KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Linas Žičevičius

Atvirojo kodo JPEG realizacijų, aprašytų aparatūros aprašymo kalbomis, tyrimas

Magistro darbas

Darbo vadovas:

Prof. Vacius Jusas

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

Linas Žičevičius

Atvirojo kodo JPEG realizacijų, aprašytų aparatūros aprašymo kalbomis, tyrimas

Magistro darbas

Recenzentas:

Dr. Darius Matulis

2012-05-

Darbo vadovas:

Prof. Vacius Jusas

2012-05-

Atliko:

IFM-0/5 gr. stud.

Linas Žičevičius

2012-05-

Kaunas, 2012

Research of Open Source JPEG Implementations Described in Hardware Description Languages

Summary

In this age of technological advances more and more technologies interact with environment. It is various devices, systems, robots which are capable to process and/or interpret visual information. Visual information is very space consuming information in comparison with other types of information, so problems like storing, downloading emerges and data compression is trying to resolve these problems. These data compression solutions may be implemented using software or hardware. Hardware solutions characterized with their high speed. In future more and more technologies will be capable to interact with environment, so tools capable of fast, precise and good compression render of images will become very important. These tools may be designed and studied using hardware description languages. Furthermore, visual information processing tools becoming very complex. Reuse methodology enables easier, faster and cheaper design of such tools.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

1. Įvadas

Šiame technologijų amžiuje vis daugiau technologijų sąveikauja su aplinka. Tai įvairūs prietaisai, sistemos, robotai, kurie geba apdoroti vaizdinę informaciją ir/arba ją interpretuoti. Kadangi vaizdinė informacija užima labai daug kietojo disko vietos, palyginus su kitokios rūšies informacija, iškyla jos saugojimo bei parsiuntimo problemos. Šioms problemoms spręsti pasitelkiamas duomenų glaudinimas (data compression). Šie sprendimai gali būti įgyvendinti programiškai (software) ir aparatūriškai (hardware). Aparatūriniai sprendimai pasižymi dideliu greičiu taip pat panaudojimo galimybėmis. Kadangi ateityje technologijos vis labiau sąveikaus su aplinka, įrenginiai, sugebantys greitai, tiksliai ir kuo mažesnėmis kietojo disko sąnaudomis apdoroti vaizdus, taps ypač reikšmingi. Tokius įrenginius patogu projektuoti ir tirti naudojantis aparatūros aprašymo kalbomis (angl. Hardware Description Language). Taip pat svarbus šių įrenginių aspektas – vis didėjantis jų sudėtingumas. Pakartotinio naudojimo (*reuse*) metodologija – naudojimas sukurtų ir verifikuotų komponentų – dabar yra kertinis lustų bei SOC (**S**ystem **O**n **C**hip) kūrimo akmuo, kadangi tai yra metodologija, kuri leidžia sudėtingus lustus projektuoti prieinama kaina, geresnės kokybės taip pat taupo žmogiškuosius išteklius ir laiką.

1.1. Darbo tikslas

Darbe atliekamas atvirojo kodo JPEG realizacijų aprašytų aparatūros aprašymo kalbomis įvairių aspektų tyrimas, taip pat tobulinimas. Pagrindiniai darbo tikslai:

- Ištirti įvairias JPEG realizacijas;
- Palyginti realizacijas tarpusavyje;
- Apžvelgti aparatūros aprašymo kalbų tinkamumą realizacijoms;
- Realizuoti pasirinktą JPEG realizacijos komponentą remiantis pakartotinio naudojimo metodologija;
- Sugeneruoti keletą komponentų naudojant įvairius (atsitiktinius) parametrus;
- Įvertinti komponentų parametrų įvairius aspektus ir juos palyginti.

2. Analizė

2.1. Duomenų kodavimas

Duomenų glaudinimas arba kodavimas, tai procesas, kurio metu turima informacija užkoduojama naudojant mažiau bitų (informacinėse technologijose) nei neužkoduotas atitikmuo, naudojant atitinkamas kodavimo schemas. Duomenų glaudinimas yra informacijos teorijos šaka, kurios pagrindinis tikslas sumažinti perduodamų duomenų kiekį. Žymūs kodavimo pionieriai K. Šenonas (angl. Claude Elwood Shannon), D. Hafmanas (angl. David Albert Huffman) [1].

Duomenų parsiuntimas ir saugojimas kainuoja pinigus. Kuo daugiau informacijos turime, tuo daugiau tai kainuoja. Nepaisant to, dauguma skaitmeninių formatų yra saugomi ne pačioje kompaktiškiausioje formoje. Dažniausiai jie saugomi tokioje formoje, kurioje paprasčiau juos naudoti, pavyzdžiui: ASCII tekstas, dvejetainis kodas. Paprastai tokie lengvai naudojami formatai užima kur kas daugiau vietos. Taigi kodavimas yra įvairių algoritmų ir programų sprendžiančių šią problemą apibūdinimas. Glaudinimo programa naudojama paversti iš lengvai naudojamo formato į optimizuotą, norint duomenis padaryti kompaktiškesnius, o išskleidimo programa priešingai – grąžina informaciją į jos originalią formą. Skiriamos dvi glaudinimo metodų rūšys: be nuostolių ir su nuostoliais [2].

2.1.1. Kodavimas be nuostolių

Be nuostolių (lossless) – tai tokia glaudinimo rūšis, kai atkurta informacija yra identiška originaliai informacijai. Šią glaudinimo rūšį privaloma naudoti koduojant daugelį skaitmeninių formatų, pavyzdžiui: tekstiniai dokumentai. Tokiai informacijai net vienas bitas ne savo vietoje, gali stipriai iškreipti duomenis [2]. Keletas glaudinimo metodų be nuostolių pavyzdžių:

- RLE;
- LZW;
- JPEG 2000;

- PNG;
- TIFF;
- WAV;
- FLAC.

2.1.2. Kodavimas su nuostoliais

Duomenys, kuriuose saugoma vaizdinė informacija nebūtinai turi būti idealūs saugojimui ir persiuntimui. Visi realūs signalai pasaulyje turi kažkokią dalį triukšmo. Taigi kodavimo metu įveltos klaidos atitinka triukšmą, todėl informacijoje esant nedideliam kiekiui triukšmo, nejaučiama žala informacijai. Glaudinimo metodai, kurie panaudoja tokias glaudimo technikas, kada atsiranda triukšmas vadinami su nuostoliais (lossy). Ši glaudinimo rūšis labai svarbi, kadangi yra kur kas efektyvesnė nei glaudinimas be nuostolių. Kuo didesnis glaudinimas - tuo daugiau atsiranda triukšmo duomenyse [2]. Keletas glaudinimo metodų su nuostoliais pavyzdžių:

- JPEG;
- H.264;
- MPEG;
- MP3;
- WMA.

Yra sukurta labai daug glaudinimo su nuostoliais metodų, tačiau grupė metodų, vadinamų keitimo glaudinimu (angl. transform compression) yra vieni vertingiausių, kadangi šie metodai pasižymi ypač dideliu glaudinimu. Vienas geriausių keitimo glaudinimo pavyzdžių, tai populiarus JPEG standartas vaizdo kodavimui.

2.2. JPEG kodavimas

2.2.1. JPEG standartas

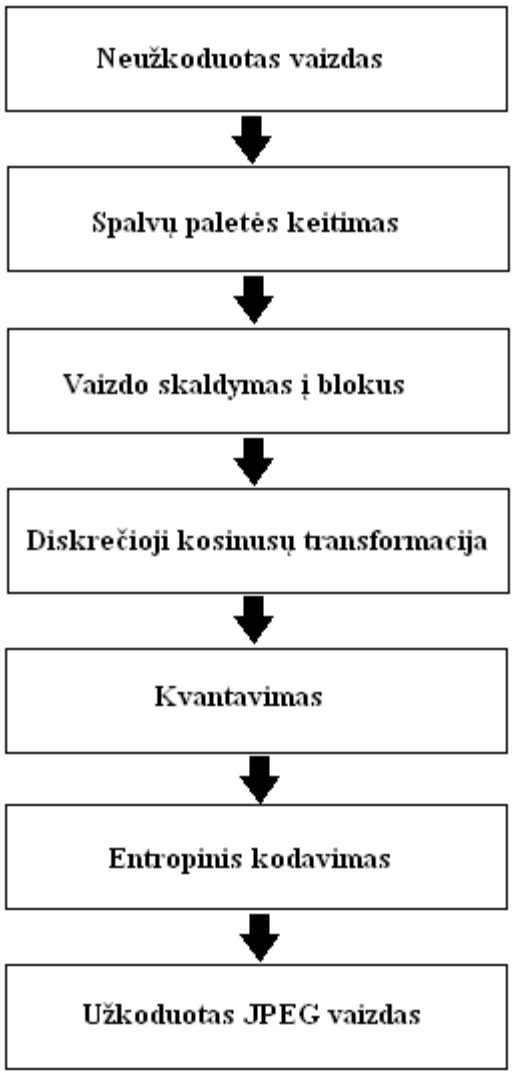
JPEG kodavimo standartas pavadintas pagal formato kūrėjų komitetą **J**oint **P**hotographers **E**xperts **G**roup. Šis formatas gali nesunkiai pasiekti 10:1 glaudinimą su labai nedideliais pastebimais vaizdo nuostoliais. JPEG standartas buvo sukurtas 1992 metais ir 1994 patvirtintas ISO standartų. JPEG standartas aprašo kodeką, kuris apibūdina kaip vaizdas suglaudinimas ir kaip grąžinamas į pradinę būseną. Exif ir JFIF standartai apibūdina dažniausiai naudojamus JPEG vaizdų keitimosi formatus. Iso standartas ISO/IEC 10918 aprašantis JPEG susideda iš tokių dalių (1 lentelė) [3]:

Standarto dalis	Patvirtinimo metai	Apibūdinimas
ISO/IEC 10918-1:1994	1992	Reikalavimai ir patarimai
ISO/IEC 10918-2:1995	1994	Atitikimo verifikavimas
ISO/IEC 10918-3:1997	1996	Plėtiniai
ISO/IEC 10918-4:1999	1998	Įvairių JPEG profilių registracijos, glaudinimo tipai
ISO/IEC FCD 10918-5	Kūrimo stadijoje	JPEG dokumentų mainų formatas (angl. File Interchange Format, JFIF)

1 lentelė. JPEG ISO standartas.

2.2.2. JPEG kodavimo pavyzdys

JPEG dokumentas gali būti užkoduotas įvairiais būdais, tačiau dažniausiai naudojamasi JFIF kodavimo formatu. Užkodavimo procesą galime suskirstyti į keletą žingsnių (1 pav.) [4][5]:



1 pav. JFIF algoritmo seka.

1. Spalvų paletės keitimas

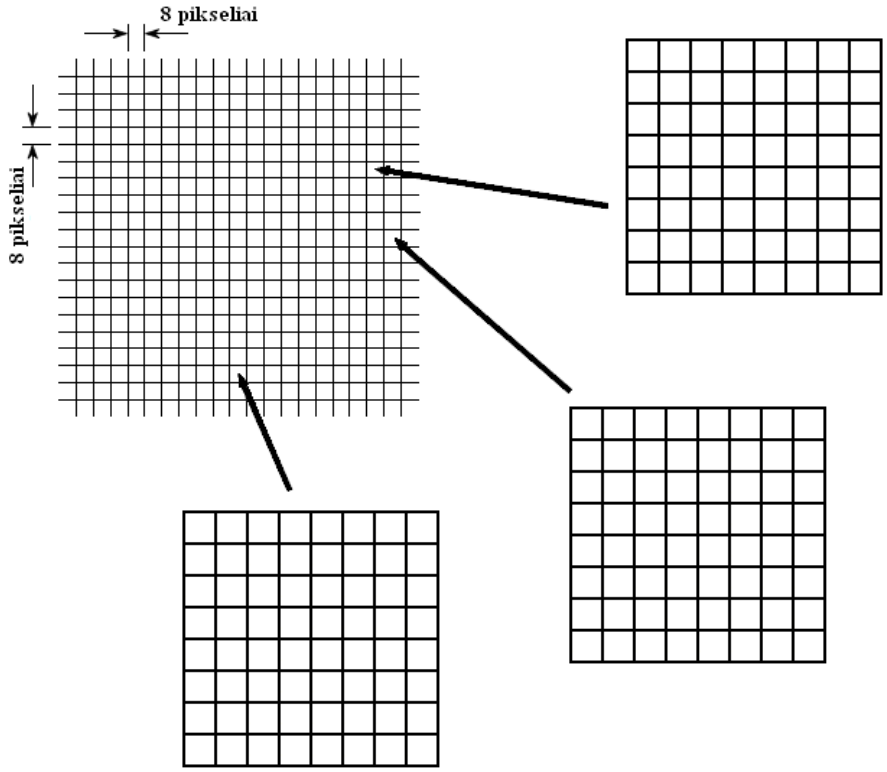
Vaizdo paletės keitimas iš RGB į YCbCr. Šis žingsnis kartais praleidžiamas. Priklausomai nuo spalvų gylio (*color depth*), t.y. naudojamų bitų kiekio užkoduoti atspalviams, naudojamos atitinkamos formulės. Paprastai paletės keitimas vyksta 24 bitų (*true color*) spalvų gylio vaizduose, pasinaudojus šiomis formulėmis [4][6]:

- Y = 0.299*R + 0.587*G + 0.114*B
- Cb = -0.1687*R – 0.3313*G + 0.5*B +128

- Cr = 0.5*R – 0.4187*G – 0.0813*B +128

2. Vaizdo skaldymas į blokus

Vaizdas yra suskaldomas į blokus, kurių dydis 8x8 pikselių. Kiekvieno pikselio duomenys užkoduoti Y′CbCr paletės komponentėmis (2 pav.) [4].



2 pav. Vaizdo skaldymas į blokus.

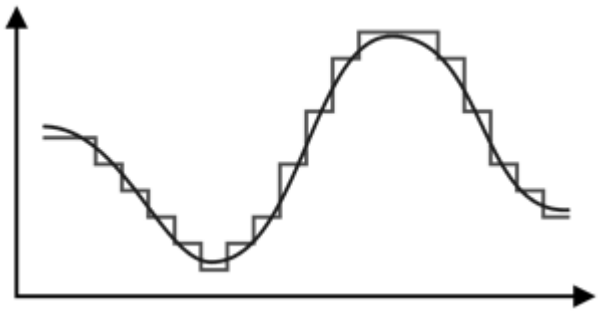
3. Diskrečioji kosinusų transformacija

Kiekvienas blokas su YCbCr komponentėmis transformuojamas į tam tikrų dažnių sudedamąsias, naudojant normalizuotą dviejų dimensijų antrojo tipo diskrečiąją kosinusų transformaciją (angl. Discrete cosine transform II, arba DCT-II). Diskrečioji antrojo tipo kosinusų transformacijos formulė:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1.$$

4. Kvantavimas

Kvantuojamos dažnių komponenčių amplitudės, kadangi žmogaus rega nėra labai jautri nedideliems ryškumo pokyčiams. Todėl kvantavimo (3 pav.) etapas leidžia stipriai sumažinti informacijos kiekį naudojamą kodavimui. Ši operacija yra vienintelė JPEG vaizdų kodavime, kurios metu prarandami duomenys, jeigu diskrečioji kosinusų transformacija buvo atlikta dideliu tikslumu.



3 pav. Kvantavimo pavyzdys.

Kvantavimui paprastai naudojama kvantavimo matrica (4 pav.), kuri aprašyta JPEG standarte [3][6]:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

4 pav. Kvantavimo matrica.

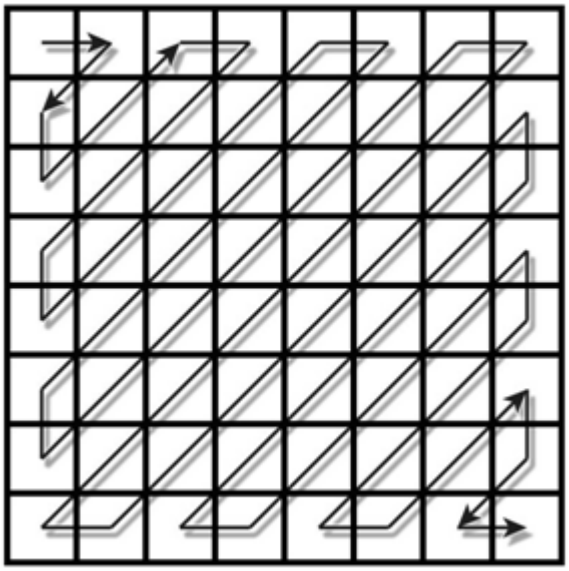
Kvantavimo procesas atliekamas labai paprastai. Turime vieną vaizdo bloką G[8x8] ir kvantavimo matricą Q[8x8], taigi naują kvantuotų reikšmių matricą B[8x8] gauname naudodamiesi šia formule:

$$B[i,j] = \left(\frac{G[i,j]}{Q[i,j]}\right), \text{ čia i} = 0, 1, 2...7; \text{j} = 0, 1, 2...7.$$

Gauti rezultatai visada apvalinami iki sveikosios dalies.

5. Entropinis kodavimas

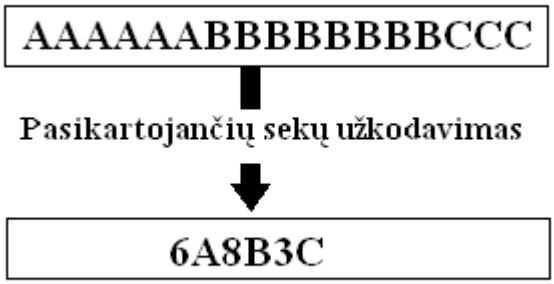
Tai informacijos kodavimo būdas be nuostolių. Entropinis kodavimas susideda iš vaizdo skenavimo zizagu (5 pav.) ir naudoja pasikartojančių sekų kodavimą (angl. run-length encoding, RLE), kuris sugrupuoja vienodas dažnių sekas. Galiausiai naudojamas Hafmano kodavimas.



5 pav. Skenavimas zigzagu.

5.1. Pasikartojančių sekų kodavimas (angl. run-length encoding)

Vienas paprasčiausių kodavimo būdų, kai pasikartojančios sekos pakeičiamos atitinkama trumpesne seka (6 pav.).



6 pav. Pasikartojančių sekų kodavimo pavyzdys.

5.2. Hafmano kodavimas (angl. Huffman coding)

Hafmano kodavimas, tai tokia entropinio kodavimo rūšis kuomet pasikartojantys simboliai užkoduojamos trumpesnėmis sekomis ar simboliais, kurie gaunami naudojantis tikimybių teorija. Koduojant Hafmano būdu sudaroma speciali kintamų sekų ilgių arba simbolių lentelė. Ši lentelė gaunama analizuojant koduojamus duomenis – skaičiuojant įvairių simbolių pasirodymo tikimybes koduojamuose duomenyse. Taigi trumpiausia seka ar simbolis koduoja dažniausiai duomenyse pasirodantį simbolį, o ilgiausia – rečiausiai.

2.3. Aparatūrinės įrangos aprašymo kalbos

Aparatūrinės įrangos aprašymo kalbos (angl. **H**ardware **D**escription **L**anguage) suteikia galimybę projektuoti ir verifikuoti lustus neturint realios įrangos. Dabar labiausiai paplitusios ir išsamiausiai dokumentuotos HDL kalbos (2 lentelė):

- VHDL
- Verilog
- SystemC

HDL Kalba	Sukūrimo metai	Įtakota	Kūrėjas	Verifikavimo galimybės
VHDL	1987	Ada, Pascal	US Department of Defense	Plačios
Verilog	1983/1984	C	Phil Moorby, Prabhu Goel (Automated Integrated Design Systems)	Plačios
SystemC	2000	C++	ARM Ltd., CoWare, Synopsys	Labai plačios

2 lentelė. HDL kalbos.

2.3.1. VHDL

VHDL arba **V**HSIC **HDL** (VHSIC - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit). Kalba sukurta Jungtinių Valstijų gynybos departamento (**D**epartment **o**f **D**efense, **DOD**) iniciatyva, siekiant užtikrinti kuriamų skaitmeninių sistemų aukštą efektyvumą. Kalba yra patvirtinta IEEE standartų asociacijos ir yra labai plačiai paplitusi pasaulyje. Jos sukūrimas palengvino skaitmeninių sistemų projektavimą ir kūrimą, kadangi atsirado galimybė kompiuterio pagalba per gana trumpa laiką suprojektuoti lustus susidedančius iš daugybės loginių elementų. Taip pat atsirado galimybė kur kas paprasčiau testuoti sistemą [7]. VHDL kalba buvo sukurta projektuoti aukštesniame abstrakcijos lygyje, tačiau su laiku, nusileido iki žemesnio abstrakcijos lygio. Kalba palaiko paralelias struktūras naudojant *process* konstrukcijas. Kadangi VHDL yra naudojama jau 15 metų kaip standartas Elektronikos ir Elektronikos Inžinerijos Institute (IEEE) daugelis specialistų yra įsitikinę, kad VHDL kalba lyginant su kitomis aparatūrinės įrangos aprašymo kalbomis yra patikimesnė.

2.3.2. Verilog

Verilog yra pirmoji moderni aparatūrinės įrangos aprašymo kalba. Ši kalba iš pradžių atsirado kaip labai žemo abstrakcijos lygio aparatūrinės įrangos aprašymo kalba, tačiau laiku bėgant pakilo į aukštesnį abstrakcijos lygį. Verilog yra patvirtinta IEEE standartais. Paprastai Verilog naudojama skaitmeninių mikroschemų projektavime, testavime ir kūrime RT (register transfer) abstrakcijos lygyje. Kalba savo sintaksę perėme iš C, kadangi C kalba plačiai paplitusi inžinerinės programinės įrangos kūrime. Kalbai paplitus atsirado jos patobulinimų bei atmainų, kaip: Verilog-2001, Verilog 2005, SystemVerilog, Verilog-AMS [8].

2.3.3. SystemC

SystemC yra naujos kartos sistemų aprašymo kalba (angl. System Description Language). Ši kalba yra patvirtinta IEEE standartų asociacijos. Kalba turi semantinių panašumų į VHDL ir Verilog, tačiau turi ir objektinio programavimo savybių, kurių nėra šiose kalbose. SystemC kalba padaryta standartinės C++ pagrindu ir išplečia C++ naujomis klasėmis naudojant šablonus ir makro komandas. Procesų veikimas kalboje paremtas įvykiais, kurie gali būti aprašyti naudojant paprastą C++ kalbą. SystemC labai naudinga sistemų projektavime ir tikrinime, kadangi ji apjungia aparatinę įrangą su programine įranga. Kalba pritaikyta modeliuoti sistemą blokais. Kadangi SystemC yra dar ganėtinai jauna palyginus su VHDL ar Verilog, jos paplitimas nėra labai didelis [9].

2.4. Pakartotinio naudojimo metodologija

Silicio technologijos dabar leidžia kurti lustus susidedančius iš šimtų milijonų tranzistorių. Keičiantis ir tobulėjant lustams keičiasi ir jų projektavimo metodologijos. Lustų projektavimas rašant RTL kodą iš pradžių naujai (*from scratch*), RTL blokų integravimas į aukščiausio lygio projektą ir sintezė kartu su elementų išdėstymu yra nebetinkamos technologijos sudėtingiems lustams. Pakartotinio naudojimo (*reuse*) metodologija – naudojimas sukurtų ir verifikuotų komponentų – dabar yra kertinis lustų bei SOC (**S**ystem **O**n **C**hip) kūrimo

akmuo, kadangi tai vienintelė metodologija, kuri leidžia sudėtingus lustus projektuoti prieinama kaina, geresnės kokybės taip pat taupo žmonių išteklius ir laiką. Projektuotojų iššūkis ne tame ar taikyti pakartotinio naudojimo metodologiją ar ne, o tame kaip ją pritaikyti efektyviai [10].

Projektavimo metodologija nebūtinai sutampa tarp visų sistemų projektuotojų, tačiau egzistuoja bendros problemos, su kuriomis susiduria visi projektuotojai:

- Produkto paleidimo į rinką terminai;
- Našumo, ploto, galios parametrai;
- Didelio sudėtingumo lustų sudėtingas verifikavimas;
- Reikalingi skirtingų sričių specialistai;
- Yra panašių projektų, tačiau neįmanomas jų pritaikymas dėl skirtingų įrankių, gairių.

Remiantis šiomis problemomis projektuotojai pritaikė blokinį projektavimo metodą, kuris remiasi pakartotiniu naudojimu. Naudojant komponentus, kurie jau buvo prieš tai naudoti ir verifikuoti padengiama dalį aukščiau išvardintų problemų. Tačiau susiduriama su problema, kad komponentai, kurie nebuvo kuriami remiantis pakartotinio naudojimo metodologija turi mažai arba išvis neturi naudos projektuotojams [10]. Norint kad komponentas būtų pakartotinai naudojamas, pirma jis turi būti korektiškai suprojektuotas ir įgyvendintas. Keletas gero komponento savybių:

- Suprojektuotas išspręsti bendrinę problemą – lengvai konfigūruojamas įvairiems problemos sprendimams;
- Suprojektuotas palaikyti keletą skirtingų technologijų;
- Suprojektuotas testavimui įvairiais įrankiais, o ne tik konkrečiu vienu įrankiu;
- Turėtų naudoti standartines bendravimo sąsajas;
- Ištestuotas nepriklausomai nuo lusto, kuriame jis bus naudojamas;

- Turėtų dokumentaciją, kurioje aprašytas tinkamas taikymas ir įvairūs parametrų apribojimai.

Atkartojimas yra labai efektyvi metodika projektuojant įvairias sistemas. Atkartojimas pasiekiamas per abstrakcijos lygmens kėlimą (aukšto lygmens abstrakcijos leidžia kurti lengvai atkartojamus komponentus ir jų bibliotekas), apibendrinimą (leidžia aprašyti atkartojamus (bendrinius) komponentus naudojant bendrines kalbos abstrakcijas) ir automatizavimą (apima programų generatorių kūrimą ir leidžia sumažinti pastangas reikalingas atkartojamumui realizuoti). Atkartojimo metodai [14]:

1. Komponentinis atkartojimas pagrįstas sistemos dalių atkartojimu;
    - Baltos dėžės atkartojimas (*white-box reuse*) – vidinė komponento struktūra matoma programuotojui, atkartojimo metu ją galima modifikuoti pritaikant prie sistemos konteksto.
    - Juodos dėžės atkartojimas (*black-box reuse*) – vidinė komponento struktūra nematoma programuotojui, ir atkartojimo metu jos modifikuoti negalima.
    - Pilkos dėžės atkartojimas (*gray-box reuse*) – vidinė komponento struktūra matoma programuotojui, tačiau atkartojimo metu jos modifikuoti negalima.
2. Generatyvinis atkartojimas pagrįstas komponentų kūrimo ir modifikavimo procesų atkartojimu naudojant specialius įrankius.

2.4.1. Metakalbos

Kompiuterio programa yra užkoduotas algoritmas, turintis sintaksę (struktūrą) ir semantiką. Struktūra įgalina aprašyti transformacijas, kad jas būtų galima atlikti, o semantika pateikia būdą lyginti programas ir spręsti apie transformacijų pagrįstumą. Programa užrašoma kokia nors kalba. Todėl kalbos klasė, tipas, struktūra ir galimybės veikia transformacijas tiesiogiai. Kalba, iš kurios transformuojama vadinama pirmine (*source*), o kalba į kurią transformuojama vadinama paskirties (*target*) kalba. Atitinkamai vadinamos ir programos – pirmine ir paskirties. Pirminės programos vertimą į paskirties programą atlieka transformavimo

sistemos. Apskirtai, transformavimo sistema gauna pirminę programą kaip įvesties duomenis ir išduoda transformuotą programą, vadinamą paskirties programa, kaip išvesties duomenis. Galima sakyti, kad transformavimo sistema traktuoja programas lygiai taip, kaip tradicinė programa elgiasi su paprastais duomenimis. Sistemos, kurios traktuojamos kaip programos, apdorojančios kitas programas kaip duomenis, vadinamos metaprogramavimo sistemomis [11].

Metaprogramavimas tai manipuliavimas programomis kaip duomenimis (sąsajos su automatiniu programų kūrimu). Atkartojimas gali būti pasiektas naudojant metaprogramavimą. Metaprogramavimas su atkartojamu siejasi taikymo prasme, metaprogramavimas leidžia specifikuoti (t.y., tiksliai aprašyti) iš anksto numatomus keitimus [14]. Homogeninis metaprogramavimas yra atskiras metaprogramavimo atvejis, kai bendriniai komponentai kuriami vienos kalbos aplinkoje, naudojant vien tik tos kalbos abstrakcijas, pvz.:

- VHDL generic;
- SystemC šablonai.

Bendrinių komponentų kūrimas vienoje aplinkoje yra paprastesnis, tačiau sukurti komponentai ne visada yra sintezuojami, taip pat ne visas struktūras įmanoma įgyvendinti. Pavyzdžiui VHDL generic neturi galimybės nurodyti įeinančių (*input*) ar išeinančių (*output*) jungčių kiekio, tačiau gali nurodyti jau egzistuojančių jungčių pločius. Todėl kuriant sudėtingesnius bendrinius komponentus patogiau naudoti heterogeninį metaprogramavimą.

Bazinis metaprogramavimo mechanizmas yra parametrizavimas, kuris palaikomas visų metakalbų. Bendriniai parametrai turi reikšmes, kurios išreiškia tam tikrus taikomosios srities aspektus. Bendrinių komponentų kūrimą sudaro trys stadijos [14]:

- Koncepcijų atskyrimas - atskiriamas pastovus srities funkcionalumas;
- Koncepcijų realizavimas - pastovus srities funkcionalumas yra aprašomas tikslo kalba be papildomų programos kodo modifikacijų, o kintamas srities funkcionalumas modifikuojamas priklausomai nuo bendrinių parametrų reikšmių naudojant metakalbą;
- Koncepcijų integravimas - metakalbos ir tikslo kalbos abstrakcijos yra apjungiamos į bendrinį komponentą.

2.4.2. Open PROMOL

Open PROMOL (**Pr**ogram **Mo**dification **L**anguage) metakalba buvo sukurta bendrinių pakartotinai naudojamų komponentų kūrimui, taip pat kaip jau sukurtų komponentų apjungimui palengvinti. Open PROMOL turi nesunkią ir lengvai išmokstamą sintaksę. Pradinė Open PROMOL versija buvo sukurta 1997 metais Kauno technologijos universitete kaip bandomasis išorinis VHDL kodo modifikavimo įrankis. Nuo to laiko kalba sulaukė daug patobulinimų. Open PROMOL yra nepriklausoma nuo tikslo kalbos, todėl galimi įvairūs jos pritaikymai. Open PROMOL metakalbos kodo procesorius EREBUS sukurtas naudojant Lex & Yacc technologiją [12]. Open PROMOL metakalba yra patogi, nes:

- Metakalba sukurta atsižvelgiant į pakartotinai naudojamų komponentų kūrimo aspektus.
- Nesunku perprasti;
- Įmanoma aprašyti kur kas sudėtingesnes struktūras nei naudojant homogenines metakalbas.

2.5. Analizės išvados

1. JPEG kodavimas yra puikus metodas vaizdų saugojimui, kadangi pasižymi geromis glaudinimo savybėmis, taip pat išlaikoma vaizdų kokybe.
2. Patogiausias ir dažniausiai naudojamas JPEG kodavimo algoritmas – JFIF.
3. Kodavimo metu duomenys prarandami tik atliekant kvantavimą.
4. Kodavimo metu įmanoma vietoje diskrečiosios kosinusų transformacijos naudoti diskrečiąją furjė transformaciją.
5. JPEG realizacijas galima aprašyti visomis aptartomis aparatūrinės įrangos aprašymo kalbomis, tačiau daugiausiai realizacijų atlikta VHDL kalba.
6. Pakartotinio naudojimo (*reuse*) metodologija taps vis populiaresnė ateityje dėl didėjančių ir sudėtingėjančių sistemų.
7. Open PROMOL metakalba patogi pakartotinai naudojamų bendrinių komponentų kūrimui.

3. Atvirojo kodo JPEG realizacijų apžvalga

Šiame skyriuje bus apžvelgiami įvairūs prieinamų atvirojo kodo JPEG realizacijų aspektai.

3.1. EV_JPEG_ENC realizacija

EV_JPEG_ENC realizacija skirta užkoduoti bitmap tipo vaizdus į JPEG vaizdo formato bitų masyvą. Šioje realizacijoje naudojamas *JPEG baseline* kodavimo metodas.

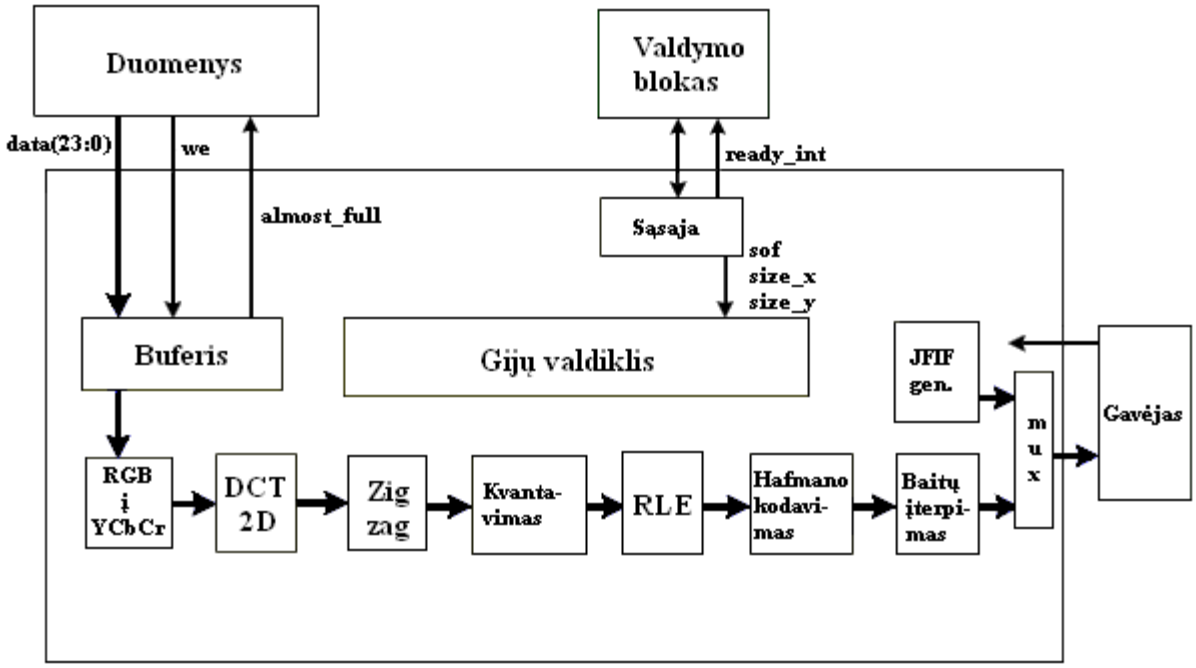
3.1.1. EV_JPEG_ENC realizacijos ypatybės

Ši atvirojo kodo JPEG realizacija pasižymi tokiomis savybėmis:

- JPEG kodavimas paremtas JPEG ITU-T T.81 | ISO/IEC 10918-1 standartais;
- Standartinės JFIF v 1.01 antraštės generavimas;
- Tik spalvotiems vaizdams (RGB įvestis);
- Dvi programuojamos kvantavimo lentelės;
- Neprogramuojamos Hafmano lentelės;
- Palaiko 24 bitų RGB ir 16 bitų RGB565 tipų duomenis;

3.1.2. EV_JPEG_ENC realizacijos architektūra

Bendra sistemos architektūra (7 pav.) susideda iš kodavimo grandinės, kuri paleidžiama naudojantis valdymo bloko sąsaja.



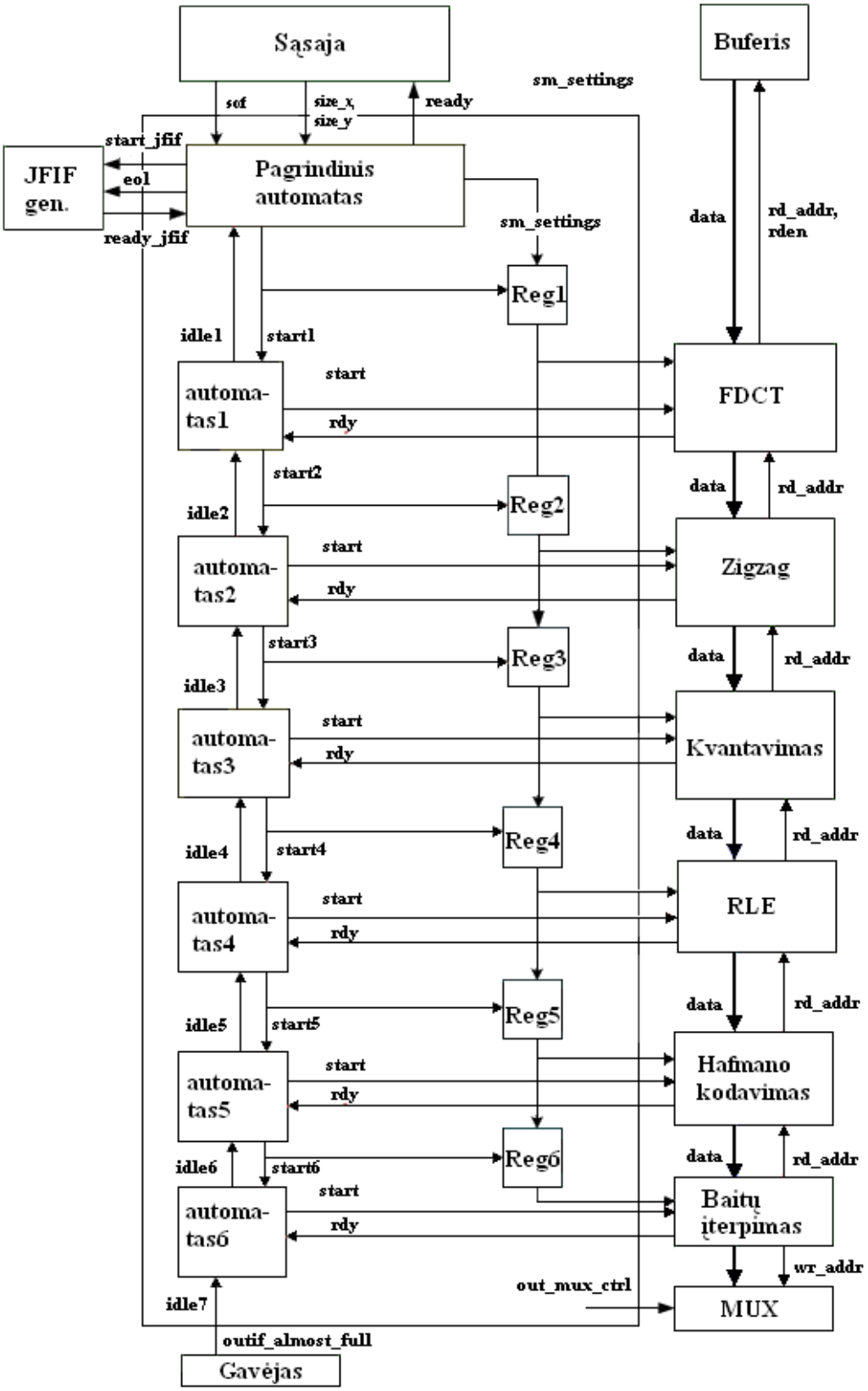
7 pav. EV_JPEG_ENC realizacijos architektūros schema.

Gaunami duomenys rašomi į buferį, kol buferis užpildomas beveik pilnas. Kodavimas yra valdomas valdiklio, kuris taip pat kontroliuoja gijas. Kiekviena gija apdoroja 8x8 dydžio blokus arba kitaip vadinamus duomenų vienetus. Galiausiai užkoduotas paveikslėlis įterpiamas ir saugomas išėjimo atmintyje. Kodavimo eigoje atliekami šie žingsniai:

- Vaizdo paletės keitimas iš RGB į YCbCr;
- Dviejų dimensijų antrojo tipo diskrečioji kosinusų transformacija;
- Skenavimas zizagu;
- Kvantavimas;
- Pasikartojančių sekų kodavimas;
- Hafmano kodavimas;
- JFIF antraštės įterpimas;

3.1.3. Valdantysis automatas

Pagrindinis valdantysis automatas (8 pav.) yra atsakingas už visą kodavimo proceso kontrolę. Gavęs neužkoduotą vaizdą jis paverčia vaizdą į užkoduotą JPEG bitų seką. Pirma patvirtinamas *sof* (start of frame) signalas taip pat kartu siunčiami *size_x* (vaizdo plotis) ir *size_y* (vaizdo aukštis) signalai. Atlikus šiuos veiksmus pradedamas vaizdo įkrovimas. Vaizdai apdorojami 8x8 pikselių blokeliais, einant horizontaliai iš dešinės į kairę ir vertikaliai iš viršaus į apačią. Galiausiai apdorotas bitų masyvas saugomas išvesties RAM atmintyje.



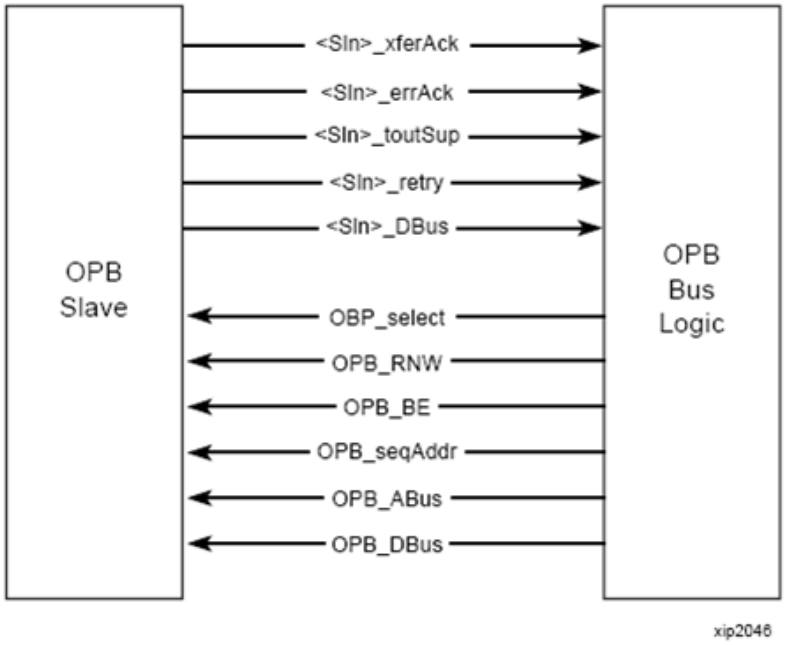
8 pav. Valdančiojo automato schema.

3.1.4. Buferis

Prieš pradedant duomenų apdorojimą vaizdas eilutė po eilutės rašomas į buferį. Buferis reikalingas, kadangi norima sumažinti gaišaties laiką tarp vaizdo įkėlimo ir vaizdo kodavimo pradžios. Šis buferis iš tikro atitinka RAM atmintį. Maksimalus palaikomas vaizdo plotis turi būti sukonfigūruotas naudojant C_EXTRA_LINES konstantą esančią JPEG_PKG.VHD faile, norint nurodyti RAM atminties dydį prieš sintezę. Buferis privalo saugoti bent aštuonias vaizdo eilutes, kad JPEG kodavimas galėtų prasidėti. Maksimaliai leidžiamas šešiolikos eilučių buferis.

3.1.5. Sąsaja

Sąsaja yra tarpininkas tarp CPU ir kodavimo įrenginio branduolio. Šiame darbe parinktas Xilinx OPB (On-Chip Peripheral Bus) v2.1 protokolas, kuris naudojamas tarpininkavime (9 pav.). JPEG kodavimo įrenginys čia veikia kaip OPB slave įrenginys.



9 pav. OPB v2.1 sąsajos schema.

Daugiau informacijos apie šią sąsają galima rasti OPB v2.1 specifikacijoje.

3.1.6. FDCT komponentas

FDCT komponentas apima keletą funkcijų: RGB paletės keitimą į YCbCr paletę, spalvų diskretizaciją, diskrečiąją kosinusų transformaciją. RGB paletės keitimas į YCbCr atliekamas naudojant šiuos metodus:

Y = (0.299*R) + (0.587*G) + (0.114*B)

Cb = (-0.1687*R) - (0.3313*G) + (0.5*B) + 128

Cr = (0.5*R) - (0.4187*G) - (0.0813*B) + 128

2D diskrečioji kosinusų transformacijos branduolys dirba su duomenų blokais, kurių dydis 64 kintamieji. Realizacijoje naudojamai 2D diskrečiajai kosinusų transformacijai atlikti naudojama ši formulė:

$$X(u,\ v) = \frac{2}{N} \cdot C(u) \cdot C(v) \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i,\ j) \cdot \cos \frac{(2i\ +1)u\ \pi}{2N} \cdot \cos \frac{(2j\ +1)v\ \pi}{2N}$$

Čia formulėje:

- C(u) , C(v) = $2^{-(1/2)}$, kai u = 0 ir v = 0, kitais atvejais C(u) , C(v) = 1;
- x(i,j) - įvesties kintamieji ties atitinkama pozicija (i,j);
- X(u,v) - išvesties kintamieji ties atitinkama pozicija (u,v);
- N = 64, nes naudojami blokai 8x8;

Kadangi atlikus transformaciją duomenys išvedami stulpeliais, nors jie gaunami eilutėmis, tai po transformacijos reikalinga gautos matricos transponacija.

3.1.7. Zigzag komponentas

Zigzago komponentas yra atsakingas už skanavimo zigzagu atlikimą. Komponentas tiesiog pakeičia kintamųjų seką 8x8 duomenų blokuose. Kintamieji pakeičiami pagal šias lenteles (3, 4 lentelės):

Įvestis:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

3 lentelė. Pradinė duomenų seka.

Išvestis:

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

4 lentelė. Duomenų seka po sukeitimo.

3.1.8. Kvantavimo komponentas

Kvantavimo komponentas atlieka įvestų duomenų dalybą pagal nustatytas kvantavimo reikšmes. Kvantavimo koeficientus galima nurodyti naudojant vidinę RAM atmintį (64x8 bitų). Kvantavimui naudojama tokia formulė:

$$FOUT(u, v) = \text{round}(FINPUT(u,v) / Q(u,v))$$

Čia formulėje:

- u, v – stačiakampės koordinatės;
- FINPUT – įvesties duomenys;
- FOUT – išvesties duomenys;
- round – apvalinimas iki sveikojo skaičiaus;

Standartinė kvantavimo lentelė yra tokia, kurioje visos reikšmės yra vienetai. Tokia lentelė reiškia, kad kvantavimas neatliekamas visiškai.

3.1.9. RLE komponentas

RLE branduolys atlieka pasikartojančių sekų kodavimą. Kiekvienam įvesties blokui (8x8) yra sukuriami kintamieji: RUNLENGHT, SIZE, AMPLITUDE.

- RUNLENGHT – sekoje iš eilės einančiu nulių skaičius;
- SIZE – bitų kiekis naudojamas užkoduoti AMPLITUDE kintamajam;
- AMPLITUDE – sveikasis skaičius su ženklu;

Taip pat šiame branduolyje atliekamas entropinis kodavimas, kuris užkoduoja duomenis į duomenų porą: AMPLITUDE ir SIZE remiantis šia lentele (5 lentelė):

SIZE	AMPLITUDE
1	-1,1
2	-3,-2,2,3
3	-7..-4,4..7
4	-15..-8,8..15
5	-31..-16,16..31
6	-63..-32,32..63
7	-127..-64,64..127
8	-255..-128,128..255
9	-511..-256,256..511
10	-1023..-512,512..1023
11	–2 047..–1 024,1 024..2 047

5 lentelė. Kintamųjų SIZE ir AMPLITUDE reikšmių kitimas.

3.1.10. Hafmano kodavimas

Hafmano kodavimo komponentas atlieka Hafmano kodavimo operaciją. Šis komponentas konvertuoja paralelius duomenis į nuoseklią bitų seką. Nuosekli bitų seka įrašoma į išvesties FIFO tipo atmintį. Naudojami du atminties blokai kurių dydis 2x64x8. Dviejų atminties blokų naudojimas paspartina tolesnių operacijų seką, kadangi gali būti vykdomas baitų įterpimas skaitant duomenis iš vieno atminties bloko, kol Hafmano kodavimo komponentas rašo užkoduotus duomenis į kitą atmintį. Operacijos šiame komponente atliekamos su 8x8 duomenų blokais.

3.1.11. Baitų įterpimo komponentas

Šis komponentas reikalingas norint apsaugoti duomenis nuo netikrų žymeklių atsiradimo. Žymeklio (*marker*) atsiradimą sukelia 0xFF reikšmės atsiradimas sekoje. Taigi norint išvengti tokios problemos po kiekvienos 0xFF reikšmės įterpiama nulinė reikšmė. Pavyzdys: po Hafmano kodavimo iš atminties nuskaitoma AA BB CC FF 11 22 seka, po baitų įterpimo seka atrodys taip - AA BB CC FF 00 11 22. Šiame komponente 0xFF sekos aptikime naudojamas paprasčiausias komparatorius kuris tikrina ar įvesties baitai lygūs 0xFF.

3.1.12. JFIF antraštės generatorius

Atliekamas standartinės JFIF antraštės generavimas. Generavimas atliekamas iš komponente esančios RAM atminties. Dauguma JFIF antraštės laukų yra nekonfigūruojami, tačiau keletas iš jų gali būti sukonfigūruoti per sąsają. Konfigūruojami laukai:

- SIZE_Y_H, SIZE_Y_L – viršutinis ir apatinis paveikslėlio aukščio baitas;
- SIZE_X_H, SIZE_X_L – viršutinis ir apatinis paveikslėlio pločio baitas;
- QL0....QL63 – apšviestumo kvantavimo lentelė zigzago seka;
- QC0....QC63 – spalvingumo kvantimo lentelė zigzago seka;
- Number of components – 0 = nespalvotas, 3 = spalvotas (RGB);

3.1.13. EV_JPEG_ENC realizacijos apibendrinimas

1. Realizacija turi sąlyginai plačias kofigūravimo galimybes.
2. Koduojamas vaizdas turi būti tekstinio (*.txt) formato *bitmap* tipo, taigi reikia naudoti išorinę programinę įrangą norint paruošti duomenis kodavimui.
3. Realizacija sukurta neatsižvelgiant į pakartotinės panaudos (reuse) metodologiją, todėl ją ganėtinai sunku pritaikyti savo reikmėmis.
4. Realizacija yra pritaikyta konkrečiai technologijai ir nepalaiko skirtingų technologijų.
5. Realizacija yra ganėtinai tvarkinga, turi dokumentaciją.

3.2. JPEG Hardware Compressor realizacija

JPEG Hardware Compressor realizacija skirta užkoduoti bitmap tipo vaizdus į JPEG vaizdo formato bitų masyvą. Šioje realizacijoje taip pat naudojamas standartinis *JPEG baseline* kodavimo metodas.

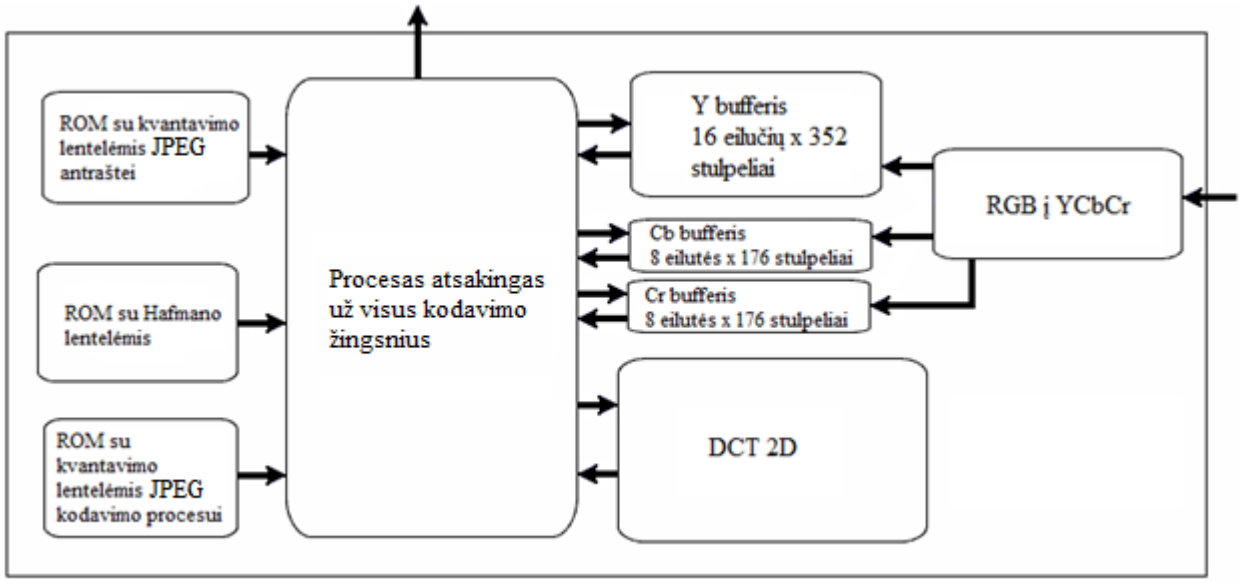
3.2.1. JPEG Hardware Compressor realizacijos ypatybės

Ši atvirojo kodo JPEG realizacija pasižymi tokiomis savybėmis:

- Standartinės JFIF antraštės generavimas;
- Tik spalvotiems vaizdams (RGB įvestis);
- Maksimalus koduojamo vaizdo dydis 352x288 pikselių;
- Programuojamos Hafmano lentelės;
- Programuojamos kvantavimo lentelės.

3.2.2. JPEG Hardware Compressor realizacijos architektūra

Sistemos architektūra (10 pav.) susideda iš kodavimo komponentų, kurių valdymas atliekamas iš pagrindinio komponento.



10 pav. JPEG Hardware Compressor architektūros schema.

Pagrindinis procesas valdantis visus kodavimo žingsnius valdo šiuos kodavimo etapus:

- Vaizdo paletės keitimas iš RGB į YCbCr;
- Dviejų dimensijų antrojo tipo diskrečioji kosinusų transformacija;
- JPEG antraštės įterpimas;
- Kvantavimas;
- Hafmano kodavimas;

3.2.3. Procesas atsakingas už visų kodavimo žingsnių valdymą

Procesas, kuris valdo visus kodavimo žingsnius ir yra atsakingas už korektišką realizacijos veikimą. Šiame procese duomenys siunčiami į DCT komponentą (komponentas atsakingas už diskrečiąją kosinusų transformaciją), gauti rezultatai kvantuojami ir įrašomi į tą

patį adresą iš kur buvo nuskaityti (buferiai) siunčiant į DCT. Taip pat šio proceso metu sukuriama JPEG užkoduoto vaizdo antraštė. Antraštės informacija saugoma pačiame procese šešioliktaine forma. Šiame procese taip pat atliekamas entropinis kodavimas bei kvantavimas.

3.2.4. Procesas atsakingas už spalvų paletės transformavimą

Šis procesas atlieka spalvų paletės transformavimą iš RGB į YCbCr. Transformacijos rezultatą perduoda JPEG procesui. Rezultate gauti duomenys išsaugomi buferiuose pritaikius JPEG lygio perstūmimą. Taigi po transformacijos [0;128;128] taps -[128;0;0]. Transformacijoje naudojama standartiniai RGB į YCbCr keitimo metodai:

$$Y = (0.299*R)+(0.587*G)+(0.114*B)$$

$$Cb = (-0.1687*R)-(0.3313*G)+(0.5*B)+128$$

$$Cr = (0.5*R)-(0.4187*G)-(0.0813*B)+128$$

3.2.5. ROM atmintys

Realizacijoje naudojami trys ROM atminties blokai, kuriuose saugoma informacija reikalinga kvantavimams ir Hafmano kodavimui:

- ROM su kvantavimo lentele, skirta JPEG antraštei sukurti;
- ROM su kvantavimo lentele, skirta JPEG kodavimo procesui atlikti;
- ROM su Hafmano lentele, skirta Hafmano kodavimo operacijai atlikti.

Duomenys šiuose atminties blokuose saugomi dvejetaine bei šešioliktaine forma. Informaciją šiuose atminties blokuose galime sukonfigūruoti iš anksto prieš sukuriant šiuos blokus. Konfigūravimas atliekamas redaguojant atitinkamus ROM failus, kurie pridėti prie realizacijos. Tai:

- *huff_rom.coe*;
- *q_rom.coe*;

- *tabla_q.coe*.

3.2.6. JPEG Hardware Compressor realizacijos apibendrinimas

1. Realizacijos konfigūravimo galimybės yra ganėtinai ribotos.
2. Realizacija neturi dokumentacijos, taip pat VHDL kodas sunkiai suprantamas, kadangi nesilaikoma daugumos gero kodavimo stiliaus taisyklių.
3. Realizacija yra pritaikyta konkrečiai technologijai ir nepalaiko skirtingų technologijų.
4. Realizacija sukurta neatsižvelgiant į pakartotinės panaudos (reuse) metodologiją, todėl ją ganėtinai sunku pritaikyti savo reikmėmis.
5. Koduojamas vaizdas turi būti *bitmap* (.bmp) formato, todėl nereikia išorinės programinės įrangos duomenų paruošimui kodavimui.

3.3. Realizacijų palyginimas

Ištirtos realizacijos yra ganėtinai panašios naudojamų JPEG kodavimo metodų atžvilgiu, tačiau skirtingos įgyvendinimais bei kodo kokybe.

	EV_JPEG_ENC	**JPEG Hardware Compressor**
Dokumentacija	Gera	Nėra
Geras kodavimo stilius	Taip	Ne
Įeinantys duomenys	*.txt* tipo, reikalinga išorinė programinė įranga paruošimui	*.bmp* tipo, nereikalinga išorinė programinė įranga
Koduojamo vaizdo rezoliucija	640x480	352x288
Konfigūravimo galimybės	Sąlyginai plačios	Siauros
Kodo suprantamumas	Geras	Prastas
Pritaikymas pakartotiniam naudojimui	Ne	Ne

	EV_JPEG_ENC	JPEG Hardware Compressor
Skirtingų technologijų palaikymas	Nėra	Nėra

6 lentelė. Realizacijų palyginimas įvairiais aspektais.

Kaip matome (6 lentelė) EV_JPEG_ENC realizacija laimi dauguma aspektų. Tačiau JPEG Hardware Compressor realizacija turi vieną stiprų privalumą, jog įeinantys duomenys gali būti *bitmap* tipo. Taip pat matome, jog abi realizacijos buvo sukurtos kažkokiems konkretiems uždaviniams ir neatsižvelgiant į pakartotinio naudojimo (*reuse*) metodologijos gaires. Dėl šių aspektų yra ganėtinai sudėtinga šias realizacijas pritaikyti savo reikmėmis.

4. Parametrizuotas spalvų paletės keitimo iš RGB į YCbCr komponentas

Spalvų modelis yra abstraktus matematinis modelis aprašantis spalvas kaip skaičių sekas. Tipiškai tai būna trys ar keturios reikšmės (pvz. RGB, CMYK spalvų paletėse). YCbCr (Y'CbCr, Y Pb/Cb Pr/Cr) spalvų modelių šeima dažniausiai naudojama vaizdinėse ir skaitmeninių fotografijų sistemose. YCbCr nėra absoliuti spalvų erdvė, o labiau būdas RGB informacijai užkoduoti. Daugiausiai naudojama video ir vaizdo medžiagos suspaudimo metoduose, tokiuose kaip MPEG ar JPEG. Pats spalvų modelio keitimo procesas yra konvertavimas iš vienos spalvų vaizdavimo formos į kitą. Konvertavimo tikslas yra išlaikyti kuo panašesnį vaizdą į originalų vaizdą (vaizdą prieš spalvų modelių pakeitimą) po spalvų modelio keitimo. Parametrizavimui pasirinktas spalvų paletės keitimo iš RGB į YCbCr komponentas nes:

- Yra vienas iš svarbesnių JPEG kodavimo realizacijose;
- Apimtis nėra labai didelė, kaip tarkim diskrečiosios antrojo tipo kosinusų transformacijos, kuriai įgyvendinti gali prireikti keleto žmonių. Taip pat apimtis nėra labai maža.
- Turi įvairių parametrizavimo galimybių;
- Komponentas turi plačią pritaikymo sritį, kadangi paletės keitimas naudojamas ne tik JPEG realizacijose, bet ir įvairiose su vaizdu susijusiose sistemose.

Bendrinimui pasirinkta Open PROMOL metakalba, kadangi:

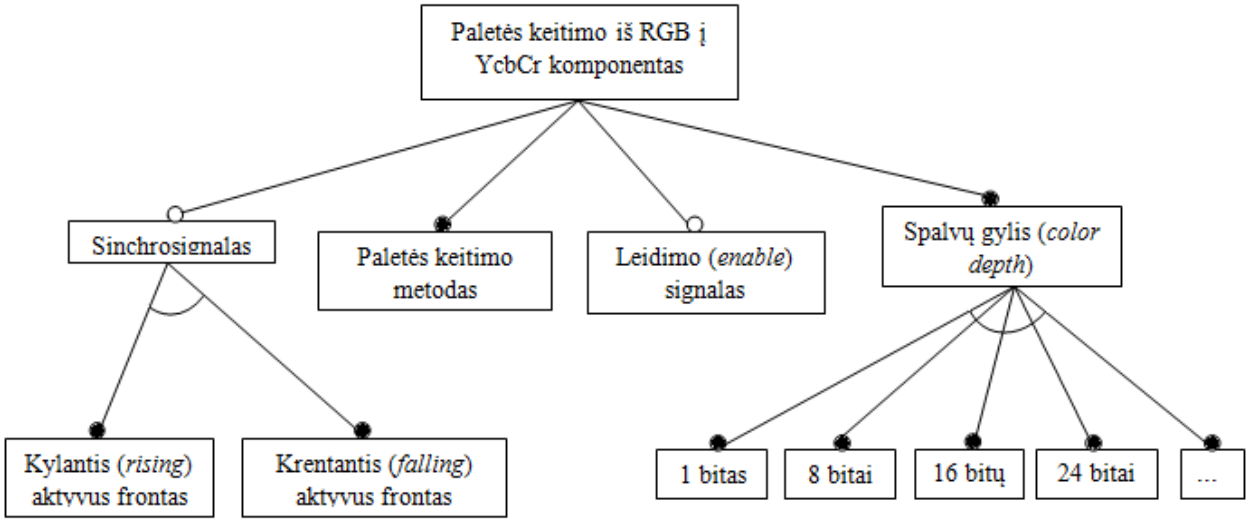
- Turima patirties naudojant open PROMOL metakalbą;
- Bendrinimo reikalavimai reikalauja keisti duomenų įėjimo/išėjimo signalų skaičių, ko negalima daryti VHDL generics metakalboje;
- Nesukelia problemų komponentų sintezėje, jeigu naudojamos leistinos VHDL struktūros.

4.1. Komponento atributų bei koncepcijų analizė

Spalvų paletės keitimo iš RGB į YCbCr komponento požymius patogu atvaizduoti naudojant požymių modelius. Požymių modeliai aprašo pastovius ir kintamus srities sąvokos požymius ir ryšius tarp požymių. Paprastai požymių modelis yra aprašomas naudojant požymių diagramas. Šių diagramų sudėtingumas gali kisti nuo labai nesudėtingų ir mažos apimties iki labai sudėtingų ir milžiniškos apimties, priklausomai nuo tiriamos srities, reikalavimų, poreikių. Požymių diagramą sudaro kryptinis beciklis grafas, sudarytas iš mazgų aibės, kraštinių aibės ir kraštinių žymų aibės. Grafo šakninis mazgas atvaizduoja aukščiausio lygmens požymį. Tarpiniai mazgai atvaizduoja sudėtinius požymius, o grafo lapai atvaizduoja atominius požymius, kuriuos neskaidomi į dar smulkesnius požymius [13]. Šie požymiai gali būti trijų tipų:

- Privalomi (AND) - grafas turės privalomąjį požymį, jeigu jis turi to požymio tėvinį požymį;
- Neprivalomieji (OR) - grafas gali turėti arba neturėti neprivalomą požymį, jeigu jis turi to požymio tėvinį požymį.
- Alternatyviniai (CASE/CASE OR) - grafas gali turėti tik vieną variantinį požymį, jeigu ji turi to požymio tėvinį požymį. Grafas turi mažiausiai vieną ARBA (OR) tipo alternatyvinį požymį, jeigu ji turi to požymio tėvinį požymį.

Spalvų paletės keitimo iš RGB į YCbCr komponento specifikavimas nesudėtinga požymių diagrama (11 pav.):



11 pav. Paletės keitimo iš RGB į YCbCr komponento požymių diagrama.

Iš požymių diagramos pasirinkti šie atributai parametrizavimui:

- Metodas skirtas apskaičiuoti šviesumo (skaisčio) komponentei Y;
- Metodas skirtas apskaičiuoti spalvinei komponentei Cb;
- Metodas skirtas apskaičiuoti spalvinei komponentei Cr;
- *Enable* signalas;
- Aktyvus sinchrosignalo frontas.

Komponentas pritaikytas 24 bitų spalvų (*True Color*) paletės keitimui. *True Color* paprastai apibrėžiama kaip 256 raudonos, žalios ir mėlynos atspalviai, kurie kartu sudaro 16,777,216 įmanomas spalvų variacijas.

Metodai skirti apskaičiuoti šviesumo (skaisčio) komponentei Y:

1. 16 + 66*R/256 + 129*G/256 + 25*B/256;
2. 77*R/256 + 150*G/256 + 29*B/256; *
3. ((16*256 + 119*G) + (66*R + 25*B))/256;

Metodai skirti apskaičiuoti spalvinei komponentei Cb:

1. 128 - 38*R/256 - 74*G/256 + 112*B/256;
2. -44*R/256 - 87*G/256 + 131*B/256 + 128; *

3. ((128*256 + 112*B) - (38*R + 74*G))/256;

Metodai skirti apskaičiuoti spalvinei komponentei Cr:

1. 128 + 112*R/256 - 94*G/256 - 18*B/256;
2. 131*R/256 - 110*G/256 - 21*B/256 + 128; *
3. ((128*256 + 112*R) - (94*G + 18*B))/256;

Čia formulėje:

- R - 8 bitų raudono atspalvio reikšmė;
- G - 8 bitų žalio atspalvio reikšmė;
- B - 8 bitų mėlyno atspalvio reikšmė;

PASTABA: metodai pažymėti * (antrųjų metodų grupė) netinkami R, G, B reikšmių skaičiavimui, kai šios reikšmės patenka į intervalą {250;255}, tačiau šie metodai teoriškai yra tiksliausi ir gauti rezultatai, naudojant šiuos metodus, turėtų nedaug skirtis nuo rezultatų gautų naudojant standartinį metodą.

Iš viso, atsižvelgiant į esamų parametrizuojamų atributų kiekį, įmanomi 3*3*3*2*2 = 108 skirtingų komponentų variantai.

Spalvų paletės keitimo iš RGB į YCbCr komponento įvesties (*input*) ir išvesties (*output*) signalai (7 lentelė):

Signalo pavadinimas	**Kryptis**	**Tipas**	**Trumpas apibūdinimas**
R	įvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų raudono atspalvio reikšmė
G	įvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų žalio atspalvio reikšmė
B	įvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų mėlyno atspalvio reikšmė

Signalo pavadinimas	**Kryptis**	**Tipas**	**Trumpas apibūdinimas**
CLK	įvestis	STD_LOGIC	Sinchrosignalas, priklausomai nuo parametrų gali būti kylantis (*rising*) arba krentantis (*falling*) aktyvus frontas
EN	įvestis	STD_LOGIC	Leidimo (*Enable*) signalas, kurio gali ir nebūti komponente, priklausomai nuo generavimo parametrų. Aktyvus aukštas signalo lygis. Kai signalas neaktyvus R, G, B įvestys tiesiogiai perduodamos į Y, Cb, Cr išvestis
Y	išvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų šviesumo (skaisčio) komponentė
Cb	išvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų mėlyna spalvinė komponentė
Cr	išvestis	STD_LOGIC_VECTOR (7 DOWNTO 0)	8 bitų raudona spalvinė komponentė

7 lentelė. Spalvų paletės keitimo iš RGB į YCbCr komponento įvesties (*input*) ir išvesties (*output*) signalai.

4.2. Komponentų modeliavimo rezultatai

Modeliavimui buvo atsitiktinai pasirinkti keturi sugeneruoti komponentai. Komponentus patogumo dėlei pavadinkime A, B, C ir D. Šių komponentų parametrai (8 lentelė):

Parametras	**Komponentas**			
	A	B	C	D
Metodas skirtas apskaičiuoti šviesumo (skaisčio) komponentei Y	1	1	1	2
Metodas skirtas apskaičiuoti spalvinei komponentei Cb	1	2	3	3
Metodas skirtas apskaičiuoti spalvinei komponentei Cr	1	3	1	1
Enable signalas	Yra	Nėra	Yra	Nėra
Aktyvus sinchrosignalo frontas	Kylantis	Kylantis	Krentantis	Krentantis

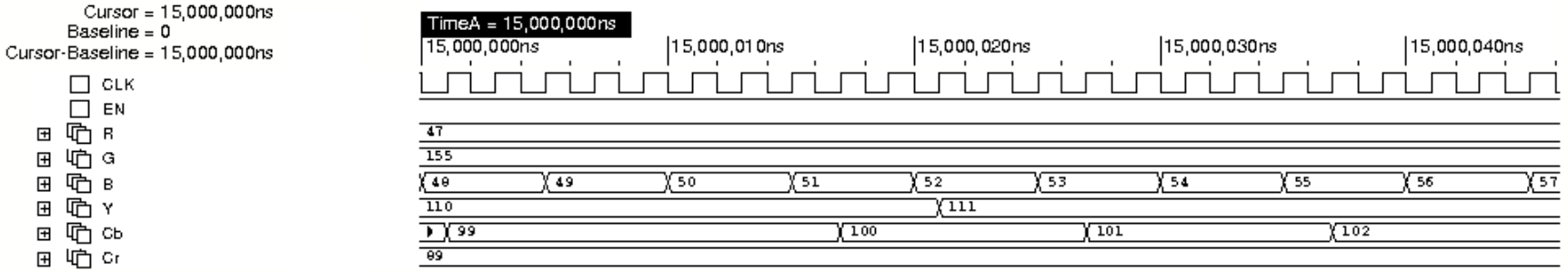
8 lentelė. Sugeneruotų komponentų parametrai.

Modeliavimo metu naudojamas testas kuris apima visus įmanomus leidžiamus variantus R, G ir B įvestyse. Ciklo generuojančio R, G, B įvestis kodo ištrauka:

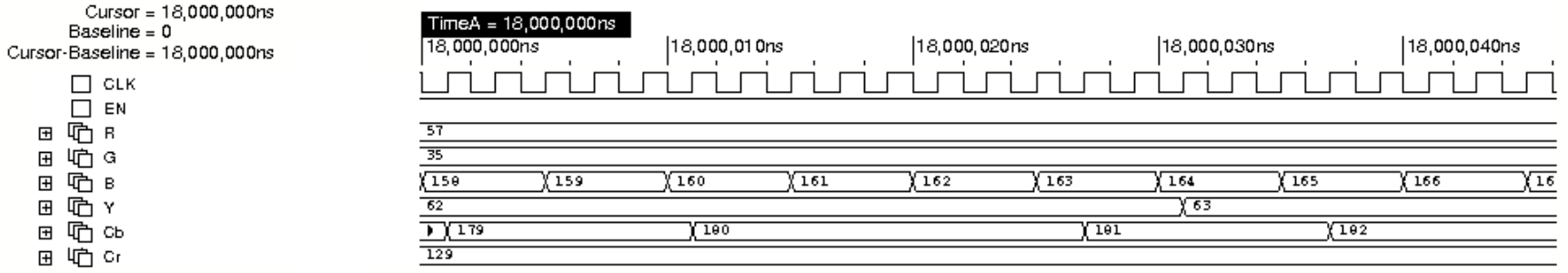
```
for i in 0 to 250 loop
 R <= std_logic_vector(to_unsigned(i,8));
 for j in 0 to 250 loop
  G <= std_logic_vector(to_unsigned(j,8));
  for k in 0 to 250 loop
   B<= std_logic_vector(to_unsigned(k,8));
   wait for 5 ns;
  end loop;
 end loop;
end loop;
```

Komponentas A

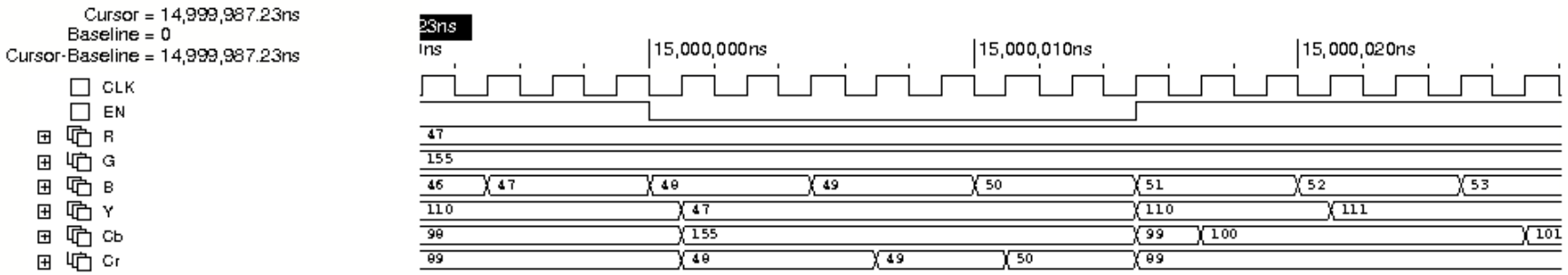
Laiko diagramos (12, 13, 14 pav.):



12 pav. Paletės keitimo iš RGB į YCbCr komponento A laiko diagrama.



13 pav. Paletės keitimo iš RGB į YCbCr komponento A laiko diagrama.



14 pav. Paletės keitimo iš RGB į YCbCr komponento A laiko diagrama.

Kaip matome iš laiko diagramų, rezultatai išvedami prie kylančio sinchrosignalo, o tai atitinka komponento parametrus, taigi sinchrosignalas veikia korektiškai. Taip pat iš 16 pav. matome, jog nustačius *En* = 0 komponentas nebeatlieka paletės keitimo, o tiesiog perduoda R, G, B įėjimo signalų reikšmes į Y, Cb ir Cr išėjimo signalus, taigi *Enable* signalas veikia korektiškai. Modeliavimo metu gautų rezultatų palyginimas (9 lentelė):

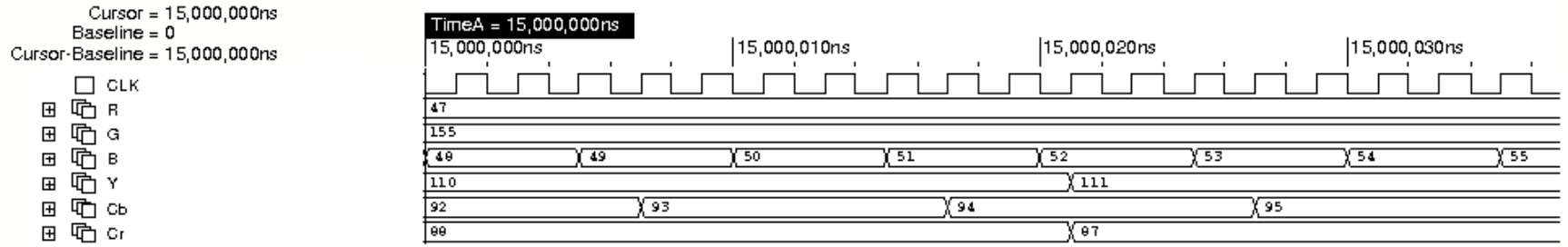
Įeinančios R, G, B reikšmės (pagal 14 pav. ir 15 pav. laiko diagramas)			**Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės**			**Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 14 pav. ir 15 pav. laiko diagramas)**		
R	G	B	Y	Cb	Cr	Y	Cb	Cr
47	155	48	110,51	92,72	82,70	110	99	89
47	155	49	110,62	93,22	82,62	110	99	89

Įeinančios R, G, B reikšmės (pagal 14 pav. ir 15 pav. laiko diagramas)			Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės			Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 14 pav. ir 15 pav. laiko diagramas)		
47	155	50	110,74	93,72	82,54	110	99	89
47	155	51	110,85	94,22	82,46	110	100	89
47	155	52	110,97	94,72	82,37	111	100	89
47	155	53	111,08	95,22	82,29	111	101	89
47	155	54	111,19	95,72	82,21	111	101	89
47	155	55	111,31	96,22	82,13	111	102	89
47	155	56	111,42	96,72	82,05	111	102	89
57	35	158	55,60	185,79	129,00	62	179	129
57	35	159	55,71	186,29	128,92	62	179	129
57	35	160	55,83	186,79	128,84	62	180	129
57	35	161	55,94	187,29	128,76	62	180	129
57	35	162	56,06	187,79	128,67	62	180	129
57	35	163	56,17	188,29	128,59	62	181	129
57	35	164	56,28	188,79	128,51	63	181	129
57	35	165	56,40	189,29	128,43	63	182	129
57	35	166	56,51	189,79	128,35	63	182	129

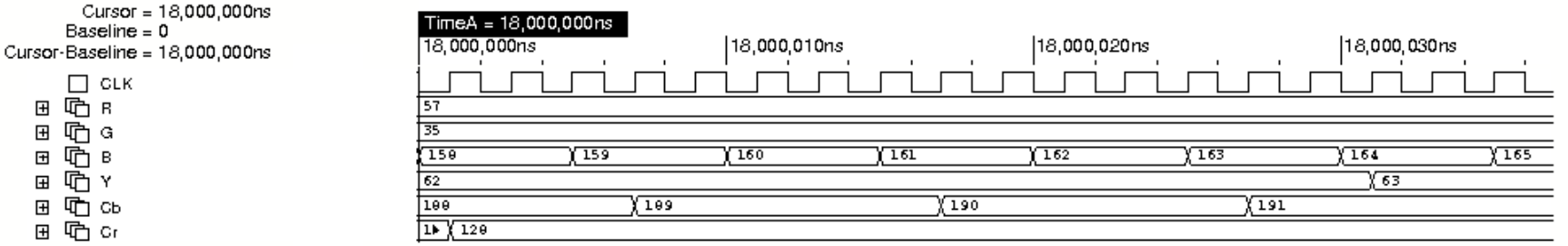
9 lentelė. Gautų A komponento paletės keitimo rezultatų palyginimas.

Komponentas B

Laiko diagramos (15, 16 pav.):



15 pav. Paletės keitimo iš RGB į YCbCr komponento B laiko diagrama.



16 pav. Paletės keitimo iš RGB į YCbCr komponento B laiko diagrama.

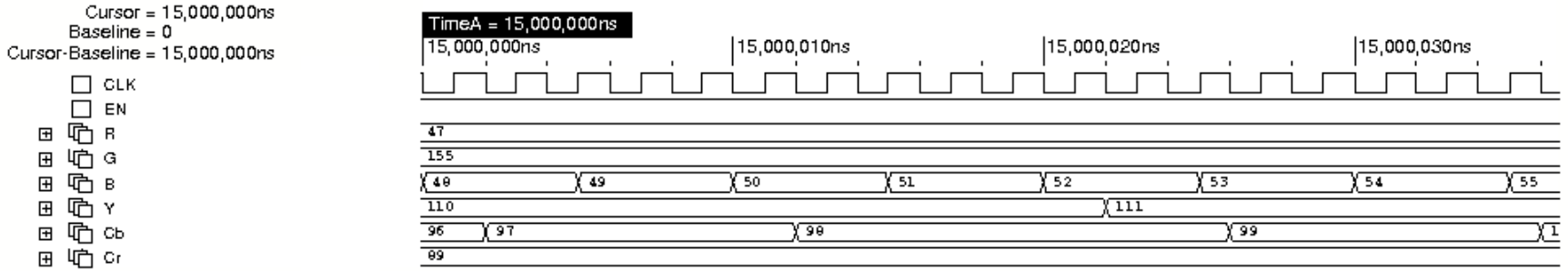
Kaip matome iš laiko diagramų, rezultatai išvedami prie kylančio sinchrosignalo, o tai atitinka komponento parametrus, taigi sinchrosignalas veikia korektiškai. Modeliavimo metu gautų rezultatų palyginimas (10 lentelė):

Įeinančios R, G, B reikšmės (pagal 17 pav. ir 18 pav. laiko diagramas)			**Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės**			**Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 17 pav. ir 18 pav. laiko diagramas)**		
R	G	B	Y	Cb	Cr	Y	Cb	Cr
47	155	48	110,51	92,72	82,70	110	92	88
47	155	49	110,62	93,22	82,62	110	93	88
47	155	50	110,74	93,72	82,54	110	93	88
47	155	51	110,85	94,22	82,46	110	94	88
47	155	52	110,97	94,72	82,37	111	94	87
47	155	53	111,08	95,22	82,29	111	95	87
47	155	54	111,19	95,72	82,21	111	95	87
57	35	158	55,60	185,79	129,00	62	188	128
57	35	159	55,71	186,29	128,92	62	189	128
57	35	160	55,83	186,79	128,84	62	189	128
57	35	161	55,94	187,29	128,76	62	190	128
57	35	162	56,06	187,79	128,67	62	190	128
57	35	163	56,17	188,29	128,59	62	191	128
57	35	164	56,28	188,79	128,51	63	191	128

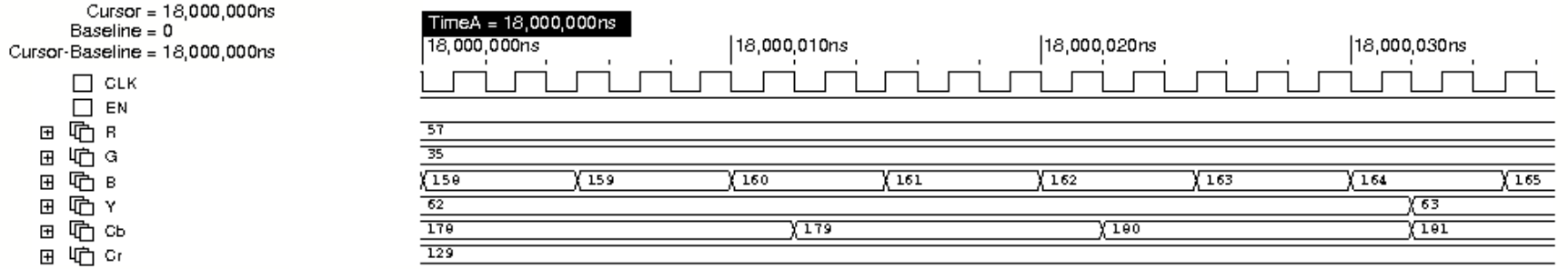
10 lentelė. Gautų B komponento paletės keitimo rezultatų palyginimas.

Komponentas C

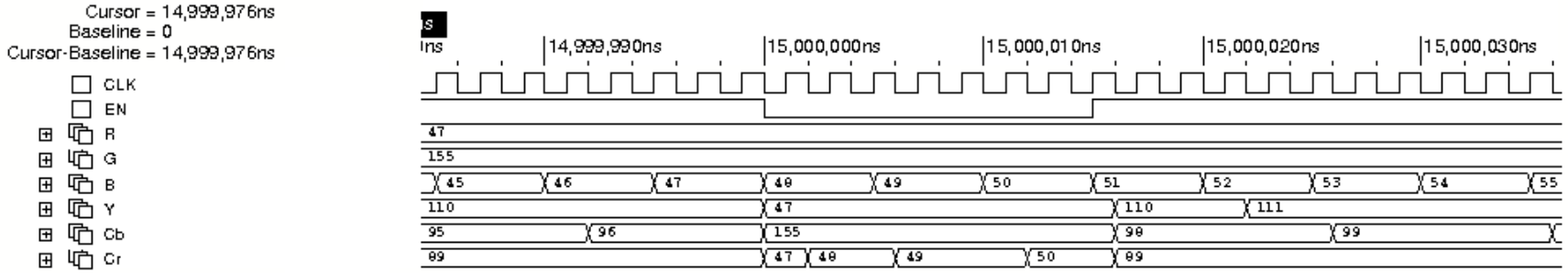
Laiko diagramos (17, 18, 19 pav.):



17 pav. Paletės keitimo iš RGB į YCbCr komponento C laiko diagrama.



18 pav. Paletės keitimo iš RGB į YCbCr komponento C laiko diagrama.



19 pav. Paletės keitimo iš RGB į YCbCr komponento C laiko diagrama.

Kaip matome iš laiko diagramų, rezultatai išvedami prie krentančio sinchrosignalo, o tai atitinka komponento parametrus, taigi sinchrosignalas veikia korektiškai. Taip pat iš 21 pav. matome, jog nustačius *En* = 0 komponentas nebeatlieka paletės keitimo, o tiesiog perduoda R, G, B įėjimo signalų reikšmes į Y, Cb ir Cr išėjimo signalus, taigi *Enable* signalas veikia korektiškai. Modeliavimo metu gautų rezultatų palyginimas (11 lentelė):

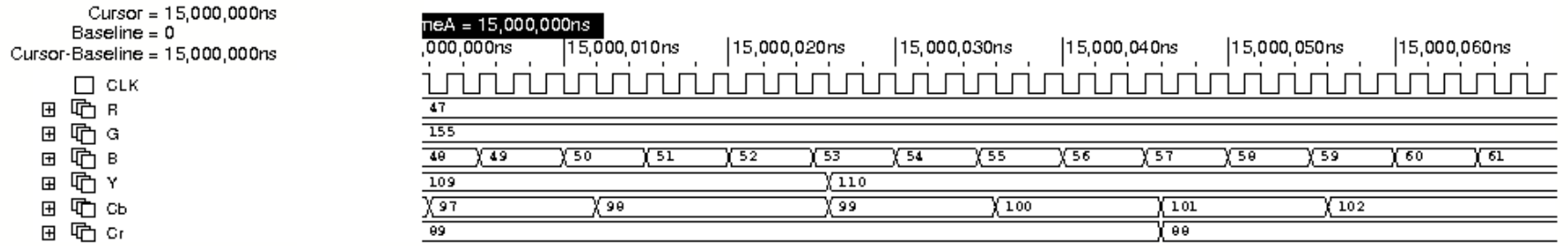
Įeinančios R, G, B reikšmės (pagal 19 pav. ir 20 pav. laiko diagramas)			**Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės**			**Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 19 pav. ir 20 pav. laiko diagramas)**		
R	G	B	Y	Cb	Cr	Y	Cb	Cr
47	155	48	110,51	92,72	82,70	110	97	89
47	155	49	110,62	93,22	82,62	110	97	89

Įeinančios R, G, B reikšmės (pagal 19 pav. ir 20 pav. laiko diagramas)			Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės			Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 19 pav. ir 20 pav. laiko diagramas)		
47	155	50	110,74	93,72	82,54	110	98	89
47	155	51	110,85	94,22	82,46	110	98	89
47	155	52	110,97	94,72	82,37	111	98	89
47	155	53	111,08	95,22	82,29	111	99	89
47	155	54	111,19	95,72	82,21	111	98	89
57	35	158	55,60	185,79	129,00	62	178	129
57	35	159	55,71	186,29	128,92	62	178	129
57	35	160	55,83	186,79	128,84	62	179	129
57	35	161	55,94	187,29	128,76	62	179	129
57	35	162	56,06	187,79	128,67	62	180	129
57	35	163	56,17	188,29	128,59	62	180	129
57	35	164	56,28	188,79	128,51	63	181	129

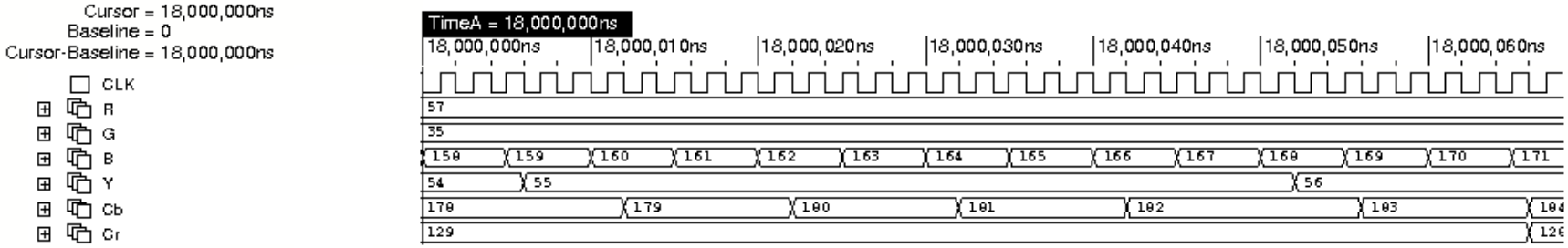
11 lentelė. Gautų C komponento paletės keitimo rezultatų palyginimas.

Komponentas D

Laiko diagramos (20, 21 pav.):



20 pav. Paletės keitimo iš RGB į YCbCr komponento D laiko diagrama.



21 pav. Paletės keitimo iš RGB į YCbCr komponento D laiko diagrama.

Kaip matome iš laiko diagramų, rezultatai išvedami prie krentančio sinchrosignalo, o tai atitinka komponento parametrus, taigi sinchrosignalas veikia korektiškai. Modeliavimo metu gautų rezultatų palyginimas (12 lentelė):

Įeinančios R, G, B reikšmės (pagal 22 pav. ir 23 pav. laiko diagramas)			**Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės**			**Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 22 pav. ir 23 pav. laiko diagramas)**		
R	G	B	Y	Cb	Cr	Y	Cb	Cr
47	155	48	110,51	92,72	82,70	109	97	89
47	155	49	110,62	93,22	82,62	109	97	89
47	155	50	110,74	93,72	82,54	109	98	89
47	155	51	110,85	94,22	82,46	110	100	89
47	155	52	110,97	94,72	82,37	111	100	89
47	155	53	111,08	95,22	82,29	110	99	89
47	155	54	111,19	95,72	82,21	110	99	89
47	155	55	111,31	96,22	82,13	110	100	89
47	155	56	111,42	96,72	82,05	110	100	89
57	35	158	55,60	185,79	129,00	54	178	129
57	35	159	55,71	186,29	128,92	55	178	129
57	35	160	55,83	186,79	128,84	55	179	129
57	35	161	55,94	187,29	128,76	55	179	129
57	35	162	56,06	187,79	128,67	55	180	129
57	35	163	56,17	188,29	128,59	55	180	129

Įeinančios R, G, B reikšmės (pagal 22 pav. ir 23 pav. laiko diagramas)			**Paletės keitimo iš RGB į YCbCr naudojant standartinę formulę gautos reikšmės**			**Paletės keitimo iš RGB į YCbCr reikšmės gautos naudojant A komponentą (pagal 22 pav. ir 23 pav. laiko diagramas)**		
57	35	164	56,28	188,79	128,51	55	181	129
57	35	165	56,40	189,29	128,43	55	181	129
57	35	166	56,51	189,79	128,35	55	182	129

12 lentelė. Gautų D komponento paletės keitimo rezultatų palyginimas.

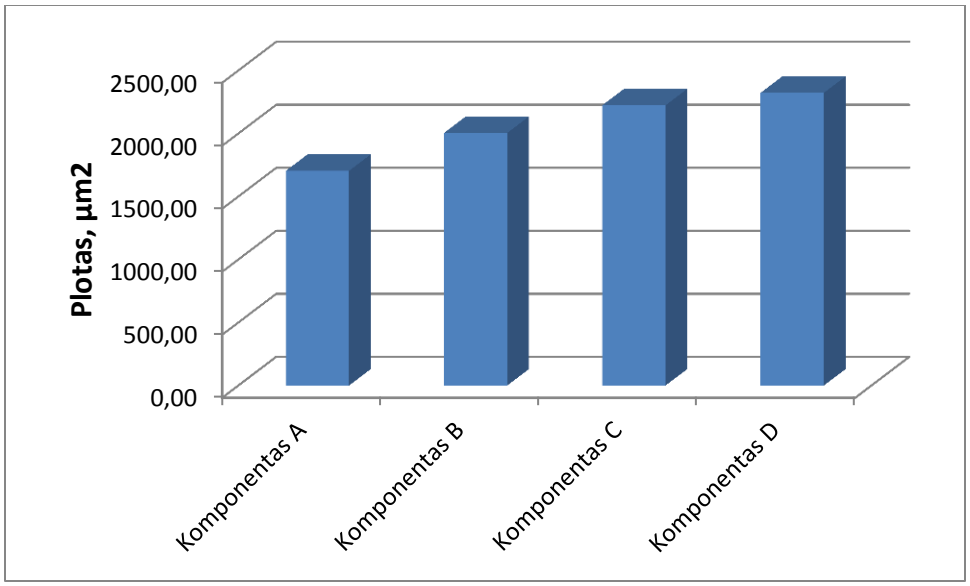
4.3. Komponentų sintezės rezultatai

Sintezė atlikta naudojant *gscl45nm* biblioteką ir vidutinius (*medium*) reikalavimus sintezuojamo komponento galiai, plotui bei sujungimams. Toliau pateikiami susintezuotų komponentų parametrai (13 lentelė):

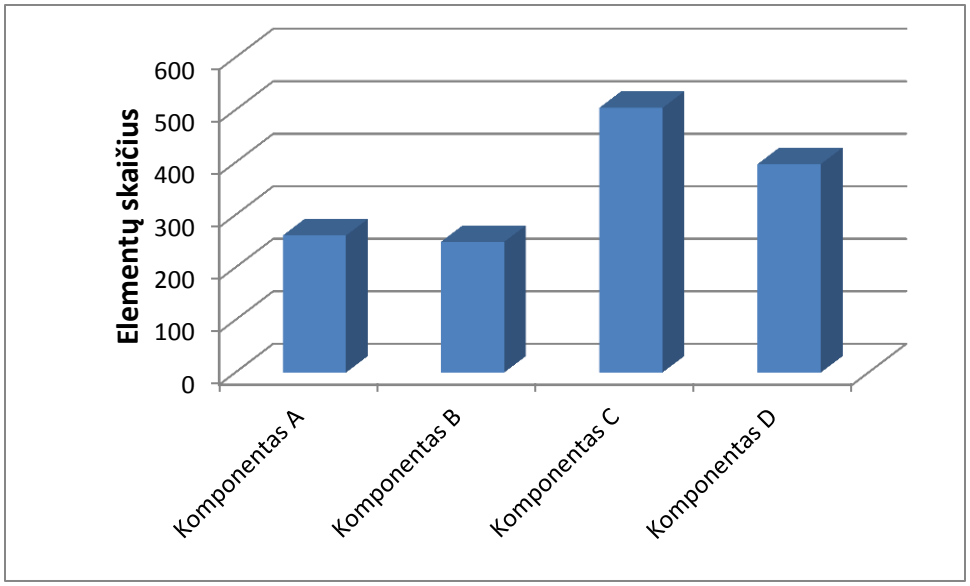
Parametras	**Komponentas A**	**Komponentas B**	**Komponentas C**	**Komponentas D**
Plotas	1703.559 μm^2	2004.380 μm^2	2223.543 μm^2	2322.565 μm^2
Elementų skaičius	261	249	504	397
Sujungimų skaičius	352	361	585	489
Galia	1.0146 mW	1.3195 mW	1.3805 mW	1.5810 mW
Vėlinimas	0.10 ns	0.10 ns	0.07 ns	0.07 ns

13 lentelė. Komponentų sintezės rezultatų palyginimas.

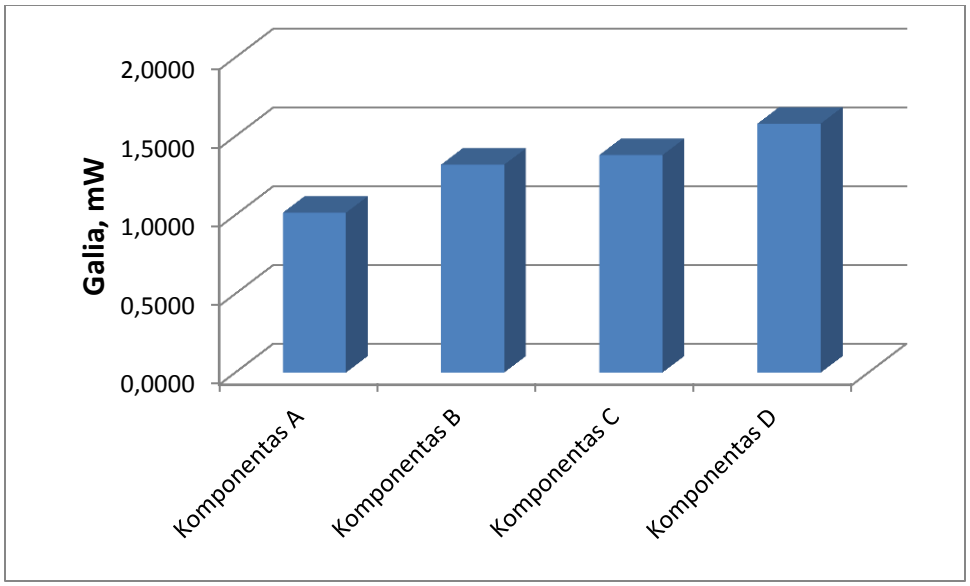
Iš gautų sintezės duomenų matome, kad komponentų parametrai ganėtinai skirtingi, taigi parametrizuotas komponentas gali pritraukti platesnį naudotojų ratą, kurie pagal savo norus gali išsirinkti sau tinkamą komponentą. Komponentų parametrai, vaizdingumo dėlei, palyginti stulpelinėse diagramose (22, 23, 24, 25 pav.):



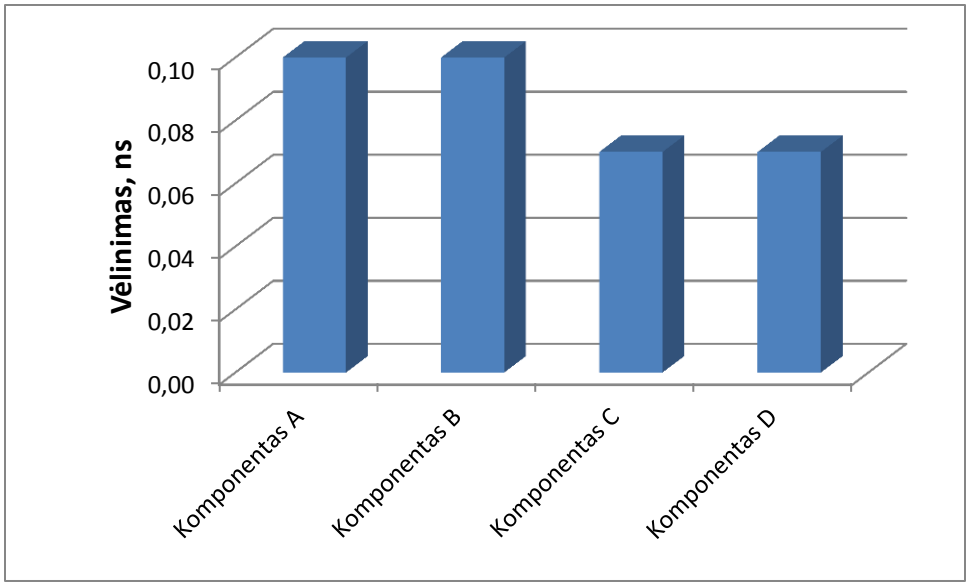
22 pav. Stulpelinė komponentų plotų diagrama.



23 pav. Stulpelinė komponentų elementų skaičiaus diagrama.



24 pav. Stulpelinė komponentų galios diagrama.



25 pav. Stulpelinė komponentų vėlinimo diagrama.

4.4. Gautų rezultatų išvados

1. Pasirinktas spalvų paletės keitimo iš RGB į YCbCr komponentas, kadangi šis komponentas yra svarbus JPEG realizacijose, savo apimtimi nėra per daug

mažas arba per daug didelis, turi platesnę pritaikymo sritį (ne tik JPEG realizacijose), turi nemažas parametrizavimo galimybes.

2. Atliktas sukurto parametrizuoto spalvų paletės keitimo iš RGB į YCbCr komponento modeliavimas bei sintezė. Pastebėta, jog sugeneruoti komponentai ganėtinai skirtingi savo savybėmis, taigi šitaip sukuriama didesnė komponentų įvairovė ir didesnė pakartotinio panaudojimo galimybė.
3. Kaip matome iš gautų modeliavimo rezultatų gautieji paletės keitimo rezultatai šiek tiek skiriasi nuo rezultatų apskaičiuotų naudojant standartines paletės keitimo formules. Taip yra dėl skaičiavimo metodų netikslumo bei apvalinimo paklaidų, tačiau nedideli rezultatų svyravimai neturi didelės įtakos galutiniam rezultatui.

5. Išvados

1. Yra nedaug laisvai prieinamų atvirojo kodo JPEG realizacijų, aprašytų aparatūros aprašymo kalbomis. Taip pat dauguma atvirojo kodo realizacijų turi prastokas vartotojo instrukcijas arba jų išvis neturi.
2. JPEG realizacijas galima aprašyti įvairiomis HDL kalbomis, tačiau beveik visos realizacijos aprašytos VHDL kalba. Taip yra dėl didelio VHDL populiarumo ir laiko patikrinto patikimumo.
3. Ištirtos atvirojo kodo JPEG realizacijos yra pritaikytos konkrečiai technologijai ir nepalaiko skirtingų technologijų.
4. Prieinamos atvirojo kodo JPEG realizacijos, sukurtos neatsižvelgiant į pakartotinio naudojimo (*reuse*) metodologiją, todėl jas sunku pritaikyti savo reikmėms.
5. Sukurtas parametrizuotas spalvų paletės keitimo iš RGB į YCbCr komponentas ir atliktas sugeneruotų (parametrus pasirinkus atsitiktinai) komponentų modeliavimas bei sintezė. Pastebėta, jog sugeneruoti komponentai ganėtinai skirtingi savo savybėmis, taigi šitaip sukuriama didesnė komponentų įvairovė ir didesnė pakartotinio panaudojimo galimybė.
6. Sukurtų komponentų paletės keitimo rezultatai šiek tiek skiriasi nuo laukiamų apskaičiuotų rezultatų naudojant standartinį metodą. Taip atsitinka dėl apvalinimo paklaidų, taip pat dėl skaičiavimo metodų netikslumo. Nedideli rezultatų svyravimai neturi didelės įtakos galutiniam rezultatui.
7. Vystant projektą ateityje įmanoma dar plačiau išplėsti parametrizuoto spalvų paletės keitimo iš RGB į YCbCr komponento parametrizavimo galimybes. Taip pat įmanoma įgyvendinti ir kitus JPEG realizacijų komponentus, naudojant pakartotinio naudojimo metodologiją.

6. Naudotos literatūros sąrašas

[1] Straipsnis iš duomenų bazės:

Debra A. Lelewer and Daniel S. Hirschberg, Data Compression [žiūrėta 2011-01-22]. Prieiga per internetą: <http://www.ics.uci.edu/~dan/pubs/DataCompression.html>

[2] Knyga:

Steven W. Smith, Ph.D., The Scientist and Engineer's Guide to Digital Signal Processing, Chapter 27: Data Compression [žiūrėta 2011-01-22]. Prieiga per internetą: < http://www.dspguide.com>

[3] Straipsnis iš duomenų bazės:

International Organization for Standardization [žiūrėta 2011-01-22]. Prieiga per internetą: <www.iso.org>

[4] Straipsnis iš duomenų bazės:

Joint Photographic Experts Group [žiūrėta 2011-01-22]. Prieiga per internetą: <http://www.jpeg.org/public/jfif.pdf>

[5] Straipsnis iš duomenų bazės:

Mohammed Elbadri, Raymond Peterkin, Voicu Groza, Dan Ionescu, Abdulmotaleb El Saddik. Hardware support of JPEG [žiūrėta 2011-01-22]. Prieiga per internetą: <http://ieeexplore.ieee.org>

[6] Straipsnis iš duomenų bazės:

Ramesh Neelamani, Ricardo de Queiroz, Zhigang Fan, Sanjeeb Dash, Richard G. Baraniuk. JPEG Compression History Estimation for Color Images [žiūrėta 2011-01-22]. Prieiga per internetą: <http://ieeexplore.ieee.org>

[7] Straipsnis iš duomenų bazės:

VHDL Analysis and Standardization Group [žiūrėta 2011-01-22]. Prieiga per internetą: <http://www.eda.org/vasg>

[8] Straipsnis iš duomenų bazės:

Stuart Sutherland (1997).Verilog-1995 Quick Reference Guide [žiūrėta 2011-01-22]. Prieiga per internetą <http://www.sutherland-hdl.com/online_verilog_ref_guide/vlog_ref_top.html>

[9] Straipsnis iš duomenų bazės:

Open SystemC Initiative [žiūrėta 2011-01-22]. Prieiga per internetą <http://www.systemc.org>

[10] Knyga:

Micheal Keating, Pierre Bricaud. Reuse Methodology Manual for system-on-a-chip designs, third edition. Kluwer Academic Publishers, 2002.

[11] Knyga:

Vytautas Štuikys, Robertas Damaševičius. Modelių ir programų abstrakčiosios transformacijos. Kauno technologijos universitetas, 2008.

[12] Specifikacijų aprašas:

V. Štuikys, G. Ziberkas, R. Damaševičius. Open PROMOL – the program modification language manual. Kauno technologijos universitetas, 2000-07-31.

[13] Straipsnis iš duomenų bazės:

Požymių diagramos (Feature diagrams) [žiūrėta 2012-03-15]. Prieiga per internetą < http://soften.ktu.lt/~damarobe/T120M608/Lab3/lt_FD.htm>

[14] Straipsnis iš duomenų bazės:

Bendrinių komponentų kūrimo metodikos tyrimas [žiūrėta 2012-03-15]. Prieiga per internetą <http://soften.ktu.lt/~damarobe/T120M608/Lab3/Lab3.html>