KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Kazimierz Wojno

Fizinės realaus laiko modeliavimo posistemės sudarymas bei tyrimas

Magistro darbas

Darbo vadovas:
doc. dr. Egidijus Kazanavičius

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. Egidijus Kazanavičius
2004 05

Fizinės realaus laiko modeliavimo posistemės sudarymas bei tyrimas

Magistro darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. Jurgita Mikelionienė
2004 05

Recenzentas
doc. dr. Gintaras Palubeckis
2004 05

Vadovas
doc. dr. Egidijus Kazanavičius
2004 05

Atliko
IFM-8/1 gr. stud.
Kazimierz Wojno
2004 05

Kaunas, 2004

Turinys

1 Įvadas

Skaitmeninių signalų apdorojimas SSA (*angl. DSP – digital signal processing*) tai tolydinio (analoginio) signalo apdorojimas skaitmeniniu būdu realiame laike. Signalu yra vadinama registruojama funkcija, apibūdinanti fizinės sistemos būseną tam tikru laiko momentu. Funkcijos aprašomos vienu ar daugiau parametrų. Realaus laiko sistema vadiname tokią sistemą, kurios teisingumas priklauso ne tik nuo loginių skaičiavimų rezultatų, bet ir nuo laiko, per kurį tie rezultatai gaunami.

Šiais laikais realaus laiko sistemos tampa vis sudėtingesnės. Nepakanka programuotojo rašančio programinę įrangą , kuri sprendžia duotą problemą. Sistemos yra tiek sudėtingos, kad reikalauja sistemos architektų, analitikų, programuotojų bei testuotojų suderinto darbo. Organizuotas darbas bendram tikslui pasiekti vadinamas procesu (Russel Kay, 2002).

Sistemos kūrimo procesas vadinamas - sistemos kūrimo ciklu SKC (*angl. SDLC – system development life cycle*). SKC kaštų bei trukmės mažinimui yra taikomi įvairūs valdymo metodai: krioklio, spiralinis, spartaus prototipo, augimo ir kiti. Daugumoje įprastų sistemų SKC valdymo metodai tinka ir realaus laiko sistemoms.

Nepaisant sistemų kūrimo ciklo pažangumo, sudarant realaus laiko sistemas susiduriama su įvairiomis problemomis iteracijų vykdymo metu, kas ilgina produkto pristatymą rinkai (*angl. time to maket*), didina projekto kaštus ir didina projekto žlugimo riziką. Problemos atsiranda sistemos prototipo etape. Prototipo modelis yra sudaromas neautomatizuotu būdu – procesas būna ilgas ir gali įsivelti klaidų. Jis yra modeliuojamas virtualioje aplinkoje ir virtualiame laike, todėl modeliavimo rezultatai neatspindi laikinių modelio charakteristikų.

Metodologija apibrėžianti virtualių RLS komponentų perkėlimą fizinę SSA platformą leistų sukurti priemonę, kuri tai automatiškai atliktų. Atsirastų papildomas abstrakcijos lygis žiūrint iš sistemos realizacijos aspekto. Paprastumas kokį suteikia grafinis RLS projektavimo būdas sumažintų klaidų kiekį bei projektavimo ciklo trukmę. Tai leistų ženkliai sutaupyti laiką bei jėgas kuriant RLS prototipą.

Problema – Vizualaus RLS projektavimo ir modeliavimo sistemos jau egzistuoja. Jos yra patogios ir lengvai suprantamos, tačiau RLS modeliuojamos virtualioje aplinkoje ir virtualiame laike. Toks modeliavimo metodas negarantuoja, kad projektuojama RLS funkcionuos teisingai fizinėje aplinkoje.

Pagrindinis uždavinys - pasiūlyti metodiką leidžiančią iš sistemos struktūrinio aprašo sukurti posistemę, kuri projektuojamą realaus laiko sistemą modeliuotų fizinėje aplinkoje. Šios metodikos dėka galima būtų automatizuoti posistemės sudarymo procesą.

Tikslai:

- atlikti skaitmeninių signalų apdorojimo analizę,
- išnagrinėti įprastas bei įterptines realaus laiko sistemas,
- atlikti RLS programinės įrangos analizę,
- išanalizuoti sistemų kūrimo ciklų bendrus trūkumus
- pasiūlyti metodiką, leidžiančia iš sistemos aprašo sukurti modeliavimo posistemę fizinėje aplinkoje
- metodiką eksperimentiškai patikrinti.

Darbo struktūra:

- Analizės dalis – atliekama skaitmeninių signalų apdorojimo, realaus laiko sistemų analizė. Analizuojama realaus laiko sistemos naudojamą programinę įrangą, jos tipus bei savybes. Surandamos bendros sistemų kūrimo ciklų problemos ir pasiūloma jų spendimo metodai. Apžvelgiama keletas sistemų, kurios bando spręsti tas problemas.
- Teorinė dalis – pateikiama metodika skirta fizinei modeliavimo posistemei sudaryti. Pateikiamas sistemos aprašymo modelis praplėstas programinių komponentų generavimo metodika. Metodikos elementai pateikiami algoritmo forma.
- Eksperimentinė dalis – aprašomas atliktas eksperimentas. Pasirinktas skaitmeninio filtravimo uždavinys modeliuojamas virtualioje aplinkoje bei fizinėje aplinkoje. Palyginami modeliavimo rezultatai.

2 Analizė

2.1 Tiriamojo objekto analizė

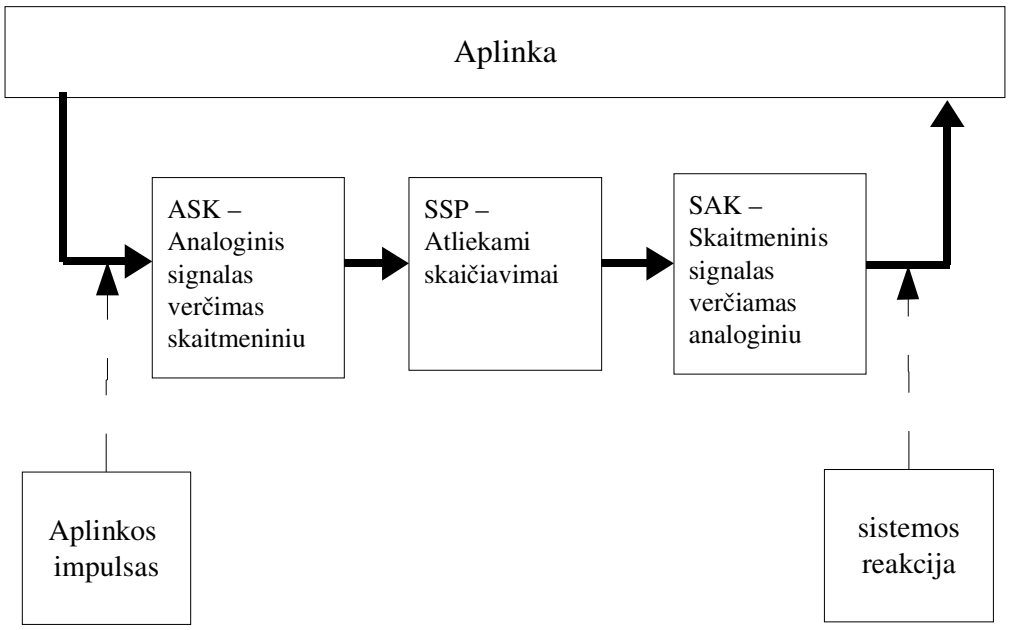
Skaitmeninių signalų apdorojimas SSA (*angl. DSP – digital signal processing*) tai tolydinio (analoginio) signalo apdorojimas skaitmeniniu būdu realiame laike. Signalu yra vadinama registruojama funkcija, apibūdinanti fizinės sistemos būseną tam tikru laiko momentu. Funkcijos aprašomos vienu ar daugiau parametrų.

Signalai pagal laikinį pasiskirstymą būna dviejų rūšių: analoginiai ir diskretieji. Analoginiai signalai yra tokie, kurie tolygiai einant laikui apibūdina tolygiai kintantį parametrą: pvz., nuolatos matuojamas objekto greitis tam skirtu prietaisu (automobilio spidometras) arba nuolatos įvertinamas virpesio dažnis, stipris (kalba, registruota žmogaus ausies). Kai tolygiai kintančio parametro amplitudė išmatuojama, atskaitoma tam tikrais diskretiniais laiko momentais, gaunamas diskretusis signalas. Laiko tarpas tarp atskaitų yra vadinamas diskretizavimo žingsniu. Priimta, kad tam pačiam signalui diskretizacijos žingsnis lieka pastovus. Diskretieji signalai yra impulsai arba impulsų sekos, atitinkantys kintančiojo laike parametro reikšmę diskretiniais laiko momentais (Evaldas Garška, 1998).

Dabartiniu metu vyrauja informacijos perdavimas skaitmeniniais signalais. Įtaisas, kuris analoginį signalą paverčia skaitmeniniu, vadinamas analoginiu skaitmeniniu keitikliu ASK (*angl. ADC – analog to digital converter*). Įtaisas, kuris skaitmeninį signalą paverčia analoginiu vadinamas skaitmeniniu analoginiu keitikliu SAK (*angl. DAC – digital to analog converter*).

SSA uždaviniams spręsti naudojami specialios paskirties mikroprocesoriai – skaitmeninių signalų procesoriai SSP. Kadangi SSA uždaviniai yra imlūs matematiniams skaičiavimams, todėl bendros paskirties procesoriai nėra pajėgūs tuos uždavinius spręsti. Be to, SSP procesoriams keliami energijos sunaudojimo bei išskiriamos šilumos kiekio reikalavimai (Craig Marven, Gillian Evers, 1996).

Skaitmeninių signalų apdorojimui naudojamas SSP, kuris dirba tik su skaitmeniu signalu, bei ASK ir SAK įrenginiai. Iš aplinkos gautas tolyginis signalas verčiamas skaitmeniniu, kurį apdoroja SSP procesorius. Apdorotas signalas vėl verčiamas anloginiu.



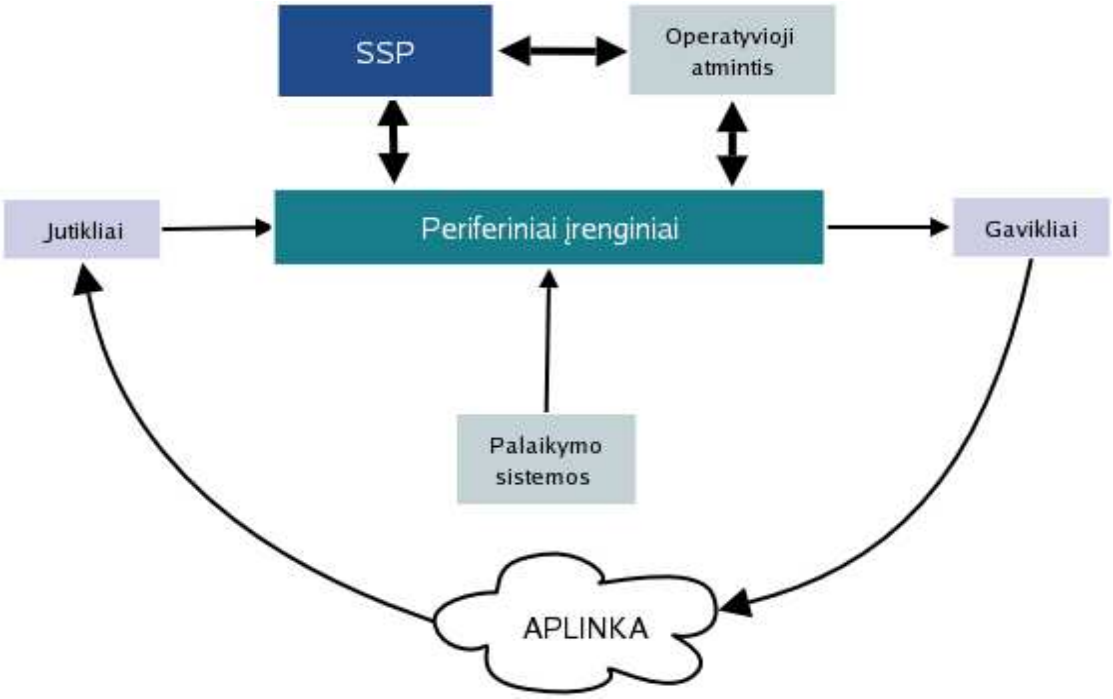
1 pav. Skaitmeninių signalų apdorojimas

Realaus laiko sistema RLS (angl. RTS – real time system) vadiname tokią sistemą, kurios teisingumas priklauso ne tik nuo loginių skaičiavimų rezultatų, bet ir nuo laiko, per kurį tie rezultatai gaunami. Rezultatų skaičiavimo laikas tai sistemos reakcijos laikas į išorinį impulsą. Tarkim t_{imp} tai laiko momentas kada į RLS įėjimą patenka aplinkos sukeltas impulsas, o t_{ats} – laiko momentas kada RLS atlikus skaičiavimus išsiunčia atsako impulsą – tuomet sistemos reakcijos laikas t_{reak} yra skirtumas tarp t_{imp} ir t_{ats} ($t_{reak} = t_{imp} - t_{ats}$) (Raimund Kirner, 2003).

Pavyzdžiui robotas, kuris paima dėžes nuo konvejerio pakavimo mašinai, yra realaus laiko sistema. Kiekviena dėžė juda konvejeriu ir robotas turi siaurą laiko tarpą ją paimti. Robotas nepaims dėžės ne tik kai jo alkūnė nusileis per vėlai, bet kai ir per anksti. Egzistuoja daugybė RLS panaudojimų: automobiliuose reguliuojant variklio darbą ar važiuoklės elgesį nenumatytose situacijose, gamyklose reguliuoja konvejerio bei automatų darbą ir taip toliau. Ypač plačiai telekomunikacijos sistemose (modemuose, DSL, skaitmeninėje telefonijoje, GPS ir pan.).

Sunkiausio atvejo apdorojimo laikas (*angl WCET – worst case execution time*) – tai maksimalaus sistemai reikalingas laikas užduočiai atlikti (Changhao Jiang, 1998).

Tai tokios sistemos, kurios veikia didesnės sistemos viduje. Įterptinė sistema dažniausiai padeda išorinei sistemai vykdyti tam tikras užduotis. Pagrindinis sistemos komponentas yra SSP procesorius.



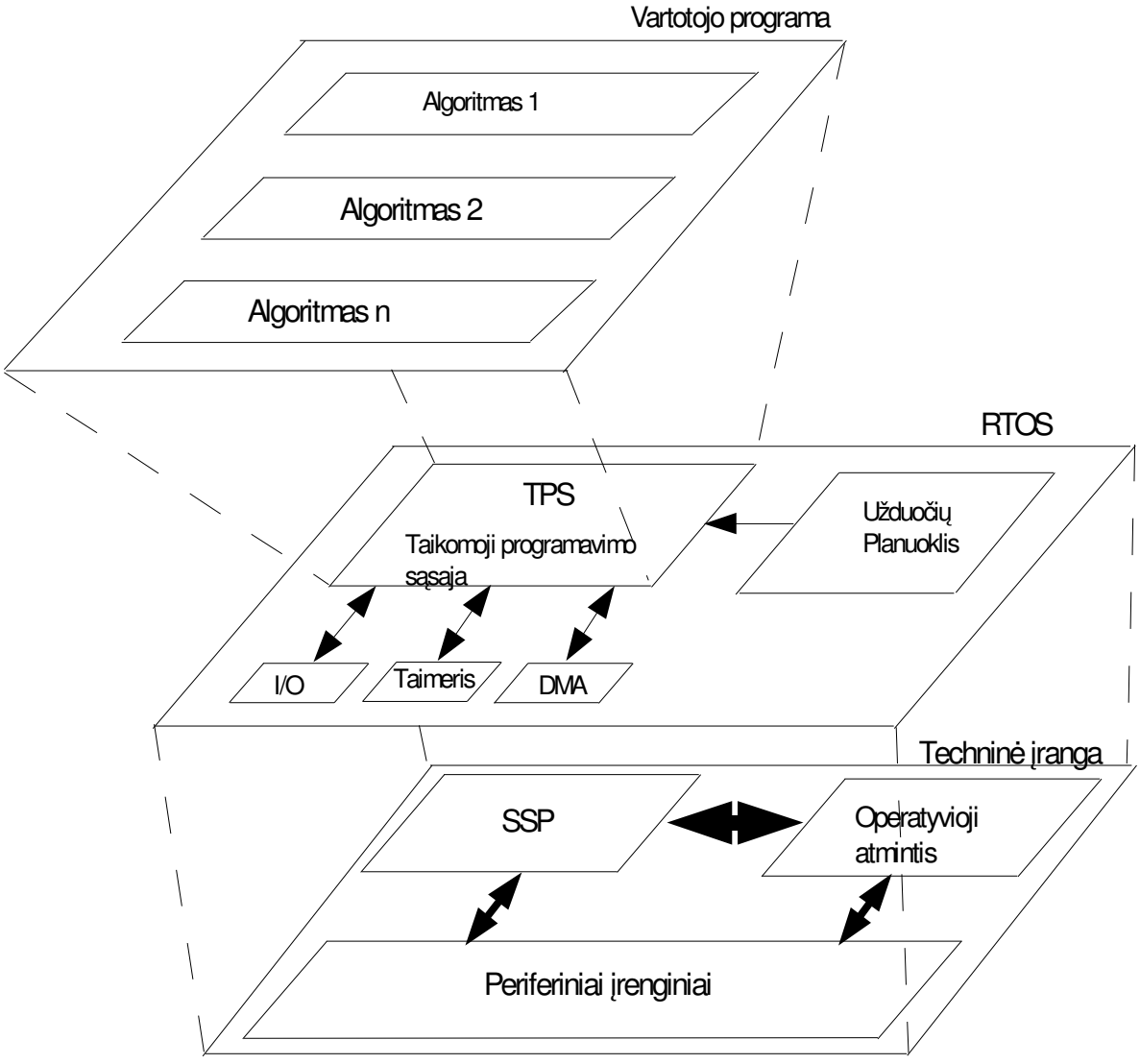
2 pav. Įterptinės RLS techninės įrangos komponentai

Signalas iš aplinkos į RLS paduodamas per periferinius įrenginius. Procesorius gali nuskaityti duomenis tiesiogiai iš periferinių įrenginių. Kad neužkrauti procesoriaus bereikalingu darbu duomenys į atmintį užkraunami tiesiai taip vadinamais DMA (angl. Direct Memory Access) kanalais. Apdoroti duomenys perduodami periferiniams įrenginiams analogiškai. Apdorotas signalas patenka į aplinką. Prie įprastų RLS prijungiami vartojo sąsajos komponentai (pvz,: jungikliai, mygtukai, skystųjų kristalų monitoriai ir pan.), kad galima būtų keisti sistemos vidinius parametrus. Įterptinės RLS tokių komponentų neturi.

Palaikymo sistemos, tokios kaip elektros energijos šaltinis ar aušinimo sistema, reikalingos RLS funkcionavimo palaikyti. Įterptinėse RLS skaitmeninių signalų procesorius, operatyvinė atmintis bei periferiniai įrenginiai sudaro integrinę schemą. Konkretus minėtų techninės įrangos komponentų rinkinys vadinamas SSA platforma. Priklausomai nuo SSP procesoriaus tipo-rūšies platformos klasifikuojamos į SSA platformų šeimas.

RLS sudaro tiek techninės įrangos tiek programinės įrangos komponentai. Sistemą galėtume atvaizduoti dviejų hierarchinių sluoksnių pjūviu – programinės įrangos sluoksnis valdantis techninės įrangos komponentus. SSA uždavinių keliami reikalavimai neleidžia naudoti bendros paskirties techninės įrangos, todėl naudojama aritmetiniams skaičiavimas optimizuota SSA platforma. Programinė įrangos (PĮ) išnaudoja platformos skaičiavimų optimizacijas.

RLS programinė įranga skirstoma į dvi kategorijas: vartotojo arba taikomosios ir sisteminė programinė įranga. Sisteminė PĮ skirta techninės įrangos valdymui. Taikomoji PĮ naudoja sisteminės PĮ funkcijas konkrečioms užduotims atlikti. Sisteminė PĮ tai specialios paskirties operacinės sistema vadinama realaus laiko operacinė sistema RLOS. Realaus laiko operacinė sistema kaip ir įprasta operacinė sistema užtikrina daugiaprograminį režimą bei vieningą taikomojo programavimo sąsają (*angl. API – application programming interface*) „slepiant“ techninės įrangos apdorojimą. RLOS yra tarpinis sluoksnis tarp vartotojo programų ir techninės įrangos – kaip pavaizduota 3 paveikslėlyje.



3 pav. Sluoksninė RLS struktūra

Jei programą, kuri realizuoja tam tikrą užduotį, išsivaizduosime kaip nuosekliai

vykdomų komandų seką, tai bus procesas arba užduotis (*angl. Task*). Operacinė sistema užtikrina daugiaprograminį (*angl, multitasking*) režimą – vienas procesorius gali vykdyti kelias užduotis vienu metu (sistema periodiškai keičia vykdomą užduotį).

Užduočių planuoklis (*angl. scheduler*) sprendžia kuris procesas kuriuo metu yra vykdomas. Planuoklis nutraukia aktyviosios užduoties vykdymą tam, kad leistų dirbti svarbesnei užduočiai (turinčiai didesnį prioritetą). Vykdomosios užduoties pakeitimo metu, planuoklis užsaugo nutrauktos užduoties kontekstą, kad ši galėtų pratęsti darbą vėliau, išrenka kitą užduotį vykdymui, atstato naujai išrinktos užduoties kontekstą ir ją paleidžia (Rulph Chassing, 2002).

Operacinės sistemos paskirtis yra valdyti visus techninės įrangos komponentus, tuo pačiu taikomosios programinės įrangos programuotojui leisti naudotis sistemos ištekliais tik per specialias TPS funkcijas tam, kad kiek įmanoma mažiau algoritmo realizacija būtų priklausom nuo techninės įrangos.

Realaus laiko operacine sistema vadiname tokią OS, kuri:

- gali skirtingiems procesams skirti skirtingus prioritetus tam, kad galima būtų patenkinti kritinių procesų laiko reikalavimus;
- nusako prioritetų paveldimumo mechanizmą;
- palaiko daugiaprograminį režimą bei prevencinį (*angl. preemptible*) procesų vykdymo planavimą – tai reiškia kad planuoklis gali nutraukti kurio nors proceso vykdymą bet kuriuo metu, kad užleisti vietą svarbesniam procesui arba aptarnauti kokį nors įrenginį;
- užtrunka fiksuotą arba numatomą laiko kiekį vykdomo proceso perjungimui atlikti. Jeigu sugaištamas laikas būtų neprognozuojamas tai visos sistemos veikimas būtų neprognozuojamas ir netenkintų laikinių charakteristikų reikalavimų;
- turi pastovų arba numatomą gaišatį (*angl. latency*) – laikas per kurį sistema aptarnauja aparatūrinį pertraukimą ir leidžia procesui tęsti darbą turi būti numatomas. Vėl gi, jei nenumatomą laiką sistema uždels techninei įrangai aptarnauti, tai paties proceso veikimo laikas irgi tampa neprognozuojamas;
- palaiko numatomą procesų sinchronizavimo mechanizmą – paprastai procesas nėra vykdomas pats sau nieko apie kitus nežinantis, paprastai

procesai turi keistis duomenimis tarpusavy. OS tiekia priemones komunikacijai tarp procesų užtikrinti. Jei siunčiami vieno proceso (siuntėjo) duomenys uždels nenumatomą laiko tarpą, tai kitas procesas (gavėjas) negalės atlikti savo darbo prognozuojamu laiku;

Pati RLOS tai karkaso (*angl. framework*), skirto RLS kūrimui, dalis. Karkasus bei kitas priemones (kompiliatorius, emuliatorius, integruotas programavimo aplinkas) tiekia SSA platformų gamintojai. Tų priemonių pagalba algoritmus galima realizuoti aukšto lygio programavimo kalba, leisti modeliuojamoje arba fizinėje aplinkoje, stebėti sistemos funkcionavimą. Visos šitos priemonės skirtos produkto sudarymo ciklui paspartinti bei galutinio produkto kokybiškumui užtikrinti.

2.2 Problema

Šiais laikais RLS tampa vis sudėtingesnės. Nepakanka programuotojo rašančio programinę įrangą , kuri sprendžia duotą problemą. Sistemos yra tiek sudėtingos, kad reikalauja sistemos architektų, analitikų, programuotojų bei testuotojų suderinto darbo. Organizuotas darbas bendram tikslui pasiekti vadinamas procesu (Russel Key, 2002).

Sistemos kūrimo procesas vadinamas - sistemos kūrimo ciklu SKC (*angl. SDLC – system development life cycle*). SKC kaštų bei trukmės mažinimui yra taikomi įvairūs valdymo metodai: krioklio, spiralinis, spartaus prototipo, augimo ir kiti. Daugumoje įprastų sistemų SKC valdymo metodai tinka ir RLS. Seniausias iš metodų ir labiausiai suprantamas yra krioklio metodas – sudarytas iš etapų sekos, kurioje vieno etapo rezultatas yra panaudojamas kaip kito etapo įvestis (Russel Key, 2002). Išskiriami penki etapai:

- projekto planavimas, tinkamumo analizė – aukšto lygio požiūris į sistemą, kuriame yra apibrėžti sistemos tikslai.
- reikalavimų apibrėžimas – sistemos tikslai patikslinami sistemos funkcijomis bei operacijomis kurias turi atlikti. Analizuojami galutino vartotojo poreikiai.
- sistemos dizainas – detalus sistemos aprašymas: sistemos funkcijos, struktūra, funkcionavimo aprašas ir panašiai.
- sistemos realizacija – paprastai susiveda į programinės įrangos prašymą.
- integravimas ir testavimas – apjungiami sistemos komponentai ir testuojami specialioje aplinkoje.

- diegimas – klientui įdiegiamas galutinis produktas, sistema pradedama realiai naudoti.
- palaikymas – ištaisomos likę smulkios sistemos klaidos arba nežymiai papildomas sistemos funkcionalumas.

Klasikinis krioklio metodas yra rizikingas - blogai specifikuotai sistemai specifikacijos patobulinimus galima įvesti tik po testavimo etapo. Todėl buvo įvestas papildomas prototipo sukūrimo etapas. Jis yra gretimas sistemos dizaino ir realizacijos etapams. Jo dėka sukūrus prototipą galima pastebėti klaidas ir iš karto taisyti sistemos specifikaciją.

Taikant krioklio metodą sistemos kūrimo ciklui jis praktiškai užima vieną ciklą, kai tuo tarpu modernesni metodai, tokie kaip spiralinis arba spartaus prototipo, turi keletą smulkesnių ciklų. Modernūs metodai paveldi krioklio metodo etapus.

Aprašant RLS neužtenka apibrėžti sistemos komponentus, ryšius tarp jų, būsenas ir panašius dalykus. Kadangi sistemos korektiškas funkcionavimas yra ne tik teisingi rezultatai bet ir laikas per kuri jie yra gaunami, todėl specifikuojant tokią sistemą yra būtina:

- nusakyti laiką per kurį sugeneruojamas sistemos atsakas į duotą išorinį impulsą,
- nustatyti sistemos blogiausią atvejį bei jo pasirodymo tikimybę,
- apibrėžti laiką, per kurį pasiruošia darbui (pasileidžia),
- apibrėžti laiką, per kurį sistema pradeda funkcionuoti teisingai po įvykusios klaidos (*angl. fault recovery time*)

Grafinės specifikavimo notacijos yra labai paplitę vien dėl savo vaizdumo ir paprastumo. Jos leidžia geriau suprasti sudėtingas problemas slepiant neesminius sistemos komponentų bruožus (Rational Software Corporation, 2003). Kuo paprasčiau sistema yra aprašoma tuo mažesnė tikimybė kad ji bus klaidingai suprasta. Viena jų yra UML (*angl. Unified Modeling Language*). Antroji jos versija leidžia aprašyti ir realaus laiko sistemas.

Nepaisant sistemų kūrimo ciklo pažangumo, sudarant realaus laiko sistemas susiduriama su įvairiomis problemomis iteracijų vykdymo metu, kas ilgina produkto pristatymą rinkai (*angl. time to maket*), didina projekto kaštus ir didina projekto žlugimo riziką. Problemos atsiranda sistemos prototipo etape. Prototipo modelis yra sudaromas neautomatizuotu būdu – procesas būna ilgas ir gali įsivelti klaidų. Jis yra modeliuojamas virtualioje aplinkoje ir virtualiame laike, todėl modeliavimo rezultatai neatspindi laikinių

modelio charakteristikų. Pagrindinės dabartinės situacijos priežastys:

- RLS specifikavimo kalbos, notacijos yra orientuotos sistemos funkcionavimą apibrėžti. Jos elementai nenumatyti transformacijom į modeliuojamus objektu, todėl negalima automatizuotai iš sistemos aprašymo gauti sistemos prototipą-modelį. Visą darbą atliekant rankiniu būdu sugaištamas laikas.
- Esančios CASE priemonės, kurios gali sugeneruoti iš sistemos specifikacijos daugumą išeities teksto nėra tiek universalios kad jų generuojamas išeities tekstas būtų suderinamas su visom SSA platformom ir nėra tiek pragmatiškas kad būtų suderintas su kuria nors SSA platforma.
- RLS modelis sudarymas tokių priemonių kaip petri tinklų ar baigtinių automatų pagalba sunkesnis ir užima daugiau laiko nei grafinių priemonių pagalba.
- Esančios grafinės RLS modeliavimo priemonės yra paprastos vartoti, lengvai suprantamos ir greitai išmokstamos, tačiau modeliuojama RLS yra sudaryta iš virtualių komponentų, kurie yra modeliuojami virtualioje aplinkoje ir virtualiame laike. Tokio modeliavimo pagalba galima nustatyti ar sistema duoda teisingus rezultatus, tačiau ar rezultatai bus pateikti laiku iš jos spręsti negalima.
- Modeliuojamų RLS komponentų perkėlimas rankiniu būdu į fizinę SSA platformą yra neefektyvus – užima daug laiko ir padidina klaidų atsiradimo riziką.

Darbe bandoma surasti sprendimą minėtoms problemoms kylančias pereinant nuo RLS aprašymo etapo prie sistemos prototipo arba sistemos realizacijos etapo. Siekiant patikimam RLS modeliui sudaryti ir patikimiems modeliavimo rezultatams gauti – siūloma RLS modeliavimą perkelti į fizinę SSA platformą arba, kitaip tariant, įvesti fizinę posistemę RLS modeliavimui. Tam atlikti reikia:

- Paprastos, gerai suprantamos ir lengvai išmokstamos grafinės aplinkos leidžiančios RLS modeliuojamą sistemą sudaryti, kad nereiktų sistemų projektuotojams papildomų įgūdžių gauti.
- Galimybės atvaizduoti modeliuojamą RLS paprasta komponentine struktūra, kur bet kuris struktūros elementas galėtų būti sudėtinis

komponentas - tas leistų sudaryti neribojamo sudėtingumo RLS vaizduojant tik esminius sistemos aspektus.

- Taikomosios SSA platformos ir priemonių, kurių dėka galima būtų leisti ir stebėti modeliuojamą RLS.
- Automatizuoto grafinio RLS modelio perkėlimo į fizinę aplinką galimybės, kad sistemos projektuotojas galėtų sutelkti daugiau dėmesio produkto dizainui, o ne jos modeliui sudaryti.
- Realaus laiko operacinės sistemos sudarančios abstrakcijos sluoksnį tarp modeliuojamos fizinės posistemės ir techninės įrangos – modeliuojama sistema būtų tiesiogiai nepriklausoma nuo SSA platformos.

Darbo tikslas yra pasiūlyti metodiką, kuri leistų iš sistemos struktūrinio aprašo sukurti posistemę modeliuojančią projektuojamą realaus laiko sistemą fizinėje aplinkoje. Šios metodikos dėka galima būtų automatizuoti posistemės sudarymo procesą.

2.3 Panašių sistemų analizė

EventStudio – tai EventHelix kompanijos CASE priemonė. Ji naudoja FDL kalbą (*angl. FDL – Feature Description Language*) sistemai specifikuoti. Priemonė specifikuojamą sistemą žinučių sekų (*angl. message sequence*), bendradarbiavimo (*angl. collaboration*), sąsajų diagramomis automatiškai atvaizduoja grafinėje aplinkoje. Priemonės privalumai:

- iš vieno duomenų failo galima automatiškai sugeneruoti įvairaus tipo diagramas: žinučių sekų, bendradarbiavimo, sąsajų,
- priemonė pati suranda specifikacijos klaidas.

Trūkumai:

- reikia mokytis naujos specifikavimo kalbos,
- negalima sistemos specifikacijos keisti grafinėje aplinkoje,
- neturi modeliavimo posistemės

Rational Rose Realtime – tai IBM kompanijos produktas. Tai plataus panaudojimo vizualaus programinės įrangos kūrimo aplinka. Aplinka palaiko UML 2.0 kalbos notacijas, tiekia priemones beveik visiems sistemos kūrimo etapams: specifikavimo, modeliavimo, realizavimo, testavimo. Privalumai:

- Naudojama standartinė UML notacija.
- Sistemą specifikuoti galima grafinėje aplinkoje.

- Iš specifikacijos galima sugeneruoti iki 70% sistemos taikomosios programinės įrangos išeities tekstų.

Trūkumai:

- Generuojami išeities tekstai nėra tiek universalūs, kad galima būtų juos panaudoti su kuria nors RTOS, nei nėra adaptuoti konkrečiai platformai.
- Generuojami išties tekstai yra karkasiniai, nepilni – programuotojas turi atskirai juos papildyti.

RTOS Builder – KTU studentų sukurta programa. Ji leidžia projektuoti neriboto sudėtingumo ir norimo abstrakcijos lygio RLS sistemas bei modeliuoti jų funkcionavimą. RLS projektuojama grafinėje aplinkoje naudojant sudėtinius arba paprastuosius RLS komponentus. Privalumai:

- Patogi ir greitai išmokstama vartotojo sąsaja.
- Paprastai ir greitai sukuriami savi komponentai.
- Geras ryšių tarp komponentų piešimas.
- Yra galimybė stebėti rezultatus grafiniu pavidalu.
- Prisitaikanti įrankių juosta.
- Vartotojui priimtina sudėtinių komponentų meniu hierarchija.
- Aukštas abstrakcijos lygis, leidžiantis aprašyti bet kokius veikiančius realiame laike komponentus iš bet kurios dalykinės srities.
- Programa yra parašyta Java mašinai, todėl ji gali veikti bet kokioje ją palaikančioje platformoje.

Trūkumai:

- modeliuojama RLS yra sudaryta iš virtualių komponentų, kurie yra modeliuojami virtualioje aplinkoje ir virtualiame laike. Tokio modeliavimo pagalba galima nustatyti ar sistema duoda teisingus rezultatus, tačiau ar rezultatai bus pateikti laiku iš jos spręsti negalima.
- Programa nesugeba iš modeliuojamos sistemos komponentų sugeneruoti taikomosios PĮ išeities tekstų, kurie realizuotų tuos komponentus.

Code Composer Studio – Texas Instruments programavimo aplinka *(angl. IDE – integrated development enviromnent*). Skirta taikomosios programinės įrangos kūrimui ir derinimui. Integruoti kompiliatoriai transliuoja assemblerio ir aukštojo lygio programavimo kalbos kodą.

Privalumai:

- Geras taikomosios programinės įrangos palaikymas – turtingos sisteminės bibliotekos.
- Taikomoji programinės įranga automatiškai integruojama į RLOS.
- Integruota derinimo programa.
- Galima naudoti C++ programavimo kalbą.
- Galima stebėti fizinės RLS funkcionavimą realiu laiku.

Trūkumai:

- Neturi grafinio RLS modelio sudarymo priemonių.
- Neturi sistemos projektavimo priemonių – skirtas tik RLS realizacijai.

2.4 Išvados

Vizualaus RLS projektavimo ir modeliavimo sistemos jau egzistuoja. Jos yra patogios ir lengvai suprantamos, tačiau RLS modeliuojamos virtualioje aplinkoje ir virtualiame laike. Toks modeliavimo metodas negarantuoja, kad projektuojama RLS funkcionuos teisingai fizinėje aplinkoje.

Sistemos skirtos taikomosios programinės įrangos kūrimui ir derinimui yra išbaigtos turtingos derinimo priemonėmis. Aukšto lygio programavimo kalbos bei sisteminių bibliotekų gausa leidžia kurti lanksčią vartotojo programinę įrangą. Tačiau tokios sistemos nėra pritaikytos nei RLS projektavimui nei modeliavimui.

Metodologija apibrėžianti virtualių RLS komponentų perkėlimą fizinę SSA platformą leistų sukurti priemonę, kuri tai automatiškai atliktų. Atsirastų papildomas abstrakcijos lygis žiūrint iš sistemos realizacijos aspekto. Paprastumas kokį suteikia grafinis RLS projektavimo būdas sumažintų klaidų kiekį bei projektavimo ciklo trukmę. Tai leistų ženkliai sutaupyti laiką bei jėgas kuriant RLS prototipą.

Toks požiūris į RLS modeliavimą, sistemos projektavimo procesą perkelia į kitą lygmenį – modeliavimu paremtą projektavima (*angl. MMD – Model Driven Developmnet*). Sistema sudaroma ir derinama patogioje grafinėje aplinkoje iš gerai patikrintų komponentų.

3 Teorinė dalis

3.1 Sistemos aprašymas

Sistemos modelis yra sistemos supaprastintas atvaizdavimas, aprašymas. Pilnai sistemai atvaizduoti naudojamas jos konceptualus trijų lygmenų modelis:

- Funkcinės struktūros lygmuo,
- Elgsenos lygmuo.
- Vykdomasis arba architektūros lygmuo.

Funkcinis modelis atvaizduoja sistemos struktūrą ir elgseną, neatsižvelgiant į kūrimo techniką, metodus ir apribojimus. Paprastai funkcija nusako esybę, kuri atlieka sistemos specifinį veiksmą. Sistemos panaudojimo atvejų veiksmų dekompozicija į funkcijas nusako sistemos „topologija“, kai tuo tarpu tarpinis lygmuo neatvaizduoja struktūrinės prigimties, o nusako sistemos elgseną. Ryšiai tarp funkcijų yra ypatingai patogus būdas realaus laiko sistemoms aprašyti (doc. E. Kazanavičius, 2004). Naudojami tik trys ryšių tipai:

- **Sinchronizacijos ryšys**, nusakomas įvykiu. Šis ryšys naudojamas nusakyti pilną arba dalinę tvarką, pagal kurią atitinkamos funkcijos aktyvuojamos bendrai sistemos funkcijai atlikti. Įvykių pasirodymas susijęs su intervencijomis, kurios nusako sistemos būsenų kaitą. Šio tipo ryšys susijęs su įvykiais valdomais (event – driven) taikymais.
- **Bendro kintamojo ryšys** naudojant būsenos kintamąjį. Šio tipo ryšys naudojamas kai keletas funkcijų gali tiesiogiai nuskaityti informacijos reikšmę arba gali momentaliai modifikuoti šio kintamojo reikšmę. Tai svarbus ryšys realaus laiko taikymuose aprašant susjungimus tarp fizinio proceso ir valdymo sistemos. Kai kurios funkcijos gali veikti aplinką ir turėti tikslią arba momentinę informaciją apie aplinkos būseną.
- **Informacijos perdavimo apsikeičiant pranešimais ryšys**. Šis ryšys naudojamas perduoti informacijai iš vienos funkcijos į kitą. Tai yra „gamintojo vartotojo ryšys“, kuris nusako informacijos keitimo savininką. Funkcija yra aktyvuojama tik tada kai reikalinga informacija yra prieinama ir perduodama pranešimo forma. Toks informacijos mainų ryšys funkcijų vykdymą laike. Tai naudojama duomenims valdomiems (*angl. data driven*) taikymams.

Funkcijos naudojamos sudaryti šį modelį gali būti elementarios arba kompleksinės. Sėkmingo tobulinimo dėka kompleksinė funkcija gali būti dekomponuota į struktūrine formą. Taigi funkcijos modelis yra hierarchinis (doc. E. Kazanavičius, 2004).

Elgsenos lygmens modelis nusako funkcijos indėlį aplinkai. Tai yra specifikavimo atvejis. Funkcija turi būti suprantama kaip įėjimo ir išėjimo transformacijos operatorius. Elgsenos modelis būna nuoseklus arba su ciklais. Aprašant juo nuoseklios elgsenos modeliai realizuojami baigtinių būsenų diagrama, Petri tinklu ar blokiniu algoritmu. Kiekviena funkcija atlieka vieną transformaciją tam tikru laiko momentu(doc. E. Kazanavičius, 2004). Yra du elgsenos tipai:

- Nuolatinė transformacija – funkcijos atlieka nuosekliai ir tolydžiai transformaciją be aktyvuojančių įėjimų į vykių ar pranešimų.
- Nenuolatinė transformacija – funkcijos yra sužadinamo kokio nors sistemos įvykio ar pranešimo.

Architektūros modelis nusako apratūrinę įrangą. Kiekviena elektroninė sistema susideda iš:

- Procesorių, kurie yra skirti informacijos transformavimo, apdorojimo ir loginių veiksmų atlikimui.
- Atminčių skirtų duomenų saugojimui.
- Komunikavimo mazgų, kaip tarpinių elementų duomenų perdavimui tarp komponentų.
- Valdymo įtaisų, kurie atlieka komponentų valdymą ir visos sistemos funkcijos realizaciją.

Šie keturių tipų tarpusavyje susieti technologiniai komponentai sudaro aparatūrinio aprašymo sprendimą, kurio pasekoje gaunama **vykdymo struktūra** arba **architektūra**.

Aukščiau aprašytas konceptualinis trijų lygmenų modelis aprašo sistemas aukštame abstrakcijos lygyje, siekiant gerai specifikuoti plačias paskirstytas realaus laiko sistemas. Mano metodikoje apsiribojama mažesnio mąsto taikymais – įterptinėms realaus laiko sistemoms. Iki šiol dėmesys buvo skiriamas sistemos modelio aprašymui. Dabar šis modelis praplečiamas programinių komponentų generavimo metodika, bet su tam tikrais apribojimais:

- Metodika yra taikoma tik dekomponuotam funkciniam modeliui – visos jį sudarančios funkcijos turi būti elementarios.
- Metodika apima funkcinio modelio struktūrą sudarytą iš veiksmų ir įvykių.
- Metodikoje nenumatytas sistemos paskirstymas tarp techninės įrangos komponentų – sistemos modeliavimas numatytas tik vienoje SSA platformoje.
- Metodika neapima architektūros modelio aprašymo ar jo komponentų tarpusavio ryšių.

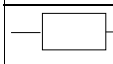
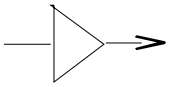
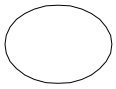
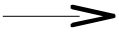
3.2 Elgsenos modelio realizavimo fizinėje aplinkoje koncepcija

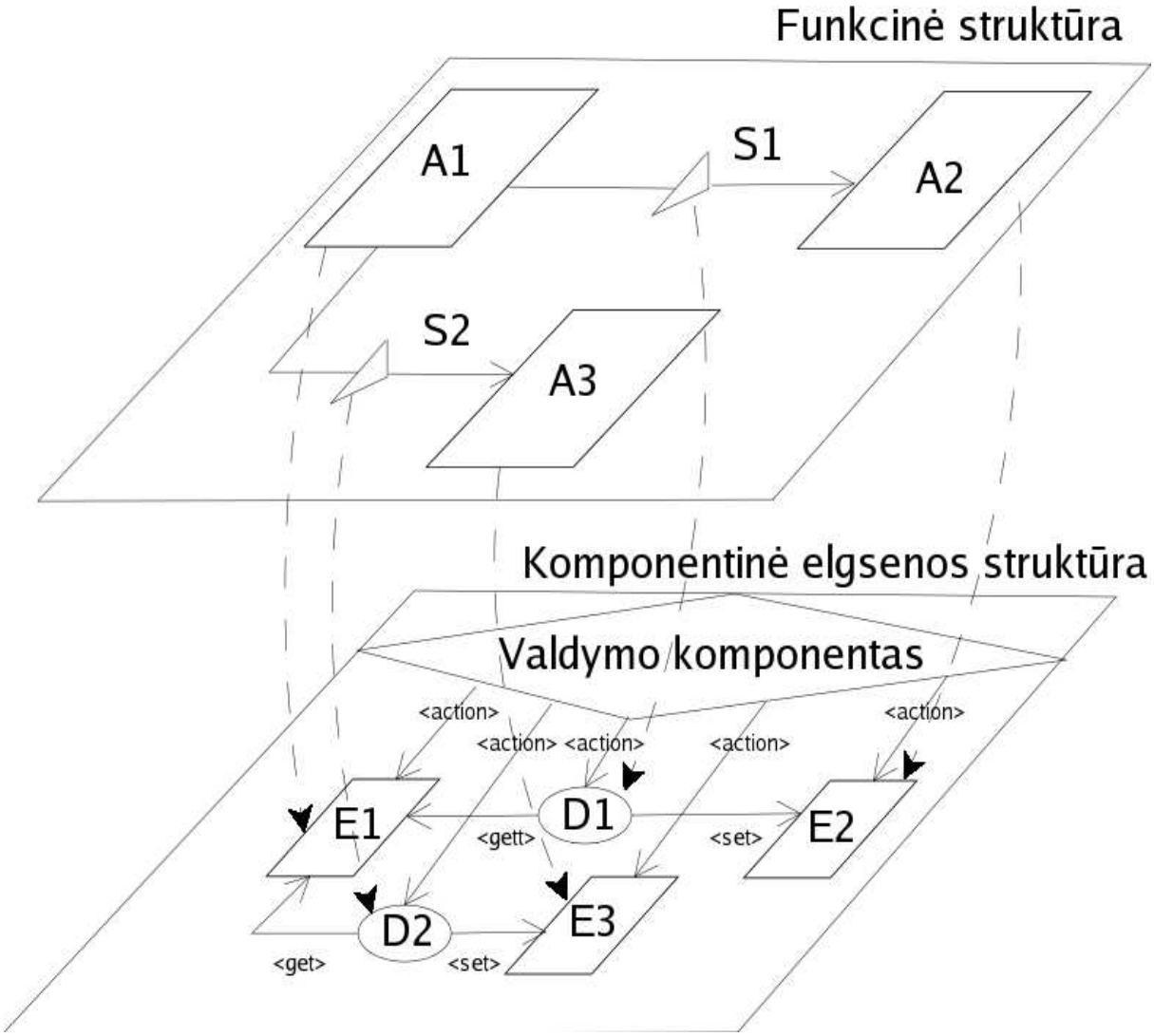
Sistemai modeliuoti pakanka elgsenos lygmens aprašo. Nuoseklios operacijos gali būti aprašomas komponentais, kurie atlieka atitinkamą veiksmą arba sukelia atitinkamą įvykį aprašytą funkciniame lygmenyje. Komponentas šiuo atveju yra suprantamas kaip duomenų struktūros ir su ja atliekamų operacijų visuma. Operacija suprantama kaip duomenų transformacija atliekama komponento ribose. Komponentai manipuliuojami pranešimais (*pastaba. sąvoka nėra tapatinama su pranešimais minimais funkciniame modelyje*). Pranešimai gali turėti parametrus. Naudojami dviejų rūšių pranešimai: reikalaujantys atsakymo ir nereikalaujantys atsakymo.

Mano koncepcija siūlo elgseną modeliuoti ne virtualioje, o fizinėje aplinkoje – architektūros lygmeny. Metodologija apibrėžia principus, kurių dėka galima būtų sukruti sistemos pilną elgsenos modelį tinkantį realizuoti fizinėje aplinkoje. Tiktais pilnas sistemos elgsenos modelis įmanomas realizuoti programavimo kalba naudojamą fizinėje SSA platformoje. Algoritminis sistemos elgsenos aprašas yra pilnas tik tuomet, kai jame kompleksinės operacijos yra detalizuotos.

Elgsenos modelis priklauso nuo funkcinio sistemos aprašo ir jo detalumo. Siūlomas elgsenos struktūros komponentų atitaikymas funkcinės struktūros elementams pavaizduotas 4 paveiksle. Naudojama funkcinė struktūra sudaryta iš funkcijų ir sinchronizacijos tipo ryšių. Funkcija nusakoma veiksmu, sinchronizacijos ryšys nusakomas įvykiu.

1. lentelė Sutartiniai pažymėjimai

Funkcinė struktūra	**Elgsenos struktūra**
 Veiksmas	 Funkcinis komponentas
 įvykis	 Ryšio komponentas
	 pranešimas



4 pav. Elgsenos komponentų struktūra

Funkcinis komponentas atlieka atitinkamą veiksmą (transformaciją) nusakytą funkciniam modelyje. Jo duomenų struktūra, įėjimų ir išėjimų kiekis, priklauso nuo transformacijos pobūdžio. Komponentas atlieka nenuolatinę transformaciją - veiksmas atliekamas tik gavus pranešimą iš valdymo komponento. Komponentas gali priimti trijų tipų pranešimus:

- <action> - jo priėmimo metu komponentas atlieka turimų iėjimų transformaciją į išėjimus,
- <set> - jo priėmimo metu nustatomos komponento įėjimų reikšmės nurodytos pranešime.
- <get> - pranešimo atsakymą sudaro komponento išėjimų reikšmės

Funkcinis komponentas neturi informacijos apie su juo susietus kitus komponentus – tam yra skirti ryšio komponentai. Algoritmas funkciniam komponentui sudaryti yra toks:

1. Nustatomas transformacijos įėjimų kiekis INPUT_COUNT ir transformacijos išėjimų kiekis OUTPUT_COUNT.
2. Nustatomas transformacijos tipas. Jei egzistuoja komponentas apibrėžtas tokio tipo funkcijai pereinam prie 5 punkto.
3. Apibrėžiamas komponentas duotai transformacijai atlikti toks, kad jo duomenų struktūrą sudaro:
   - *in_cnt* kintamasis nusakantis transformacijos įėjimų skaičių,
   - *out_cnt* kintamasis nusakantis transformacijos išėjimų skaičių,
   - *inputs[in_cnt]* kintamųjų vektorius saugojantis įėjimų reikšmes,
   - *outputs[out_cnt]* kintamųjų vektorius saugojantis išėjimų reikšmes.
4. Komponento apibrėžimas papildomas pranešimų apdorojimo procedūromis:
   - *get(output_num)* – procedūra aptarnaujanti <get> pranešimą, kuri grąžina *outputs[output_num]* kintamojo reikšmę.
   - *set(input_num, input_value)* – procedūra aptarnaujanti <set> pranešimą, kuri kintamajam *inputs[input_num]* nustato reikšmę *input_value*.
   - *action()* - procedūra aptarnaujanti <action> pranešimą, kuri atlieka duotą transformaciją su *inputs[in_cnt]* ir su *outpus[out_cnt]* kintamaisiais.
5. Sukuriamas komponento egzempliorius, kurio kintamųjų *in_cnt* ir *out_cnt* reikšmės yra nustatytos INPUT_COUNT ir OUTPUT_COUNT atitinkamai.

Ryšio komponentai atvaizduoja sinchronizacijos ryšių struktūrą. Šie komponentai priima tik <action> tipo pranešimą. Vidinės komponento struktūra saugo dviejų funkcinių komponentų nuorodas (šaltinio ir paskirties). <action> pranešimo tipo metu vieno komponento išėjimo reikšmė perduodama kito komponento įėjimui siunčiant jiems atitinkamai <get> ir <set> tipo pranešimus. Šio tipo komponentas sudaromas bendras aprašas, kuri sudaro:

- *source* – kintamasis nusakantis šaltinio komponentą.
- *destination* – kintamasis nusakantis paskirties komponentą.
- *output_num* – kintamasis nusakantis šaltinio komponento išėjimo indeksą.
- *input_num* – kintamasis nusakantis paskirties komponento įėjimo indeksą.
- *action()* - procedūra aptarnaujanti <action> pranešimą, kuri reikšmę *value*, gautą nusiuntus <get> (*get(output_num)*) pranešimą *source* komponentui, siunčia <set> (*set(input_num, value)*) pranešimą *destination* komponentui.

Kiekvienam sinchronizacijos ryšiui kuriamas ryšio komponento egzempliorius. Jam nustatoma, priklausomai nuo sudarytų funkcinių komponentų, kintamieji: *source*, *destination*, *output_num* bei *input_num*.

Valdymo komponentas – atlieka valdymo funkciją. Jis atlieka, priešingai nei kiti komponentai, transformacijas (pavyzdžiui CLOCK) periodiškai – siunčia nustatyta tvarka <actiion> tipo pranešimus kitiems komponentams. Šis komponentas gali būti statinis (statiškai sukuriamas). Komponentas turėti du sąrašus: funkcinių komponentų ir ryšio komponentų. Komponentai sudaromi pagal aukščiau aprašytas taisykles. Valdymo komponento veikiamas yra paremtas iteracinių veiksmų vykdymu. Iteracijų skaičius gali būti baigtinis arba begalinis. Kiekvienos iteracijos metu komponentas atlieka:

- Siunčia visiems sąraše esantiems ryšio komponentams <action> pranešimą.
- Siunčia visiems sąraše esantiems funkciniams komponentams <action> pranešimą.

Metodika neapima duomenų įvedimo ir išvedimo komponentų sudarymo – jie priklauso nuo fizinės aplinkos, kuriai sudaroma modeliavimo posistemė.

3.3 Išvados

Metodika numatyta programinių komponentų generavimas tik vienai fizinei platformai, tačiau koncepcija nekliudo kurti paskirstytas sistemas. Jei tik funkcinio medelio elementus sugrupuotume į blokus ir tiems blokams atskirai pritaikytume metodiką – tai elgsenos lygmenyje reiktų tam tikrus ryšio komponentus pakeisti atitinkamais „komunikavimo mazgais“, kad sistemą galima būtų modeliuoti paskirstytoje fizinėje aplinkoje.

Elgsenos modelio komponentų ir pranešimų sąvokos yra labai artimos objektinio programavimo sąvokom, todėl nesunkų jį būtų realizuoti objektiškai orientuota programavimo kalba. Dauguma SSA platformų kompiliatoriai palaiko objektišai orientuotas programavimo kalbas.

Modeliuojant fizinėje aplinkoje naudojamės kompiliatorių optimizacijomis. Techninės įrangos gamintojai tiekia kompiliatorius, kurių transliuojamos programos yra optimizuotos duotai platformai. Iš kitos pusės, susiduriame su konkrečios platformos apribojimais (pvz.: atminties, skaičiavimų tikslumo), dėl kurių iš anksto galime įvesti sistemoje pakeitimus.

4 Eksperimentinė dalis

4.1 Skaitmeninių signalų uždavinių klasės

SSA uždavinio tipas priklauso RLS panaudojimo atvejo. SSA uždaviniai yra skirstomi į klases. Pagrindinės uždavinių klasės yra:

- Klaidų taisymas – ši uždavinių klasė susijusi su klaidų aptikimu ir jų taisymu kai skaitmenis signalas yra transportuojamas (elektromagnetinių bangų pavidalu telefono linija, oru tarp palydovo ir antenos ar pan.) arba skaitomas iš tam tikros laikmenos pavyzdžiui kompaktinio disko. Siunčiamas signalas nuolat gali būti įtakojamas įvairiausių trikdžių susijusių su temperatūros pakitimais ar kitais atmosferiniais veiksniais, o tuo tarpu informaciją įrašytą į laikmeną gali gadinti įvairiausi laikmenos mechaniniai pažeidimai. Todėl labai svarbu yra aptikti neteisingai gautą ar nuskaitytą signalą. Paprasčiausias tokio pobūdžio uždavinių sprendimas yra susijęs su plačiai žinomu kontrolinių sumų skaičiavimu.
- Skaitmeninis filtravimas – šios klasės uždavinių tikslas yra dažniausiai atskirti ar išryškinti tik tam tikrą signalo dalį. Pavyzdžiui radijo-komunikacijos įrenginiuose montuojamuose sraigtasparniuose yra slopinamas sraigtasparnio variklio garsas kad galima būtų lengviau suprasti žmogaus balsą. Skaitmeninių filtrų privalumas tas, kad galima keisti filtro charakteristikas visai nekeičiant techninės įrangos ar netgi konstruoti „adaptyvius“ filtrus. Adaptyvus filtras tai toks filtras kuris gali pakeisti savo parametrus veikimo metu priklausomai nuo gaunamo signalo. Be to skaitmeninius filtrus gana patogu užrašyti matematinėmis išraiškomis. Patys filtrai yra skirstomi i du pagrindinius tipus FIR (Finite Impulse Response) ir IIR (Inifinite Impulse Response) filtrus. Pirmu atveju filtro skaičiuojamas rezultatas priklauso nuo baigtinio skaičiaus filtro įėjime patekusių reikšmių, kai antru atveju filtro skaičiuojamas rezultatas priklauso nuo visų reikšmių patekusių į filtro įėjimą. Egzistuoja adaptyvaus filtro sąvoka kuri gali papildyti vieno ar kito tipo filtro apibrėžimą. Adaptyvūs filtrai gali patys keisti savo parametrus

priklausomai nuo buvusių reikšmių.

- Efektyvus kodavimas/suspaudimas – Duomenų perdavimui ar saugojimui naudojami du duomenų suspaudimo tipai: su praradimais ir be praradimų. Pirmu atveju duomenys atkuriami tiksliai tokie patys kiek buvo prieš užkoduojant. Antru atveju atkurtų duomenų tikslumas yra aukojamas užkodavimo efektyvumo labui. Kaip ir minėjau ši uždavinių klasė yra dažnai sprendžiama telekomunikacijose bei skaitmeninių vaizdų saugojimui.

Eksperimentams atlikti buvo pasirinkta skaitmeninio filtravimo uždavinių klasė- FIR (*angl. Finite Response Filter*) tipo filtrų modeliavimą. Filtras užrašomas formule:

$$y(n) = a_0x(n) + a_1x(n - 1) + a_2x(n - 2), \quad [10: 98]$$

kur x(n) tai įėjimo signalas laiko momentu n, o a_0, a_1, a_2 filtro koeficientai.

Koeficientai buvo pasirinkti:

$a_0 = 0.25$

$a_1 = 0.5$

$a_2 = 0.25$

Reikia pabrėžti kad eksperimentas buvo atliekamas metodologijos teisingumą patikrinti, o ne filtro efektyvumą tirti.

4.2 Techninė ir programinė įranga

Sistemos modeliavimui virtualioje aplinkoje buvo pasirinkta RTOS Builder modeliavimo sistema. RTOS Builder – KTU studentų sukurta programa. Ji leidžia projektuoti neriboto sudėtingumo ir norimo abstrakcijos lygio RLS sistemas bei modeliuoti jų funkcionavimą. RLS projektuojama grafinėje aplinkoje naudojant sudėtinius arba paprastuosius RLS komponentus. Teoriniams skaičiavimams atlikti buvo naudojamas MATHLAB paketas (versija 6.0). Šis paketas atlieka skaičiavimus dideliu tikslumu.

Sistemos modeliavimui fizinėje aplinkoje buvo pasirinktas Texas Instruments kompanijos gaminys TMS320C6711 DSK. Gaminį sudaro SSP plokštė bei Code Composer Studio programavimo aplinka. Plokštėje esantis procesorius naudojant 150 MHz išorinį dažnį ir nenaudojant vidinio daugiklio iš keturių gali atlikti 900 milijonų slankaus kablelio operacijų per sekundę.

Code Composer Studio programavimo aplinka skirta TMS320 šeimos SSA platformomis. Integruotas kompiliatorius transliuoja tiek asemblerio išeities tekstus, tiek C/C++ programavimo kalbos išeities tekstus. Gausu įvairiausios paskirties bibliotekų kurių dėka galima sudaryti programinę įranga pritaikomą visai SSP procesorių šeimai. Integruota derinimo priemonė leidžia stebėti programos funkcionavimą realiu laiku. Kartu su programavimo aplinka tiekiamas specialus modulis – DSP Bios. Tai galingas sistemos modulis – atlieka realaus laiko operacinės sistemos funkcijas (tiekia sąsają su periferiniams įrenginiams naudoti, valdo procesoriaus laiko paskirstymą tarp vartotojo programos užduočių, reguliuoja pertraukimus ir pan.).

4.3 Eksperimentas

Eksperimentas buvo atliekamas tam, kad patikrinti ar, pasiūlytos metodikos dėka sukurta fizinė modeliavimo posistemė funkcionuoja korektiškai. Fizinės modeliavimo posistemės funkcionavimas buvo laikomas korektišku, kai jos skaičiavimų rezultatai nesiskiria daugiau nei 0.1% nuo teorinių. Teoriniais rezultatais buvo laikomi rezultatai gauti modeliuojant uždavinį MATHLAB aplinkoje. Be to, modeliuojant RTOS Builder aplinkoje gauti rezultatai buvo lyginami su teoriniais.

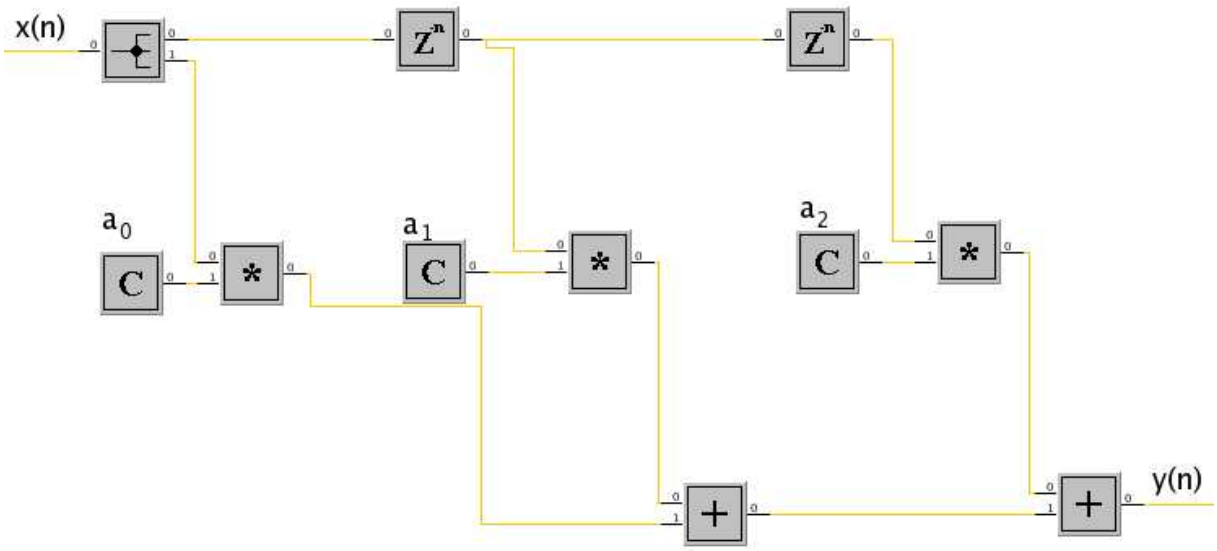
Įėjimų reikšmės buvo generuojamos. RTOS Builder aplinkoje įėjimo reikšmių generavimui buvo panaudotas standartinis komponentas. SSA platformai buvo sudarytas panašus komponentas. MATHLAB aplinkoje duomenų generavimas buvo įtrauktas į filtro algoritminį aprašą. Eksperimento rezultatus sudaro dvi dalys:

- Sugeneruotų įėjimo reikšmių palyginimas – tiriamas skirtumas tarp teorinių įėjimo reikšmių ir generuojamų duomenų SSA platformoje bei RTOS Builder aplinkoje.
- Apskaičiuotų reikšmių palyginimas – tiriamas skirtumas tarp teoriškai apskaičiuotų reikšmių ir rezultatų gautų SSA platformoje bei RTOS Builder aplinkoje.

Modeliuojamam FIR filtrui buvo paduodamas įėjimo signalas x(n), kur

$x(n)=sin(0.02\pi n)$, $n=0, 1, ..., 100$

Kadangi RTOS Builder aplinkoje laikas irgi modeliuojamas, todėl x(n) atskaičiuojama nepriklausomai nuo laiko. FIR filtro modelis RTOS Builder sistemoje pavaizduotas 5 paveiksle. Naudojantis aprašyta šiame darbe metodika buvo sudaryta programa modeliuojanti FIR filtrą TMS320C6711 platformoje. Be to buvo sudaryti komponentai generuojantys įėjimo signalą.



5 pav. FIR filtras RTOS Builder sistemoje

Modeliavimo rezultatams sulyginti RTOS Builder sistema bei programa sudaryta SSA platformai buvo papildyta duomenų išvedimo funkcijom. RTOS Builder aplinkoje generuojami rezultatai yra sudėti 2 lentelėje. Naudojami pažymėjimai:

$x_{RTOS}(n)$ – tai x(n) generuojamas RTOS builder aplinkoje,

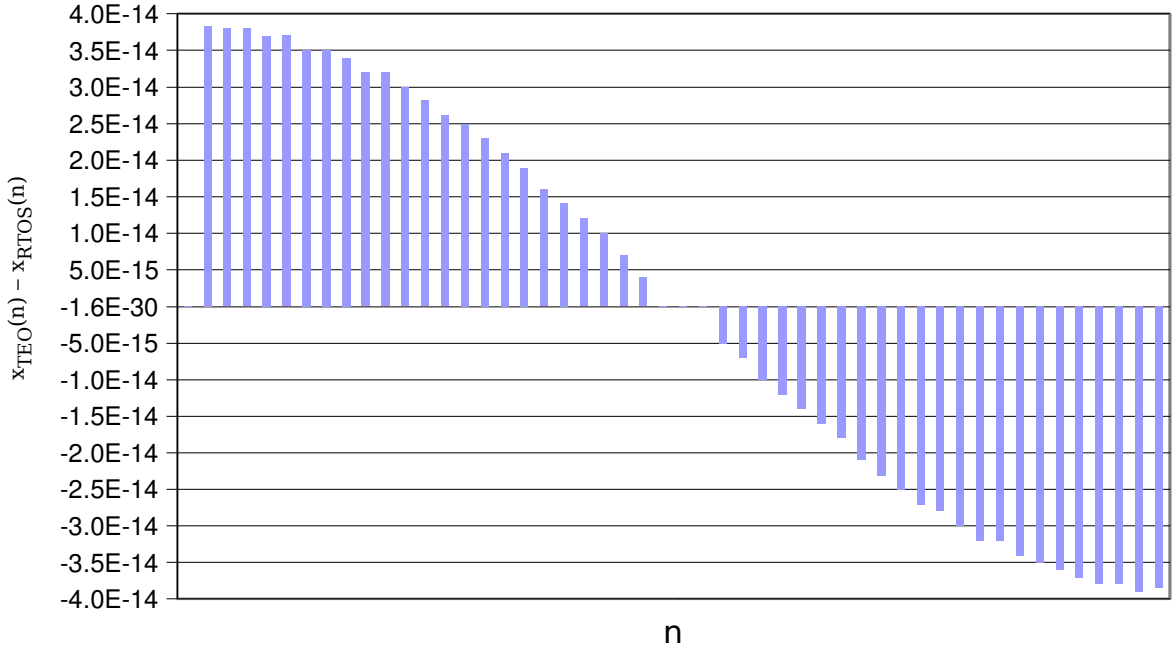
$x_{TEO}(n)$ – tai teorinės x(n) reikšmės skaičiuojamos MATHLAB paketo pagalba.

2. lentelė Sugeneruoti įėjimo duomenys

n	$x_{TEO}(n)$	$x_{RTOS}(n)$	$x_{TEO}(n) - x_{RTOS}(n)$
0	0.000000000000000	0.000000000000000	0.00E+00
1	0.062790519529313	0.062790519529275	3.84E-14
2	0.125333233564304	0.125333233564266	3.80E-14
3	0.187381314585724	0.187381314585686	3.80E-14
4	0.248689887164854	0.248689887164817	3.70E-14
5	0.309016994374947	0.309016994374910	3.70E-14
6	0.368124552684677	0.368124552684642	3.50E-14
7	0.425779291565072	0.425779291565037	3.50E-14
8	0.481753674101715	0.481753674101681	3.40E-14
9	0.535826794978996	0.535826794978964	3.20E-14
10	0.587785252292473	0.587785252292441	3.20E-14
11	0.637423989748689	0.637423989748659	3.00E-14
12	0.684547105928688	0.684547105928660	2.81E-14
13	0.728968627421411	0.728968627421385	2.61E-14
14	0.770513242775789	0.770513242775764	2.50E-14
15	0.809016994374947	0.809016994374924	2.31E-14
16	0.844327925502015	0.844327925501994	2.10E-14
17	0.876306680043863	0.876306680043844	1.90E-14
18	0.904827052466019	0.904827052466003	1.60E-14
19	0.929776485888251	0.929776485888237	1.41E-14
20	0.951056516295153	0.951056516295141	1.20E-14
21	0.968583161128631	0.968583161128621	9.99E-15
22	0.982287250728688	0.982287250728681	6.99E-15
23	0.992114701314477	0.992114701314473	4.00E-15
24	0.998026728428271	0.998026728428269	0.00E+00
25	1.000000000000000	1.000000000000000	0.00E+00
26	0.998026728428271	0.998026728428273	0.00E+00
27	0.992114701314477	0.992114701314482	-5.00E-15
28	0.982287250728688	0.982287250728695	-6.99E-15
29	0.968583161128630	0.968583161128640	-9.99E-15
30	0.951056516295153	0.951056516295165	-1.20E-14
31	0.929776485888251	0.929776485888265	-1.40E-14
32	0.904827052466019	0.904827052466035	-1.61E-14
33	0.876306680043863	0.876306680043881	-1.80E-14
34	0.844327925502014	0.844327925502035	-2.10E-14
35	0.809016994374946	0.809016994374969	-2.31E-14
36	0.770513242775788	0.770513242775813	-2.50E-14
37	0.728968627421410	0.728968627421437	-2.71E-14
38	0.684547105928687	0.684547105928715	-2.80E-14
39	0.637423989748688	0.637423989748718	-3.00E-14
40	0.587785252292471	0.587785252292503	-3.20E-14

Teoriškai paskaičiuoti įėjimo duomenys skiriasi nuo reikšmių sugeneruotų virtualioje aplinkoje. Skirtumas nėra pastovus, kinta intervale ($-3.2 \cdot 10^{-14}$, $3.84 \cdot 10^{-14}$). Lentelėje nėra rodomi visi rezultatai, tačiau likusių generuojamų duomenų skirtumas patenka į minėtą intervalą.

Generuojamų duomenų skirtumas pateikiamas grafiškai 6 paveiksle.



6 pav. Teorinių ir RTOS Builder aplinkos generuojamų duomenų skirtumas

Duomenys generuojami SSA platformoje yra sudėti 3 lentelėje. Naudojami pažymėjimai:

$x_{SSA}(n)$ – tai x(n) generuojamas SSA platformoje,

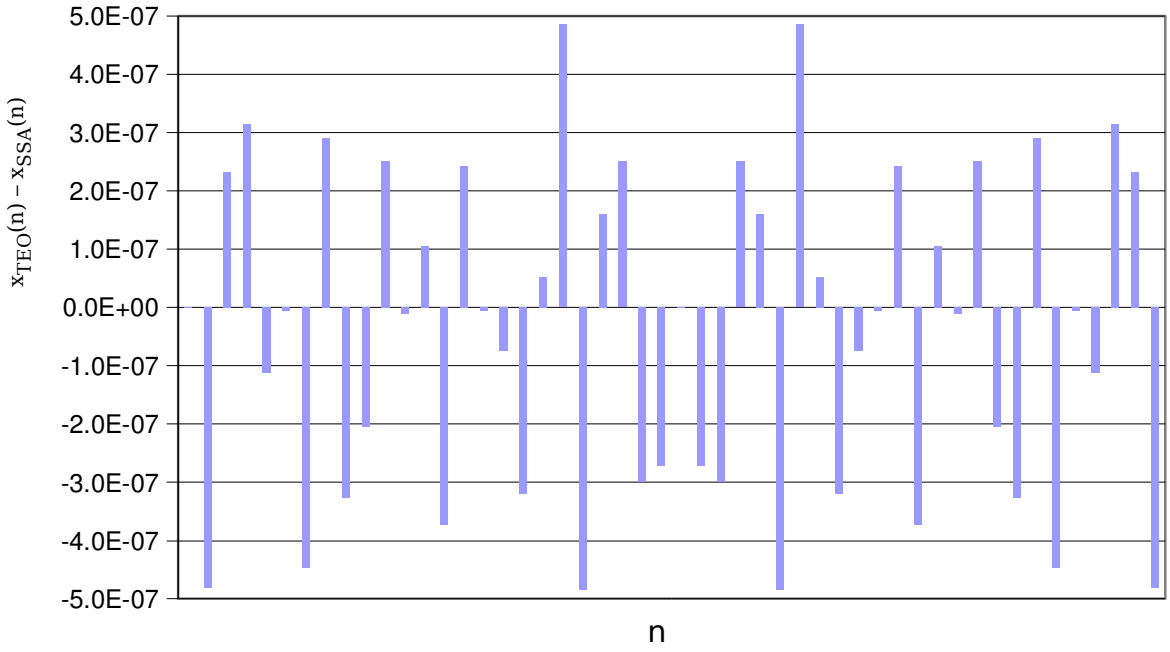
$x_{TEO}(n)$ – tai teorinės x(n) reikšmės skaičiuojamos MATHLAB paketo pagalba.

3. lentelė Sugeneruoti įėjimo duomenys

n	$x_{TEO}(n)$	$x_{SAA}(n)$	$x_{TEO}(n) - x_{SSA}(n)$
0	0.00000000	0.000000	0.00E+00
1	0.06279052	0.062791	-4.80E-07
2	0.12533323	0.125333	2.34E-07
3	0.18738131	0.187381	3.15E-07
4	0.24868989	0.248690	-1.13E-07
5	0.30901699	0.309017	-5.63E-09
6	0.36812455	0.368125	-4.47E-07
7	0.42577929	0.425779	2.92E-07
8	0.48175367	0.481754	-3.26E-07
9	0.53582679	0.535827	-2.05E-07
10	0.58778525	0.587785	2.52E-07
11	0.63742399	0.637424	-1.03E-08
12	0.68454711	0.684547	1.06E-07
13	0.72896863	0.728969	-3.73E-07
14	0.77051324	0.770513	2.43E-07
15	0.80901699	0.809017	-5.63E-09
16	0.84432793	0.844328	-7.45E-08
17	0.87630668	0.876307	-3.20E-07
18	0.90482705	0.904827	5.25E-08
19	0.92977649	0.929776	4.86E-07
20	0.95105652	0.951057	-4.84E-07
21	0.96858316	0.968583	1.61E-07
22	0.98228725	0.982287	2.51E-07
23	0.99211470	0.992115	-2.99E-07
24	0.99802673	0.998027	-2.72E-07
25	1.00000000	1.000000	0.00E+00
26	0.99802673	0.998027	-2.72E-07
27	0.99211470	0.992115	-2.99E-07
28	0.98228725	0.982287	2.51E-07
29	0.96858316	0.968583	1.61E-07
30	0.95105652	0.951057	-4.84E-07
31	0.92977649	0.929776	4.86E-07
32	0.90482705	0.904827	5.25E-08
33	0.87630668	0.876307	-3.20E-07
34	0.84432793	0.844328	-7.45E-08
35	0.80901699	0.809017	-5.63E-09
36	0.77051324	0.770513	2.43E-07
37	0.72896863	0.728969	-3.73E-07
38	0.68454711	0.684547	1.06E-07
39	0.63742399	0.637424	-1.03E-08
40	0.58778525	0.587785	2.52E-07

Teoriškai paskaičiuoti įėjimo duomenys skiriasi nuo reikšmių sugeneruotų SSA platformoje. Skirtumas nėra pastovus, kinta intervale ($-4.8 \cdot 10^{-7}$, $4.4 \cdot 10^{-7}$). Skirtumas yra didesnis nei lyginant su RTOS Builder generuojamais duomenimi. Tai gali būti susiję su išvedimo funkcijos apribojimais.

Generuojamų duomenų skirtumas grafiškai pavaizduotas 7 paveiksle.



Antroje dalyje pateikiami modeliuojamo FIR filtro skaičiavimo rezultatai. Filtras užrašomas formule:

$y(n) = a_0x(n) + a_1x(n - 1) + a_2x(n - 2),$ [10: 98]

kur x(n) tai įėjimo signalas laiko momentu n, o a_0, a_1, a_2 filtro koeficientai.

Koeficientai buvo pasirinkti:

$a_0 = 0.25$

$a_1 = 0.5$

$a_2 = 0.25.$

Teoriniai skaičiavimai atliekami MATHLAB paketo pagalba. Suskaičiuotas skirtumas tarp teoriškai paskaičiuotų rezultatų ir rezultatų gautų RTOS Builder aplinkoje pateiktas 4 lentelėje. Naudojami pažymėjimai:

$y_{RTOS}(n)$ – tai y(n) skaičiuojamas RTOS builder aplinkoje,

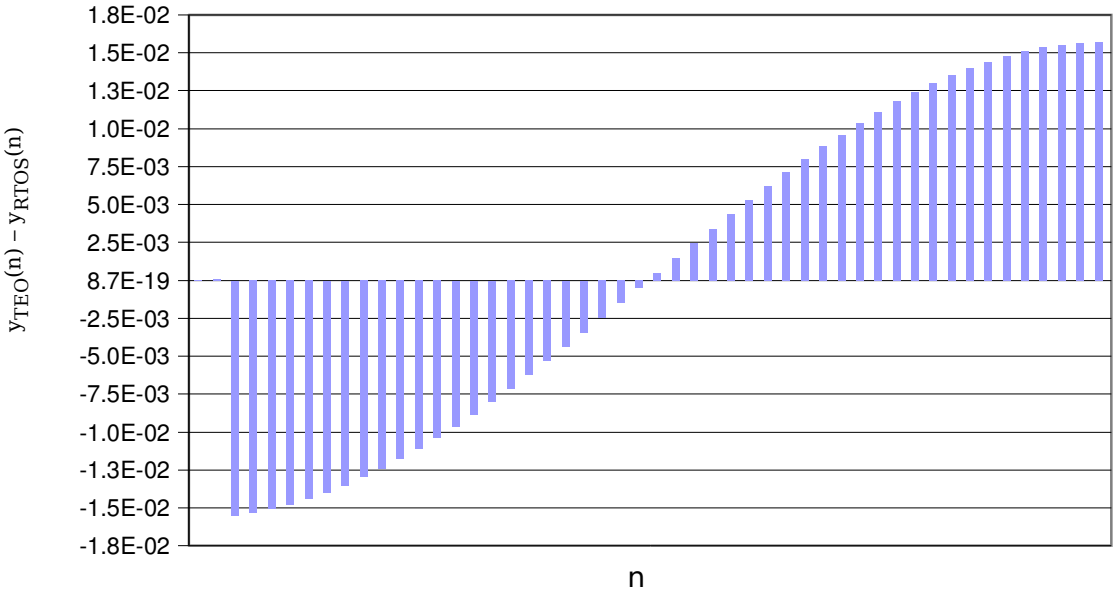
$y_{TEO}(n)$ – tai teorinės y(n) reikšmės skaičiuojamos MATHLAB paketo pagalba.

4. lentelė Skaičiavimų rezultatai

n	$y_{TEO}(n)$	$y_{RTOS}(n)$	$y_{TEO}(n) - y_{RTOS}(n)$
0	0.00000000	0.00000000	0.00000000
1	0.01569763	0.01563568	0.00006195
2	0.06272857	0.07824059	-0.01551202
3	0.12520958	0.14053672	-0.01532714
4	0.18719644	0.20227821	-0.01508178
5	0.24844452	0.26322141	-0.01477689
6	0.30871211	0.32312579	-0.01441368
7	0.36776135	0.38175494	-0.01399360
8	0.42535920	0.43887748	-0.01351828
9	0.48127836	0.49426797	-0.01298961
10	0.53529813	0.54770781	-0.01240968
11	0.58720532	0.59898610	-0.01178078
12	0.63679508	0.64790046	-0.01110538
13	0.68387171	0.69425786	-0.01038615
14	0.72824940	0.73787534	-0.00962594
15	0.76975303	0.77858076	-0.00882773
16	0.80821879	0.81621348	-0.00799469
17	0.84349488	0.85062497	-0.00713009
18	0.87544208	0.88167944	-0.00623736
19	0.90393432	0.90925433	-0.00532001
20	0.92885914	0.93324080	-0.00438166
21	0.95011817	0.95354419	-0.00342602
22	0.96762752	0.97008438	-0.00245686
23	0.98131809	0.98279610	-0.00147801
24	0.99113585	0.99162916	-0.00049332
25	0.99704204	0.99654872	0.00049332
26	0.99901336	0.99753536	0.00147801
27	0.99704204	0.99458518	0.00245686
28	0.99113585	0.98770982	0.00342602
29	0.98131809	0.97693643	0.00438166
30	0.96762752	0.96230751	0.00532001
31	0.95011817	0.94388081	0.00623736
32	0.92885914	0.92172904	0.00713009
33	0.90393432	0.89593963	0.00799469
34	0.87544208	0.86661435	0.00882773
35	0.84349488	0.83386894	0.00962594
36	0.80821879	0.79783264	0.01038615
37	0.76975303	0.75864765	0.01110538
38	0.72824940	0.71646862	0.01178078
39	0.68387171	0.67146202	0.01240968
40	0.63679508	0.62380547	0.01298961

Virtualioje aplinkoje modeliuojamo FIR filtro skaičiuojamos reikšmės skiriasi nuo teorinių. Skirtumas nėra pastovus, kinta intervale (-0.016, 0.013). Jis yra žymiai didesnis nei duomenų generavimo skirtumas. Tai gali įtakoti tarpinių rezultatų apvalinimo operacijos.

Rezultatų skirtumas grafiškai pavaizduotas 8 paveiksle.



8 pav. Teorinių ir RTOS Builder aplinkos modeliavimo rezultatų skirtumas

Toliau skaičiuojamas tarp teorinių rezultatų ir rezultatų gautų SSA platformoje skirtumas, kuris pateikiamas 5 lentelėje. Naudojami pažymėjimai:

$y_{SSA}(n)$ – tai y(n) reikšmės skaičiuojamas fizinėje SSA platformoje,

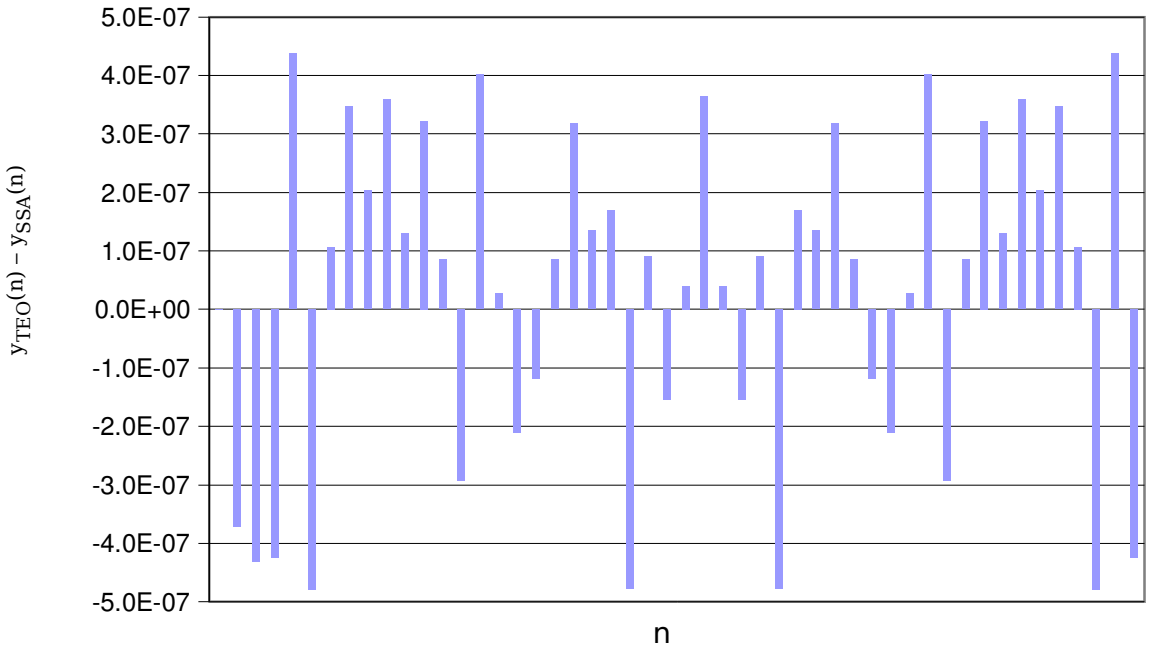
$y_{TEO}(n)$ – tai teorinės y(n) reikšmės skaičiuojamos MATHLAB paketo pagalba.

5. lentelė Skaičiavimų rezultatai

n	$y_{TEO}(n)$	$y_{SSA}(n)$	$y_{TEO}(n) - y_{SSA}(n)$
0	0.00000000	0.000000	0.0E+0
1	0.01569763	0.015698	-3.7E-7
2	0.06272857	0.062729	-4.3E-7
3	0.12520958	0.125210	-4.2E-7
4	0.18719644	0.187196	4.4E-7
5	0.24844452	0.248445	-4.8E-7
6	0.30871211	0.308712	1.1E-7
7	0.36776135	0.367761	3.5E-7
8	0.42535920	0.425359	2.0E-7
9	0.48127836	0.481278	3.6E-7
10	0.53529813	0.535298	1.3E-7
11	0.58720532	0.587205	3.2E-7
12	0.63679508	0.636795	8.4E-8
13	0.68387171	0.683872	-2.9E-7
14	0.72824940	0.728249	4.0E-7
15	0.76975303	0.769753	2.7E-8
16	0.80821879	0.808219	-2.1E-7
17	0.84349488	0.843495	-1.2E-7
18	0.87544208	0.875442	8.5E-8
19	0.90393432	0.903934	3.2E-7
20	0.92885914	0.928859	1.4E-7
21	0.95011817	0.950118	1.7E-7
22	0.96762752	0.967628	-4.8E-7
23	0.98131809	0.981318	9.1E-8
24	0.99113585	0.991136	-1.5E-7
25	0.99704204	0.997042	4.0E-8
26	0.99901336	0.999013	3.6E-7
27	0.99704204	0.997042	4.0E-8
28	0.99113585	0.991136	-1.5E-7
29	0.98131809	0.981318	9.1E-8
30	0.96762752	0.967628	-4.8E-7
31	0.95011817	0.950118	1.7E-7
32	0.92885914	0.928859	1.4E-7
33	0.90393432	0.903934	3.2E-7
34	0.87544208	0.875442	8.5E-8
35	0.84349488	0.843495	-1.2E-7
36	0.80821879	0.808219	-2.1E-7
37	0.76975303	0.769753	2.7E-8
38	0.72824940	0.728249	4.0E-7
39	0.68387171	0.683872	-2.9E-7
40	0.63679508	0.636795	8.4E-8

SSA platformoje modeliuojamo FIR filtro skaičiuojamos reikšmės skiriasi nuo teorinių. Skirtumas nėra pastovus, kinta intervale ($-4.9 \cdot 10^{-7}$, $4.5 \cdot 10^{-7}$). Jis šiek tiek didesnis nei generuojamų duomenų skirtumas, tačiau rezultatai yra tikslesni nei modeliuojamo uždavinio virtualioje aplinkoje.

Rezultatų skirtumas grafiškai pavaizduotas 9 paveiksle.



9 pav. Teorinių ir SSA platformoje modeliavimo rezultatų skirtumas

4.4 Išvados

Eksperimento metu pastebėta, jog signalo reikšmės generuojamas fizinėje aplinkoje skiriasi nuo teorinių. Apskaičiuotas skirtumas yra didesnis nei skirtumas tarp teorinių ir virtualioje aplinkoje sugeneruotų reikšmių. Rezultatų išvedimui naudojama primityvi SSA platformos bibliotekos funkcija. Ji nėra pritaikyta dideliu tikslumu slankaus kablelio skaičiams versti dešimtainiu kodu, todėl reikšmės gali būti apvalinamos.

Rezultatai skaičiuojami fizinėje aplinkoje taip nesutampa su teoriniais. Rezultatų ir generuojamų duomenų nesutapimai yra vienodos eiles. Tai gali būti sukelta tos pačios išvedimo funkcijos apribojimų. Netikslumai yra pakankamai nedideli – svyruoja tarp $-4.9 \cdot 10^{-7}$ ir $4.5 \cdot 10^{-7}$, todėl galima teigti, kad fizinės modeliavimo posistemės skaičiavimo rezultatai yra teisingi.

Pastebėta, jog RTOS Builder modeliaviomo sistemoje generuojami duomenys yra tikslesni nei rezultatai. Tai gali būti sukelta tarpiniuose skaičiavimuose atliekamų apvalinimo operacijų. Eksperimento rezultatai rodo, kad RTOS Builder modeliavimo sistemoje yra vietos patobulinimams.

Modeliavimo rezultatai gauti fizinėje aplinkoje yra teisingi, todėl darbe siūloma metodika yra teisinga. Jos pagrindu galima sudaryti priemonę, kuri automatiškai generuotų fizinę modeliavimo posistemę. Be to Texas Instruments tiekia išorines bibliotekas, kurių dėka SSA platformą galima valdyti iš vartotojo programinės įrangos (galimos komandos: vykdomojo failo užkrovimas, paleidimas, vykdymo sustabdymas ir pan.), todėl galima būtų į modeliavimo posistemės generavimo priemones įtraukti programos užkrovimo funkciją.

5 Išvados

Metodika apibrėžianti virtualių RLS komponentų perkėlimą fizinę SSA platformą leistų sukurti priemonę, kuri tai automatiškai atliktų. Atsirastų papildomas abstrakcijos lygis žiūrint iš sistemos realizacijos aspekto. Paprastumas kokį suteikia grafinis RLS projektavimo būdas sumažintų klaidų kiekį bei projektavimo ciklo trukmę. Tai leistų sutaupyti laiką bei pastangas kuriant RLS prototipą.

Toks požiūris į RLS modeliavimą, sistemos projektavimo procesą perkelia į kitą lygmenį – modeliavimu paremtą projektavima (*angl. MMD – Model Driven Developmnet*). Sistema sudaroma ir derinama patogioje grafinėje aplinkoje iš gerai patikrintų komponentų.

Konceptualinis trijų lygmenų modelis [4] aprašo sistemas aukštame abstrakcijos lygyje, siekiant gerai specifikuoti plačias paskirstytas realaus laiko sistemas. Mano metodikoje apsiribojama mažesnio mąsto taikymais – įterptinėms realaus laiko sistemoms. Iki šiol dėmesys buvo skiriamas sistemos modelio aprašymui. Dabar šis modelis praplečiamas programinių komponentų generavimo metodika, bet su tam tikrais apribojimais.

Metodika numatyta programinių komponentų generavimas tik vienai fizinei platformai, tačiau koncepcija nekliudo kurti paskirstytas sistemas. Jei tik funkcinio medelio elementus sugrupuotume į blokus ir tiems blokams atskirai pritaikytume metodiką – tai elgsenos lygmenyje reiktų tam tikrus ryšio komponentus pakeisti atitinkamais „komunikavimo mazgais“, kad sistemą galima būtų modeliuoti paskirstytoje fizinėje aplinkoje.

Elgsenos modelio komponentų ir pranešimų sąvokos yra labai artimos objektinio programavimo sąvokom, todėl nesunkų jį būtų realizuoti objektiškai orientuota programavimo kalba. Dauguma SSA platformų kompiliatoriai palaiko objektišai orientuotas programavimo kalbas.

Darbe siūloma metodika yra eksperimentiškai patikrinta. Jos pagrindu galima sudaryti priemonę, kuri automatiškai generuotų fizinę modeliavimo posistemę. Be to Texas Instruments tiekia išorines bibliotekas, kurių dėka SSA platformą galima valdyti iš vartotojo programinės įrangos (galimos komandos: vykdomojo failo užkrovimas, paleidimas, vykdymo sustabdymas ir pan.), todėl galima būtų į modeliavimo posistemės generavimo priemones įtraukti programos užkrovimo funkciją.

6 Literatūra

[1] Chasing R. DSP Applications Using C and the TMS320C6x DSK. New York: John Wiley & Sons, Inc. 2002. 329p.

[2] Garška E. Signalų apdorojimo įtaisai - analitiniai pagrindai [interaktyvus]. - [žiūrėta 2004-02-08]. Preiga per internetą: <http://rfk.ff.vu.lt/doc/dielektrikai.pdf>

[3] IBM Rational Rose RealTime: A guide for evaluation and review. Rational Software Corporation [interaktyvus]. 2003 Birželis [žiūrėta 2003-12-10]. Preiga per internetą: <http://www-106.ibm.com/developerworks/rational/library/622.html>

[4] Jiang C. Survey on Realtime Multiprocessor Scheduling [interaktyvus]. - [žiūrėta 2003-05-22]. Prieiga per internetą: <http://www-courses.cs.uiuc.edu>

[5] Kay R. System Development Life Cycle [interaktyvus], - [žiūrėta 2003-12-10], Preiga per internetą: <http://www.computerworld.com>

[6] Kalinsky David. Support for High Availability Applications in New-Generation RTOS's [interaktyvus]. - [žiūrėta 2003-09-21]. Prieiga per internetą: <http://www.realtime-info.be/magazine/01q3/2001q3_p047.pdf>

[7] Kazanavičius E. Signalų apdorojimo sistemos, Kaunas: 2004. 60p.

[8] Kirner R. Extending Optimising Compilation to Support Worst-Case Execution Time Analysis [interaktyvus], - [žiūrėta 2004-02-08]. Preiga per internetą: <http://www.vmars.tuwien.ac.at>

[9] Lehoczky J. On line scheduling for Hard-Real-Time Systems // The Journal of Real-Time Systems. -, Nr. 1, p. 29

[10] Marven C.; Ewers G. A Simple Approach to Digital Signal Processing. New York: John Wiley & Sons, Inc. 1996. 236p.

[11] Sarris A. K. Integration Toolkit and Methods, Corporate Data Integration Tools [interaktyvus]. - [žiūrėta 2004-04-24]. Preiga per internetą: <http://www.idef.com/idef5.html>

[12] Tindell K.; Clark J. Holistic Schedulability Analysis for Distributed Real-Time Systems // Real-Time Systems Journal. - , Nr. 9, p. 32

7 Darbo santrauka anglų kalba

A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred. Real-time system consist specialized hardware an software components. Nowadays, systems are so big and complex that teams of architects, analysts, programmers, testers and users must work together to create reliable real-time system. To manage this, a number of system development life cycle models have been created.

System development life cycle refers to a methodology for developing systems. It provides a consistent framework of tasks and deliverables needed to develop systems. System development consist stages, that are common for all models: project planning, requirements definition, system design, implementation, testing, deployment and maintenance. However, there still are problems, that lead project to the failure. Problems appear while iterating from design stage to the implementation or prototype creation.

Thesis describes methodology, that provide a way to overcome these problems. The main idea is to transform functional structure of the real-time system, that is designed using easy to understand graphical environment, to the executable code that will be able to run on target hardware components. Thesis describe the methods of doing such transformation. Methodology allows to create tool automating this process.

8 Terminų ir santrumpų žodynas

Santrumpa	Paaiškinimas
RLS	Realaus laiko sistemai
SSA	Skaitmeninių signalų apdorojimas
SKC	Sistemos kūrimo ciklas
RTOS	Realaus laiko operacinė sistema
SSP	Skaitmeninių signalų procesorius
TPS	Taikomoji programavimo sąsaja
UML	Unified Modeling Language
FDL	Feature Description Language
MMD	Model Driven Development
FIR	Finite Impulse Response
IIR	Infinite Impulse Response