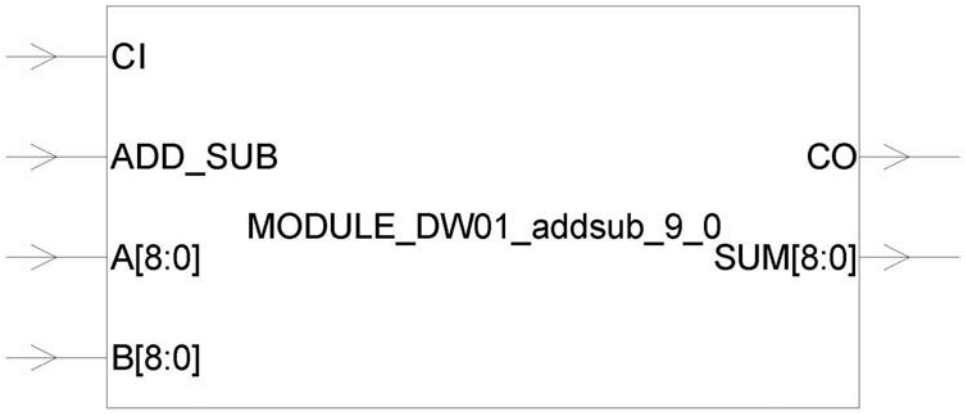


a) Schemos simbolis

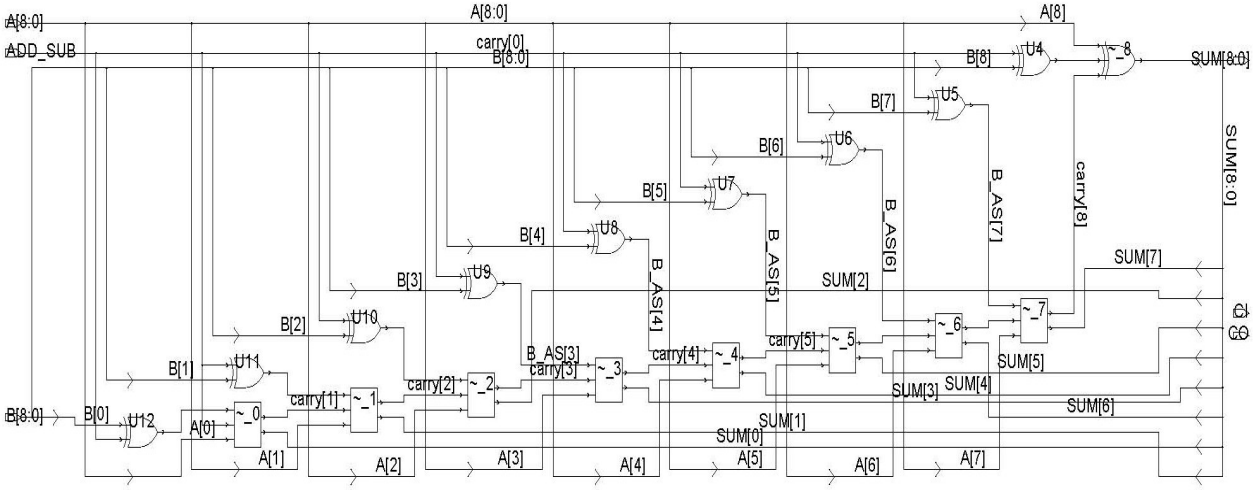


b) Sintezuota schema

Pav. nr. 56 sintezuoto modelio atskira dalis

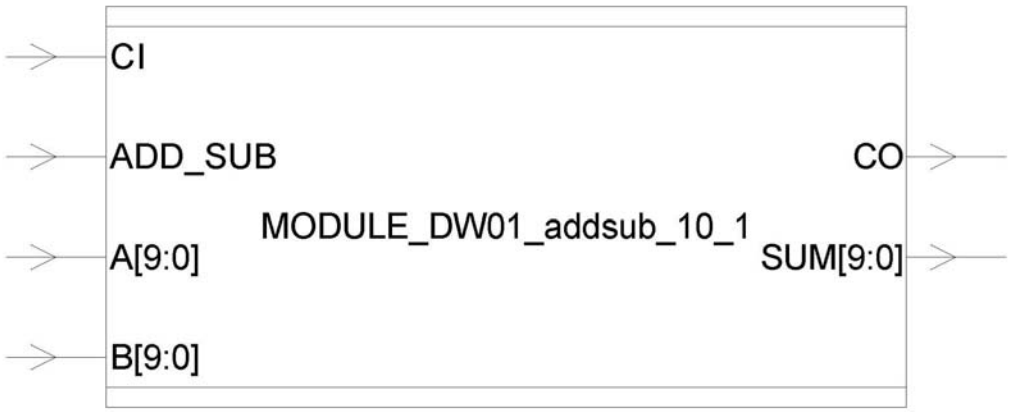


a) Schemos simbolis

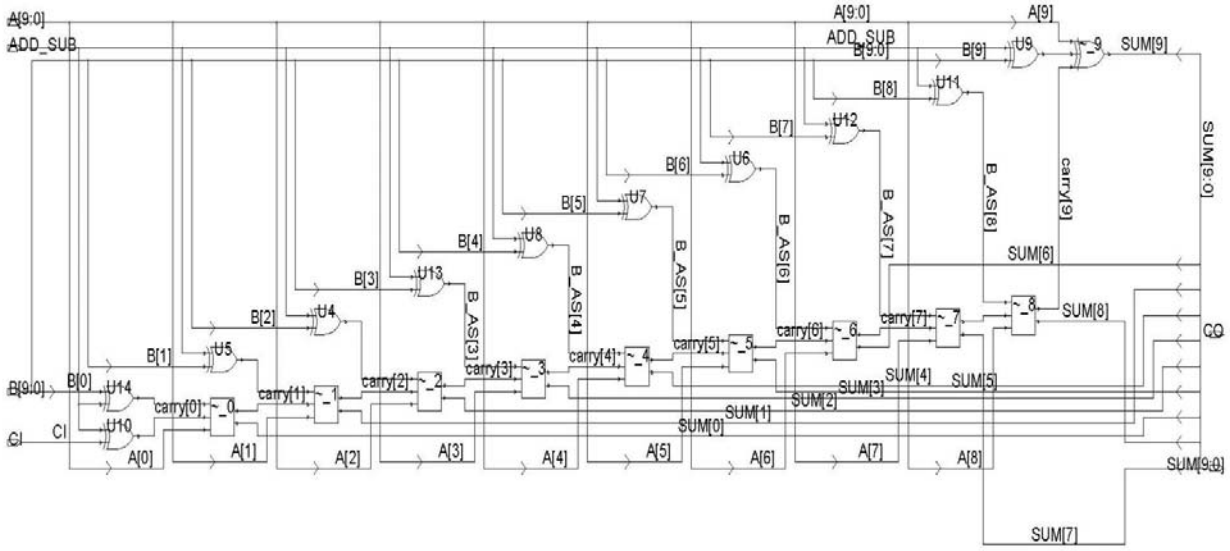


b) Sintezuota schema

Pav. nr. 57 sintezuoto modelio atskira dalis

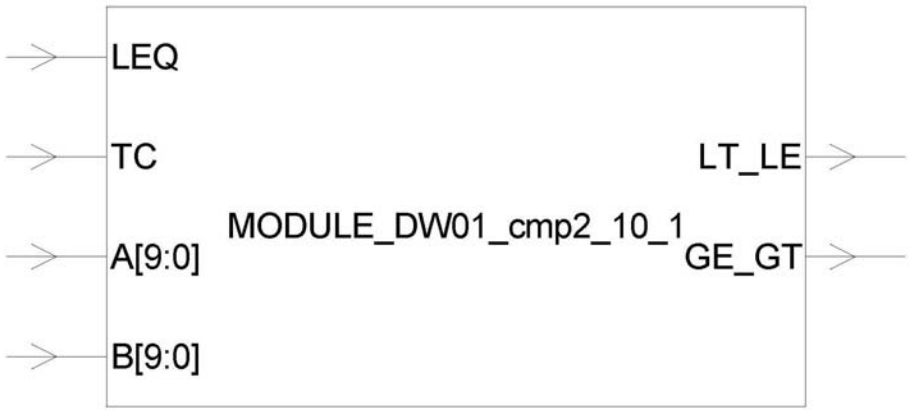


a) Schemos simbolis

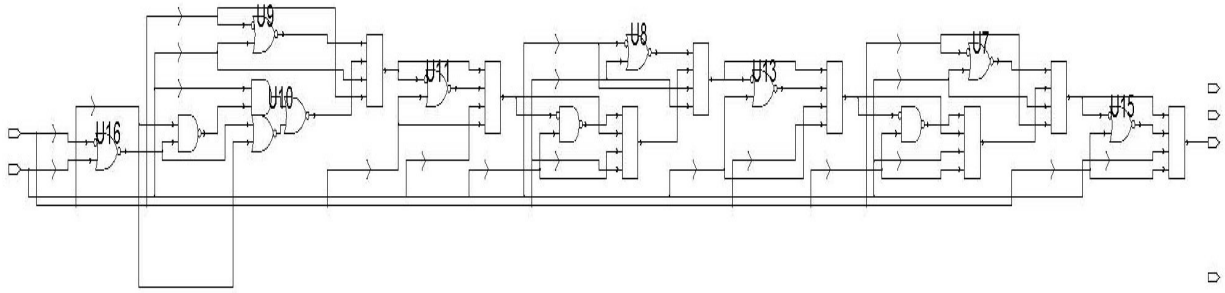


b) Sintezuota schema

Pav. nr. 58 sintezuoto modelio atskira dalis



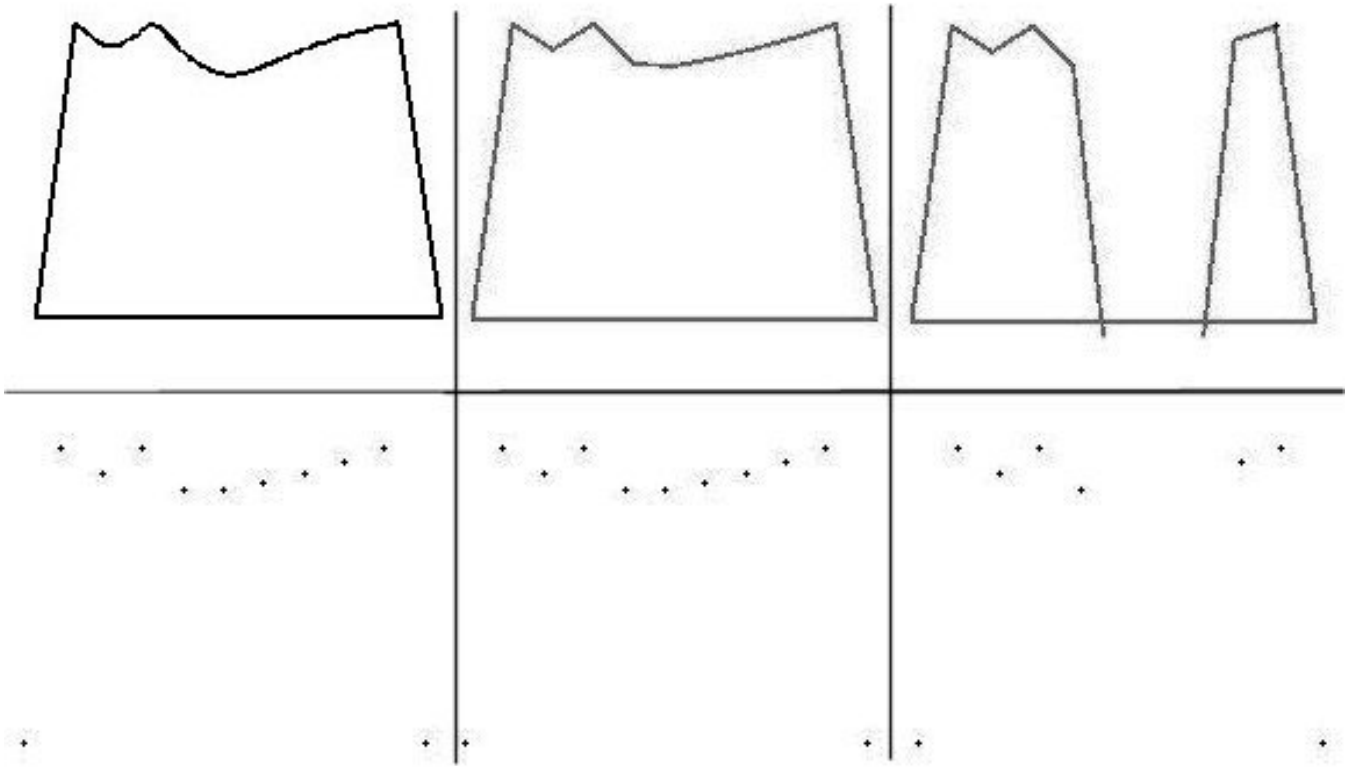
a) Schemos simbolis



b) Sintezuota schema
Pav. nr. 59 sintezuoto modelio atskira dalis

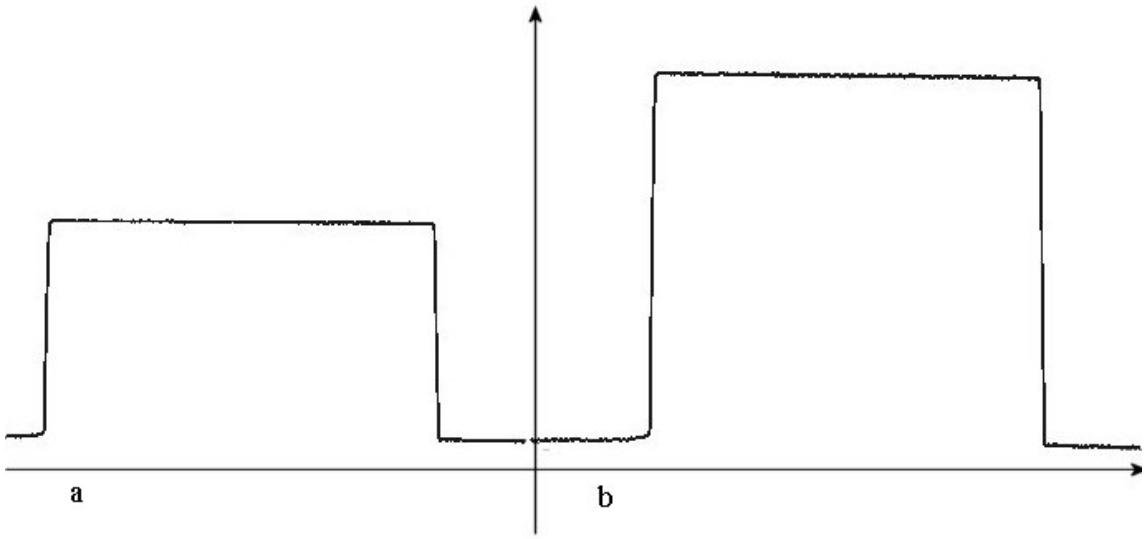
Detalesnė sintezės informacija pateikiama priede Nr.2

4.2 Pradinės specifikacijos palyginimas su eksperimento rezultatais



a)teorinis b)interpoliuotas c) neinterpoliuotas

Pav. nr. 60 interpoliacijos algoritmo rezultatai gauti su C++



a) gautas kamera b)transformuotas i pradinį variantą

Pav. nr. 61 transformacijos algoritmo rezultatai gauti su C+

4.3 TLM modelio rezultatai

Algoritmas	Laikas, ns
Duomenų įrašymas	3072000
Filtravimas	30496400
Kreives paieška	2194090
Interpoliacija	28950
Bendras sistemos veikimo laikas	35791450

Lentelė nr. 9 TLM laikai

TLM modelio žurnalas pateikiamas priede nr. 1

5 Išvados

- ✓ Išanalizavome vaizdų apdorojimo sistemą
- ✓ Apžvelgėme trimačių skenerių sistemas ir jų pritaikymo sritis
- ✓ Naudodamiesi C++ programavimo kalba sukūrėme modeliuojamą pradinę specifikaciją
- ✓ Apžvelgėme C++ modulių perėjimo į *SystemC TLM* ir *RTL* bendrąsias taisykles
- ✓ Pradinės modeliuojamos specifikacijos ir bendrųjų perėjimo taisyklių pagalba sukūrėme ir sumodeliavome *TLM* modulį.
- ✓ Pasinaudodami sukurtu transakcijos (*TLM*) modulio pagalba ištyrėme kelis sistemos realizacijos variantus.
- ✓ Naudodamiesi *C++* ir *TLM* aprašais sumodeliavome sintezuojamą *SystemC* aprašą.
- ✓ Verifikavome sintezuotą aprašą, pasinaudodami pradinės modeliuojamos specifikacijos rezultatais.
- ✓ Pateikėme sistemų modeliavimo ir sintezės rezultatus

6 Žodynėlis

3D - Trimatis, trimatė erdvė

CMM - Coordinate Measurement Macine Koordinačių matavimo mašina.

3ds - Trimačio vaizdo saugojimo formatas

CCD - charge-coupled devize

RGB - Raudona Žalia Mėlyna (Red Green Blue)

SDK - Programinės įrangos kūrimo įrankių rinkinys (Sofware development kit)

C++ - Programavimo kalba

TLM - SytemC Transakcijų lygio modeliavimas yra paremtas systemC biblioteka.

RTL - Registre trasfer level.

IP – Intectula property, intelektualusis komponentas.

OSCI – Open SystemC initiavive

Soc - System on chip, vienlustė sistema.

3D - trimatė erdvė.

RAM – random access memory,

DMI - direct memory interface,

FIFO – firs in firs out. Duomenų saugojimo būdas.

SW - programinė įranga

HW - aparatūra

SW/HW – programinė / aparatūrinė įranga

Verilog – aparatūros aprašymo kalba

VHDL – aparatūros aprašymo kalba

IBM – international business machines corporation, JAV kompiuteriu gamybos bedrovė

Windows - operacinė sistema

Linux – operacinė sistema

UNIX – operacinė sistema

Channe-kanalail

Interface- sąsaja

Wait – laukimo operacija

Netlist- struktūrinis aprašas

Testbench- bandymų stendas

Socket- prievadas

7 Literatūra

1. Skaitiniai metodai ir algoritmai : vadovėlis aukštųjų mokyklų studentams / Kostas Plukas Kaunas : Naujasis lankas, 2001 (Kaunas : Morkūnas ir Ko) ISBN 9955-03-061-5
2. Winning the SoC Revolution– Experiences in Real Design, Autoriai: Grant Martin, Henry Chang, Grant Edmund Martin, Išleista 2003, ISBN:1402074956
3. UML for SOC Design, Autoriai: Grant Martin, Grant Edmund Martin, Wolfgang Muller, Išleista 2005, ISBN:0387257446
4. TLM Transaction Level Modeling Library, Release 2.0 Draft 2 http://www.systemc.org/members/download_files/check_file?agreement=tlm-2-draft2
5. TLM-2 Requirements Specification v1.1 http://www.systemc.org/members/download_files/check_file?agreement=tlm2_requirements
6. A Tutorial Introduction to the System TLM Standard http://www.nascug.org/PDF/stuart_tlm_tutorial_dvcon06_SHORT.pdf
7. **System-platforms-based SystemC TLM design of image processing chains for embedded applications** http://www.hindawi.com/Getpdf.aspx?doi=10.1155/2007/71043
8. Getting Started with TLM 2.0 http://www.doulos.com/knowhow/systemc/tlm2/
9. Transaction Level Platform Modeling in SystemC for Multi-Processor Designs http://www.gigascale.org/pubs/988/TLM_SystemC_wUBC.pdf
10. Integration of Instruction Set Simulators into SystemC High Level Models http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1115360

11. **SystemC: Key modeling concepts besides TLM to boost your simulation performance** http://www.us.design-reuse.com/articles/17877/systemc-tlm.html

12. http://www.sjbaker.org/projects/scanner/ [2007 m. sausis 19 d.]
13. http://www.muellerr.ch/engineering/laserscanner/tutorial/the_tutorial.html [2007 m. sausis 19 d.]
14. http://n.ethz.ch/student/pleiness/en/index.php [2007 m. sausis 19 d.]
15. http://scantech.dk/showpage.asp?page=16&root=2 [2007 m. sausis 19 d.]
16. http://www.laserdesign.com/products.htm [2007 m. sausis 19 d.]
17. http://www.simple3d.com/ [2007 m. sausis 19 d.]
18. http://umsis.miami.edu/~rhartma1/een571_proj2.html [2007 m. birželis 19 d.]
19. http://www.likit.lt/term/z2odynas.html

8 Priedai

Nr. 1

TLM aplikacijos žurnalas [20]

```
:pav_Initiator: Nuskaitymas iš failo
:pav_Initiator: Nuskaitytas aukštis: 480, ilgis: 640
:pav_Initiator: Nuskaitymas iš failo << END
Pradedame generuoti Gausian filtro matricą
0.3679, 0.6065, 0.3679,
0.6065, 1.0000, 0.6065,
0.3679, 0.6065, 0.3679,
TLM DMI Allowed 1
TLM Start address 0, end addr 307839
Gauta eilute=640, stulpelis=480
Pradedame įrašym į atmintį at time 10 ns
Duomenų įrašymas į atmintį baigtas at time 3072010 ns
Duomenų patikrinimas baigtas
Pradedame filtravimą: at time 3072010 ns
Filtravimas baigtas  : at time 33568410 ns
Išvedimas duomenų baigtas fala galite rasti '../data/Nufiltruotas.out.dat'
Duomenų analizė Kreivės paieška at time 33568410 ns
Duomenų analizė Kreivės: paieška baigta  at time 35762500 ns
Pradedame Interpoliaciją at time 35762500 ns
Pabaigėme Interpoliaciją at time 35791450 ns
Saugome kreivę į atmintį '../data/kreivė_iš_atminties_interpoluota.out.dat'
```

Nr. 2

Sintezacijos “command_view.log” failo fragmentai

```
design_analyzer> search_path = ". /opt/synopsys_syn-2006.06/libraries/syn
/opt/synopsys_syn-2006.06/dw/sim_ver /opt/synopsys_syn-2006.06/dw/syn_ver "
{".", "/opt/synopsys_syn-2006.06/libraries/syn", "/opt/synopsys_syn-
2006.06/dw/sim_ver", "/opt/synopsys_syn-2006.06/dw/syn_ver"}
design_analyzer> link_library = "class.db "
{"class.db"}
design_analyzer> target_library = "class.db "
{"class.db"}
design_analyzer> symbol_library = "class.sdb "
{"class.sdb"}
design_analyzer> default_schematic_options = "-size infinite"
"-size infinite"
design_analyzer> compile  -map_effort high
Loading db file '/opt/synopsys_syn-2006.06/libraries/syn/class.db'
Information: Evaluating DesignWare library utilization. (UISN-27)

============================================================================
| DesignWare Building Block Library       |      Version       | Available |
============================================================================
| Basic DW Building Blocks                | Y-2006.06-DWBB_0606 |    *      |
| Licensed DW Building Blocks             |                     |    *      |
============================================================================


Information: There are 26 potential problems in your design. Please run
'check_design' for more information. (LINT-99)
```

```
  Loading target library 'class'

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'MODULE_DW_div_uns_16_8_0'
  Processing 'MODULE_DW01_add_8_0'
  Processing 'MODULE_DW01_sub_8_0'
  Processing 'MODULE'

  Updating timing information
Information: Design 'MODULE' has no optimization constraints set. (OPT-108)

  Beginning Implementation Selection
  ----------------------------------



Current design is 'MODULE'.
1
design_analyzer> current_design =
"/home/ramucvir/systemc_lab/lab1/rtl_work/dff1_rtl_gate.db:MODULE"
Current design is 'MODULE'.
"/home/ramucvir/systemc_lab/lab1/rtl_work/dff1_rtl_gate.db:MODULE"
design_analyzer> create_schematic -size infinite -schematic_view -symbol_view -
hier_view
Loading db file '/opt/synopsys_syn-2006.06/libraries/syn/class.sdb'
Generating schematic for design: MODULE
The schematic for design 'MODULE' has 1 page(s).

1
design_analyzer> designer = ""
""
design_analyzer> company = ""
""
design_analyzer> search_path = ". /opt/synopsys_syn-2006.06/libraries/syn
/opt/synopsys_syn-2006.06/dw/sim_ver /opt/synopsys_syn-2006.06/dw/syn_ver "
{".", "/opt/synopsys_syn-2006.06/libraries/syn", "/opt/synopsys_syn-
2006.06/dw/sim_ver", "/opt/synopsys_syn-2006.06/dw/syn_ver"}
design_analyzer> link_library = "tc6a_cbacore.db"
{"tc6a_cbacore.db"}
design_analyzer> target_library = "tc6a_cbacore.db"
{"tc6a_cbacore.db"}
design_analyzer> symbol_library = "tc6a_cbacore.sdb"
{"tc6a_cbacore.sdb"}
design_analyzer> default_schematic_options = "-size infinite"
"-size infinite"
design_analyzer> compile  -map_effort high
Information: Choosing a test methodology will restrict the optimization of
sequential cells. (UIO-12)

Information: There are 28 potential problems in your design. Please run
'check_design' for more information. (LINT-99)


  Loading target library 'cba_core'

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'MODULE_DW_div_uns_16_8_0'
  Processing 'MODULE_DW01_add_8_0'
  Processing 'MODULE_DW01_sub_8_0'
  Processing 'MODULE'

  Updating timing information
Information: Design 'MODULE' has no optimization constraints set. (OPT-108)

  Beginning Implementation Selection
  ----------------------------------
```

```
design_analyzer> report_area
Information: Updating design information... (UID-85)

****************************************
Report : area
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:14:56 2008
****************************************

Library(s) Used:

    cba_core (File: /opt/synopsys_syn-2006.06/libraries/syn/tc6a_cbacore.db)

Number of ports:               46
Number of nets:                62
Number of cells:                4
Number of references:           4

Combinational area:        885.700073
Noncombinational area:       0.000000
Net Interconnect area:    4488.593750

Total cell area:           885.700012
Total area:               5374.293945
1
design_analyzer> report_reference

****************************************
Report : reference
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:14:56 2008
****************************************

Attributes:
    b - black box (unknown)
   bo - allows boundary optimization
    d - dont_touch
   mo - map_only
    h - hierarchical
    n - noncombinational
    r - removable
    s - synthetic operator
    u - contains unmapped logic

Reference          Library       Unit Area   Count    Total Area   Attributes
--------------------------------------------------------------------------------
MODULE_DW01_add_8_0              46.279999       1     46.279999  h
MODULE_DW01_sub_8_0              61.510006       1     61.510006  h
MODULE_DW_div_uns_16_8_0        777.909973       1    777.909973  h
gnd                cba_core       0.000000       1      0.000000
--------------------------------------------------------------------------------
Total 4 references                                    885.699951
1
design_analyzer> report_area

****************************************
Report : area
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:15:05 2008
****************************************

Library(s) Used:
```

```
    cba_core (File: /opt/synopsys_syn-2006.06/libraries/syn/tc6a_cbacore.db)

Number of ports:               46
Number of nets:                62
Number of cells:                4
Number of references:           4

Combinational area:        885.700073
Noncombinational area:       0.000000
Net Interconnect area:    4488.593750

Total cell area:           885.700012
Total area:               5374.293945
1
design_analyzer> report_reference

****************************************
Report : reference
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:15:05 2008
****************************************

Attributes:
    b - black box (unknown)
   bo - allows boundary optimization
    d - dont_touch
   mo - map_only
    h - hierarchical
    n - noncombinational
    r - removable
    s - synthetic operator
    u - contains unmapped logic

Reference          Library       Unit Area   Count    Total Area   Attributes
--------------------------------------------------------------------------------
MODULE_DW01_add_8_0              46.279999       1     46.279999  h
MODULE_DW01_sub_8_0              61.510006       1     61.510006  h
MODULE_DW_div_uns_16_8_0        777.909973       1    777.909973  h
gnd                cba_core       0.000000       1      0.000000
--------------------------------------------------------------------------------
Total 4 references                                    885.699951

design_analyzer> report_area
Information: Updating design information... (UID-85)

****************************************
Report : area
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:30:43 2008
****************************************

Library(s) Used:

    cba_core (File: /opt/synopsys_syn-2006.06/libraries/syn/tc6a_cbacore.db)

Number of ports:               58
Number of nets:               792
Number of cells:              604
Number of references:          29

Combinational area:       1298.465454
Noncombinational area:       0.000000
Net Interconnect area:    4894.687500

Total cell area:          1298.469971
Total area:               6193.152832
```

```
1
design_analyzer> report_reference

****************************************
Report : reference
Design : MODULE
Version: Y-2006.06
Date   : Wed Feb 20 14:30:43 2008
****************************************

Attributes:
    b - black box (unknown)
   bo - allows boundary optimization
    d - dont_touch
   mo - map_only
    h - hierarchical
    n - noncombinational
    r - removable
    s - synthetic operator
    u - contains unmapped logic

Reference          Library       Unit Area   Count    Total Area   Attributes
--------------------------------------------------------------------------------
and2a2             cba_core       1.580000       4      6.320000
and2b1             cba_core       1.580000      40     63.200002
and2c0             cba_core       0.860000     154    132.440002
and3b1             cba_core       1.580000       6      9.480000
ao1a0              cba_core       1.730000       3      5.190000
ao1c0              cba_core       1.730000       1      1.730000
ao1d1              cba_core       1.580000       4      6.320000
ao1e0              cba_core       1.730000       1      1.730000
ao1f1              cba_core       1.580000      33     52.140001
ax1a2              cba_core       3.310000       3      9.930000
fa1a0              cba_core       5.180000     129    668.219978  r
fa1a1              cba_core       5.890000       1      5.890000  r
ha1a0              cba_core       3.450000      18     62.100001  r
inv1a0             cba_core       0.860000      12     10.320000
inv1a1             cba_core       0.860000      59     50.740001
mx2d0              cba_core       1.730000       2      3.460000
nand2a0            cba_core       0.860000       4      3.440000
nand2b1            cba_core       1.580000       2      3.160000
nor2a0             cba_core       0.860000      18     15.480000
nor3a1             cba_core       1.580000       1      1.580000
oa1a0              cba_core       1.730000       3      5.190000
oa1b1              cba_core       2.440000       2      4.880000
oa1c0              cba_core       1.730000       1      1.730000
oa1f1              cba_core       1.580000      17     26.860001
or2c0              cba_core       0.860000      42     36.120001
xnor2a2            cba_core       2.440000       3      7.320000
xor2a2             cba_core       2.440000      20     48.800001
xor2b2             cba_core       2.440000      19     46.360001
xor3a2             cba_core       4.170000       2      8.340000
--------------------------------------------------------------------------------
Total 29 references                                  1298.469727
1
design_analyzer>
Thank you...
```

Nr.3 Modeliuojama pradinė specifikacija, TLM ir RTL aprašai pateikiami kompaktiniame diske kartu su duomenų ir rezultatų failais.