KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Simonas Kuprys

ALU SystemC modelių tyrimas ir kūrimas

Magistro darbas

Darbo vadovas
dr. R. Damaševičius

Kaunas, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Simonas Kuprys

ALU SystemC modelių tyrimas ir kūrimas

Magistro darbas

Recenzentas
dr. G. Ziberkas
2007-05-25

Vadovas
dr. R. Damaševičius
2007-05-25

Atliko
IFM-1/5 gr. stud.
Simonas Kuprys
2007-05-25

Kaunas, 2007

Kuprys S. ALU SystemC modelių tyrimas ir kūrimas: Programų inžinerijos magistro darbas / vadovas dr. R.Damaševičius; Kauno technologijos universitetas, Informatikos fakultetas, Programų inžinerijos katedra. – Kaunas, 2007.- 77 p.

SANTRAUKA

Magistro darbe analizuojami aritmetinio loginio įtaiso (ALU) modeliai, operacijos ir architektūros. Išanalizavus mokslinę literatūrą, pasirenkama ALU architektūra bei atliekamų operacijų aibė. Realizuojamas dviejų pakopų sudalintos operacijų aibės ALU modelis. Atlikus apibendrinimą atliekami eksperimentai. Eksperimento modeliai modifikuojami – atliekamas ALU operacijų sudalinimas tarp ALU ir valdančios logikos (CU) operacijų poaibių. Nagrinėjami konkrečių ALU operacijų atlikimo pirmos arba antros pakopos modulyje pranašumai ir trūkumai. Sukurti parametrizuoti (bendriniai) ALU modeliai su kintamo duomenų pločio operandais. ALU modeliai aprašyti aparatūros aprašymo kalba *SystemC*, sumodeliuoti ir susintezuoti.

Gauti ALU *SystemC* modelių sintezės rezultatai parodė operacijų aibės sudalinimo bei skirtumo tarp retai ir dažnai naudojamų operacijų perkėlimo į skirtingas pakopas poveikį pagrindinėms lusto charakteristikoms: plotui, signalo vėlinimui ir energijos suvartojimui. Optimalus operacijų aibės sudalinimas dviejų pakopų architektūroje leidžia pasiekti reikiamos projektuojamo įtaiso charakteristikos pagerėjimą. Taikant šį metodą, galima kritinei (daug aparatūros resursų naudojančiai) ALU operacijai sudaryti mažesnį plotą luste užimantį arba mažiau energijos suvartojantį ALU variantą.

Kuprys S. ALU SystemC model analysis and development: Master's Work in Software Engineering / supervisor dr. R.Damaševičius; Department of Software Engineering Faculty of Informatics Kaunas University of Technology. – Kaunas, 2007. – 77 p.

SUMMARY

This work studies arithmetic logic unit (ALU) models, operations and architectures. ALU architecture and operation set is chosen based on the analysis of the known scientific publications. Two-stage divided operation set ALU model is implemented and used for the experiments.

The experimental ALU models are modified using different variants of partitioning of ALU operation set, when ALU operations are divided between main ALU block and control unit (CU). Pros and cons of ALU operation performance in the first or the second stage are examined.

Developed generic ALU models can be instatiated for data operands with variable data width. ALU models are coded in a high-level hardware description language SystemC, simulated and synthesized. The results of ALU SystemC model synthesis showed the effect of the division of the operation set on the main chip characteristics: area, delay and energy consumption and the difference of subdivision of rare or often used operations into different ALU stages. Optimal subdivision of operation set in two-stage ALU architecture allows getting a better performance of the designed device. Using this method the designer can select an instance of ALU that has a smaller area and consumes less energy for critical (using more hardware resources) operation.

Santrumpų ir terminų žodynas

AAL – žiūr. HDL

ALU – aritmetinis loginis įtaisas (*angl. Arithmetic Logic Unit*)

ASIC – specializuotų instrukcijų rinkinių schema (*angl. Application-Specific Integrated Circuit*)

CU – valdymo įtaisas (*angl. Control Unit*)

HDL – aparatūros aprašymo kalba (*angl. Hardware Desctiption Language*)

IP – intelektualinės nuosavybės komponentas (*angl. Intellectual Property*)

SystemC – artima C++ kalbai aparatūros aprašymo kalba

SoC – vienlustės sistemos (*angl. System on Chip*)

Verilog – viena iš naudojamų aparatūros aprašymo kalbų

VHDL – viena iš aparatūros aprašymo kalbų (*angl. Very high speed integrated circuit Hardware Description Language*)

VLSI – didelio integralumo laipsnio (*angl. Very Large Scale Integration*)

TURINYS

Lentelių sąrašas

Paveikslų sąrašas

1 Įvadas

Darbo aktualumas

Kasdienybėje šiuolaikinis žmogus naudoja įvairius skaičiavimus atliekančius prietaisus, o jei ir nenutuokia, kad naudoja – po eilinio įrenginio korpusu slypi visą darbą atliekantys lustai. Į lustą galima sutalpinti visą kompiuterinę ar elektroninę sistemą [15]. Jis gali atlikti skaitmenines, analogines, mišraus signalo arba radijo dažnio funkcijas – visas viename luste. Tipinis šių sistemų pritaikymas – integruotos sistemos: maršrutizatoriai, skaitmeniniai laikrodžiai ir laikmačiai, MP3 grotuvai, šviesoforai. Net ir delniniai kompiuteriai yra laikomi integruotomis sistemomis, tik jie turi didesnes programinio išplėtimo galimybes [16].

Iškylant naujoms technologijoms smulkėja schemų komponentai, didėja juose esančių loginių elementų skaičiaus. Kyla problemų dėl visų lusto dalių aprūpinimo energija, aušinimo, srovės nutekėjimo [11] bei dėl didėjančio glaudumo – elektromagnetinė jų elementų sąveika. Vis plačiau pritaikomi mobilūs įrenginiai, dažnai maitinami baterijos. Lusto suprojektavimas atsilieps įrenginio energijos vartojimui, šilumos išsklaidymui, aplinkos įtakos pakantumui.

Skaičiavimus atliekančios schemos turi procesorius. Procesoriaus viena sudėtinių dalių yra aritmetinis loginis įtaisas (ALU). Svarbu tirti ir gerinti šio komponento savybes. Dėl dažno jo panaudojimo pagerėjusios charakteristikos atsilieps visos sistemos spartai.

ALU komponentai – matematinius veiksmus atliekantys įtaisai, prie kurių pajungus valdančią logiką galima realizuoti procesorių [9]. Kompiuterio daugelis veiksmų yra atliekami ALU pagalba. ALU gauna duomenis iš procesoriaus registrų, duomenys apdorojami ir operacijų rezultatai patenka į ALU išvedimo registrus. Valdymo įtaisas (CU) valdo ALU nustatydamas valdančias grandis ir kontroliuodamas, kuri operacija atliekama.

ALU gali būti sudėtingi – tokiu būdu turintys didelę greitaveiką, kas tiesiogiai atsiliepia aparatūros plotui, įtaisai gali būti paprasti, tokiu būdu sudėtingėja valdymo logika, dažniau atliekamas operandų reikšmių keitimas, kas sąlygoja padidėjusį energijos suvartojimą [5].

Panaudojant universalią aparatūros, programinės įrangos ar visos sistemos aprašymo kalbą galima modeliuoti vienlustės sistemos elgesį, optimaliai padalinti kuri dalis bus realizuojama kaip programinis kodas, kuri kaip aparatūrinė įranga. Optimaliai, pagal poreikius numatyti operacijų aibės naudojimo dažnumą: aukštas ar žemas. Svarbu ir paties projekto universalumas – ar jį arba jo dalis bus galima panaudoti pakartotinai.

Vienas iš akivaizdžių *SystemC* privalumų [12] yra galimybė greitai pasidaryti paprastą lusto modelį. Šis modelis gali būti naudojamas būsimos architektūros įvertinimui. Be to, jau galima nuspręsti kurie blokai bus realizuoti kaip aparatūra, kurie kaip programinis kodas. Aparatūros blokai gali būti skaldomi į dar mažesnius, kurie visi bus sintezuojami.

Mikroprogramininė ir programinė dalys gali toliau būti realizuotos remiantis tuo pačiu *SystemC* aparatūros modeliu. Iš *SystemC* yra galimybė konvertuoti kodą į kitas populiarias AAL: *VHDL* ir *Verilog* – ir pagrindinis privalumas: aprašas bus sintezuojamas. *SystemC* modeliavimo greitis didesnis nei *VHDL* (ypač naudojant aukštesnį abstrakcijos lygį) [13] taip pat pagreitėja bendras projektavimo greitis [14], taupumas (plotas ir elementų skaičius po sintezės) didesnis nei *Verilog* [12].

Mokslinis naujumas

Darbe eksperimentiniai modeliai aprašomi aparatūros aprašymo kalba *SystemC*. Tai plačiai paplitusios programinės įrangos aprašymo kalbos C++ atšaka. [10] *SystemC* yra sistemos aprašymo kalba, nes savo tikrą galią demonstruoja transakcijų ir elgsenos modeliavime. *SystemC* bibliotekos aprašytos C++, yra galimybė modeliuoti lygiagrečius procesus, naudojami C++ duomenų tipai, bet yra ir papildomų, būdingu tik *SystemC*. *SystemC* platforma yra tuo pačiu AAL ir modeliavimo branduolys, rezultate gaunami vykdomi failai, kurie elgiasi, kaip aprašytas modelis.

Aktualu tirti aparatūros komponentų projektavimą dėl didėjančios lustų spartos. Šioje vietoje atsiranda problemos [11] dėl energijos išsklaidymo, lusto ploto, dėl itin glaudaus komponentų sąlyčio atsiranda parazitiniai triukšmai ir taip toliau.

Aktualu tirti ALU modelius, nes tai vienas pagrindinių skaičiavimo schemų komponentų, jo optimalus suprojektavimas, panaudojimas ir savybės įtakoja viso įrenginio spartą ir energijos suvartojimą.

Projektuojant ALU ir jį valdančią logiką galima atsižvelgti į šiame darbe pateiktas rekomendacijas.

Darbo tikslas – ištirti ALU architektūras, modelius ir operacijų sistemas, išanalizuoti ALU architektūrą realizuojančią operacijų sudalinimo modelį, eksperimentiškai ištirti įvairius operacijų sudalinimo modelius ir nustatyti optimalų operacijų sudalinimą.

Uždaviniai:

1) Išanalizuoti ALU atliekamas operacijas;
2) Ištirti ALU architektūrinius sprendimus vedančius prie laikinių ir energijos išsklaidymo charakteristikų gerėjimo;
3) Realizuoti operacijų sudalinimo modelį panaudojant *SystemC* AAL;
4) Ištirti įvairius operacijų aibės sudalinimo variantus;
5) Palyginti analizuotos literatūros išvadas su gautais rezultatais;

Darbo metodika

Darbą sudaro įžanga, trys skyriai: *2 Probleminės srities apžvalga*, *3 Metodai*, *4 Tyrimo rezultatai ir jų įvertinimas*; išvados, naudotos literatūros sąrašas ir priedai.

Darbo apimtis 77 puslapiai, jame yra 29 lentelės, 58 paveikslai, literatūros sąrašą sudaro 24 šaltiniai. Darbo planas:

1) mokslinės literatūros analizė;
2) eksperimentas;
3) testavimas;
4) rezultatų pateikimas ir įvertinimas.

Pirmame etape analizuojama mokslinė literatūra, apžvelgiamos ALU įtaisų atliekamos operacijos. Taip pat apžvelgiamos galimos ALU architektūros ir sprendimai siekiant pagerinti svarbiausias lusto charakteristikas.

Antrame etape, pagal atliktą apžvalgą siūlomas ALU ir jį valdančios logikos modelis, kuriamos jo modifikacijos ir atliekama sintezė. Pateikiamas siūlomos architektūros detalizavimas.

Trečiame etape pateikiami ir įvertinami gauti rezultatai.

Turint rezultatus ir analizę, ketvirtame etape skelbiamos išvados.

ALU ir jį valdančios logikos modeliai aprašomi *SystemC* sintakse ir modeliuojami *SystemC 2.0.1* kompiliatoriumi, gaunamas vykdomas failas. Kompiliuotą failą įvykdžius tarpiniai rezultatai matomi ekrane, sukuriamas *trace* failas. *Trace* failo turinys analizuojamas su *gtkwave 2.0.0*. Modeliui sukuriamas scenarijaus failas, kuriame nurodomi laiko ir kompiliavimo parametrai, scenarijaus failas skaitomas *Synopsys Design Analyzer 2006.03*. Atliekama sintezė, įrankis pateikia ataskaitas. Sintezėje naudojama *tc6a_cbacore* technologinė biblioteka.

2 Probleminės srities apžvalga

Apžvalga padalinta į dvi dalis. Pirmoje jų analizuojamos 4 ALU modelių atliekamos operacijos. Antroje dalyje apžvelgiami galimi architektūriniai sprendimai ir jų rezultatai. Apibendrinus pateikiamos išvados.

2.1 ALU operacijų apžvalga

Šios apžvalgos tikslas yra išanalizuoti ALU architektūras, įvairius jų tipus, palyginti pranašumus ir trūkumus, įvairias charakteristikas, palyginti ir suklasifikuoti ALU atliekamas operacijas. Apsibrėžti operacijų aibę eksperimentui. Nagrinėjimui pateikiami šie ALU:

- ✓ L4C383;
- ✓ 74HC/HCT181;
- ✓ CD40181BMS;
- ✓ PDSP1601/PDSP1601A.

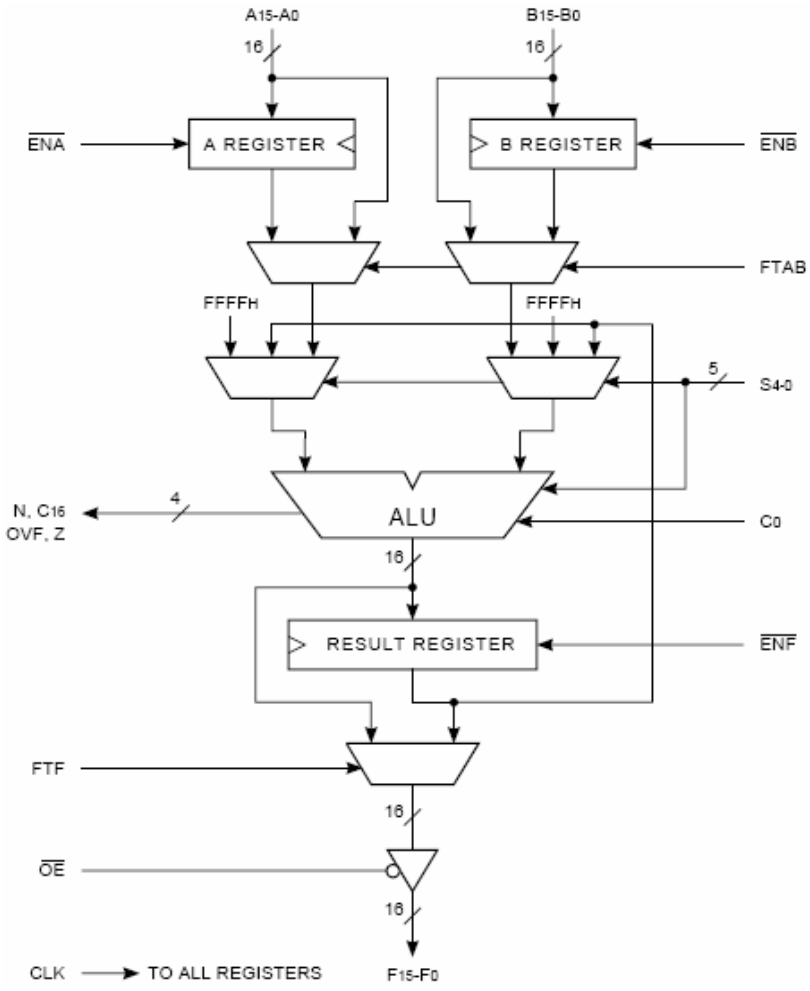
Toliau yra pateikiama kiekvieno įtaiso atliekamų operacijų apžvalga.

2.1.1 L4C383

L4C383 [1] yra 16 bitų kaskadinis ALU. Jis gali atlikti iki 32 skirtingų aritmetinių arba loginių operacijų.

2.1.1.1 L4C383 architektūros schema, jos aprašymas

Paveiksle 1 pateikiama L4C383 blokinė diagrama iš ALU specifikacijos duomenų lapo.



1 pav. *L4C383 architektūros schema*

L4C383 [1] yra 16 bitų kaskadinis ALU. Jis gali atlikti iki 32 skirtingų aritmetinių arba loginių operacijų. Šis įtaisas yra lankstus, pasižymi greitaveika. Naudojant kaskadinį jungimą įtaisas gali būti pritaikytas 32 bitų ar ilgesnėms instrukcijoms atlikti.

Įtaiso architektūra pateikiama sekančiuose skyriuose.

2.1.1.2 L4C383 architektūros funkcinės charakteristikos

2.1.1.2.1 L4C383 architektūra

Galima lygiagrečiai apjungti 2 ALU operacijoms su 32 bitų ir ilgesniais duomenims atlikti. L4C383 operuoja su dviem 16 bitų operandais A ir B, jo išėjimas yra 16 bitų pločio ir vadinamas F. Įtaisas yra kontroliuojamas 5 bitų funkcijos išrinkimo magistrale ir gali atlikti 19 aritmetinių ir 13 loginių funkcijų. Tiek įėjimų tiek išėjimų reikšmės yra laikomos registruose, tačiau naudojantis vidiniais grįžtamaisiais ryšiais juos galima apeiti ir, pavyzdžiui, išėjimo registre esančią ALU reikšmę perduoti vienam arba abiems įėjimams, taip pritaikant grandinines operacijas ir reikšmės kaupimą (akumuliaciją).

2.1.1.2.2 L4C383 ALU operacijos

Operacija išrenkama 5 bitų magistrale $S_4 - S_0$. ALU atliekamos funkcijos pateiktos 1 lentelėje.

1 lentelė. L4C383 atliekamos operacijos

Op. kodas	***Funkcija***
00000	A + B + C0
00001	A OR B
00010	A + !B + C0
00011	!A + B + C0
00100	A + C0
00101	!A OR F
00110	A – 1 + C0
00111	!A + C0
01000	A + F + C0
01001	A OR F
01010	A + !F + C0
01011	!A + F + C0
01100	F + B + C0
01101	!A OR B
01110	F + !B + C0
01111	!F + B + C0
10000	A XOR B
10001	A AND B
10010	!A AND B
10011	A XNOR B
10100	A XOR F
10101	A AND F
10110	!A AND F
10111	ALL 1's + C0
11000	B + C0
11001	A AND !B
11010	!B + C0
11011	B – 1 + C0
11100	F + C0
11101	A OR !B
11110	F – 1 + C0
11111	!F + C0

2.1.2.3.4 L4C383 ALU vėliavėlės

L4C383 turi perpildymo, pernašos, ženklo ir nulio statuso vėliavėles. Apribojimą turi pernašos bitas, jis nusako reikšmę tik su 19 aritmetinių operacijų. Nulio vėliavėlė nustatoma kai visi 16 išėjimo bitų yra žemame lygyje. Vėliavėlės žymimos C_{16}, *OVF* , *Zero* ir *N*.

$C_{16} = G_{12} + P_{15}C_0$ - pernaša.

$OVF = C_{15} \oplus C_{16}$ - perpildymas.

$Zero = 1$ - kai visi 16 išėjimo (F) bitų yra nulis.

N – ALU operacijos ženklas.

2.1.2.3.5 L4C383 registrai

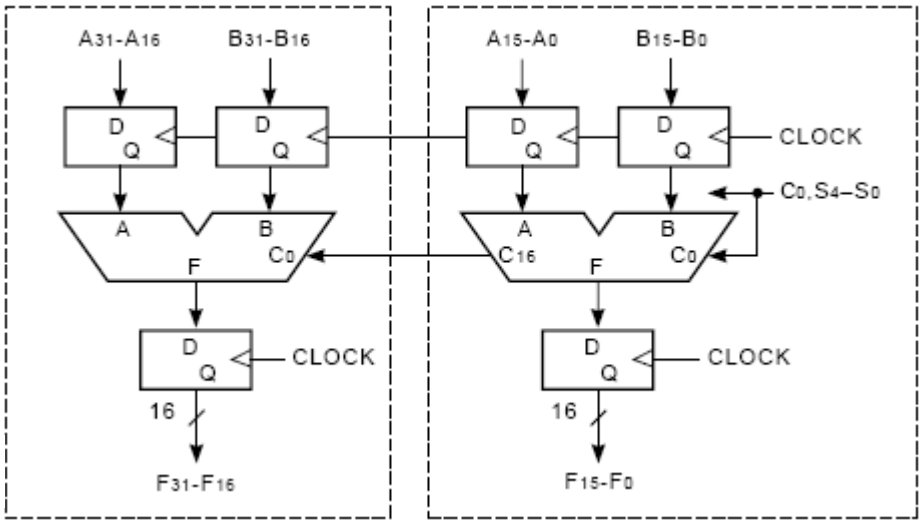
L4C383 turi du 16 bitų įeigos registrus operandams A ir B bei vieną operacijų rezultatui F. Registrai, tai galioja ir įėjimo ir išėjimo registrams, yra sužadinami kylančiu bendro sisteminio laikrodžio frontu. Registras A įjungiamas reikšmės įvedimui nustatant įėjime $\overline{ENA}$ žemą lygį, registras B – nustatant įėjime $\overline{ENB}$ žemą lygį. Kai nors vienas signalas $\overline{ENA}$ arba $\overline{ENB}$ yra aukštame lygyje – valdomo registro reikšmė nesikeis. L4C383 architektūra leidžia priimti argumentus ir iš vienos 16 bitų magistralės. Tais atvejais kai nenaudojami įėjimai su registrais abu operandai A ir B gali būti perduoti FTAB linija.

Išvesties registras valdomas $\overline{ENF}$ signalu, kai šis yra žemo lygio – duomenys iš ALU sekančiame takte bus persiųsti į išvesties registrą (schemoje „Result register“). Išjungiant ši registrą, jame bus saugojami tarpiniai rezultatai, o tuo pat metu įėjime jau bus naujos operandų reikšmės. Tribūvio elemento valdomo $\overline{OE}$ signalu pagalba L4C383 gali būti sukonfigūruotas kaip dvikryptė magistralinė sistema.

Išvesties registras gali būti apeitas naudojant *FTF* kontrolinį signalą (kai *FTF* = 1). Kai panaudojamas *FTF* signalo valdymas – ALU operacijos rezultato duomenys patenka į išėjimą aplenkdami išvesties registrą, tačiau jis veikia toliau ir jo valdymas vis tiek kontroliuojamas $\overline{ENF}$ signalu. Išvesties registro turinys vėl bus išėjime, kai *FTF* nustatomas į žemą lygį.

2.1.2.3.6 L4C383 kaskadinis apjungimas

L4C383 sujungimas į 32 bitus yra atliekamas sujungiant žemiausios skilties C_{16} išėjimą su didžiausia įėjimo C_0 skiltimi. Signalai $S_4 - S_0$, $\overline{ENA}$, $\overline{ENB}$ ir $\overline{ENF}$ yra bendros abiems įtaisams. Abiejų įtaisų $Zero$ vėliavėles reikia sujungti logine *AND* operacija. Didžiausios skilties OVF ir C_{16} reikšmės yra teisingos 32 bitų rezultatui. Paveiksle 2 schematiškai pateiktas šio jungimo pavyzdys.



Aukščiausia skiltis

Žemiausia skiltis

2 pav. *Kaskadinis jungimas*

Kaskadinis jungimas ilgesnėms instrukcijoms nei 32 bitai atliekamas kaip ir 32 bitų atveju – tiesiog kiekvienos skilties išėjimą C_{16} sujungiant su sekančios aukščiausios skilties C_0 įėjimu.

2.1.1.3 L4C383 ALU architektūros technologinės charakteristikos

Lentelėje 2 pateikta L4C383 svarbesnės technologinės charakteristikos.

2 lentelė. L4C383 technologinės charakteristikos

Charakteristika	***Vertė, vienetai***
Minimalus ciklo periodas	15 *ns*
Įtaiso užimamas plotas	*ND*
Energijos sunaudojimas	*125 mW*
Maitinimo įtampa (karinis standartas)	4.50 V ≤ VCC ≤ 5.50 V
Veikimo dažnis	66 *MHz*

2.1.2 74HC/HCT181

74HC/HCT181 [2] yra 4 bitų ALU. Jis gali atlikti 16 aritmetinių arba loginių operacijų.

74HC/HCT181 funkcinė diagramos pateikiamoje literatūroje nėra, todėl ji darbe nepateikiama.

2.1.2.1 74HC/HCT181 architektūros funkcinės charakteristikos

2.1.2.1.1 74HC/HCT181 architektūra

74HC/HCT181 yra 4 bitų didelės spartos lygiagretus ALU. Įtaisas kontroliuojamas 4 bitų magistrale $S_0 - S_3$ ir režimo kontrolės įėjimu *M*. Įtaisas gali atlikti 16 loginių arba 16 aritmetinių operacijų, viso 32 skirtingas operacijas. Operacijų sąrašas pateiktas 3 lentelėje, sekančioje analizės dalyje.

2.1.2.1.2 74HC/HCT181 ALU operacijos

Operacija išrenkama 4 bitų magistrale $S_0 - S_3$. ALU atliekamos funkcijos pateiktos 3 lentelėje. Priklausomai nuo *M* signalo reikšmės atliekama 16 loginių (*M = 1*) arba aritmetinių (*M = 0*) operacijų.

3 lentelė. 74HC/HCT181 atliekamos operacijos

Loginės (M=1)	
0000	!A
0001	!(A OR B)
0010	!A AND B
0011	0
0100	!(A AND B)
0101	!B
0110	A XOR B
0111	A AND !B
1000	!A OR B
1001	!(A XOR B)
1010	B
1011	A AND B
1100	1
1101	A OR !B
1110	A OR B
1111	A

Aritmetinės (M=0)	
0000	A
0001	A OR B
0010	A OR !B
0011	-1
0100	A + A!B
0101	(A OR B) + A!B
0110	A – B – 1
0111	A!B - 1
1000	A + AB
1001	A + B
1010	(A OR !B) + AB
1011	AB - 1
1100	A + A
1101	(A OR B) + A
1110	(A OR !B) + A
1111	A - 1

2.1.2.1.3 74HC/HCT181 ALU vėliavėlės

Kai režimo parinkimo signalas *M* yra aukšto lygio, visos vidinės pernašos yra sulaikomos ir įrenginys atlieka logines operacijas kurios pateiktos lentelėje. Kai *M* yra žemas lygis – pernašos yra aktyvios ir įtaisas atlieka aritmetines operacijas su dviem 4 bitų operandais. 74HC/HCT181 turi pilną vidinę paspartintąją pernašą ir teikia ištisinę pernašą tarp įrenginių

naudodamas C_{n+4} išėjimą arba paspartintą pernašą tarp paketų naudodamas pernašos perdavimo ($\overline{P}$) ir generavimo ($\overline{G}$) signalus. Signalai $\overline{P}$ ir $\overline{G}$ neįtakojami ateinančios pernašos.

Kai spartos reikalavimas nėra griežtas galima naudoti paprastą pernašą apjungiant vieno įrenginio pernašos išėjimą C_{n+4} su sekančio įrenginio pernašos įėjimu C_n. Kai reikalinga didelė operacijų greitaveika įrenginys naudojamas apjungiant su „182“ paspartintos pernašos grandimi. Vienas toks įrenginys reikalingas kiekvienai apjungtai keturių „181“ įtaisų grupei.

2.1.2.1.4 74HC/HCT181 komparatorius

Komparatoriaus išėjimas ($A = B$) yra aukšto lygio kai visi keturi funkcijos išvesties bitai ($\overline{F_0} - \overline{F_3}$) yra „1“ ir gali būti panaudotas parodyti loginį visų 4 bitų ekvivalentiškumą kai įtaisas yra atimties režime. Išėjimas $A = B$ yra atviro kolektoriaus, todėl gali logine operacija AND būti apjungtas su kitų įtaisų rezultatais ir taip gaunamas palyginimas daugiau nei 4 bitais. $A = B$ taip pat gali būti panaudotas kartu su C_{n+4} signalu, taip gaunant rezultatus $A > B$ ir $A < B$.

2.1.2.2 74HC/HCT181 ALU architektūros technologinės charakteristikos

Lentelėje 4 pateiktos 74HC/HCT181 svarbesnės technologinės charakteristikos.

4 lentelė. 74HC/HCT181 technologinės charakteristikos

Charakteristika	***Vertė, vienetai***
Minimalus ciklo periodas	30 *ns*
Įtaiso užimamas plotas	*214.7 mm*^{2}
Energijos sunaudojimas	*26 mW*
Maitinimo įtampa (karinis standartas)	2 V ≤ VCC ≤ 6 V
Veikimo dažnis	*200 MHz*

2.1.3 CD40181BMS

CD40181BMS [3] yra 4 bitų lygiagretus ALU. Jis gali atlikti 16 skirtingų aritmetinių arba loginių operacijų.

2.1.3.1 CD40181BMS architektūros schema, jos aprašymas

CD40181BMS duomenų lape nėra pateikiama kita schema išskyrus loginę diagramą. Ši analizėje dėl netinkamo aiškumo nepateikiama. Ją rasti galima [3] šaltinyje, 8 puslapyje.

CD40181BMS yra 4 bitų lygiagretus ALU. Jis gali atlikti 16 skirtingų aritmetinių arba loginių operacijų. Įtaisas pasižymi tokiomis savybėmis kaip didelė maitinimo įtampa (iki 20V), turi palyginimo išėjimą, pernašą.

2.1.3.2 CD40181BMS ALU architektūros funkcinės charakteristikos

2.1.3.2.1 CD40181BMS architektūra

CD40181BMS yra mažai energijos suvartojantis 4 bitų lygiagretus aritmetinis loginis įtaisas galintis atlikti 16 dvejetainės aritmetikos operacijų su dviem keturių bitų operandais ir 16 loginių funkcijų su dviem loginiais kintamaisiais. Režimą kontroliuojantis įėjimas *M* parenka logines (*M* = *aukštas*) arba aritmetines (*M* = *žemas*) operacijas. Keturi įėjimai $S_0 - S_3$ parenka norimą loginę ar aritmetinę funkciją kaip AND, OR, NAND, NOR, XOR ir XNOR loginiame režime ir sumą, skirtumą, mažinimą vienetu, perstūmimą kairėn ir tiesioginį perdavimą aritmetiniame režime. ALU atliekamos operacijos pateikiamos 5 lentelėje.

2.1.3.2.2 CD40181BMS ALU operacijos

Operacija išrenkama 4 bitų magistrale $S_0 - S_3$. ALU atliekamos funkcijos pateiktos 5 lentelėje.

5 lentelė. CD40181BMS atliekamos operacijos

Loginės (M=1)	
0000	!A
0001	!(A AND B)
0010	!A OR B
0011	1
0100	!(A OR B)
0101	!B
0110	!(A XOR B)
0111	A OR !B
1000	!A AND B
1001	A XOR B
1010	B
1011	A OR B
1100	0
1101	A AND !B
1110	A AND B
1111	A

Aritmetinės (M=0)	
0000	A – 1
0001	AB – 1
0010	!(AB) – 1
0011	-1
0100	A + (A OR !B)
0101	AB + (A OR !B)
0110	A – B – 1
0111	A OR !B
1000	A + (A OR B)
1001	A + B
1010	A!B + (A OR B)
1011	A OR B
1100	A + A
1101	AB + A
1110	A!B + A
1111	A

2.1.3.2.3 CD40181BMS ALU vėliavėlės

CD40181BMS turi logiką pilnai paspartintos pernašos operacijai naudojamai greitai signalo generacijai pernašos perdavimo ($\overline{P}$) ir generavimo ($\overline{G}$) signalus. Naudojant įtaiso paspartintos pernašos generatorių kartu su keliais CD40181BMS atliekama didelės greitaveikos aritmetinė operacija su ilgais žodžiais. Sistemose kur sparta nėra pirmas prioritetas galima naudoti ištisinės pernašos išėjimą C_{n+4}.

2.1.3.2.4 CD40181BMS Komparatorius

CD40181BMS turi komparatoriaus funkciją atliekantį išėjimą $A = B$, kuris išduodą aukštą lygį kai abu 4 bitų įėjimai A ir B yra lygūs ir įtaisas yra atimties režime. Taip pat galima išgauti santykinį operandų A ir B palyginimo rezultatą panaudojant pernašos įėjimą C_n ir paspartintos pernašos išėjimą C_{n+4}. Tam papildomai reikia pervesti įtaisą į atimties režimą bei atlikti išorinį dekodavimą.

2.1.3.3 CD40181BMS ALU architektūros technologinės charakteristikos

Lentelėje 6 pateikta CD40181BMS svarbesnės technologinės charakteristikos.

6 lentelė. CD40181BMS technologinės charakteristikos

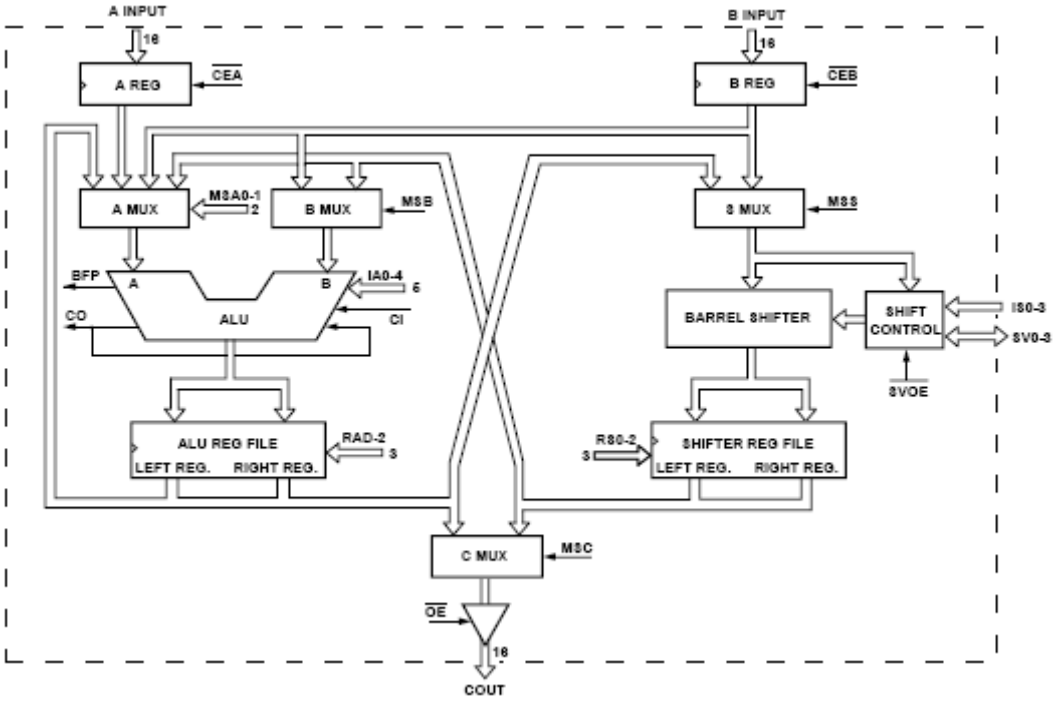
Charakteristika	***Vertė, vienetai***
Minimalus ciklo periodas	800 ns
Įtaiso užimamas plotas	*8.470 mm*^{2}
Energijos sunaudojimas	*100 mW, 500 mW (max)*
Maitinimo įtampa (karinis standartas)	-0.50 V ≤ VCC ≤ 20 V
Veikimo dažnis	1,25 *MHz* (5V), 5 *MHz* (10V)

2.1.4 PDSP1601/PDSP1601A

PDSP1601/PDSP1601A [4] yra 16 bitų ALU su integruotu 16 bitų ciklinio postūmio registru. Jis gali atlikti 32 skirtingas aritmetines arba logines operacijas. Gali būti naudojamas kaskadiniam apjungimui taip gaunant ilgesnius žodžius.

2.1.4.1 PDSP1601 ALU architektūros schema, jos aprašymas

Paveiksle 3 pateikiama PDSP1601/PDSP1601A blokinė diagrama iš ALU specifikacijos duomenų lapo.



3 pav. *PDSP1601/PDSP1601A architektūros schema*

PDSP1601/PDSP1601A yra 16 bitų ALU su integruotu 16 bitų ciklinio postūmio registru. Jis gali atlikti 32 skirtingas aritmetines arba logines operacijas. Įtaisas palaiko daugiaciklines operacijas ir gali būti naudojamas kaskadiniam apjungimui taip gaunant ilgesnius žodžius.

PDSP1601 sudaro keturi blokai: ALU, ciklinio postūmio registras ir du registrų failai.

2.1.4.2 PDSP1601 architektūros funkcinės charakteristikos

2.1.4.2.1 PDSP1601 architektūra

PDSP1601/PDSP1601A atlieka 32 instrukcijas kurios pateiktos žemiau esančioje 7 lentelėje. ALU įėjimai parenkami multiplekseriais *A* ir *B*. Instrukcijos yra buferizuojamos ir užlaikomos kol negaunamas sisteminio laikrodžio impulsas. Operandų *A* ir *B* reikšmės gali būti paimtos iš: *A* ir *B* įėjimų registrų, ALU registrų failo, ciklinio postūmio registro registrų failo.

PDSP1601/PDSP1601A yra ALU ir ciklinio postūmio registras vienoje mikroschemoje. Įtaisas turi pernašos įėjimą *CI* ir teikia pernašos išėjimą *CO*. Papildomai, kiekvieno ciklo

pabaigoje pernašos išėjimo reikšmė *CO* yra perkeliama į vieno bito vidinį registrą, tokiu būdu ją galimą ALU sekančio ciklo įėjime. Pats įtaisas palaiko daugiacikles, daugkartinio tikslumo operacijas. Plačiau apie jas pateikta dalyje 2.4.3.3 „Daugiaciklės kaskadinės operacijos“.

Rezultato išdavimas kontroliuojamas signalu $\overline{OE}$ kuris tiesiogiai valdo išėjime esantį tribūvį elementą.

2.1.4.2.2 PDSP1601 ALU operacijos

Lentelėje 7 pateikiamos PDSP1601/PDSP1601A atliekamos operacijos.

7 lentelė. PDSP1601/PDSP1601A atliekamos operacijos

Op. kodas	***Funkcija***	
00000	0	(reset)
00001	!A + 1	
00010	!A + CI	(kaskadinė)
00011	!A + CO	(multiciklinė)
00100	A/2	(su ženklu)
00101	A/2	(multiciklinė su kairiu ALU registru)
00110	A/2	(multiciklinė su dešiniu ALU registru)
00111	A/2	(multiciklinė su ciklinio stūmimo registru)
01000	A + B + CI	(kaskadinė)
01001	A + B + CO	(multiciklinė)
01010	A + !B + 1	
01011	A + !B + CI	(kaskadinė)
01100	A + !B + CO	(multiciklinė)
01101	!A + B + 1	
01110	!A + B + CI	(kaskadinė)
01111	!A + B + CO	(multiciklinė)
10000	A AND B	
10001	A AND !B	
10010	!A AND B	
10011	A OR B	
10100	!A OR B	
10101	A XOR B	
10110	A	
10111	!A	
11000	0 (OVF)	
11001	0 (UND1)	
11010	0 (UND2)	
11011	0 (ZERO)	
11100	h0001	
11101	h00FF	
11110	h000F	
11111	h5555	

2.1.4.2.3 PDSP1601 daugiaciklės ir kaskadinės operacijos

ALU aritmetinės instrukcijos turi dvi ar tris opcijas kiekvienai aritmetinei operacijai. ALU operacijoms, pavyzdžiui atimčiai, atlikti reikia dviejų operandų, todėl kai kuriose instrukcijose žemiausia skiltis visada priskiriama priverstinai.

Vartotojas gali apjungti įtaisus kaskadiniu būdu taip pat atlikti daugiacikles operacijas su vienu įtaisu taip padidinant aritmetinį tikslumą.

Norint apjungti įtaisus kaskadiniu jungimu, naudojamos kaskadinės aritmetinės instrukcijos visoms skiltims išskyrus mažiausios skilties 16 bitų baitą.

Norint atlikti daugiaciklę operacija su vienu įtaisu naudojamos *daugiaciklinės aritmetinės instrukcijos* panaudojant vidinį *CO* bitą ir pritaikant visoms skiltims išskyrus žemiausią 16 bitų baitą.

2.1.4.2.4 PDSP1601 dalyba iš dviejų

ALU turi keturias instrukcijas skirtas postūmio dešinėn (dalybai iš 2) atlikimui su išplėstinio tikslumo žodžiais. Šie žodžiai (iki 64 bitų) gali būti saugomi dvejuose mikroschemoje esančiuose registrų failuose. Kai stumiami žemiausios skilties 16 bitų, likę, aukštesnės skilties bitai turi būti užpildomi mažiausios skilties reikšmėmis iš sekančios aukščiausios 16 bitų skilties. Tai atliekama ALU operacijomis *A2RAL, A2RAR, A2RSX*, kurios ir parodo naujos aukščiausios skilties duomenų šaltinį.

Kai stumiama aukščiausia 16 bitų skiltis, kita aukščiausia skiltis turi būti užpildoma stumiamosios kopija, kad nebūtų prarastas ženklas.

2.1.4.2.5 PDSP1601 konstantos

PDSP1601/PDSP1601A turi keturias instrukcijas skirtas nustatyti ALU į pastovią reikšmę. Šios reikšmės naudojamos kaip šablonas. Pavyzdžiui, nustačius vieną bitą į aukštą lygį pačioje žemiausioje pozicijoje jį po to galima perstumti į bet kurią kitą vietą iš 16 bitų ciklinio postūmio registro pagalba. Tai įgalina, pavyzdžiui, ALU *AND* funkciją pasirinkti vieną iš įėjimo bitų.

ALU taip pat turi iš pradinio nustatymo operaciją (*CLR*). Ją įvykdžius išvalomi *A* ir *B* įėjimo registrai, *R1* ir *R2* postūmio kontrolės registrai, vidinis *CO* bitas, abu registrų failai, taip pat šiuo metu atliekamas BFP vėliavėlės atliekamos funkcijos programavimas.

2.1.4.2.6 PDSP1601 ciklinis postūmis

PDSP1601/PDSP1601A turi integruotą ciklinio postūmio registrą, kuris gali atlikti 16 papildomų instrukcijų su duomenimis. Toks integruotas sprendimas naudojamas siekiant sparčiau atlikti postūmio ar persukimo operacijas su ALU operandais. Duomenys į postūmio modulį išrenkami multiplekseriu *S* (matomas 3 paveiksle pateiktoje įtaiso schemoje). Postūmio modulio instrukcijos buferizuojamos kaip ir ALU įėjimas ir vykdomos tik atėjus sinchro signalui. Pats ciklinio postūmio registras gali atlikti loginio, aritmetinio ir ciklinio postūmio operacijas į abi puses. Atliekama instrukcija valdoma 4 bitų magistrale ateinančia iš atskiro valdymo bloko (jis taip pat matomas schemoje 3 paveiksle).

Kadangi darbe analizuojamos ALU architektūros, toliau nepateikiama ciklinio postūmio registro instrukcijų ir jų vykdymo apžvalga.

2.1.4.2.7 PDSP1601 registrai

PDSP1601/PDSP1601A turi du 16 bitų duomenų įvedimo registrus *A* ir *B*, taip pat du programuojamus 4 bitų registrus *R1* ir *R2* esančius ciklinio postūmio registro valdymo schemoje, bei dar du bendro naudojimo registrų failus, kurių kiekvienas turi po du 16 bitų registrus.

Operandų registrai *A* ir *B* yra sužadinami kylančiu bendro sisteminio laikrodžio frontu. Registras *A* įjungiamas reikšmės įvedimui nustatant įėjime $\overline{CEA}$ žemą lygį, registras *B* – nustatant įėjime $\overline{CEB}$ žemą lygį. Kai nors vienas signalas $\overline{CEA}$ arba $\overline{CEB}$ yra aukštame lygyje – valdomo registro reikšmė nesikeis. Abiejuose skiltyse 15 skiltis yra aukščiausia bito skiltis.

Registrai *R1* ir *R2* gali būti užkrauti iš $SV_0 - SV_3$ įėjimo arba ciklinio postūmio registro valdymo schemoje esančio pirmenybės šifratoriaus. Registrai naudojami postūmio kontrolėje.

Reikšmės į du registrų failus patenka kylančio sinchro signalo metu, vienas registro failas yra valdomas ALU išėjimo, kitas ciklinio postūmio registro. Abu registrų failai gali atlikti po 8 analogiškas instrukcijas, tokias kaip duomenų užkrovimas į nei vieną, vieną ar abu registrus esančius faile. Taip pat yra instrukcijos duomenų nuskaitymui iš vieno iš failo registrų arba ciklinio postūmio registro rezultato tiesioginis perdavimas.

2.1.4.2.8 PDSP1601 ALU vėliavėlės

PDSP1601/PDSP1601A turi vartotojo programuojamą BFP vėliavėlę. Ją galima suprogramuoti reaguoti į vieną iš keturių sąlygų. Dvi sąlygos skirtos blokinėms slankaus kablelio operacijoms, kitos dvi sąlygos aptinka perpildymą arba nulinį rezultatą. Kad įvyktų perpildymas

ALU rezultatas turi būti perpildęs ir 16 (ženklo) skiltį. Nulinio rezultato sąlyga tenkinama kai ALU rezultatas yra nulis visuose išėjimuose.

Vėliavėlės *CO* ir *CI* skirtos pernašos operacijoms. *CO* yra vidinis vieno bito registras ir išduoda pernašą iš aukščiausios ALU rezultato skilties. *CI* yra ateinanti pernaša, reikšmė įrašoma žemiausioje ALU skiltyje. Kiekvieno ALU ciklo pabaigoje pernašos išėjimo reikšmė *CO* yra perkeliama į vieno bito vidinį registrą, tokiu būdu ją galimą ALU sekančio ciklo įėjime.

2.1.4.3 PDSP1601 ALU architektūros technologinės charakteristikos

Lentelėje 8 pateikta PDSP1601/PDSP1601A svarbesnės technologinės charakteristikos.

8 lentelė. PDSP1601/PDSP1601A technologinės charakteristikos

Charakteristika	***Vertė, vienetai***
Minimalus ciklo periodas	100 ns
Įtaiso užimamas plotas	*ND*
Energijos sunaudojimas	*300 mW (max)*
Maitinimo įtampa (karinis standartas)	-0.50 V ≤ VCC ≤ 7.00 V
Veikimo dažnis	20 *MHz*

2.2 ALU architektūrų apžvalga

ALU architektūrų apžvalgoje pateikiami trys sprendimai, kurių tikslas paspartinti veiksmus ALU įtaise juos sudalinant į pakopas, lygiagretinant pačius įtaisus, bandant nuspėti galimus rezultatus ir tikintis, kad korekcijai, neteisingai nuspėjus, sunaudojama mažiau resursų nei tikrinimui prieš operaciją.

Pirmoji apžvelgiama architektūra skirta ASIC. Pateikiamas tyrimas orientuotas į energijos suvartojimo sumažinimą. Analizuojamas santykis tarp paprastos aparatūros su dideliu instrukcijų skaičiumi ir sudėtingais aparatūriniais elementais su bendru mažesniu instrukcijų skaičiumi.

Antroji pristatoma architektūra – dviejų pakopų ALU paremtas ekvivalentinių transformacijų matematiniu principu. Iš pradžių reikalingos operacijos perrašomos panaudojant keturias bazines funkcijas, tada atliekami veiksmai supaprastintu ALU įtaisu.

Trečioji architektūra remiasi galimų ALU rezultatų nuspėjimu, atlieka išankstinį apvalinimą, vieno bito nuspėjimą. Taip tikimasi sumažinti vėlinimą kritinėse grandyse.

2.2.1 ALU architektūrų, skirtų *ASIC*, energijos išsklaidymo palyginimas

2.2.1.1 Probleminė sritis

[5] Dėl didėjančio didelio integralumo laipsnio (VLSI) schemų projektavimo sudėtingumo, paskutiniais metais energijos išsklaidymas tapo dideliu rūpesčiu. Mobiliose sistemose nuo įrenginio energijos išsklaidymo priklauso jo veikimo laikas po baterijos įkrovimo. Didelio našumo sistemose didelį susirūpinimą kelia pakavimo ir šilumos pašalinimo kaštai atsiradę dėl žymaus energijos išsklaidymo. Tuo pat metu, vis daugiau ir daugiau skaitmeninių schemų dalių yra realizuojamos naudojant programuojamus ASIC, kurie yra pritaikyti konkrečioms programoms ir dėl programavimo galimybės yra lankstūs. Tuo pačiu energijos sklaida mažesnė nei bendros paskirties procesorių.

ALU architektūra turi įtakos laiko planavimui, energijos išsklaidymui ir užimamam plotui. ALU energijos išsklaidymas labiausiai priklauso nuo ALU komponentų atliekamų aritmetinių operacijų tokių kaip sudėtis, sandauga, postūmiai ir kitos, skaičiaus. Veiksmų sinchronizavimas ir užimamas plotas priklauso nuo ALU komponentams realizuoti panaudotų aritmetinių grandinių tipo. Tarp šių parametrų egzistuoja abipusė priklausomybė ir todėl, kuriant mažai energijos suvartojančią architektūrą ASIC, reikia skirti dėmesį mažam energijos išsklaidymui, pralaidos optimizavimui ir užimamam plotui.

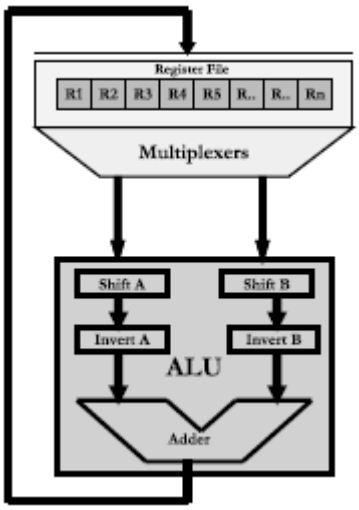
Tam, kad būtų sukurta efektyviai energiją naudojanti ALU architektūra, galima rinktis iš mažo sudėtingumo laipsnio architektūrų su dideliu naudojimo dažniu arba architektūrų su dideliu sudėtingumo lygiu ir žemu naudojimo dažniu ir stebėti ar pasirinkimas veda prie mažo energijos išsklaidymo. Pirmu atveju ALU yra sukurtas naudojant mažiau sudėtingas aritmetines grandis tokias kaip sudėties elementai ir kombinacinius postūmio registrus. Visos sandaugos operacijos reikalingos realizuoti filtravimo galimybę yra įvykdytos naudojant paprastas postūmio ir sudėties operacijas, ko pasėkoje ALU tampa dažnai naudojamu ir pasižymi dideliu naudojimo dažniu bei dideliu resursų dalinimosi laipsniu. Antru atveju daugybos procesas realizuotas naudojant sudėtingas aritmetines grandines tokias kaip aparatūrinius daugiklius kurie atlieka darbą viena instrukcija vietoj pakartotino postūmio ir sudėties. Tokiu būdu prie šio tipo ALU priėjimo dažnis yra ribotas ir ALU pasižymi žemu naudojimo dažniu ir mažu resursų dalinimosi laipsniu.

2.2.1.2 Efektyvių ALU architektūrų ASIC projektavimas

Tam, kad būtų suprojektuotos skirtingos ALU architektūros ir ištirtas jų energijos išsklaidymas ASIC, svarstoma IIR (Infinite Impulse Response) filtro realizacija. Filtras realizuojamas ASIC kuris susideda iš registrų failo, multiplekserių, ALU ir kontrolerio.

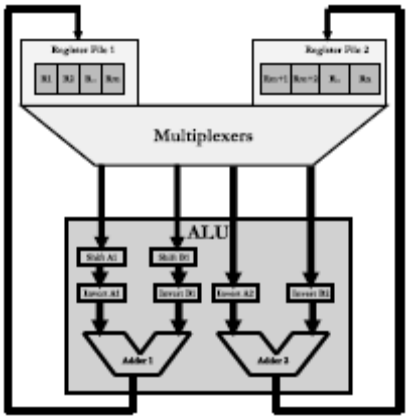
Filtras susideda iš 4 adapterių, kiekvienas iš jų atitinka vieną filtro koeficientą, reikalauja trijų sudėties ir vienos daugybos operacijos, turi du išėjimus. Kad būtų išsiaiškinta ALU architektūra, kuri efektyviausiai panaudoja energiją, kiekvienai kategorijai, išskirtai įžangoje, yra modeliuojamos ir tiriamos dvi architektūros.

Pirmoji architektūra postūmio ir sudėties architektūra. Vidutiniškai, pasirinktiems filtro koeficientams, vieno adapterio funkcionalumui realizuoti, kai koeficiento žodžio ilgis 10, šis metodas reikalauja 6-7 instrukcijų. Ši architektūra pasižymi didžiausiu resursų dalinimosi laipsniu, nes atlieka visas reikalingas aritmetines operacijas, reikalingas pilnos filtro struktūros realizavimui, vienu sumatoriumi. Architektūros struktūrinė schema pateikiama 4 paveiksle.



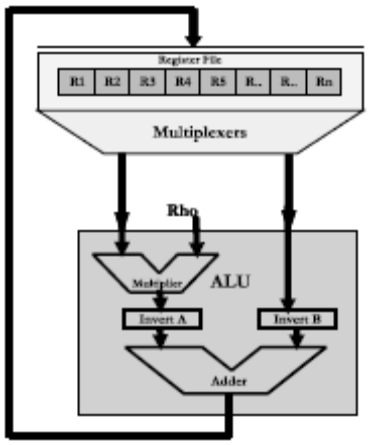
4 pav. *Postūmio ir sudėties architektūra*

Antroji architektūra yra postūmio ir sudėties architektūra su dviem sumatoriais, ši architektūra yra pirmosios modifikacija. Jos struktūra parodyta 5 paveiksle. Pagrindinis šios architektūros privalumas, kad du adapterio išėjimai yra suskaičiuojami per atskiras duomenų magistrales ir vienu metu. Registrai pirmoje architektūroje yra paskirstyti dviem registro failo dalims ir kiekviena dalis sujungta su viena iš magistralių. Šis registrų valdymo būdas sąlygoja smarkų klaidingų perėjimų sumažėjimą registrų įėjimuose ir bendrą perjungimo talpą vienai instrukcijai architektūros registro faile.



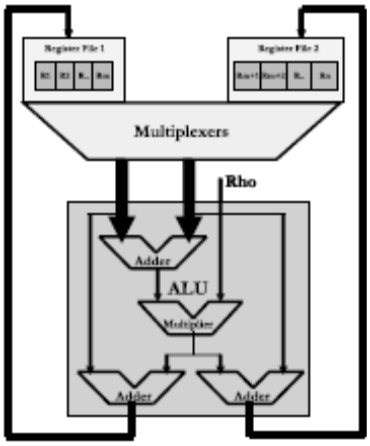
5 pav. *Postūmio ir sudėties architektūra su dviem sumatoriais*

Trečioji architektūra realizuota daugiklio ir akumuliatoriaus pagrindu. Schema pateikta 6 paveiksle. Kad atliktų vieno adapterio veiksmus architektūra reikalauja trijų instrukcijų. Dėl aparatūrinio daugiklio panaudojimo, smarkiai sumažėja instrukcijų, reikalingų filtro struktūros realizavimui, skaičius, lyginant su dviem ankstesnėmis architektūromis.



6 pav. *Daugiklio ir akumuliatoriaus architektūra*

Ketvirtoji architektūra paremta adapterio struktūra yra specifinė, skirta bangų skaitmeniniams filtrams. Schematinis vaizdas pateiktas 7 paveiksle. Svarbiausias šios architektūros privalumas, kad ji reikalauja tik vienos instrukcijos atlikti vieno adapterio funkcijai. Tokiu būdu bendras instrukcijų, reikalingų realizuoti išbaigtą filtro struktūra, skaičius sumažinamas iki minimumo.



7 pav. *Bangų skaitmeniniu filtru paremta architektūra*

2.2.1.3 Modeliavimo rezultatai

Architektūros buvo suprojektuotos kaip parametrizuoti *VHDL* modeliai su 10, 24 ir 32 bitų duomenų žodžiais bei 10 ir 16 bitų koeficientų ilgiais. Visi užimamo ploto ir galios modeliavimo rezultatai paremti sinteze su *Synopsys Design* kompiliatoriumi, po to atliekant išdėstymą su *Cadence Encounter*, remiantis 0,18µm specializuotų integrinių schemų technologija, ir galiausiai išdėstymo galios modeliavimą atliekant su *Synopsys Nanosim*.

Lentelėse 9 ir 10 pateikti galios ir užimamo ploto rezultatai atsižvelgiant į keturias ALU architektūras ir duomenų žodžio ilgius 16 ir 32 bitai, kai koeficientas yra 16 bitų.

9 lentelė. ALU architektūrų galios matavimo rezultatai

Architektūra	***Duomenų žodžio ilgis***	***Top, µW***	***ALU, µW***	***Registrai, µW***	***Multiplek-seriai, µW***	***Valdymas, µW***
Postūmio ir sumatoriaus	16	1467,00	610,97	277,93	171,36	92,93
	32	3960,00	1729,90	524,88	572,00	98,09
Postūmio ir dviejų sumatorių	16	1352,20	527,31	188,57	158,41	133,68
	32	3653,30	1705,30	390,47	592,72	136,25
Daugiklio ir akumuliatoriaus	16	795,46	458,74	131,41	49,71	36,34
	32	1784,70	1209,50	281,33	101,08	37,13
Bangų skaitmeninis filtras	16	656,20	491,71	68,53	32,05	9,84
	32	1430,50	1128,40	144,93	66,89	10,17

10 lentelė. ALU architektūrų ploto ir spartos rezultatai

Architektūra	***Duomenų žodžio ilgis***	***Instrukcijų skaičius***	***Taktinis dažnis, MHz***	***Plotas, µm²***	***Pralaidumas, MHz***
Postūmio ir sumatoriaus	16	42	83,3	46690	3,71
	32	42	83,3	133378	3,33
Postūmio ir dviejų sumatorių	16	38	83,3	51531	4,72
	32	38	83,3	134578	3,76
Daugiklio ir	16	14	27,7	54723	3,96

akumuliatoriaus	32	14	27,7	108900	3,82
Bangų skaitmeninis filtras	16 32	6 6	11,9 11,9	56803 107057	6,98 5,55

Iš 9 lentelės galima matyti, kad energijos išsklaidymas dviejų architektūrų su postūmio registru ir sumatoriumi registrų failuose yra didesnis nei kitose dvejose architektūrose. Tai yra dėl didesnio instrukcijų skaičiaus. Kiekvienai instrukcijai į registrus įrašomos laikinos reikšmės ir tokiu būdu bendras duomenų kitimo registrų faile skaičius yra didesnis nei kitose dvejose architektūrose.

Kadangi bendras instrukcijų skaičius abiejose postūmio ir sudėties architektūrose yra didesnis, tam, būtų realizuoti abiejų architektūrų kontroleriai, yra įtraukiama vis daugiau apjungiamosios logikos. Tai daro valdymo bloką sudėtingesniu. Taip pat dėl padidėjusio instrukcijų skaičiaus padidėja taktinis dažnis, o nuo jo priklauso valdymo bloko galia. Su dideliu taktiniu dažniu ir sudėtinga valdiklio grandine abiejų postūmio ir sumavimo architektūrų valdiklio galia yra smarkiai didesnė nei kitų lyginamų architektūrų.

Abiejose postūmio ir sudėties architektūrose energijos išsklaidymas multiplekseriuose taip pat yra didesnis. Dėl didelio instrukcijų skaičiaus jie naudojami pakankamai dažnai. Vykdant kiekvieną instrukciją, dėl kintančio registro išėjimo, duomenų įėjime įvyksta perėjimas, tas pats vyksta ir išrinkimo linijose, instrukcijoms naudojant skirtingus registrus. Tai sąlygoja didelį energijos išsklaidymą.

Dviejų postūmio ir sudėties architektūrų atveju, dėl pasikartojančių postūmio ir sudėties operacijų ALU įėjimo reikšmė kinta gana dažnai, palyginus su kitom, tai taip pat sąlygoja energijos išsklaidymo padidėjimą. Didesnis energijos išsklaidymas ir esant ilgesniam duomenų žodžio ilgiui. Tai vyksta dėl vėlinimo konstantų kurios nustatomos sintezės įrankiui, kuris bando optimizuoti architektūrą ir tilpti į apribojimus panaudodamas papildomą logiką, pavyzdžiui operacijas atliekant lygiagrečiai. Pažymėtina, kad nors daugiklio-akumuliatoriaus ir bangų skaitmeninio filtro architektūros turi sudėtingų aparatūrinių elementų kaip, pavyzdžiui, daugiklis, šiose architektūrose energijos išsklaidymas yra mažesnis. Taip yra dėl retesnio ALU įėjimų kitimo ir mažų vėlinimo konstantų apribojimų. Be to, sintezės įrankis optimizuoja ALU panaudodamas tris daugiklius kurie turi mažiau sumatorių nei masyvo daugikliai.

Dėl padidėjusio energijos išsklaidymo registruose, kontroleryje, multipleksersiuose bei ALU abiejose postūmio ir sumavimo architektūrose energijos išsklaidymas visame ASIC būtų didesnis. ASIC suprojektuotas remiantis daugiklio ir akumuliatoriaus bei bangų skaitmeninio filtro ALU architektūrų pagrindu turi mažesnį energijos išsklaidymą dėl mažesnio instrukcijų skaičiaus, retesnio multiplekserių ir registrų naudojimo, sumažėjusio taktinio dažnio ir mažesnio papildomos logikos kiekio kurį sąlygoja ALU vėlinimo apribojimai.

Iš 10 lentelės matoma, kad esant ilgesniam duomenų žodžio ilgiui abiejų postūmio ir sumavimo architektūrų užimamas plotas yra didesnis, nei kitų dviejų, nors šios yra suprojektuotos su mažiau sudėtingu ALU. Tai visų pirma, kaip aprašyta ALU architektūrų energijos palyginime, įtakoja vėlinimo apribojimai nustatomi sintezės įrankiui, o antra – architektūros pralaidumas.

2.2.1.4 Apibendrinimas

Straipsnyje [5] buvo sumodeliuotos skirtingos ALU architektūros skirtos ASIC. Didesnio laipsnio resursų dalinimasis, daugybos pakeitimas postūmio ir sudėties operacijomis neveda link sumažėjusio energijos išsklaidymo. Padidėjusio energijos išsklaidymo didelio resursų dalinimosi schemose priežastis yra papildoma energijos sklaida sąlygojama papildomo atminties ar registrų skaitymo. Svarbu atkreipti dėmesį ne tik į ALU energijos išsklaidymą, bet ir į padidėjusį energijos išsklaidymą dėl registrų ir atminties skaitymo. Apžvelgtuose atvejuose energijos sklaida du kartus mažesnė variantuose kur pasirenkamas ALU su daugikliu.

2.2.2 Dviejų pakopų ALU architektūra

[6] Pagal ekvivalentinės transformacijos matematinį principą (detalizuotas žemiau sekančiame skyriuje), visos funkcijos gali būti suskaičiuotos remiantis keturiomis bazinėmis funkcijomis (*sudėtis su pernaša*, *bitinė sudėtis*, *ARBA* bei *IR*). Dviejų pakopų ALU architektūroje visi įėjimo operandai pirmoje pakopoje transformuojami į atitinkamą formą, tada antroje pakopoje jie paduodami galutiniam skaičiavimui panaudojant vieną iš keturių bazinių funkcijų. Nauja architektūra naudinga dizaino kokybei, modifikuojamumui ir sutrumpina laiką perderinant ALU funkcijų kaitą.

2.2.2.1 Teorinis pagrindimas

Technologijos progresas leidžia viename luste integruoti didelį skaičių tranzistorių, tokiu būdu galima viename luste realizuoti ištisas sistemas, įskaitant mikroprocesorius, ASIC, atmintis ir periferinius įtaisus. Šių naujos klasės sistemų plėtra vadinama vienlustėmis sistemomis (SoC). SoC projektavimo praktikoje, sistemos lygio projektavimas, viso darbo eigoje, reikalauja periodinio perdalinimo atlikimo tarp programinės ir aparatūrinės įrangos. Todėl yra būtina suskurti naują projektavimo metodologiją, kad būtų patenkinti reikalavimai ir būtų lengviau modifikuoti ar papildyti projektą kai prireikia funkcijos kaitos.

Įvairios grandinės, vykdančios duomenų apdorojimo instrukcijas, dažniausiai apjungiamos į vieną schemą vadinamą aritmetiniu-loginiu įrenginiu. ALU sudėtingumas apsprendžiamas pagal tai kokias aritmetines instrukcijas jis atlieka. Rankinio projektavimo laikotarpiu buvo pasirinktas schematinis rinkimas ir ALU buvo gerai realizuojami suderinant visas reikalingas funkcijas. Dėl ALU schemos reguliarumo buvo pasiūlyta bitinių sekcijų metodologija. Ji realizuoja projekto pakartotinį panaudojimą ir sumažina projektavimo sudėtingumą, tokiu būdu schemą vėliau galima optimizuoti rankiniu būdu. Vis dėlto schemos pagaminimas remiantis šia metodologija yra varginantis ir ją sunku modifikuoti norint perderinti ALU funkcijų pasikeitimus.

Nauja dviejų pakopų ALU architektūra įveikia abiejų: bitinių sekcijų bei paprastos aparatūros aprašymo kalbos dizaino trūkumus. Tam tikroje apimtyje tai kompromisas tarp šių dizaino požiūrių.

2.2.2.2 Ekvivalentinė transformacija

Ekvivalentinė transformacija [23, 24] yra naujas skaičiavimo požiūris, paremtas semantiką išlaikančiomis transformacijomis. Duotas aprašas sėkmingai paverčiamas ekvivalentišku, kuris yra paprastesnis. Tai kartojama, kol pasiekiamas itin paprastas aprašas, panaudojantis minimalią kalbos konstrukcijų aibę. Atliekami pasižymėjimai: ET srities kintamieji, d - deklaruojamas aprašas, Γ - specializavimosi sistema, visa sistema turi prasmę $M(\Gamma, d)$. Ekvivalentinė transformacija yra (Γ, d) transformaciją išlaikant prasmę $M(\Gamma, d)$. Tada galimos dvi aprašo transformacijos:

(1) $ET_1 : (\Gamma_1, d) \to (\Gamma_2, d)$ - keičiama sistema iš Γ_1 į Γ_2 išlaikant d;

(2) $ET_{21} : (\Gamma, d_1) \to (\Gamma, d_2)$ - keičiamas aprašas iš d_1 į d_2 išlaikant Γ.

Visos PIC16C57 instrukcijos yra 12 bitų ilgumo ir turi įvairius formatus ir operandus. Į ALU ateina du duomenų šaltiniai, vienas yra įmontuotas registras W, kitas – duomenys ir sistemos duomenų magistralės.

11 lentelė. ALU operacijos

Funkcija	***Ekvivalentinė funkcija***	***Panaudotos instrukcijos***
W[7:0] + bus[7:0]	W[7:0] + bus[7:0] + 0	ADDWF
W[7:0] and bus[7:0]	W[7:0] and bus[7:0]	ANDWF, ANDLW
0	00h and 00h	CLRF, CLRW, CLRWDT
!(bus[7:0])	!00h ⊕ bus[7:0]	COMF
bus[7:0] – 1	!00h + bus[7:0] + 0	DECF, DECFSZ
bus[7:0] + 1	!00h + bus[7:0] + 1	INCF, IORLW
W[7:0] or bus[7:0]	W[7:0] or bus[7:0]	IORWF, IORLW

bus[7:0]	00h ⊕ bus[7:0]	MOVF, MOVLW, RETLW, GOTO, CALL, RLF, RRF, SWAPF
W[7:0]	W[7:0] ⊕ 00h	MOVWF, NOP, OPTION, SLEEP, TRIS
bus[7:0] – W[7:0]	!W[7:0] + bus[7:0] + 1	SUBWF
W[7:0] ⊕ bus[7:0]	W[7:0] ⊕ bus[7:0]	XORWF, XORLW
bus[i] = 0, i = b bus[i], i ≠ b	$!2^h$ and bus[7:0]	BCF
bus[i] = 1, i = b bus[i], i ≠ b	2^h or bus[7:0]	BSF
2^h and bus[7:0]	2^h and bus[7:0]	BTFCS, BTFSS

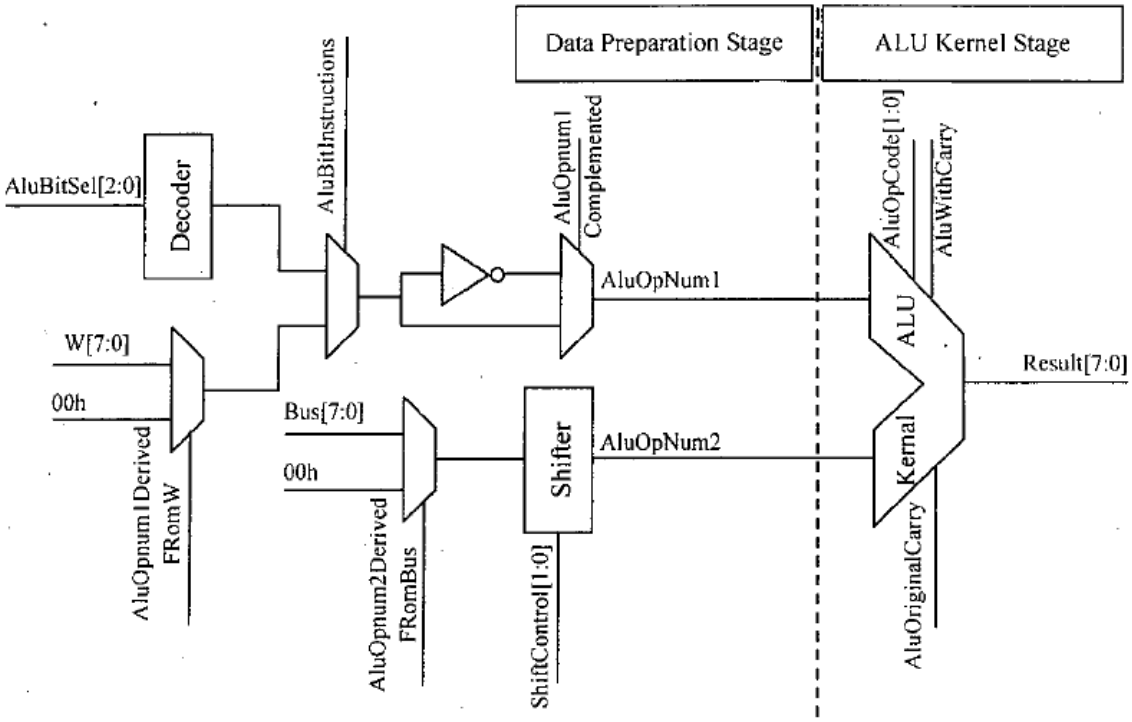
11 lentelėje pateikiamas galimų ekvivalentinių išraiškų rinkiniai kiekvienai PIC16C57 instrukcijai. Jie perrašyti įprastine forma, operandai suskaičiuojami naudojant vieną iš keturių bazinių funkcijų. Keturios bazinės funkcijos yra *sudėtis su pernaša*, *bitinė sudėtis* (išskirties disjunkcija), *ARBA* bei *IR*. Pavyzdžiui, funkcija !bus[7:0] perrašoma !00h ⊕ bus[7:0], o funkcija bus[7:0] – W[7:0] perrašoma į !W[7:0] + bus[7:0] + 1.

2.2.2.3 Dviejų pakopų architektūra

Dviejų pakopų architektūra paremta ekvivalentinės transformacijos principu. Visos instrukcijos yra perrašomos reguliaria forma, kiekvienos funkcijos pirminis skaičiavimas gali būti išskaidytas į dvi pakopas. Pirmoje pakopoje pradiniai duomenys transformuojami į ekvivalentinę įprastinę formą, po transformacijos jie išrenkami šiuo metu vykdomai instrukcijai. Antroje pakopoje, reikiamai transformuoti duomenys galiausiai skaičiuojami ALU branduolio grandinėje panaudojant bazines funkcijas. Taigi pirmoji pakopa – terminuota duomenų paruošimo pakopa.

ALU branduolys atlieka tik 4 funkcijas. Tokiu būdu ALU branduolys yra labai paprastas ir jį itin nesunku suprojektuoti, kad būtų išgautos skirtingos charakteristikos, skirtingais būdais, kaip, pavyzdžiui, su ištisine arba su paspartinta pernaša.

8 paveikslas iliustruoja dviejų pakopų ALU architektūrą, kuri duomenų paruošimo pakopoje yra optimizuota kelių lygių multiplekseriais, remiantis resursų dalinimosi principu. Pavyzdžiui, kaip matoma 12 lentelėje, antrasis ekvivalentinės išraiškos instrukcijos operandas yra arba duomenys iš sistemos duomenų magistralės, arba pastovus „0“. Taigi, mes galime panaudoti multiplekserį vykdomos instrukcijos reikiamų duomenų pasirinkimui. Kelių lygiu multipleksavimo būdu, yra projektuojamos pirmo ekvivalentinių išraiškų operando duomenų šakos.



8 pav. *Dviejų pakopų ALU architektūra*

Dviejų pakopų architektūros privalumai yra šie:

- ✓ kai pakeičiamas ALU funkcionalumas (pašalinamos ar papildomos funkcijos) – keičiama tik duomenų paruošimo pakopa, ALU branduolys lieka nepakitęs;
- ✓ pajungiant naujus duomenų kelius, nesunku modifikuoti duomenų paruošimo pakopą;
- ✓ lengvesnis ALU valdymo signalų projektavimas, nes projektuotojas žino valdymo signalų semantiką;
- ✓ per duomenų paruošimo pakopą ALU branduolį dalinasi visos instrukcijos, todėl reikia mažiau grandžių nei naudojant aparatūros aprašymo projektavimą;
- ✓ šia architektūra pagrįstą ALU nesunku realizuoti srautinėje (konvejerinėje) schemoje;
- ✓ tai šakota grandinė, todėl ją galima efektyviai ištestuoti.

2.2.2.4 Eksperimento rezultatai

Remiantis pasiūlyta architektūra ir naudojant *Verilog* aparatūros aprašymo kalbą, buvo aprašytas su PIC16C57 instrukcijų rinkiniu suderinamas procesoriaus ALU. ALU buvo susintezuotas *Synplicity Synpilify 7.0.2* sintezatoriumi, *Actel ProASIC* technologijos pagrindu. 12 lentelėje palyginami skirtingo projektavimo realizavimo rezultatai. Projektuose, paremtuose dviejų pakopų architektūra vartojama mažiau grandžių (tuo pačiu mažesnis užimamas plotas). Taip pat tenkinamas kritinių sričių vėlinimo reikalavimas.

12 lentelė. Projektavimo rezultatų palyginimas

Su AAK projektuojama schema		***Plotas (grandys)***	***Kritiniai keliai, ns***
Paprastas projektavimas (sumatorius su ištisine pernaša)		382	5,068
Dviejų pakopų architektūra	Sumatorius su ištisine pernaša, paprastas tiesinis projektavimas	212	5,780
	Sumatorius su paspartinta pernaša, paprastas tiesinis projektavimas	169	4,098
	Sumatorius su paspartinta pernaša, kombinacinis bitinių sekcijų projektavimas	135	3,820
	Sumatorius su paspartinta pernaša, kombinacinis bitinių sekcijų projektavimas, konvejerinis ALU	151	2,470

2.2.2.5 Apibendrinimas

Visų pirma efektyvu daryti gerai suprojektuotus ALU. Antra, sutrumpinamas projekto pertvarkymo laikas pasikeitus funkcijų reikalavimui, gaunama lanksti architektūra, sutalpinant galimas ALU funkcijų variacijas. Galiausiai tai pritaikoma ne tik į aparatūros aprašymo kalbas orientuotame projektavime, bet ir tradiciniame schematiniame.

2.2.3 13,3ns dvigubo tikslumo slankaus kablelio ALU su daugintuvu

Slankaus kablelio aritmetiniam loginiam blokui (ALU) sukurtos vieno bito išankstinio postūmio prieš išlygiavimo postūmį, normalizacijos su galimu „1“ bitu priekyje bei išankstinio apvalinimo metodikos. Papildomai, slankaus kablelio daugintuvui sukurtos pernašos išrinkimo sudėties bei išankstinio apvalinimo metodikos. Suprojektuota ir ALU bei daugintuvui pritaikyta triukšmams pakanti (TP) išankstinio įkrovimo grandis. Šios metodikos sumažino vėlinimą kritinėse grandyse 24%. Kiekviena grandis buvo pagaminta 0,3µm 2,5V keturių sluoksnių metalo CMOS technologijos pagrindu ir pasiekė dviejų taktų vėlinimą esant 150MHz.

Mokslinės ir inžinerinės taikomosios programos reikalauja išskirtinai didelio tikslumo slankaus kablelio operacijų, kas savo ruožtu reikalauja didelės spartos slankaus kablelio ALU bei daugintuvų, tam, kad būtų sutrumpintas vykdymo laikas. Paskutiniais metais pristatyta nemažai didelės spartos slankaus kablelio operacijų vykdymo blokų.

Suprojektuoti slankaus kablelio ALU ir daugintuvai kurių kiekvienas gali atlikti veiksmą per 13,3ns. Tiek ALU, tiek daugintuvas gali atskirai išduoti rezultatą per vieno ciklo konvejerinį žingsnį, pasiekdami maksimalų 300MFLOPS dažnį esant 150MHz dažniui. Blokai pilnai suderinami su IEEE dvejetainės slankaus kablelio aritmetikos standartu.

ALU atlieka sudėties, atimties, palyginimo, pavertimo į mažesnio/didesnio tikslumo slankaus kablelio reikšmę, vertimą iš/į slankaus kablelio į/iš sveiką skaičių instrukcijas tiek dvigubo, tiek paprasto tikslumo operandams. ALU gali išduoti denormalizuotą skaičių be papildomo ciklo.

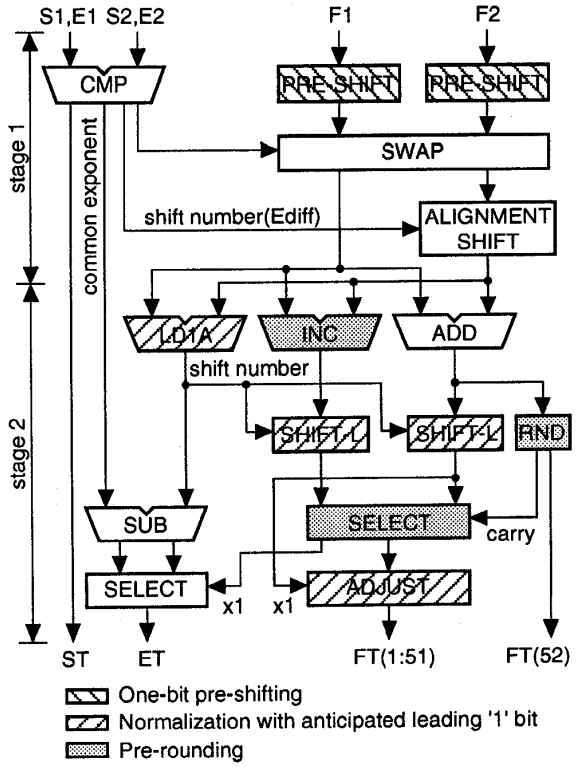
Daugintuvas atlieka slankaus kablelio daugybą tiek su dvigubo, tiek su paprasto tikslumo operandais ir sveikų skaičių daugybą su paprasto tikslumo operandais. Daugintuvas negali išduoti denormalizuoto skaičiaus, tačiau, vietoje to, gali sugeneruoti nulį su tiksliu ženklu ir taip išvengti spartos praradimo dėl vidinės pertraukties.

Tam, kad būtų įgyvendintas 13,3ns vykdymo laikas, abu blokai buvo suprojektuoti panaudojant kelias naujas aritmetines ir grandinių metodikas bei pažangiausią silicio technologiją.

2.2.3.1 Slankaus kablelio ALU architektūra

Slankaus kablelio ALU blokinė diagrama pavaizduota 9 paveiksle. Tai dviejų pakopų konvejerizuotas mechanizmas. Pirmoje pakopoje išrenkamas didesnio operando laipsnio rodiklis kaip bendras rodiklis, o operando su mažesniu rodikliu trupmenos ženklas pastumiamas dešinėn išlyginamojo postūmio registre. Antroje pakopoje atliekamos: operando su didesniu rodikliu trupmenos ir dešinėn pastumtos trupmenos sudėtis/atimtis, normalizacija, IEEE apvalinimas, bendro rodiklio korekcija.

ALU naudojamos trys aritmetinės metodikos. *Pirmoji* – vieno bito išankstinis postūmis abiejose trupmenose efektyvios sudėties atvejais. Ši metodika palengvina apvalinimo procesą. *Antroji* – normalizacija su galimu „1“ bitu priekyje sudėties ar atimties rezultatuose. Šis normalizacijos procesas yra greitas, net jei ir spėjamas bitas yra klaidingas, nes klaidingai pastumta trupmena gali būti pataisoma paprastu vieno bito postūmiu dešinėn. *Trečioji* panaudota metodika – išankstinis suapvalinimas. Jis paruošia visus galimus suapvalintus rezultatus lygiagrečiai su išlygiuotų trupmenų sudėtimi ar atimtimi bei išrenka teisingą šių operacijų rezultatą su „1“ bitu priekyje. Naudojant šią metodiką, apvalinimo procesas pagreitinamas 51%.



9 pav. *Slankaus kablelio ALU blokinė diagrama*

2.2.3.2 Vieno bito išankstinis postūmis

Kai atliekama efektyvioji sudėtis, abi operandų dalys yra stumiamos dešinėn, tada operando su mažesniu rodikliu pastumta dalis dar karta stumiama dešinėn, per tiek, kiek skiriasi operandų rodiklis (*Ediff*). Išlygintų dalių sudėties rezultatas yra kažkur tarp 0,1 ir 1,111... ir gali viršyti IEEE formato ilgį. Jei būtina, atliekamas normalizacijos postūmis kairėn arba apvalinimas.

Kai atliekama efektyvi atimtis operando su mažesniu rodikliu dalis stumiama dešinėn per *Ediff* rezultato dydį. Jei *Ediff*=0 arba 1, sulygiuotų dalių atimties rezultatas yra mažiau arba lygus 1, taigi būtina atlikti didelį normalizacijos postūmį. Kaip bebūtų, normalizuotas rezultatas jau yra suderinamas su IEEE formato ilgiu, taigi apvalinimas nebeatliekamas. Jei *Ediff*<1, atimties rezultatas yra tarp 0,1 ir 1,111... ir gali viršyti IEEE formato ilgį. Tokiais atvejais, jei būtina, atliekamas normalizacijos postūmis kairėn bei apvalinimas.

2.2.3.3 Normalizacija su galimu „1“ priekyje

Normalizacijos procesas susideda iš šių žingsnių: 1) priekyje esančio „1“ bito nuspėjimas iš sudėties ar atimties rezultato; 2) postūmio kontrolės signalo generavimas; 3) esant postūmio

valdymo signalui atlikti sudėties ar atimties rezultato postūmį kairėn; 4) jei klaidingas spėjamas „1“ bitas, atliekamas vieno bito išlyginantis postūmis dešinėn.

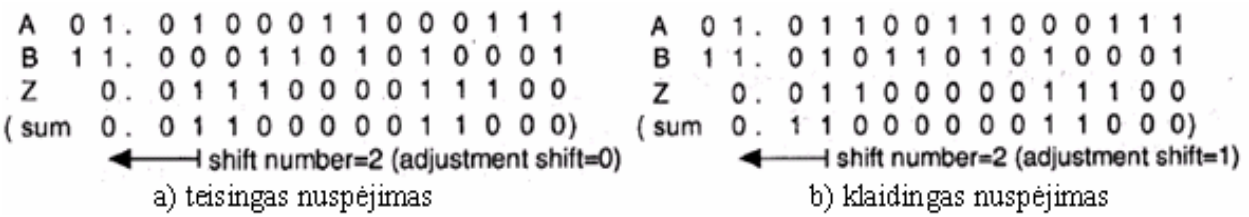
Toliau aprašomas priekyje esančio „1“ bito nuspėjimo algoritmas. Priekyje esančio „1“ bito nuspėjimo signalas Z aprašomas sekančiai:

$$Z = z_0, z_1 z_2 ... z_i ... z_{52} \qquad (1)$$

i-tasis signalo Z bitas aprašomas:

$$z_i = (a_{i-1} \wedge b_{i-1}) \wedge (a_i \mid b_i) \qquad (2)$$

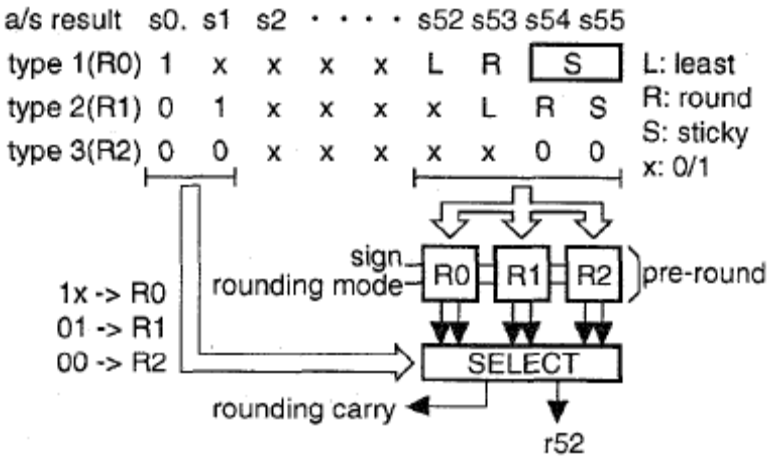
a_i ir b_i yra i-tieji sudedamų operandų dalių bitai ($0 \le i \le 52$). (2) lygybėje simbolis „^“ reiškia išskirties disjunkciją, o simbolis „|“ – loginę ARBA operaciją. Išgaunant signalą Z daugiausia sutinkamas 2 ventilių vėlinimas (2 išskirties disjunkcijos), kas yra mažiau nei 7-8 ventilių vėlinimas kuris būtinas 55 bitų sumatoriui su paspartinta pernaša. Z signalo priekyje einančio „1“ bito pozicija yra lygi arba tik vienu bitu žemesnė nei sumos (atimties) rezultato. Jei nuspėtas bitas klaidingas – normalizacijos postūmis taip pat klaidingas vienu bitu ir gali būti pataisytas paprastu postūmiu dešinėn. Jei nuspėtas bitas teisingas postūmio atlikti nebereikia. Toliau, 10 paveiksle, pateikti spėjimo pavyzdžiai:



10 pav. *Priekyje einančio „1“ bito nuspėjimas*

2.2.3.4 Išankstinis apvalinimas

11 paveiksle vaizduojama išankstinio apvalinimo schema. ALU skaičiavimų išankstinio apvalinimo procesas susideda iš 4 žingsnių.



11 pav. *Išankstinio apvalinimo schema*

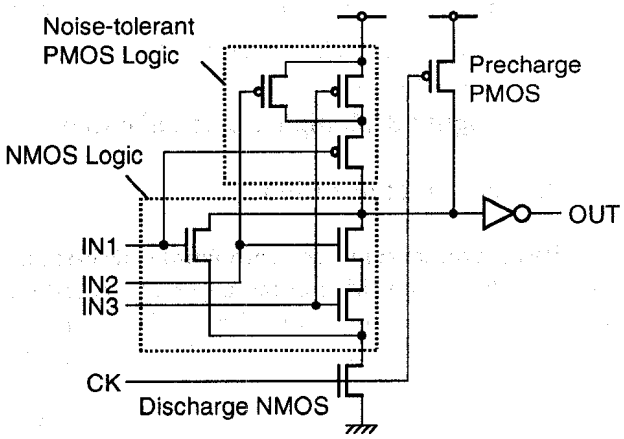
Pirmame žingsnyje 52-oje dešimtainėje skiltyje sudėties (atimties) rezultatas padidinamas vienetu. Padidinimas atliekamas lygiagrečiai su sudėtimi (atimtimi), o rezultatas ignoruojamas jei po apvalinimo nebūna pernašos. *Antrame* žingsnyje atliekami trys nepriklausomi apvalinimai - visoms galimoms priekyje einančio „1“ bito pozicijoms (tipas 1, tipas 2 ir tipas 3). Tipai 1, 2 ir 3 reiškia galimas „1“ bito pozicijas: vienu bitu kairiau, vienu dešiniau ir dviem ar daugiau bitų dešiniau nuo dešimtainio skaičiaus kablelio. Sudėties (atimties) rezultato 52..55 bitai, ženklo bitas ir apvalinimo režimo signalai yra naudojami suskaičiuoti trims apvalinimo pernašoms bei trims žemiausių skilčių bitams išankstinio apvalinimo įtaisų R0, R1 ir R2 rezultate. *Trečiame* žingsnyje pagal sudėties (atimties) rezultato aukščiausių dviejų skilčių reikšmes išrenkamas teisingas iš anksto suapvalintas rezultatas. Jei bitai yra „10“ arba „11“ – naudojamas R1 rezultatas. Jei „01“ naudojamas R1 įtaiso rezultatas, kitu atveju naudojamas R2 rezultatas. *Ketvirtame* žingsnyje, panaudojant išrinktą pernašą parenkamas arba pirmame žingsnyje suskaičiuotas inkrementuotas rezultatas, arba sudėties (atimties) rezultatas.

Sudėties (atimties) rezultato dviejų aukščiausių bitų suskaičiavimas bei po to sekantis apvalinimo pernašos išrinkimas yra vienos labiausiai kritinių dalių, todėl normalizacijos postūmio registrai specialiai perkeliami iš šios dalies. Tokiu būdu jie gali funkcionuoti lygiagrečiai, kartu su apvalinimo pernašos skaičiavimu.

2.2.3.5 Grandinės charakteristikos

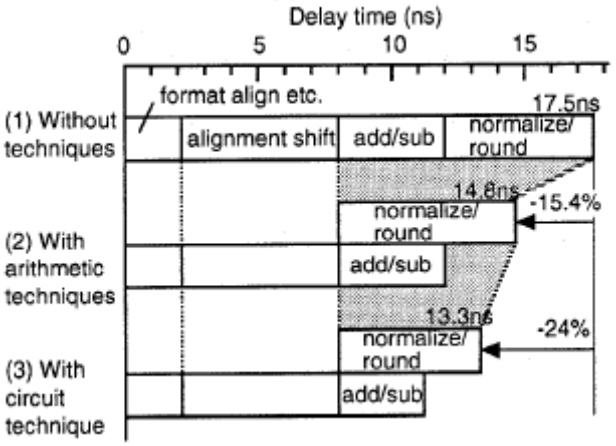
Kritinėms ALU bei daugintuvo dalims sukurta ir pritaikyta triukšmams pakanti (TP) grandinė, taip pat didelės spartos ir triukšmams pakanti CMOS grandinė. 12 paveiksle pateikta triukšmams pakanti grandinė. Grandinė turi triukšmams pakančią PMOS logiką kuri teikia didelį triukšmų imunitetą. TP grandinė užkraunama kai taktas yra žemo lygio, skaičiuojama kai taktas

yra aukšto lygio. Grandinės vėlinimas apsprendžiamas NMOS logikos. TP grandinė turi 30-36% vėlinimo privalumą prieš įprastines CMOS grandines. Paspartintos pernašos sumatoriams ir priekyje einančio „1“ bito nuspėjimo įtaisui suprojektuoti trys TP grandinių tipai tam, kad būtų paspartintos kritinės sritys.



12 pav. *TP grandinės blokinė diagrama*

13 paveiksle pateiktas slankaus kablelio ALU vėlinimas. Kiekvienas vėlinimas skaičiuotas grandinės simuliacijoje. Naudojant aukščiau aprašytas aritmetines metodologijas, vėlinimo laikas kritinėse srityse sumažėjo 15,4%. Be to, naudojant TP grandinę vėlinimo laikas sudėties (atimties) pernašos sklidimas ir priekyje einančio „1“ nuspėjimas normalizacijoje taip pat sumažėja, bendras vėlinimo laikas sumažinamas 24%.

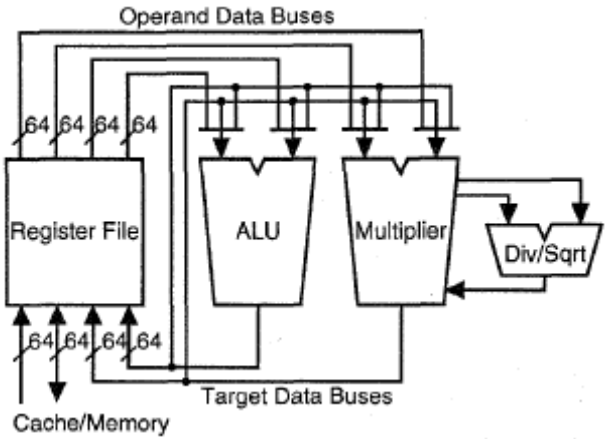


13 pav. *Vėlinimo skaičiavimas ALU*

2.2.3.6 Slankaus kablelio įtaisas

Slankaus kablelio ALU ir daugintuvas pagaminti 0,3µm keturių sluoksnių CMOS technologija. 14 paveiksle parodyta viso slankaus kablelio įtaiso blokinė diagrama. Slankaus

kablelio įtaisas susideda iš 4 didesnių dalių: 128 x 64 bitų registrų failo, ALU, daugintuvo ir dalybos bei kvadratinės šaknies įtaiso. Registrų failas turi keturis skaitymo ir keturis rašymo prievadus, kas leidžia lygiagretų užkrovimo, ALU ir dauginimo operacijų vykdymą.



14 pav. *Slankaus kablelio įtaiso blokinė diagrama*

2.2.3.7 Apibendrinimas

Slankaus kablelio ALU įtaisui sukurtos vieno bito išankstinio postūmio prieš išlyginantį postūmį, normalizacijos su nuspėjamu priekyje einančiu „1“ bitu bei išankstinio suapvalinimo metodikos. Slankaus kablelio daugintuvui sukurtus pernašos išrinkimo sudėties ir išankstinio apvalinimo metodikos. Tam, kad būtų paspartintos ALU bei daugintuvo kritinės sritys buvo sukurta didelės spartos ir didelio atsparumo triukšmams CMOS grandinė. Šios metodikos sumažino kritinių sričių vėlinimą 24%. Kiekvienas įtaisas pagamintas 0,3µm keturių sluoksnių CMOS technologija ir pasiekia dviejų ciklų vėlimą esant 150 MHz dažniui.

2.2.4 Kitos ALU architektūros

2.2 skyriuje išnagrinėti tik kelios ALU pritaikymo architektūros. ALU yra svarbiausias matematinius veiksmus atliekančių lustų modulis. Šioje srityje atliekami tyrimai ir daromos įvairios rekomendacijos. ALU bendras vėlimas įtakojamas jo ventilių vėlinimo, todėl Ozawa et. al. [17] siūlo galimai sumažinti patį ALU, tai išsprendžiama panaudojant asinchroninį kaskadinį jungimą. Šis sprendimas neigiamai atsiliepia bendram plotui.

Kitas dažnai sutinkamas sprendimas – ALU asinchroninis valdymas [17, 18, 19]. Šio sprendimo pasėkoje tipinės loginės operacijos atliekamos greičiau, bei vartojama mažiau energijos, tačiau tam tikrais atvejais operacija gali būti atliekama ilgiau.

ALU schema talpinama mikroprocesorių lustuose, palaipsniui ji pati įgauna procesoriaus sandaros bruožų. Parashar, Gurumurthi ir Sivasubramaniam [20] aprašo ALU architektūra, kurioje yra instrukcijų pakartotinio panaudojimo buferis talpinantis instrukcijas, operandus ir gautus rezultatus. Panaudojus procesoriuje ALU 1024 įrašų buferį gautas 23% bendras sistemos paspartėjimas. Didžiausia įtaka šis sprendimas turi ALU architektūrose su dideliu resursų dalinimosi laipsniu.

Dar vienas būdas paspartinti ALU darbą – įtaisų lygiagretinimas. Gali būti lygiagretinami ALU komponentai [5] arba ištisa jų grupė [21, 22].

2.3 Apžvalgos išvados

ALU operacijų apžvalgos rezultatai ir išvados

- ✓ Peržvelgtos keturių ALU atliekamos operacijos: L4C383, 74HC/HCT181, CD40181BMS, PDSP1601/PDSP1601A;
- ✓ Pateiktas pagrindinių ALU savybių palyginimas;

Apžvelgtų ALU savybės apibendrintos 13 lentelėje. Pateikiamas ALU tipas, atliekamų operacijų skaičius bei privalumai/trūkumai.

13 lentelė. Nagrinėtų ALU architektūrų apžvalgos suvestinė

Pavadinimas	***Tipas***	***Duomenų plotis, bitais***	***Op. skaičius***	***Pranašumai***	***Trūkumai***
LC4C383	Kaskadinis ALU	16	32	Didelis greitis, mažas energijos suvartojimas	Didelė panašių instrukcijų įvairovė, greitaveikai tai gali būti privalumas, ploto atžvilgiu - trūkumas
74HC/HCT181	Lygiagretus ALU	4	32	Didelės spartos operacijos su ilgais žodžiais, didelė greitaveika. Plati atliekamų operacijų aibė	Didelis užimamas plotas
CD40181BMS	Lygiagretus ALU	4	32	Komparatoriaus išėjimas, plati atliekamų operacijų aibė	Didelis maksimalus energijos suvartojimas, mažas veikimo dažnis
PDSP1601(A)	Kaskadinis, ciklinis ALU	16	32	Turi integruotą ciklinio postūmio reg., 4x16 bitų registrų failas.	Specifinio pritaikymo. Iš 32 instrukcijų 9 yra konstantos arba ALU išėjimo nulinimas

Analizuoti ALU yra specializuoti, nors ir visi atlieka 32 įvairias operacijas, renkantis pritaikymui reikia atidžiai išstudijuoti jų aibę.

Architektūrinių sprendimų apžvalgos išvados

- ✓ Peržvelgtos trys ALU architektūros: 1) ALU architektūra, skirta *ASIC*;
  2) Dviejų pakopų ALU architektūra;
  3) 13,3ns dvigubo tikslumo slankaus kablelio ALU su daugintuvu architektūra.
- ✓ Didesnio laipsnio resursų dalinimasis neveda link sumažėjusio energijos išsklaidymo;
- ✓ Daugybos pakeitimas postūmio ir sudėties operacijomis neveda link sumažėjusio energijos išsklaidymo;
- ✓ Panaudojant dviejų pakopų architektūra vartojama mažiau grandžių (tuo pačiu mažesnis užimamas plotas);
- ✓ Panaudojant dviejų pakopų architektūrą gaunami paprasti ir modifikuojami moduliai;
- ✓ Panaudojant triukšmams pakančias grandines sumažėja vėlinimas grandyse;
- ✓ Išankstinio rezultatų nuspėjimo logika su greita blogo spėjimo korekcija gali būti operacijos paspartinimo veiksnys.

3 Metodai

3.1 Siūlomos ALU architektūros analizė

Remiantis 2.1 ir 2.2 atlikta apžvalga pasirenkama operacijų aibė ir projektuojamas ALU įtaisas. Atsižvelgiama į 2.3 dalyje padarytas išvadas ir naudotus metodus.

3.2 Operacijos

Remiantis 2.1 dalyje apžvelgtais ALU komponentais sudaryta dažnai naudojamų operacijų aibė. Renkantis operacijas taip pat numatyta, kad atliekant modeliavimą bus galima atlikti sudėtingas funkcijas per tarpines operacijas, pavyzdžiui *XOR* išreikšti per *AND*, *NOT* ir *OR*. ALU operacijų aibę galima aprašyti sekančiai (14 lentelėje atskirų operacijų paaiškinimas):

```
Opera={op_lda,op_ldm,op_ldi,op_ldii,op_and,op_or,op_add,op_sub,op_inc,op_dec,
op_not,op_shfl,op_shfr,op_nop};
```

14 lentelė. ALU branduolio atliekamos operacijos

Komanda	***Kodas***	***Operacija***	***Aprašymas***
LDA A,rrr	op_lda	A <- R[rrr]	Į akumuliatorių užkraunama reikšmė iš registro įėjimo
LDM A,aaaaaa	op_ldm	A <- M[aaa aaa]	Į akumuliatorių užkraunama reikšmė iš magistralės įėjimo
LDI A,iiii	op_ldi	A <- xxxx iiii	Į akumuliatoriaus paskutines skiltis užkraunamos L/2 paskutinės registro įėjimo skiltys
LDII A,iiii	op_ldii	A <- iiii xxxx	Į akumuliatoriaus paskutines skiltis užkraunamos L/2 aukštesnės registro įėjimo skiltys
AND A,rrr	op_and	A <- M[aaa aaa] and R[rrr]	Loginė AND operacija, operandai: atminties ir registro įėjimas
OR A,rrr	op_or	A <- M[aaa aaa] or R[rrr]	Loginė OR operacija, operandai: atminties ir registro įėjimas
ADD A,rrr	op_add	A <- M[aaa aaa] + R[rrr]	Sudėtis, operandai: atminties ir registro įėjimas
SUB A,rrr	op_sub	A <- M[aaa aaa] – R[rrr]	Atmintis, operandai: atminties ir registro įėjimas
NOT A	op_not	A <- not A	Akumuliatoriaus reikšmės inversija
INC A	op_inc	A <- A + 1	Akumuliatoriaus reikšmės inkrementavimas
DEC A	op_dec	A <- A - 1	Akumuliatoriaus reikšmės dekrementavimas
SHF left	op_shfl	A <- M[aaa aaa] << 1	Atminties reikšmės postūmis į kairę
SHF right	op_shfr	A <- M[aaa aaa] >> 1	Atminties reikšmės postūmis į dešinę
NOP	op_nop	no operation	Jokios operacijos

Siūlomas ALU branduolys turi kelias instrukcijas, kurių nebuvo nagrinėtuose pavyzdžiuose, pvz.: *op_lda, op_ldm, op_ldi, op_ldii*. Šios instrukcijos yra iš naudojamo turėto branduolio kodo, tuo pačiu priartina branduolio pasirinkimą prie realių sąlygų, kai tenka rinktis perteklinių savybių turinčius universalius modulius.

3.3 Architektūra

3.3.1 ALU kaip IP komponento analizė

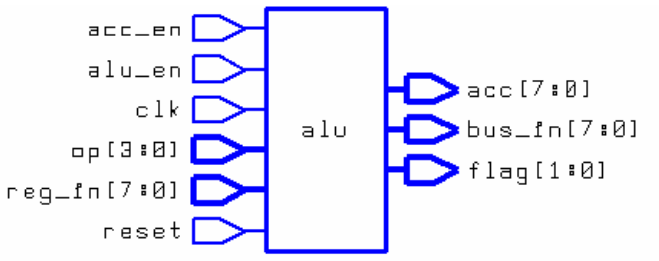
Darbo tikslas – ALU komponento sukūrimas pagal išanalizuotus pavyzdžius ir ištestavimas. IP komponentu pasirinktas universalių savybių turintis ALU komponentas.

SystemC aparatūros aprašymo kalba realizuotas modeliuojamas ir sintezuojamas ALU branduolys su L bitų (kai L gali būti bet koks lyginis skaičius 2..128, dažniausiai naudojami standartiniai pločiai: 8, 16, 32, 64) operandais atliekantis 14 lentelėje pateiktus veiksmus.

15 paveiksle pateikiamas ALU branduolio kaip „juodos dėžės“ paveikslas. 15 lentelėje pateikti įtaiso kojų aprašymai su paaiškinimu.

15 lentelė. ALU branduolio kojos

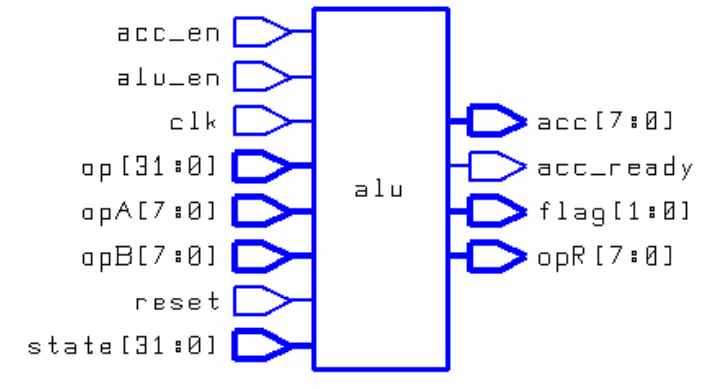
Koja	Tipas	Plotis	Paskirtis
acc_en	Įėjimas	1	ALU akumuliatoriaus leidimas išduoti saugomą reikšmę
alu_en	Įėjimas	1	Viso ALU leidimo dirbti signalas
clk	Įėjimas	1	Sinchro signalas
op	Įėjimas	4	Operacijos kodas iš 2 lentelėje pateiktos aibės (stulpelis „kodas“)
reg_in	Įėjimas	L	L pločio registro operando įėjimas, vienas iš ALU operandų
reset	Įėjimas	1	Reset signalo įėjimas
acc	Išėjimas	L	ALU akumuliatorius – saugomas atlikto veiksmo rezultatas
bus_in	Įėjimas/ Išėjimas	L	L pločio atminties operando įėjimas/išėjimas, vienas iš ALU operandų, gali būti panaudotas kaip išėjimas atgal į atmintį
flag	Išėjimas	2	2 bitų vėliavėlių išėjimas, 0 bitas – zero, 1 bitas – ženklo skiltis



15 pav. *ALU branduolys*

Sujungus ALU su jį naudojančia sistema, atsirado papildomų valdymo kojų. Gauto ALU „juoda dėžė“ parodyta 16 paveiksle, o kojų aprašymas pateikiamas 16 lentelėje. ALU

funkcionalumas nesikeičia, pervadintos operandų reikšmių padavimo kojos, kad nebūtų analogijos su valdančia logika, taip pat dubliuojama ALU rezultato ACC reikšmė (operandu opR) – taip išlaikoma IP komponento logika, nekeičiant kodo. Papildomai panaudojamas būsenos įėjimas *state*, jis ALU funkcionalumo taip pat neįtakoja, naudojamas jautrumo sąraše įtaiso sužadinimui, taip ALU tampa valdomas ne nuo *CLK* signalo, o būsena.



16 pav. *ALU naudojamas kaip IP*

16 lentelė. ALU kaip IP kojos

Koja	***Tipas***	***Plotis***	***Paskirtis***
Acc_en	Įėjimas	1	ALU akumuliatoriaus leidimas išduoti saugomą reikšmę
alu_en	Įėjimas	1	Viso ALU leidimo dirbti signalas
Clk	Įėjimas	1	Sinchro signalas
Op	Įėjimas	4	Operacijos kodas iš 2 lentelėje pateiktos aibės (stulpelis „kodas“)
opB	Įėjimas	L	L pločio registro operando įėjimas, vienas iš ALU operandų, prieš tai vadinosi *Reg_in*
reset	Įėjimas	1	Reset signalo įėjimas
state	Įėjimas	2	Įėjimas iš valdymo logikos, sužadina ALU vietoj CLK signalo. Taip ALU kviečiamas tik prireikus
Acc	Išėjimas	L	ALU akumuliatorius – saugomas atlikto veiksmo rezultatas
opA	Įėjimas/ Išėjimas	L	L pločio atminties operando įėjimas/išėjimas, vienas iš ALU operandų, gali būti panaudotas kaip išėjimas atgal į atmintį. Prieš tai vadinosi *Bus_in*
Flag	Išėjimas	2	2 bitų vėliavėlių išėjimas, 0 bitas – zero, 1 bitas – ženklo skiltis
opR	Išėjimas	L	ALU akumuliatorius – saugomas atlikto veiksmo rezultatas, dubliuoja ACC išėjimą, grąžina reikšmę valdymo logikai

3.3.2 Sistemos analizė

SystemC realizuotas modeliuojamas ir sintezuojamas *Control Unit* (CU) – valdymo įtaiso branduolys su L bitų (kai L gali būti bet koks lyginis skaičius 2..128, dažniausiai naudojami standartiniai pločiai: 8, 16, 32, 64) operandais atliekantis 17 lentelėje pateiktus veiksmus. Savo

funkcijų realizavimui CU naudoja 3.3.1 dalyje aprašytą ir pritaikytą ALU komponentą. Lentelėje operandas „A“ atitinka CU įėjimą iš atminties, „B“ – iš registrų failo, „CI“ – pernaša įėjime. CU operacijų aibę galima aprašyti sekančiai:

```
Insta={n_aab,n_axb,n_aob,asnb,nasb,naab,aanb,asbsc,nasbsc,asnbsc,adsc,bdsc};
```

17 lentelė. CU atliekamos operacijos

Kodas	***Operacija***	***Aprašymas***
n_aab	!(A AND B)	Loginė AND operacija su operandais, rezultatas invertuojamas.
n_axb	!(A XOR B)	Loginė XOR operacija su operandais, rezultatas invertuojamas. ALU neturi vienos XOR operacijos
n_aob	!(A OR B)	Loginė OR operacija su operandais, rezultatas invertuojamas.
asnb	A + !B	Registro reikšmė invertuojama, atliekamas sumavimas su operandu „A“
nasb	!A + B	Atminties reikšmė invertuojama, atliekamas sumavimas su operandu „B“
naab	!A AND B	Atminties reikšmė invertuojama, atliekama loginė operacija AND su operandu „B“
aanb	A AND !B	Registro reikšmė invertuojama, atliekama loginė operacija AND su operandu „A“
asbsc	A + B + CI	Sumuojami operandai ir pernaša
nasbsc	!A + B + CI	Sumuojama invertuota operando „A“ reikšmė bei „B“ ir pernaša
asnbsc	A + !B + CI	Sumuojama invertuota operando „B“ reikšmė bei „A“ ir pernaša
adsc	A - 1 + CI	Mažinama vienetu atminties reikšmė, pridedama pernaša
bdsc	B - 1 + CI	Mažinama vienetu registro reikšmė, pridedama pernaša

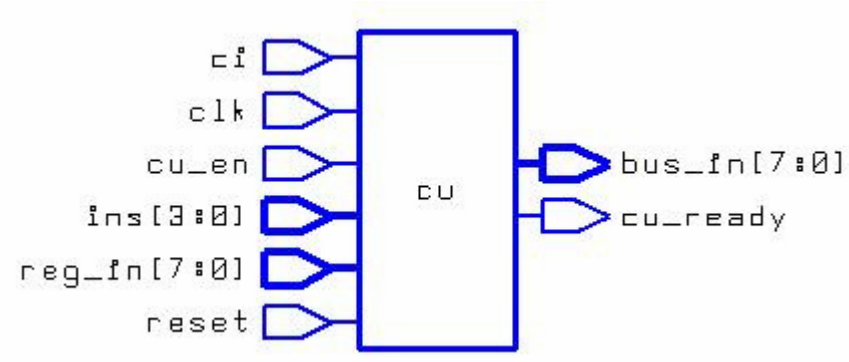
Kadangi CU naudoja ALU savo reikmėms, atliekant vieną CU operaciją gali būti atliekamas didelis skaičius ALU operacijų. Pagal pateiktą 17 lentelę sudaryta 18 lentelė, kurioje pateikiama ALU panaudojimo schema. ALU valdomas būsenų automatu, vienoje būsenoje atliekama operacija, tada pereinama prie kitos būsenos arba atiduodamas galutinis rezultatas. Naudojami sutrumpinimai: *A* – operandas su reikšme iš atminties, *B* – operandas su reikšme iš registrų failo, *R* – ALU operacijos rezultatas, *bus* – reikšmė magistralėje (atmintyje).

18 lentelė. ALU panaudojimas

CU operacija	***ALU operacijų***	***CU būsenų seka***
n_aab	3	**[1]** R = A and B **[2]** not R **[3]** bus = R
n_axb	7	**[1]** not A **[2]** not B **[3]** A and !B **[4]** !A and B **[5]** R = (3) or (4) **[6]** not R **[7]** bus = R
n_aob	3	**[1]** R = A or B **[2]** not R **[3]** bus = R
asnb	3	**[1]** R = not B **[2]** R + A **[3]** bus = R
nasb	3	**[1]** R = not A **[2]** R + B **[3]** bus = R
naab	3	**[1]** R = not A **[2]** R and B **[3]** bus = R
aanb	3	**[1]** R = not B **[2]** R and A **[3]** bus = R
asbsc	3	**[1]** R = A + B **[2]** R = R + ci **[3]** bus = R
nasbsc	4	**[1]** not A **[2]** R = !A + B **[3]** R = R + ci **[4]** bus = R

asnbsc	4	**[1]** not B **[2]** R = A + !B **[3]** R = R + ci **[4]** bus = R
adsc	3	**[1]** R = dec(a) **[2]** R = R + ci **[3]** bus = R
bdsc	3	**[1]** R = dec(b) **[2]** R = R + ci **[3]** bus = R

17 paveiksle parodytas CU kaip "juoda dėžė". CU kojų aprašymas pateikiamas 19 lentelėje.

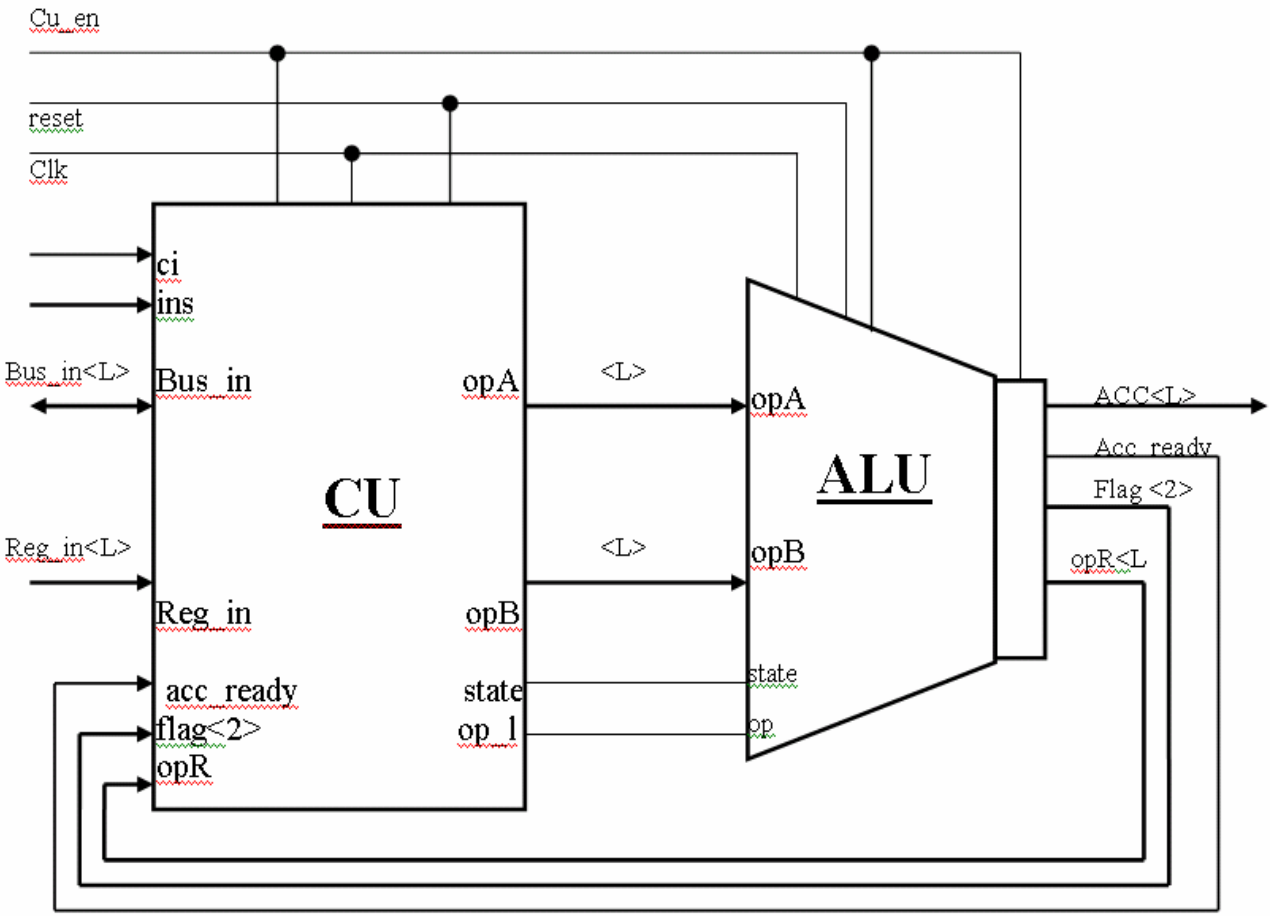


17 pav. *CU branduolys*

19 lentelė. CU komponento kojos

Koja	***Tipas***	***Plotis***	***Paskirtis***
ci	Įėjimas	1	Pernaša įėjime. Perduodama iš kito įrenginio (grąžinama iš ALU)
clk	Įėjimas	1	Sinchro signalas
cu_en	Įėjimas	1	Viso CU leidimo dirbti signalas
ins	Įėjimas	4	Operacijos kodas iš 4 lentelėje pateiktos aibės (stulpelis „kodas“)
reg_in	Įėjimas	L	L pločio registro operando įėjimas, vienas iš CU operandų.
reset	Įėjimas	1	Reset signalo įėjimas
bus_in	Įėjimas/ Išėjimas	L	L pločio atminties operando įėjimas/išėjimas, vienas iš CU operandų, gali būti panaudotas kaip išėjimas atgal į atmintį, į jį atiduodamas CU rezultatas
cu_ready	Išėjimas	1	1 bito išėjimas, rodantis, kad CU turi rezultatą ir laukia kitos instrukcijos

Paveiksle 18 parodyta kaip atrodo bendras visos sistemos schematinis vaizdas su CU ir ALU (kaip IP) vidiniais signalais užtikrinančiais bendravimą tarp komponentų.



18 pav. *CU ir ALU branduolio schema*

3.3.3 Sistemos modifikacijos

Sukurtos šios CU-ALU architektūros modifikacijos ir modeliai:

- ✓ *Bazinis* – taip vadinamas standartinis, 3.3.2 dalyje aprašytas sistemos variantas;
- ✓ *Bazinis, invertavimas realizuotas CU* – pagal 18 lentelėje pateiktą CU operacijų aprašymą, matoma, kad dažnai sutinkama NOT instrukcija. Ji iškeliama iš ALU ir realizuota CU;
- ✓ *XOR perkeltas į ALU* – pasirinktas ALU modulis neturi loginės *XOR* operacijos, ją galima išreikšti kitomis ALU operacijomis: *AND, NOT* ir *OR*. Šiame modelyje ALU papildomas *XOR* operacija;
- ✓ *XOR perkeltas į ALU, invertavimas į CU* – bendras ankstesnių dviejų architektūrų junginys. Imamas modelis, kuriame ALU papildytas *XOR* ir jame *NOT* perkeliamas iš ALU į CU;

- ✓ *Iš ALU pašalinamos visos nenaudojamos instrukcijos* – sistemoje panaudotas ALU savo savybėmis yra perteklinis, modeliuojamas variantas, kai ALU pilnai atitinka CU poreikius;
- ✓ *Bazinis, ALU papildytas daugybos operacija* – praplečiamas ALU ir jame realizuojama daugybos operacija. Pradiniame variante jos nebuvo visai. CU papildytas aukščiau dar neaprašyta operacija *amb* – operandas *A* dauginamas iš operando *B*;
- ✓ *Bazinis, daugyba išreikšta per ADD ir SHFL* – kaip ir *XOR* atveju, daugybą galima realizuoti panaudojant turimas paprastas ALU instrukcijas. Daugyba realizuota sumavimo ir postūmio kairėn operacijomis. Modelis pagal paskirtį analogiškas aukščiau sudarytam su pilna daugybos operacija, realizuojama ALU. CU papildytas aukščiau dar neaprašyta operacija *amb* – operandas *A* dauginamas iš operando *B*;
- ✓ *Bazinis, CU papildytas atimtimi* – pradinis CU modelis nenaudoja atimties operacijos, nors ji ALU ir yra. CU papildomas atimties operacija, neaprašyta aukščiau pateikiamame sistemos aprašyme. Nauja operacija *a_b* – iš operando *A* atimamas *B*;
- ✓ *Bazinis, atimtis realizuota per sudėtį ir invertavimą* – analogiškas aukščiau aprašytai architektūrai, atimtį realizuojant per *ADD*, *NOT* ir *INC* ALU instrukcijas. Iš ALU pašalinama nenaudojama *SUB* (atimties) instrukcija. CU papildomas atimties operacija, neaprašyta aukščiau pateikiamame sistemos aprašyme. Nauja operacija *a_b* – iš operando *A* atimamas *B*.

3.3.4 Būsenų automatas

CU atliekamos operacijos pateikiamos 18 lentelėje. CU visos operacijos yra sudėtinės. Naudojant būsenų automatą, keičiamas *state* signalas ir su kiekvienu ciklu į opR gaunamas ALU skaičiavimų rezultatas. 20 lentelėje pateiktas CU būsenų automato žingsnių skaičius kiekvienai sistemos modifikacijai.

20 lentelė. Modelių būsenų lentelė

Modelis	**n_aab**	**n_axb**	**n_aob**	**asnb**	**nasb**	**naab**	**aanb**	**asbsc**	**nasbsc**	**asnbsc**	**adsc**	**bdsc**	**amb**	**a_b**	**Σ**
Bazinis	3	7	3	3	3	3	3	3	4	4	3	3	-	-	42
Bazinis, invertavimas realizuotas CU	*2*	**4**	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	-	-	31
XOR perkelta į ALU	3	**3**	3	3	3	3	3	3	4	4	3	3	-	-	38
XOR perkelta į ALU, invertavimas į CU	*2*	**2**	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	-	-	**29**
Iš ALU pašalinamos visos	*2*	**2**	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	-	-	**29**

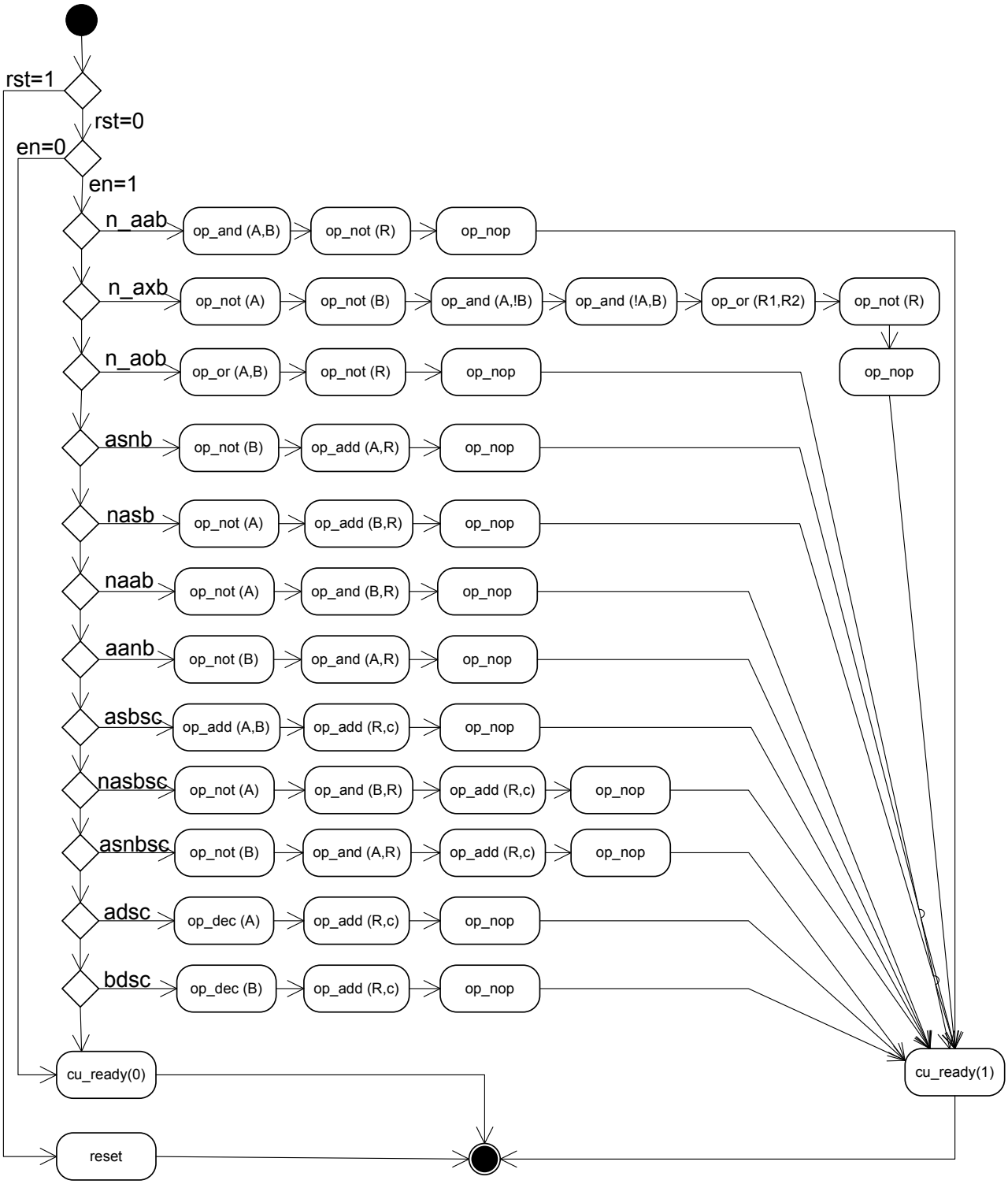
nenaudojamos instrukcijos															
Bazinis, ALU papildytas daugybos operacija	3	7	3	3	3	3	3	3	4	4	3	3	**2**	-	44
Bazinis, daugyba išreikšta per ADD ir SHFL	3	7	3	3	3	3	3	3	4	4	3	3	**18***	-	**60**
Bazinis, CU papildytas atimtimi	3	7	3	3	3	3	3	3	4	4	3	3	-	**2**	44
Bazinis, atimtis realizuota per sudėtį ir invertavimą	3	7	3	3	3	3	3	3	4	4	3	3	-	**4**	46

* - būsenų automato žingsnių skaičius ciklinio sumavimo operacijoje. Priklausomai nuo daugiklio L_i dvejetainių narių („0“ ar „1“ skaičiaus) skiriasi ciklišKai atliekamų žingsnių kiekis. Jei daugiklio L_i narys yra „0“ – atliekamas dauginamojo poslinkis kairėn (2 būsenos). Jei daugiklio L_i narys yra „1“ – atliekamas dauginamojo sumavimas prie sumos kaupiklio ir poslinkis kairėn (3 būsenos). Taigi, jei daugiklis turi vieną „1“ per visą ilgį *L* – sandaugai reikės 19 ciklų (18, jei visi „0“), jei turės *L* kiekį „1“ per visą savo ilgį, tai sandaugai reikės 26 ciklų.

Taigi, operacijos *amb* žingsnių skaičius apskaičiuojamas: $S_0 \cdot 2$ („0“ skaičius kart 2) + $S_1 \cdot 3$ („1“ skaičius kart 3) + 2.

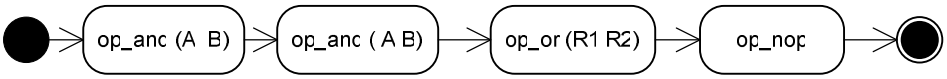
Dvi modelių grupės (po 2 variantus, viso 4 modeliai) papildomi operacijomis, kurių neturi kiti modeliai. 20 lentelėje tai matoma *amb* ir *a_b* stulpeliuose.

Paveiksle 19 pateikiama pradinio modelio *Bazinis* valdymo bloko (CU) būsenų automato schema. Kiekvienai valdymo bloko operacijai atlikti pereinamos parodytos ALU būsenos.



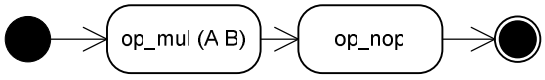
19 pav. *Bazinio modelio būsenų automatas*

Antrajame sukurtame modelyje *Bazinis, invertavimas realizuotas CU* – iš ALU į CU perkeliama invertavimo operacija. Tai daugiausia įtakos turi CU operacijos *n_axb* būsenoms. 20 paveiksle pateikiama šios operacijos būsenų automato schema po pakeitimų.



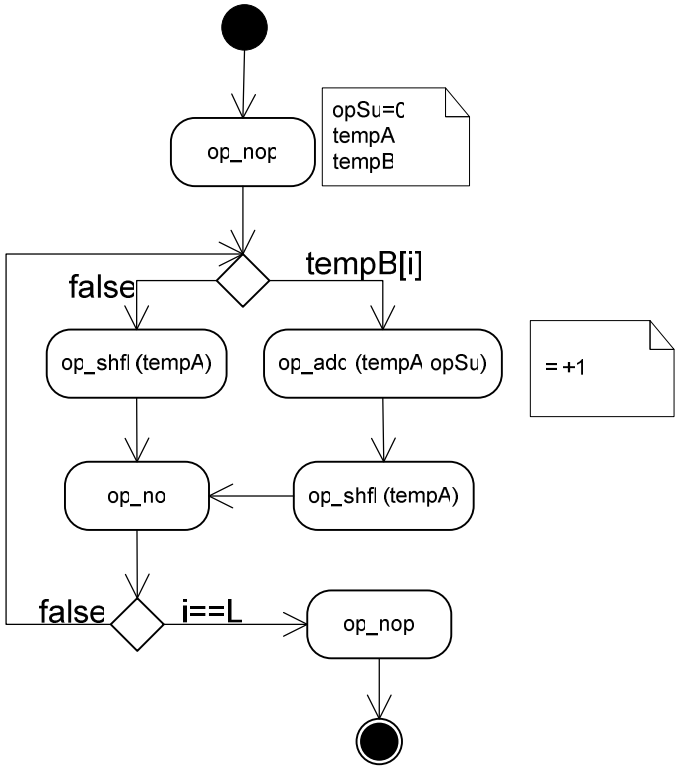
20 pav. *n_axb po NOT iškėlimo iš ALU*

Modelyje *Bazinis, ALU papildytas daugybos operacija* ALU papildomas vieno ciklo daugybos operacija. CU modulis taip pat papildomas nauja operacija *amb*. 19 paveiksle pavaizduotas CU būsenų automatas būtų papildytas šaka parodyta 21 paveiksle.



21 pav. *amb operacija su daugintuvu ALU*

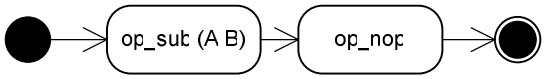
Modelyje *Bazinis, daugyba išreikšta per ADD ir SHFL* panaudojamos *Bazinio* varianto ALU turimos instrukcijos ir daugyba realizuojama per ciklines *ADD* bei *SHFL* instrukcijas. Būsenų automate atsiranda sudėtingesnė šaka, parodyta 22 paveiksle.



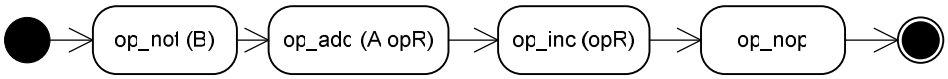
22 pav. *amb operacija išreikšta per ADD ir SHFL*

Bazinis modelis papildomas atimties operacija CU, nauja būsenų automato šaka parodyta 23 paveiksle. ALU turi atimties instrukciją, tačiau CU jos iki šiol nenaudojo. 24 paveiksle

parodomos būsenos, kai CU atimtis išreiškiama per sudėties (*ADD*) ir invertavimo (*NOT*) operacijas, tuo pačiu iš ALU šalinama tiesioginė vieno veiksmo atimtis (*SUB*).



23 pav. *Tiesioginės atimties šakos būsenos*



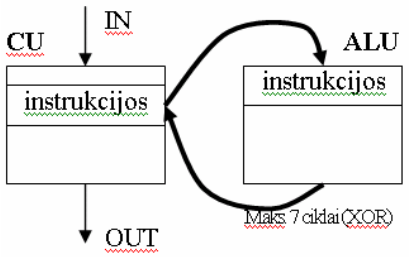
24 pav. *Atimtis išreikšta per ADD ir NOT*

3.3.5 Operacijų sudalinimas tarp CU ir ALU

3.3.3 dalyje pateikti sukurti ALU modeliai. Toliau pateikiamas smulkesnis jų aprašymas.

3.3.5.1 Bazinis

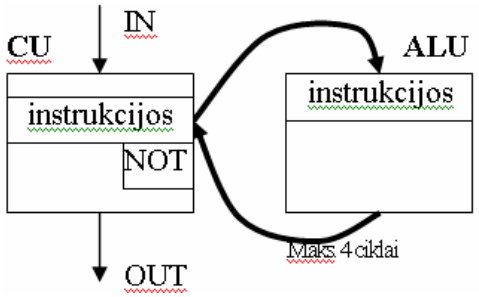
Taip vadinamas standartinis, 3.3.2 dalyje aprašytas sistemos variantas. ALU modulis atlieka 14 lentelėje aprašytas operacijas, CU – 17 lentelės operacijas. Schematiškai instrukcijų sudalinimas pavaizduotas 25 paveiksle.



25 pav. ***Bazinis modelis***

3.3.5.2 Bazinis, invertavimas realizuotas CU

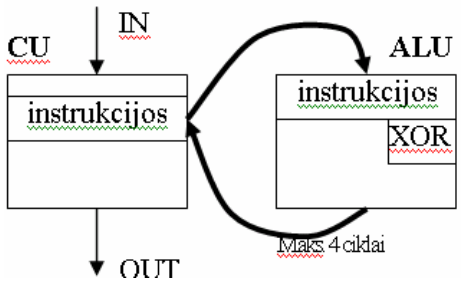
Pagal 18 lentelėje pateiktą CU operacijų aprašymą, matoma, kad dažnai sutinkama *NOT* instrukcija. Ji iškeliama iš ALU ir realizuota CU. 20 lentelėje matoma, kad visos operacijos naudojusios invertavimą iš ALU atliekamos vienu ciklu greičiau, sudėtinė *XOR* operacija, turėjusi 3 invertavimo būsenas, atliekama per 4 ciklus vietoje 7. Bendras būsenų automato būsenų skaičius sumažėja nuo 42 iki 31. 26 paveiksle schematiškai parodytas *NOT* perkėlimas.



26 pav. *Invertavimo perkėlimas į ALU*

3.3.5.3 XOR perkeltas į ALU

Pasirinktas ALU modulis neturi loginės *XOR* operacijos, ją galima išreikšti kitomis ALU operacijomis: *AND, NOT* ir *OR*. Šiame modelyje ALU papildomas vieno žingsnio *XOR* operacija. Būsenų automato žingsnių skaičius *n_axb* šakoje sutrumpėja nuo 7 (*Bazinis*) iki 3. Toks ALU papildymas įtakoja tik vienos modelio operacijos atlikimą. Atliktas sudalinimas schematiškai pavaizduotas 27 paveiksle.



27 pav. *ALU papildymas XOR*

3.3.5.4 XOR perkeltas į ALU, invertavimas į CU

Bendras ankstesnių dviejų architektūrų (3.3.5.2 ir 3.3.5.3) junginys. Imamas modelis, kuriame ALU papildytas *XOR* ir jame *NOT* perkeliamas iš ALU į CU. CU ilgiausia operacija *n_axb* sutrumpėja iki 3 būsenų, papildomai visų veiksmų atlikimas sutrumpėja per tiek, kiek reikėdavo invertavimo operacijų iš ALU – jos dabar vykdomos CU modulyje.

Pagal bendrą būsenų automato žingsnių skaičių gaunamas vienas taupiausių (mažesnis elementų skaičius ir užimamas plotas) sistemos variantų.

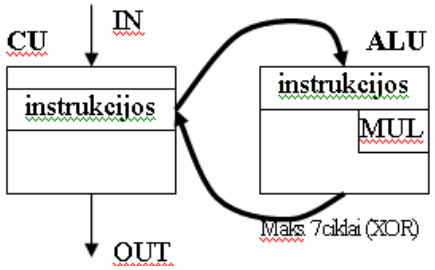
3.3.5.5 Iš ALU pašalinamos visos nenaudojamos instrukcijos

Sistemoje panaudotas ALU savo savybėmis yra perteklinis, modeliuojamas variantas, kai ALU pilnai atitinka CU poreikius. Toks modelis yra laikomas idealiu sistemos modeliu. Visos operacijos atliekamos minimaliu žingsnių skaičiumi, dažnai atliekamas invertavimas perkeliamas į valdymo bloką (CU). ALU modulis neturi neišnaudotų instrukcijų – jos pašalinamos.

Pagal bendrą būsenų automato žingsnių skaičių gaunamas vienas taupiausių (mažesnis elementų skaičius ir užimamas plotas) sistemos variantų.

3.3.5.6 Bazinis, ALU papildytas daugybos operacija

Praplečiamas ALU, jame realizuojama daugybos operacija. Pradiniame variante jos nebuvo visai. CU taip pat papildytas operacija *amb* – operandas *A* dauginamas iš operando *B*. Reikia pasidaryti palyginimui vieno žingsnio daugintuvo modelį. Schematinis sistemos vaizdas po papildymo nauja instrukcija parodytas 28 paveiksle.



28 pav. *ALU papildymas MUL*

3.3.5.7 Bazinis, daugyba išreikšta per ADD ir SHFL

Kaip *Bazinis* modelyje buvo daroma *XOR* atveju, taip daugybą galima realizuoti panaudojant turimas paprastas ALU instrukcijas. Daugyba realizuota sumavimo (*ADD*) ir postūmio kairėn (SHFL) ALU instrukcijomis. Modelis pagal funkcionalumą analogiškas aukščiau sudarytam su vieno žingsnio daugybos operacija, realizuojama ALU.

CU papildytas aukščiau dar neaprašyta operacija *amb* – operandas *A* dauginamas iš operando *B*. Pagal būsenų skaičių gaunamas pats ilgiausias CU operacijos atlikimo kelias. Priklausomai nuo daugiklio L_i dvejetainių narių („0“ ar „1“ kiekio) skiriasi būsenų automato žingsnių skaičius. Jei daugiklio L_i narys yra „0“ – atliekamas dauginamojo poslinkis kairėn (2 būsenos). Jei daugiklio L_i narys yra „1“ – atliekamas dauginamojo sumavimas prie sumos

kaupiklio ir poslinkis kairėn (3 būsenos). Taigi, jei daugiklis turi vieną „1“ per visą ilgį *L=8* – sandaugai reikės 19 ciklų (18, jei visi „0“), jei turės *L=8* kiekį „1“ per visą savo ilgį, tai sandaugai reikės 26 ciklų. Taigi, operacijos *amb* žingsnių skaičius *K* apskaičiuojamas:

$$S_0 \cdot 2 \text{ („0“ kiekis kart 2)} + S_1 \cdot 3 \text{ („1“ kiekis kart 3)} + 2 = K$$

Šios sistemos modifikacijos, *amb* operacijos atlikimo būsenų diagramos šaka parodyta 22 paveiksle.

3.3.5.8 Bazinis, CU papildytas atimtimi

Pradinis CU modelis nenaudoja atimties operacijos, nors ji ALU ir yra. CU papildomas atimties operacija, neaprašyta aukščiau pateikiamame sistemos aprašyme. Nauja CU operacija *a_b* – iš operando *A* atimamas *B*, atliekama per 2 būsenas.

3.3.5.9 Bazinis, atimtis realizuota per sudėtį ir invertavimą

*A*nalogiškas pagal operacijų aibę aukščiau aprašytai architektūrai, tik atimtis realizuota per *ADD*, *NOT* ir *INC* ALU instrukcijas. Iš ALU pašalinama nenaudojama *SUB* (atimties) instrukcija. CU papildomas atimties operacija, neaprašyta aukščiau pateikiamame sistemos aprašyme. Nauja CU operacija *a_b* – iš operando *A* atimamas *B*. Atimtis šiame modelyje atliekama per 4 būsenas.

3.4 Metaprogramavimas

Aprašant kuriamos sistemos modelius panaudotas homogeninis metaprogramavimas, tiksliau parametrizavimas. ALU ir CU moduliai yra parametrizuojami prieš modeliavimą – galima keisti operandų bitų plotį *L*. Operandų plotis nurodytas bibliotekoje *typelib.h* kurią naudoja visi sistemos komponentai (tekstas pateiktas 2 priede). Toje pačioje bibliotekoje aprašomos ir ALU bei CU operacijų aibės.

ALU architektūra ir operacijų aibė leidžia nesunkiai realizuoti parametrizavimą – veiksmai atliekami su pilnu arba puse operando (nesunkiai atskiriama aukštesnė ir žemesnė skiltys), *SystemC* tipas duomenų *bv<>* (*bit vector*) automatiškai užpildo nenaudojamas bitų sekcijas „0“ konvertuojant iš kito (pvz.: *unsigned int* tipo).

Nurodžius *L* ilgį atliekamas modeliavimas, kodo modifikuoti nereikia. Atliekant sintezavimą – norint gauti testuoto modelio schemą – parametro keisti negalima.

3.5 Siūlomos ALU architektūros analizės apibendrinimas

Apibendrinama pasirinkta ALU atliekama operacijų aibė ir architektūrinis sprendimas:

1) Pagal 2 dalyje atliktą probleminės srities apžvalgą sudarytas ALU modelis, pasirinkta operacijų aibė aprašyta 14 lentelėje;
2) Pagal 2 dalyje atliktą probleminės srities apžvalgą sudarytas CU modelis, pasirinkta dviejų pakopų architektūra su nustatomo pločio (L) operandais;
3) Modelis aprašytas aparatūros aprašymo kalba *SystemC*. Ši kalba pasirinkta dėl savo naujumo. *SystemC* pagal kilmę artima programinės įrangos aprašymo kalbai C++, todėl pritaikyme galima optimizuoti kodą išskaidant į aparatūrinę ir programinę dalis;
4) Aprašytos 9 sistemos modifikacijos perkeliant veiksmus iš ALU į CU ir atvirkščiai. Modifikacijos plačiau aprašytos 3.3.5 dalyje:
   - Bazinis;
   - Bazinis, invertavimas realizuotas CU;
   - XOR perkeltas į ALU;
   - XOR perkeltas į ALU, invertavimas į CU;
   - Iš ALU pašalinamos visos nenaudojamos instrukcijos;
   - Bazinis, ALU papildytas daugybos operacija;
   - Bazinis, daugyba išreikšta per ADD ir SHFL;
   - Bazinis, CU papildytas atimtimi;
   - Bazinis, atimtis realizuota per sudėtį ir invertavimą.
5) Išskirti galimi modeliuojamos sistemos trūkumai:
   - Ne visos ALU instrukcijos panaudotos – naudojamas ALU perteklinis savo galimybėmis, arba reikia kurti CU kuris pritaikys visas ALU funkcijas. Pritaikant realų IP komponentą tokia situacija normali;
   - Galima bandyti optimizuoti kai kurias operacijas – papildoma logika pačiame CU sutrumpintų rezultato gavimo kelią, nereiktų laukti kol ALU pereis visas būsenas;
   - Galima turėti invertuotas įėjimų reikšmes – tai pakankamai dažnai atliekama operacija, konkretus optimizavimo atvejis;
   - Reikia išanalizuoti dažniausiai atliekamas instrukcijas bei jų atlikimui reikalingus ALU žingsnius ir optimaliai paskirstyti veiksmus tarp valdymo logikos ir valdomo komponento.

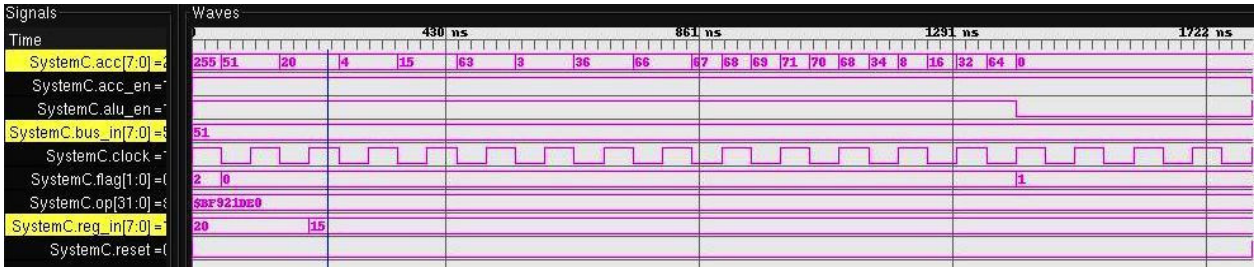
4 Tyrimo rezultatai ir jų įvertinimas

ALU branduolys ir valdanti schema CU aprašyti *SystemC* aparatūros aprašymo kalba. Sudarytos sistemos modifikacijos (3.3.5) modeliuojamos *SystemC* kompiliatoriumi, modelio signalų reikšmės išsivedamos terminale. Po modeliavimo gautas *trace* failas analizuojamas grafine *gtkwave* programa. Sintezei naudojamas *Synopsys Design Analyzer* įrankis.

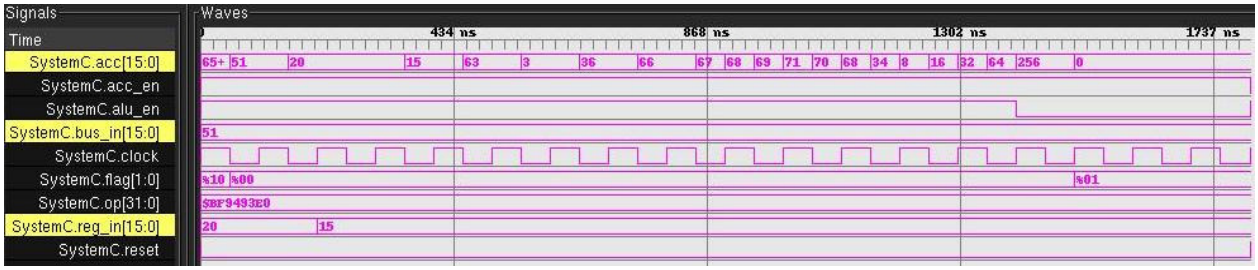
4.1 ALU modeliavimo rezultatai

Žemiau matomuose paveiksluose pateikiami 3 variantų modeliavimo rezultatai. ALU išbandomas su L=8, L=16 ir L=64 nustatymais (*L* – operandų plotis, bitais). Testavimui naudojamas tie patys testiniai rinkiniai (pateikiama 2 priede). ALU modeliavimas atliktas dar nesujungus su CU – rezultatuose nematomos po adaptavimo naujai atsiradusios ALU kojos.

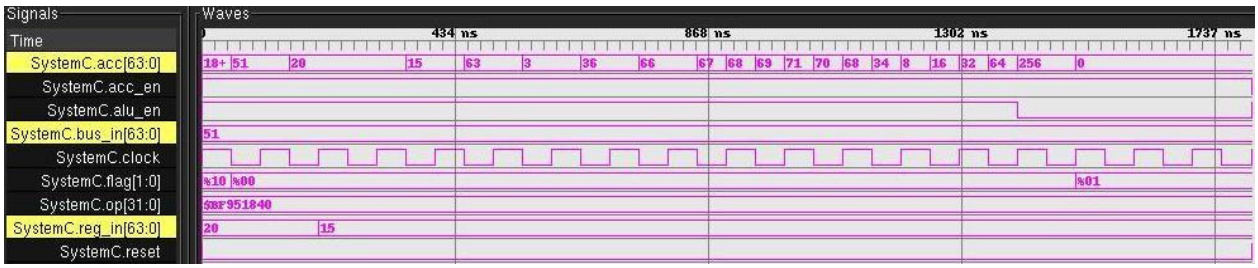
Vykdomas dvejetainis sukompiliuotas programos kodas, rezultatai peržiūrimi *gtkwave* programiniu paketu. Modeliuojamas ALU modulis priklausantis sistemos modeliui *Bazinis*. Modeliavimo rezultatai paveiksluose 29, 30 ir 31.



29 pav. *ALU L=8 modeliavimas*



30 pav. *ALU L=16 modeliavimas*



31 pav. *ALU L=64 modeliavimas*

ALU išėjimas *ACC* visuose modeliavimo rezultatuose atiduoda tą pačią reikšmę, nepriklausomai nuo operandų pločio (*L* dydis matomas paryškintose eilutėse, laužtiniuose skliaustuose). Tai parodo, kad pakeitus parametro, nustatančio ALU operandų plotį, reikšmę – įtaisas veikia taip pat.

4.2 ALU sintezės rezultatai

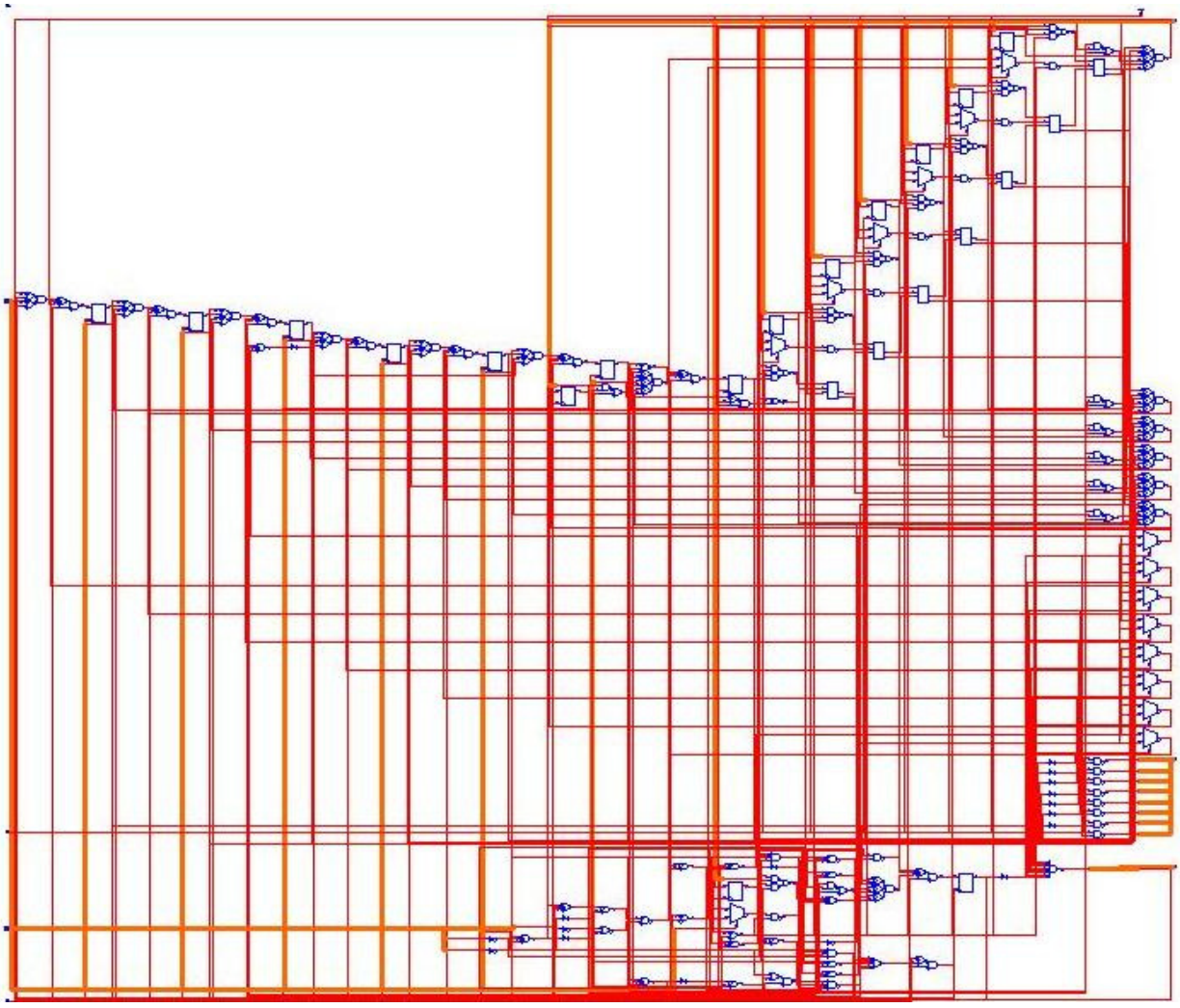
21 lentelėje pateiktas ALU modelių sintezės rezultatų palyginimas. Sintezuotas *Bazinis* modelyje naudotas ALU, kurio modeliavimo rezultatai buvo pateikti ankstesniame skyriuje.

21 lentelė. ALU sintezės rezultatai

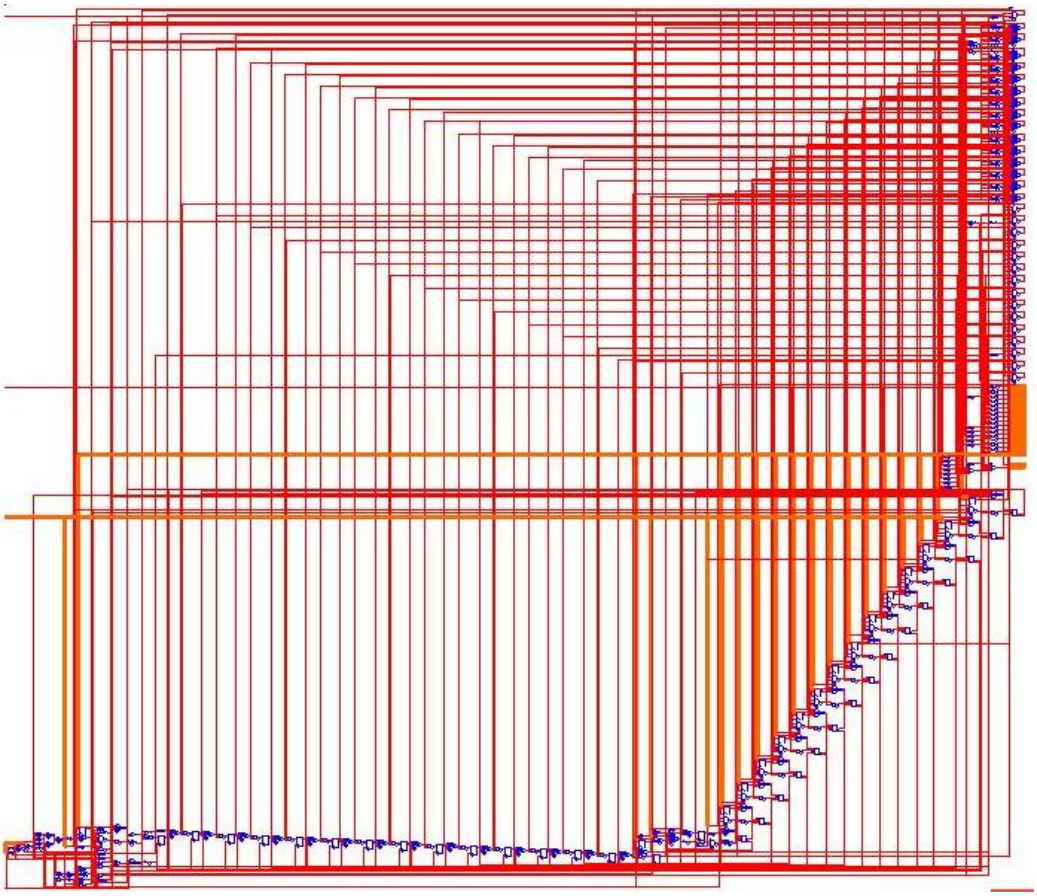
Modelis	***Ventilių sk.***	***Laidų sk.***	***Prievadų sk.***	***Plotas, μm^2***	***Vėlinimas, ns***	***Energija, mW***
ALU L=8	139	161	34	258,17	2,11	125,76
ALU L=16	263	301	58	495,35	2,74	207,68
ALU L=64	906	1103	202	2060,82	2,82	697,48

Ventilių skaičius lentelėse pateiktas vienetais. Plotas matuojamas μm^2. Vėlinimais taktais, prilygstančiais nanosekundėms. Modelio suvartojama energija matuojama mW. Schema maitinama 5V įtampa.

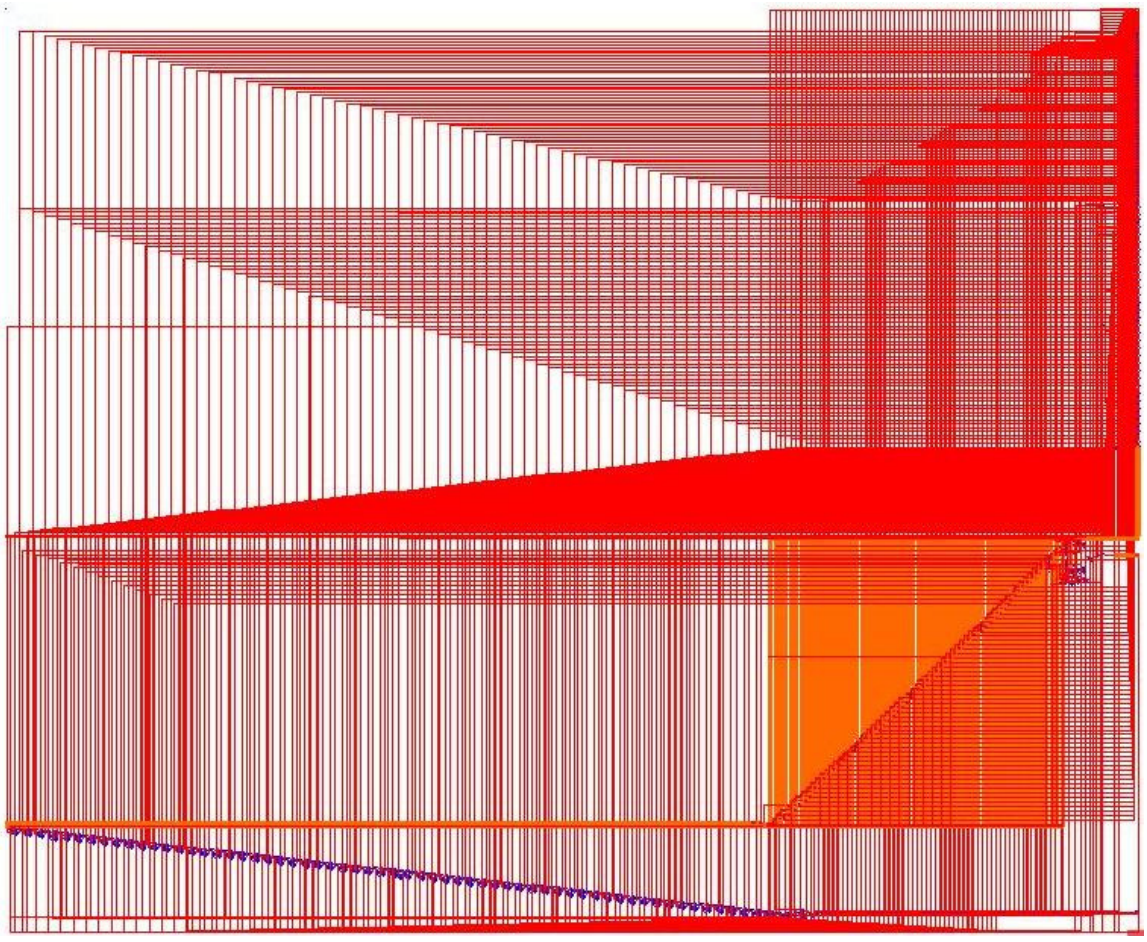
Toliau paveiksluose 32, 33 ir 34 pateikiamos susintezuotos ALU su skirtingais operandų pločiais schemos.



32 pav. *ALU L=8 susintezuota schema*



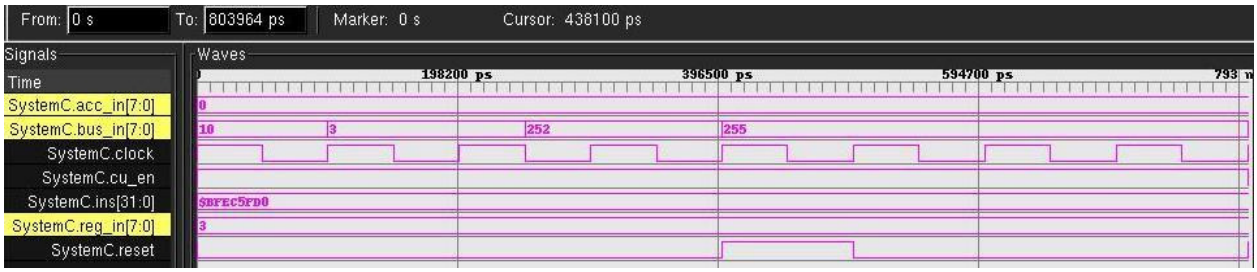
33 pav. *ALU L=16 susintezuota schema*



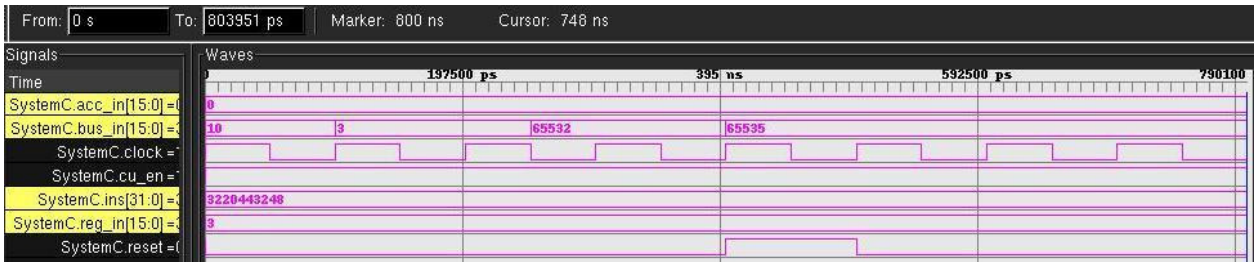
34 pav. *ALU L=64 susintezuota schema*

4.3 CU modeliavimo rezultatai

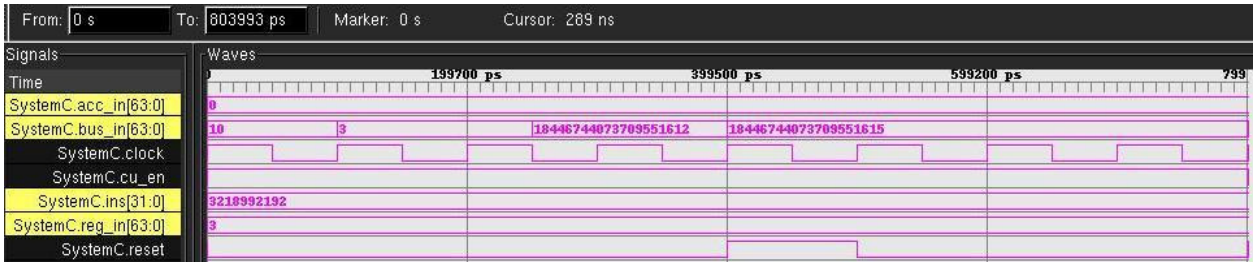
Žemiau matomuose paveiksluose pateikiami 3 variantų modeliavimo rezultatai. CU išbandomas su L=8, L=16 ir L=64 nustatymais (L – operandų plotis, bitais). Testavimui naudojamas tie patys testiniai rinkiniai (pateikiama 2 priede). Vykdomas dvejetainis sukompiliuotas programos kodas, rezultatai peržiūrimi *gtkwave* programiniu paketu. Dėl naudojamų invertavimo operacijų skiriasi su įvairaus ilgio operandais gaunami rezultatai. Modeliuotas modelis *Bazinis*. Modeliavimo rezultatai paveiksluose 35, 36, 37:



35 pav. *CU L=8 modeliavimas*



36 pav. *CU L=16 modeliavimas*



37 pav. *CU L=64 modeliavimas*

Išanalizavus CU modeliavimo rezultatus, matoma, kad naudojant tą patį testinių duomenų rinkinį su skirtingo operandų pločio modeliais gaunamas skirtingas rezultatas. Tai yra panaudotų invertavimo operacijų pasekmė. CU rezultatas atiduodamas į įėjimo/išėjimo koją *bus_in*. Testinio rikinio pirmos dvi operacijos nenaudoja inversijos, tad sutampa visų modeliuotų

įtaisų išduodamas rezultatas. Atlikus *n_aab* operaciją, modeliavimo rezultatai išsiskiria: kai L=8 – *not(3)* rezultatas 252; kai L=16 – *not(3)* rezultatas 65532; kai L=64 – *not(3)* rezultatas 18446744073709551612. CU modelių su skirtingais operandų pločiais veikimas vienodas, išsiskiria tik invertavimo operacijos dešimtainė reikšmė (dvejetainės reikšmės invertuoti bitai sutampa).

4.4 CU sintezės rezultatai

22 ir 23 lentelėse pateiktas CU modelių sintezės palyginimas. Duomenys lentelėje 23 – ALU išskleidus, 22 – ALU skaičiuojamas kaip vienas elementas.

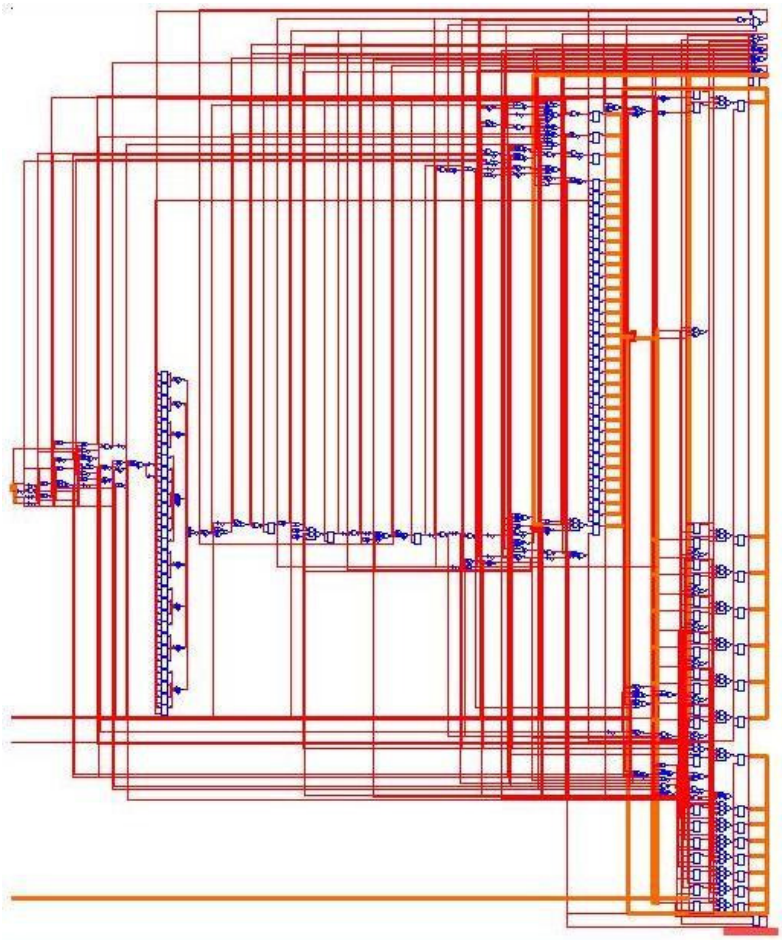
22 lentelė. CU sintezės rezultatai. Be ALU

Modelis	***Ventilių sk.***	***Laidų sk.***	***Prievadų sk.***	***Plotas, μm^2***	***Vėlinimas, ns***	***Energija, mW***
CU L=8	260	282	25	741,42	6,07	131,22
CU L=16	337	375	41	733,31	6,58	205,66
CU L=64	730	864	137	3468,45	7,19	384,26

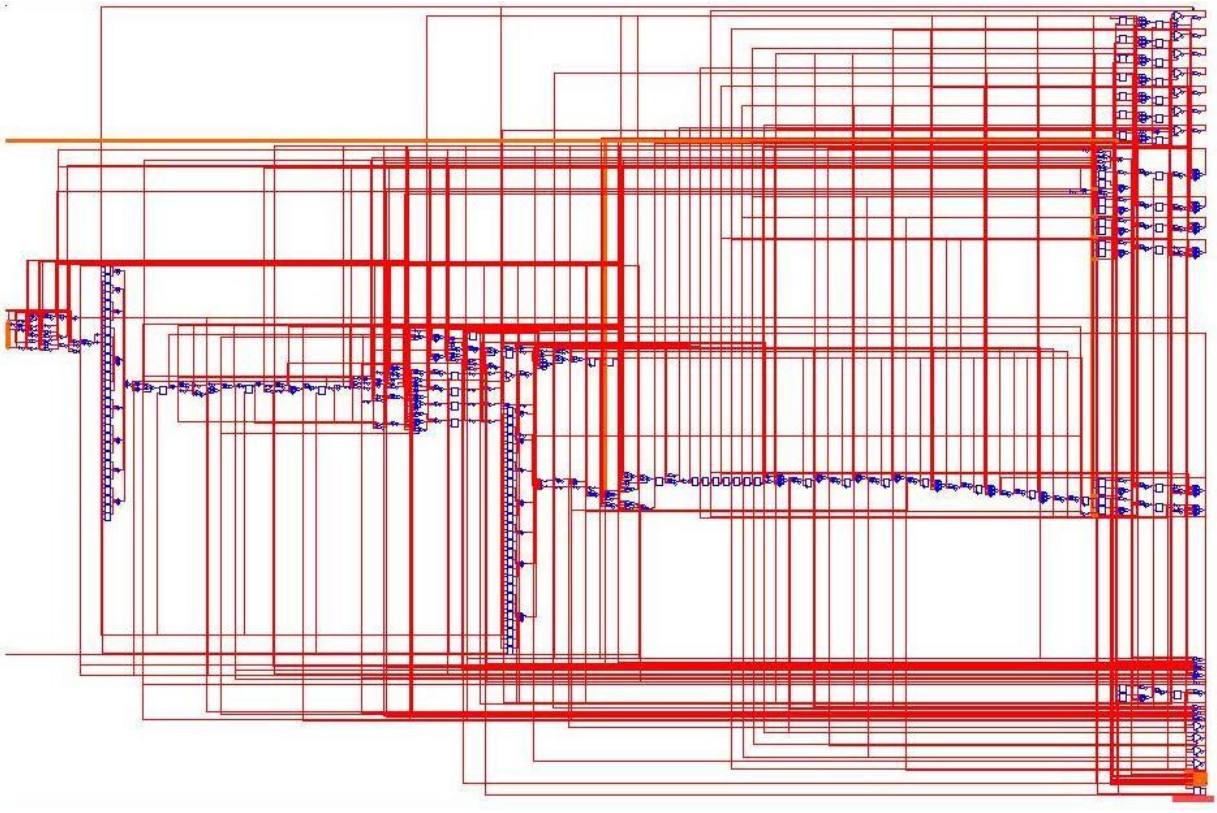
23 lentelė. CU sintezės rezultatai. Su ALU

Modelis	***Ventilių sk.***	***Laidų sk.***	***Prievadų sk.***	***Plotas, μm^2***	***Vėlinimas, ns***	***Energija, mW***
CU L=8	388	410	25	767,33	6,07	132,98
CU L=16	536	574	41	1098,00	6,93	180,24
CU L=64	1441	1637	137	3487,76	7,43	458,01

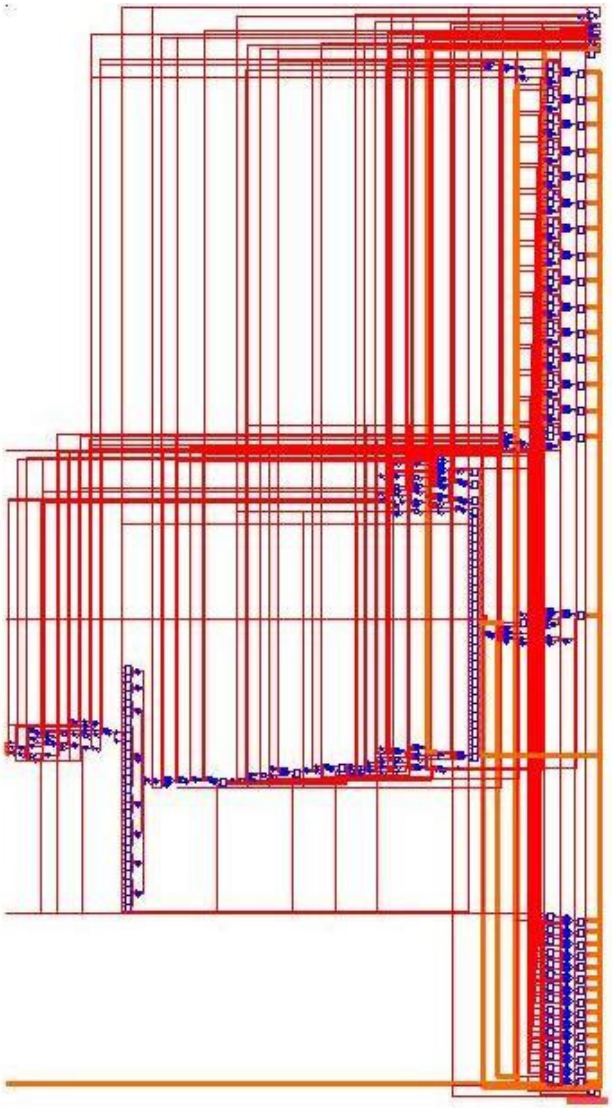
Ventilių skaičius lentelėse pateiktas vienetais. Plotas matuojamas μm². Vėlinimais taktais, prilygstančiais nanosekundėms. Modelio suvartojama energija matuojama mW. Schema maitinama 5V įtampa. Toliau paveiksluose pateikiamos susintezuotos schemos. Sintezuojamas modelis *Bazinis*, kurio modeliavimo rezultatai pateikti ankstesniame skyriuje. Susintezuotos schemos, tiek išskleidžiant ALU, tiek jį laikant vienu loginiu elementu pateiktos paveiksluose 38-42.



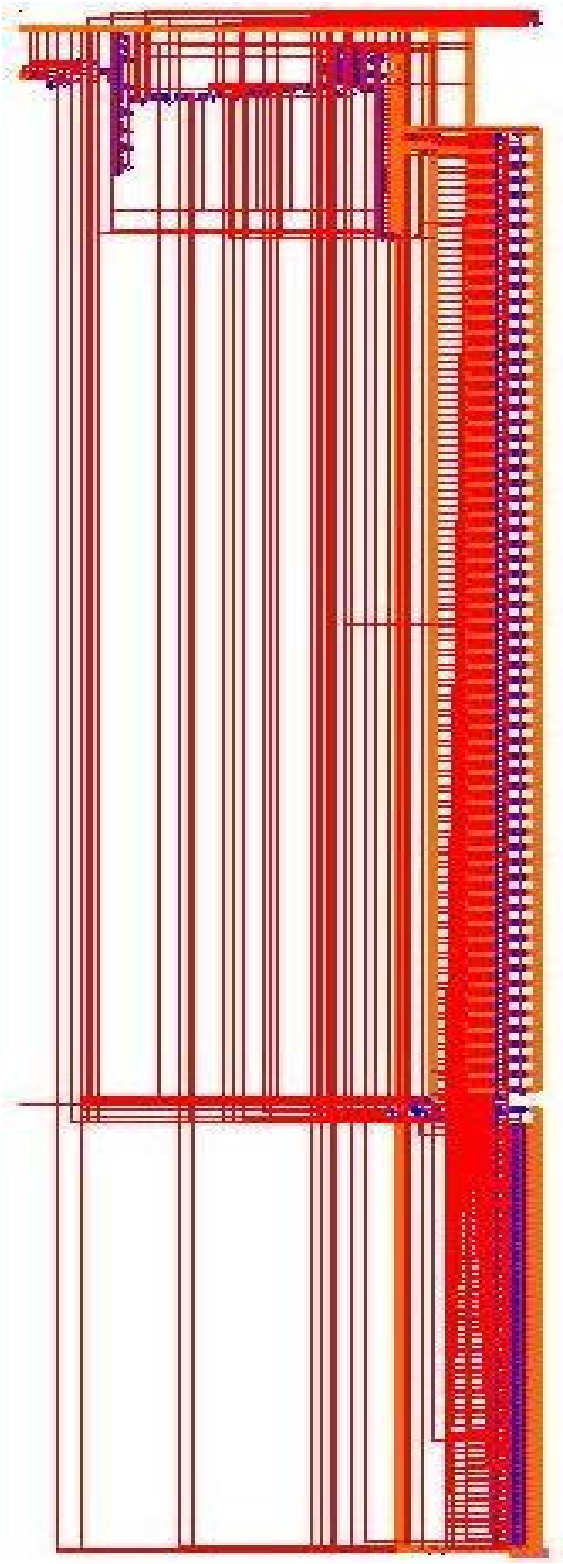
38 pav. *CU L=8 susintezuota schema. ALU neiššskleidus*



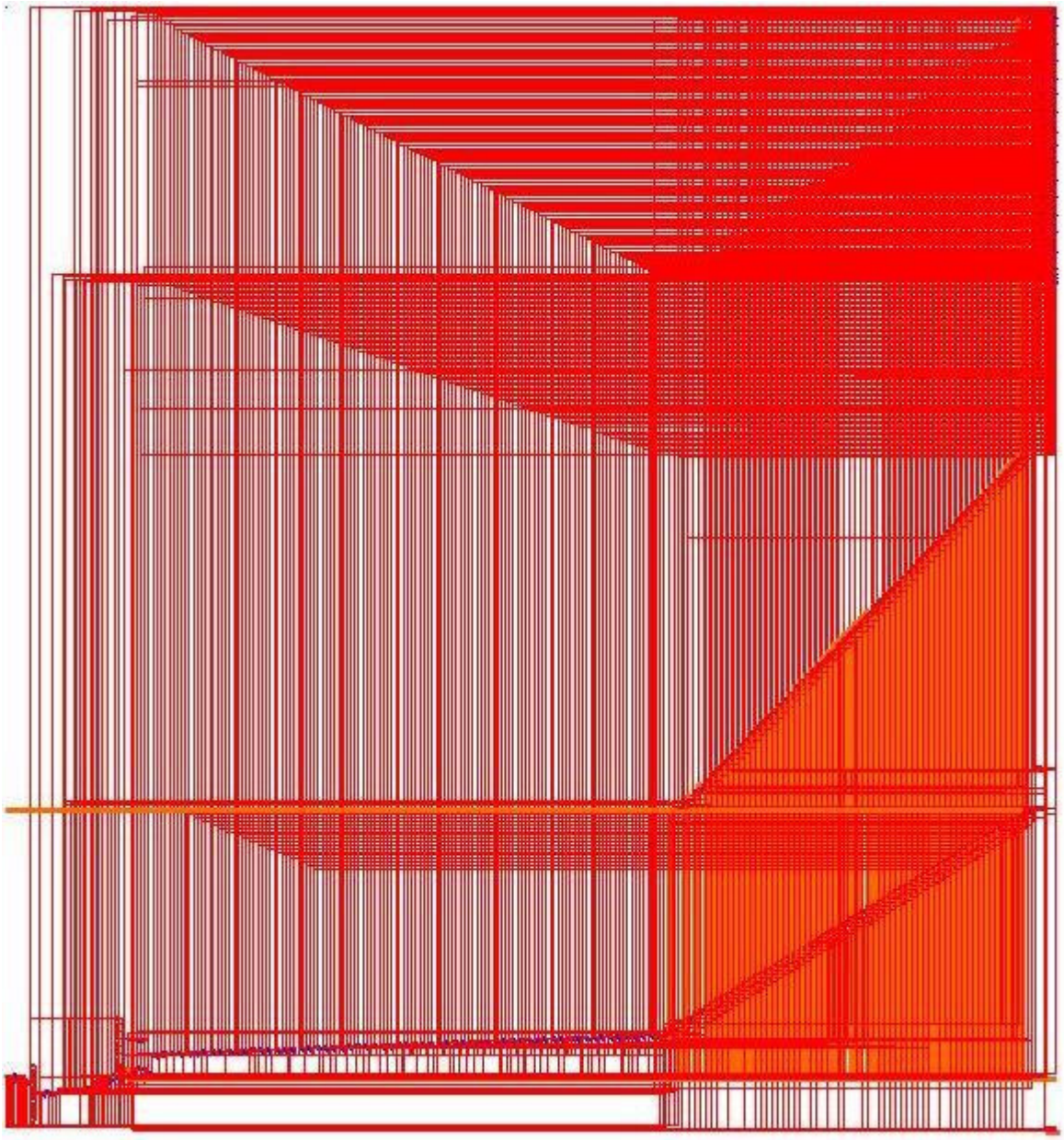
39 pav. *CU L=8 susintezuota schema su ALU*



40 pav. *CU L=16 susintezuota schema. ALU neiššskleidus*



41 pav. *CU L=64 susintezuota schema. ALU neiššskleidus*



42 pav. *CU L=64 susintezuota schema su ALU*

4.5 Operacijų sudalinimas tarp CU ir ALU

Pateikiami tik vieno bitų pločio L = 8 eksperimentų rezultatai. Analizuojami sukurtų CU-ALU modelių modifikacijų sintezės rezultatai. Ventilių skaičius, laidų skaičius, plotas, laikinės bei energijos išsklaidymo charakteristikos pateikiamos pilnai išskleistai schemai (įeina ALU

komponento vidiniai elementai), atliekamas optimizavimas su nustatymu *High* ir neleidžiant ploto optimizavimo [8].

Susintezuoti kiti 8 modeliai ir pateiktos kiekvieno jų 6 pagrindinės charakteristikos: ventilių skaičius, laidų skaičius, prievadų skaičius, plotas, vėlinimas ir energijos suvartojimas. Svarbiausios iš jų laikomos: *plotas* – lusto bendras užimamas plotas, priklauso nuo naudojamos technologijos, bei elementų skaičiaus, naudojant universalius komponentus plotas gali būti didesnis, atlikus sudėtingesnius operacijų ir aparatūros aprašymo kodo optimizavimus plotas gali būti mažesnis. Plotas pateiktas μm^2. *Vėlinimas* – kritinio kelio vėlinimas tai ilgiausio tiesaus kelio tarp schemos įėjimo ir išėjimo laikinės charakteristikos įvertinimas. Sintezuoti modeliai turi būsenų automatus, tad šis įvertinimas tiksliai neatspindi galimo maksimalaus energijos suvartojimo. Tai svarbus parametras apibudinantis schemos greitaveiką, lentelėse pateiktas *ns*. *Energija* – vienas svarbiausių parametrų, keičiant schemos sudėtingumą ir funkcionalumą labiausiai juntamas energijos suvartojimo pokytis. Ši charakteristika įtakoja lusto kaitimą ir galios reikalavimus jį naudojančiai sistemai. Lentelėse pateikiama *mW*.

4.5.1 Inversijos iškėlimas

Pirmas eksperimentas atliktas iškeliant iš ALU invertavimo operaciją į CU. Pagal 18 lentelėje pateiktą ALU būsenų aprašymą matoma, kad dažniausiai kartojasi *NOT* operacija. Modelyje *Bazinis* ji panaudojama 9 iš 12 atliekamų operacijų. Atliekami pakeitimai:

- ✓ ALU pašalinama *NOT* operacija;
- ✓ CU padaromi du buferiai, kurie visada turi A ir B įėjimų invertuotas reikšmes !A ir !B;
- ✓ visose operacijose, kur reikia inversijos, naudojamos !A ir !B reikšmės.

Modelio sintezės rezultatai pateikti 24 lentelėje.

24 lentelė. NOT iškėlimas iš ALU

Modelis	***Ventilių sk.***	***Laidų sk.***	***Prievadų sk.***	***Plotas, μm^2***	***Vėlinimas, ns***	***Energija, mW***
Bazinis	388	410	25	767,33	6,07	132,98
Bazinis, invertavimas realizuotas CU	338 (-15%)	360 (-14%)	25	690,74 (-11%)	6,36 (5%)	153,18 (15%)

Atlikus aukščiau minėtus pakeitimus gauta schema turi mažiau ventilių (15%), tuo pačiu mažiau laidų ir mažesnį plotą. Tačiau, tokiu būdu išauga kritinio kelio vėlinimas bei energijos suvartojimas.

4.5.2 XOR įkėlimas į ALU

Pradiniame sistemos modelyje *Bazinis* ALU modulyje nėra operacijos *XOR*, ji gaunama per *AND*, *OR* ir *NOT* kombinacijas. Atliekamas eksperimentas papildant ALU tiesiogine *XOR* operacija, visos kitos paliekamos, nes yra naudojamos kitiems veiksmams. Sutrumpėja būsenų automato pati ilgiausia šaka.

Taip pat padaromas jungtinis šio ir 4.5.1 dalyje aprašyto eksperimentų modelis. Tokiu būdu sistemos būsenų automato bendras atliekamas būsenų skaičius trumpėja iki minimumo. Būsenų automatas aprašytas 3.3.4 dalyje.

Lentelėje 25 pateikiami sintezės rezultatai.

25 lentelė. XOR realizavimas ALU ir NOT iškėlimas

Modelis	***Ventilių sk.***	***Laidų sk.***	***Prievadų sk.***	***Plotas, μm^2***	***Vėlinimas ns***	***Energija, mW***
Bazinis	388	410	25	767,33	6,07	132,98
XOR perkelta į ALU	369 (-5%)	391 (-5%)	25	713,77 (-8%)	5,66 (-7%)	118,00 (-13%)
XOR perkelta į ALU, invertavimas į CU	347 (-12%)	369 (-11%)	25	685,99 (-12%)	5,44 (-12%)	150,73 (13%)

Iš gautų rezultatų matyti ventilių ir laidų skaičiaus mažėjimas abiejų eksperimentų atveju. Sumažėja užimamas plotas, tokia tendencija pastebėta ir aukščiau aprašytame eksperimente. Įkėlus *XOR* vykdymą į ALU papildomai sumažėja kritinių kelių vėlinimas bei energijos suvartojimas. Optimizavus *NOT* vykdymą po *XOR* įkėlimo pastebima atvirkštinė prieš tai atliktam eksperimentui tendencija su vėlinimu. Šį kartą jis ne didėja, o mažėja. Energijos išsklaidymui *XOR* įkėlime pastebimas sumažėjimas 13%, o papildomai iš *ALU* iškėlus inversiją išlieka energijos vartojimo padidėjimas.

Lentelėje pateiktas pagerėjimas/pablogėjimas skliausteliuose skaičiuotas nuo lyginamojo modelio *Bazinis*.

4.5.3 Iš ALU pašalinamos visos nenaudojamos instrukcijos

Sudaromas sistemos modelis, kuriame abu sistemos komponentai ALU ir CU pilnai atitinka vienas kitą, imamas 4.5.2 dalyje aprašytas modelis, ir iš ALU šalinamos visos nereikalingos operacijos: *op_lda, op_ldm, op_ldi, op_ldii, op_sub, op_inc, op_not, op_shfl, op_shfr*. Be to, *XOR* realizuotas ALU, ir dažnai naudojama inversija perkelta į CU pakopą.

Gauta schema pagal būsenų skaičių yra pati taupiausia, taip pat pašalinus perteklinę logiką tikimasi visų kitų charakteristikų pagerėjimo. Rezultatai pateikti 26 lentelėje.

26 lentelė. Iš ALU pašalintos visos CU nenaudojamos instrukcijos

Modelis	Ventilių sk.	Laidų sk.	Prievadų sk.	Plotas, μm^2	Vėlinimas, ns	Energija, mW
Bazinis	388	410	25	767,33	6,07	132,98
Iš ALU pašalinamos visos nenaudojamos instrukcijos	301 (-29%)	322 (-27%)	25	591,31 (-29%)	5,49 (-11%)	142,50 (7%)

Iš sintezavimo ataskaitos matomas visų charakteristikų pagerėjimas išskyrus energijos suvartojimą. Gauta schema užima trečdaliu mažesnį plotą, turi trečdaliu mažiau ventilių ir laidų nei pradinis modelis. Pagerėjus vienai charakteristikai, remiantis energijos tvermės dėsniu kenčia kita – eksperimentui sudarytas modelis vartoja 7% daugiau energijos.

4.5.4 Daugybos operacijos paskirstymas

Pirminis sistemos modelis neturi dauginimo nei ALU nei jį per tarpines operacijas naudoja CU schema. Šio eksperimento metu praplečiamas sistemos funkcionalumas, sukuriamos dvi modifikacijos. Pirmu atveju į ALU operacijų aibę įtraukiama vieno ciklo daugybos operacija, taip pat CU modulis papildomas dviejų operandų sandaugos funkcija. Antru - sandauga išreiškiama panaudojant jau turimas ALU sudėties (*ADD*) ir postūmio kairėn (*SHFL*)operacijas, CU papildomas iki tol neturėta daugybos operacija. Gauti rezultatai pateikti 27 lentelėje.

27 lentelė. ALU papildyta daugyba, daugyba CU išreikšta per sudėtį ir postūmį

Modelis	Ventilių sk.	Laidų sk.	Prievadų sk.	Plotas, μm^2	Vėlinimas ns	Energija, mW
Bazinis	388	410	25	767,33	6,07	132,98
Bazinis, ALU papildyta daugybos operacija	525 (35%)	572 (40%)	25	1014,82 (32%)	6,00 (-1%)	147,23 (11%)
Bazinis, daugyba išreikšta per ADD ir SHFL	573 (48%)	626 (53%)	25	1196,44 (56%)	5,79 (-5%)	201,22 (51%)

Iš pateiktų rezultatų matyti, kad schemą papildžius daugintuvu smarkiai didėja ventilių, tuo pačiu laidų skaičius ir bendras plotas. Visais atvejais, naudojant vieno žingsnio daugintuvą ALU, charakteristikos padidėjimas siekia 32-40%. Daugintuvą gaminant panaudojant jau turimas ALU operacijas charakteristikos dar didėja apie 23%.

Modelis su cikliniu daugintuvu užima 56% didesnį plotą nei prieš modifikaciją. Analogiškai išauga energijos išsklaidymas – 51%. 22 paveiksle parodyta šios modifikacijos būsenų diagrama.

Šis modelis turi papildomą apribojimą – sandaugos rezultatas negali viršyti maksimalios operandų pločio reikšmės 2^L, kitu atveju operacijos rezultatas bus neteisingas.

4.5.5 Atimties operacijos paskirstymas

Pirminis sistemos modelis nenaudoja ALU turimos atimties operacijos. Pagaminti du modeliai: pirmasis CU papildomas atimties operacija ir naudoja vieno žingsnio ALU atimties (*SUB*) operaciją, antrame atimtis realizuota per ALU sudėties (*ADD*) ir invertavimo (*NOT*) operacijas, iš ALU pašalinama nebenaudojama vieno ciklo atimties operacija *SUB*. Sintezavimo rezultatai pateikti 28 lentelėje.

28 lentelė. CU papildytas atimtimi. Atimtis CU realizuota per inversiją ir sudėtį

Modelis	**Ventilių sk.**	**Laidų sk.**	**Prievadų sk.**	**Plotas, μm^2**	**Vėlinimas ns**	**Energija, mW**
Bazinis	388	410	25	767,33	6,07	132,98
Bazinis, CU papildytas atimtimi	403 (4%)	425 (4%)	25	771,29 (0.5%)	6,32 (4%)	152,55 (15%)
Bazinis, atimtis realizuota per sudėtį ir invertavimą	400 (3%)	421 (3%)	25	775,99 (1%)	5,39 (-13%)	116,90 (-14%)

Iš pateiktų rezultatų pastebimas tendencingas ir nežymus ventilių, laidų ir ploto padidėjimas abiem atvejais. CU modulis padidėja 2 (pirmu atveju) ir 4 (antru atveju) būsenomis. Priešingai pakinta kritinio kelio vėlinimo bei energijos suvartojimo charakteristikos. *Bazinis* modelį CU modulyje papildžius logika, kuri išnaudoja ALU atimties operaciją, 4% padidėja vėlinimas ir 15% suvartojamos energijos kiekis. Atimtį realizuojant per sudėtį ir invertavimą, bei iš ALU pašalinus perteklinę logiką pastebimas paspartėjimas (13%) bei sumažėjęs energijos vartojimas (14%).

4.6 Rezultatų įvertinimas

Atlikus sudarytų ALU modelių sintezavimą gauti rezultatai pateikiami 4.5 skyriuje. Susintezuoti 9 modeliai ir pateiktos kiekvieno jų 6 pagrindinės charakteristikos: ventilių skaičius, laidų skaičius, prievadų skaičius, plotas, vėlinimas ir energijos suvartojimas. Svarbiausios iš jų yra: plotas, vėlinimas ir energijos suvartojimas.

Visi sintezuoti modeliai turi būsenų automatus, tad pateiktas lentelėse 24-28 vėlinimas nėra tikslus, pateiktas kritinio kelio vėlinimas. Reikia turėti kiekvienos iš galimų būsenų šakos maksimalų arba vidutinį vėlinimą.

Rezultatai apibendrinti 5 skyriuje *Išvados*.

Bendra duomenų imtis yra per maža, kad rezultatų įvertinimui būtų panaudoti statistiniai metodai. Visi rezultatai bendroje lentelėje pateikti 1 priede. Remiantis atliktais eksperimentais galima teigti, jog naudojant dviejų pakopų ALU modelį ir operacijų aibės sudalinimą i ALU ir CU poaibius galima pasiekti:

– nuo 8% iki 29% ploto sumažėjimą;

– nuo 13% iki 14% energijos vartojimo sumažėjimą;

– nuo 1% iki 13% kritinio kelio vėlinimo trumpėjimą;

– nuo 5% iki 29% ventilių skaičiaus sumažėjimą.

Panaudojus ne optimalų operacijų aibės sudalinimą galima pralaimėti:

– nuo 1% iki 56% ploto padidėjimą;

– nuo 7% iki 51% energijos vartojimo padidėjimą;

– nuo 4% iki 5% kritinio kelio vėlinimo pailgėjimą;

– nuo 3% iki 48% ventilių skaičiaus padidėjimą.

Sistemos modeliai aprašyti *SystemC*. Atliekant sintezavimą *SystemC* aprašas verčiamas *Verilog* struktūriniu aprašu, todėl tikėtina, kad gali būti neadekvatus kai kurių konstrukcijų traktavimas ataskaitose, lyginant aprašus, pavyzdžiui, *if* ir *case* sakiniai.

Sintezės įrankis gali ne tiksliai paskaičiuoti schemos vėlinimą, nes jos turi būsenų automatus, reikalingas atskirų būsenų vėlinimo matavimas.

Matavimo vienetai ir dydžiai priklauso nuo panaudotos technologinės bibliotekos, atliekant eksperimentą kitomis sąlygomis bus gauti kiti sintezės rezultatai.

5 Išvados

1) Remiantis atlikta ALU architektūrų analize identifikuotos tokios ALU charakteristikų gerinimo strategijos:
   - Didesnio laipsnio resursų dalinimasis padidina energijos vartojimą;
   - Daugybos pakeitimas postūmio ir sudėties operacijomis padidina energijos vartojimą;
   - Efektyvu daryti gerai suprojektuotus, lanksčios ir komplektuojamos (iš mažesnių modulių) architektūros ALU;
   - Naudojant išankstinio suapvalinimo metodiką apvalinimo procesą galima pagreitinti 51%.
2) Realizuotas dviejų pakopų operacijų sudalinimo ALU modelis. *SystemC* aprašyti 9 šio modelio variantai. Ištirta:
   - Dažnai pasikartojančių operacijų optimizavimo įtaka ALU charakteristikoms;
   - Daugintuvo realizavimo įtaka ALU charakteristikoms;
   - Loginių operacijų išreiškimo per tarpines operacijas įtaka ALU charakteristikoms;
   - Aritmetinės atimties operacijos atsisakymo įtaka.
3) Sukurti bendriniai ALU modeliai parametrizuoti pagal operandų plotį, panaudojant metaprogramavimo metodus. Tai leidžia vartotojui atlikti įtaiso parametrų konfigūravimą ir nemodifikuojant modulio aprašo gauti naują įrenginio variantą pritaikytą prie konkrečių, vartotojo reikalavimų.
4) Remiantis atliktais eksperimentais formuluojamos tokios išvados:
   - Patvirtinta [5] straipsnyje pateikta išvada, kad dažnesnis operacijų poaibio naudojimas neveda prie energijos vartojimo mažėjimo;
   - Patvirtinta [5] straipsnyje pateikta išvada, kad daugybos pakeitimas postūmio ir sudėties operacijomis neveda link sumažėjusio energijos išsklaidymo, jis padidėjo 51%;
   - Dažnai naudojamos operacijos perkėlimas į rečiau naudojamų operacijų poaibį įtakoja lusto ploto ir ventilių skaičiaus mažėjimą atitinkamai po 11% ir 15%;
   - Dažnai naudojamos operacijos perkėlimas į rečiau naudojamų operacijų poaibį įtakoja energijos vartojimo padidėjimą 13-15%;
   - Per tarpines operacijas valdymo logikos pakopoje realizuojamus veiksmus perkėlus į ALU pakopą ir atliekant vieno žingsnio įtaisu pagerėja visos pagrindinės lusto charakteristikos;

- Daugintuvo realizavimas tiek per tarpines tiek per tiesiogines ALU operacijas sąlygoja visų lusto charakteristikų pablogėjimą apie 50%. Realizuojant daugintuvą tiesiogine ALU operacija charakteristikų pablogėjimas nėra toks žymus (apie 35%) dėl rečiau naudojamo operacijų poaibio;
- Papildomos operacijos įterpimas valdymo pakopoje, panaudojant jau turimą ALU vieno ciklo operaciją įtakoja 4% įtaką lusto ventilių, laidų skaičiaus ir vėlinimo padidėjimą;
- Atimties operaciją optimaliau realizuoti per sudėtinę sudėties-invertavimo operaciją, nes smarkiai pagerėja dvi pagrindinės charakteristikos: vėlinimas (13%) ir energijos suvartojimas (14%);

5) Optimalus ALU operacijų aibės sudalinimas dviejų pakopų architektūroje leidžia laimėti reikiamos projektuojamo įtaiso charakteristikos pagerėjimą. Taikant šį metodą, galima kritinei (daug aparatūros resursų naudojančiai) ALU operacijai sudaryti mažesnį plotą luste užimantį arba mažiau energijos suvartojantį ALU variantą.

6 Literatūra

[1] Logic Devices Incorporated. *L4C383: 16-bit Cascadable ALU*. [žiūrėta 2005-12-11]. Prieiga per internetą: <http://www.logicdevices.com/ps/mma/l4c383.htm>.

[2] Philips. *74HC/HCT181 4-bit arithmetic logic unit*. [žiūrėta 2005-12-14]. Prieiga per internetą: <http://www.montefiore.ulg.ac.be/~pw/cours/struct-tp/files/74HC_HCT181.pdf>.

[3] Intersil. *CD40181BMS – CMOS 4 bit Arithmetic Logic Unit.* [žiūrėta 2005-12-14]. Prieiga per internetą: <http://www.tranzistoare.ro/datasheets/150/108783_DS.pdf>.

[4] Zarlink Semiconductor. *PDSP1601/PDSP1601A ALU and Barrel Shifter*. [žiūrėta 2005-12-14]. Prieiga per internetą: <http://assets.zarlink.com/DS/zarlink_PDSP1601A_NOV_98.pdf>.

[5] KALYANARAMAN V., et al. *A Power Dissipation Comparison of ALU-Architectures for ASIPs*. [žiūrėta 2006-01-04]. Prieiga per internetą: <http://ieeexplore.ieee.org/iel5/10211/32579/01523032.pdf?arnumber=1523032>.

[6] SUN H., GAO M. *Two-staged ALU architecture*. Institute of VLSI Design, 2002. [žiūrėta 2007-04-02]. Prieiga per internetą: <http://ieeexplore.ieee.org/iel5/8405/26512/01181077.pdf>.

[7] YAMADA H., et al. *A 13.3ns Double-precision Floating-point ALU and Multiplier*. IEEE 1995. [žiūrėta 2007-04-02]. Prieiga per internetą: <http://ieeexplore.ieee.org/iel3/4053/11610/00528909.pdf?arnumber=528909>.

[8] *Synopsys Synthesis Tutorial*. San Jose State University. [žiūrėta 2007-04-14]. Prieiga per internetą: <http://www.engr.sjsu.edu/tle/271_Syn_tut.pdf>.

[9] *Arithmetic logic unit*. Wikipedia. 2007 [žiūrėta 2007-04-16]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Arithmetic_logic_unit>.

[10] *SystemC*. Wikipedia. 2007 [žiūrėta 2007-04-16]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Systemc>.

[11] GWENNAP L. *MicroTech Watch: Power Limits Hit Processor Design.* Nikkei Electronics Asia, 2003. [žiūrėta 2007-04-16]. Prieiga per internetą: <http://techon.nikkeibp.co.jp/NEA/archive/200312/278986/>.

[12] YAROM I. *Three Different Usages of SystemC in Chip Design*. Intel 2003. Prieiga per internetą: <http://www.cs.huji.ac.il/~jarom/papers/SystemC_SNUG03.pdf>.

[13] SCHEPENS C. *Fast (near-)cycle-accurate SystemC Simulations*. 2004. [žiūrėta 2007-04-27]. Prieiga per internetą: <http://obelix.informatik.uni-tuebingen.de/~systemc/Documents/Presentation-10-SF_2_schepens.pdf>.

[14] DeGROAT J., RAMAN A., YOUNIS B. *A design project for system design with systemC*. Microelectronic Systems Education, 2003. Proceedings. IEEE International Conference 2003: 108 – 109.

[15] *System-on-a-chip*. Wikipedia. 2007 [žiūrėta 2007-04-27]. Prieiga per internetą: <http://en.wikipedia.org/wiki/System-on-a-chip>.

[16] *Embedded systems*. Wikipedia. 2007 [žiūrėta 2007-04-27]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Embedded_systems>.

[17] OZAWA M., et al. *Performance Evaluation of Cascade ALU Architecture for Asynchronous Super-Scalar Processors*. ASYNC 2001: 162-172.

[18] GARSIDE J.D. *A CMOS VLSI Implementation of an Asynchronous ALU*. Asynchronous Design Methodologies 1993: 181-192.

[19] HOGG R.S., HUGHES W.I., LLOYD D.W. *A Novel Asynchronous ALU for Massively Parallel Architectures*. PDP 1996: 282-289.

[20] PARASHAR A., GURUMURTHI S, SIVASUBRAMANIAM A. *A Complexity-Effective Approach to ALU Bandwidth Enhancement for Instruction-Level Temporal Redundancy*. ISCA 2004: 376-386.

[21] KECKLER S.W., et al. *Exploiting Fine-grain Thread Level Parallelism on the MIT Multi-ALU Processor*. ISCA 1998: 306-317.

[22] CHANG A., et al. *The Effects of Explicitly Parallel Mechanisms on the Multi-ALU Processor Cluster Pipeline*. IEEE International Conference on Computer Design 1998: 474.

[23] Knowledge Representation Laboratory. *Equivalent Transformation*. 2004. [žiūrėta 2007-05-16]. Prieiga per internetą: <http://kr.cs.ait.ac.th/Equivalent_Transformation>.

[24] AKAMA K., KOIKE H,. ir MABUCHI H. *Equivalent Transformation by Safe Extension of Data Structures*. Lecture Notes In Computer Science 2001: 140 – 148.

Priedai

Priedas 1: sintezės rezultatų bendra lentelė

29 lentelė. Sintezės rezultatų bendra lentelė

Modelis	Ventilių sk.	Laidų sk.	Prievadų sk.	Plotas, μm^2	Vėlinimas ns	Energija, mW
Bazinis	388	410	25	767,33	6,07	132,98
Bazinis, invertavimas realizuotas CU	338	360	25	690,74	6,36	153,18
XOR perkelta į ALU	369	391	25	713,77	5,66	118,00
XOR perkelta į ALU, invertavimas į CU	347	369	25	685,99	5,44	150,73
Iš ALU pašalinamos visos nenaudojamos instrukcijos	301	322	25	591,31	5,49	142,50
Bazinis, ALU papildytas daugybos operacija	525	572	25	1014,82	6,00	147,23
Bazinis, daugyba išreikšta per ADD ir SHFL	573	626	25	1196,44	5,79	201,22
Bazinis, CU papildytas atimtimi	403	425	25	771,29	6,32	152,55
Bazinis, atimtis realizuota per sudėtį ir invertavimą	400	421	25	775,99	5,39	116,90

Priedas 2: pavyzdinis programos kodas

Typelib.h failas – operacijų aibės, globalūs kintamieji ir parametrai.

```
#include "systemc.h"
#pragma enum small
#ifndef some
#define some

enum opera
{op_lda,op_ldm,op_ldi,op_ldii,op_and,op_or,op_add,op_sub,op_inc,op_dec,op_not,op_shfl,op_shfr,op_
nop};
enum insta {n_aab,n_axb,n_aob,asnb,nasb,naab,aanb,asbsc,nasbsc,asnbsc,adsc,bdsc};
 //wait:       2     4     2    2    2    2    2     2     3      3     2    2

/* !(A AND B) n_aab
   !(A XOR B) n_axb
   !(A OR B)  n_aob
   A + !B     asnb
   !A + B     nasb
   !A AND B   naab
   A AND !B   aanb
   A + B + CI asbsc
   !A + B + CI             nasbsc
   A + !B + CI             asnbsc
   A - 1 + CI adsc
   B - 1 + CI bdsc
*/
  const int L = 8;  //mano universali konstanta
#endif
```

cu.cc failas – valdymo logika, aprašyta būsenų automatu. Pateikiamas daugybos operacijos išreikštos per postūmį ir sudėtį fragmentas:

```
case amb:                                                  //operacija A*B
         {cout << "CU OPERACIJA: amb" << endl;
          cout << "amb state: " << state << endl;
          switch (state) {case 1:                          //vidinis busenu automatas: busena [1]
                               {i = 0;
                                opSu = 0;
                                tempA = bus_in.read();  //operandas "tempA"=prad bus_in reiksme
                                tempB = reg_in.read();  //operandas "tempB"=prad reg_in reiksme
                                op_l = op_nop;          //ALU operacijos kodas
                                cout << "operacija NOP" << endl;
                                cout << " opSu: " << opSu  << endl;
                                state=2;                //sekanti busena
                               }
                               break;
                          case 2:                          //busena [2]
                               {//cout << "tempB[i]: " << tempB[i] << endl;
                                if ( tempB[i] == true )      //naudojamas laikinas B
                                   {opA = tempA;             //operandas "A" = turima "A" reiksme
                                    opB = opSu.range(L-1,0); //operandas "B" = kaupiama suma
                                    op_l = op_add;           //ALU operacijos kodas
                                    cout << "operacija ADD" << endl;
                                    state=4;                 //sekanti busena
                                   }
                                else                         //kai B bitukas "0" - tik stumimas
                                   {opA = tempA;             //operandas "A" = turima "A" reiksme
                                    op_l = op_shfl;          //ALU operacijos kodas
                                    cout << "operacija SHFL" << endl;
                                    state=3;                 //sekanti busena
                                   }
                                cout<<"opSu: "<<opSu<<" tempA: "<<tempA<<" tempB: "<<tempB<<endl;
                                i = i + 1;                   //didinamas ciklo kintamasis
                               }
                               break;
                          case 3:                          //busena [3]
                               {tempA = opR.read();        //A = A po SHIFT_LEFT operacijos
                                op_l = op_nop;             //ALU operacijos kodas
                                cout << "operacija NOP" << endl;
                                if (i == L)                //for i=0 to L
                                  state = 5;               //paskutine busena
                                else
                                  state=2;                 //sekanti busena
                               }
                               break;
                          case 4:                              //busena [4]
                               {opSu = ("00000000",opR.read());//suma gauna ALU rezultata
                                opA = tempA;                   //operandas "A"=turima "A" reiksme
                                op_l = op_shfl;                //ALU operacijos kodas
                                cout << "operacija SHFL" << endl;
                                state=3;                       //sekanti busena
                               }
                               break;
                          case 5:
                               {cu_ready=1;             //CU jau turi rezultata
                                op_l = op_nop;          //ALU operacijos kodas
                                //opR = sM;
                                cout << "ALU rezultatas:" << opSu << endl;
                                bus_in.write(opSu.range(L-1,0));//atiduodamas rezultatas
                                state=1;                //sekanti busena vel [1]
                               }
                               break;
                          default:                         //busena when others...
                               {state=1;                  //senakti busena [1]
                                cu_ready=0;               //cu nepasiruoses
                                op_l = op_nop;            //ALU atlieka no-operation
                                cu_ready=0;               //cu nepasiruoses
                               }
                         }
         }
         break;
```

test_cu.cc failas – valdymo logikos testo failas:

```
#include "test_cu.h"
//------------------------------------------------
/*
 * Testbench functionality for Control Unit
 */
char* cu_names[] =
{"n_aab","n_axb","n_aob","ansb","nasb","naab","aanb","asbsc","nasbsc","asnbsc","adsc","bdsc","amb
"};
//------------------------------------------------
void test_cu::wait(int num_clocks)
{for(int i = 0; i < num_clocks; i++)
   sc_module::wait();
}//eo WAIT
//------------------------------------------------
void test_cu::test_me()
{cu_en = 1;
 reset = 0;
 bus_in.write(10);
 reg_in.write(3);
 ci.write(1);

 ins = bdsc;
 wait(2);
 ins = n_aab;
 wait(3);
 reset = 1;
 wait(1);
 reset = 0;
 ins = n_axb;
 wait(3);

 ins = nasbsc; //priedas testui
 wait(2);

// bus_in.write(54);
// reg_in.write(2);
 bus_in.write(255);
 reg_in.write(0);

 ins = amb;    //priedas daugybos operacijai
 wait(14);
 sc_stop();
}//eo TEST_ME
//------------------------------------------------
void test_cu::check_results()                 //metodas parodantis ekrane esamas kintamuju
reiksmes
{sc_uint<L> bin,rin,aou,ouA,ouB;              //lokalkes, bin 2 dec konvetavimui
 bin = bus_in.read();                         //konvertavimas bin 2 dec
 rin = reg_in.read();
 aou = acc_in.read();
 cout << endl;                                //formuojama eilute ekrane
 cout << "rst=" << reset.read() << " ";
 cout << "cu_en=" << cu_en.read() << " ";
 cout << "ins=" << cu_names[ins] << " ";
 cout << "bus_in=" << bus_in.read() << "(" << bin << ") ";
 cout << "reg_in=" << reg_in.read() << "(" << rin << ") ";
 cout << "acc_in=" << aou << " ";
 cout << "cu_ready=" << cu_ready.read();
// cout << "flag=" << flag.read() << " ";
 cout << endl;
}//eo CHECK_RESULTS
//------------------------------------------------
```

test_alu.cc failas – ALU testo failas, 8 bitų pavyzdys:

```
#include "test_alu.h"
//------------------------------------------------
/* Testbench functionality  */
char* names[] =
{"op_lda","op_ldm","op_ldi","op_ldii","op_and","op_or","op_add","op_sub","op_inc","op_dec","op_no
t","op_shfl","op_shfr","op_nop"};
```

```
//-----------------------------------------------
void test_alu::wait(int num_clocks)
{for(int i = 0; i < num_clocks; i++)
   sc_module::wait();
}//eo WAIT
//-----------------------------------------------
void test_alu::test1()
{acc_en = 1;
 alu_en = 1;
 reset = 0;
wait(1);
 reset = 1;
wait(1);
 reset = 0;
 reg_in = "00010100";
 bus_in = "00110011";
 op = op_ldm;
wait(1);
 op = op_lda;
wait(1);
 reg_in = "00001111";
 op = op_ldi;
wait(1);
 op = op_ldii;
wait(1);
 op = op_or;
wait(1);
 op = op_and;
wait(1);
 op = op_sub;
wait(1);
 op = op_add;
wait(1);
 op = op_inc;
wait(2);
 op = op_dec;
wait(1);
 op = op_shfr;
wait(1);
 op = op_shfl;
wait(2);
 alu_en = 0;
wait(1);
 reset = 1;
wait(1);
 reset = 0;
 op = op_lda;
wait(1);
 op = op_shfl;
wait(2);
 sc_stop();
}//eo TEST1
//-----------------------------------------------
void test_alu::check_results()
{sc_int<L> bin,rin,aou;
 bin = bus_in.read();
 rin = reg_in.read();
 aou = acc.read();
 cout << "rst=" << reset.read() << " ";
 cout << "alu_en=" << alu_en.read() << " ";
 cout << "acc_en=" << acc_en.read() << " ";
 cout << "op=" << names[op] << " ";
 cout << "bus_in=" << bus_in.read() << "(" << bin << ") ";
 cout << "reg_in=" << reg_in.read() << "(" << rin << ") ";
 cout << "acc=" << acc.read() << "(" << aou << ") ";
 cout << "flag=" << flag.read() << " ";
 cout << endl;
}//eo CHECK_RESULTS
//-----------------------------------------------
```

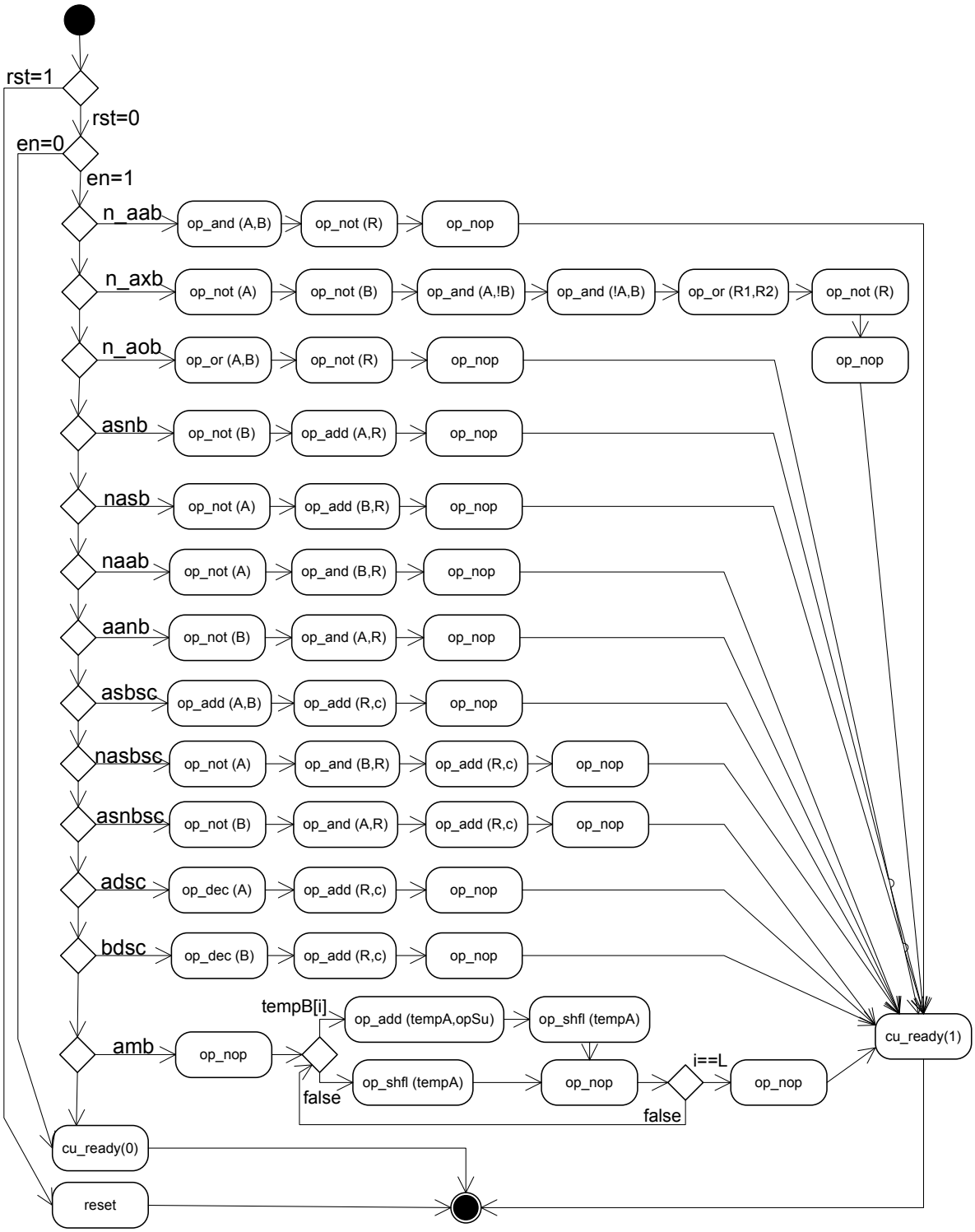
cu.scr failas – CU sintezės scenarijaus failas

```
search_path       = search_path + "$SYNOPSYS/libraries/syn"
target_library    = {"tc6a_cbacore.db"};
symbol_library    = {"tc6a_cbacore.sdb"};
synthetic_library = {"dw01.sldb","dw02.sldb"}
```

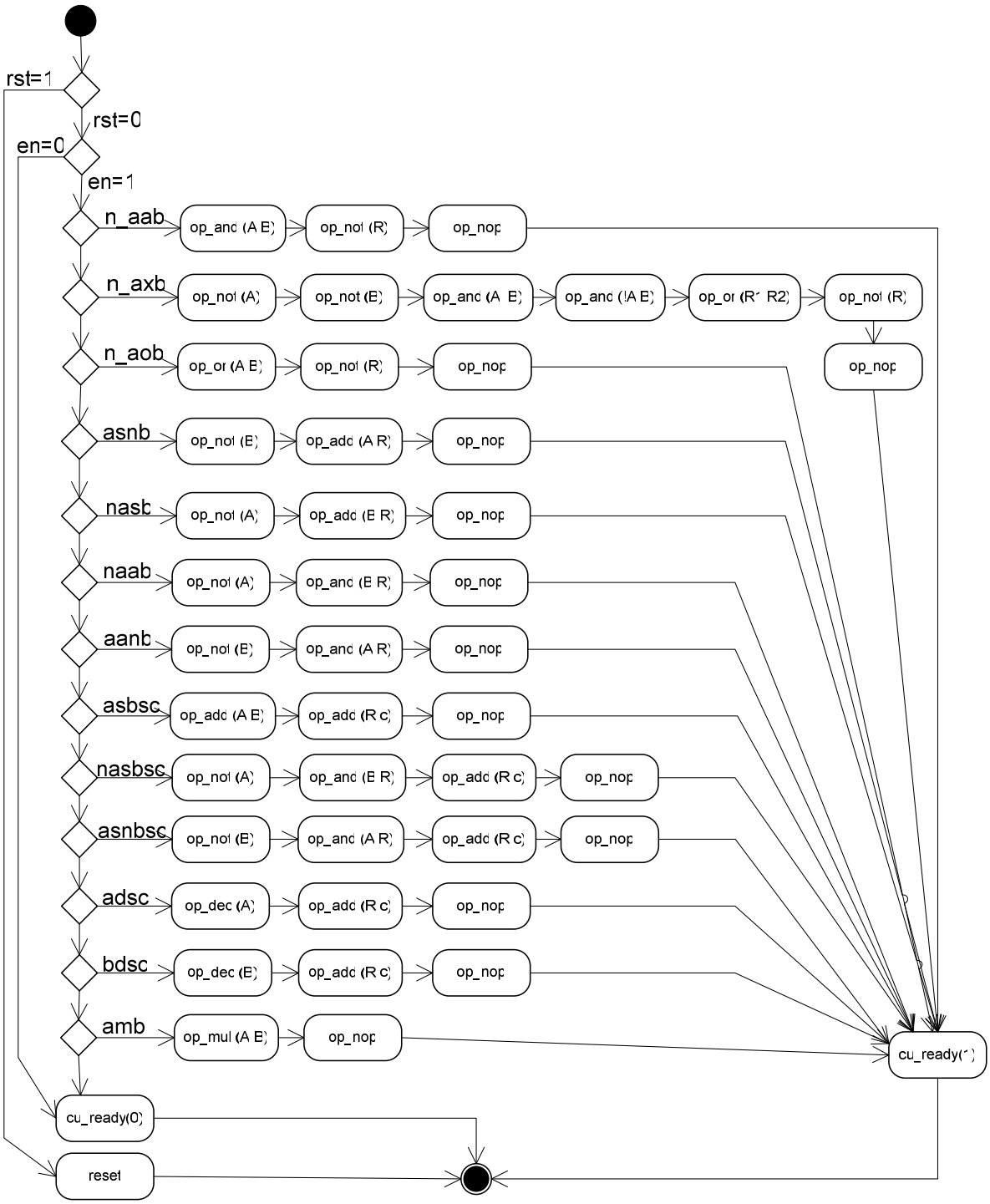
```
link_library      = {"*"} + target_library + synthetic_library
/* BUTINA NURODYTI */
top_unit          = "cu"
/*************************************************************/
/* RTL Level                                                 */
/*************************************************************/
sh date
/*
compile_systemc -rtl dff1.cc -rtl_format db
*/
/*
compile_systemc -rtl -cpp "/usr/local/bin/g++ -trigraphs -E -C -U__GNUC__
-U__GNYG__ -Wp,-no-gcc,-pedantic "dff1.cc
*/
/* BUTINA NURODYTI */
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp,-pedantic "top_unit + ".cc"
-rtl -rtl_format db
create_clock clock -p 10
compile -map_effort medium

write -hier -f db -o "rtl_work/" + top_unit + ".db"
write -hier -f verilog -o "rtl_work/" + top_unit + "_gate.v"
report_timing > "rtl_work/" + top_unit + "_timing.rpt"
report_area > "rtl_work/" + top_unit + "_area.rpt"
quit
```

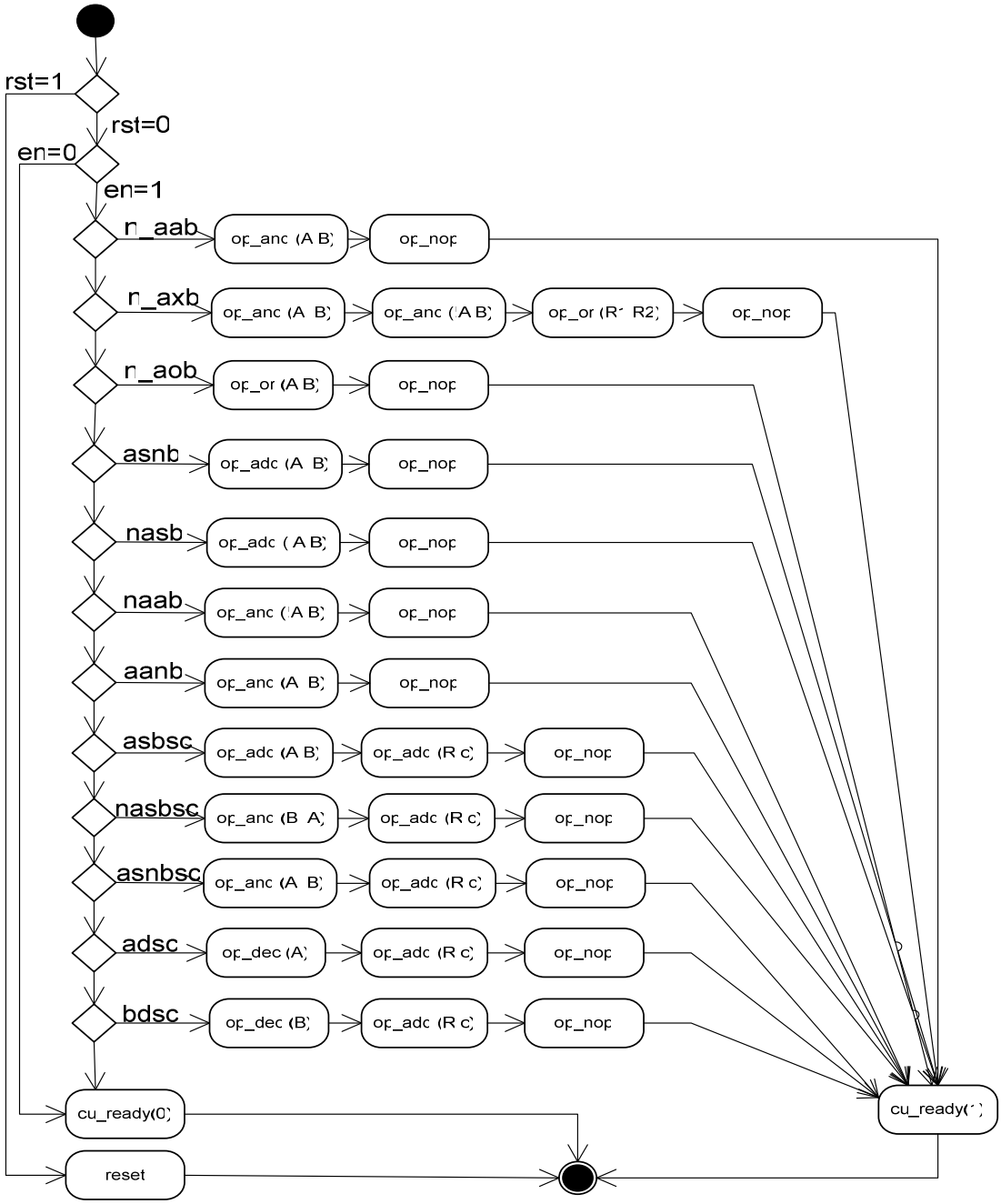
Priedas 3: kitų modelių pilni būsenų automatai



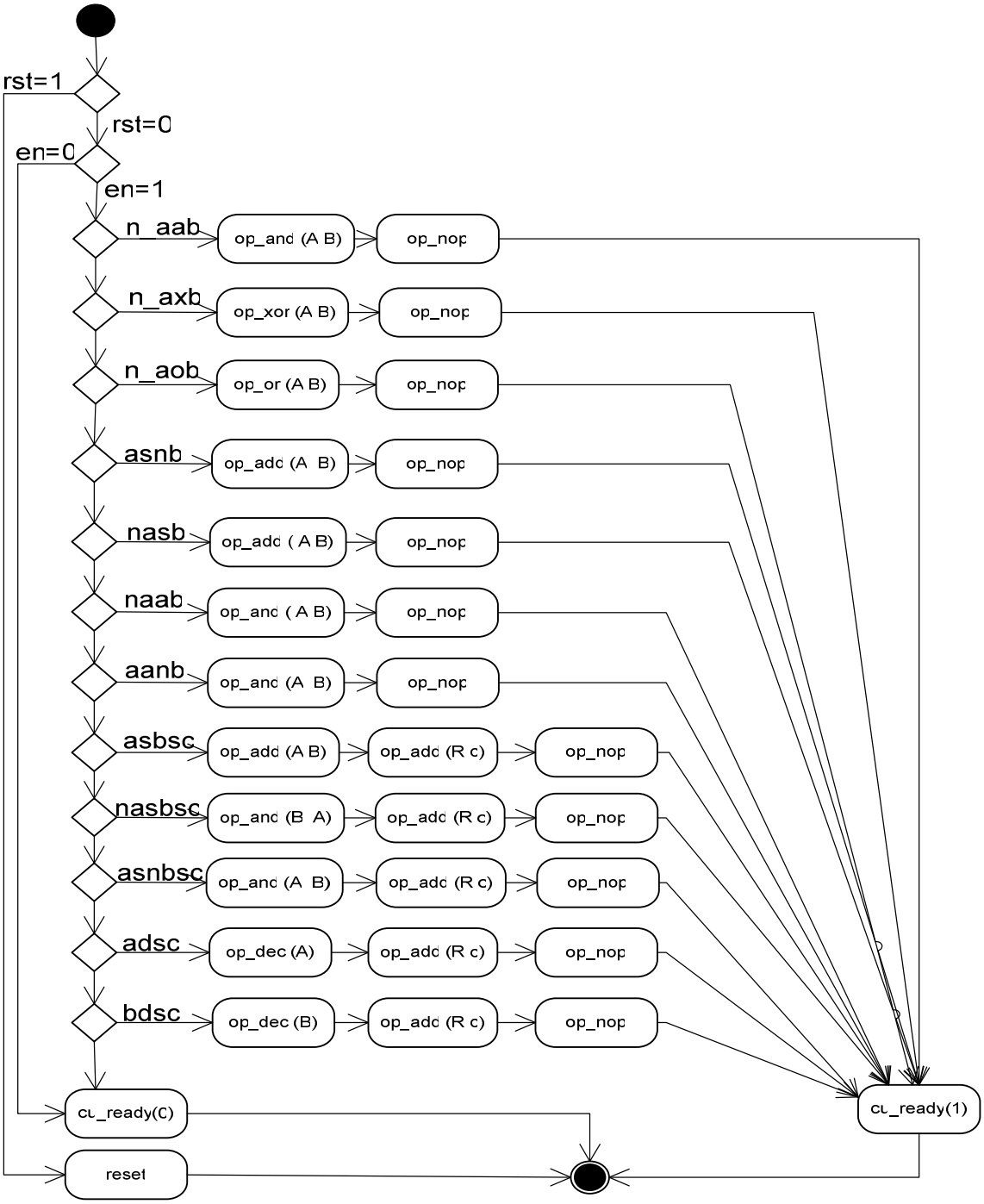
43 pav. *Bazinis, daugyba išreikšta per ADD ir SHFL modelio būsenų automatas*



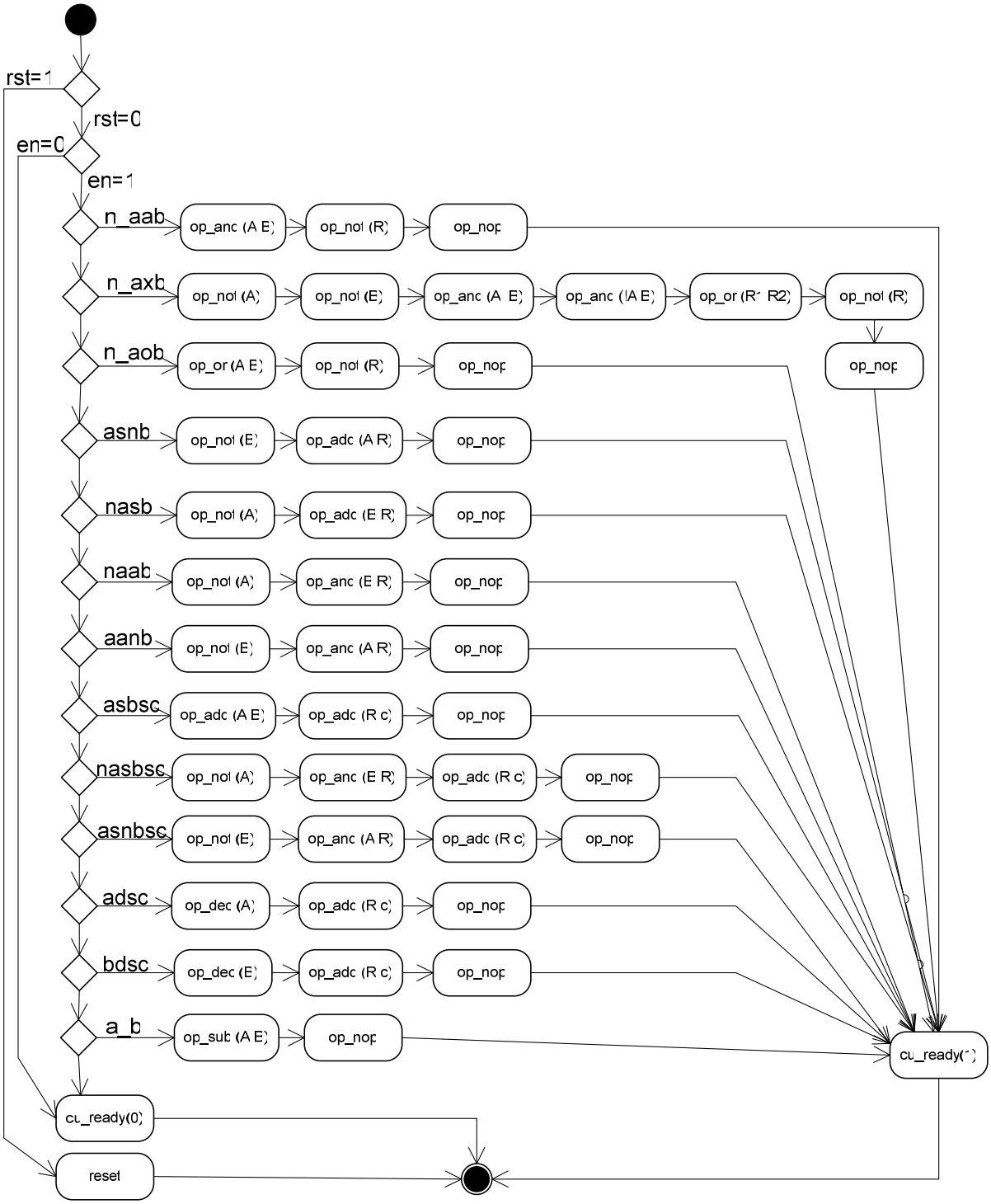
44 pav. *Bazinis, ALU papildytas daugyba, modelio būsenų automatas*



45 pav. *Iš ALU pašalinta operacija NOT. Modelio būsenų automatas*

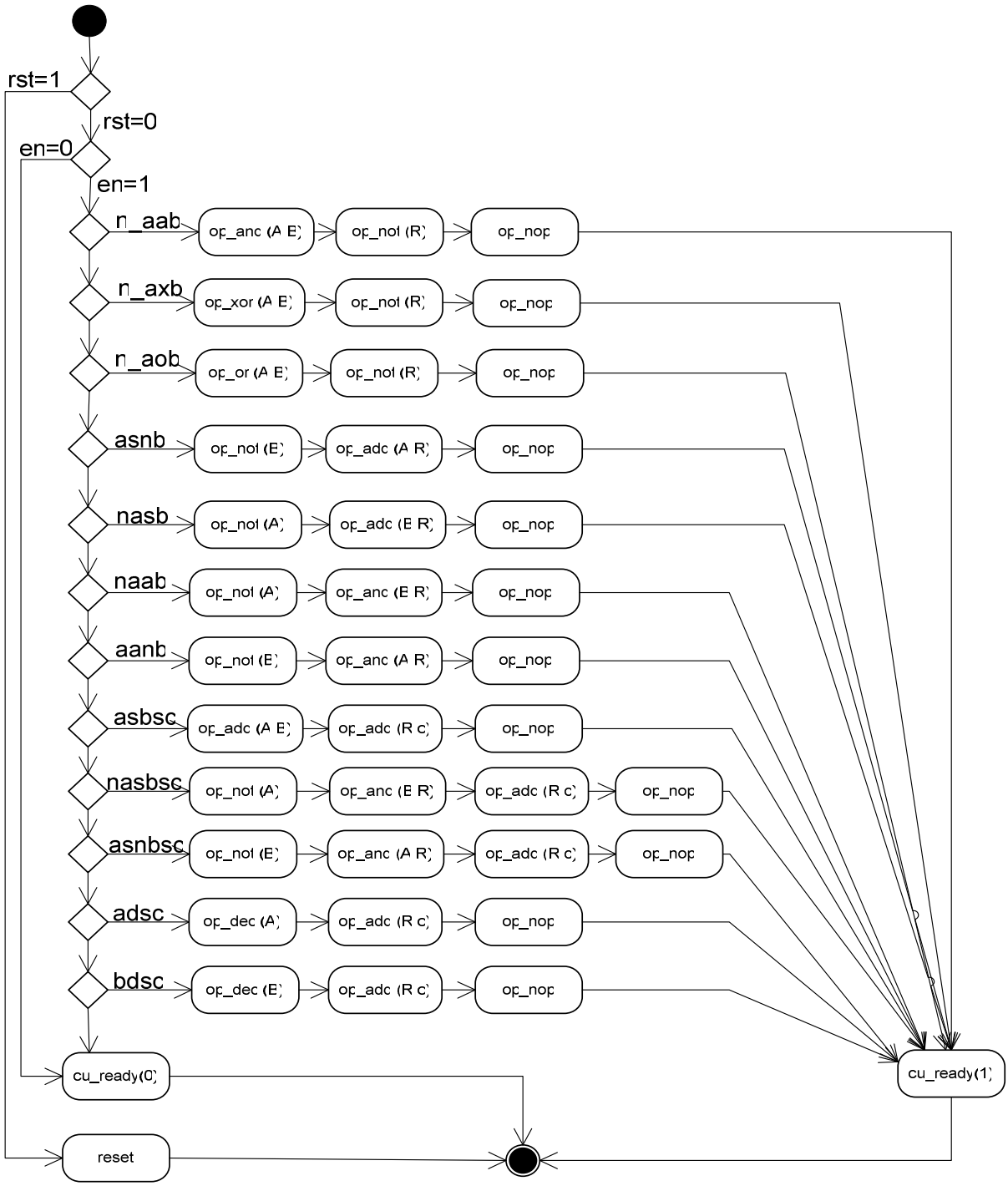


46 pav. *Iš ALU pašalinta operacija NOT, realizuota XOR. Modelio būsenų automatas*

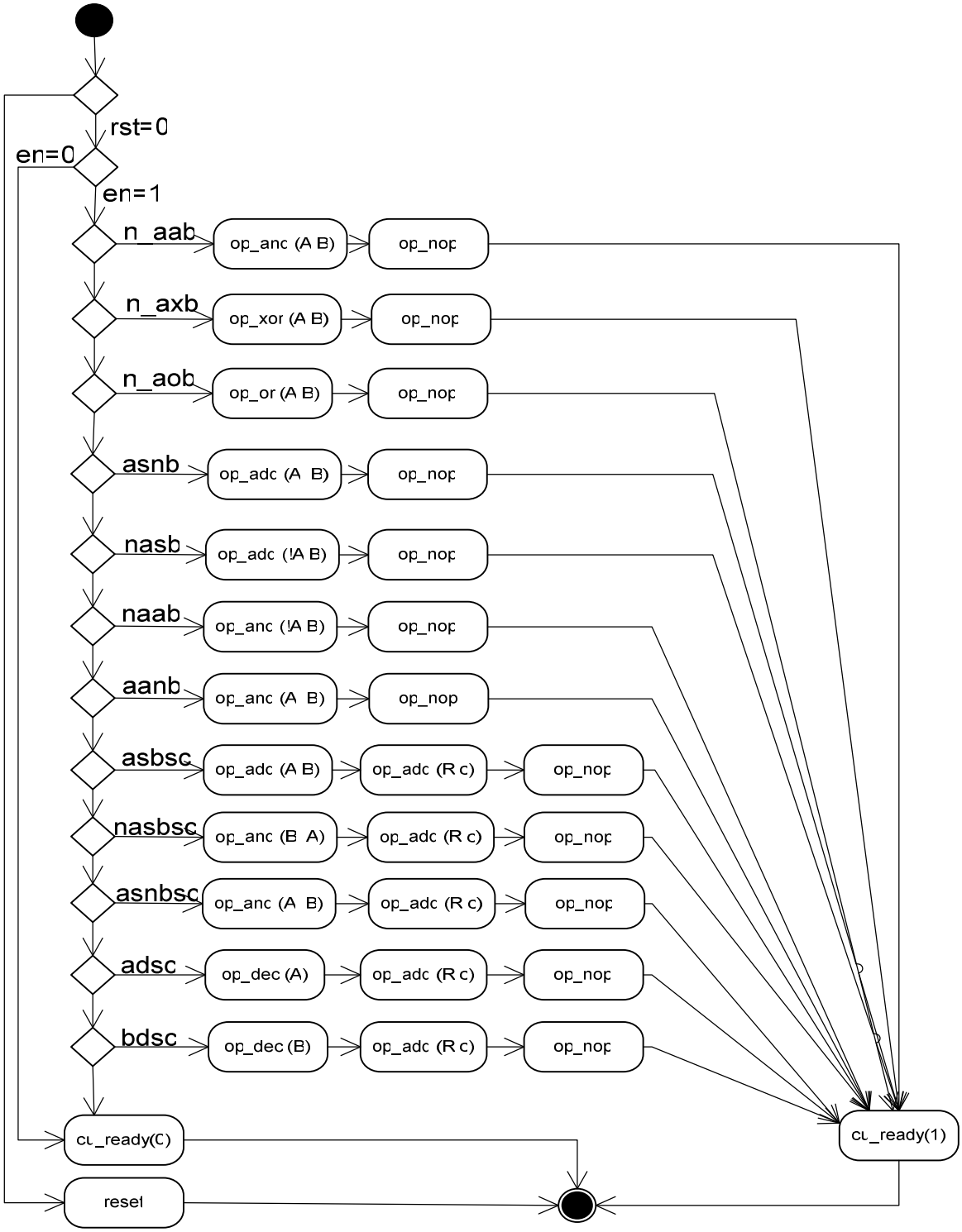


47 pav. *CU panaudoja ALU atimties operaciją. Modelio būsenų automatas*

48 pav. *CU papildytas atmintimi, realizuota per sudėtį. Modelio būsenų diagrama*

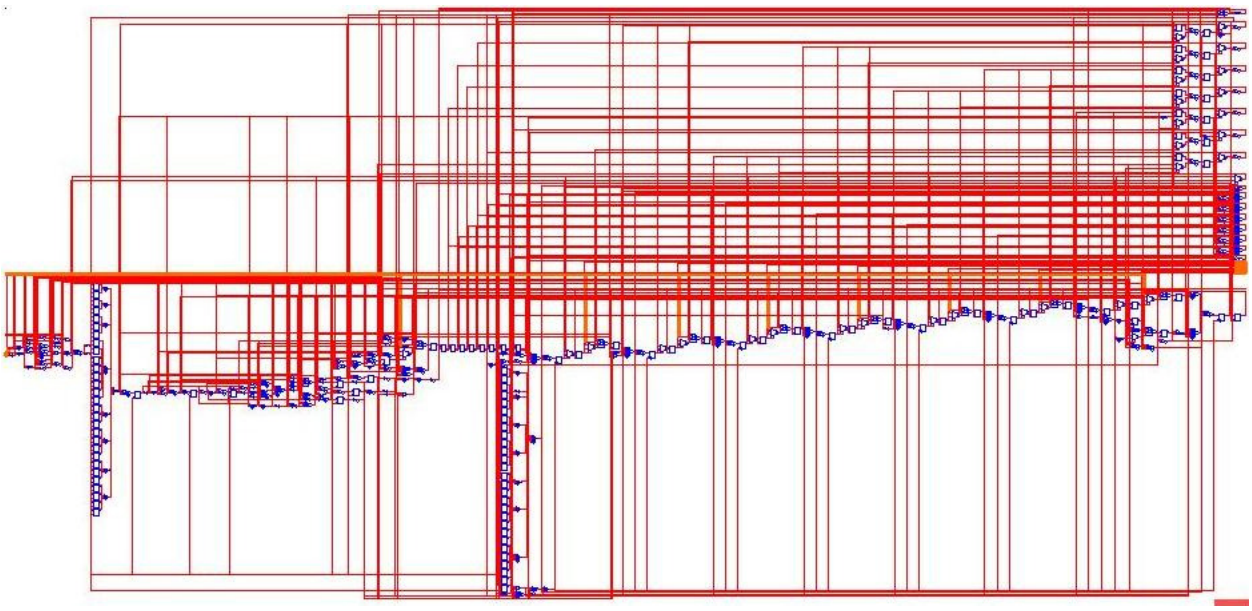


49 pav. *Bazinis, XOR operacija realizuota per ALU, modelio būsenų diagrama*

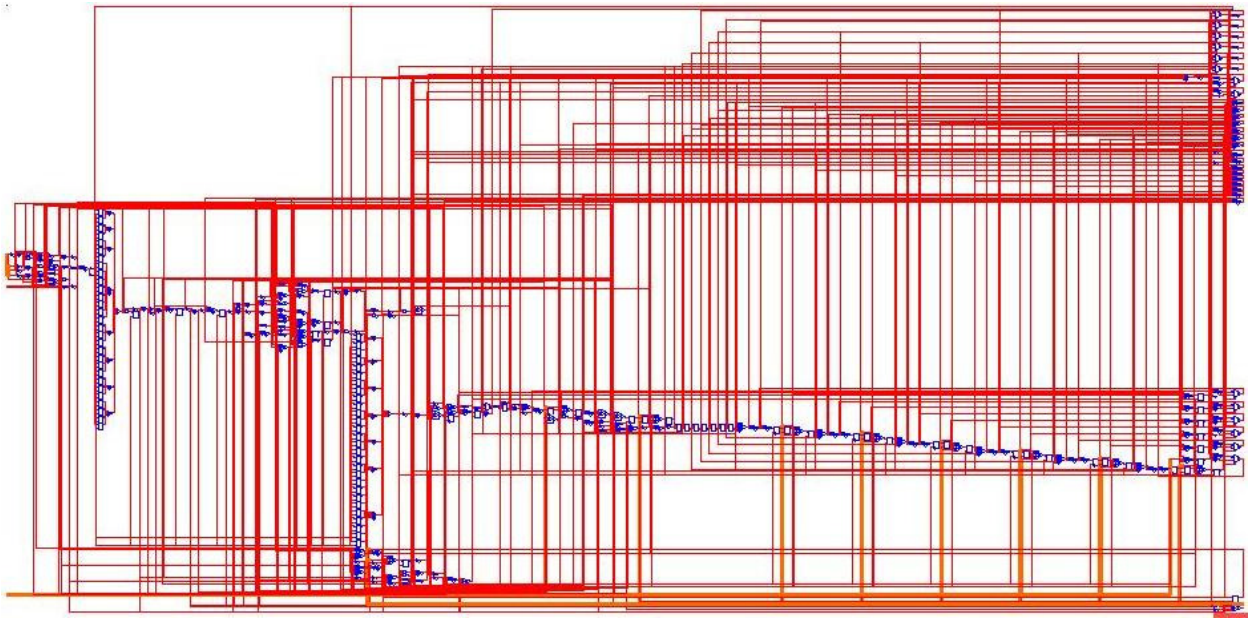


50 pav. *Pašalintos visos ALU nenaudojamos operacijos. Švaraus modelio būsenų automatas*

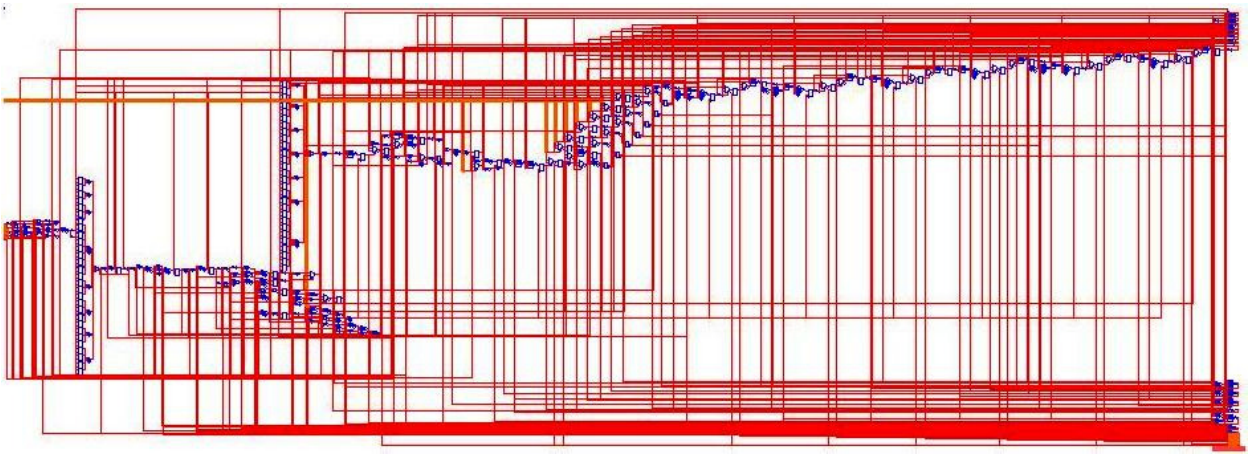
Priedas 4: kitų modelių sintezuotos schemos



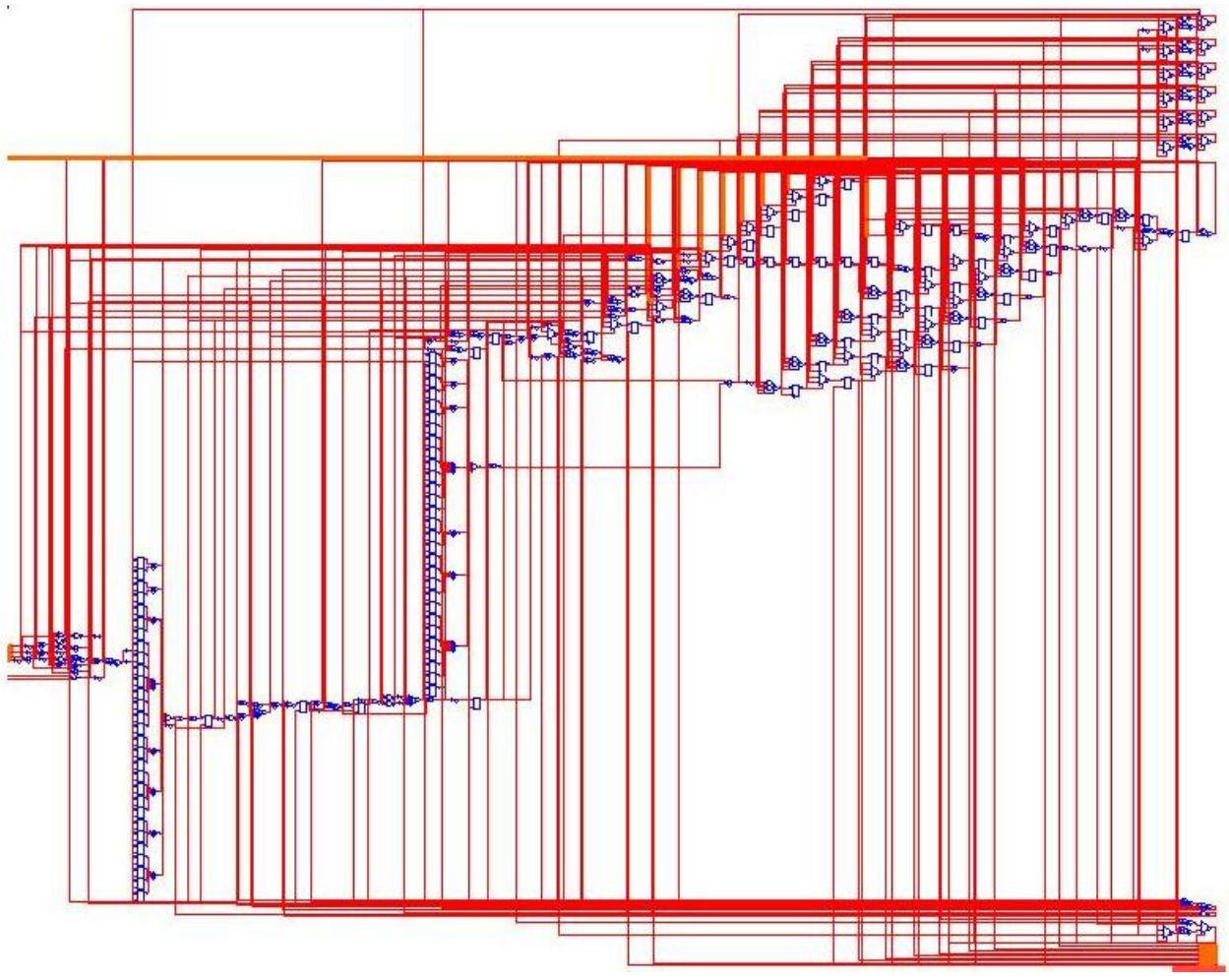
51 pav. *Bazinis, invertavimas realizuotas CU. 8 bitai*



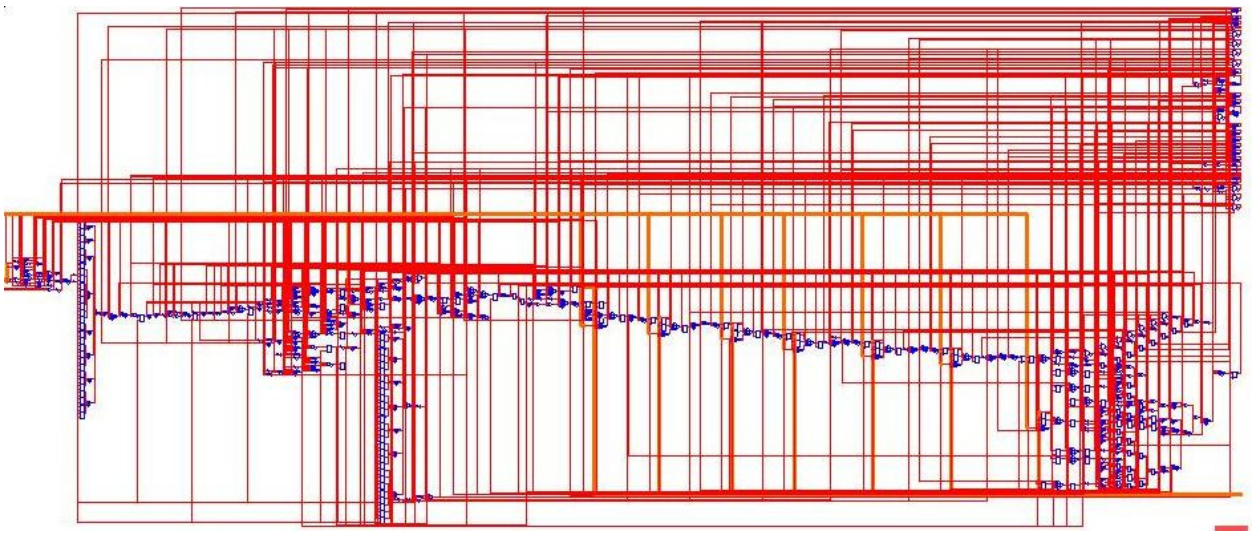
52 pav. *XOR operacija realizuota ALU. 8 bitai*



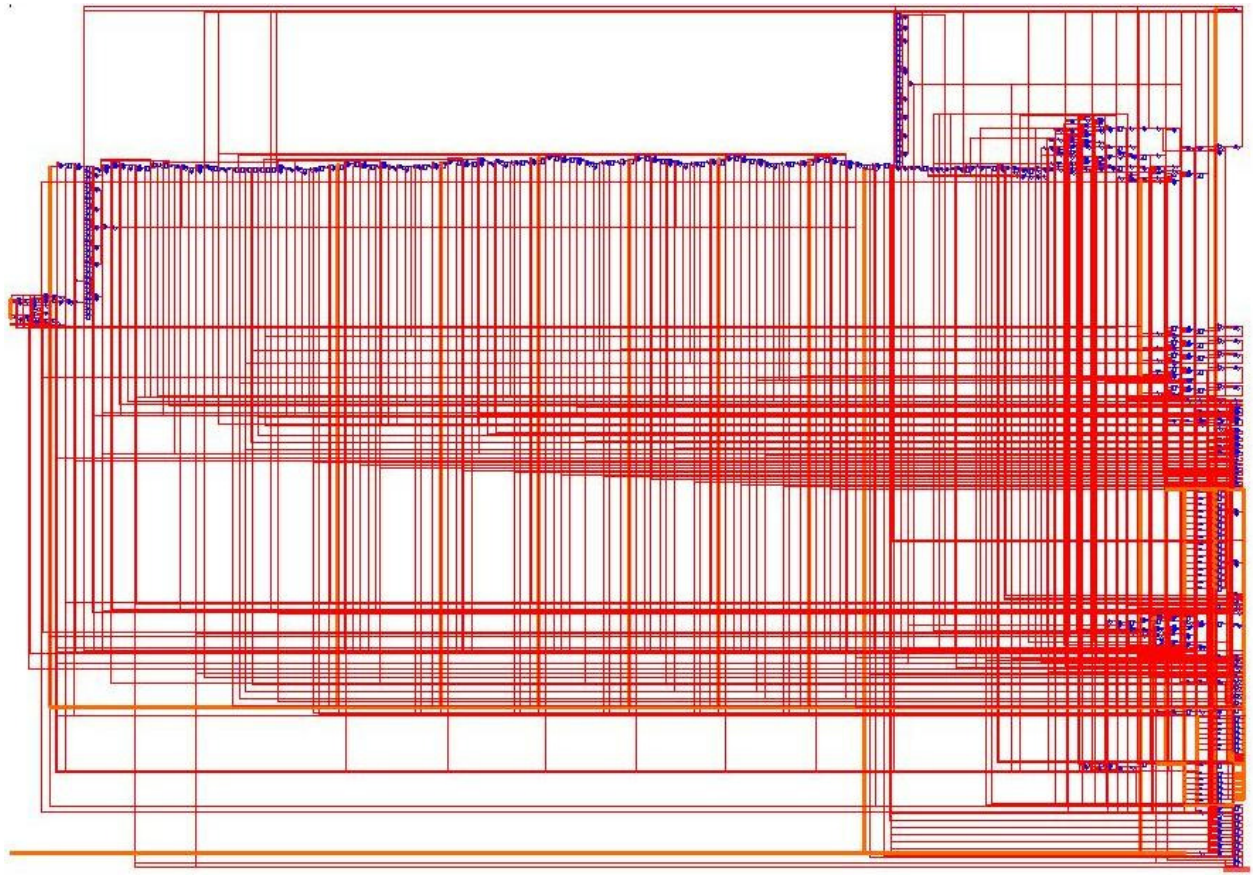
53 pav. *XOR operacija realizuota ALU, inversija CU. 8 bitai*



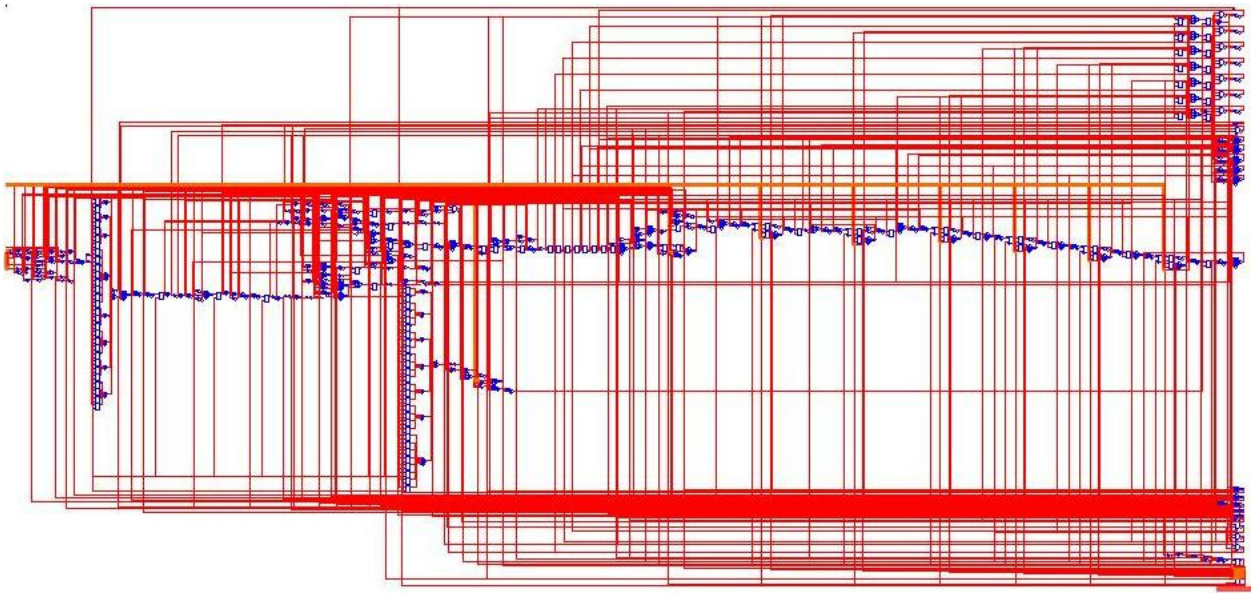
54 pav. *Iš ALU pašalinamos nenaudojamos operacijos, XOR realizuotas ALU, inversija CU. 8 bitai*



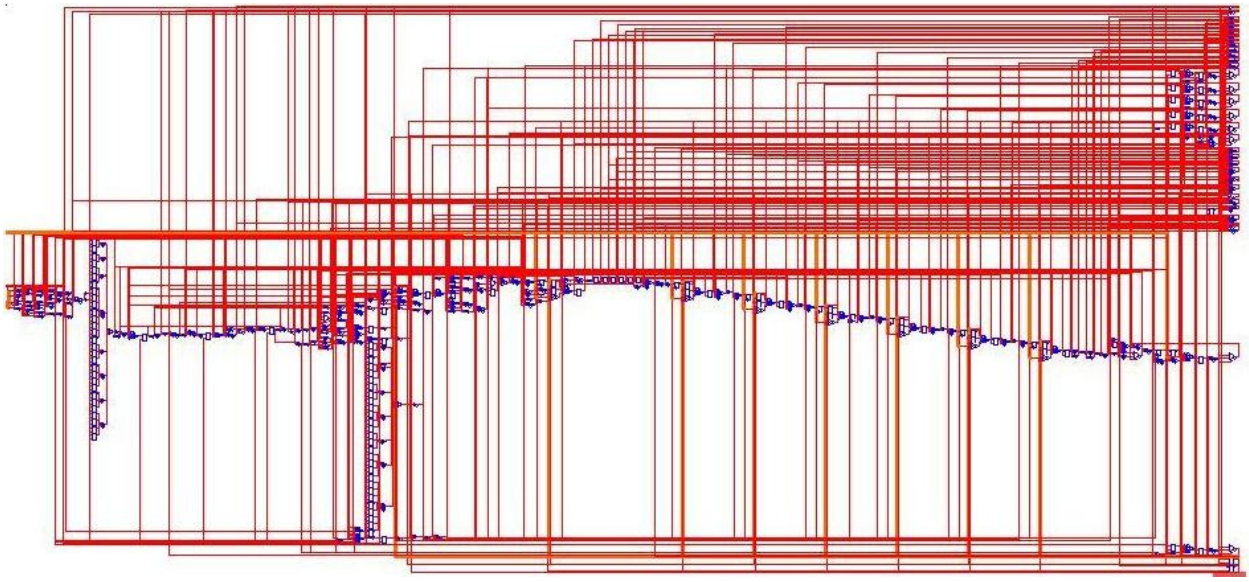
55 pav. *ALU papildytas daugybos operacija. 8 bitai*



56 pav. *Daugybos operacija realizuota per sumavimą ir postūmį kairėn. 8 bitai*



57 pav. *CU panaudoja atimties operaciją. 8 bitai*



58 pav. *CU panaudoja atimties operaciją, išreiškiamą per invertavimą ir sudėtį. 8 bitai*