Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

FPGA IP šerdžių šiuolaikinių kūrimo metodų tyrimas

Baigiamasis magistro projektas

Mindaugas Šulcas

Projekto autorius

prof. dr. Žilvinas Nakutis

Vadovas

Kaunas, 2020

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

FPGA IP šerdžių šiuolaikinių kūrimo metodų tyrimas

Baigiamasis magistro projektas

Elektronikos inžinerija (6211EX012)

Mindaugas Šulcas

Projekto autorius

prof. dr. Žilvinas Nakutis

Vadovas

doc. dr. Mindaugas Knyva

Recenzentas

Kaunas, 2020

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Mindaugas Šulcas

FPGA IP šerdžių šiuolaikinių kūrimo metodų tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Mindaugo Šulco, baigiamasis projektas tema „FPGA IP šerdžių šiuolaikinių kūrimo metodų tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

Mindaugas Šulcas	
(vardą ir pavardę įrašyti ranka)	(parašas)

Šulcas, Mindaugas. FPGA IP šerdžių šiuolaikinių kūrimo metodų tyrimas. Magistro baigiamasis projektas / vadovas prof. dr. Žilvinas Nakutis; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, inžinerijos mokslai.

Reikšminiai žodžiai: programuojamos logikos matricos, aukšto lygio sintezė, intelektualios nuosavybės šerdys.

Kaunas, 2020. 73 p.

Santrauka

Magistriniame darbe nagrinėjami šiuolaikiniai programuojamos logikos matricų intelektualios nuosavybės šerdžių kūrimo metodai. Šiame darbe siekiama ištirti konkretaus gamintojo siūlomus įrankius, skirtus šerdžių kūrimui. Tyrimo metu taip pat siekiama sukurti universalią intelektualios nuosavybės šerdžių vertinimo ir palyginimo metodiką, kuri programuojamos logikos matricų programuotojams leistų lengvai pasirinkti efektyviausią šerdžių kūrimo metodą, iškilusiai užduočiai spręsti. Tyrimas atliekamas analizuojant *Xilinx* programuojamos logikos matricų gamintojo siūlomus šerdžių kūrimo metodus ir sprendžiant vieną iš pagrindinių vaizdų apdorojimo užduočių – objektų kontūrų radimą.

Šulcas, Mindaugas. Research of IP Cores State-of-art Design Techniques. Master's Final Degree Project / supervisor prof. Žilvinas Nakutis; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics Engineering, Engineering Sciences.

Keywords: Field-Programmable Gate Arrays, High Level Synthesis, Intellectual Property Cores.

Kaunas, 2020. 73 p.

Summary

In this master thesis various intellectual property core design methodologies for field-programmable gate arrays are being analyzed. The goal of this work is to research specific field programmable gate arrays manufacturers recommended tools for core development. The other goal of this work is to create universal intellectual property core benchmarking methodology which would enable digital design engineers to easily select most effective core design tool for specific task solving. The research is carried out by analyzing *Xilinx* field-programmable gate arrays manufacturers recommended tools for core development and by using these tools to solve most common image processing task – object edge detection.

Turinys

Santrumpų ir terminų sąrašas **7**
Įvadas **8**
1. Tyrimo apžvalga **9**
1.1. *FPGA* gamintojų apžvalga ir palyginimas 9
1.2. *Xilinx* IP šerdžių kūrimui skirtų įrankių apžvalga 10
1.2.1. *Xilinx Vivado Design Suite* 11
1.2.2. *Xilinx Vivado HLS* 11
1.2.3. *MathWorks Simulink Model Composer* 13
1.3. Panašių tyrimų apžvalga 14
1.3.1. IP šerdžių kūrimo metodų palyginamųjų tyrimų apžvalga 14
1.3.1. IP šerdžių vertinimo ir palyginimo metodikų apžvalga 18
1.4. Apibendrinimas 19
2. IP šerdžių vertinimo ir palyginimo metodika **20**
2.1. Parametrų vertinimas, jų paskirtis, svarba ir savybės 20
2.2. Tyrimo metodika ir planas 21
2.2.1. Tyrimo metodika 21
2.2.2. Tyrimo planas 26
3. IP šerdžių realizacija ir tyrimas **27**
3.1. IP šerdžių realizacija 27
3.1.1. Slenkančio lango algoritmas 27
3.1.2. IP šerdies realizacija *Xilinx Vivado Design Suite* aplinkoje 28
3.1.3. IP šerdies realizacija *Xilinx Vivado HLS* aplinkoje 30
3.1.4. IP šerdies realizacija *MathWorks Simulink Model Composer* aplinkoje 31
3.2. IP šerdžių tyrimas 33
3.2.1. IP šerdžių tyrimas *Spartan-7* šeimoje 36
3.2.2. IP šerdžių tyrimas *Artix-7* šeimoje 40
3.2.3. IP šerdžių tyrimas *Kintex-7* šeimoje 42
3.2.4. IP šerdžių tyrimas *Zynq Ultrascale+* šeimoje 46
3.2.5. IP šerdžių kūrimo metodų palyginimas *Spartan-7* šeimoje 49
3.2.6. IP šerdžių kūrimo metodų palyginimas *Artix-7* šeimoje 50
3.2.7. IP šerdžių kūrimo metodų palyginimas *Kintex-7* šeimoje 51
3.2.8. IP šerdžių kūrimo metodų palyginimas *Zynq Ultrascale+* šeimoje 52
3.2.9. Apibendrintas IP šerdžių kūrimo metodų palyginimas 52
Išvados **55**
Literatūros sąrašas **56**
Priedai **59**
1 priedas. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies kodas 59
2 priedas. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies kodas 68
3 priedas. *Xilinx Vivado Design Suite* aplinkoje naudojamo testo kodas 70

Santrumpų ir terminų sąrašas

Santrumpos:

C – bendros paskirties programavimo kalba.
C++ – bendros paskirties objektinė programavimo kalba.
CPLD – (angl. *Complex Programmable Logic Device*) programuojamos logikos įrenginys skirtas nesudėtingiems procesams įgyvendinti.
FPGA – (angl. *Field-Programmable Gate Array*) programuojamos logikos matrica.
HDL – (angl. *Hardware Description Language*) programavimo kalbų grupė, skirta aprašyti skaitmeninius bei mišrių signalų sistemas.
HLS – (angl. *High-Level Synthesis*) aukšto lygio sintezės procesas, skirtas konvertuoti aukštesnio abstrakcijos lygio kodą į žemesnio abstrakcijos lygio kodą.
IDE – (angl. *Integrated Development Environment*) skaitmeninių įtaisų kūrimui skirta aplinka, apjungianti derinimui, testavimui bei programavimui skirtus įrankius
IP Core – (angl. *Intellectual Property Core*) intelektualios nuosavybės šerdis (programuojamos logikos matricos projekto dalis).
MPSoC – (angl. *Multi Processor System on Chip*) – mikrograndynas, apjungiantis kelias procesoriaus šerdis bei skirtingos paskirties grandynus viename luste.
RFSoC – (angl. *Radio Frequency Multi Processor System on Chip*) mikrograndynas, apjungiantis kelis, skirtingos paskirties grandynus ir aukšto radijo dažnių konverterius viename luste.
SoC – (angl. *System on Chip*) mikrograndynas, apjungiantis kelis, skirtingos paskirties grandynus viename luste.
Verilog – *HDL* kalbų grupei priklausanti programavimo kalba.
VHDL – (angl. *VHSIC Hardware Description Language*) *HDL* kalbų grupei priklausanti programavimo kalba.

Įvadas

Tobulėjant šiuolaikinėms elektronikos technologijoms didėja rinkos paklausa įvairiems prietaisams bei įrenginiams. Siekiant patenkinti šiuos rinkos poreikius įvairūs gamintojai atsižvelgdami į klientų ir pirkėjų norus kuria ir gamina vis naujus, geresnį funkcionalumą turinčius prietaisus. Kūrimo ir gamybos procesas užima ilgą laiko dalį kol produktas patenka į rinką, todėl įmonės šiuos procesus siekia paspartinti. Konkuruojant su kitomis įmonėmis greitas idėjos įgyvendinimas gali atnešti daugiau pelno bei leisti užsiimti didesnę rinkos dalį, tačiau tai gali turėti ir pasekmių – dėl nepastebėtų klaidų kūrimo ar gamybos proceso metu po tam tikro laiko prietaisai pradeda prastai funkcionuoti, sugenda ar net pridaro žalos pačiam vartotojui, įmonės praranda gerą reputaciją. Kartais greitas idėjos įgyvendinimas yra tiesiog neįmanomas dėl ilgų kūrimo procesų. Vienas iš tokių elektronikos įrenginių bei prietaisų kūrimo procesų yra programuojamos logikos matricų integravimas į sistemą t.y. kodo reikalingo prietaisui funkcionuoti parašymas bei ištestavimas. Programuojamos logikos matricų gamintojai atsižvelgdami į šią problemą siūlo įvairius pagalbinius įrankius, leidžiančius realizuoti norimą funkcionalumą sparčiau bei su mažesne klaidos tikimybe nei rašant įprastinį programos kodą. Pastaruoju metu šie įrankiai vis labiau populiarėja, tačiau nėra pateikiamų duomenų kaip jų panaudojimas paspartina kūrimo procesus, koks yra šių įrankių efektyvumas, taip pat nepateikiamos rekomendacijos kaip pasirinkti efektyviausią įrankį iškilusios užduoties sprendimui programuojamos logikos matricose.

Darbo tikslas – ištirti konkretaus gamintojo siūlomų *FPGA* IP šerdžių kūrimo metodų efektyvumą.

Darbo uždaviniai:

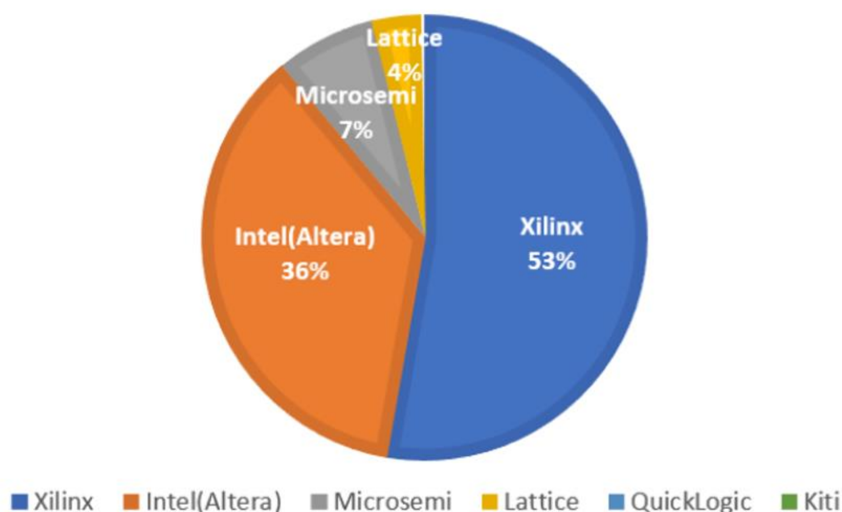
1. išanalizuoti *Xilinx* programuojamos logikos matricų gamintojo siūlomus intelektualios nuosavybės šerdžių kūrimo metodus;
2. išanalizuoti šerdžių vertinimo ir palyginimo metodikas;
3. sukurti universalią šerdžių vertinimo ir palyginimo metodiką;
4. skirtingais šerdžių kūrimo metodais realizuoti dvimačiame vaizde esančių objektų kontūrų radimui skirtas šerdis;
5. tiriant realizuotas šerdis bei naudojantis sukurta metodika palyginti intelektualios nuosavybės šerdžių kūrimo metodų efektyvumą.

1. Tyrimo apžvalga

1.1. *FPGA* gamintojų apžvalga ir palyginimas

Programuojamos logikos matricos *(FPGA)* – sudėtingi puslaidininkiniai įrenginiai realizuoti konfigūruojamų blokų matricomis, kurios tarpusavyje yra susietos programuojamais sujungimais. Programuojamos logikos matricos gali būti perprogramuojamos daugybę kartų taip, kad būtų įgyvendinama norima funkcija [1]. Šiuo metu *FPGA* rinkoje egzistuoja keliolika gamintojų, siūlančių įvairių galimybių programuojamas logines matricas bei jungtines sistemas viename luste *(SoC)*, kurias sudaro specialios paskirties įrenginiai, procesoriai bei programuojamos logikos matricos. Pagrindiniai *FPGA* gamintojai bei šių gamintojų užimama rinkos dalis pateikiama grafike (žr. 1 pav.).

***FPGA* gamintojų užimamos rinkos dalies analizė**



1 pav. *FPGA* gamintojų užimamos rinkos dalies analizė

Pagal 2016 metų viešai pateikiamus duomenis galima teigti jog didžiausią rinkos dalį tuo metu užėmė kompanija *Xilinx*, o antroji kompanija pagal užimama rinkos dalį buvo *Intel*, nupirkusi ankščiau *Altera* vardu žinoma programuojamos logikos matricų gamintoją [2].

Xilinx šiuo metu siūlo inovatyvius sprendimus *FPGA, SoC, MPSoC, RFSoC* bei *CPLD* srityse. Naujausi septintos ir aštuntos kartos mikrograndynai yra palaikomi programinėje įrangoje *Xilinx Vivado Design Suite* bei papildomuose įrankiuose skirtuose paspartinti programavimo procesus. *Xilinx* siūlomą programinę įrangą *Xilinx Vivado Design Suite* sudaro aplinka, skirta darbui su *HDL* kalbomis, integruoti kompiliatoriai, modeliavimo, grafinio programavimo bei derinimo įrankiai [3].

Intel siūlomi sprendimai apima *FPGA, SoC, MPSoC* bei *CPLD* sritis. Šie įrenginiai yra palaikomi *Intel Quartus Prime* programinėje įrangoje bei kituose *Intel* rekomenduojamuose programavimui skirtuose įrankiuose. *Intel Quartus Prime* programinę įrangą sudaro aplinka skirta programavimui *HDL* kalbomis, integruotas kompiliatorius bei programavimo ir derinimo įrankiai. Modeliavimo įrankis *Intel Quartus Prime* programinėje įrangoje yra išorinis – *Mentor Graphics ModelSim*, kuris nemokamai atsisiunčiamas kartu su *Intel Quartus Prime* programine įranga. Pravartu žinoti jog nemokama *Mentor Graphics ModelSim* įrankio versija yra ribotų funkcijų [4].

Lyginant *Xilinx* ir *Intel* siūlomą programinę įrangą pastebima jog didesnį funkcionalumą derinimo ir modeliavimo srityse, kurios ypač aktualios testuojant įrenginį bei siekiant panaikinti esamas klaidas programos kode, siūlo kompanija *Xilinx*. Abi kompanijos nemokamoje versijoje turi aukšto lygio sintezės įrankius. *Xilinx Vivado HLS* įrankis valdomas grafinėje aplinkoje, o tuo tarpu *Intel HLS* dalis valdymo yra komandinėje eilutėje, kuria naudotis yra sunkiau nei grafine aplinka.

Remiantis patogumų, derinimo ir modeliavimo skirtumais tarp programavimui skirtų aplinkų bei duomenimis apie rinkoje užimamą dalį tolimesniems tyrimams pasirenkama kompanija *Xilinx* ir jos darbui su programuojamos logikos matricomis siūlomi įrankiai.

1.2. *Xilinx* IP šerdžių kūrimui skirtų įrankių apžvalga

Xilinx darbui su IP šerdimis rekomenduoja naudotis šerdžių archyvu, kuris yra prieinamas *Xilinx Vivado Design Suite* programinėje įrangoje. Šiame archyve laikomos įvairių paskirčių mokamos ir nemokamos IP šerdys, kurias galima naudoti savo projekte dirbant su septintos bei aštuntos kartos programuojamos logikos matricomis.

Archyve esančios IP šerdys yra parametrizuojamos, tačiau redaguoti ir peržiūrėti programinio kodo daugumoje atveju negalima. Šiame archyve esančios šerdys turi gimtąją (angl. *native*) arba *AXI4* sąsajas.

Galimos šios *AXI4* sąsajos atmainos IP šerdžių duomenų mainams, kurios skiriasi duomenų interpretavimu, perdavimo sparta bei lankstumu [5]:

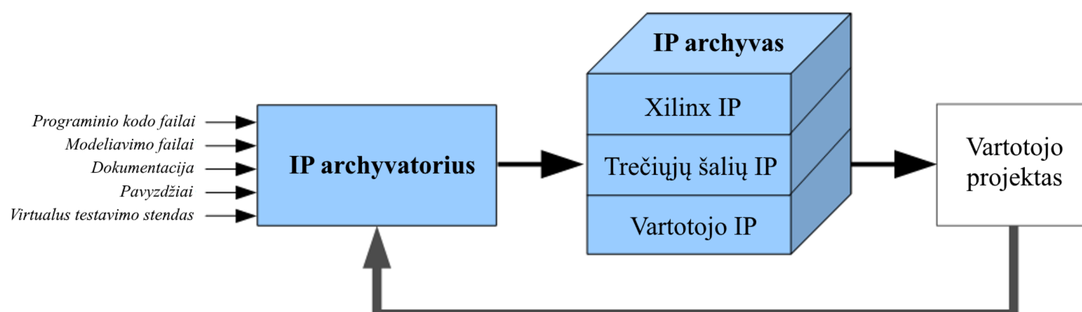
- *AXI4-Lite* – supaprastinta *AXI4* sąsaja, galinti atlikti vieną duomenų paketo perdavimą iš adresuojamos atminties (angl. *memory mapped*) į adresuojamą atmintį konkrečiu laiko momentu. Ji pasižymi mažu loginių resursų panaudojimu programuojamos logikos matricose ir dažniausiai yra naudojama IP šerdžių parametrų konfigūravimui veikimo metu.
- *AXI4-Stream* – *AXI4* sąsajos atmaina, nereikalaujanti adresų nustatymo fazės, galinti perduoti neriboto paketų skaičiaus duomenis vienu inicializavimu. *AXI4-Stream* yra laikoma neadresuojamos atminties sąsaja ir yra skirta didelės spartos duomenų mainams.
- *AXI4* – pagrindinė duomenų perdavimo sąsaja, galinti viena adreso nustatymo faze perduoti iki 256 duomenų paketų, kurių dydis priklauso nuo sąsajos pločio. Šios sąsajos panaudojamų resursų kiekis programuojamos logikos matricose yra didžiausias.

IP šerdys esančios archyve gali būti jungiamos tarpusavyje taip suformuojant sudėtingų funkcijų įrenginius, tačiau ne visuomet šiame kataloge esančios šerdys ar jų junginiai gali patenkinti užduotyje keliamus specifinius poreikius, todėl dažnai šerdis realizuoti tenka patiems inžinieriams, sprendžiantiems paskirtą uždavinį. Programuojamos logikos matricoms skirtas šerdis galima realizuoti keliais skirtingais įrankiais, kurie tarpusavyje skiriasi naudojama programavimo kalba bei generavimo procesais. *Xilinx* IP šerdžių kūrimui rekomenduoja šiuos, oficialiai palaikomus įrankius [6]:

- *Xilinx Vivado Design Suite*
- *Xilinx Vivado HLS*
- *MathWorks Simulink Model Composer*

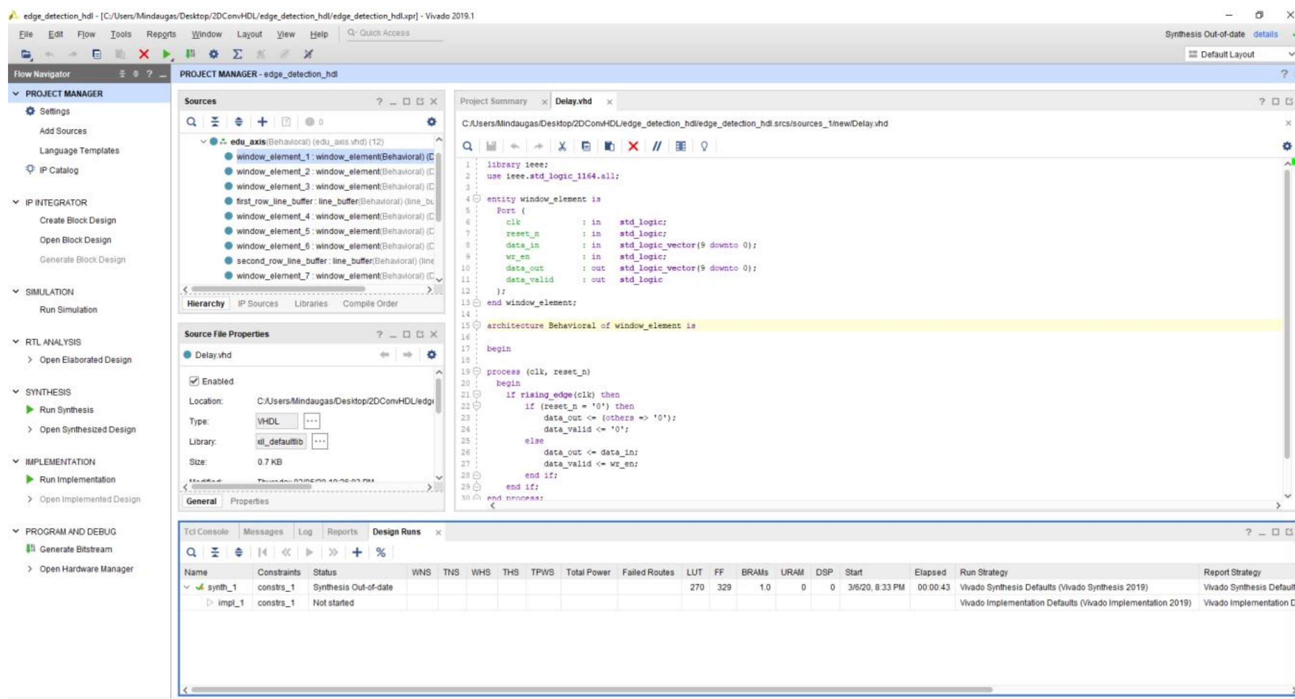
1.2.1. *Xilinx Vivado Design Suite*

Xilinx Vivado Design Suite – šią programinę įranga sudaro daugybė įrankių, vienas iš jų *Xilinx Vivado IP Packager*. Šis įrankis leidžia suarchyvuoti viena ar keliomis *HDL* kalbomis (*VHDL* ir *Verilog)* parašytą programinį kodą į IP šerdį. *Xilinx Vivado Design Suite* programinėje įrangoje esantis įrankis *Xilinx Vivado IP Packager* taip pat palengvina šerdžių kūrimo procesą pateikiant *AXI4* sąsajų šablonus [7]. Pateikiama IP šerdžių kūrimo eigos programinėje įrangoje *Xilinx Vivado Design Suite* diagrama (žr. 2 pav.).



2 pav. IP šerdžių kūrimo eiga programinėje įrangoje *Xilinx Vivado Design Suite* [8]

Didžiausias trūkumas dirbant su *Xilinx Vivado Design Suite* programine įranga yra IP šerdies kodo rašymas *HDL* kalbomis. Programavimas šiomis kalbomis bei parašyto kodo testavimas užtrunka ypač ilgai, o sudėtingų dizainų kūrimui reikalingi didelę patirtį turintys specialistai [8]. Pateikiamas *Xilinx Vivado Design Suite* programines įrangos darbinės aplinkos paveikslėlis (žr. 3 pav.).

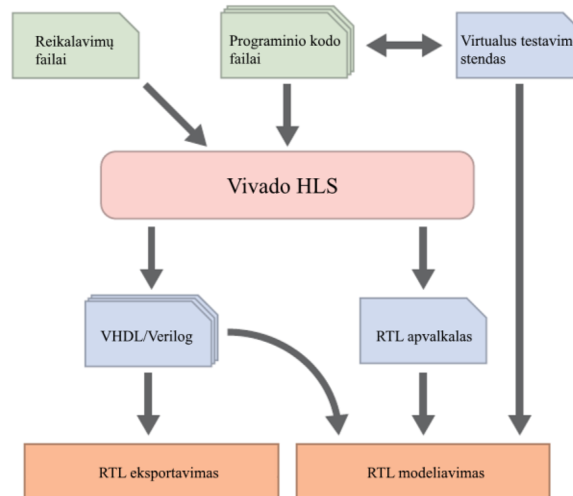


3 pav. *Xilinx Vivado Design Suite* programinės įrangos darbinė aplinka

1.2.2. *Xilinx Vivado HLS*

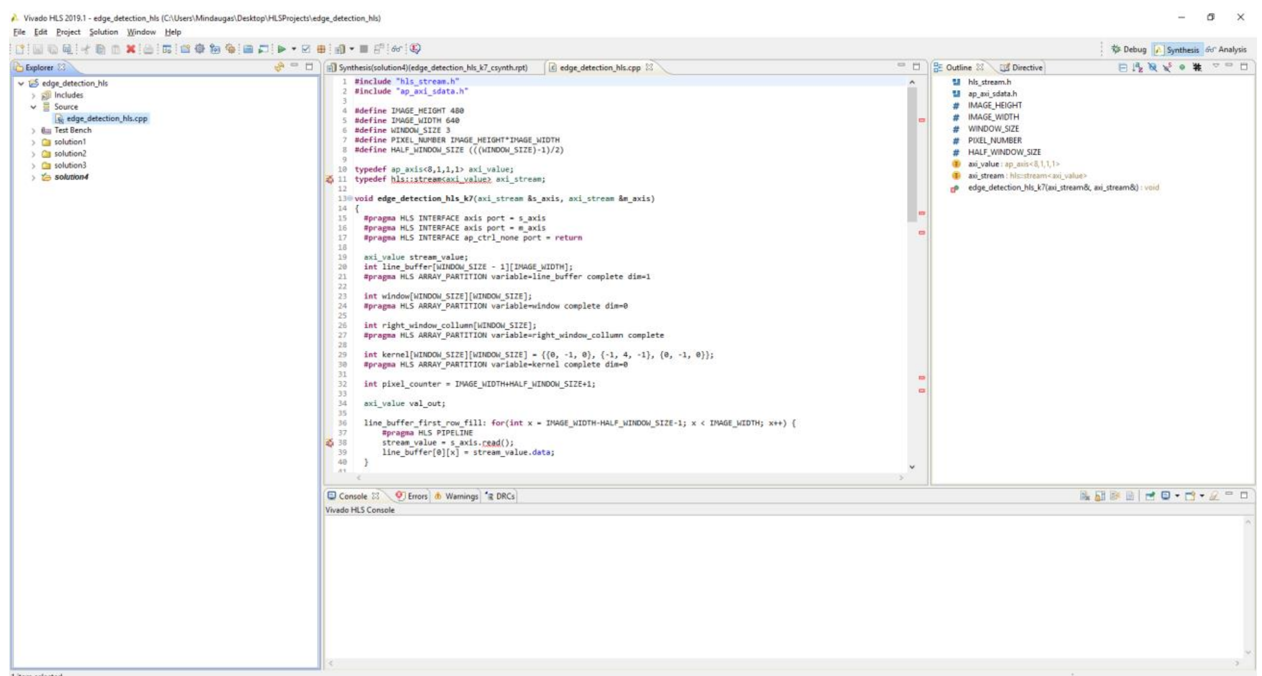
Xilinx Vivado HLS – kompanijos *Xilinx* sukurta programinė įranga, leidžianti kodą, parašytą aukštesnio abstrakcijos lygio kalba (*C* ar *C++*), konvertuoti į *HDL* kalba parašytą programos kodą.

Šis įrankis leidžia ne tik atlikti programinio kodo konvertavimą, bet ir testavimą bei testavimui skirtų virtualių stendų iš aukštesnio abstrakcijos lygio kalbos konvertavimą į *HDL* kalbas [9]. Pateikiama IP šerdžių kūrimo eigos programinėje įrangoje *Xilinx Vivado HLS* diagrama (žr. 4 pav.).



4 pav. IP šerdžių kūrimo eiga programinėje įrangoje *Xilinx Vivado HLS* [8]

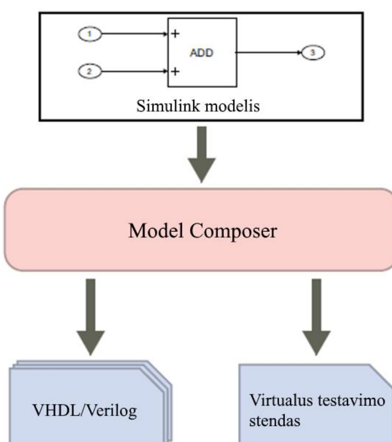
Pagrindinis *Xilinx Vivado HLS* programinės įrangos trūkumas yra reikalingos žinios apie įrangą, kuriai bus konvertuojamas programos kodas. Prieš rašant programos kodą aukštesnio abstrakcijos lygio kalba būtina žinoti apie pasirinktos programuojamos logikos matricos vidinius resursus bei *FPGA* veikimo principus, kadangi *Xilinx Vivado HLS* programinėje įrangoje privaloma nurodyti kaip bus realizuojamas kodas programuojamos logikos matricos architektūroje, priešingu atveju jis bus neoptimalus. Šis įrankis leidžia žymiai sparčiau realizuoti norimą funkcionalumą pasirinktoje programuojamos logikos matricoje bei kurti sudėtingus projektus neturint didelės patirties programavime *HDL* kalbomis [8]. *Xilinx Vivado HLS* programinė įranga yra nemokamas, paremta *Eclipse IDE* programinės įrangos paketu. Pateikiamas *Xilinx Vivado HLS* programines įrangos darbinės aplinkos paveikslėlis (žr. 5 pav.).



5 pav. *Xilinx Vivado HLS* programinės įrangos darbinė aplinka

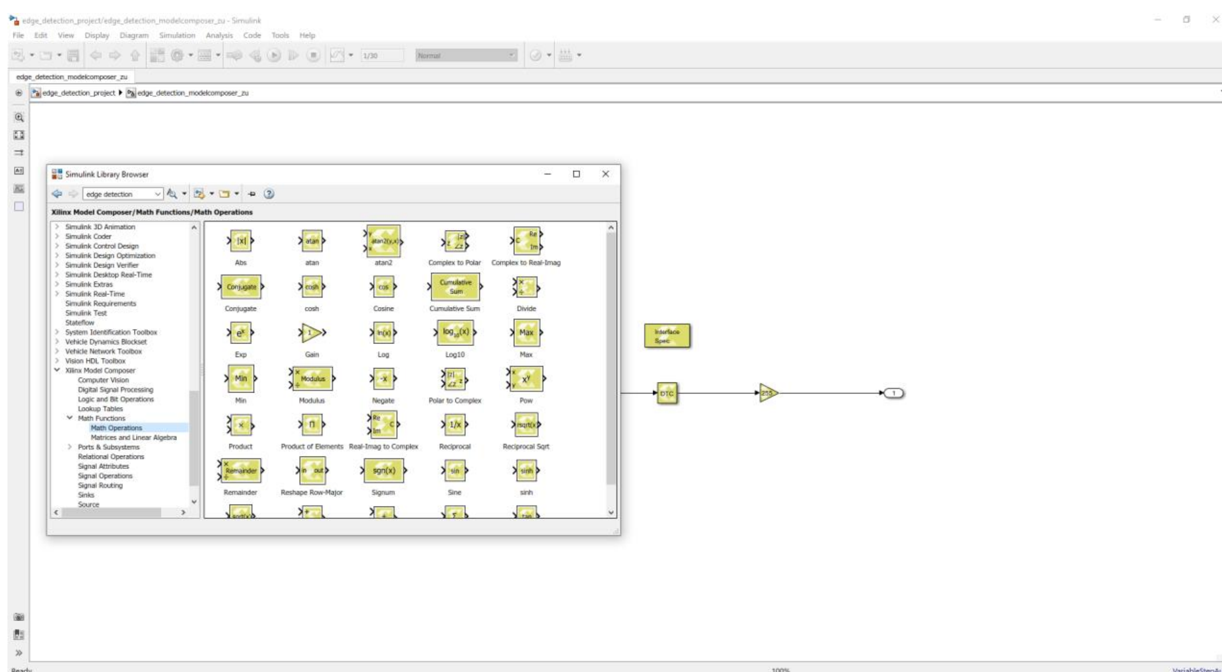
1.2.3. *MathWorks Simulink Model Composer*

MathWorks Simulink Model Composer – *MathWorks* programinėje įrangoje esantis įrankis, skirtas *Simulink* aplinkoje blokeliais realizuotų programų konvertavimui į *HDL* kalba parašytų programų kodą. Šis įrankis automatiškai aptinka slankaus kablelio skaičiaus atvaizdavimo formą bei pakeičia ją į fiksuoto kablelio skaičiaus atvaizdavimo formą, kadangi slankaus kablelio skaičiaus atvaizdavimo forma programuojamos logikos matricose yra neefektyvi ir panaudoja daugybe resursų [10]. Pateikiama IP šerdžių kūrimo eigos programinėje įrangoje *MathWorks Simulink Model Composer* diagrama (žr. 6 pav.).



6 pav. IP šerdžių kūrimo eiga programinėje įrangoje *MathWorks Simulink Model Composer* [8]

MathWorks šį įrankį siūlo naudoti sprendžiant skaitmeninių signalų apdorojimo užduotis. Vienas iš šio įrankio trūkumų yra ne visos palaikomos galimos *Simulink* programinės įrangos funkcijos. Taip pat *MathWorks Simulink Model Composer* įrankiu sugeneruotas kodas yra sunkiai skaitomas, tačiau vienas didžiausių privalumų naudojantis šiuo įrankiu tai nereikalingos žinios apie programavimą *HDL* kalbomis ir naudojamo įrenginio architektūrą [8]. Pateikiamas *MathWorks Simulink Model Composer* programines įrangos darbinės aplinkos paveikslėlis (žr. 7 pav.).



7 pav. *MathWorks Simulink Model Composer* programinės įrangos darbinės aplinka

1.3. Panašių tyrimų apžvalga

Siekiant susipažinti su skirtingų įrankių, skirtų IP šerdžių kūrimui, efektyvumu ir galimybėmis išanalizuojami jau atlikti tyrimai panašiomis temomis. Analizuojamuose straipsniuose aptariami IP šerdžių kūrimui skirti įrankiai, naudojimosi jais specifika, gauti palyginamųjų tyrimų rezultatai bei vertinimo metodikos.

1.3.1. IP šerdžių kūrimo metodų palyginamųjų tyrimų apžvalga

Didžiausias dėmesys skiriamas tyrimams, kuriuose palyginami įvairūs IP šerdžių kūrimo metodai. Vienas iš tokių tyrimų aptaria *Xilinx Vivado HLS* įrankyje *C++* kalba sukurtos IP šerdies efektyvumą lyginant su *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota šerdimi. Analizuojame tyrime *C++* ir *VDHL* kalbomis realizuojamas 8 bitų rašymui ir skaitymui skirtas registras. Pateikiami tyrimo metu gauti rezultatai (žr. 1 lentelė.) [11].

1 lentelė. *C++* kalba ir *VHDL* kalba realizuotų 8 bitų registrų palyginimas

	Xilinx Vivado Design Suite (VHDL)	*Xilinx Vivado HLS (C++)*
LUT	64	9
FF	24	10
BRAM	0	1

Šio tyrimo metu dėl prastos šerdies realizacijos *C++* kalba programuojamos logikos matricoje panaudojami brangūs *BRAM* architektūriniai elementai, tačiau dėl to kitų panaudojamų architektūrinių elementų kiekis yra ženkliai mažesnis nei lyginant su *VHDL* kalba realizuota šerdimi. Dėl šios priežasties tiesioginis dviejų skirtingų IP šerdžių kūrimo metodų palyginimas tyrime yra negalimas.

Kitame tyrime, kuriame taip pat lyginamas *Xilinx Vivado HLS* įrankio efektyvumas su įprastiniu kodo rašymu *HDL* kalbomis, realizuojama IP šerdis greitosios *Furje* transformacijos skaičiavimui. Straipsnyje IP šerdžių kūrimo metodų palyginimas atliekamas vertinant architektūrinių elementų *BRAM* ir *DSP* panaudojimą programuojamos logikos matricose bei laiko trukmę, skirtą IP šerdžių realizacijai. Pateikiami tyrimo metu gauti rezultatai (žr. 2 lentelė.) [12].

2 lentelė. *C++* ir *Verilog* kalbomis realizuotų greitosios *Furje* transformacijos šerdžių palyginimas

	Xilinx Vivado Design Suite (Verilog)	*Xilinx Vivado HLS (C++)*
BRAM	109	93
DSP	266	242
Realizacijos trukmė	2 savaitės	1 savaitė

Realizuojant greitosios *Furje* transformacijos skaičiavimui skirtą šerdį, kuri susideda iš daugybės sandaugų ir sumavimo operacijų, pastebėta jog *Xilinx Vivado HLS* įrankiu realizuota šerdis efektyviau panaudojo *BRAM* ir *DSP* architektūrinius elementus. Taip pat laikas, užtruktas realizuoti norimą funkcionalumą skirtingais IP šerdžių kūrimo metodais, skiriasi dvigubai. Tačiau šiame straipsnyje neaptariami kiti panaudojami programuojamos logikos matricos architektūriniai elementai, kurie galėjo būti panaudojami efektyviau *Verilog* kalba realizuotoje IP šerdyje.

VHDL kalba ir *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotų IP šerdžių palyginimas taip pat atliekamas dar viename tyrime. Šiame tyrime analizuojamas ir lyginamas skirtingais šerdžių kūrimo metodais realizuotų galios elektronikos valdymo algoritmų efektyvumas. Abiem atvejais realizuotos

IP šerdys ištestuojamos *Xilinx Spartan3-1600* programuojamos logikos matricoje. Pateikiami straipsnyje aprašyto tyrimo rezultatai (žr. 3 lentelė.) [13].

3 lentelė. *C++* kalba ir *VHDL* kalba realizuotų galios elektronikos valdymo algoritmų palyginimas

	Xilinx ISE (VHDL)	***Xilinx Vivado HLS (C++)***
SLICES	2445	5221
LUT	3848	3523
LUTRAM	0	84
BRAM	5	1
MULT18X8	6	4
Vėlinimas (taktinio signalo periodų skaičius)	86	77

Šiame straipsnyje lyginant du skirtingus IP šerdžių kūrimo metodus vertinami ne tik panaudojami programuojamos logikos matricos architektūriniai elementai, bet ir signalų vėlinimas. Signalų vėlinimas nusako kiek laiko prabėga kol įrenginio išėjime gaunami rezultatai nuo laiko momento kuomet duomenys patenka į įrenginio įėjimą. Tyrime teigiama jog galios elektronikos valdymo algoritmas efektyviau realizuojamas naudojantis *Xilinx Vivado HLS* įrankiu *C++* kalba, kadangi šios realizacijos atveju panaudojama mažiau ypač vertingų architektūrinių elementų (*BRAM*, *MULT18X8*) bei pasiekiami geresni laikiniai parametrai. Verta paminėti jog straipsnyje IP šerdžių kūrimo metodai lyginami galutinę sintezę atliekant šeštos kartos programuojamos logikos matricoje naudojantis *Xilinx ISE* įrankiu. Šiuo metu naudojamos septintos ir aštuntos kartos programuojamos logikos matricos bei joms skirti įrankiai leidžia algoritmus realizuoti efektyviau nei jų pirmtakai.

Dar vienas tyrimas skirtas palyginti *Xilinx Vivado HLS* įrankio pagalba sukurtas IP šerdis su *HDL* kalbomis sukurtomis šerdimis atliktas realizuojant *Lloyd's* ir filtravimo algoritmus. Šiame tyrime IP šerdys realizuojamos septintos kartos *Xilinx Virtex-7* programuojamos logikos matricoje. Pateikiami straipsnyje aprašyto tyrimo rezultatai realizuojant *Lloyd's* (žr. 4 lentelė.) ir filtravimui (žr. 5 lentelė.) skirtus algoritmus [14].

4 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto *Lloyd's* algoritmo palyginimas

	Xilinx Vivado Design Suite	***Xilinx Vivado HLS***
LUT	66472	68360
FF	62168	63878
DSP	120	120
BRAM	83	89
t_V, ***taktinio signalo periodų skaičius***	53000	66000
T_{MIN}, ***ns***	5.0	9.7

5 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto filtravimo algoritmo palyginimas

	Xilinx Vivado Design Suite	***Xilinx Vivado HLS***
LUT	10418	16106
FF	19008	17013
DSP	40	38
BRAM	448	507
t_V, ***taktinio signalo periodų skaičius***	54000	165000
T_{MIN}, ***ns***	5.0	6.3

Straipsnyje aprašomame tyrime lyginant skirtingus IP šerdžių kūrimo metodus vertinami panaudojami programuojamos logikos matricos architektūriniai elementai bei laikiniai parametrai (T_{MIN} – minimalus taktinio signalo periodas, t_V – algoritmo vykdymo trukmė). Tyrimo metu gauti rezultatai rodo jog *Xilinx Vivado HLS* įrankio pagalba realizuotos IP šerdys buvo ne tokios efektyvios kaip *HDL* kalbomis realizuotos šerdys, tokia išvada priimta dėl didesnio architektūrinių elementų panaudojimo skaičiaus ir dėl prastesnių laikinių parametrų.

Panašaus pobūdžio rezultatai taip pat aptariami tyrime, kuriame lyginamos *Xilinx Vivado HLS* aplinkoje *C* kalba ir *Xilinx Vivado Design Suite* aplinkoje *HDL* kalbomis realizuotos IP šerdys. Pateikiami straipsnyje aprašyto tyrimo rezultatai realizuojant *Galois* laukų daugybos (žr. 6 lentelė.) ir dalybos (žr. 7 lentelė.) operacijas [15].

6 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuotos *Galois* laukų daugybos operacijos palyginimas

	Xilinx Vivado Design Suite	***Xilinx Vivado HLS***
LUT	1039	2735
FF	1017	2695

7 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuotos *Galois* laukų dalybos operacijos palyginimas

	Xilinx Vivado Design Suite	***Xilinx Vivado HLS***
LUT	277	1038
FF	575	1515

Tyrimo metu gauti rezultatai yra palyginami vertinant panaudojamų architektūrinių elementų skaičių, remiantis šiuo kriterijumi galima teigti jog *Xilinx Vivado HLS* įrankio pagalba realizuotos IP šerdys buvo ne tokios efektyvios kaip *HDL* kalbomis realizuotos šerdys, tačiau straipsnyje didelis dėmesys skiriamas šerdžių realizavimo trukmei, kuri *Xilinx Vivado HLS* įrankio atžvilgiu yra ženkliai mažesnė.

Aptariamas dar vienas tyrimas, kuriame lyginamos *Xilinx Vivado HLS* įrankio pagalba realizuotos IP šerdys su *HDL* kalbomis realizuotomis šerdimis, skirtomis vaizdų apdorojimui. Šiame tyrime *Xilinx Zynq-7020 SoC* mikrograndyne realizuojamas *Step Row* algoritmas, skirtas kelio ženklinimo linijų atpažinimo uždaviniui spręsti, *Data Binning* algoritmas, skirtas sumažinti priimamo kadro rezoliuciją, bei *Sobel* algoritmas, skirtas įvairių objektų kontūrams išryškinti. Pateikiami straipsnyje aprašyto tyrimo rezultatai įgyvendinant *Step Row* (žr. 8 lentelė.), *Data Binning* (žr. 9 lentelė.) ir *Sobel* (žr. 10 lentelė.) algoritmus [16].

8 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto *Step Row* algoritmo palyginimas

	Xilinx Vivado Design Suite	***Xilinx Vivado HLS***
LUT	145	414
FF	120	370
DSP	2	2
BRAM	2	0
f, MHz	200	200
tv, ms	0.95	4.50
Realizacijos trukmė	10 dienų	4 dienos

9 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto *Data Binning* algoritmo palyginimas

	Xilinx Vivado Design Suite	*Xilinx Vivado HLS*
LUT	196	18136
FF	121	20379
DSP	0	36
BRAM	0	6
f, MHz	200	164
tv, ms	1.89	2.05
Realizacijos trukmė	12 dienų	2 dienos

10 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto *Sobel* algoritmo palyginimas

	Xilinx Vivado Design Suite	*Xilinx Vivado HLS*
LUT	1111	21946
FF	748	25439
DSP	2	138
BRAM	3	18
f, MHz	200	200
tv, ms	0.39	0.49
Realizacijos trukmė	12 dienų	1 dienos

Šiame tyrime gauti rezultatai lyginami panaudojamų architektūrinių elementų ir laikinių parametrų atžvilgiu. Remiantis šiais kriterijais efektyvesniu IP šerdžių kūrimo metodu laikomas programavimas *HDL* kalbomis, tačiau straipsnyje pabrėžiama jog realizacijos laikas naudojantis *Xilinx Vivado HLS* įrankiu yra ženkliai mažesnis.

Daugiau IP šerdžių kūrimo metodų aptariama tyrime, kuriame realizuojamos IP šerdys, skirtos atvirkštinei diskrečiajai kosinusinei transformacijai atlikti. Tyrime lyginamos *Xilinx Vivado Design Suite* aplinkoje *Verilog* kalba, *Xilinx VIvado HLS* aplinkoje *C* kalba ir *MathWorks Simulink* aplinkoje realizuotos šerdys. Pateikiami straipsnyje aprašyto tyrimo metu gauti rezultatai (žr. 11 lentelė.) [17].

11 lentelė. Skirtingais IP šerdžių kūrimo metodais realizuoto algoritmo palyginimas

	Xilinx Vivado Design Suite	*Xilinx Vivado HLS*	*MathWorks Simulink*
LUT	38790	50566	13669
FF	11762	34955	1140
BRAM	32	13	0
f, MHz	150	208	110

Tyrime didžiausias dėmesys skiriamas panaudojamų architektūrinių elementų skaičiui bei maksimaliam IP šerdies taktinio signalo dažniui. Architektūrinių elementų atžvilgiu efektyviausia IP šerdys buvo sukurta naudojantis *MathWorks Simulink*, tačiau maksimalaus taktinio signalo dažnio atžvilgiu efektyvesne laikoma *Xilinx VIvado HLS* aplinkoje *C* kalba realizuota IP šerdis.

Galima pastebėti jog beveik visuose aptartuose tyrimuose apžvelgiami tik du pagrindiniai IP šerdžių kūrimo metodai: *Xilinx Vivado Design Suite* aplinkoje *HDL* kalbomis ir *Xilinx VIvado HLS* aplinkoje *C/C++* kalbomis. Šerdžių kūrimo metodų palyginimui dažniausias naudojamas kriterijus yra panaudojamų architektūrinių elementų skaičius, mažesnis dėmesys skiriamas laikiniams parametrams, tačiau nei viename aptartame tyrime neatsižvelgiama į energetinius parametrus,

nepateikiama universali IP šerdžių vertinimo metodika. Dėl šių priežasčių tolimesniam analizavimui pasirinkti moksliniai straipsniai, aprašantys įvairias IP šerdžių vertinimo metodikas.

1.3.1. IP šerdžių vertinimo ir palyginimo metodikų apžvalga

Pirmajame aptariamame tyrime aprašomi pagrindiniai metodai programuojamos logikos matricoms skirtų IP šerdžių palyginimui. Šiame tyrime palyginamos dvi *HDL* kalbomis aprašytos automobilių industrijoje naudojamos sąsajos: *CAN* ir *LIN*. Jų palyginimui straipsnyje siūloma naudoti panaudojamų architektūrinių elementų skaičių, IP šerdies suvartojamą galią bei specifinius sąsajų parametrus [18].

Toks IP šerdžių vertinimo metodas nėra universalus, kadangi vertinami specifiniai realizuotų šerdžių parametrai. Tyrime nevertinami laikiniai parametrai, o energetiniai parametrai apskaičiuojami sumuojant programuojamos logikos matricos prievadų ir architektūrinių elementų suvartojamą galią. Dėl šios priežasties suvartojamos galios parametras priklauso nuo pasirinktos programuojamos logikos matricos architektūros ir šeimos.

Kitame aptariamame tyrime siekiant palyginti tris *Microchip Technology* įmonės *PIC* architektūros mikrokontrolerių IP šerdis jos vertinamos atsižvelgiant į tris parametrų rūšis: panaudojamų resursų kiekį, laikinius ir energetinius parametrus. Panaudojamų resursų kiekis tyrime pateikiamas kaip viena skaitinė išraiška – ekvivalenčių ventilių skaičius. Ekvivalenčių ventilių skaičius randamas perskaičiuojant skirtingų panaudojamų vidinių resursų kiekį į NAND elementų, reikalingų įgyvendinti toki patį funkcionalumą, kiekį. Laikinius parametrus šiame tyrime siūloma vertinti uždavinio vykdymo laiku, kuris apskaičiuojamas pagal toliau pateikiamą formulę (1):

$$T_V = T_{clk} \cdot C_{clk}, \tag{1}$$

čia T_{clk} – taktinio signalo periodas, o C_{clk} – taktinio signalo periodų skaičius, reikalingas tam tikrai operacijai atlikti. Apskaičiuotas parametras naudojamas ne tik IP šerdžių palyginimui, bet ir kito vertinimui naudojamo parametro apskaičiavimui – suvartojamai energijai. Straipsnyje aprašytame tyrime teigiama jog suvartojamos energijos parametras yra labiau tinkama IP šerdžių palyginimui nei momentinės suvartojamos galios parametras. Suvartojama energija apskaičiuojama pagal toliau pateikiamą formulę (2):

$$E = T_V \cdot P, \tag{2}$$

čia T_V – vykdymo laikas, o P – momentinė IP šerdies suvartojama galia [19].

Šiame tyrime aptartas IP šerdžių panaudojamų vidinių resursų palyginimo metodas yra nepriklausomas nuo programuojamos logikos matricos gamintojo ir architektūros, kadangi palyginimui naudojamas ekvivalenčių ventilių skaičius. Tačiau šis parametras dažnai nėra pateikiamas programuojamos logikos matricų gamintojų specifikacijose, todėl perskaičiavimas į ekvivalenčių ventilių skaičių tampa neįmanomas. Taip pat tyrimo metodikoje aprašomi energetiniai parametrai priklauso nuo laikinių parametrų ir neatspindi realių suvartojamos energijos sąlygų, kadangi programuojamos logikos matricos naudoja energiją ne tik uždavinio vykdymo laiku.

Kitokio pobūdžio analizuojamame tyrime lyginamos dvi, skirtingų kartų, programuojamos logikos matricos sprendžiant uždavinius su fiksuoto ir slankaus kablelio skaičių atvaizdavimo formomis. Tyrime *FPGA* našumams palyginti vertinami šie parametrai: panaudojamų architektūrinių elementų skaičius bei maksimalus taktinio signalo dažnis. Šie parametrai randami programuojamos logikos matricose realizuojant pagrindines matematines operacijas: sudėtį, atimtį, daugybą, dalybą bei trigonometrinių funkcijų aproksimacijas. Straipsnyje aprašomo tyrimo metu gautiems rezultatams apskaičiuojamas aritmetinis vidurkis, kuris pateikiamas kaip viena skaitinė išraiška kiekvienam iš vertinamų parametrų [20]. Toks duomenų pateikimas leidžia skaitytojui greičiau priimti išvadas apie skirtingų programuojamos logikos matricų efektyvumą.

1.4. Apibendrinimas

Remiantis užimama rinkos dalimi tolimesniam tyrimui pasirinktas *Xilinx* programuojamos logikos matricų gamintojas. Kompanija *Xilinx* IP šerdžių kūrimui rekomenduoja tris, oficialiai palaikomus metodus: *HDL* kalbomis *Xilinx Vivado Design Suite* aplinkoje, *C/C++* kalbomis *Xilinx Vivado HLS* aplinkoje bei funkciniais blokeliais *MathWorks Simulink Model Composer* aplinkoje. Toliau tiriamajame darbe analizuojami išvardinti intelektualios nuosavybės šerdžių kūrimo metodai.

Apžvelgus panašaus tiriamojo darbo pobūdžio mokslinius straipsnius pastebėta jog vertinant IP šerdis ir programuojamos logikos matricas didžiausias dėmesys yra skiriamas panaudojamų architektūrinių elementų skaičiui, tačiau tik keliuose straipsniuose vertinami ir kiti kertiniai programuojamos logikos matricų parametrai. Duomenų pateikimui dažniausiai naudojamos procentinės, normuotos ar kitaip matematiškai apdorotos skaitinės išraiškos, leidžiančios skaitytojui greičiau suprasti tyrimo metu gautus rezultatus.

Sekančiame skyriuje pateikiama sukurta universalesnė bei pranašesnė IP šerdžių vertinimo ir palyginimo metodika lyginant su moksliniuose straipsniuose aptartomis metodikomis. Vertinant IP šerdis siekiama atkreipti dėmesį į visus kertinius programuojamos logikos matricų parametrus, o gautus palyginimų rezultatus pateikti kuo aiškesne ir skaitytojui lengviau suprantama grafine bei skaitine forma.

2. IP šerdžių vertinimo ir palyginimo metodika

2.1. Parametrų vertinimas, jų paskirtis, svarba ir savybės

Siekiant palyginti IP šerdis, sukurtas trimis skirtingais metodais, didžiausias dėmesys skiriamas šioms kertinėms programuojamos logikos matricų parametrų grupėms:

- panaudojamų vidinių resursų skaičiui;
- laikiniams parametrams;
- energetiniams parametrams.

Toliau aptariama šias grupes sudarančių parametrų paskirtis, svarba ir savybės programuojamos logikos matricose bei jų pagrindu realizuotose sistemose. Pirmoji iš vertinamų parametrų grupių yra panaudojamų vidinių resursų skaičius. Vidiniai resursai – architektūriniai elementai, kurių pagrindu programuojamos logikos matricose realizuojami įvairūs uždavinių sprendimai. Vertinamus vidinius resursus sudaro šie architektūriniai elementai:

- *LUT* – peržiūros lentelės (angl. *look-up table*), šešių tarpusavyje nepriklausomų įėjimo ir dviejų tarpusavyje nepriklausomų išėjimo signalų elementai, skirti Būlio algebros funkcijoms realizuoti [21].
- *FF* – saugojimo elementas (angl. *flip-flop*), kuris gali būti sukonfigūruotas kaip signalo frontui jautrus D tipo registras arba kaip signalo lygiui jautri skląstis. Saugojimo elementai skirti išsaugoti signalo, paduoto į elemento įėjimą, būseną elemento išėjime [21].
- *BRAM* – blokinė operatyvioji atmintis (angl. *block random access memory*), atminties ląstelės, skirtos binarinių duomenų saugojimui įvairiais formatais (skirtingas bitų skaičius vektoriuje). Šios ląstelės gali būti konfigūruojamos kaip viena 36Kb arba kaip dvi nepriklausomos 18Kb atminties ląstelės [22].
- *DSP* – skaitmeninių signalų apdorojimo blokai (angl. *digital signal processing*), skirti pagreitinti daugybos, dalybos, sudėties, atimties bei kitus veiksmus, dažnai naudojamus skaitmeniniams signalams apdoroti [23].

Šių vidinių programuojamos logikos matricų resursų panaudojimas priklauso nuo realizuojamo uždavinio funkcionalumo. Didėjant realizuojamų užduočių skaičiui ir sudėtingumui atitinkamų panaudojamų vidinių resursų skaičius taip pat didėja. Vidinių resursų negalima lyginti ir sutapatinti vieno su kitu, todėl bendram jų įvertinimui negalimas vienas kriterijus. Kiekvieną iš architektūrinių elementų privaloma vertinti atskirai.

Antroji IP šerdžių vertinamų parametrų grupė yra laikiniai parametrai. Ši parametrų grupė yra aktuali sistemose, kurios realizuojamos programuojamos logikos matricų pagrindu. Tokių sistemų našumas bei sprendimų priėmimo trukmė priklauso nuo realizuotų IP šerdžių laikinių parametrų. Vertinami šie IP šerdžių parametrai:

- Minimalus taktinio signalo periodas – (angl. *minimum period*) nurodo patį mažiausią taktinio signalo periodą, kuriam esant sistema veikia sparčiausiai. Jei taktinio signalo periodas būtų sumažinamas dar labiau, sistema taptų nestabili, tuo metu padidinus taktinio signalo periodą sistemos našumas sumažėtų.
- Uždavinio vykdymo trukmė – (angl. *execution time*) nurodo, kiek taktinio signalo impulsų turi praeiti nuo duomenų pateikimo į IP šerdies įėjimą iki kol visi duomenys bus apdoroti ir paskutinysis bus priimtas IP šerdies išėjime.

Minimalus IP šerdies taktinio signalo periodas priklauso nuo pasirinktos programuojamos logikos matricos architektūros ir naudojamų architektūrinių elementų tipo, nes skirtingose architektūrose naudojamų elementų realizacijos sprendimai lemia skirtingą stabilumo užtikrinimą mažinant taktinio signalo periodą. Likę laikiniai parametrai priklauso nuo sprendžiamo uždavinio sudėtingumo ir realizacijos programuojamos logikos matricoje.

Trečioji IP šerdžių vertinamų parametrų grupė yra energetiniai parametrai. Ši parametrų grupė kaip ir laikinių parametrų grupė yra aktuali sistemose, kurios realizuojamos programuojamos logikos matricų pagrindu. Tokių sistemų energetinis efektyvumas, galimybė veikti iš baterijų ar alternatyvių energijos šaltinių yra sąlygojama realizuotų IP šerdžių energetiniais parametrais. Atskirus programuojamos logikos matricų energetinius parametrus, tokius kaip signalų, logikos, atminties bei kitų elementų suvartojamą galią galima apjungti ir pateikti kaip vieną, bendrą parametrą:

- Momentinė suvartojama galia – (angl. *power consumption*), nurodo kokia galia yra suvartojama iš energijos šaltinio per sekundę naudojant IP šerdį, realizuotą programuojamos logikos matricoje.

IP šerdies suvartojama momentinė galia priklauso nuo pasirinktos programuojamos logikos matricos architektūros, taip pat ir nuo sprendžiamo uždavinio sudėtingumo bei realizacijos programuojamos logikos matricoje.

2.2. Tyrimo metodika ir planas

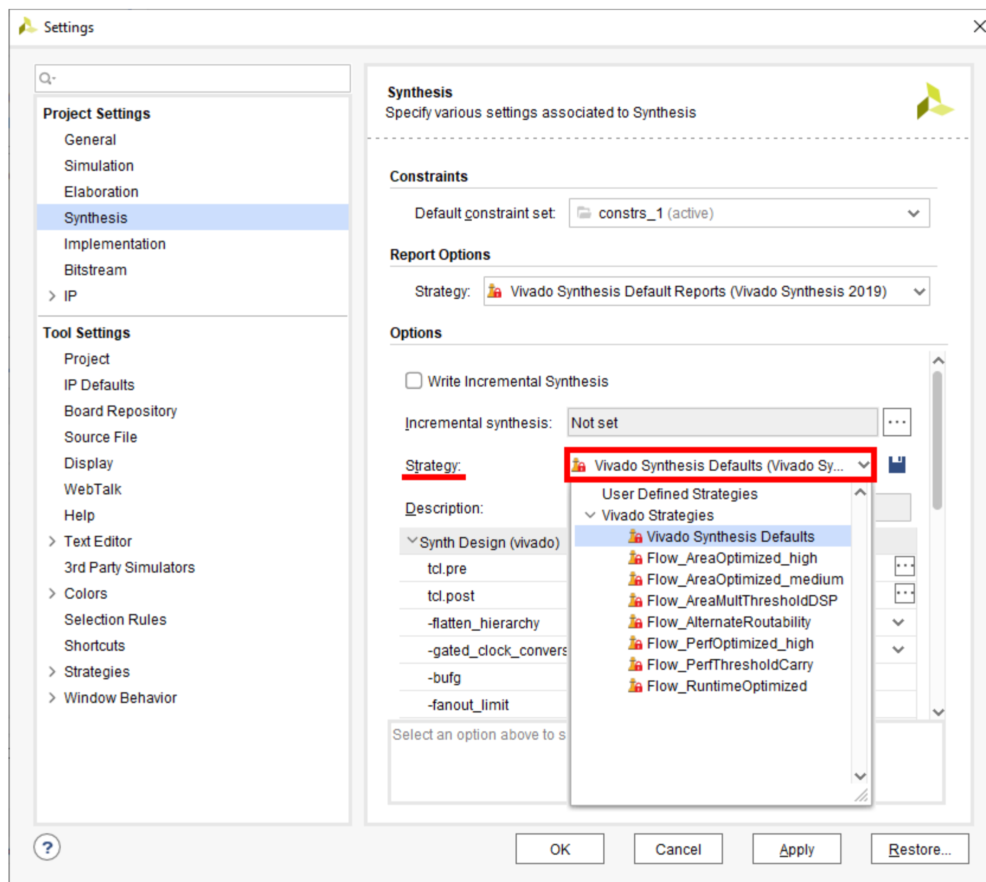
Tyrimo metu siekiama įvertinti bei tarpusavyje palyginti IP šerdžių skirtingų kūrimo metodų efektyvumą. IP šerdys ir jų funkcionalumas realizuojamas trimis skirtingais metodais tam skirtose programose: *HDL* kalba *Xilinx Vivado Design Suite* aplinkoje, *C++* kalba *Xilinx Vivado HLS* aplinkoje ir blokinėmis diagramomis *MathWorks Simulink Model Composer* aplinkoje. Šerdys kuriamos taip, kad atliktų tokį patį funkcionalumą, nerealizuotų perteklinių veiksmų bei naudotų tas pačias sąsajas, skirtas IP šerdies prijungimui programuojamos logikos matricų sistemose. Sukūrus IP šerdis, visos jos yra importuojamos į *Xilinx Vivado Design Suite* aplinką, kurioje atliekami tolimesni šerdžių tyrimai skirtingose programuojamos logikos matricų šeimose.

2.2.1. Tyrimo metodika

IP šerdys tiriamos keliais, dažniausiai naudojamais, sintezės strategijos atvejais [24]:

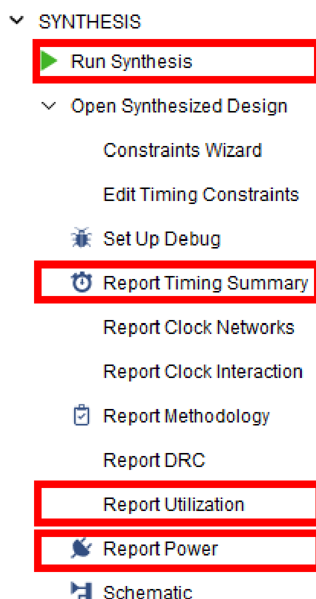
- *Defaults* – standartinė sintezės strategija, kuomet naudojama žemiausio lygio optimizacija.
- *Flow_AreaOptimized_high* – sintezės strategija, taikoma tam, kad būtų pasiekiamas mažesnis lusto ploto panaudojimas. Taikant šią strategiją optimizuojami sudėties veiksmai, pernešimo ir multipleksavimo grandinės.
- *Flow_AlternateRoutability* – sintezės strategija, taikoma siekiant pagerinti tarpusavio elementų sujungimus mažinant multipleksorių ir pernešimo grandinių skaičių.
- *Flow_PerfOptimized_high* – sintezės strategija, skirta sumažinti minimalų taktinio signalo periodą, tarpusavyje nekombinuojant peržiūros lentelių.
- *Flow_RuntimeOptimized* – sintezės strategija, skirta paspartinti IP šerdies sintezavimo procesą. Šios strategijos metu optimizacija nėra taikoma.

Dėl tyrime naudojamų kelių sintezės strategijų, importavus šerdį, ji pasirenkama *Xilinx Vivado Design Suite* aplinkoje spaudžiant *Įrankiai* (angl. *Tools*), *Nustatymai* (angl. *Settings*), *Sintezė* (angl. *Synthesis*), *Strategija* (angl. *Strategy*) (žr. 8 pav.).



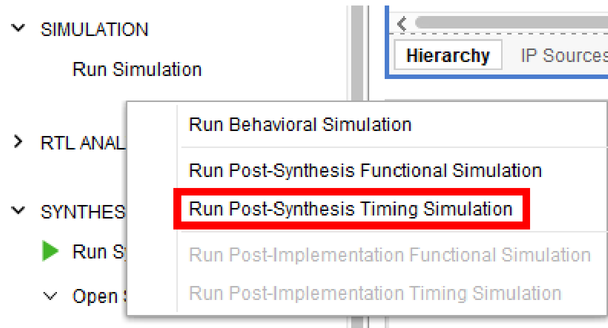
8 pav. Sintezės strategijos pasirinkimo langas *Xilinx Vivado Design Suite* aplinkoje

Pasirinkus sintezės strategiją atliekama IP šerdies sintezė skiltyje *Sintezė* spaudžiant *Pradėti sintezę (*angl. *Run Synthesis)* (žr. 9. pav.).



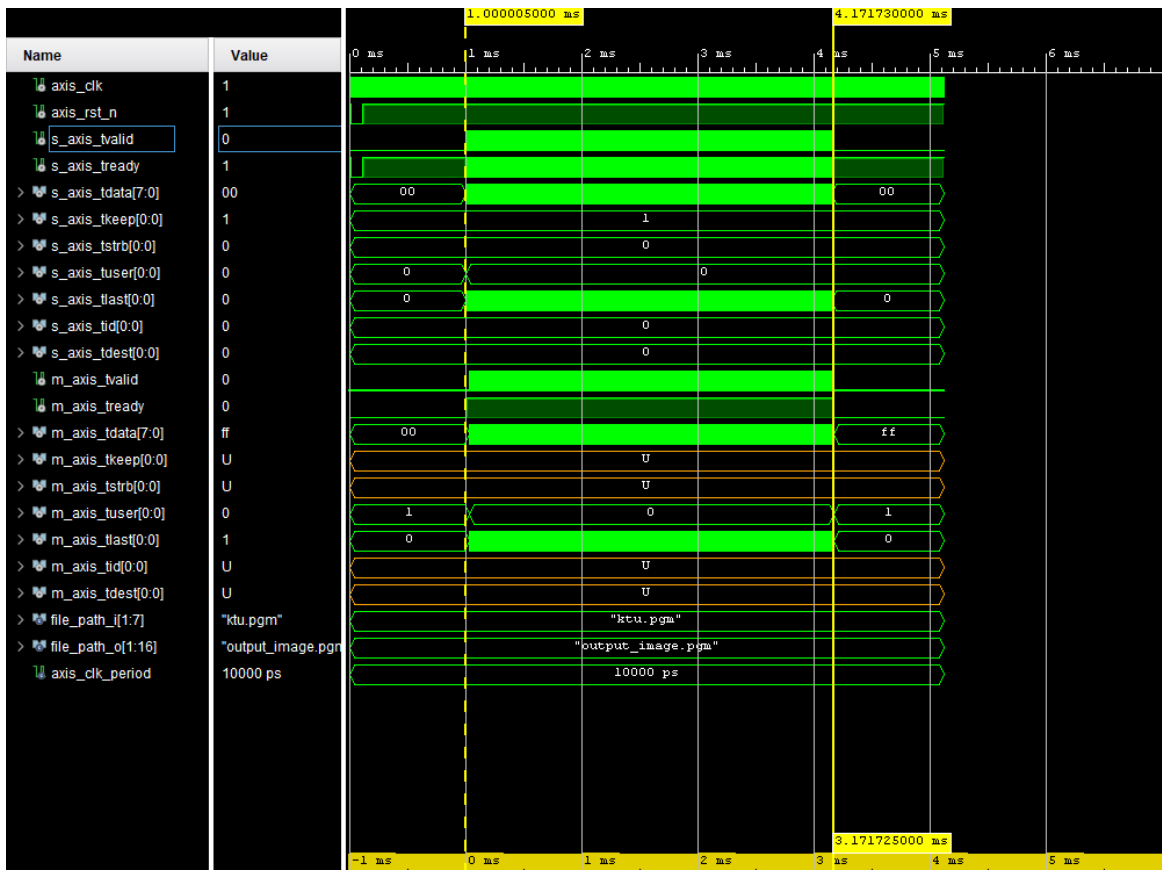
9 pav. Sintezės skilties pasirinkimų langas *Xilinx Vivado Design Suite* aplinkoje

Po sintezės įsitikinama jog IP šerdis yra veiksni atliekant laikinį modeliavimą skiltyje *Modeliavimas* (angl. *Simulation*) spaudžiant *Pradėti modeliavimą* (angl. *Run Simulation*), *Pradėti laikinį modeliavimą po sintezės gautai IP šerdžiai* (angl. *Run Post-Synthesis Timing Simulation*) (žr. 10. pav.).



10 pav. Laikinio modeliavimo pasirinkimas *Xilinx Vivado Design Suite* aplinkoje

Atliekant laikinį modeliavimą ir įsitikinus jog IP šerdis yra veiksni, išmatuojama uždavinio vykdymo trukmė. Šis parametras pateikiamas taktinių impulsų skaičiumi ir yra matuojamas naudojantis programiniais žymekliais (žr. 11 pav). Uždavinio vykdymo trukmė matuojama tarp pirmųjų duomenų patenkančių į IP šerdies įėjimą ir tarp paskutinių apdorotų duomenų IP šerdies išėjime.



11 pav. Laikinių parametrų matavimas programiniais žymekliais

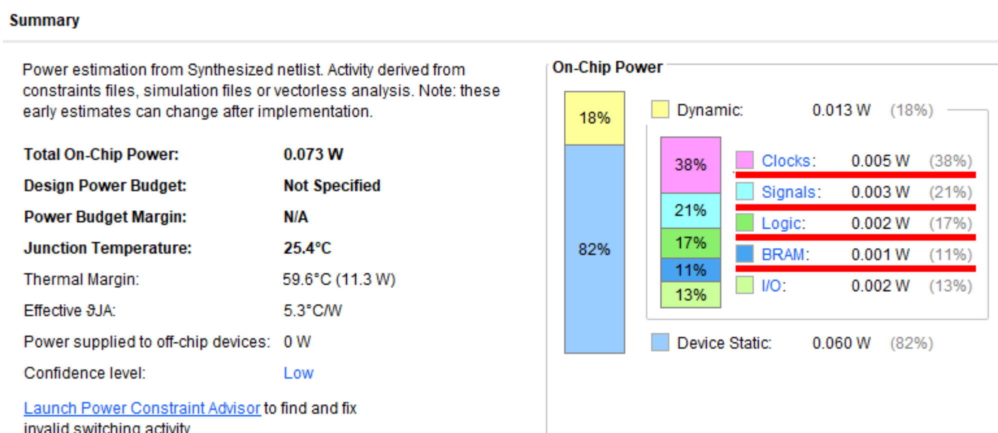
Minimalus taktinio signalo periodas paskaičiuojamas pagal toliau pateiktą formulę (3) [25].

$$T_{MIN} = T - WNS\,, \qquad (3)$$

čia T – nustatytas taktinio signalo periodas laikinių reikalavimų faile, WNS – laiko atsarga (angl. *worst negative slack*). Laiko atsarga nustatoma *Sintezės* skiltyje pasirinkus funkciją *Pateikti laikinių parametrų santrauką* (angl. *Report Timing Summary*). Atsidariusiame naujame lange šis parametras pateikiamas nanosekundėmis.

Užfiksavus visus laikinius parametrus, toliau fiksuojami panaudojami vidiniai resursai. Tai atliekama skiltyje *Sintezė* pasirenkant funkciją *Pranešti apie vidinių resursų panaudojimą* (angl. *Report Utilization*). Atsidariusiame lange panaudojamų vidinių resursų kiekis pateikiamas lentelėje [26].

Energetiniai parametrai fiksuojami skiltyje *Sintezė* pasirinkus funkciją *Pranešti apie suvartojamą galią* (angl. *Report Power).* Atsidariusiame naujame lange pateikiama pagrindinių elementų suvartojama momentinė galia milivatais (*mW*) (žr. 12 pav.). Siekiant pateikti apibendrintus rezultatus, nepriklausomus nuo tyrimo metu pasirinkto programuojamos logikos matricos įrenginio, vertinant suvartojamą galią, į galutinį rezultatą neįtraukiama statinė galia bei *I/O* (įėjimo ir išėjimo) prievadų suvartojama galia. Susumavus likusias galias, sumos rezultatas pateikiamas kaip momentinė IP šerdies suvartojama galia.



12 pav. Energetinių parametrų analizė *Xilinx Vivado Design Suite* aplinkoje

Užfiksavus visus vertinamus parametrus tyrimas pakartojamas su kita sintezės strategija. Tokiu būdu ištiriamos trimis skirtingais metodais sukurtos IP šerdys pasirinktoje programuojamos logikos matricos šeimoje. Vėliau tyrimas kartojamas iš naujo pasirinkus kitą programuojamos logikos matricos šeimą. Gauti duomenys pateikiami lentelėse, taip pat ir grafiškai naudojantis radaro tipo diagramomis. Patogesniam duomenų atvaizdavimui grafiškai duomenys normuojami šimto balų sistemoje. Tai atliekama naudojantis santykinio balo metodu, kuris apskaičiuojamas pagal toliau pateikiamą formulę (4) [27]:

$$T_i = 100 - \left(\frac{|x_i - Y|}{\sum_{j=1}^{N} x_j} \cdot 100\right), \qquad (4)$$

čia T_i – vertinamo parametro balas, N – vertinamos imties dydis, x_i – vertinamo parametro skaitinė išraiška, Y – geriausio parametro skaitinė išraiška. Toks duomenų normavimo metodas leidžia atvaizduoti skirtingų kategorijų, tipų ir dydžių duomenis viename grafike.

Galutinį IP šerdies kūrimo metodo įvertinimą siekiama pateikti kaip vieną, konkrečią skaitinę išraišką, kuri vartotojui leistų lengviau palyginti ir pasirinkti IP šerdžių kūrimo metodą. Ši skaitinė išraiška apskaičiuojama naudojantis reitingavimo (angl. *ranking*) metodu, kuriame rezultatas apskaičiuojamas taikant formulę (5) [28]:

$$Ef = \sum_{i=1}^{N} w_i T_i, \tag{5}$$

čia Ef – vertinamo IP šerdies kūrimo metodo efektyvumo balas, N – vertinamų parametrų skaičius, w_i – vertinamo parametro svertinis koeficientas skaičiuojant efektyvumo balą, T_i – vertinamo parametro skaitinė vertė (parametras normuotas santykiniu balo metodu).

Reitingavimo metodas leidžia subjektyviai įvertinti, palyginti bei pasirinkti tinkamą IP šerdžių kūrimo metodą taikant svertinius koeficientus. Priklausomai nuo sistemos dizaino reikalavimų, laisvų vidinių resursų ar kitų kriterijų, svarbesniems vertinamiems parametrams priskiriami aukštesni svertiniai koeficientai. Svertinių koeficientų suma privalo būti lygi vienetui.

Detalesnei IP šerdžių kūrimo metodų analizei naudojami standartiniai statistiniai metodai, kuriais apskaičiuojami pagrindiniai parametrai. Pirmiausiai randamas efektyvumo balo aritmetinis vidurkis, kuris apskaičiuojamas taikant formulę (6) [29]:

$$\overline{Ef} = \frac{1}{n}\left(\sum_{i=1}^{N} Ef_i\right), \tag{6}$$

čia $\overline{Ef}$ – vertinamo IP šerdies kūrimo metodo efektyvumo balo aritmetinis vidurkis, N – efektyvumo balo imties dydis, Ef_i – efektyvumo balo imties elementas.

Tuomet taikant pateiktą formulę (7) apskaičiuojama efektyvumo balo mediana [29]:

$$\widetilde{Ef} = \frac{Ef_{\lfloor N/2 \rfloor} + Ef_{\lceil (N+1)/2 \rceil}}{2}, \tag{7}$$

čia $\widetilde{Ef}$ – vertinamo IP šerdies kūrimo metodo efektyvumo balo mediana, N – efektyvumo balo imties dydis, Ef – efektyvumo balo imties elementas, $\lfloor\ \rfloor$ – žymimas apvalinimo veiksmas į mažesnio skaičiaus pusę, $\lceil\ \rceil$ – žymimas apvalinimo veiksmas į didesnio skaičiaus pusę.

Taip pat randamas efektyvumo balo standartinis nuokrypis, kuris apskaičiuojamas taikant formulę (8) [30]:

$$S_{Ef} = \sqrt{\frac{\sum_{i=1}^{N}(Ef_i - \overline{Ef})}{N-1}}, \tag{8}$$

Čia S_{Ef} – vertinamo IP šerdies kūrimo metodo efektyvumo balo standartinis nuokrypis, Ef_i – efektyvumo balo imties elementas, $\overline{Ef}$ – vertinamo IP šerdies kūrimo metodo efektyvumo balo aritmetinis vidurkis, N – efektyvumo balo imties dydis.

2.2.2. Tyrimo planas

Pateikiamas apibendrintas tyrimo eigos planas eilės tvarka, t. y. žingsniais.

1. IP šerdys, realizuotos *HDL* kalba *Xilinx Vivado Design Suite* aplinkoje, *C++* kalba *Xilinx Vivado HLS* aplinkoje ir blokinėmis diagramomis *MathWorks Simulink Model Composer* aplinkoje, importuojamos į *Xilinx Vivado Design Suite* programinę įrangą.
2. Pasirenkama viena programuojamos logikos matricos šeima iš tiriamųjų šeimų sąrašo: *Spartan-7, Artix-7, Kintex-7, ZynqUltrascale+*.
3. Pasirenkama viena iš tiriamųjų IP šerdžių.
4. Pasirenkama viena sintezės strategija iš tiriamųjų strategijų sąrašo: *Defaults*, *Flow_AreaOptimized, Flow_AlternateRoutability, Flow_PerfOptimized_high, Flow_RuntimeOptimized.*
5. Atliekama IP šerdies sintezė.
6. Atlikus sintezę nustatoma ar IP šerdis yra veiksni, užfiksuojami visi tiriami parametrai: sunaudojamų vidinių resursų skaičius, laikiniai parametrai bei energetiniai parametrai.
7. Atlikus tyrimą su pasirinkta sintezės strategija grįžtama į 4 žingsnį. Sintezės strategija pakeičiama ir atliekami 5–6 žingsniai.
8. Atlikus tyrimą su visomis sintezės strategijomis grįžtama į žingsnį 3. Pasirenkama kita tiriama IP šerdis ir atliekami 4–7 žingsniai.
9. Atlikus tyrimą su visomis realizuotoms IP šerdims grįžtama į žingsnį 2. Pakeičiama tiriama programuojamos logikos matricos šeima ir atliekami žingsniai 3–8.
10. Atlikus tyrimą visoms programuojamos logikos matricos šeimomis gauti duomenys pateikiami lentelėse.
11. Duomenys sunormuojami šimto balų sistemoje ir grafiškai atvaizduojami radaro tipo grafikuose.
12. Šiuolaikinių IP šerdžių kūrimo metodų įvertinimui pateikiami efektyvumo balai ir jų statistiniai parametrai. Remiantis gautais duomenimis atliekamas metodų palyginimas, pateikiamos išvados.

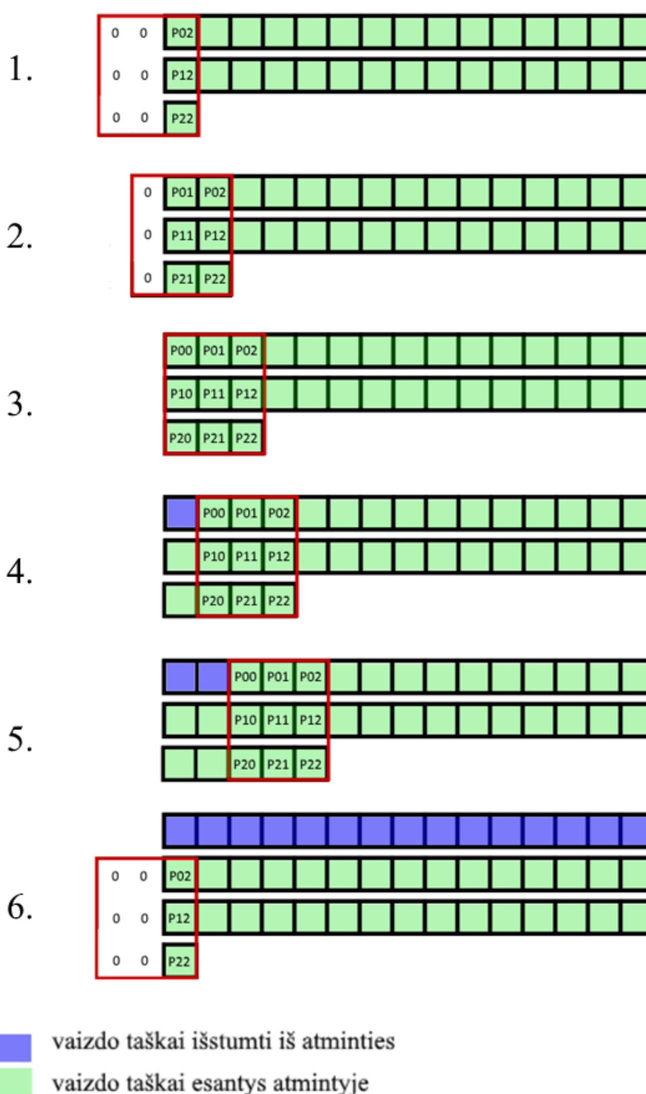
3. IP šerdžių realizacija ir tyrimas

3.1. IP šerdžių realizacija

IP šerdžių palyginimui atlikti reikalingi duomenys gaunami tiriant sukurtas vaizdų apdorojimui skirtas šerdis. Šios šerdys apdorojamoje nuotraukoje ar vaizdo įraše išskiria užfiksuotų objektų briaunas. Objektų briaunų išskyrimui naudojama dvimatė sąsūka (angl. *two-dimensional convolution*), kuri apskaičiuojama naudojantis trijų eilučių ir trijų stulpelių *Canny* šablonu (angl. *kernel*). Objektų briaunų išskyrimui taip pat naudojamas filtravimas, kuris atliekamas tikrinant ar apdorojamo vaizdo taško intensyvumo reikšmė viršija užduotą slenkstinę reikšmę.

3.1.1. Slenkančio lango algoritmas

Dėl programuojamos logikos matricų architektūrinių savybių dvimatei sąsūkai realizuoti pasirinktas slenkančio lango (angl. *sliding window*) algoritmas [31]. Šis metodas leidžia pasiekti mažiausią atminties resursų panaudojimą. Slenkančio lango algoritmas taip pat leidžia pasiekti mažesnį minimalų IP šerdies taktinį signalo periodą nei kiti įprastiniai metodai, kadangi duomenys apdorojami kaskadiškai. Pateikiamas grafinis slenkančio lango metodo veikimo principo paaiškinimas (žr. 13 pav.).



13 pav. Slenkančio lango metodo grafinis veikimo principo paaiškinimas [32]

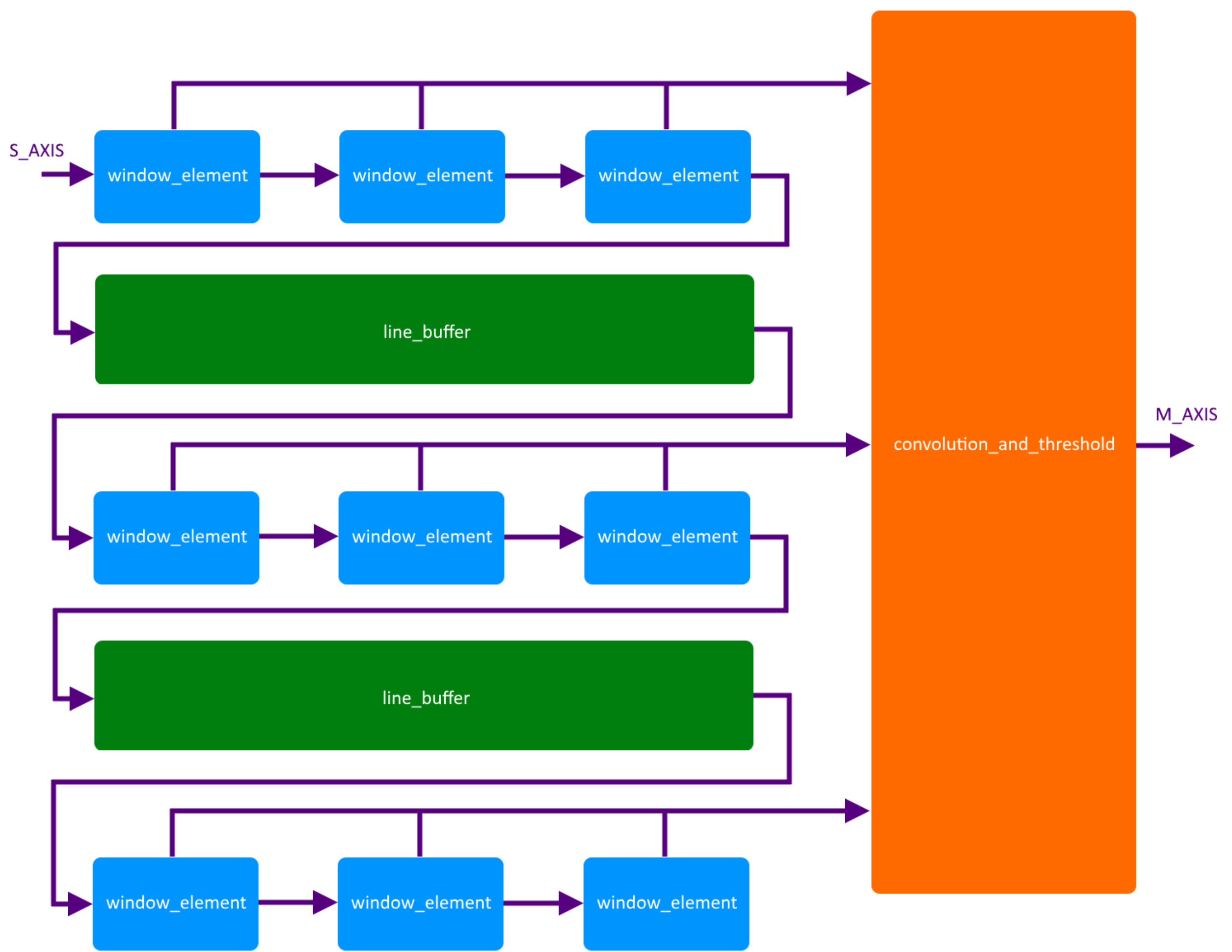
Toliau trumpai aptariamas kiekvienas žingsnis, apibūdinantis slenkančio trijų eilučių ir trijų stulpelių lango metodo veikimo principą.

1. Dvimatė sąsūka pradedama skaičiuoti tik tuomet, kai atmintyje išsaugomos pirmos dvi apdorojamo vaizdo eilutės. Tuomet užfiksavus einamojo kadro taško vertę P22 atliekamas skaičiavimas ir gautas rezultatas pateikiamas kaip pirmasis apdoroto vaizdo taškas.
2. Užfiksavus sekančią einamojo kadro taško vertę P22 prieš tai buvusios vertės yra perslenkamos į kairę išsaugant jas atmintyje. Apskaičiuota dvimatės sąsūkos reikšmė pateikiama kaip antrasis apdoroto vaizdo taškas.
3. Užfiksavus naujesnę einamojo kadro taško reikšmę P22, prieš tai buvusios vertės yra perslenkamos į kairę išsaugant jas atmintyje. Apskaičiuota dvimatės sąsūkos reikšmė pateikiama kaip trečiasis apdoroto vaizdo taškas.
4. Užsipildžius langui kadro reikšmėmis seniausia išsaugota reikšmė yra išstumiama iš atminties, o buvusios reikšmės perstumiamos į kairę. Užfiksavus naujesnę einamojo kadro taško reikšmę P22 apskaičiuotas dvimatės sąsūkos rezultatas pateikiamas kaip ketvirtasis apdoroto kadro taškas.
5. Užfiksavus sekančią einamojo kadro taško vertę P22, seniausia saugoma vertė yra išstumiama iš atminties, o užfiksuotos reikšmės perstumiamos į kairę. Apskaičiuota dvimatės sąsūkos reikšmė pateikiama kaip penktasis apdoroto vaizdo taškas.
6. Langas tokiu pat principu yra slenkamas per visą apdorojamą kadrą. Senosios vertės yra išstumiamos iš atminties, išsaugotos vertės perslenkamos į kairę, o naujosios užfiksuotos vaizdo taško reikšmės saugomos atmintyje. Dvimatės sąsūkos rezultatai skaičiuojami naudojantis lange esančiomis vertėmis ir pateikiami kaip apdoroto kadro vertės.

Naudojantis minėtais apdorojimo algoritmais ir metodais IP šerdys realizuojamos *Xilinx Vivado Design Suite* aplinkoje naudojantis *VHDL* kalba, *Xilinx Vivado HLS* aplinkoje naudojantis *C++* kalba bei *MathWorks Simulink Model Composer* aplinkoje naudojantis iš anksto realizuotais ir optimizuotais programiniais blokeliais. Visos minėtos šerdys realizuojamos su *AXI4-Stream* magistrale duomenų įvedimui ir išvedimui, kadangi šis duomenų magistralės tipas dažniausiai naudojamas vaizdų apdorojimui skirtose IP šerdyse.

3.1.2. IP šerdies realizacija *Xilinx Vivado Design Suite* aplinkoje

IP šerdies realizavimui *Xilinx Vivado Design Suite* aplinkoje naudojami anksčiau aptarti vaizdų apdorojimo metodai ir jų realizacija (slenkančio lango metodas), kuri aprašoma *VHDL* kalba. Pateikiama *VHDL* kalba aprašytos IP šerdies struktūros blokinė diagrama (žr. 14 pav.).



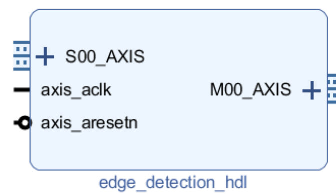
14 pav. *VHDL* kalba realizuojamos IP šerdies struktūros blokinė diagrama

VHDL kalba realizuotą IP šerdį sudaro šie pagrindiniai elementai:

- window_element – lango elementas, kurio reikšmė naudojama dvimatės sąsūkos rezultato skaičiavimui. Lango elemente saugojamos vaizdo taško reikšmės, kurios su kiekvienu taktiniu signalu yra perduodamos iš vieno lango elemento į kitą, taip pat ir į vaizdo eilutės saugojimo elementus. .
- line_buffer – vaizdo eilutės saugojimo elementas, skirtas išsaugoti vienos kadro eilutės duomenis. Su kiekvienu taktiniu signalu vaizdo taško reikšmė iš lango elemento yra įrašoma į atmintį. Taip pat kiekvieno taktinio signalo metu iš atminties nuskaitoma seniausia išsaugota vaizdo taško reikšmė ir perduodama į sekantį lango elementą. Vaizdo eilutės saugojimo elementas *VHDL* kalboje realizuojamas pasinaudojant *FIFO* atminties primityvais, todėl duomenys saugomi *BRAM* architektūriniuose elementuose.
- *convolution_and_threshold* – sąsūkos ir slenkstinės reikšmės skaičiavimo elementas. Šiame elemente su kiekvienu taktiniu signalu reikšmės nuskaitomos iš devynių lango elementų, jos sudauginamos su *Canny* šablono koeficientais ir susumuojamos. Gautas dvimatės sasūkos rezultatas yra palyginamas su užduota slenkstine reikšme. Daugybos veiksmai elemente realizuojami naudojantis loginiais poslinkiais.

IP šerdis *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuojama per 96 valandas. Realizuotos šerdies kodas pateikiamas prieduose (žr. 1 priedas. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies kodas). Šerdis išsaugoma *Xilinx Vivado Design Suite* aplinkos kataloge, vėliau naudojantis

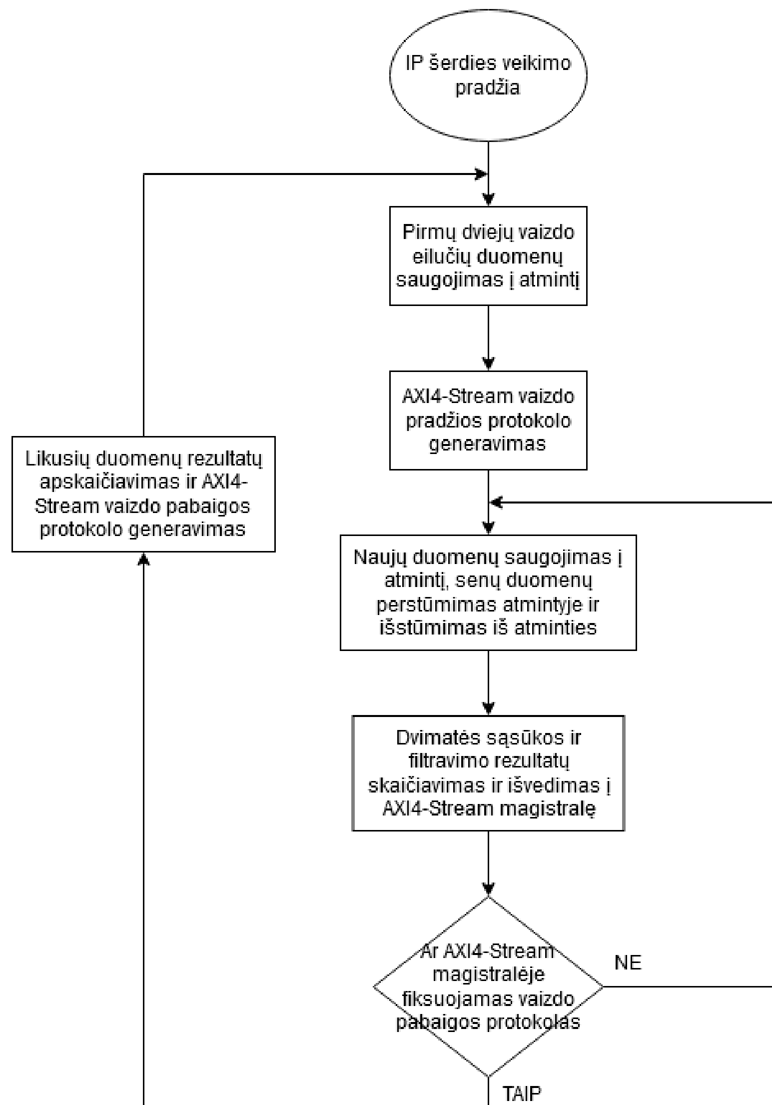
šia aplinka bus atliekamas tolimesnis tyrimas. Pateikiamas realizuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje (žr. 15 pav.).



15 pav. Realizuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje

3.1.3. IP šerdies realizacija *Xilinx Vivado HLS* aplinkoje

Xilinx Vivado HLS aplinkoje IP šerdies realizavimui naudojami anksčiau aptarti vaizdų apdorojimo metodai ir jų realizacija, kuri aprašoma *C++* kalba. Pateikiamas *C++* kalba aprašytos IP šerdies veikimo algoritmas (žr. 16 pav.).



16 pav. *C++* kalba aprašytos IP šerdies veikimo algoritmas

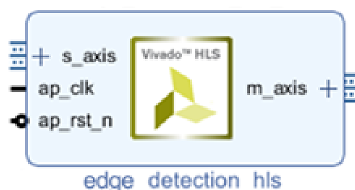
IP šerdies kodas aprašomas naudojantis standartiniais *C++* kalbos ciklais, masyvais, klasėmis bei matematinėmis ir loginėmis operacijomis (sudėtimi, daugyba, loginiais patikrinimais):

- ciklai naudojami vaizdo taškų skaičiavimui, atmintyje saugomų duomenų užpildymui bei iteracijai per šiuos duomenis.
- masyvais realizuojamas slenkantis langas, kuriame saugomi vaizdo tašai bei Canny šablonas, kuriame saugomi sąsūkai naudojami daugybos koeficientai.
- klasės naudojamos AXI4-Stream magistralėms realizuoti. Jos apibrėžią pačią magistralės sandarą (nusako magistralę sudarančius signalus), taip pat realizuoja funkcijas, kurios skirtos nuskaityti ir perduoti duomenis šio tipo magistralėse.
- matematinės ir loginės operacijos naudojamos dvimatės sąsūkos rezultato skaičiavimui bei gautos reikšmės palyginimui su užduota slenkstine verte.

Realizuotas kodas yra papildomas *Xilinx Vivado HLS* aplinkos specifiniais atributais (angl. *pragma*), kurie leidžia efektyviau panaudoti programuojamos logikos matricos architektūrinius resursus atliekant šias funkcijas [33]:

- *#pragma HLS INTERFACE axis port* – sukuria pasirinkto tipo (šiuo atveju *AXI4-Stream*) magistralę duomenų įvedimui arba išvedimui į arba iš IP šerdies, taip pat pasirūpina magistralės protokolo generavimu.
- *#pragma HLS INTERFACE ap_ctrl_none port* – panaikina papildomus signalus, skirtus informuoti apie duomenų mainų pradžią, IP šerdies būseną bei įvykdytą užduotį. Šie papildomi signalai yra nereikalingi kadangi AXI4-Stream magistralė šiuos informacinius signalus jau turi savyje.
- *#pragma HLS ARRAY_PARTITION variable* – padalina pasirinktą masyvą į smulkesnius masyvus ar atskirus narius. Padalinimas į smulkesnius masyvus ar atskirus elementus leidžia užtikrinti efektyvesni duomenų nuskaitymą iš masyvo taip padidinant IP šerdies greitaveiką.
- *#pragma HLS PIPELINE* – leidžia ciklinį procesą vykdyti lygiagrečiai išskaidant jį į atskirus vykdymo žingsnius taip padidinant IP šerdies greitaveiką.

IP šerdis *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuojama per 23 valandas. Programos kodas pateikiamas prieduose (žr. 2 priedas. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies kodas). Realizuota šerdis naudojantis *Xilinx Vivado HLS* aplinkoje pateiktais įrankiais yra išeksportuojama kaip archyvas, kuris importuojamas į *Xilinx Vivado Design Suite* aplinką. Šioje aplinkoje vėliau bus atliekamas tolimesnis tyrimas. Pateikiamas importuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje (žr. 17 pav.).

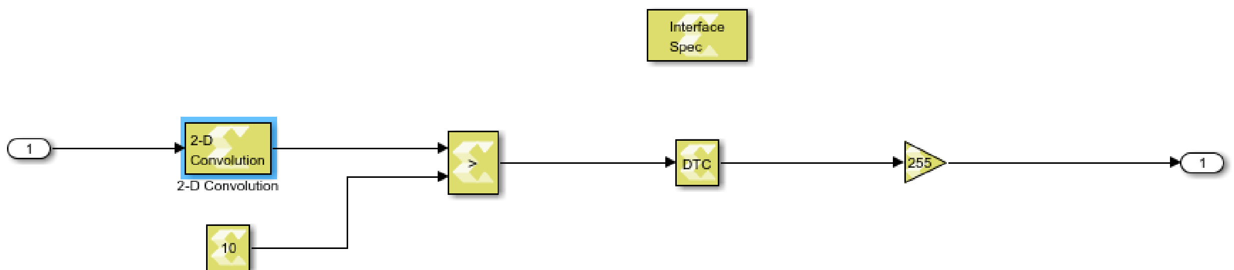


17 pav. Realizuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje

3.1.4. IP šerdies realizacija *MathWorks Simulink Model Composer* aplinkoje

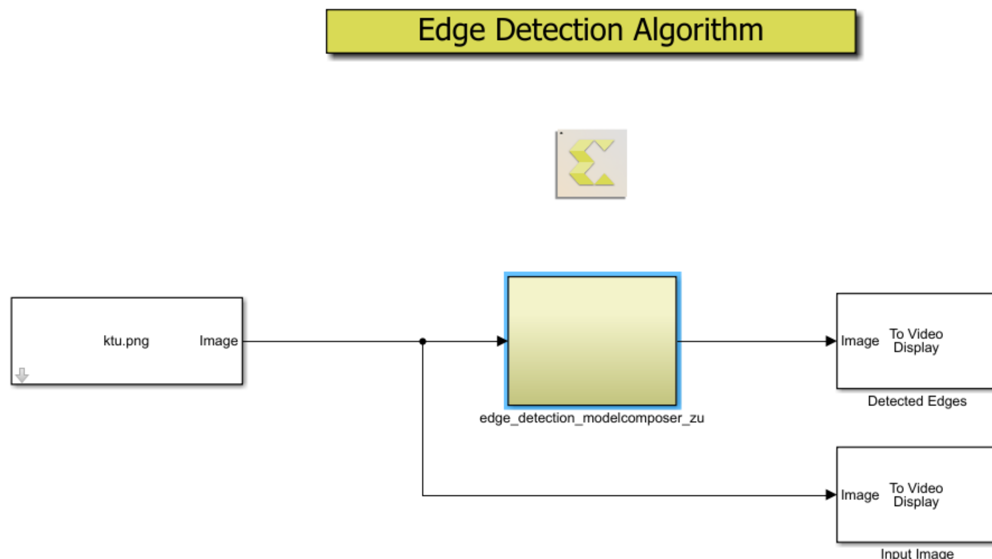
MathWorks Simulink Model Composer aplinkoje vaizde esančiu objektų briaunų aptikimui skirta IP šerdis realizuojama naudojantis dvimatei sąsūkai skirtu blokeliu, kurio veikimo principas paremtas slenkančio lango metodu [34]. Šiame blokelyje nurodomos sąsūkos skaičiavimo metu naudojamo šablono koeficientų reikšmės, *Canny* šablono atveju jos yra lygios šioms reikšmėms *C = {0, -1, 0; -1, 4, -1; 0, -1, 0}*. Filtravimas atliekamas naudojantis palyginimo operacijos blokeliu, kurio

gražinamos išėjimo reikšmės yra lygios loginiam nuliui, jei lyginamas dydis yra mažesnis ar lygus užduotai slenkstinei reikšmei, arba loginiam vienetui, jei lyginamas dydis yra didesnis nei užduota slenkstinė reikšmė. Kadangi IP šerdis skirta apdoroti vaizdams, kurių taškų gylis yra 8 bitai, gauta reikšmė po palyginimo operacijos yra padauginama iš konstantos *cons* = *255* tam, kad būtų panaudojamas visas vaizdo taško gylio diapazonas. Pateikiama realizuotos IP šerdies blokinė *MathWorks Simulink Model Composer* aplinkos diagrama (žr. 18 pav.).



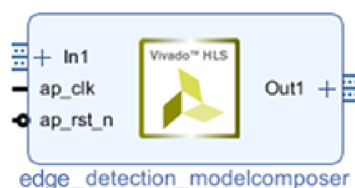
18 pav. Realizuotos IP šerdies blokinė *MathWorks Simulink Model Composer* aplinkos diagrama

Norint išeksportuoti sukurtą IP šerdį kaip archyvą į *Xilinx Vivado Design Suite* aplinką tolimesniems tyrimams, sukuriamas blokinės diagramos apvalkalas (angl. *wrapper*), kuris leidžia atlikti blokinės diagramos testavimą, paruošimą konvertavimui į *HDL* arba *Verilog* kalbos kodą bei gautų konvertavimo rezultatų supakavimą į archyvą. Pateikiama realizuoto apvalkalo blokinė *MathWorks Simulink Model Composer* aplinkos diagrama (žr. 19 pav.).



19 pav. Realizuotos IP šerdies apvalkalo blokinė *MathWorks Simulink Model Composer* aplinkos diagrama

IP šerdis *MathWorks Simulink Model Composer* aplinkoje realizuojama per 2 valandas, atlikus šerdies suarchyvavimą ji importuojama į *Xilinx Vivado Design Suite* aplinką. Šioje aplinkoje vėliau bus atliekamas tolimesnis tyrimas. Pateikiamas importuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje (žr. 20 pav.).



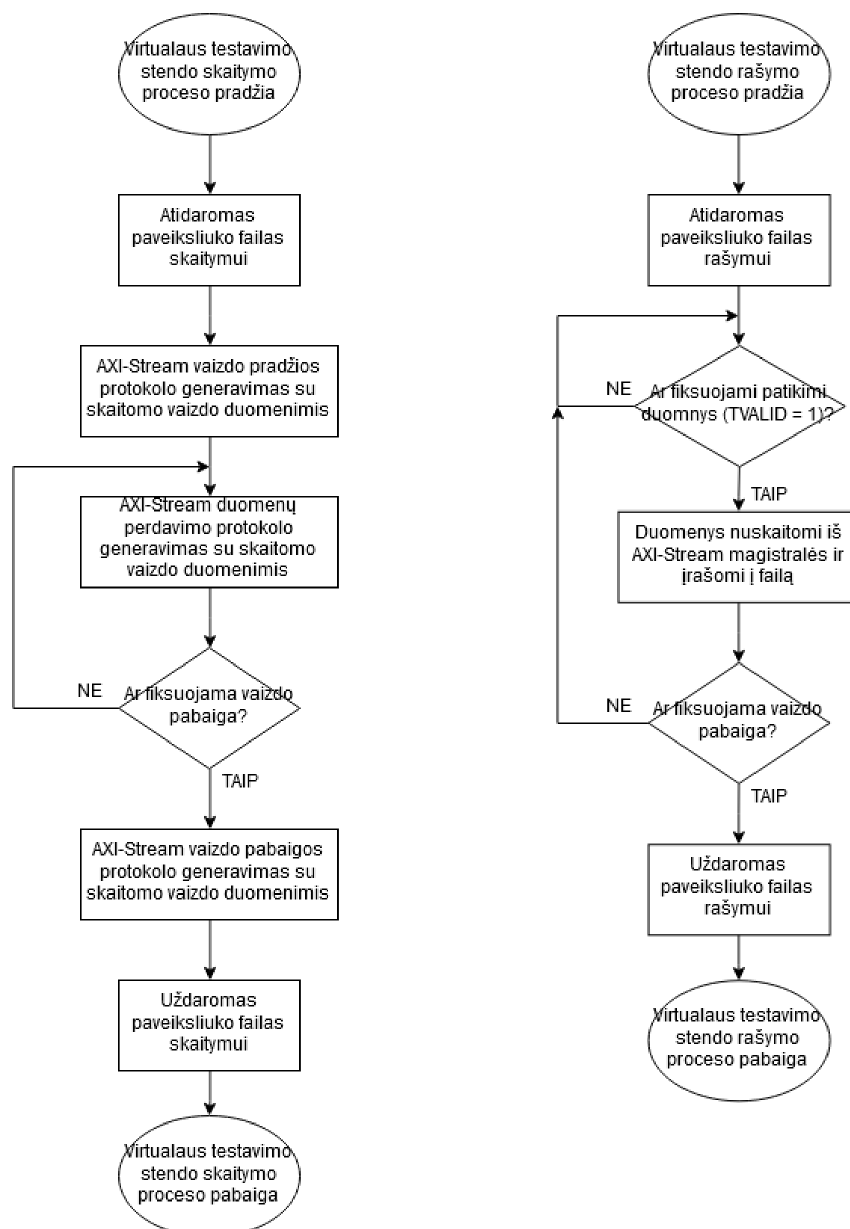
20 pav. Realizuotos IP šerdies blokinis simbolis *Xilinx Vivado Design Suite* aplinkoje

3.2. IP šerdžių tyrimas

IP šerdžių tyrimui atlikti visos prieš tai realizuotos IP šerdys importuojamos į *Xilinx Vivado Design Suite* aplinką. Laikinių parametrų matavimui ir IP šerdies funkcionalumo testavimui naudojamas virtualus testavimo stendas (angl. *testbench*), aprašytas VHDL kalba. Virtualus testavimo stendas realizuojamas dviem, lygiagrečiai veikiančiais, procesais:

- *Skaitymo procesas* – šio proceso metu atidaromas stende aprašytas *pgm* (*portable graymap*) formato vaizdo failas. Šio formato faile kiekvienoje atskiroje eilutėje pateikiami vaizdo taško intensyvumo duomenys. Skaitymo procesas su kiekviena iteracija nuskaito vis naują eilutę ir priklausomai nuo skaitomo taško pozicijos vaizde generuoja *AXI4-Stream* magistralės duomenų perdavimo protokolo signalus. Galimi šie vaizdo duomenų perdavimo protokolai: vaizdo pradžios, įprastinio vaizdo bei vaizdo pabaigos. Šie sugeneruoti signalai perduodami į tiriamos IP šerdies įėjimą *S_AXIS*. Nuskaičius visą vaizdo failą ir jo duomenys perdavus *AXI4-Stream* magistrale failas yra uždaromas ir laukiama kol programos vykdymas bus nutrauktas kito proceso.
- *Rašymo procesas* – šio proceso metu sukuriamas ir atidaromas naujas *pgm* formato failas. Su kiekviena iteracija tikrinama ar IP šerdies duomenų išėjimo magistralėje *M_AXIS* generuojami patikimi (angl. *valid*) duomenys. Jei duomenys patikimi, jie yra nuskaitomi ir įrašomi į vis naują atidaryto failo eilutę. Užfiksavus vaizdo pabaigą rašymui skirtas failas yra uždaromas ir virtualus testavimo stendas yra sustabdomas. Vartotojas informuojamas apie virtualaus testavimo stendo pabaigą pranešimu informaciniame lange.

Testavimui skirti ir testavimo metu sugeneruoti *pgm* formato failai gali būti peržiūrimi įvairiomis programomis, viena iš jų – *GIMP (GNU Image Manipulation Program)* [35]. Pateikiamas virtualaus testavimo stendo veikimo algoritmas (žr. 21 pav.).



21 pav. Virtualaus testavimo stendo veikimo algoritmas

Realizuoto virtualaus testavimo stendo kodas pateikiamas prieduose (žr. Priedas 3. *Xilinx Vivado Design Suite* aplinkoje naudojamo testo kodas). Tyrimo ir testavimo metu kaip virtualaus testavimo stendo stimulas naudojama Kauno technologijos universiteto nuotrauka, kuri yra konvertuojama į pustonį vaizdą (angl. *grayscale*) ir išsaugoma *pgm* failo formatu. Pateikiamas virtualaus testavimo stendo stimulo parametrai (žr. 12 lentelę) bei vaizdas (žr. 22 pav.).

12 lentelė. Virtualaus testavimo stendo stimulo parametrai

Parametras	**Išraiška**
Aukštis, h	640
Plotis, w	480
Taško gylis, bpp	8
Formatas	RAW8

22 pav. Virtualiame testavimo stende naudoto stimulo vaizdas [36]

Pateikiami trimis skirtais metodais sukurtų IP šerdžių virtualaus testavimo stendo rezultatai: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies testavimo rezultatas (žr. 23 pav.), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies testavimo rezultatas (žr. 24 pav.), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies testavimo rezultatas (žr. 25 pav.). IP šerdys atlieka tą patį funkcionalumą, todėl esminių skirtumų tarp gautų rezultatų nėra.

23 pav. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies testavimo rezultatų vaizdas

24 pav. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies testavimo rezultatų vaizdas

25 pav. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies testavimo rezultatų vaizdas

3.2.1. IP šerdžių tyrimas *Spartan-7* šeimoje

Toliau pagal aprašytą tyrimo metodiką ištiriamos IP šerdys, realizuotos trimis skirtingais kūrimo metodais. Pirmiausia tyrimas atliekamas su *Spartan-7* programuojamos logikos matricos šeima, taikant skirtingas sintezės optimizacijos strategijas. Tyrimo metu gauti rezultatai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai (žr. 13 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai (žr. 14 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai (žr. 15 lentelę).

13 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	238	231	285	242	254
FF	329	329	367	442	409
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ***ciklai***	308491	308491	308491	308491	308491
T_{MIN}, ***ns***	5.20	5.29	5.20	5.20	5.22
P, mW	7	7	7	8	8

14 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai

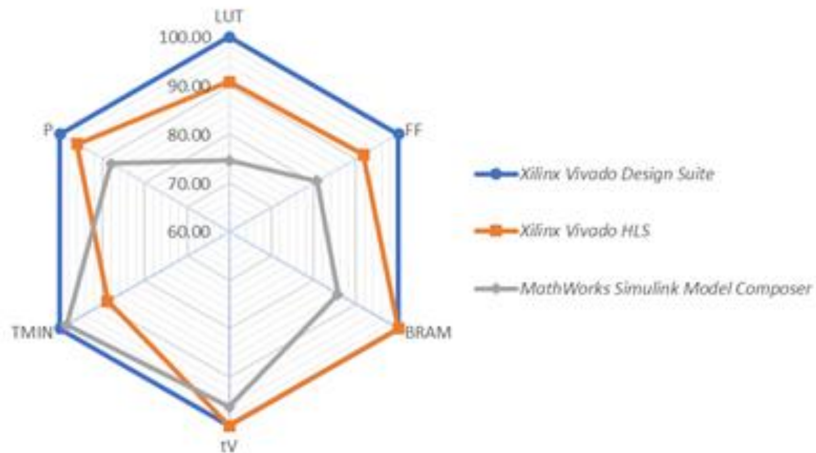
***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	339	355	485	689	405
FF	442	442	442	769	442
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ***ciklai***	308514	308514	308514	308514	308514
T_{MIN}, ***ns***	7.21	7.20	6.08	6.26	6.08
P, mW	8	8	8	11	8

15 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai

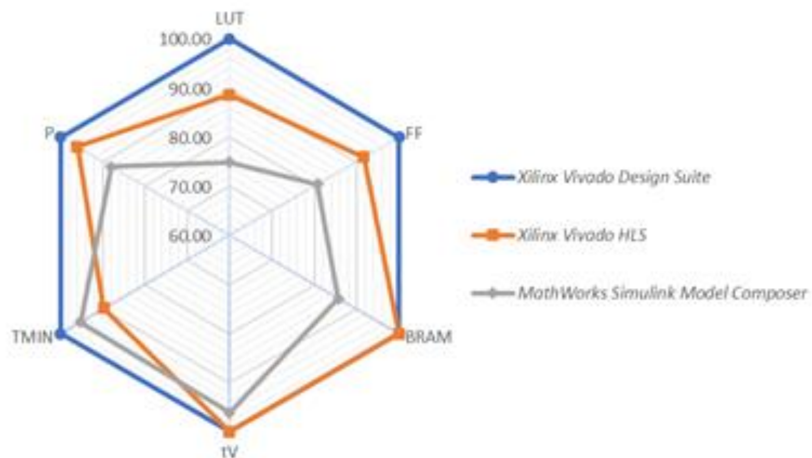
***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	516	505	714	642	**-**
FF	591	591	593	627	**-**
BRAM	1.5	1.5	1.5	1.5	**-**
DSP	0	0	0	0	**-**
t_V, ***ciklai***	346565	346565	346565	346565	**-**
T_{MIN}, ***ns***	5.48	6.16	5.60	5.48	**-**
P, mW	10	10	11	10	**-**

Pastaba: brūkšneliais pažymėti IP šerdies, kuriai pritaikius pasirinktą optimizacijos tipą ji prarado savo funkcionalumą (nustojo veikti), parametrai.

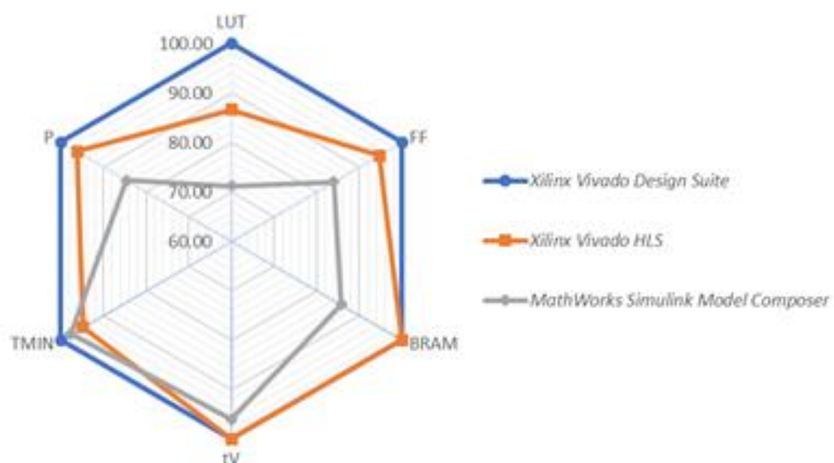
Gauti rezultatai normuojami šimto balų sistemoje taikant tyrimo metodikoje nurodytą formulę (4). Normuoti rezultatai pateikiami radaro tipo grafikuose: IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults*" tipo optimizaciją (žr. 26 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized*" tipo optimizaciją (žr. 27 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability*" tipo optimizaciją (žr. 28 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*PerformanceOptimized*" tipo optimizaciją (žr. 29 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*RuntimeOptimized*" tipo optimizaciją (žr. 30 pav.).



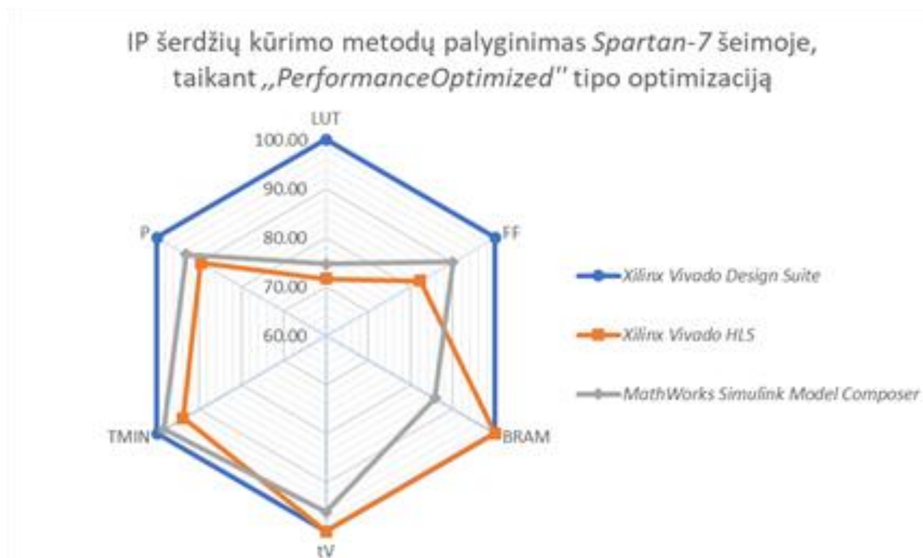
26 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults''* tipo optimizaciją



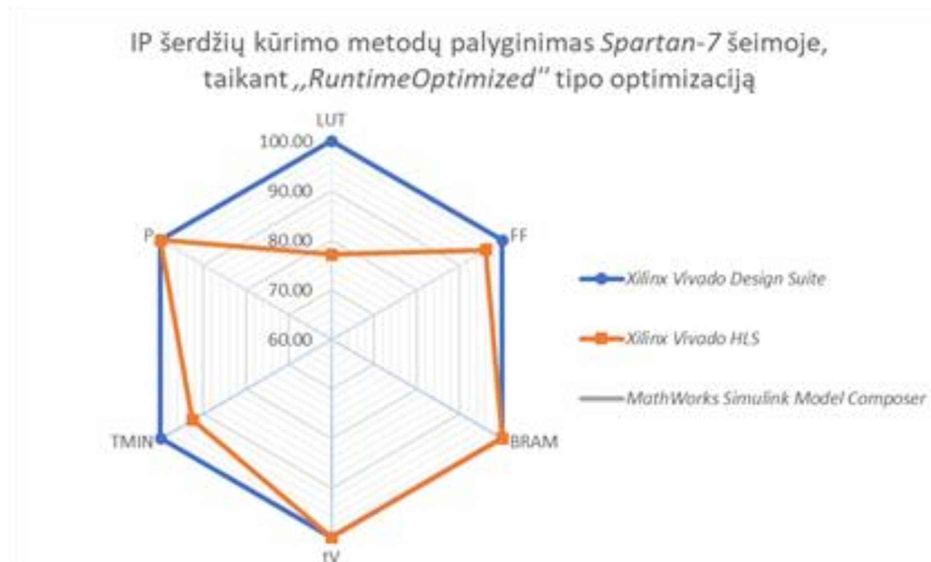
27 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized''* tipo optimizaciją



28 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability''* tipo optimizaciją



29 pav. IP šerdžių kūrimo metodų palyginimas taikant *„PerformanceOptimized"* tipo optimizaciją



30 pav. IP šerdžių kūrimo metodų palyginimas taikant *„RuntimeOptimized"* tipo optimizaciją

Atlikus tyrimą *Spartan-7* šeimoje pastebėta jog *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis yra efektyviausia, taip pat pastebėta jog gauti rezultatai yra panašaus pobūdžio trijuose optimizacijų tipuose: *„Defaults", „AreaOptimized" ir „AlternateRoutability".* Esant šio tipo optimizacijoms *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis energetinių parametrų ir panaudojamų vidinių resursų atžvilgiu yra efektyvesnė už *MathWorks Simulink Model Composer* aplinkoje modeliais realizuotą IP šerdį, tačiau optimizacijos tipui pasikeitus į *„PerformanceOptimized"* pastebimas rezultatų pobūdžio pasikeitimas. *MathWorks Simulink Model Composer* aplinkoje modeliais realizuotos šerdies vidinių resursų panaudojimas tampa efektyvesnis, dėl to pagerėja energetiniai parametrai, lyginant su *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotos šerdies parametrais, kurie suprastėja. Optimizacijos tipui pasikeitus į *„RuntimeOptimized" MathWorks Simulink Model Composer* aplinkoje modeliais realizuota šerdis tampa neveiksni. *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotos šerdies rezultatų pobūdis tampa panašus į pirmų trijų aptartų optimizacijos tipų rezultatų pobūdį, pastebimas *LUT* architektūrinių elementų panaudojimo efektyvumo sumažėjimas bei energetinių parametrų pagerėjimas.

3.2.2. IP šerdžių tyrimas *Artix-7* šeimoje

Tyrimas pakartojamas su *Artix-7* programuojamos logikos matricos šeima, taikant skirtingas sintezės optimizacijos strategijas. Tyrimo metu gauti rezultatai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai (žr. 16 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai (žr. 17 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai (žr. 18 lentelę).

16 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	238	231	285	242	254
FF	329	329	367	442	409
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ***ciklai***	308491	308491	308491	308491	308491
T_{MIN}, ***ns***	4.22	4.29	4.23	4.23	4.24
P, mW	7	7	7	8	8

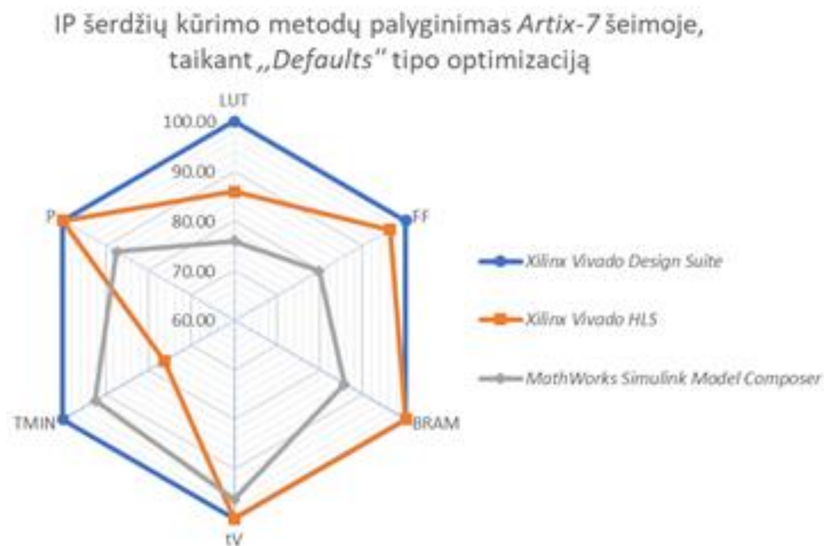
17 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	401	346	478	715	399
FF	376	376	376	703	376
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ***ciklai***	308514	308514	308514	308514	308514
T_{MIN}, ***ns***	8.58	8.54	8.64	8.13	8.58
P, mW	7	7	7	10	7

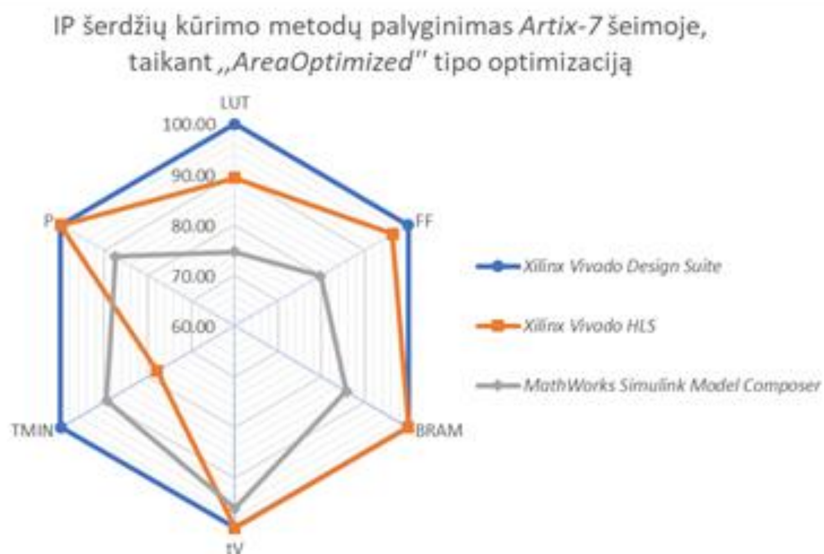
18 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	516	505	710	624	**-**
FF	591	591	593	627	**-**
BRAM	1.5	1.5	1.5	1.5	**-**
DSP	0	0	0	0	**-**
t_V, ***ciklai***	346167	346167	346167	346167	**-**
T_{MIN}, ***ns***	5.63	6.30	5.66	5.63	**-**
P, mW	10	10	11	10	**-**

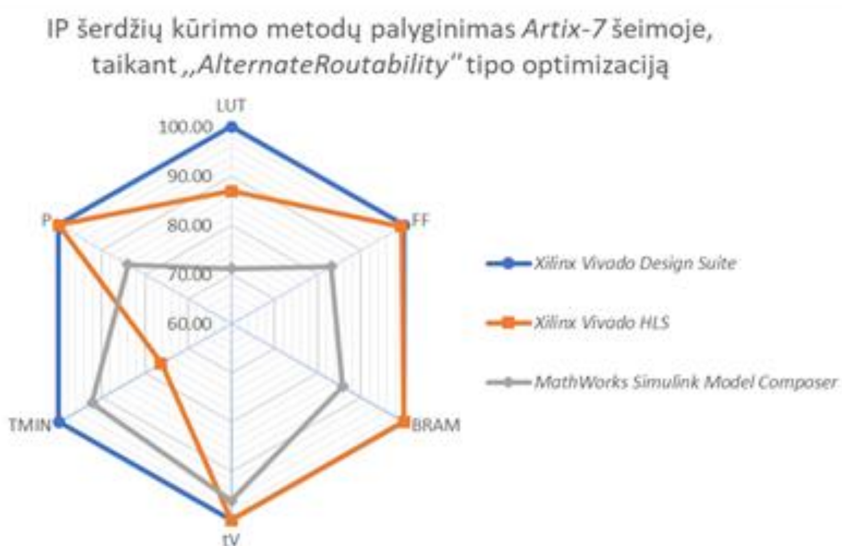
Gauti rezultatai normuojami šimto balų sistemoje taikant tyrimo metodikoje nurodytą formulę (4). Normuoti rezultatai pateikiami radaro tipo grafikuose: IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults"* tipo optimizaciją (žr. 31 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized"* tipo optimizaciją (žr. 32 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability"* tipo optimizaciją (žr. 33 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*PerformanceOptimized"* tipo optimizaciją (žr. 34 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*RuntimeOptimized"* tipo optimizaciją (žr. 35 pav.).



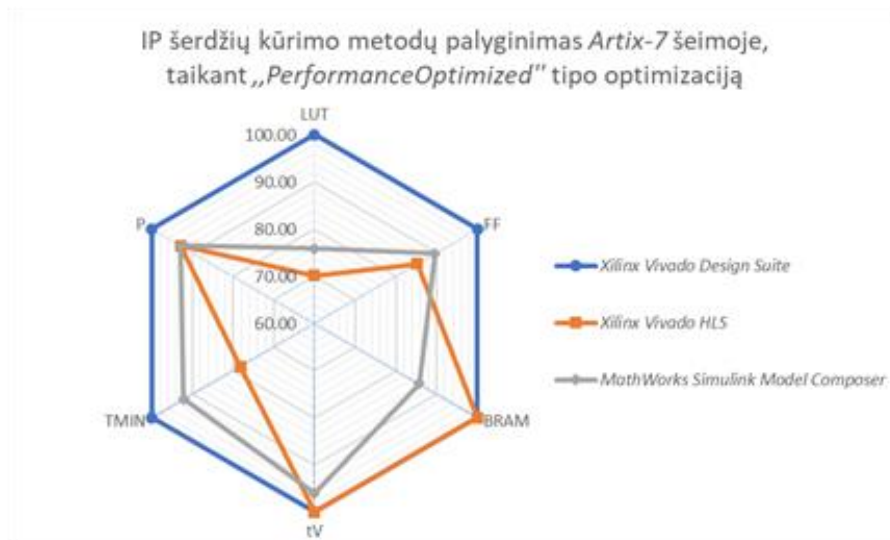
31 pav. IP šerdžių kūrimo metodų palyginimas taikant „*Defaults"* tipo optimizaciją



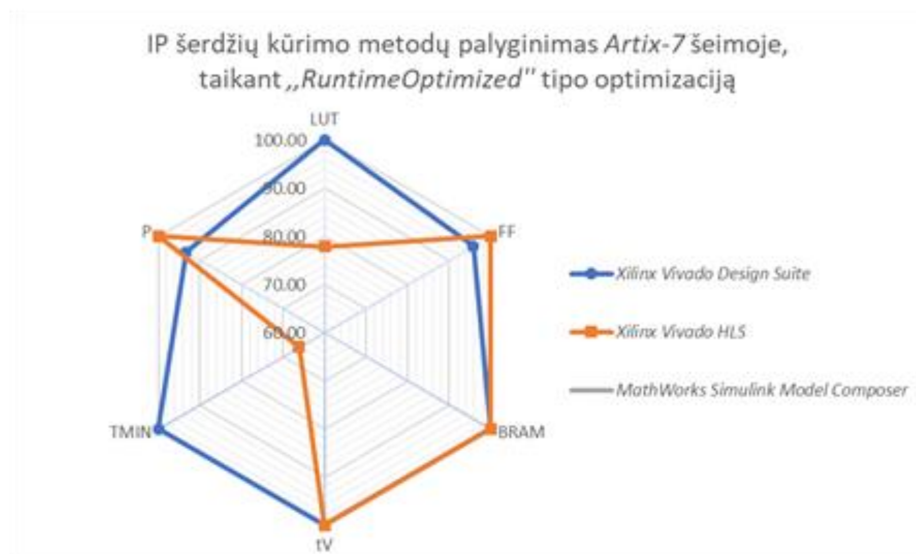
32 pav. IP šerdžių kūrimo metodų palyginimas taikant „*AreaOptimized"* tipo optimizaciją



33 pav. IP šerdžių kūrimo metodų palyginimas taikant „*AlternateRoutability"* tipo optimizaciją



34 pav. IP šerdžių kūrimo metodų palyginimas taikant „*PerformanceOptimized"* tipo optimizaciją



35 pav. IP šerdžių kūrimo metodų palyginimas taikant „*RuntimeOptimized"* tipo optimizaciją

Atlikus tyrimą *Artix-7* šeimoje pastebėta jog *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis, kaip ir *Spartan-7* šeimoje, yra efektyviausia. Keičiantis optimizacijos tipams stebima panaši rezultatų tendencija kaip ir *Spartan-7* šeimoje, tačiau *Artix-7* šeimoje matomas žymesnis skirtumas tarp *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotos IP šerdies ir kitais metodais realizuotų IP šerdžių minimalaus taktinio signalo periodo parametro. Optimizacijos tipui pasikeitus į „*RuntimeOptimized" MathWorks Simulink Model Composer* aplinkoje modeliais realizuota šerdis tampa neveiksni, o rezultatų pobūdis tarp likusių dviejų skirtingų IP šerdžių kūrimo metodų pasikeičia. *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotos šerdies energetiniai parametrai tampa geresni, taip pat pastebimas efektyvesnis *FF* architektūrinių elementų panaudojimas.

3.2.3. IP šerdžių tyrimas *Kintex-7* šeimoje

Tyrimas pakartojamas su *Kintex-7* programuojamos logikos matricos šeima, taikant skirtingas sintezės optimizacijos strategijas. Tyrimo metu gauti rezultatai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai (žr. 19 lentelę), *Xilinx Vivado HLS*

aplinkoje realizuotos IP šerdies parametrai (žr. 20 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai šeimoje (žr. 21 lentelę).

19 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	238	231	285	242	254
FF	329	329	367	442	409
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ciklai	308491	308491	308491	308491	308491
T_{MIN}, ns	3.02	3.01	3.02	3.02	3.03
P, mW	8	8	8	9	8

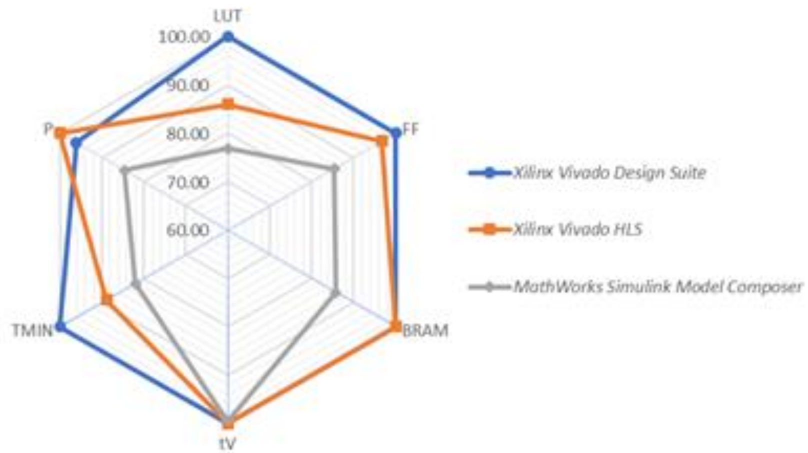
20 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	398	352	481	740	400
FF	368	368	368	695	368
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ciklai	308514	308514	308514	308514	308514
T_{MIN}, ns	4.44	4.72	4.44	4.53	4.43
P, mW	7	7	7	9	7

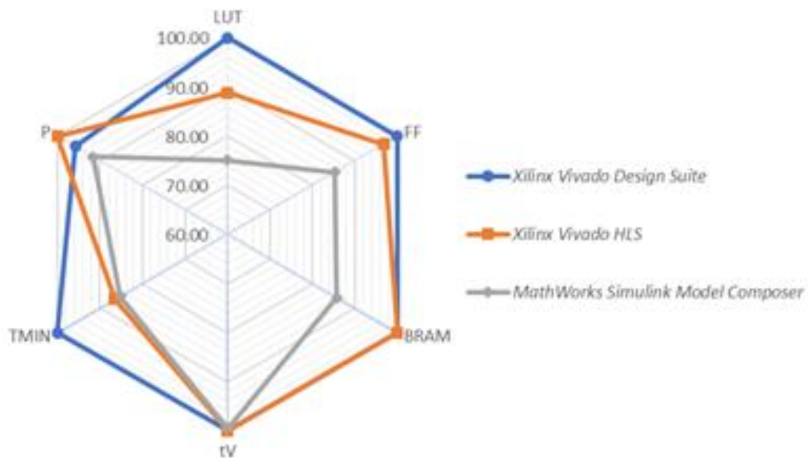
21 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	501	501	690	615	**-**
FF	506	506	506	540	**-**
BRAM	1.5	1.5	1.5	1.5	**-**
DSP	0	0	0	0	**-**
t_V, ciklai	313801	313801	313801	313801	**-**
T_{MIN}, ns	5.31	4.88	5.72	5.31	**-**
P, mW	11	9	12	12	**-**

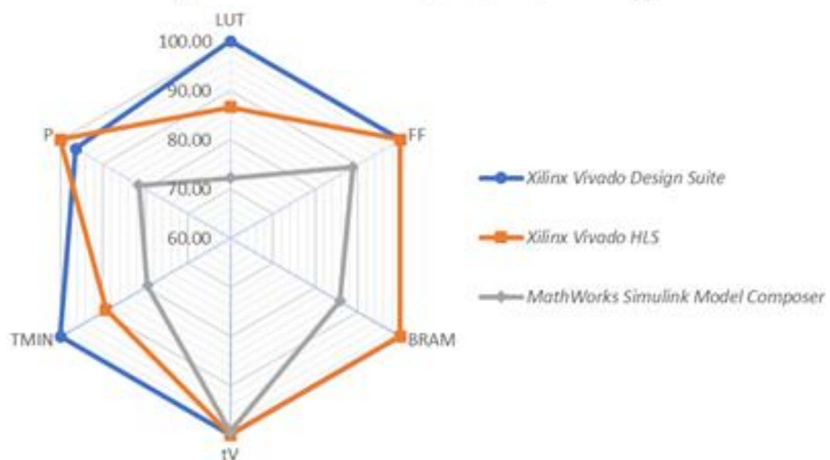
Gauti rezultatai normuojami šimto balų sistemoje taikant tyrimo metodikoje nurodytą formulę (4). Normuoti rezultatai pateikiami radaro tipo grafikuose: IP šerdžių kūrimo metodų palyginimas taikant *,,Defaults"* tipo optimizaciją (žr. 36 pav.), IP šerdžių kūrimo metodų palyginimas taikant *,,AreaOptimized"* tipo optimizaciją (žr. 37 pav.), IP šerdžių kūrimo metodų palyginimas taikant *,,AlternateRoutability"* tipo optimizaciją (žr. 38 pav.), IP šerdžių kūrimo metodų palyginimas taikant *,,PerformanceOptimized"* tipo optimizaciją (žr. 39 pav.), IP šerdžių kūrimo metodų palyginimas taikant *,,RuntimeOptimized"* tipo optimizaciją (žr. 40 pav.).



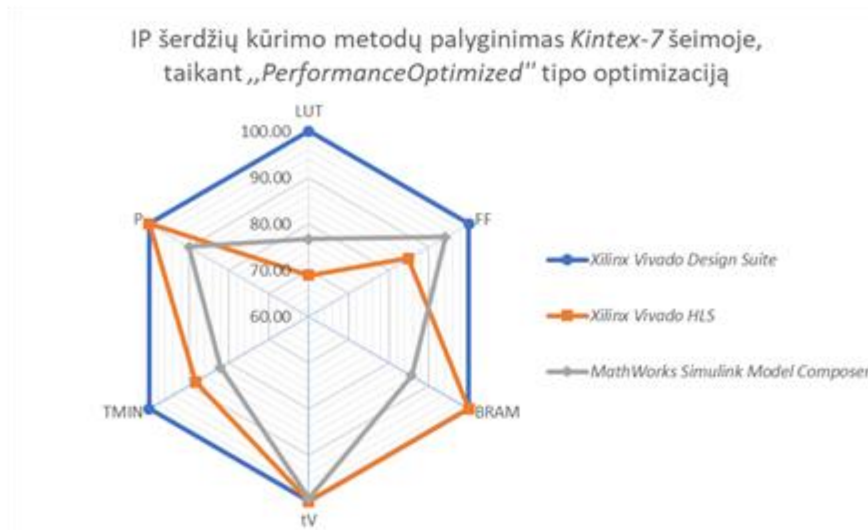
36 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults"* tipo optimizaciją



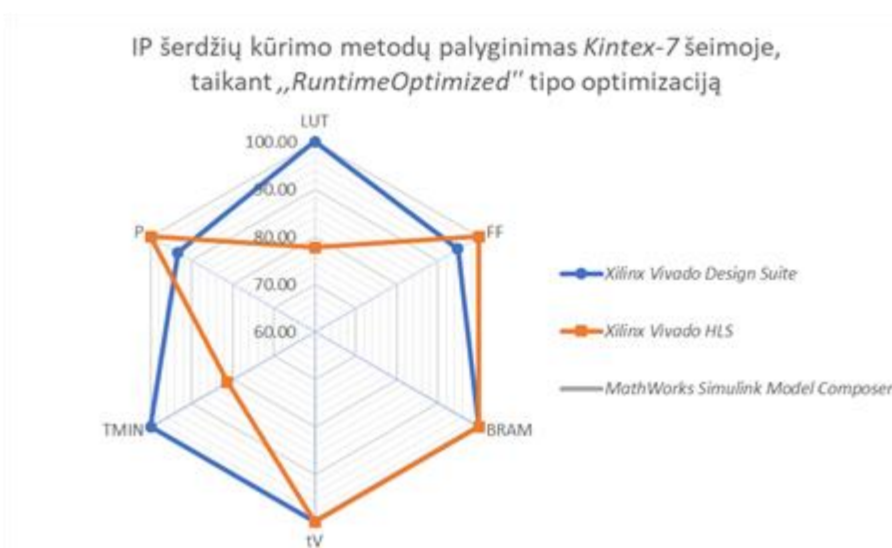
37 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized"* tipo optimizaciją



38 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability"* tipo optimizaciją



39 pav. IP šerdžių kūrimo metodų palyginimas taikant „*PerformanceOptimized"* tipo optimizaciją



40 pav. IP šerdžių kūrimo metodų palyginimas taikant „*RuntimeOptimized"* tipo optimizaciją

Atlikus tyrimą *Kintex-7* šeimoje pastebėta jog gauti rezultatai yra panašaus pobūdžio trijuose optimizacijų tipuose: *„Defaults", „AreaOptimized" ir „AlternateRoutability"*. Esant šio tipo optimizacijoms *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis yra efektyviausia penkių parametrų atžvilgiu, bet pasižymi prastesniais energetiniais parametrais lyginant su *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota šerdimi. Optimizacijos tipui pasikeitus į *„PerformanceOptimized"* pastebimas rezultatų pobūdžio pasikeitimas. *MathWorks Simulink Model Composer* aplinkoje modeliais realizuotos šerdies vidinių resursų panaudojimas tampa efektyvesnis lyginant su *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota šerdimi. Optimizacijos tipui pasikeitus į *„RuntimeOptimized" MathWorks Simulink Model Composer* aplinkoje modeliais realizuota šerdis tampa neveiksni, o rezultatų pobūdis tarp likusių dviejų skirtingų IP šerdžių kūrimo metodų tampa panašus į pirmų trijų aptartų optimizacijos tipų rezultatų pobūdį. *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis tampa efektyviausia keturių parametrų atžvilgiu, o *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotoje šerdyje pastebimas *FF* architektūrinių elementų panaudojimo efektyvumo padidėjimas.

3.2.4. IP šerdžių tyrimas *Zynq Ultrascale+* šeimoje

Tyrimas pakartojamas su *Zynq Ultrascale+* programuojamos logikos matricos šeima, taikant skirtingas sintezės optimizacijos strategijas. Tyrimo metu gauti rezultatai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai (žr. 22 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai (žr. 23 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai (žr. 24 lentelę).

22 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	270	258	285	286	253
FF	329	329	367	442	409
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ciklai	308491	308491	308491	308491	308491
T_{MIN}, ns	1.64	1.62	1.64	1.64	1.67
P, mW	5	5	4	5	5

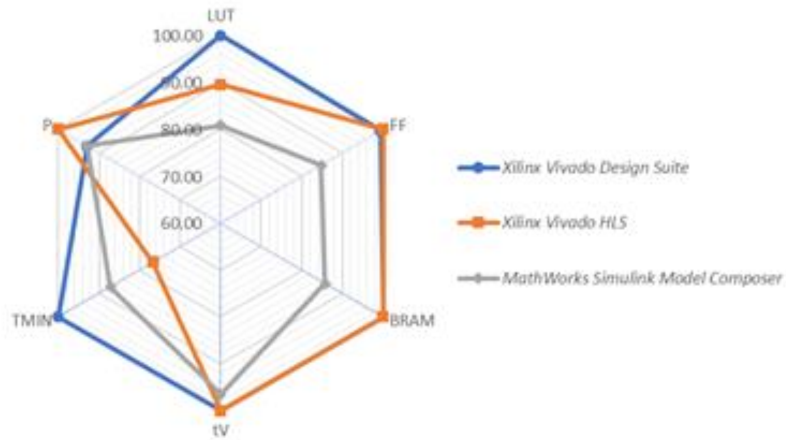
23 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies parametrai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	391	349	491	694	387
FF	321	321	321	624	321
BRAM	1	1	1	1	1
DSP	0	0	0	0	0
t_V, ciklai	308514	308514	308514	308514	308514
T_{MIN}, ns	3.44	3.42	3.30	4.08	3.35
P, mW	4	4	4	7	4

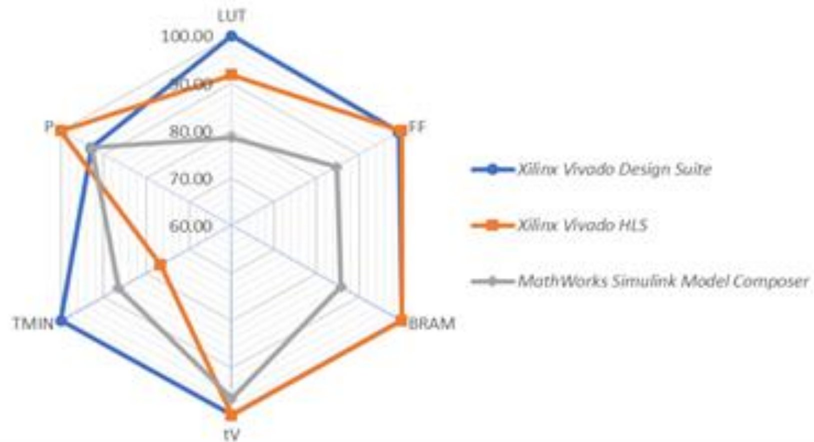
24 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies parametrai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
Vertinamas parametras	***Defaults***	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
LUT	491	495	699	651	577
FF	496	496	496	530	496
BRAM	1.5	1.5	1.5	1.5	1.5
DSP	0	0	0	0	0
t_V, ciklai	342605	342605	342605	342605	342605
T_{MIN}, ns	2.63	2.67	2.84	2.46	2.71
P, mW	5	5	8	5	7

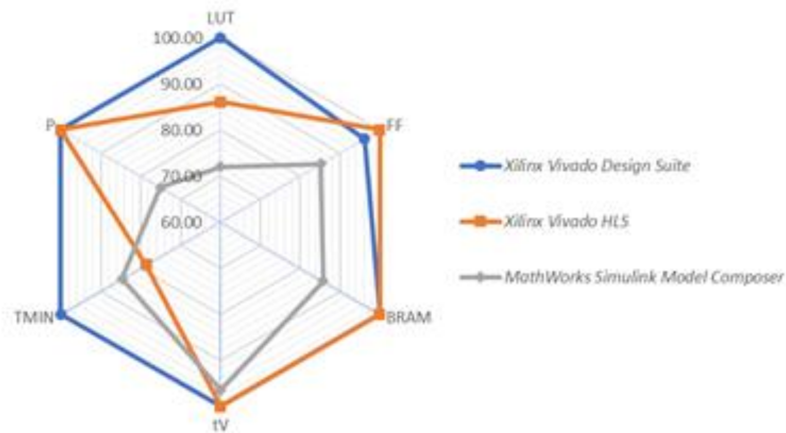
Gauti rezultatai normuojami šimto balų sistemoje taikant tyrimo metodikoje nurodytą formulę (4). Normuoti rezultatai pateikiami radaro tipo grafikuose: IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults*" tipo optimizaciją (žr. 41 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized*" tipo optimizaciją (žr. 42 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability*" tipo optimizaciją (žr. 43 pav.), IP šerdžių kūrimo metodų palyginimas taikant ,,*PerformanceOptimized*" tipo optimizaciją (žr. 44 pav.), IP šerdžių kūrimo metodų taikant ,,*RuntimeOptimized*" tipo optimizaciją (žr. 45 pav.).



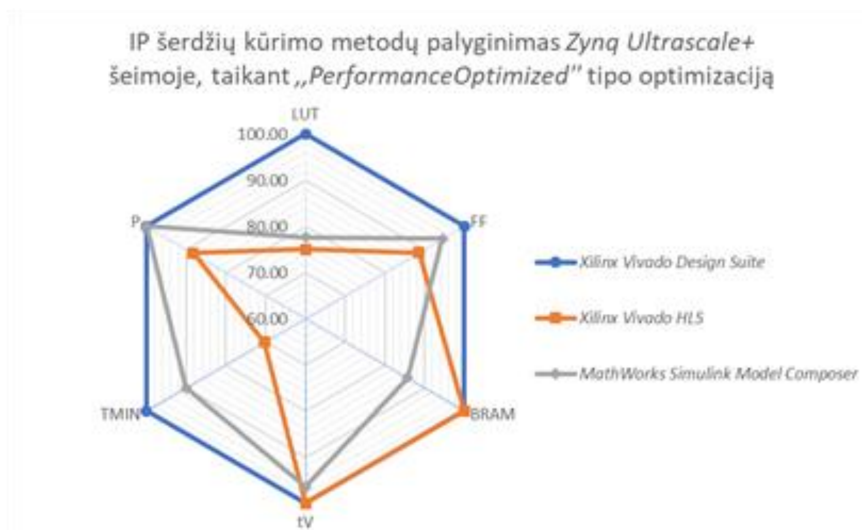
41 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*Defaults"* tipo optimizaciją



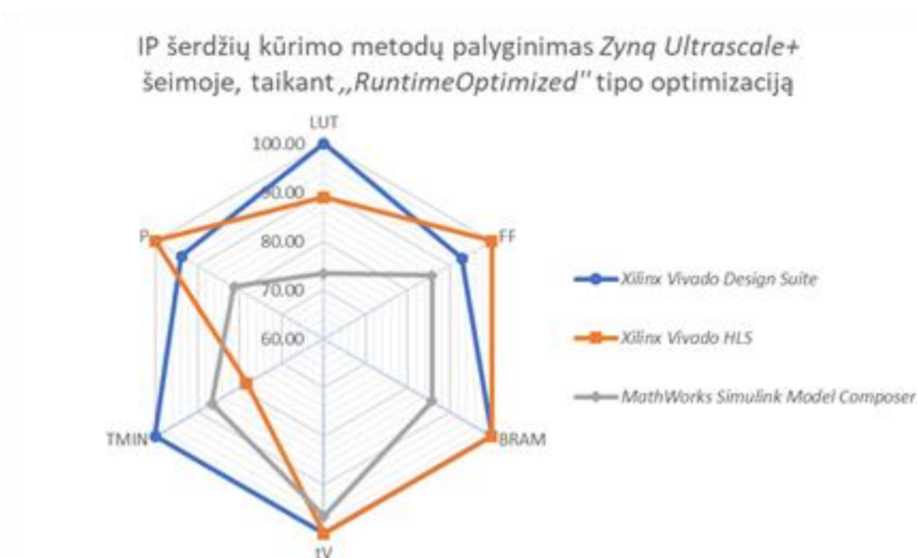
42 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AreaOptimized"* tipo optimizaciją



43 pav. IP šerdžių kūrimo metodų palyginimas taikant ,,*AlternateRoutability"* tipo optimizaciją



44 pav. IP šerdžių kūrimo metodų palyginimas taikant *„PerformanceOptimized"* tipo optimizaciją



45 pav. IP šerdžių kūrimo metodų palyginimas taikant *„RuntimeOptimized"* tipo optimizaciją

Atlikus tyrimą *Zynq Ultrascale+* šeimoje pastebėta jog gauti rezultatai yra panašaus pobūdžio keturiuose optimizacijų tipuose: *„Defaults", „AreaOptimized", „AlternateRoutability" ir „RuntimeOptimized".* Esant šio tipo optimizacijoms *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis yra efektyviausia penkių parametrų atžvilgiu, bet pasižymi prastesniais energetiniais parametrais bei prastesniu *FF* architektūrinių elementų panaudojimu esant *„RuntimeOptimized"* optimizacijos tipui lyginant su *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota šerdimi. Taip pat pastebėta jog *MathWorks Simulink Model Composer* aplinkoje modeliais realizuota IP šerdis pasižymi geresniu minimalaus taktinio signalo periodo parametru lyginant su *Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota šerdimi. Optimizacijos tipui pasikeitus į *„PerformanceOptimized" Xilinx Vivado HLS* aplinkoje *C++* kalba realizuotos šerdies vidinių resursų panaudojimo efektyvumas sumažėja, o energetiniai parametrai suprastėja, šių parametrų atžvilgiu *MathWorks Simulink Model Composer* aplinkoje modeliais realizuota IP šerdis tampa efektyvesnė, tačiau efektyviausia šerdimi išlieka *Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis.

3.2.5. IP šerdžių kūrimo metodų palyginimas *Spartan-7* šeimoje

Kiekvienam IP šerdies kūrimo metodui apskaičiuojamas efektyvumo balas pagal tyrimo metodikoje nurodytą reitingavimo metodo formulę (5). Kadangi reitingavimo metodas yra subjektyvus vertinamiems parametrams priskiriami svertiniai koeficientai, kurių vertės pasirenkamos priklausomai nuo sistemos dizaino reikalavimų, laisvų vidinių architektūrinių elementų ir kitų kriterijų. Šiame darbe didžiausias dėmesys skiriamas panaudojamiems vidiniams programuojamos logikos matricos resursas, todėl bendras svertinis koeficientas, skiriamas šiai grupei, yra aukščiausias – 0.7. Priklausomai nuo laisvų resursų kiekio svertinis koeficientas padalinamas atskiriems architektūriniams elementams: w_{LUT} – 0.15, w_{FF} – 0.2, w_{BRAM} – 0.35. Laikinių parametrų grupei skiriamas svertinis koeficientas – 0.25 (w_{tv} – 0.1, w_{TMIN} – 0.15). Žemiausias svertinis koeficientas skiriamas energetiniams parametras w_P – 0.05. Svertiniai koeficientai pateikiami lentelės formoje (žr. 25 lentelę).

25 lentelė. Tyrime naudojamų svertinių koeficientų vertės

Svertinis koeficientas w_i	**Skaitinė išraiška**
w_{LUT}	0.15
w_{FF}	0.20
w_{BRAM}	0.35
w_{tV}	0.10
w_{TMIN}	0.15
w_P	0.05
Suma	1.00

Tyrimo metu gauti *Spartan-7* šeimos efektyvumo balai ir jų statistiniai parametrai, apskaičiuoti naudojantis formulėmis (6), (7), (8), pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 26 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 27 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 28 lentelę).

26 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	100.00	100.00	100.00	100.00	100.00
Arit. vidurkis $\overline{Ef}$	100.00				
Mediana $\widetilde{Ef}$	100.00				
Stan. nuokrypis S_{Ef}	0.00				

27 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	95.07	94.90	95.94	90.73	94.65
Arit. vidurkis $\overline{Ef}$	94.26				
Mediana $\widetilde{Ef}$	94.90				
Stan. nuokrypis S_{Ef}	2.03				

28 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP efektyvumo balai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	86.11	85.69	85.92	88.19	-
Arit. vidurkis $\overline{Ef}$	86.48				
Mediana $\widetilde{Ef}$	86.01				
Stan. nuokrypis S_{Ef}	1.15				

Apskaičiavus efektyvumo balų statistinius parametrus pastebėta jog gauti standartiniai nuokrypiai nėra dideli, o efektyvumo balas *Spartan-7* šeimoje išlieka panašus nepriklausomai nuo optimizacijos rūšies esant tam pačiam IP šerdžių kūrimo metodui. Suskaičiavus efektyvumo balo vidurkį galima teigti jog esant konkretiems svertiniams koeficientas efektyviausias IP šerdžių kūrimo metodas yra *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (100.00 balų). Antras pagal efektyvumą IP šerdžių kūrimo metodas yra *C++* kalba *Xilinx Vivado HLS* aplinkoje (94.26 balų), o mažiausiai efektyvus IP šerdžių kūrimo metodas iš anksto paruoštais modeliais *MathWorks Simulink Model Composer* aplinkoje (86.48 balų).

3.2.6. IP šerdžių kūrimo metodų palyginimas *Artix-7* šeimoje

Tyrimo metu gauti *Artix-7* šeimos efektyvumo balai ir jų statistiniai parametrai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 29 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 30 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 31 lentelę).

29 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	100.00	100.00	100.00	100.00	98.83
Arit. vidurkis $\overline{Ef}$	99.77				
Mediana $\widetilde{Ef}$	100.00				
Stan. nuokrypis S_{Ef}	0.53				

30 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	93.61	94.35	94.33	88.95	91.59
Arit. vidurkis $\overline{Ef}$	92.57				
Mediana $\widetilde{Ef}$	93.61				
Stan. nuokrypis S_{Ef}	2.31				

31 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	85.19	84.56	84.93	87.37	-
Arit. vidurkis $\overline{Ef}$	85.51				
Mediana $\widetilde{Ef}$	85.06				
Stan. nuokrypis S_{Ef}	1.26				

Apskaičiavus efektyvumo balus bei jų statistinius parametrus pastebėta jog *Artix-7* šeimoje, kaip ir *Spartan-7* šeimoje, nepriklausomai nuo optimizacijos rūšies esant tam pačiam IP šerdžių kūrimo metodui efektyvumo balas išlieka panašus, o standartiniai efektyvumo balo nuokrypiai nėra dideli. Taip pat pastebėta jog suskaičiavus efektyvumo balo vidurkį gautos rezultatų skaitinės reikšmės yra panašios *Spartan-7* šeimoje gautoms skaitinės reikšmėms, todėl pagal efektyvumą IP šerdžių kūrimo metodai išsirikiuoja tuo pačiu eiliškumu: *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (99.77 balų), *C++* kalba *Xilinx Vivado HLS* aplinkoje (92.57 balų), *MathWorks Simulink Model Composer* aplinkoje (85.51 balų).

3.2.7. IP šerdžių kūrimo metodų palyginimas *Kintex-7* šeimoje

Tyrimo metu gauti *Kintex-7* šeimos efektyvumo balai ir jų statistiniai parametrai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 32 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 33 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 34 lentelę).

32 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	99.81	99.79	99.81	100.00	98.61
Arit. vidurkis $\overline{Ef}$	99.61				
Mediana $\widetilde{Ef}$	99.81				
Stan. nuokrypis S_{Ef}	0.56				

33 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	95.57	95.65	96.35	90.55	93.84
Arit. vidurkis $\overline{Ef}$	94.39				
Mediana $\widetilde{Ef}$	95.57				
Stan. nuokrypis S_{Ef}	2.34				

34 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai

***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	85.07	85.63	84.53	87.10	-
Arit. vidurkis $\overline{Ef}$	85.58				
Mediana $\widetilde{Ef}$	85.35				
Stan. nuokrypis S_{Ef}	1.11				

Apskaičiavus efektyvumo balus bei jų statistinius parametrus pastebėta jog *Kintex-7* šeimoje, kaip ir prieš tai aptartose šeimose, nepriklausomai nuo optimizacijos rūšies esant tam pačiam IP šerdžių kūrimo metodui efektyvumo balas išlieka panašus. Apskaičiavus efektyvumo balo vidurkį bei kitus statistinius parametrus pastebėta jog gautos rezultatų skaitinės reikšmės yra panašios prieš tai aptartose šeimose gautoms skaitinės reikšmėms. Pagal efektyvumą IP šerdžių kūrimo metodai išsirikiuoja tuo pačiu eiliškumu: *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (99.61 balų), *C++*

kalba *Xilinx Vivado HLS* aplinkoje (94.39 balų), *MathWorks Simulink Model Composer* aplinkoje (85.58 balų).

3.2.8. IP šerdžių kūrimo metodų palyginimas *Zynq Ultrascale+* šeimoje

Tyrimo metu gauti *Zynq Ultrascale+* šeimos efektyvumo balai ir jų statistiniai parametrai pateikiami lentelių formoje: *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 35 lentelę), *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 36 lentelę), *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai (žr. 37 lentelę).

35 lentelė. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado Design Suite* aplinkoje *VHDL* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	99.50	99.50	99.22	100.00	98.25
Arit. vidurkis $\overline{Ef}$	99.30				
Mediana $\widetilde{Ef}$	99.50				
Stan. nuokrypis S_{Ef}	0.64				

36 lentelė. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies efektyvumo balai

***Xilinx Vivado HLS* aplinkoje *C++* kalba realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	94.92	95.26	94.71	88.91	95.09
Arit. vidurkis $\overline{Ef}$	93.78				
Mediana $\widetilde{Ef}$	94.92				
Stan. nuokrypis S_{Ef}	2.73				

37 lentelė. *MathWorks Simulink Model Composer* aplinkoje realizuotos IP šerdies efektyvumo balai

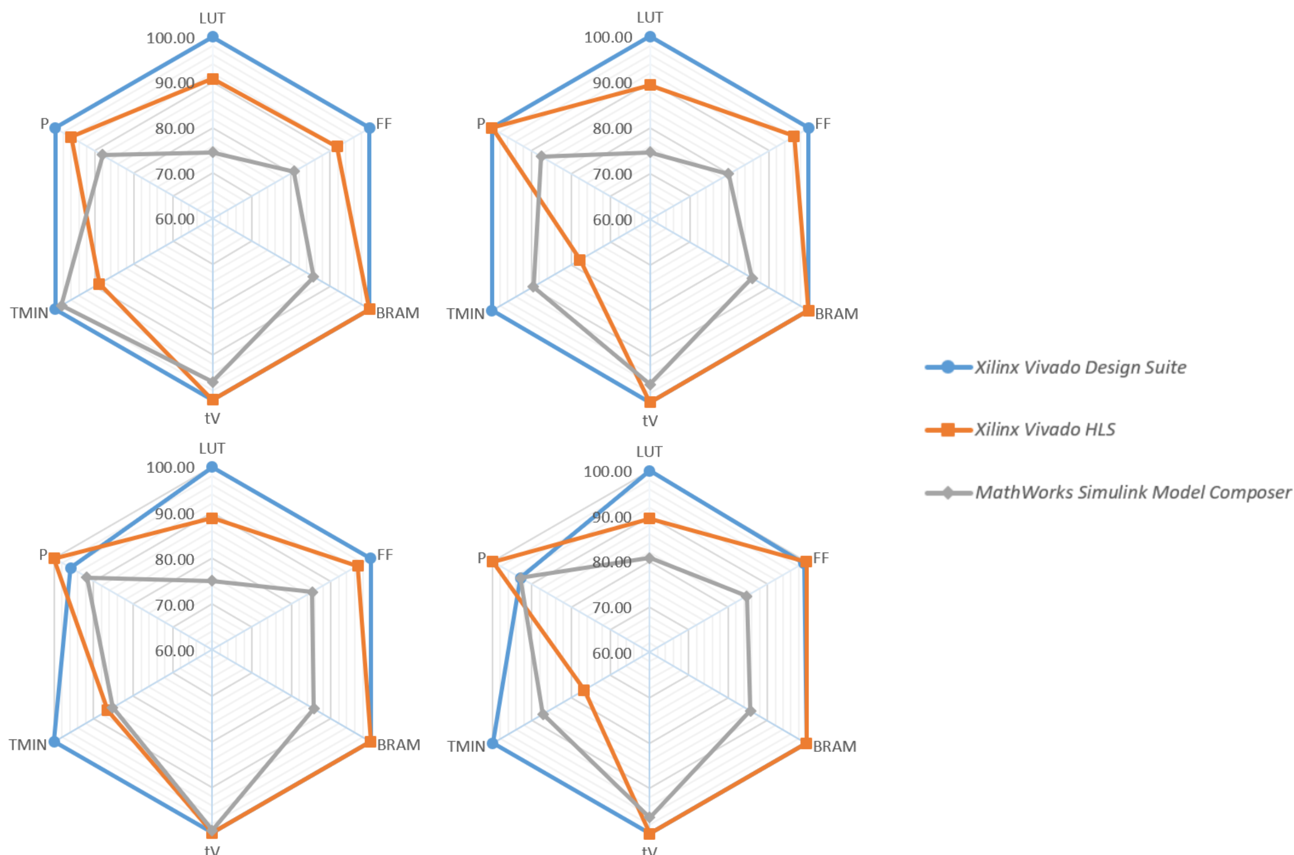
***MathWorks Simulink Model Composer* aplinkoje realizuota IP šerdis**					
Optimizacijos rūšis					
	Defaults	***AreaOptimized***	***AlternateRoutability***	***PerformanceOptimized***	***RuntimeOptimized***
Efektyvumo balas Ef	86.43	85.98	83.92	88.69	84.85
Arit. vidurkis $\overline{Ef}$	85.97				
Mediana $\widetilde{Ef}$	85.98				
Stan. nuokrypis S_{Ef}	1.80				

Apskaičiavus efektyvumo balus bei jų statistinius parametrus pastebėta jog *Zynq Ultrascale+* šeimoje, kaip ir prieš tai aptartose šeimose, nepriklausomai nuo optimizacijos rūšies esant tam pačiam IP šerdžių kūrimo metodui efektyvumo balas išlieka panašus, o gauti standartiniai nuokrypiai nėra dideli. Taip pat pastebėta jog apskaičiavus efektyvumo balo statistinius parametrus gautos rezultatų skaitinės reikšmės yra panašios prieš tai aptartose šeimose gautoms skaitinės reikšmėms. IP šerdžių kūrimo metodų eiliškumas pagal efektyvumą nepasikeičia ir išlieka toks pats: *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (99.30 balų), *C++* kalba *Xilinx Vivado HLS* aplinkoje (93.78 balų), *MathWorks Simulink Model Composer* aplinkoje (85.97 balų).

3.2.9. Apibendrintas IP šerdžių kūrimo metodų palyginimas

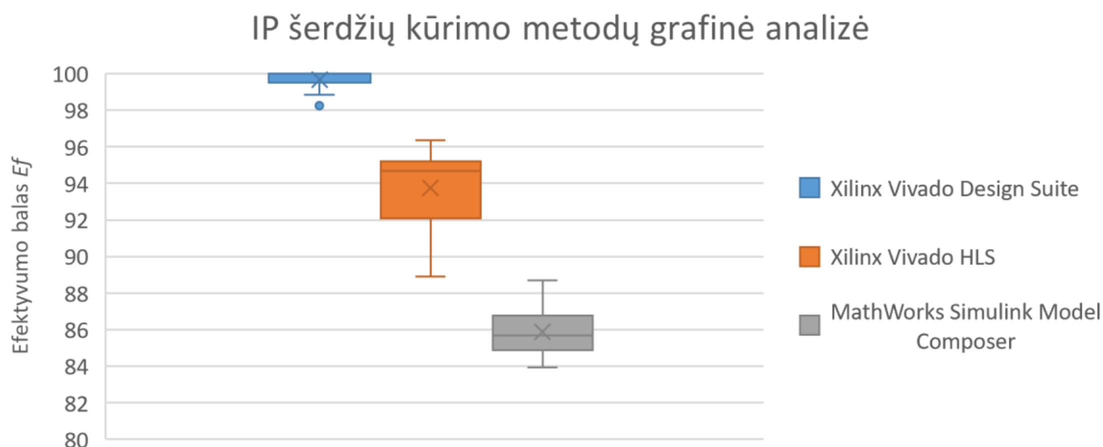
Remiantis tyrimo metu gautais rezultatais galima teigti jog IP šerdžių kūrimo metodų efektyvumas nepriklauso nuo pasirenkamos optimizacijos rūšies ar programuojamos logikos matricos šeimos.

Kaip pavyzdys pateikiamas grafinis, skirtingose šeimose (viršuje kairėje – *Spartan-7*, dešinėje *Artix-7*, apačioje kairėje *Kintex-7*, dešinėje *Zynq Ultrascale+*) realizuotų IP šerdžių (taikant „*Defaults"* tipo optimizaciją), palyginimas (žr. 46 pav.).



46 pav. IP šerdžių kūrimo metodų grafinis palyginimas

Taip pat atliekama grafinė trijų grupių (skirtingų IP šerdžių kūrimo metodų) analizė, kurioje analizuojami visų IP šerdžių, nepriklausomai nuo pasirenkamos optimizacijos rūšies ar programuojamos logikos matricos šeimos, duomenys (žr. 47 pav.). Grafinėje analizėje atvaizduojamos imčių maksimalios, minimalios vertės, medianos (pažymėta x) bei vidurkiai (pažymėta –) [37].



47 pav. IP šerdžių kūrimo metodų grafinė analizė

Galima pastebėti jog kiekvieno IP šerdies kūrimo metodo atveju efektyvumo balo mediana yra artima vidurkiui, o balo kitimo ribos yra mažos bei tarpusavyje nepersidengiančios su kitų metodų duomenimis. Šie požymiai leidžia teigti jog pasirenkant programuojamos logikos matricos šeimą ar optimizacijos tipą apskaičiuojamas IP šerdies efektyvumo balas nėra paveikiamas šių pasirinkimų ir priklauso tik nuo naudojamo šerdies kūrimo metodo bei pačios šerdies realizacijos. Pateikiami apskaičiuoti IP šerdžių kūrimo metodų efektyvumo balų statistiniai parametrai, nepriklausomai nuo programuojamos logikos matricos šeimos ar optimizacijos tipo (žr. 38 lentelę).

38 lentelė. IP šerdžių kūrimo metodų efektyvumo balų statistiniai parametrai

	Efektyvumo balo *Ef* statistiniai parametrai		
IP šerdžių kūrimo metodas	***Arit. vidurkis*** $\overline{Ef}$	***Mediana*** $\widetilde{Ef}$	***Stan. nuokrypis*** S_{Ef}
Xilinx Vivado Design Suite aplinkoje *VHDL* kalba	99.67	100.00	0.53
Xilinx Vivado HLS aplinkoje *C++* kalba	93.75	94.68	2.29
MathWorks Simulink Model Composer aplinkoje	85.89	85.69	1.32

Remiantis tyrimo metu gautais efektyvumo balo vidurkiais bei kitais statistiniais parametrais galima teigti jog esant konkrečioms svertinių koeficientų reikšmėms, kuomet didžiausias dėmesys yra skiriamas panaudojamų architektūrinių elementų skaičiui, efektyviausias IP šerdžių kūrimo metodas yra *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (99.66 balų). Antras pagal efektyvumą IP šerdžių kūrimo metodas yra *C++* kalba *Xilinx Vivado HLS* aplinkoje (93.74 balų), o mažiausiai efektyvus IP šerdžių kūrimo metodas yra funkciniais blokeliais *MathWorks Simulink Model Composer* aplinkoje (85.89 balų).

Išvados

1. Atliekant tiriamąjį darbą išanalizuoti *Xilinx* programuojamos logikos matricų gamintojo siūlomi IP šerdžių kūrimo metodai: kodą rašant *HDL* kalbomis *Xilinx Vivado Design Suite* aplinkoje, kodą rašant *C/C++* kalbomis *Xilinx Vivado HLS* aplinkoje, apjungiant funkcinius blokelius *MathWorks Simulink Model Composer* aplinkoje. Naudojantis šiais trimis metodais realizuotos IP šerdys, skirtos spręsti vieną iš pagrindinių vaizdų apdorojimo užduočių – objektų kontūrų radimą.
2. IP šerdžių vertinimui ir palyginimui sukurta universali metodika, kuri nėra priklausoma nuo programuojamos logikos matricos gamintojo ar pasirinktos programuojamos logikos matricos šeimos. Ši metodika paremta duomenų normalizavimu ir efektyvumo balo apskaičiavimu. Skaičiuojant efektyvumo balą naudojami svertiniai koeficientai, kurių vertės pasirenkamos priklausomai nuo realizuojamai sistemai keliamų reikalavimų. Apskaičiuotas rezultatas pateikiamas kaip konkreti skaitinė išraiška, leidžianti lengviau palyginti ir pasirinkti IP šerdžių kūrimo metodą.
3. Remiantis tyrimo metu gautais efektyvumo balai ir efektyvumo balų statistiniais parametrais nustatyta jog esant konkrečioms svertinių koeficientų reikšmėms IP šerdžių kūrimo metodų efektyvumas nepriklauso nuo pasirenkamos programuojamos logikos matricos šeimos ar naudojamo optimizacijos tipo. Tokia išvada priimta šerdis įvertinus reitingavimo metodu ir pastebėjus jog kiekvieno IP šerdies kūrimo metodo atveju gautų efektyvumo balų vidurkiai yra panašus bei artimi medianoms, standartinis nuokrypiai yra maži, o balų kitimo diapazonas nėra platus.
4. Tyrimo metu objektų kontūrų radimui skirta IP šerdis *Xilinx Vivado Design Suite* aplinkoje realizuota per 96 valandas, tuo tarpu *Xilinx Vivado HLS* aplinkoje šerdžiai realizuoti užtrukta 23 valandas. Mažiausiai laiko IP šerdžiai realizuoti užtrukta *MathWorks Simulink Model Composer* aplinkoje – 2 valandos. *Xilinx* gamintojo siūlomi alternatyvus šerdžių kūrimo metodai (*Xilinx Vivado HLS* bei *MathWorks Simulink Model Composer* aplinkose) leidžia realizuoti norimo funkcionalumo IP šerdis sparčiau nei įprastinis šerdžių kūrimo metodas (*Xilinx Vivado Design Suite* aplinkoje). Tačiau skirtingais metodais realizuotų IP šerdžių efektyvumas nėra vienodas. Apskaičiavus bendrą gautų efektyvumo balų vidurkį, nepriklausomai nuo pasirinktos programuojamos logikos matricos šeimos ar naudojamo optimizacijos tipo, nustatyta jog esant konkrečioms svertinių koeficientų reikšmėms efektyviausias šerdžių kūrimo metodas yra *VHDL* kalba *Xilinx Vivado Design Suite* aplinkoje (99.66 balų). Antras pagal efektyvumą IP šerdžių kūrimo metodas yra *C++* kalba *Xilinx Vivado HLS* aplinkoje (93.74 balų), o mažiausiai efektyviu IP šerdžių kūrimo metodu laikomas programavimas funkciniais blokeliais *MathWorks Simulink Model Composer* aplinkoje (85.89 balų).
5. Siekiant palengvinti programuojamos logikos matricos programuotojų apsisprendimus, renkantis IP šerdžių kūrimo metodą, reikėtų sukurti duomenų bazę, kurioje būtų saugomi analizuotais kūrimo metodais realizuotų skirtingų funkcionalumų IP šerdžių normuoti duomenys. Remiantis šiais duomenimis ir naudojantis reitingavimo metodu programuotojas galėtų pateiktiems parametras priskirti svertinius koeficientus, apskaičiuoti skirtingais metodais realizuotų šerdžių efektyvumo balus, juos palyginti ir pasirinkti efektyviausią IP šerdies kūrimo metodą iškilusiai užduočiai spręsti.

Literatūros sąrašas

1. [interaktyvus] Field Programmable Gate Array (FPGA) [žiūrėta 2018 spalio 24 d.]. Prieiga per internetą: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html
2. [interaktyvus] Xilinx – Investigating in FPGAs for AI Hardware [žiūrėta 2018 spalio 24 d.]. Prieiga per internetą: https://www.nanalyze.com/2017/05/xilinx-investing-fpgas-ai-hardware/
3. [interaktyvus] Xilinx Vivado Design Suite [žiūrėta 2018 lapkričio 2 d.]. Prieiga per internetą: https://www.xilinx.com/products/design-tools/vivado.html
4. [interaktyvus] Intel FPGA Development Tools [žiūrėta 2018 lapkričio 10 d.]. Prieiga per internetą: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html
5. [interaktyvus] AXI Reference Guide [žiūrėta 2018 lapkričio 10 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
6. [interaktyvus] Accelerating Integration [žiūrėta 2018 lapkričio 10 d.]. Prieiga per internetą: https://www.xilinx.com/products/design-tools/vivado/integration.
7. [interaktyvus] Xilinx Vivado Design User Guide Creating and Packaging Custom IP [žiūrėta 2018 lapkričio 17 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1118-vivado-creating-packaging-custom-ip.pdf
8. L. H. Crockett, R. A. Elliot,M. A. Enderwitz, R.W. Stewart, The Zynq Book, Strathclyde Academic Media, 2014
9. [interaktyvus] Xilinx Vivado Design Suite High-Level Synthesis [žiūrėta 2018 lapkričio 24 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf
10. [interaktyvus] Xilinx Design Suite User Guide Model-Based DSP Design Using System Generator [žiūrėta 2018 lapkričio 24 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug897-vivado-sysgen-user.pdf
11. D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons and F. Morgan, "Xilinx Vivado High Level Synthesis: Case studies," 25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014), Limerick, 2014, pp. 352-356.
12. H. Li and W. Ye, "Efficient implementation of FPGA based on Vivado High Level Synthesis," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2016, pp. 2810-2813.
13. F. M. Sánchez, R. Mateos, E. J. Bueno, J. Mingo and I. Sanz, "Comparative of HLS and HDL implementations of a grid synchronization algorithm," IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, 2013, pp. 2232-2237.
14. F. Winterstein, S. Bayliss and G. A. Constantinides, "High-level synthesis of dynamic data structures: A case study using Vivado HLS," 2013 International Conference on Field-Programmable Technology (FPT), Kyoto, 2013, pp. 362-365.

15. A. Stanciu and C. Gerigan, "Comparison between implementations efficiency of HLS and HDL using operations over Galois Fields," 2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME), Constanta, 2017, pp. 171-174, doi: 10.1109/SIITME.2017.8259883.
16. A. Cortes, I. Velez and A. Irizar, "High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies," 2016 Conference on Design of Circuits and Integrated Systems (DCIS), Granada, 2016, pp. 1-6, doi: 10.1109/DCIS.2016.7845376.
17. E. Kalali and I. Hamzaoglu, "FPGA implementations of HEVC Inverse DCT using high-level synthesis," 2015 Conference on Design and Architectures for Signal and Image Processing (DASIP), Krakow, 2015, pp. 1-6, doi: 10.1109/DASIP.2015.7367262.
18. V. V. Krishnapriya, M. Sikha, R. Nandakumar and J. U. Kidav, "Hardware efficiency comparison of IP Cores for CAN & LIN protocols," 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12), Coimbatore, 2012, pp. 1-3, doi: 10.1109/ICCCNT.2012.6395930.
19. D. d. Andres, J. Ruiz and P. Gil, "Using Dependability, Performance, Area and Energy Consumption Experimental Measures to Benchmark IP Cores," 2009 Fourth Latin-American Symposium on Dependable Computing, Joao Pessoa, 2009, pp. 49-56, doi: 10.1109/LADC.2009.17.
20. D. L. N. Hettiarachchi, V. S. P. Davuluru and E. J. Balster, "Integer vs. Floating-Point Processing on Modern FPGA Technology," 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2020, pp. 0606-0612, doi: 10.1109/CCWC47524.2020.9031118.
21. [interaktyvus] 7 Series FPGAs Configurable Logic Block [žiūrėta 2019 sausio 12 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
22. [interaktyvus] 7 Series FPGAs Memory Resources [žiūrėta 2019 sausio 12 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
23. [interaktyvus] 7 Series DSP48E1 Slice [žiūrėta 2019 sausio 12 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
24. [interaktyvus] Vivado Design Suite User Guide Synthesis [žiūrėta 2019 sausio 21 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug901-vivado-synthesis.pdf
25. [interaktyvus] Vivado Timing [žiūrėta 2019 vasario 3 d.]. Prieiga per internetą: https://www.xilinx.com/support/answers/57304.html
26. [interaktyvus] Vivado Design Suite User Guide Power Analysis and Optimization [žiūrėta 2019 vasario 4 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug907-vivado-power-analysis-optimization.pdf
27. [interaktyvus] Relative Scoring vs Absolute Scoring Explained [žiūrėta 2019 vasario 12 d.]. Prieiga per internetą: http://help.leaderboarded.com/knowledgebase/articles/613299-relative-scoring-vs-absolute-scoring-explained

28. K. Zhu, X. Liu and P. W. T. Pong, "Performance Study on Commercial Magnetic Sensors for Measuring Current of Unmanned Aerial Vehicles," in IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 4, pp. 1397-1407, April 2020, doi: 10.1109/TIM.2019.2910339.
29. [interaktyvus] Measures of Central Tendency [žiūrėta 2019 gegužės 1 d.]. Prieiga per internetą: https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php
30. [interaktyvus] Standard Deviation [žiūrėta 2019 gegužės 1 d.]. Prieiga per internetą: https://statistics.laerd.com/statistical-guides/measures-of-spread-standard-deviation.php
31. A. Amaricai, C. Gavriliu and O. Boncalo, "An FPGA sliding window-based architecture harris corner detector," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, 2014, pp. 1-4, doi: 10.1109/FPL.2014.6927402.
32. [interaktyvus] Gradient Filter implementation on FPGA [žiūrėta 2019 gegužės 5 d.]. Prieiga per internetą: https://www.element14.com/community/groups/fpga-group/blog/2015/05/27/gradient-filter-implementation-on-fpga-part2-first-modules
33. [interaktyvus] Vivado HLS Optimization Methodology Guide [žiūrėta 2019 birželio 24 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1270-vivado-hls-opt-methodology-guide.pdf
34. [interaktyvus] Model Composer User Guide [žiūrėta 2019 rugsėjo 7 d.]. Prieiga per internetą: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug1262-model-composer-user-guide.pdf
35. [interaktyvus] About GIMP [žiūrėta 2019 gegužės 16 d.]. Prieiga per internetą: https://www.gimp.org/about/
36. [interaktyvus] KTU Elektronikos rūmai [žiūrėta 2019 rugsėjo 24 d.]. Prieiga per internetą: https://renginiai.kasvyksta.lt/688/kaunas/ktu-elektronikos-rumai#!nuotrauka2
37. [interaktyvus] Box Plot Review [žiūrėta 2020 balandžio 30 d.]. Prieiga per internetą: https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/box-plot-review

Priedai

1 priedas. *Xilinx Vivado Design Suite* aplinkoje realizuotos IP šerdies kodas

edge_detection_hdl_v1_0.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity edge_detection_hdl_v1_0 is
    port (
   -- Users to add ports here

   -- User ports ends
   -- Do not modify the ports beyond this line


   -- Ports of Axi Slave Bus Interface S00_AXIS
   axis_aclk         : in std_logic;
   axis_aresetn      : in std_logic;
   s00_axis_tready   : out std_logic;
   s00_axis_tdata : in std_logic_vector(7 downto 0);
   s00_axis_tstrb : in std_logic_vector(0 downto 0);
   s00_axis_tlast : in std_logic;
         s00_axis_tuser : in std_logic_vector(0 downto 0);
   s00_axis_tvalid   : in std_logic;
   -- Ports of Axi Master Bus Interface M00_AXIS
   m00_axis_tvalid   : out std_logic;
   m00_axis_tdata : out std_logic_vector(7 downto 0);
   m00_axis_tstrb : out std_logic_vector(0 downto 0);
   m00_axis_tlast : out std_logic;
         m00_axis_tuser : out std_logic_vector(0 downto 0);
   m00_axis_tready   : in std_logic
   );
end edge_detection_hdl_v1_0;

architecture arch_imp of edge_detection_hdl_v1_0 is

   -- component declaration
    component edlu is
      Port (
        s_axis_aclk      : in  std_logic;
        s_axis_aresetn  : in  std_logic;
        S_AXIS_TREADY   : out std_logic;
        S_AXIS_TDATA : in  std_logic_vector(7 downto 0);
        S_AXIS_TVALID   : in  std_logic;
        S_AXIS_TLAST : in  std_logic;
        S_AXIS_TUSER : in  std_logic;
        M_AXIS_TREADY   : in  std_logic;
        M_AXIS_TDATA : out std_logic_vector(7 downto 0);
        M_AXIS_TVALID   : out std_logic;
        M_AXIS_TLAST : out std_logic;
        M_AXIS_TUSER : out std_logic
      );
    end component;

begin

-- Add user logic here
edlu_inst : edlu
   Port map (
       s_axis_aclk     => axis_aclk,
        s_axis_aresetn  => axis_aresetn,
        S_AXIS_TREADY   => s00_axis_tready,
        S_AXIS_TDATA => s00_axis_tdata,
        S_AXIS_TVALID   => s00_axis_tvalid,
        S_AXIS_TLAST => s00_axis_tlast,
```

```
        S_AXIS_TUSER => s00_axis_tuser(0),
        M_AXIS_TREADY   => m00_axis_tready,
        M_AXIS_TDATA => m00_axis_tdata,
        M_AXIS_TVALID   => m00_axis_tvalid,
        M_AXIS_TLAST => m00_axis_tlast,
        M_AXIS_TUSER => m00_axis_tuser(0)
   );
-- User logic ends

end arch_imp;
```

edlu.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity edlu is
  Port (
   s_axis_aclk    : in  std_logic;
   s_axis_aresetn : in  std_logic;
   S_AXIS_TREADY  : out std_logic;
   S_AXIS_TDATA   : in  std_logic_vector(7 downto 0);
   S_AXIS_TVALID  : in  std_logic;
   S_AXIS_TLAST   : in  std_logic;
   S_AXIS_TUSER   : in  std_logic;
   M_AXIS_TREADY  : in  std_logic;
   M_AXIS_TDATA   : out std_logic_vector(7 downto 0);
   M_AXIS_TVALID  : out std_logic;
   M_AXIS_TLAST   : out std_logic;
   M_AXIS_TUSER   : out std_logic
  );
end edlu;

architecture Behavioral of edlu is

component window_element
  Port (
    clk              : in    std_logic;
    reset_n          : in    std_logic;
    data_in          : in    std_logic_vector(9 downto 0);
    wr_en            : in    std_logic;
    data_out         : out   std_logic_vector(9 downto 0);
    data_valid       : out   std_logic
  );
end component;

component line_buffer
  Port (
    clk              : in    std_logic;
    reset_n          : in    std_logic;
    data_in          : in    std_logic_vector(9 downto 0);
    wr_en            : in    std_logic;
    data_out         : out   std_logic_vector(9 downto 0);
    data_valid       : out   std_logic;
    rd_en            : in    std_logic;
    empty            : out   std_logic;
    data_ready       : out   std_logic
  );
end component;

component convolution_and_threshold
  Port (
    clk          : in    std_logic;
    reset_n      : in    std_logic;
    wr_en        : in    std_logic;
    we_2         : in    std_logic_vector(9 downto 0);
    we_4         : in    std_logic_vector(9 downto 0);
    we_5         : in    std_logic_vector(9 downto 0);
    we_6         : in    std_logic_vector(9 downto 0);
    we_8         : in    std_logic_vector(9 downto 0);
    we_9         : in    std_logic_vector(9 downto 0);
    data_out     : out   std_logic_vector(7 downto 0);
    data_valid   : out   std_logic;
    user         : out   std_logic;
    last         : out   std_logic
  );
end component;
```

```
type state_machine is (write, read_out);
signal state : state_machine := write;

signal data_in_signal        :   std_logic_vector(9 downto 0);
signal data_wr_en_signal     :   std_logic;

signal we_1                  :   std_logic_vector(9 downto 0);
signal we_2                  :   std_logic_vector(9 downto 0);
signal we_3                  :   std_logic_vector(9 downto 0);
signal we_4                  :   std_logic_vector(9 downto 0);
signal we_5                  :   std_logic_vector(9 downto 0);
signal we_6                  :   std_logic_vector(9 downto 0);
signal we_7                  :   std_logic_vector(9 downto 0);
signal we_8                  :   std_logic_vector(9 downto 0);
signal we_9                  :   std_logic_vector(9 downto 0);

signal we_1_data_valid       :   std_logic;
signal we_2_data_valid       :   std_logic;
signal we_3_data_valid       :   std_logic;
signal we_4_data_valid       :   std_logic;
signal we_5_data_valid       :   std_logic;
signal we_6_data_valid       :   std_logic;
signal we_7_data_valid       :   std_logic;
signal we_8_data_valid       :   std_logic;
signal we_9_data_valid       :   std_logic;

signal fifo_fr_out           :   std_logic_vector(9 downto 0);
signal fifo_fr_data_valid    :   std_logic;
signal fifo_rd_en_fr         :   std_logic;
signal fifo_data_ready_fr    :   std_logic;
signal fifo_sr_out           :   std_logic_vector(9 downto 0);
signal fifo_sr_data_valid    :   std_logic;
signal fifo_rd_en_sr         :   std_logic;
signal fifo_data_ready_sr    :   std_logic;
signal fifo_empty            :   std_logic;

signal data_out_signal       :   std_logic_vector(7 downto 0);
signal data_valid_signal     :   std_logic;
signal user_signal           :   std_logic;
signal last_signal           :   std_logic;

signal row_counter : integer range 0 to 480 := 0;

begin

  window_element_1: window_element
    Port map (
      clk => s_axis_aclk,
      reset_n => s_axis_aresetn,
      data_in => data_in_signal,
      wr_en => data_wr_en_signal,
      data_out => we_1,
      data_valid => we_1_data_valid
    );

  window_element_2: window_element
    Port map (
      clk => s_axis_aclk,
      reset_n => s_axis_aresetn,
      data_in => we_1,
      wr_en => we_1_data_valid,
      data_out => we_2,
      data_valid => we_2_data_valid
    );

  window_element_3: window_element
    Port map (
      clk => s_axis_aclk,
      reset_n => s_axis_aresetn,
      data_in => we_2,
      wr_en => we_2_data_valid,
      data_out => we_3,
      data_valid => we_3_data_valid
    );

  first_row_line_buffer: line_buffer
    Port map(
      clk => s_axis_aclk,
      reset_n => s_axis_aresetn,
```

```
    data_in => we_3,
    wr_en => we_3_data_valid,
    data_out => fifo_fr_out,
    data_valid => fifo_fr_data_valid,
    rd_en => fifo_rd_en_fr,
    empty => open,
    data_ready => fifo_data_ready_fr
  );

window_element_4: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => fifo_fr_out,
    wr_en => fifo_fr_data_valid,
    data_out => we_4,
    data_valid => we_4_data_valid
  );

window_element_5: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => we_4,
    wr_en => we_4_data_valid,
    data_out => we_5,
    data_valid => we_5_data_valid
  );

window_element_6: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => we_5,
    wr_en => we_5_data_valid,
    data_out => we_6,
    data_valid => we_6_data_valid
  );

second_row_line_buffer: line_buffer
  Port map(
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => we_6,
    wr_en => we_6_data_valid,
    data_out => fifo_sr_out,
    data_valid => fifo_sr_data_valid,
    rd_en => fifo_rd_en_sr,
    empty => fifo_empty,
    data_ready => fifo_data_ready_sr
  );

window_element_7: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => fifo_sr_out,
    wr_en => fifo_sr_data_valid,
    data_out => we_7,
    data_valid => we_7_data_valid
  );

window_element_8: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => we_7,
    wr_en => we_7_data_valid,
    data_out => we_8,
    data_valid => we_8_data_valid
  );

window_element_9: window_element
  Port map (
    clk => s_axis_aclk,
    reset_n => s_axis_aresetn,
    data_in => we_8,
    wr_en => we_8_data_valid,
    data_out => we_9,
    data_valid => we_9_data_valid
```

```
    );

  conv_tresh_unit: convolution_and_threshold
    Port map(
      clk => s_axis_aclk,
      reset_n => s_axis_aresetn,
      wr_en => we_9_data_valid,
      we_2 => we_2,
      we_4 => we_4,
      we_5 => we_5,
      we_6 => we_6,
      we_8 => we_8,
      we_9 => we_9,
      data_out => data_out_signal,
      data_valid => data_valid_signal,
      user => user_signal,
      last => last_signal
    );

  process (s_axis_aclk, s_axis_aresetn)
   begin
        if rising_edge(s_axis_aclk) then
            if (s_axis_aresetn = '0') then
                S_AXIS_TREADY <= '0';
                data_in_signal <= (others => '0');
                data_wr_en_signal <= '0';
                M_AXIS_TDATA <= (others => '0');
                M_AXIS_TVALID <= '0';
                M_AXIS_TUSER <= '0';
                M_AXIS_TLAST <= '0';
            else
                if (S_AXIS_TVALID = '1' and S_AXIS_TLAST = '1') then
                    row_counter <= row_counter + 1;
                end if;
                data_in_signal(7 downto 0) <= S_AXIS_TDATA;
                data_in_signal(8) <= S_AXIS_TUSER;
                data_in_signal(9) <= S_AXIS_TLAST;
                data_wr_en_signal <= S_AXIS_TVALID;
                M_AXIS_TDATA <= data_out_signal;
                M_AXIS_TVALID <= data_valid_signal;
                M_AXIS_TUSER <= user_signal;
                M_AXIS_TLAST <= last_signal;
                case state is
                    when write =>
                        S_AXIS_TREADY <= M_AXIS_TREADY;
                        if(row_counter = 480) then
                            state <= read_out;
                        else
                            state <= write;
                        end if;
                    when read_out =>
                        S_AXIS_TREADY <= '0';
                        if(fifo_empty = '1') then
                            state <= write;
                            row_counter <= 0;
                        else
                            state <= read_out;
                        end if;
                    when others =>
                        state <= write;
                end case;

            end if;
        end if;
  end process;

with state select fifo_rd_en_fr <=
   (fifo_data_ready_fr and M_AXIS_TREADY) when write,
   M_AXIS_TREADY when read_out;

with state select fifo_rd_en_sr <=
   (fifo_data_ready_sr and M_AXIS_TREADY) when write,
   M_AXIS_TREADY when read_out;

end Behavioral;
```

window_element.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity window_element is
  Port (
    clk             : in    std_logic;
    reset_n         : in    std_logic;
    data_in         : in    std_logic_vector(9 downto 0);
    wr_en           : in    std_logic;
    data_out        : out   std_logic_vector(9 downto 0);
    data_valid      : out   std_logic
  );
end window_element;

architecture Behavioral of window_element is

begin

process (clk, reset_n)
  begin
   if rising_edge(clk) then
        if (reset_n = '0') then
            data_out <= (others => '0');
            data_valid <= '0';
        else
            data_out <= data_in;
            data_valid <= wr_en;
        end if;
    end if;
end process;

end Behavioral;
```

line_buffer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library xpm;
use xpm.vcomponents.all;

entity line_buffer is
  Port (
    clk          : in    std_logic;
    reset_n      : in    std_logic;
    data_in      : in    std_logic_vector(9 downto 0);
    wr_en        : in    std_logic;
    data_out     : out   std_logic_vector(9 downto 0);
    data_valid   : out   std_logic;
    rd_en        : in    std_logic;
    empty        : out   std_logic;
    data_ready   : out   std_logic
  );
end line_buffer;

architecture Behavioral of line_buffer is

signal fifo_reset     : std_logic;

begin

process (clk, reset_n)
  begin
   if rising_edge(clk) then
        if (reset_n = '0') then
            fifo_reset <= '1';
        else
            fifo_reset <= '0';
        end if;
    end if;
end process;

   xpm_fifo_sync_inst : xpm_fifo_sync
   generic map (
      DOUT_RESET_VALUE => "0",
      ECC_MODE => "no_ecc",
      FIFO_MEMORY_TYPE => "auto",
```

```
        FIFO_READ_LATENCY => 1,
        FIFO_WRITE_DEPTH => 1024,
        FULL_RESET_VALUE => 0,
        PROG_EMPTY_THRESH => 3,
        PROG_FULL_THRESH => 635,
        RD_DATA_COUNT_WIDTH => 1,
        READ_DATA_WIDTH => 10,
        READ_MODE => "std",
        USE_ADV_FEATURES => "1002",
        WAKEUP_TIME => 0,
        WRITE_DATA_WIDTH => 10,
        WR_DATA_COUNT_WIDTH => 1
    )
    port map (
        almost_empty => open,
        almost_full => open,
        data_valid => data_valid,
        dbiterr => open,
        dout => data_out,
        empty => empty,
        full => open,
        overflow => open,
        prog_empty => open,
        prog_full => data_ready,
        rd_data_count => open,
        rd_rst_busy => open,
        sbiterr => open,
        underflow => open,
        wr_ack => open,
        wr_data_count => open,
        wr_rst_busy => open,
        din => data_in,
        injectdbiterr => '0',
        injectsbiterr => '0',
        rd_en => rd_en,
        rst => fifo_reset,
        sleep => '0',
        wr_clk => clk,
        wr_en => wr_en
    );

end Behavioral;
```

convolution_and_threshold.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity convolution_and_threshold is
  Port (
    clk         : in    std_logic;
    reset_n     : in    std_logic;
    wr_en       : in    std_logic;
    we_2        : in    std_logic_vector(9 downto 0);
    we_4        : in    std_logic_vector(9 downto 0);
    we_5        : in    std_logic_vector(9 downto 0);
    we_6        : in    std_logic_vector(9 downto 0);
    we_8        : in    std_logic_vector(9 downto 0);
    we_9        : in    std_logic_vector(9 downto 0);
    data_out    : out   std_logic_vector(7 downto 0);
    data_valid  : out   std_logic;
    user        : out   std_logic;
    last        : out   std_logic
  );
end convolution_and_threshold;

architecture Behavioral of convolution_and_threshold is

constant    lplc_2              : std_logic_vector(3 downto 0):= "1111";
constant    lplc_4              : std_logic_vector(3 downto 0):= "1111";
constant    lplc_5              : std_logic_vector(3 downto 0):= "0100";
constant    lplc_6              : std_logic_vector(3 downto 0):= "1111";
constant    lplc_8              : std_logic_vector(3 downto 0):= "1111";

signal      we_2_signal         : std_logic_vector(7 downto 0);
signal      we_4_signal         : std_logic_vector(7 downto 0);
signal      we_5_signal         : std_logic_vector(7 downto 0);
```

```vhdl
signal      we_6_signal          : std_logic_vector(7 downto 0);
signal      we_8_signal          : std_logic_vector(7 downto 0);
signal      we_9_signal          : std_logic_vector(7 downto 0);

signal      multiply_result_2    : signed(12 downto 0);
signal      multiply_result_4    : signed(12 downto 0);
signal      multiply_result_5    : signed(12 downto 0);
signal      multiply_result_6    : signed(12 downto 0);
signal      multiply_result_8    : signed(12 downto 0);

signal      add_result_24        : signed(13 downto 0);
signal      add_result_56        : signed(13 downto 0);
signal      add_result_8         : signed(13 downto 0);

signal      add_result_2456      : signed(14 downto 0);
signal      add_result_8d        : signed(14 downto 0);

signal      add_result           : signed(15 downto 0);

signal      data_valid_signal    : std_logic;
signal      user_signal          : std_logic;
signal      last_signal          : std_logic;

signal      data_valid_signal_d1: std_logic;
signal      user_signal_d1       : std_logic;
signal      last_signal_d1       : std_logic;

signal      data_valid_signal_d2: std_logic;
signal      user_signal_d2       : std_logic;
signal      last_signal_d2       : std_logic;

signal      data_valid_signal_d3: std_logic;
signal      user_signal_d3       : std_logic;
signal      last_signal_d3       : std_logic;

signal      data_valid_signal_d4: std_logic;
signal      user_signal_d4       : std_logic;
signal      last_signal_d4       : std_logic;

begin

convolution_process: process(clk, reset_n)
  begin
    if rising_edge(clk) then
        if(reset_n = '0') then
            we_2_signal          <= (others => '0');
            we_4_signal          <= (others => '0');
            we_5_signal          <= (others => '0');
            we_6_signal          <= (others => '0');
            we_8_signal          <= (others => '0');
            multiply_result_2    <= (others => '0');
            multiply_result_4    <= (others => '0');
            multiply_result_5    <= (others => '0');
            multiply_result_6    <= (others => '0');
            multiply_result_8    <= (others => '0');
            add_result_24        <= (others => '0');
            add_result_56        <= (others => '0');
            add_result_8         <= (others => '0');
            add_result_2456      <= (others => '0');
            add_result_8d        <= (others => '0');
            add_result           <= (others => '0');
            data_valid_signal    <= '0';
            user_signal          <= '0';
            last_signal          <= '0';
            data_valid_signal_d1<= '0';
            user_signal_d1       <= '0';
            last_signal_d1       <= '0';
            data_valid_signal_d2<= '0';
            user_signal_d2       <= '0';
            last_signal_d2       <= '0';
            data_valid_signal_d3<= '0';
            user_signal_d3       <= '0';
            last_signal_d3       <= '0';
            data_valid_signal_d4<= '0';
            user_signal_d4       <= '0';
            last_signal_d4       <= '0';
        else
            we_2_signal          <= we_2(7 downto 0);
            we_4_signal          <= we_4(7 downto 0);
            we_5_signal          <= we_5(7 downto 0);
```

```
            we_6_signal         <= we_6(7 downto 0);
            we_8_signal         <= we_8(7 downto 0);
            we_9_signal         <= we_9(7 downto 0);
            data_valid_signal   <= wr_en;
            user_signal         <= we_9(8);
            last_signal         <= we_9(9);

            multiply_result_2   <= signed(lplc_2)*signed("0" & we_2_signal);
            multiply_result_4   <= signed(lplc_4)*signed("0" & we_4_signal);
            multiply_result_5   <= signed(lplc_5)*signed("0" & we_5_signal);
            multiply_result_6   <= signed(lplc_6)*signed("0" & we_6_signal);
            multiply_result_8   <= signed(lplc_8)*signed("0" & we_8_signal);
            data_valid_signal_d1<= data_valid_signal;
            user_signal_d1      <= user_signal;
            last_signal_d1      <= last_signal;


            add_result_24       <= (multiply_result_2(multiply_result_2'left) &
signed(multiply_result_2)) +
                                    (multiply_result_4(multiply_result_4'left) &
signed(multiply_result_4));
            add_result_56       <= (multiply_result_5(multiply_result_5'left) &
signed(multiply_result_5)) +
                                    (multiply_result_6(multiply_result_6'left) &
signed(multiply_result_6));
            add_result_8        <= (multiply_result_8(multiply_result_8'left) &
signed(multiply_result_8));
            data_valid_signal_d2<= data_valid_signal_d1;
            user_signal_d2      <= user_signal_d1;
            last_signal_d2      <= last_signal_d1;


            add_result_2456     <= (add_result_24(add_result_24'left) & signed(add_result_24)) +
                                    (add_result_56(add_result_56'left) & signed(add_result_56));
            add_result_8d       <= (add_result_8(add_result_8'left) & signed(add_result_8));
            data_valid_signal_d3<= data_valid_signal_d2;
            user_signal_d3      <= user_signal_d2;
            last_signal_d3      <= last_signal_d2;


            add_result          <= (add_result_2456(add_result_2456'left) & signed(add_result_2456))
+
                                    (add_result_8d(add_result_8d'left) & signed(add_result_8d));
            data_valid_signal_d4<= data_valid_signal_d3;
            user_signal_d4      <= user_signal_d3;
            last_signal_d4      <= last_signal_d3;
        end if;
    end if;
end process;

threshold_process: process(clk, reset_n)
  begin
    if rising_edge(clk) then
        if(reset_n = '0') then
            data_out <= (others => '0');
            data_valid <= '0';
            user <= '0';
            last <= '0';
        else
            if(add_result > 10) then
                data_out <= (others => '1');
            else
                data_out <= (others => '0');
            end if;
            data_valid <= data_valid_signal_d4;
            user <= user_signal_d4;
            last <= last_signal_d4;
        end if;
    end if;
end process;

end Behavioral;
```

2 priedas. *Xilinx Vivado HLS* aplinkoje realizuotos IP šerdies kodas

```
#include "hls_stream.h"
#include "ap_axi_sdata.h"

#define IMAGE_HEIGHT 480
#define IMAGE_WIDTH 640
#define WINDOW_SIZE 3
#define PIXEL_NUMBER IMAGE_HEIGHT*IMAGE_WIDTH
#define HALF_WINDOW_SIZE (((WINDOW_SIZE)-1)/2)

typedef ap_axis<8,1,1,1> axi_value;
typedef hls::stream<axi_value> axi_stream;

void edge_detection_hls_k7(axi_stream &s_axis, axi_stream &m_axis)
{
  #pragma HLS INTERFACE axis port = s_axis
  #pragma HLS INTERFACE axis port = m_axis
  #pragma HLS INTERFACE ap_ctrl_none port = return

  axi_value stream_value;
  int line_buffer[WINDOW_SIZE - 1][IMAGE_WIDTH];
  #pragma HLS ARRAY_PARTITION variable=line_buffer complete dim=1

  int window[WINDOW_SIZE][WINDOW_SIZE];
  #pragma HLS ARRAY_PARTITION variable=window complete dim=0

  int right_window_collumn[WINDOW_SIZE];
  #pragma HLS ARRAY_PARTITION variable=right_window_collumn complete

  int kernel[WINDOW_SIZE][WINDOW_SIZE] = {{0, -1, 0}, {-1, 4, -1}, {0, -1, 0}};
  #pragma HLS ARRAY_PARTITION variable=kernel complete dim=0

  int pixel_counter = IMAGE_WIDTH+HALF_WINDOW_SIZE+1;

  axi_value val_out;

  line_buffer_first_row_fill: for(int x = IMAGE_WIDTH-HALF_WINDOW_SIZE-1; x < IMAGE_WIDTH; x++) {
      #pragma HLS PIPELINE
      stream_value = s_axis.read();
      line_buffer[0][x] = stream_value.data;
  }

  line_buffer_second_row_fill: for(int x = 0; x < IMAGE_WIDTH; x++) {
      #pragma HLS PIPELINE
      stream_value = s_axis.read();
      line_buffer[1][x] = stream_value.data;
  }

  window_rows_fill: for(int y = HALF_WINDOW_SIZE; y < WINDOW_SIZE; y++) {
      window_collumns_fill: for(int x = HALF_WINDOW_SIZE; x < WINDOW_SIZE; x++) {
          #pragma HLS PIPELINE
          window[y][x] = line_buffer[y-1][x+IMAGE_WIDTH-WINDOW_SIZE];
      }
  }

  image_rows : for(int y = 0; y < IMAGE_HEIGHT; y++) {
      image_collumns : for(int x = 0; x < IMAGE_WIDTH; x++) {
          #pragma HLS PIPELINE
          int convolution_result = 0;
          window_row_convolution: for(int i = -HALF_WINDOW_SIZE; i <= HALF_WINDOW_SIZE; i++) {
   window_collumn_convolution: for(int j = -HALF_WINDOW_SIZE; j <= HALF_WINDOW_SIZE; j++) {
       if(((y+i) >= 0) && ((y+i) < IMAGE_HEIGHT) && ((x+j) >= 0) && ((x+j) < IMAGE_WIDTH)) {
           convolution_result += window[i+HALF_WINDOW_SIZE][j+HALF_WINDOW_SIZE]
                                     *kernel[i+HALF_WINDOW_SIZE][j+HALF_WINDOW_SIZE];
       }
   }
          }

          if(convolution_result > 10) {
   val_out.data = 255;
          }
          else {
   val_out.data = 0;
          }

          if(y == 0 && x == 0) {
    val_out.user = 1;
          }
```

```
        else{
    val_out.user = 0;
        }

        if(x == 639) {
    val_out.last = 1;
        }
        else{
    val_out.last = 0;
        }

        m_axis.write(val_out);

        int new_pixel = 0;
        if(pixel_counter < PIXEL_NUMBER) {
   stream_value = s_axis.read();
   new_pixel =  stream_value.data;
   pixel_counter++;
        }

        right_window_collumn[0] = line_buffer[0][x];
        right_window_collumn[1] = line_buffer[0][x] = line_buffer[1][x];
        right_window_collumn[2] = line_buffer[1][x] = new_pixel;

        window_row_shift: for(int wy = 0; wy < WINDOW_SIZE; wy++) {
            window_collumn_shift: for(int wx = 0; wx < WINDOW_SIZE - 1; wx++){
      window[wy][wx] = window[wy][wx+1];
   }
        }

        window_collumn_update: for(int wy = 0; wy < WINDOW_SIZE; wy++) {
   window[wy][WINDOW_SIZE-1] = right_window_collumn[wy];
        }
      }
  }
}
```

3 priedas. *Xilinx Vivado Design Suite* aplinkoje naudojamo testo kodas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use std.textio.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;

entity edge_detection_tb is
--  Port ( );
end edge_detection_tb;

architecture Behavioral of edge_detection_tb is

constant            file_path_i         : string := "ktu.pgm";
constant            file_path_o         : string := "output_image.pgm";
file                input_file          : text;
file                output_file         : text;
shared variable     magic_number_v      : string(1 to 2);
shared variable     max_encod_value_v   : integer;
shared variable     picure_hsize_v      : integer;
shared variable     picure_vsize_v      : integer;
shared variable     space_v             : character;

constant            axis_clk_period     : time := 10 ns;
signal              axis_clk            : std_logic := '0';
signal              axis_rst_n          : std_logic := '1';
signal              s_axis_tvalid       : std_logic := '0';
signal              s_axis_tready       : std_logic;
signal              s_axis_tdata        : std_logic_vector(7 downto 0) := x"00";
signal              s_axis_tkeep        : std_logic_vector(0 downto 0) := "1";
signal              s_axis_tstrb        : std_logic_vector(0 downto 0) := "0";
signal              s_axis_tuser        : std_logic_vector(0 downto 0) := "0";
signal              s_axis_tlast        : std_logic_vector(0 downto 0) := "0";
signal              s_axis_tid          : std_logic_vector(0 downto 0) := "0";
signal              s_axis_tdest        : std_logic_vector(0 downto 0) := "0";
signal              m_axis_tvalid       : std_logic;
signal              m_axis_tready       : std_logic := '0';
signal              m_axis_tdata        : std_logic_vector(7 downto 0);
signal              m_axis_tkeep        : std_logic_vector(0 downto 0);
signal              m_axis_tstrb        : std_logic_vector(0 downto 0);
signal              m_axis_tuser        : std_logic_vector(0 downto 0);
signal              m_axis_tlast        : std_logic_vector(0 downto 0);
signal              m_axis_tid          : std_logic_vector(0 downto 0);
signal              m_axis_tdest        : std_logic_vector(0 downto 0);

component edge_detection_wrapper is
  Port (
    axis_clk          : in  std_logic;
    axis_rst_n        : in  std_logic;
    s_axis_tvalid     : in  std_logic;
    s_axis_tready     : out std_logic;
    s_axis_tdata      : in  std_logic_vector(7 downto 0);
    s_axis_tkeep      : in  std_logic_vector(0 downto 0);
    s_axis_tstrb      : in  std_logic_vector(0 downto 0);
    s_axis_tuser      : in  std_logic_vector(0 downto 0);
    s_axis_tlast      : in  std_logic_vector(0 downto 0);
    s_axis_tid        : in  std_logic_vector(0 downto 0);
    s_axis_tdest      : in  std_logic_vector(0 downto 0);
    m_axis_tvalid     : out std_logic;
    m_axis_tready     : in  std_logic;
    m_axis_tdata      : out std_logic_vector(7 downto 0);
    m_axis_tkeep      : out std_logic_vector(0 downto 0);
    m_axis_tstrb      : out std_logic_vector(0 downto 0);
    m_axis_tuser      : out std_logic_vector(0 downto 0);
    m_axis_tlast      : out std_logic_vector(0 downto 0);
    m_axis_tid        : out std_logic_vector(0 downto 0);
    m_axis_tdest      : out std_logic_vector(0 downto 0)
  );
```

```
end component;

begin

edge_detection_ip : edge_detection_wrapper
Port map (
    axis_clk => axis_clk,
    axis_rst_n => axis_rst_n,
    s_axis_tvalid => s_axis_tvalid,
    s_axis_tready => s_axis_tready,
    s_axis_tdata => s_axis_tdata,
    s_axis_tkeep => s_axis_tkeep,
    s_axis_tstrb => s_axis_tstrb,
    s_axis_tuser => s_axis_tuser,
    s_axis_tlast => s_axis_tlast,
    s_axis_tid => s_axis_tid,
    s_axis_tdest => s_axis_tdest,
    m_axis_tvalid => m_axis_tvalid,
    m_axis_tready => m_axis_tready,
    m_axis_tdata => m_axis_tdata,
    m_axis_tkeep => m_axis_tkeep,
    m_axis_tstrb => m_axis_tstrb,
    m_axis_tuser => m_axis_tuser,
    m_axis_tlast => m_axis_tlast,
    m_axis_tid => m_axis_tid,
    m_axis_tdest => m_axis_tdest
);

clock_generation_process : process
begin
     axis_clk <= '1';
     wait for axis_clk_period/2;
     axis_clk <= '0';
     wait for axis_clk_period/2;
end process;

reset_generation_process : process
begin
     axis_rst_n <= '1';
     wait for 10us;
     axis_rst_n <= '0';
     wait for 100us;
     axis_rst_n <= '1';
     wait for 1000ms;
end process;

read_file_proc: process

variable fopen_status_v     : file_open_status;
variable line_v             : line;
variable pixel_v            : integer;
variable pixel_number_v     : integer := 0;
variable line_number_v      : integer := 0;

begin

    -- Open the file for reading
    file_open(fopen_status_v, input_file, file_path_i, READ_MODE);

    -- Check if the file opening was successful
    if fopen_status_v /= OPEN_OK then
        report "Error while opening the file" severity error;
    end if;

    -- Check if file type is P2 (ASCII PGM)
    readline(input_file, line_v);
    read(line_v, magic_number_v);
    if magic_number_v /= "P2" then
        report "File type not supported" severity error;
    end if;
```

```vhdl
    -- Get picture size
    readline(input_file, line_v);
    read(line_v, picure_hsize_v);
    read(line_v, space_v);
    read(line_v, picure_vsize_v);
    if((picure_hsize_v /= 640) and (picure_vsize_v = 480)) then
        report "Image size not supported" severity error;
    end if;

    -- Get max encoding value
    readline(input_file, line_v);
    read(line_v, max_encod_value_v);
    if(max_encod_value_v /= 255) then
        report "Encoding type not supported" severity error;
    end if;

    wait for 1ms;

    while not endfile(input_file) loop
        -- Read pixel value
        wait until axis_clk'event and axis_clk = '1';
        wait for axis_clk_period/2;
        s_axis_tdata <= x"00";
        s_axis_tvalid <= '0';
        s_axis_tuser <= "0";
        s_axis_tlast <= "0";
        if(s_axis_tready = '1') then
                pixel_number_v := pixel_number_v + 1;
                readline(input_file, line_v);
                read(line_v, pixel_v);
                s_axis_tdata <= std_logic_vector(to_unsigned(pixel_v, s_axis_tdata'length));
                s_axis_tvalid <= '1';
                if(pixel_number_v = 1 and line_number_v = 0) then
                    s_axis_tuser <= "1";
                end if;
                if(pixel_number_v = 640) then
                    s_axis_tlast <= "1";
                    pixel_number_v := 0;
                    line_number_v := line_number_v + 1;
                end if;
        end if;

    end loop;

    wait until axis_clk'event and axis_clk = '1';

    s_axis_tdata <= x"00";
    s_axis_tvalid <= '0';
    s_axis_tuser <= "0";
    s_axis_tlast <= "0";

    -- Close files
    file_close(input_file);

    wait for 1000ms;

end process;

write_file_proc: process

variable fopen_status_v      : file_open_status;
variable line_v              : line;
variable pixel_v             : integer;
variable line_number_v       : integer := 0;
variable pixel_number_v      : integer := 0;

begin

    wait for 100ns;
```

```
    -- Open the file for writing
    file_open(fopen_status_v, output_file, file_path_o, WRITE_MODE);
    if (fopen_status_v /= OPEN_OK) then
        report "Error while opening the file" severity error;
    end if;

    -- Write magic number to file
    write(line_v, magic_number_v);
    writeline(output_file, line_v);

    -- Write image size to file
    write(line_v, picure_hsize_v);
    write(line_v, space_v);
    write(line_v, picure_vsize_v);
    writeline(output_file, line_v);

    -- Write max encoding value
    write(line_v, max_encod_value_v);
    writeline(output_file, line_v);

    wait for 1ms;

    while (line_number_v < picure_vsize_v) loop

        -- Write pixel value
        wait until axis_clk'event and axis_clk = '1';
        wait for axis_clk_period/2;
        m_axis_tready <= '1';
        if(m_axis_tvalid = '1') then
            pixel_number_v := pixel_number_v + 1;
            pixel_v := to_integer(unsigned(m_axis_tdata));
            write(line_v, pixel_v);
            writeline(output_file, line_v);
            if(m_axis_tlast = "1") then
                line_number_v := line_number_v + 1;
            end if;
        end if;

    end loop;

    wait until axis_clk'event and axis_clk = '1';

    m_axis_tready <= '0';

    -- Close files
    file_close(output_file);

    assert FALSE Report "Simulation completed" severity failure;

end process;
```