**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
INFORMATIKOS FAKULTETAS
Programų inžinerijos katedra


Darius Birvinskas
Ignas Martišius

# Tiesioginės ir atvirkštinės dvimatės diskrečiosios kosinusinės transformacijos algoritmų tyrimas, naudojant FPGA matricas

Vienlusčių sistemų magistro baigiamasis darbas

Vadovas
prof. Vacius Jusas

KAUNAS, 2011

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
INFORMATIKOS FAKULTETAS
Programų inžinerijos katedra


TVIRTINU
Katedros vedėjas
prof. Eduardas Bareiša
2011-05-19

# Tiesioginės ir atvirkštinės dvimatės diskrečiosios kosinusinės transformacijos algoritmų tyrimas, naudojant FPGA matricas

Vienlusčių sistemų magistro baigiamasis darbas

Recenzentas

Vadovas
prof. Vacius Jusas
2011-05-19

Atliko
IFM-9/5 gr. studentas
Darius Birvinskas
2011-05-19

IFM-9/5 gr. studentas
Ignas Martišius
2011-05-19

KAUNAS, 2011

# • Turinys

Santrauka....5
Summary....6
1 Įvadas....7
2 Analitinė dalis....9
2.1 Diskrečioji kosinusinė transformacija....9
2.1.1 Vienmatė DCT....9
2.1.2 Dvimatė DCT....10
2.2 DCT savybės....12
2.2.1 Dekoreliacija....12
2.2.2 Energijos suspaudimas....12
2.2.3 Atskiriamumas....13
2.2.4 Simetrija....13
2.2.5 Ortogonalumas....14
2.3 DCT taikymai....14
2.4 Vaizdo suspaudimo principai....14
2.4.1 JPEG....14
2.4.2 Kodavimas....15
2.4.2.1 Spalvų erdvės keitimas....15
2.4.2.2 Atrinkimas....15
2.4.2.3 Blokų išskyrimas....15
2.4.2.4 Diskrečioji kosinusinė transformacija....15
2.4.2.5 Kvantavimas ir zigzag skenavimas....17
2.4.2.6 Kintamo eilutės ilgio kodavimas....17
2.4.3 Dekodavimas....18
2.4.3.1 Kintamo eilutės ilgio dekodavimas....18
2.4.3.2 Atvirkštinis kvantavimas ir deskenavimas....19
2.4.3.3 Atvirkštinė diskrečioji kosinusinė transformacija (IDCT)....19
2.5 DCT architektūros....20
2.5.1 Eilučių/stulpelių metodas....20
2.5.2 Tiesioginis 2-D skaičiavimas....21
2.5.3 Apytiksliai algoritmai....22
2.6 Pakartotinis komponentų panaudojimas....23
2.7 Sistemų aprašymo kalbos....24
2.7.1 Abstrakcijos lygiai....25
2.7.2 VHDL bibliotekos ir paketai....27
2.7.2.1 std_logic_1164....27
2.7.2.2 numeric_std....27
2.7.2.3 textio....27
2.8 Analitinės dalies išvados....28
3 Projektinė dalis....29
3.1 Tyrimui pasirinkti DCT ir IDCT algoritmai....29
3.1.1 W. Chen algoritmas....29
3.1.2 Loeffler algoritmas....31
3.1.3 BinDCT algoritmas....33
3.2 Tyrimo metodika....35
3.2.1 Matematinis modeliavimas....35
3.2.2 Projektavimo eiga....37
3.2.3 FPGA matricos....38

3.3 Papildoma aparatūra dvimačiam algoritmui....39
3.3.1 Valdymo automatas (FSM)....41
3.3.2 Skaitliukas....44
3.3.3 Buferis....46
3.3.4 Skaičiavimo blokas....48
3.4 Projektinės dalies išvados....49
4 Tyrimo dalis....50
4.1 Chen algoritmo matematinis modeliavimas....51
4.2 Chen algoritmo aparatinė realizacija....53
4.2.1 Laikinės diagramos....54
4.2.2 Rezultatai....55
4.3 Loeffler algoritmo matematinis modeliavimas....58
4.4 Loeffler algoritmo aparatinė realizacija....60
4.4.1 Laikinės diagramos....60
4.4.2 Rezultatai....62
4.5 BinDCT algoritmo matematinis modeliavimas....65
4.6 BinDCT algoritmo aparatinė realizacija....67
4.6.1 Laikinės diagramos....67
4.6.2 Rezultatai....68
4.7 FPGA galios ir greitaveikos palyginimas....71
4.8 Tyrimo dalies išvados....73
5 Išvados....75
Literatūra....77
Priedas Nr. 1. Straipsniai....82
Priedas Nr. 2. Algoritmų VHDL modeliai....91

## Santrauka

Skaitmeniniams vaizdo apdorojimo įrenginiams labai svarbūs kodavimo, atpažinimo, vaizdo suspaudimo algoritmai. Vaizdo apdorojimo funkcijas pritaikant nešiojamuose įrenginiuose, būtinu tampa ne vien šių algoritmų kokybės ar greitaveikos, bet ir galios suvartojimo problemos sprendimas.

Viena svarbiausių apdorojimo algoritmų dalių yra diskrečioji kosinusinė transformacija (Discrete cosine transform - DCT) bei atvirkštinė diskrečioji kosinusinė transformacija (Inverse discrete cosine transform - IDCT). Šios transformacijos yra sudėtingiausios kodavimo bei dekodavimo procese, sunaudojančios apie ketvirtį visų matematinių operacijų kodavimo algoritme. Didelis šios sistemos sudėtingumas lemia didelį galios suvartojimą, todėl būtina parinkti tinkamiausią algoritmą, naudojantį kiek įmanoma mažiau aparatinės įrangos, taip taupant galią, tuo pačiu neprarandant apdorojamo vaizdo kokybės.

Šiame darbe pristatomi dvimačių DCT bei IDCT algoritmų aparatinio įgyvendinimo metodai. Tiriami populiariausi ir dažniausiai praktikoje naudojami DCT bei IDCT algoritmai. Tyrimas atliekamas naudojant FPGA matricas. Tuo siekiama išsiaiškinti efektyviausią, sparčiausią, mažiausiai skaičiavimo resursų bei galios suvartojantį algoritmą. Taip pat atkreipiamas dėmesys į algoritmo tinkamumą aparatinei realizacijai, tokios realizacijos įgyvendinimo problemas. Rezultatuose pateikiami šių algoritmų greitaveikos ir kokybės palyginimai, teigiami ir neigiami aspektai, tinkamumas aparatūrai.

Prieš realizuojant algoritmus, atliekamas matematinis modeliavimas. Modeliuojant tiriamas algoritmo veikimas ir kokybės charakteristikos.

## Summary

Digital image processing devices depend on coding, edge detection and image compression algorithms. The ever increasing demand for image processing in mobile wireless devices it is becoming crucial to ensure not only the quality and speed of these algorithms, but also the problem of power dissipation.

One of the most important parts of any image processing algorithm is the discrete cosine transform (DCT) and the inverse discrete cosine transform (IDCT). These transforms are the most complex parts in the coding and decoding process, using up to a quarter of all the mathematical operations in the coding algorithm. The complexity of this system leads to great power dissipation

For the increasing number of portable wireless devices, a key design constraint is power dissipation. Limited battery life constrains portable devices to low power dissipation; advances in battery life do not grow as fast as the density and the operating frequency of ASICs [1]. The ever-growing circuit densities and operating frequencies of ASICs only result in greater power dissipation. Therefore it is necessary to find the right algorithm, using the minimum of power, while still providing sufficient coding quality and speed.

This thesis focuses on a hardware implementation of DCT and IDCT algorithms. Most popular and practically used algorithms are explored. FPGA gate arrays are used for testing of these algorithms. The main goal is to find the most effective, fastest algorithm with the least power dissipation. Also we focus on the algorithm's suitability for a hardware implementation and the problems of such design. The results display a comparison of the algorithm's speed, coding quality, positive and negative aspects of hardware design.

Before the hardware implementations, a mathematical modeling of the algorithms is performed for testing the performance and quality characteristics.

# 1 Įvadas

Skaitmeninė vaizdo informacija šiuo metu yra neatskiriama verslo, mokslo ir pramogų dalis. Ji naudojama visur – skaitmeninėje televizijoje, internete, mobiliuosiuose telefonuose. Dauguma žmonių, to nė nežinodami, turi ir kasdien naudoja vaizdo saugojimo, kodavimo ir apdorojimo įrenginius, esančius jų namuose, ar tiesiog gulinčius jų kišenėse.

Nuo šių įrenginių neatsiejamas skaitmeninis vaizdo kodavimas, apdorojimas bei suspaudimas. Pavyzdžiui tam, kad parodyti ekrane vieną TV kokybę atitinkantį kadrą, reikia 720 kilobaitų vaizdo duomenų. Visa tai susideda į 100 gigabaitų duomenų 90-ties minučių vaizdo įrašui. Saugoti tokį informacijos kiekį brangu ir nepatogu net ir galingose šiuolaikinėse talpyklose. Dar sunkiau ir brangiau, galios bei greitaveikos prasme, tokius kiekius informacijos perduoti ar apdoroti, taigi vaizdinės informacijos suspaudimas tampa būtinas. Suspaudimas sutaupo apie 85-95 % perdavimo bei apdorojimo resursų.

Nemažiau svarbus yra vaizdinės informacijos apdorojimas, t. y. tam tikros informacijos vaizde radimo ir išrinkimo algoritmai, tokie kaip veido, numerių, kontūrų atpažinimo (edge detection).

Vis plačiau naudojamoms mobilioms vaizdo apdorojimo sistemoms galios suvartojimas labai svarbus dėl naudojamų baterijų talpos, todėl stengiamasi rasti kompromisą tarp našumo ir galios. Kadangi baterijų talpa yra ribota ir nedidėja taip sparčiai, kaip mikroprocesorių galia, nešiojamų prietaisų komponentai priversti taupyti elektros energiją. Vis didėjanti sistemų integracija ir taktinis dažnis dar labiau apsunkina šią problemą, didindamas galios suvartojimą [1].

Vienas plačiausiai skaitmeninio vaizdo kodavime naudojamų algoritmų yra diskrečioji kosinusinė transformacija (DCT). Ji naudojama vaizdo ir garso apdorojimo, telekomunikacijos signalų apdorojimo bei duomenų suspaudimo algoritmams, informacijos iš vaizdo išskyrimui, vaizdo rekonstrukcijai bei filtravimui [2]

DCT algoritmo įgyvendinimas labai sudėtingas matematine prasme. Tai daugiausiai skaičiavimų atliekanti dalis daugelyje vaizdo apdorojimo algoritmų, sunaudojanti apie 21% visų skaičiavimo resursų kodavimo procese. IDCT skaičiavimui reikalingos operacijos taip pat sudaro didžiąją dalį dekodavimui reikalingų skaičiavimų. Pavyzdžiui populiariame H.263 kodavimo standarte, naudojančiame DCT bei IDCT transformacijas, algoritmui apdorojant 144 x 176 vaizdo taškų video 25 kadrų per sekundę dažniu, tiesioginiams DCT ir IDCT algoritmams reikia daugiau nei 112 milijonų daugybos ir sudėties operacijų per sekundę. Toks skaičiavimų sudėtingumas reikalauja didelių aparatinių resursų, sunaudojančių daug galios. Todėl

matematiškai paprastesnių ir greitesnių DCT bei IDCT algoritmų paieška yra svarbi tyrimų sritis [3].

Dažniausiai pasirenkamas programinis šių algoritmų įgyvendinimas, tam naudojant bendros paskirties procesorius. Toks įgyvendinimas paprastas ir nereikalaujantis specializuotų komponentų kuriamame prietaise, tačiau toks sprendimas nėra optimalus greičio ir galios suvartojimo prasme.

Atsižvelgiant į šių algoritmų panaudojimo galimybes ir paplitimą, optimalus sprendimas šiems dažnai taikomiems algoritmams yra specializuotas procesorius (Application-specific integrated circuit – ASIC). Toks įrenginys leidžia padidinti kodavimo ir dekodavimo greitaveiką ir nepadidina galios suvartojimo lyginat su realizacija bendros paskirties procesoriuje. Šiuo būdu įmanoma pasiekti norimą greitaveiką, nenaudojant papildomų aparatūros resursų ir taupant galią.

Šio darbo tikslas – surasti tinkamiausią DCT bei IDCT algoritmą specializuotam aparatiniam pritaikymui, išsiaiškinti galimą greitaveiką ir galios suvartojimą, bei užimamą lusto plotą.

# 2 Analitinė dalis

## 2.1 Diskrečioji kosinusinė transformacija

Kaip ir kitos banginės transformacijos, DCT naudojama išskaidyti signalo informacijai. Po skaidymo kiekvienas transformacijos koeficientas gali būti apdorojamas atskirai, neprarandant kodavimo efektyvumo. Šiame skyriuje aprašomos svarbiausios DCT ir IDCT savybės.

### 2.1.1 Vienmatė DCT

Dažniausiai pasitaikantis vienmatės DCT aprašas, N ilgio sekai yra (1) formulė:

$$Y(u)=\alpha(u)\sum_{x=0}^{N-1} f(x)\cos\left(\frac{\pi(2x+1)\cdot u}{2N}\right) \qquad (1)$$

čia:

- $u$ = 0, 1, 2, ... , N-1.
- $\alpha(0)=\frac{1}{\sqrt{2}}$ ir $\alpha(j)$ = 1 jei $j \neq 0$.

IDCT aprašoma (2) formule.

$$f(x)=\sum_{u=0}^{N-1} \alpha(u)Y(u)\cos\left(\frac{\pi(2x+1)\cdot u}{2N}\right) \qquad (2)$$

čia:

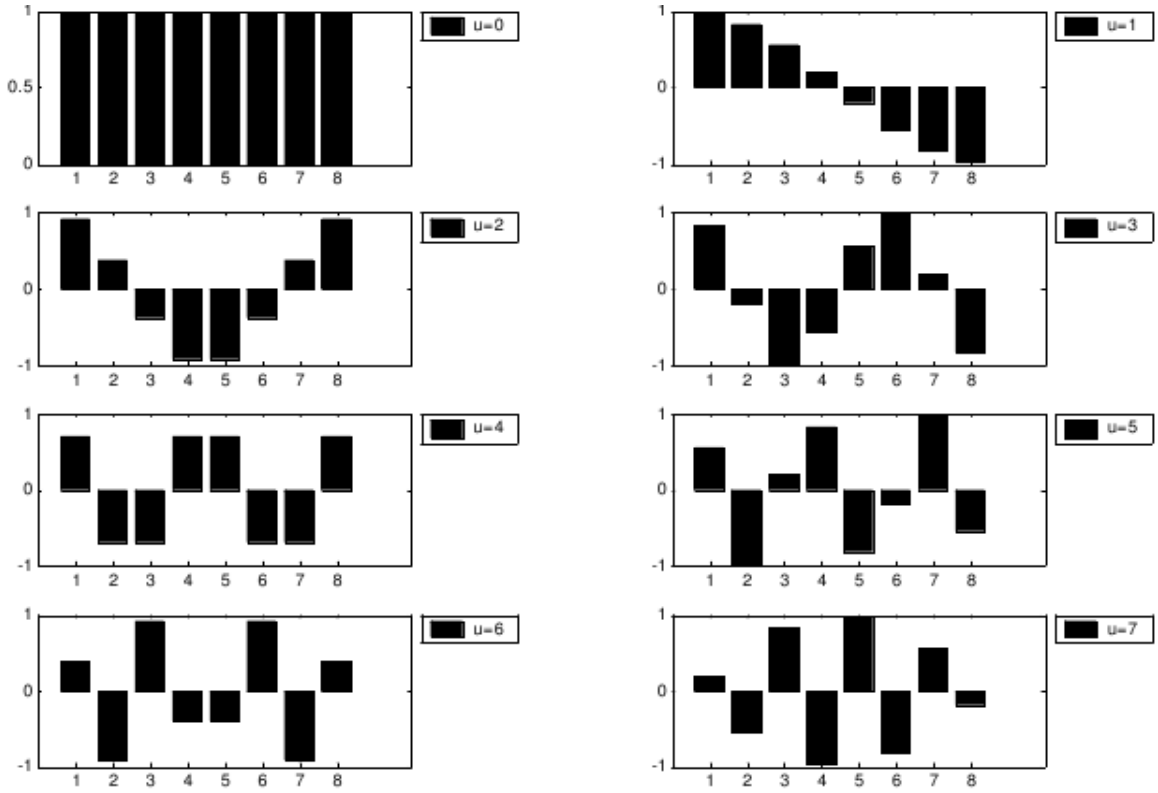
- $x$ = 0, 1, 2, ... , N-1.
- $\alpha(0)=\frac{1}{\sqrt{2}}$ ir $\alpha(j)$ = 1 jei $j \neq 0$.

Iš (1) formulės matyti, kad kai $u$ = 0, tuomet $Y(u=0)=\sqrt{\frac{1}{N}}\sum_{x=0}^{N-1} f(x)$ . Tai reiškia, jog pirmasis transformacijos koeficientas yra vidutinė įėjimo sekos reikšmė. Literatūroje ši reikšmė vadinama DC koeficientu. Visi likę koeficientai vadinami AC koeficientais (terminas kilęs iš elektronikos srities, kur DCT naudota elektrinių signalų apdorojimui). Atmetus $f(x)$ ir $\alpha(u)$ komponentus (1) formulėje, nubraižome funkcijos $\sum_{x=0}^{N-1}\cos\left(\frac{\pi(2x+1)\cdot u}{2N}\right)$ grafikus, kuriose:

- N = 8.
- $0 \leq u \leq 7$.

Grafikai pateikti 2.1 pav. Kaip pastebėta anksčiau, kai u = 0, gaunama pastovi reikšmė (DC koeficientas), tuo tarpu likę grafikai ($u$ = 1,2, ... , 7) vaizduoja vis didėjančio dažnio

periodines funkcijas. Šios funkcijos vadinamos bazinėmis kosinuso funkcijomis. Bazinės funkcijos yra ortogonalios: t. y. bet kurių dviejų šių funkcijų daugybos ir sumos visuose taškuose rezultatas yra nulis, o bet kurios funkcijos daugybos iš savęs ir sumos visuose taškuose rezultatas yra konstanta. Ortogonalios periodinės funkcijos yra nepriklausomos. Nei vienos bazinės funkcijos negalima išreikšti per kitas bazines funkcijas [4].




*2.1 pav. Vienmatės DCT transformacijos bazinės funkcijos (N = 8)*

Jei įėjimo seka turi daugiau nei N narių, ją galima suskaidyti į N ilgio dalines sekas ir taikyti DCT/IDCT transformaciją kiekvienai iš jų. Svarbu pažymėti, kad atliekant tokius skaičiavimus bazinių funkcijų reikšmės nekinta. Kinta tik *f(x)* sekos reikšmės. Tai labai svarbi savybė, kuri reiškia, kad bazines funkcijas galima suskaičiuoti iš anksto, ir atliekant skaičiavimus jas dauginti iš įėjimo sekų. Tai sumažina skaičiavimų kiekį ir padidina kodavimo efektyvumą.

### 2.1.2 Dvimatė DCT

Šiame darbe tiriama dvimatės DCT transformacijos realizacija. Dvimatė transformacija praplečia aukščiau aprašytus teiginius dvimatei erdvei.

$$Y(k,l)=\frac{2}{N}\alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}x(m,n)\cos\left(\frac{(2m+1)\pi k}{2\mathrm{N}}\right)\cos\left(\frac{(2n+1)\pi l}{2\mathrm{N}}\right) \qquad (3)$$

Dviejų matmenų (2-D) DCT transformacija atliekama naudojant (3) formulę, čia:

- $k$ – horizontalus erdvinis dažnis skaičiams $0 \le k \le N - 1$.
- $l$ – vertikalus erdvinis dažnis skaičiams $0 \le l \le N - 1$.
- $\alpha(0)=\frac{1}{\sqrt{2}}$ ir $\alpha(j) = 1$, jei $j \ne 0$.
- $x$ – vaizdo taško reikšmė koordinatėse $(m, n)$.
- $Y$ – DCT koeficientas koordinatėse $(k, l)$.

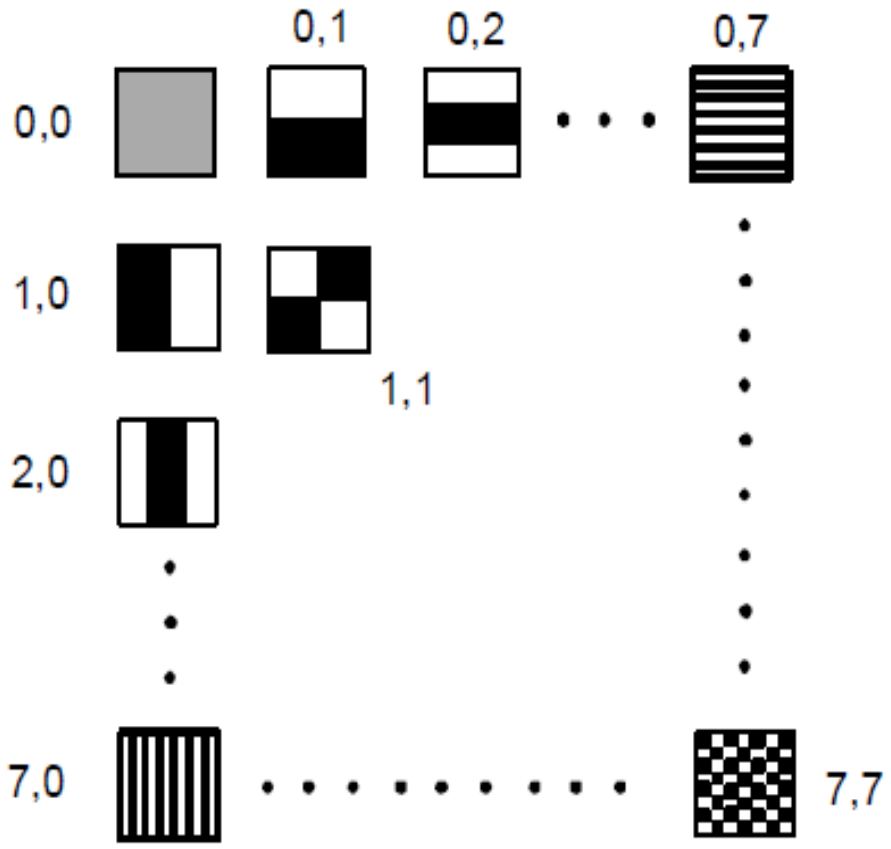
IDCT atliekama, naudojant formulę (4).

$$x(m,n)=\frac{2}{N}\alpha(k)\alpha(l)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}Y(k,l)\cos\left(\frac{(2m+1)\pi k}{2N}\right)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \qquad (4)$$

čia:

- $k$ – horizontalus erdvinis dažnis skaičiams $0 \le k \le N - 1$.
- $l$ – vertikalus erdvinis dažnis skaičiams $0 \le l \le N - 1$.
- $\alpha(0)=\frac{1}{\sqrt{2}}$ ir $\alpha(j) = 1$, jei $j \ne 0$.
- $x$ – vaizdo taško reikšmė koordinatėse $(m, n)$.
- $Y$ – DCT koeficientas koordinatėse $(k, l)$.

Dvimatės DCT transformacijos bazinės funkcijos gaunamos sudauginus horizontaliai orientuotas vienmates bazines funkcijas su vertikaliai orientuotomis tomis pačiomis bazinėmis funkcijomis. Bazinės funkcijos, kai N = 8, pavaizduotos 2.2 pav. Viršutinė kairioji bazinė funkcija įgija konstantinę reikšmę ir vadinama DC koeficientu.




*2.2 pav. Bazinės dvimatės DCT funkcijos. Balta spalva žymi teigiamą amplitudę. Juoda spalva – neigiamą amplitudę. Pilka spalva – nulinė reikšmė*

## *2.2 DCT savybės*

Šiame skyriuje aprašomos pagrindinės matematinės DCT bei IDCT savybės, turinčios reikšmę vaizdinės informacijos apdorojimui.

### 2.2.1 Dekoreliacija

Pagrindinis vaizdinės informacijos kodavimo privalumas yra perteklinės informacijos tarp vaizdo taškų pašalinimas. Tai sukuria nekoreliuotus (tarpusavyje nesusijusius) transformacijos koeficientus, kuriuos galima koduoti nepriklausomai vienas nuo kito.

### 2.2.2 Energijos suspaudimas

Transformacijos efektyvumas gali būti matuojamas pagal tai, kokiu koeficientų kiekiu aprašomi įvedimo duomenys. Kuo mažiau reikia koeficientų, tuo efektyvesnė yra

transformacija. Tai leidžia pašalinti mažiau reikšmingus koeficientus su mažomis amplitudėmis, neįnešant papildomų netikslumų vaizde. DCT puikiai suspaudžia energiją smarkiai koreliuotuose vaizduose. Tyrimai rodo, kad energijos suspaudimas artėja prie optimalaus, kai vaizdo koreliacija artėja prie vieneto. Taikant DCT tokiems vaizdams gaunami beveik optimalūs rezultatai [5].

### 2.2.3 Atskiriamumas

Dvimatės transformacijos formules (3) ir (4) galima perrašyti taip:

$$Y(k,l) = \frac{2}{N}\alpha(k)\alpha(l)\sum_{n=0}^{N-1}\cos\left(\frac{(2m+1)\pi k}{2N}\right)\sum_{m=0}^{N-1}x(m,n)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (5)$$

čia:

- $k$ – horizontalus erdvinis dažnis skaičiams $0 \le k \le N - 1$.
- $l$ – vertikalus erdvinis dažnis skaičiams $0 \le l \le N - 1$.
- $\alpha(0) = \frac{1}{\sqrt{2}}$ ir $\alpha(j) = 1$ jei $j \ne 0$.
- $x$ – vaizdo taško reikšmė koordinatėse $(m, n)$.
- $Y$ – DCT koeficientas koordinatėse $(k, l)$.

Ši savybė, vadinama atskiriamumu, suteikia galimybę skaičiuoti $Y(u, v)$ dviem žingsniais, atliekant dvi vienmates DCT transformacijas. Pirma transformacija atliekama vaizdo eilutėms, o antra - stulpeliams. Tai taip pat gali būti taikoma IDCT transformacijai.

### 2.2.4 Simetrija

Išnagrinėjus (5) formulę, matome, kad operacijos, atliekamos koduojamo vaizdo eilutėms ir stulpeliams yra identiškos. Tokia transformacija vadinama simetriška. Atskiriama ir simetriška transformacija gali būti aprašyta tokia forma:

$$T = AfA \quad (6)$$

čia:

- $f$ - NxN koduojamo vaizdo matrica.
- $A$ - yra simetrinė NxN matrica, kurios elementai $a(i,j)$ gaunami pagal (7) formulę:

$$a(i, j) = \alpha(j)\sum_{j=0}^{N-1}\cos\left(\frac{\pi(2j+1)i}{2N}\right) \quad (7)$$

Tai labai naudinga savybė, kuri reiškia, kad transformacijų matrica gali būti paskaičiuota iš anksto ir vėliau pritaikyta vaizdui, taip ženkliai sumažinant reikalingų skaičiavimų kiekį.

### 2.2.5 Ortogonalumas

Išreiškus atvirkštinę funkciją iš (6) formulės, ji gali būti užrašoma (8) formule:

$$f = A^{-1}TA^{-1} \quad (8)$$

Kaip minėta ankščiau, DCT yra ortogonali funkcija, taigi atvirkštinė transformacijos $A$ matrica yra lygi jos transpozicijai. t.y.: $A^{-1} = A^{T}$. Ši savybė leidžia sumažinti skaičiavimo sudėtingumą.

## *2.3 DCT taikymai*

DCT transformacija paverčia baigtinę duomenų aibę skirtingo dažnio kosinuso funkcijų suma. Ši transformacija moksle ir inžinerijoje naudojama labai plačiai. Tai garso kodavimo mp3 formatu ir vaizdo kodavimo algoritmai, kur aukšto dažnio komponentai gali būti išmetami, įgalinant suspaudimą, kontūrų atpažinimą, spektrinę analizę, vaizdų ženklinimą (watermarking). Taip pat DCT naudojama spektriniuose metoduose dalinių diferencinių lygčių sprendimui ir signalų filtravimui. DCT pasižymi didele „energijos suspaudimo“ galimybe: dauguma signalo informacijos sukoncentruojama keliuose žemo dažnio koeficientuose [6].

## *2.4 Vaizdo suspaudimo principai*

Prieš nagrinėjant DCT bei IDCT algoritmus, būtina susipažinti su vaizdo kodavimo koncepcija ir principais. Apibrėžti jų panaudojimą bei veikimą, aprašyti supratimui reikalingas sąvokas.

### 2.4.1 JPEG

Vienas dažniausiai naudojamų vaizdo suspaudimo algoritmų, vadinamas JPEG. Šis pavadinimas kilęs iš šio standarto kūrėjų (JPEG - Joint Photographic Experts Group). Šis algoritmas dažnai naudojamas prarandančiam kokybę (lossy), arba neprarandančiam kokybės (lossless) skaitmeninių vaizdų kodavimui. Jo suspaudimo laipsnis gali būti keičiamas, leidžiant vartotojui pasirinkti tarp suspaustų duomenų dydžio ir vaizdo kokybės. Šis formatas yra dažniausiai nuotraukų saugojimui fotoaparatuose bei kompiuteriuose ir siuntimui internete naudojamas vaizdo suspaudimo būdas. JPEG standartas aprašo kodeką (angl. codec – **co**der – **dec**oder), t.y.: metodą, leidžiantį užkoduoti vaizdą į duomenų srautą taip, kad šio koduoto srauto užimamas duomenų kiekis būtų mažesnis nei koduojamo vaizdo, ir metodą jam atstatyti (dekoduoti) peržiūrai ar redagavimui. JPEG kodekas suspaudžia vaizdinę informaciją iki 10 kartų, beveik neprarasdamas matomos kokybės [7].

Šio kodeko darbo principas – diskrečioji kosinusinė transformacija (DCT) ir jos įgalintas suspaudimas, todėl DCT bei IDCT algoritmus nagrinėsime kaip sudedamąsias kodeko dalis [8].

### 2.4.2 Kodavimas

#### *2.4.2.1 Spalvų erdvės keitimas*

JPEG algoritmas pirmiausia pakeičia spalvų erdvę iš RGB (angl. Red, Green, Blue) į kitą – YCrCb. Ši spalvų erdvė turi 3 komponentes. Y komponentė aprašo vaizdo taško ryškumą, o Cr ir Cb – spalvingumą, atitinkamai padalintą į raudoną ir mėlyną komponentes. Toks spalvų kodavimas naudojamas skaitmeniniuose televizoriuose, DVD filmuose ir kituose PAL sistemą naudojančiuose prietaisuose. YCrCb spalvų erdvė leidžia pasiekti didesnį suspaudimo laipsnį, ženkliai nesumažindama regimos vaizdo kokybės. Didesnis laipsnis pasiekiamas dėl ryškumo komponenės išskyrimo. Ši informacija yra svarbesnė regimai vaizdo kokybei, todėl atskiriama į skirtingą kanalą [9].

#### *2.4.2.2 Atrinkimas*

Dėl nevienodo akyje esančių šviesai ir spalvai jautrių receptorių kiekio, žmogus geriau mato ryškumą (Y komponentę), nei spalvą ar atspalvį (Cr ir Cb komponentes). Panaudojant šią savybę, kodekai gali padidinti vaizdo suspaudimą. Po spalvų erdvės keitimo galima atlikti kitą kodavimo žingsnį – sumažinti Cb ir Cr komponenčių raišką. Šis metodas dar vadinama atrinkimu (angl. Downsampling). Raiška gali būti nemažinama (4:4:4 atrinkimas), sumažinama 2 kartus horizontaliaja kryptimi (4:2:2 atrinkimas), arba sumažinama 2 kartus horizontaliaja bei vertikaliaja kryptimi (4:2:0 atrinkimas). Toliau atskiros komponentės apdorojamos atskirai tais pačiais kodavimo metodais.

#### *2.4.2.3 Blokų išskyrimas*

Po atrinkimo, kiekvienas spalvos kanalas suskaidomas į 8x8 vaizdo taškų dydžio blokus. Jei koduojami duomenys nesidaliną į sveikąjį blokų skaičių, kodavimo įrenginys privalo užpildyti kraštinius blokus iki pilnų. Tai padaroma su viena iš netikros informacijos formų: blokų kraštai užpildomi viena spalva (pvz.: juoda), tačiau toks užpildymas sukuria vienspalvį rėmelį atkoduotame vaizde. Dažniau taikomas užpildymo būdas – kraštinių bloko reikšmių kartojimas. Šis metodas sumažiną „rėmelio“ efektą atkoduotame vaizde. Gali būti taikomi ir sudėtingesni kraštinių blokų užpildymo metodai.

#### *2.4.2.4 Diskrečioji kosinusinė transformacija*

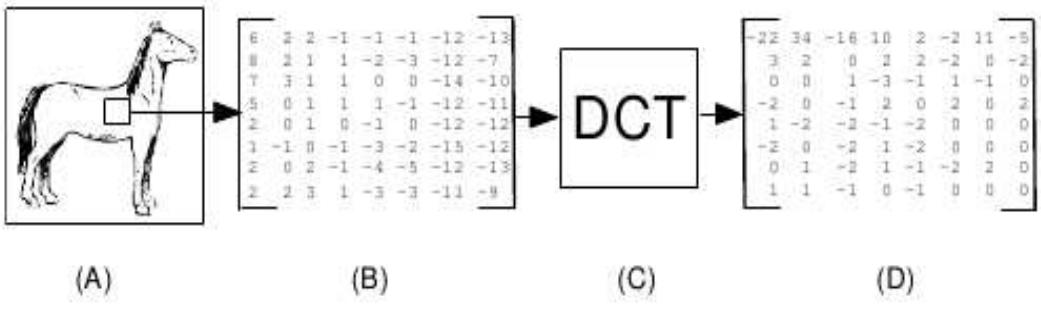
Diskrečioji kosinusinė transformacija yra JPEG spaudimo algoritmo pagrindas. JPEG algoritmui ši transformacija sukuria koeficientus, kurie tinka suspaudimui naudojant

kvantavimą. Kodavimas DCT yra grįžtamas, t.y.: kodavimo rezultatui pritaikius IDCT transformaciją, gaunamas identiškas pradiniam rezultatas. Tai leidžia jį naudoti ir prarandančioms ir neprarandančioms kokybės spaudimo technologijoms. DCT yra ypatinga Furje transformacijos atmaina. Bendru atveju, Furje transformacija teoriškai gali atvaizduoti įėjimo signalą sinuso ir kosinuso funkcijų koeficientais. Kosinusinė transformacija yra šios transformacijos atmaina, kur sinusinė dedamoji yra nenaudojama. JPEG kodavime naudojamas dvimatis DCT algoritmas. Tai reiškia, kad dydžių įvertinimas vyksta du kartus. DCT naudojama dvimatėje erdvėje, įėjime duodant 8x8 vaizdo taškų bloką. Kaip rašyta anksčiau, matematiškai DCT transformuoja erdvę aprašančius koeficientus į dažnį aprašančius koeficientus. Po transformacijos gaunamas 8x8 dažninių koeficientų blokas. Šie koeficientai – bazinės kosinuso funkcijos. Pirmas koeficientas matricos 0 eilutėje ir 0 stulpelyje vadinamas DC koeficientu – vidutine viso bloko ryškumo, arba spalvos verte (priklausomai nuo to, kurie – šviesumo ar spalvos blokai koduojami). Likę 63 elementai vadinami AC koeficientais, ir atspindi šviesumus ar spalvas, iš kurių susideda likęs blokas [6] [8].

JPEG vaizdo kodavimo algoritme paveikslėlis skaidomas į 8x8 duomenų matricas. Žemo dažnio koeficientai yra viršutiniame kairiajame išėjimo matricos kampe, o aukšto dažnio koeficientai – apačioje dešiniajame kampe.

Aukšto dažnio koeficientai turi mažai vaizdinės informacijos, žmogaus rega jiems nėra itin jautri. Kai DCT naudojama spaudimo tikslams, kvantavimo algoritmu stengiamasi aukšto dažnio koeficientus paversti nuliais, o svarbius – žemo dažnio elementus, palikti nepaliestus.

2.3 pav. pavaizduotas DCT panaudojimas, taikant algoritmą 8x8 vaizdo taškų paveikslėlio blokui. Vaizduojamas duomenų išskyrimas iš paveikslėlio, DCT algoritmo pritaikymas ir tolimesni suspaudimo veiksmai, kurie galimi tik naudojant iš DCT gautą informaciją [6][8][10].



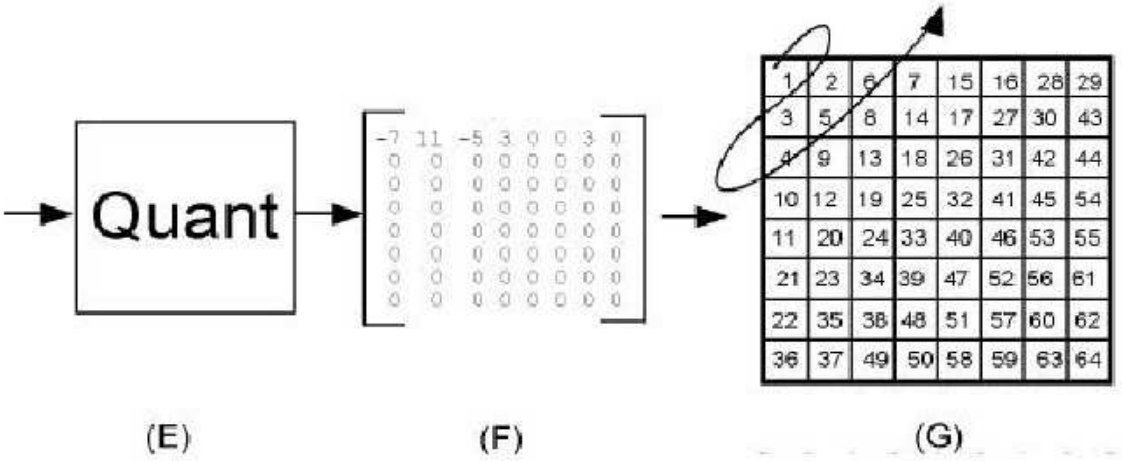

*2.3 pav. DCT transformacija[11]*

2.3 pav. pavaizduota DCT transformacija vienam paveikslėlio blokui. Iš paveikslėlio (A) paimamas 8x8 vaizdo taškų blokas (B) ir apdorojamas DCT algoritmu (C). Gautas rezultatas

(D) vaizduoja DCT algoritmo veikimo principą.

#### 2.4.2.5 Kvantavimas ir zigzag skenavimas

Po apdorojimo DCT algoritmu, žemo dažnio koeficientai, turintys daugiau vaizdinės informacijos, sutelkiami viršutinėje kairėje matricos srityje (pradžioje), o aukštesnio dažnio koeficientai, turintys mažiau vaizdinės informacijos, užima likusią matricos dalį. Šie koeficientai apdorojami kvantavimu.



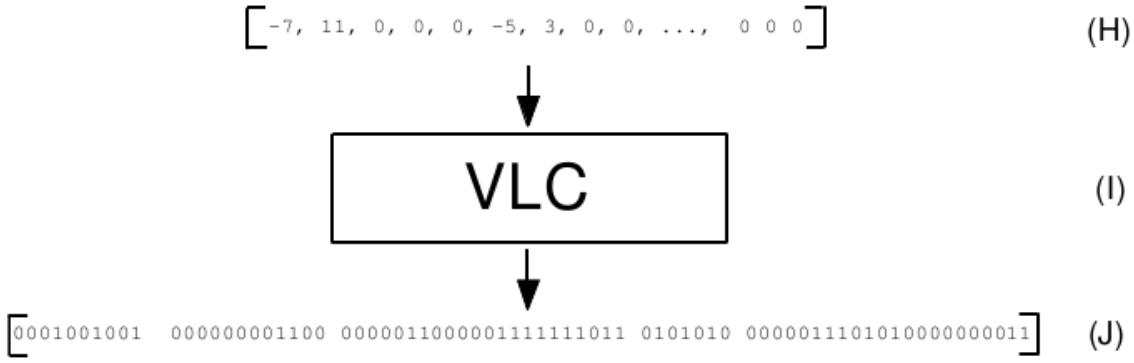

*2.4 pav. Kvantavimas ir zigzag skenavimas[11]*

2.4 pav. pavaizduotas kvantavimo procesas ir po jo sekantis zigzago skenavimas. Priklausomai nuo norimos kokybės, kvantavimo algoritme įvedamas QUANT dydis. Šis dydis nurodo, kokią dalį aukšto dažnio koeficientų kvantavimo algoritmas paverčia nuliais. Kuo šis skaičius didesnis, tuo daugiau koeficientų paverčiama nuliais. Vaizdo taškų reikšmės, pakeistos nuliais dingsta negrįžtamai. Ši informacija prarandama. Kvantavimo algoritmas verčia koeficientus nuliais, atlikdamas dalybos operaciją be liekanos. Kvantuojamoje matricoje lieka tik keli nenuliniai elementai (priklausomai nuo pasirinktos kokybės), esantys viršutiniame kairiajame kampe. Daugumą mažiau svarbių elementų prilyginami nuliui. Dėl DCT pasiekto duomenų išdėstymo, šie duomenys gali būti išsaugojami kaip vektorius naudojant zigzag skenavimo algoritmą (G). Rodyklė paveikslėlyje rodo skanavimo eigą pirmiesiems 6 matricos elementams. Tokia skanavimo eiga stengiamasi išdėstyti matricos koeficientus į vektorių taip, kad būtų gautos kuo ilgesnės nulių sekos [10].

#### 2.4.2.6 Kintamo eilutės ilgio kodavimas

Kintamo eilutės ilgio kodavimas (Variable length encoding) yra labai paprastas suspaudimo metodas, kuriame pasikartojantys elementai išsaugomi kaip vienas elementas ir

pasikartojimų skaičius. Kadangi po zigzag skenavimo gaunamos ilgos nulinių elementų eilės, šis algoritmas ženkliai sumažina informacijos kiekį.




*2.5 pav. Eilutės ilgio kodavimo algoritmo veikimas[10]*

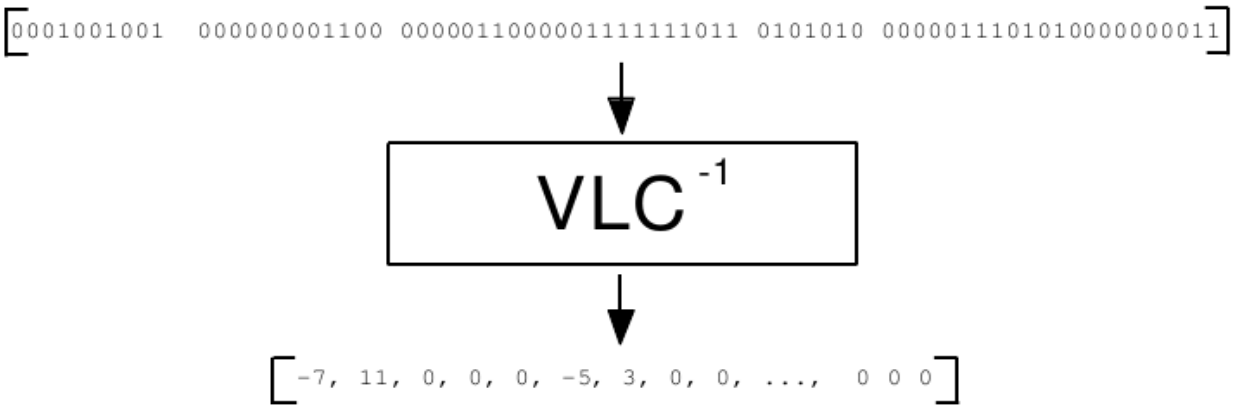
2.5 pav. pavaizduotas VLC algoritmo veikimas. Vektorius su ilgomis nulių sekomis (H) apdorojamas VLC algoritmu (I), kuriame ilgos sekos suspaudžiamos į kintamo ilgio kodus (J). Pažymėtina, kad atvaizduoti visam 8x8 duomenų blokui po algoritmo užtenka apie 73 bitų. Nespaudžiant duomenų, tokiam blokui reikėtų 64x8 = 512 bitų. DCT algoritmo įgalintas suspaudimas sutaupo virš 85% talpos [10].

### 2.4.3 Dekodavimas

Norint peržiūrėti vaizdinę informaciją, saugomą JPEG formatu, reikia atlikti dekodavimą. JPEG dekoderis veikia kartodamas kodavimo žingsnius atvirkštine tvarka.

#### *2.4.3.1 Kintamo eilutės ilgio dekodavimas*

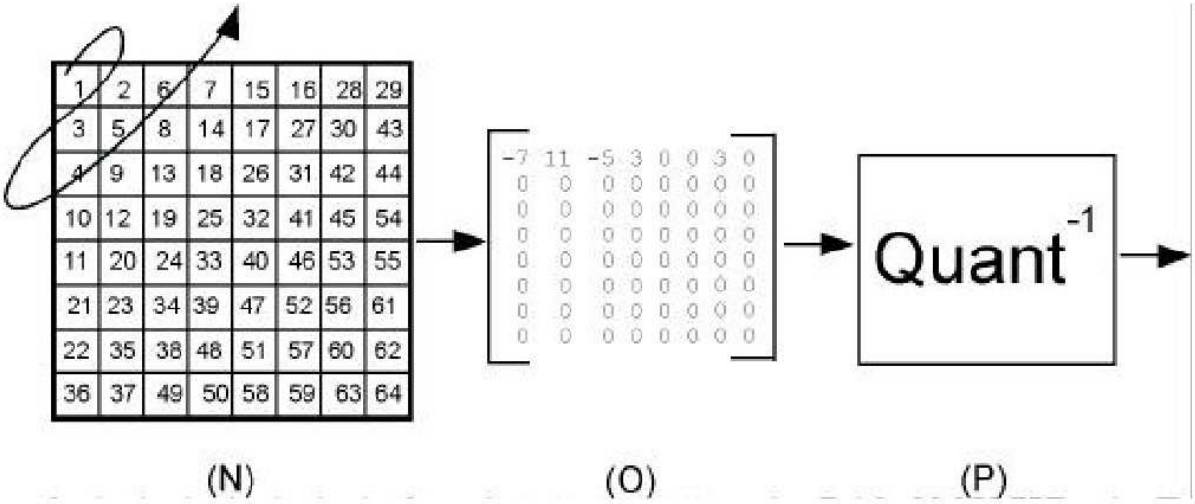
Eilutės ilgio dekodavimas vyksta priešingai kodavimui – iš elemento vertės ir pasikartojimų skaičiaus atstatoma eilutė. Šio algoritmo veikimas pavaizduotas 2.6 pav.




*2.6 pav. Atvirkštinis kintamo eilutės ilgio kodavimas[11]*

#### 2.4.3.2 Atvirkštinis kvantavimas ir deskenavimas

Šie algoritmai taip pat atliekami atvirkščia tvarka. Veikimas pavaizduotas 2.7 pav.

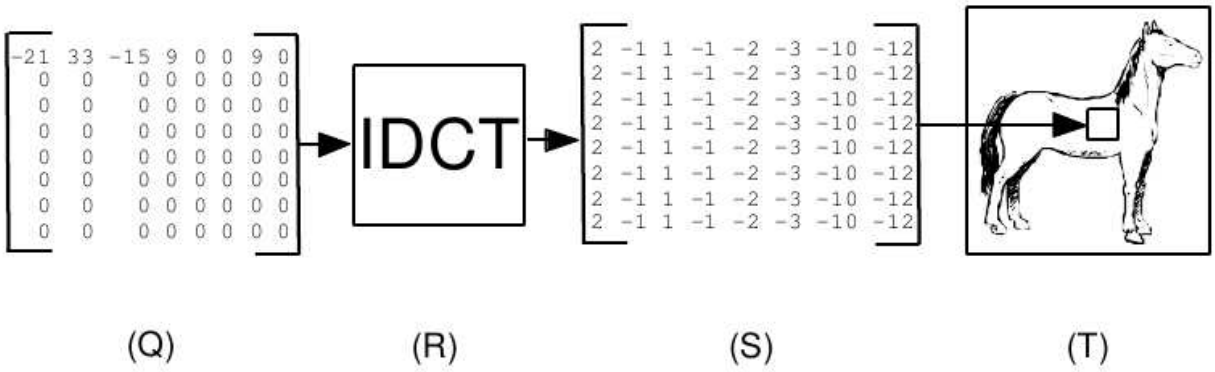



*2.7 pav. Atvirkštinis kvantavimas ir deskenavimas[11]*

Deskenavimo procesas (N) sugrąžina elementus į 8x8 vaizdo taškų dydžio matricą (O). Tuomet matrica dekvantuojama (P), padauginant kiekvieną jos elementą iš kvantavimo reikšmės QUANT. 2.7 pav. matome, kad buvę nenuliniai elementai po kvantavimo atsistatė, tačiau mažiau svarbūs aukšto dažnio elementai, kurie buvo paversti nuliais dingo negrįžtamai.

#### 2.4.3.3 Atvirkštinė diskrečioji kosinusinė transformacija (IDCT)

2.8 pav. pavaizduota IDCT transformacija vienam paveikslėlio blokui. 8x8 vaizdo taškų blokas (Q) paimamas iš kodavimo algoritmo ir apdorojamas IDCT algoritmu (R). Gautas rezultatas (S) - atstatytas vaizdas, tinkamas peržiūrai ar redagavimui.



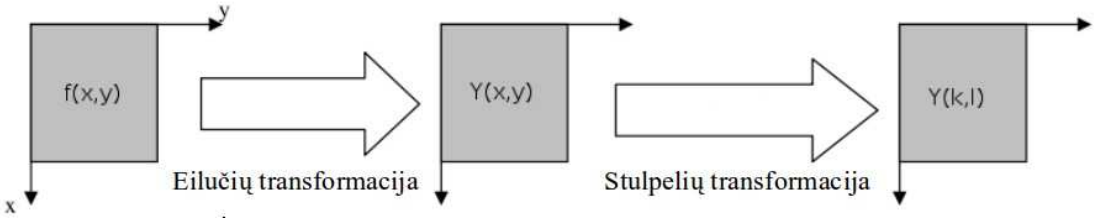

*2.8 pav. IDCT transformacija[11]*

## *2.5 DCT architektūros*

Tiesioginis formulių (1) ir (2) taikymas reikalauja labai daug matematinių skaičiavimų. Tam būtų sunaudojama begalė resursų ir tokios sistemos greitis dėl sudėtingumo būtų mažas. DCT algoritmo efektyvumo matas yra daugybos operacijų skaičius. Tiesioginė DCT realizacija naudoja $N^4$ daugybos operacijų NxN įvedimo blokui, todėl yra retai naudojama. Praktiniam taikyme dažniausiai naudojami vadinamieji „greitieji“ DCT bei IDCT algoritmai. Šie algoritmai pagreitina DCT, sumažindami atliekamų skaičiavimų kiekį. Daugybų skaičius tokiuose algoritmuose sumažėja iki $2*N*\log_2 N$ ar net mažiau. Dvimatė (2-D) DCT ir IDCT skaičiavimui naudojami du pagrindiniai metodai: tai eilučių/stulpelių metodas ir tiesioginiai 2-D algoritmai. Abi šios grupės naudoja vienmatę (1-D) DCT ar IDCT kaip pagrindą.

### 2.5.1 Eilučių/stulpelių metodas

2.2.3 skyriuje aprašyta DCT transformacijos atskiriamumo savybė. Ši savybė reiškia, kad dvimatę DCT transformaciją galima išskaidyti į dvi vienmates DCT transformacijas. Tai leidžia taikyti vienmatį algoritmą matricos eilutėms, tuomet transponuoti rezultatą, vėl pritaikyti algoritmą ir rezultatą vėl transponuoti. Tokiomis operacijomis gaunama dvimatė DCT ar IDCT transformacija. Šis skaičiavimo būdas vadinamas eilučių/stulpelių metodu. Metodas iliustruotas 2.9 pav. Nors tokie metodai matematiškai paprastesni, tačiau jiems įgyvendinti reikalingas duomenų perrašymas atmintyje, o tam gaištamas skaičiavimo laikas.




*2.9 pav. Eilučių/stulpelių metodas*

Šis metodas yra pats populiariausias DCT bei IDCT algoritmų įgyvendinimo būdas skaitmeninėje technikoje. Jo rezultatas – paprastas, nuoseklus algoritmas, nors matematiškai kiek mažiau efektyvus nei tiesioginis dvimatis algoritmas.

### 2.5.2 Tiesioginis 2-D skaičiavimas

Tiesioginiam dvimatės DCT ar IDCT algoritmo taikymui tereikia apie pusės matematinių skaičiavimų, lyginant su eilučių/stulpelių algoritmais, tačiau šie algoritmai privalo turėti sudėtingesnę valdymo logiką bei nenuoseklų duomenų kelią. Kadangi nenaudojamas transponavimas, sutaupoma laiko. Tarpinės transponavimo atminties sutaupyti nepavyksta, nes atminties registrus, naudojamus eilučių/stulpelių algoritme, šiuose algoritmuose atsveria registrai, reikalingi įėjimų pertvarkymui ir laikinų duomenų saugojimui [12].

Tiesioginiams dvimačiams algoritmams įgyvendinti dažniausiai naudojami trys skirtingi metodai: matricų, vektorių šaknies ir laiko rekursinis metodai.

Matricų metodas yra vienas efektyviausių dvimatės DCT ir IDCT taikymų. Pavyzdžiui Feigo ir Winogrado algoritmui reikalingos tik 94 daugybos operacijos dvimatei 8x8 vaizdo taškų sekai transformuoti. Daugybos operacijų skaičius sumažinamas net iki 54, jei atliekama mažoji DCT algoritmo versija [13].Šie algoritmai naudoja sudėtingus veiksmus su matricomis, todėl juos sunku realizuoti aparatiškai.

Populiarus aparatiniam realizavimui yra vektorių šaknies metodas. Šis metodas sukuria greitą dvimatės DCT/IDCT algoritmą, lengviau realizuojamą aparatūroje [14-19].

Vektoriaus-šaknies dvimatė DCT/IDCT veikimas panašus į šaknies iš 2 greitąją Furje transformaciją. Remiantis vaizdo taškų indeksais, 8x8 įvedimo matrica padalinama į keturis duomenų blokus: lyginiai/lyginiai indeksai, lyginiai/nelyginiai indeksai, nelyginiai/lyginiai indeksai ir lyginiai/lyginiai indeksai. Ši dekompoziciją sukuria formą, kurią galima apskaičiuoti tik su aštuoniomis vienmatėmis DCT/IDCT operacijomis ir šiek tiek po jų sekančių apdorojimo veiksmų. Tai tik pusė vienmačių DCT/IDCT, lyginant su eilučių/stulpelių metodu. Taipogi yra pastebėta, kad tiesioginiam dvimačiam DCT/IDCT algoritmas reikalingas mažesnis duomenų plotis įėjime ir išėjime, nes duomenys per fiksuoto kablelio daugintuvus turi pereiti tik vieną kartą, kai tuo tarpu eilučių/stulpelių algoritme duomenys privalo pereiti per daugintuvus du kartus – vieną kartą - eilučių ir vieną kartą - stulpelių apdorojime [15].

Deja vektoriaus-šaknies algoritmai smarkiai nukenčia nuo nenuoseklumo ir didelio kiekio išankstinio ir galutinio duomenų apdorojimo (pre- and post-processing) [15].

Laiko-rekursiniams DCT/IDCT algoritmams naudojama filtro su begalinio ilgio atsaku į vienetinį impulsą (Infinite Impulse Response - IIR) tipo architektūra. Ši architektūra pasižymi

paprastu įėjimų ir išėjimų valdymu, yra lygiagreti, komponentinio dizaino ir nuoseklaus išdėstymo. Laiko-rekursiniam algoritmui reikia tik aštuonių vienmačių DCT operacijų. Neigiama šių algoritmų puse yra didelis aritmetinių operacijų naudojimas, kurios privalo veikti beveik be sustojimo, kiekvienu taktinio dažnio impulsu, todėl dėl sudėtingumo yra lėtas ir suvartojantis daug galios [20] [21].

### 2.5.3 Apytiksliai algoritmai

Tiksliai skaičiuojant DCT algoritmą, jis yra grįžtamas ir visiškai neprarandantis kokybės. Apytiksliai algoritmai sumažina DCT ir IDCT sudėtingumą, paaukodami skaičiavimo tikslumą. Taip į rezultatą įnešamas papildomas kiekis klaidų, kurios, sudėjus jas su kvantavimo ir fiksuoto kablelio skaičiavimo įneštomis klaidomis praranda nemažai vaizdinės informacijos. Apytiksliai algoritmai sumažina daugybos operacijų skaičių, supaprastina duomenų kelią ir atlieka DCT bei IDCT skaičiavimą daug greičiau nei anksčiau aprašytieji algoritmai.

Kintamųjų supaprastinimas, aprašomas [22] supaprastina pačią DCT. Šių algoritmų autoriai naudoja QUANT parametrą nustatyti, kaip turėtų būti atliekama DCT. Jei kvantavimo reikšmė (QUANT) didelė, t.y. daug elementų paverčiama nuliais, prarandama daug informacijos ir rezultatas gaunamas palyginti prastos kokybės, nustatyta, kad DCT įvedamos klaidos turi labai mažai pastebimos reikšmės. Tuo tarpu skaičiavimo sudėtingumą šiuo metodu galima sumažinti 22% - 27%. Girodas ir Stuhlmulleris apskaičiuoja dvimatę DCT naudodami tik bloko DC komponentą ir pirmuosius du AC komponentus [23]. Šiuos elementus jie aproksimuoja tokiu būdu:

$$X_{0,0} = k_{DC}(c_0 + c_7 + r_0 + r_7) \quad (9)$$

$$X_{0,1} = k_{AC}(r_0 - r_1) \quad (10)$$

$$X_{1,0} = k_{AC}(c_0 - c_7) \quad (11)$$

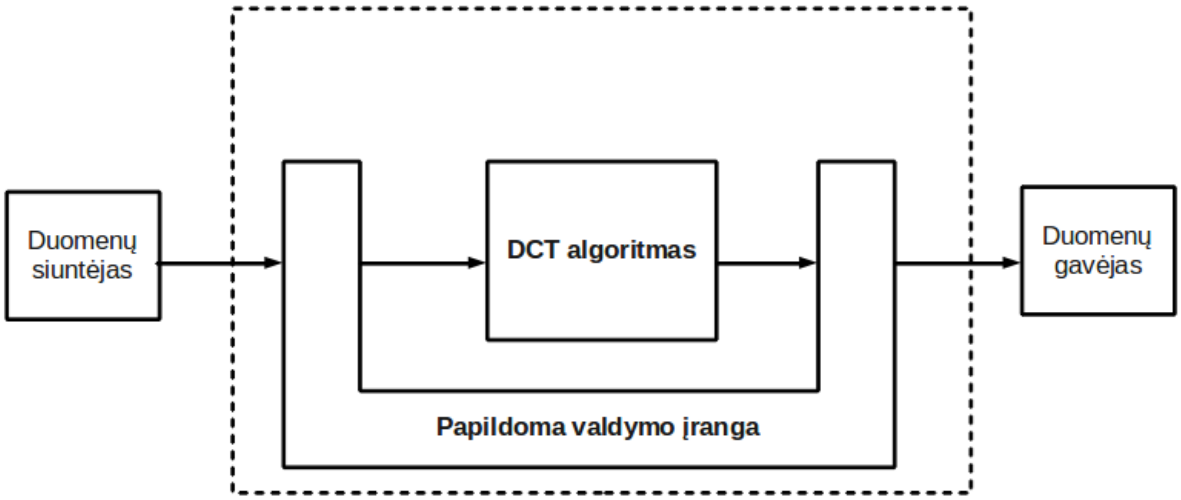
(9) - (11) formulės Girodo ir Shtuhlmullerio aproksimacija apytiksliam DCT skaičiavimui čia:

- $X$ – DCT koeficientas,
- $k$ – dydžio konstanta,
- $c_x$ – stulpelio x suma,
- $r_x$ – eilutės x suma.

Apskaičiuoti dvimatę DCT 8x8 dydžio blokui šiam metodui reikia tik 33 sumos ir 3 daugybos operacijų. Jei QUANT parametras didelis, autoriai teigia, kad įmanoma sutaupyti apie 50% skaičiavimų, papildomai nepabloginant kokybės [23].

## 2.6 Pakartotinis komponentų panaudojimas

Siekiant ištirti DCT bei IDCT algoritmus, reikia realizuoti papildomą aparatūrą duomenų keliui bei atminčiai valdyti. Pakartotinio panaudojimo koncepcija yra fundamentali ir plačiai naudojama programų inžinerijoje. Aparatūroje ši technologija neišnaudojama [24]. Mūsų tikslas – pagaminti sistemą pakartotinio panaudojimo komponentų pagrindu. Komponentinis metodas panaudoja keletą objektinio programavimo savybių, tokių kaip duomenų uždarumas (enkapsuliacija) ir matomumas. Projektuojamos aparatūros struktūra susideda iš dviejų dalių: sąsajos, leidžiančios ją sujungti su kitais komponentais ir vidinės dalies, atliekančios duomenų kelio valdymą ir duomenų saugojimą. Pastaroji dalis aprašo įrangos funkcionalumą ir elgseną. Suprojektavus šią įrangą, jos vidus nekeičiamas ir jam gali būti taikomas „juodos dėžės“ principas, t. y.: naudojimas, nematant ir nekeičiant elgsenos. Visų tiriamų algoritmų komponentai turi būti priderinami prie papildomos įrangos sąsajos (2.10 pav.).

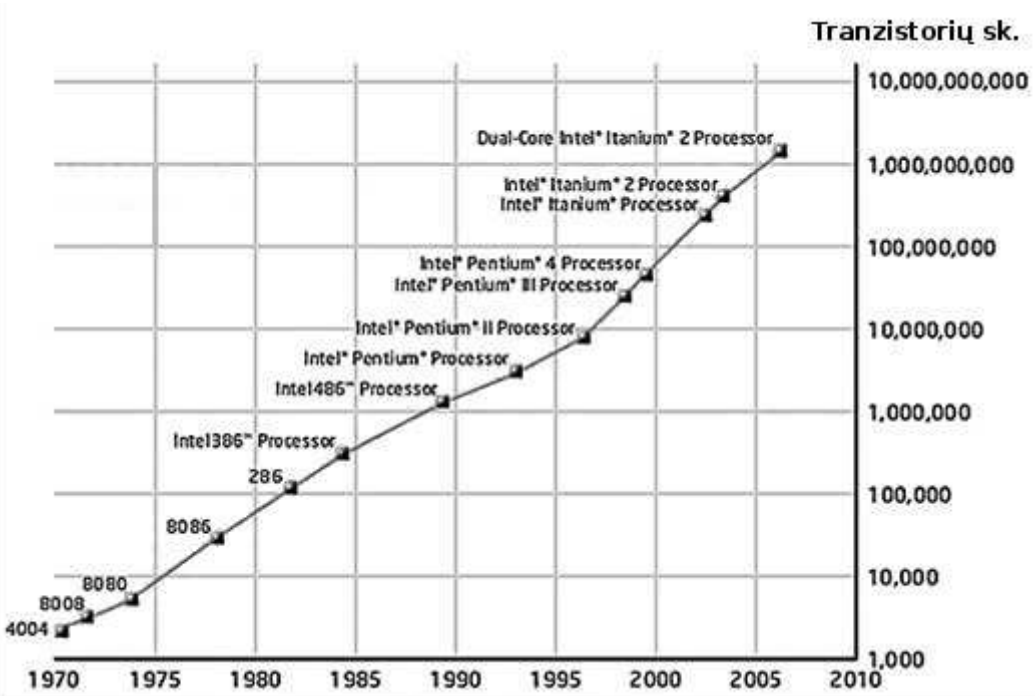



*2.10 pav. Pakartotinis valdymo įrangos panaudojimas*

Tokia papildomos įrangos realizacija leidžia pakartotinį šios įrangos panaudojimą visiems tiriamiems DCT bei IDCT algoritmams. Tai sumažina projektavimo laiką bei leidžia gauti tyrimų rezultatus, priklausančius tik nuo algoritmų veikimo ir realizacijos skirtumų, kadangi jiems naudojama ta pati papildoma įranga. Įrangos projektavime ji atskiriama nuo konkrečios aparatinės įrangos ar komponentinių bibliotekų, todėl ją galima pakartotinai panaudoti įvairių algoritmų tyrimui bei kaip sudedamąją dalį kodavimo ar dekodavimo sistemose ar signalų filtruose. Toks projektavimas įtvirtina fundamentalią pakartotinio aparatūros panaudojimo koncepciją.

## 2.7 Sistemų aprašymo kalbos

Integrinių schemų matmenys mažėja nuo pat jų sukūrimo, tai leidžia talpinti vis didesnes ir sudėtingesnes schemas vienoje vienlustėje sistemoje. Mūro dėsnis (Moore's Law) teigia, kad tranzistorių skaičius vienlustėje sistemoje padvigubėja kas du metus (2.11 pav.) [25]. Tokių būdu didinamas tokių sistemų efektyvumas arba mažinama kaina.




*2.11 pav. Tranzistorių skaičiaus procesoriuje didėjimas*

Didėjant schemų integracijai ir sudėtingumui, atsiranda būtinybė greitai ir efektyviai aprašyti aparatines sistemas. Šiuo metu schemų aprašymui naudojamos trys pagrindinės kalbos: VHDL, Verilog ir SystemC. Šios kalbos turi IEEE standartus bei projektavimo, modeliavimo ir verifikavimo įrankius. Sukurti komerciniai įrankiai dažnai palaiko kelias kalbas.

VHDL (Very-high-speed integrated circuits Hardware Description Language) kalba buvo sukurta Jungtinių Amerikos Valstijų (JAV) gynybos departamento iniciatyva, siekiant sukurti paprastą ir aiškų būdą dokumentuoti ASIC. Kalba kurta kaip alternatyva dideliems ir sudėtingiems vartotojų vadovams. 1983 metais JAV Gynybos departamentas suformulavo reikalavimus standartinei aparatūros aprašymo kalbai (VHSIC Hardware Description Language – VHDL), visuotinai priimtas kalbos standartas buvo patvirtintas 1987 metais. Šis standartas vadinasi “IEEE 1076/A VHDL Language Reference Manual” - VHDL kalbos žinynas. 1993 metais šis kalbos standartas buvo šiek tiek papildytas [26].

Pagrindinis šios kalbos privalumas – galimybė aprašyti lygiagrečius procesus. VHDL – tai duomenų srautų kalba. Ji skiriasi nuo procedūrinių kalbų, tokių kaip C, BASIC ar asembleris, kurios vykdomos nuosekliai, instrukcija po instrukcijos.

Verilog kalba buvo sukurta Gateway kompanijos 1983 metais ir naudota tik kompanijos viduje. Kai šią kompaniją nusipirko Cadence Design Systems, buvo nuspręstą kalbą paviešinti ir standartizuoti. 1995 metais kalba buvo standartizuota IEEE 1364-1995 standartu. Kalbos sintaksė labai panaši į C programavimo kalbos. Tai leidžia greitai, trumpai ir aiškiai aprašyti vienlustes sistemas net žemu abstrakcijos lygmeniu.

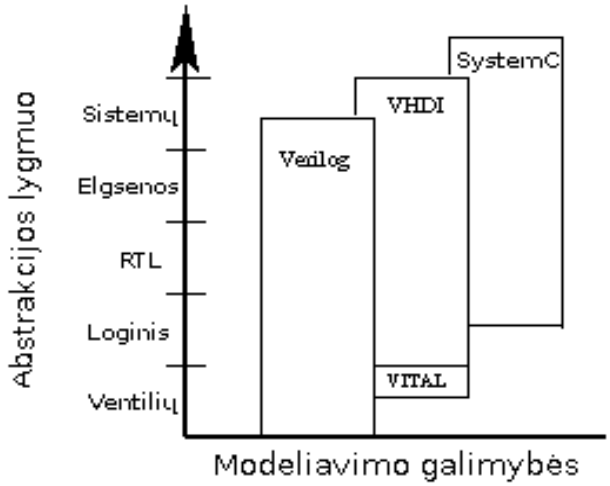
Nepriklausoma, pelno nesiekianti organizacija Open SystemC Initiative 1999 metais paskelbė apie kuriamą C++ kalbos išplėtimą sistemoms aprašyti. Šį kalbos išplėtimą sudarė C++ klasės ir išankstinės doroklės (preprocesoriaus) direktyvos, kurios suteikė galimybę atlikti įvykiais pagrįstą modeliavimą. Tai leido projektuotojui aprašyti ir modeliuoti lygiagrečius procesus naudojant C++ sintaksę. Ši nauja kalba pavadinta SystemC.

SystemC modeliai informacijos saugojimui ir komunikavimui tarp procesų gali naudoti visus C++ duomenų tipus, taip pat tipus aprašytus SystemC bibliotekoje, bei vartotojo sukurtus tipus. 2005 metais šis C++ išplėtimas buvo paskelbtas kaip IEEE 1666 standartas.

Nors SystemC savo veikimu imituoja aparatūros aprašymo kalbas, tokias kaip VHDL ar Verilog, tačiau ji labiau skirta sistemų lygmens modeliavimui.

### 2.7.1 Abstrakcijos lygiai

Naudojant sistemų aprašymo kalbas, projektą galima aprašyti skirtingais būdais. Pagrindinis skirtumas tarp aprašymo būdų yra kaip tiksliai aprašoma fizinė aparatūra. Šie būdai vadinami abstrakcijos lygmenimis. Kuo aukštesnis abstrakcijos lygmuo tuo aprašas suprantamesnis žmogui, tačiau tuo sunkiau jį modeliuoti ar transformuoti į fizinį aparatūros aprašą. Sistemų aprašymo kalbų apimami lygmenys šiek tiek skiriasi. Šis skirtumas pavaizduotas 2.12 pav.




*2.12 pav. Abstrakcijos lygmens padengimas*

Ta pačią vienlustę sistemą galima aprašyti skirtingais abstrakcijos lygmenimis. Šie lygmenys yra tokie:

- Ventilių lygmuo. Pats žemiausias integrinių schemų aprašymo lygmuo. Aprašymui naudojamos nuo konkrečios gamybinės technologijos priklausomi elementai. Aprašomi elementai atlieka pačias primityviausias logines funkcijas, tokias kaip AND, OR, NOT.
- Loginis abstrakcijos lygmuo panašus į ventilių lygmenį. Pagrindinis skirtumas yra tas, jog aprašymas nėra priklausomas nuo konkrečios gamybos technologijos. Aprašyme naudojamos vidines kalbos funkcijos konkrečiai loginiai operacijai atlikti.
- RTL (register transfer level) lygmuo. Integrinės schemos veikimas aprašomas nurodant, kokius registrus turi schema ir duomenų perdavimo kelius tarp jų. Šis tarpinis lygmuo, tarp elgsenos ir loginių aprašų, leidžia išvengti sunkaus ir sudėtingo darbo projektuojant kombinacinę schemos logika, nes tuo gali pasirūpinti automatizuoti sintezės įrankiai.
- Elgsenos lygmuo. Šiame abstrakcijos lygmenyje aprašomas schemos veikimo algoritmas. Kuriant šio lygmens integrinės schemos aprašą, aprašomos operacijos, kurias schema turi atlikti konkrečiu laiko momentu.
- Sistema – tai grupė tarpusavyje sąveikaujančių elementų, veikiančių drauge, siekiant tam tikro tikslo. Bendru atveju sistema susideda iš keleto tarpusavyje sujungtų elementų. Kiekviena sistema turi ribas, atskiriančias sistemą nuo jos aplinkos. Tiksliai apibrėžus sistemos elementus, ryšius tarp jų ir sistemos ribas,

galima sukurti sistemos modelį, aprašytą kažkuria kalba. Tokio modelio tyrimas gali padėti nustatyti kaip sistema veiks tikrame pasaulyje.

Kaip matyti iš 2.12 pav. visos sistemų aprašymo kalbos padengia panašius abstrakcijos lygius ir tinka RTL lygmens modeliavimui. Šiam tyrimui pasirinkta VHDL kalba, nes ji populiari tiek akademiniams, tiek komercinėms sistemos aprašyti, turi galingus modeliavimo ir verifikavimo įrankius bei bibliotekas, leidžiančias dirbti su tekstiniais failais [26].

### 2.7.2 VHDL bibliotekos ir paketai

VHDL bibliotekose saugomi sukompiliuoti entity/architecture blokai, paketai ir konfigūracijos. Tai naudinga darant didelės apimties projektus, nes leidžia patogiai suskirstyti kodą. Be to šios bibliotekos gali palengvinti pakartotinį komponentų panaudojimą. Be vartotojo sukurtų bibliotekų, yra standartinės VHDL kalbos bibliotekos, kurias galima naudoti projektų aprašymui, modeliavimui ar sintezei. Bibliotekose saugomi paketai, palengvinantys matematinių operacijų atlikimą. Žemiau aprašyti dažniausiai naudojami paketai.

#### *2.7.2.1 std_logic_1164*

IEEE instituto sukurta biblioteka aprašanti praplėstus loginius tipus *std_logic*, *std_ulogic*, *std_ulogic_vector*, *std_logic_vector*, bei galimas logines operacijas su jais. Šie tipai turi devynias galimas reikšmes ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' ). Bibliotekos tikslas – sukurti standartinius duomenų tipus, kuriuos projektuotojas galėtų naudoti duomenų perdavimui [27].

#### *2.7.2.2 numeric_std*

Šis paketas aprašo tipus skaičių saugojimui ir operacijoms su jais, palaikomas sintezės įrankių. Pakete aprašomi du tipai skaičiams saugoti: *signed* – skaičiams su ženklu ir *unsigned* – skaičiams be ženklo. Šie tipai aprašyti kaip *std_logic* masyvai ir pats kairiausias bitas laikomas aukščiausiu. Pakete taip pat aprašomos loginės, aritmetinės, palyginimo ir konvertavimo operacijos, kurias galima atlikti su šiais duomenų tipais [27].

Šis paketas buvo pripažintas IEEE standartu ir pakeitė Synopsys sukurtus std_logic_arith ir std_logic_unsigned paketus. Nors šiuos Synopys paketus vis dar palaiko sintezės įrankiai, naujiems projektams rekomenduojama naudoti naująjį numeric_std paketą.

#### *2.7.2.3 textio*

Šis paketas aprašo standartinius duomenų tipus ir procedūras tekstinės informacijos įvedimui ir išvedimui VHDL kalboje. Pakete aprašomi tokie duomenų tipai: *text*, *line*, *side*, bei

procedūros *readline*, *writeline*, *read*, *write*. Kadangi šios procedūros nėra palaikomos sintezės įrankių, operacijos su tekstiniais failas dažniausiai naudojamos modelių testavimo tikslais.

Šis paketas svarbus ir dėl to, kad dvejetainių failų skaitymas nėra standartizuotas VHDL kalboje, todėl norint atlikti veiksmus su tokiais failais juos pirmiausia reikia išversti į tekstinį formatą (naudojant kokią nors išankstinę doroklę (preprocesorių)) ir įvesti naudojant textio paketą [27]

## 2.8 Analitinės dalies išvados

Vienas pagrindinių ir dažniausiai naudojamų algoritmų vaizdo apdorojime yra DCT transformacija. Tai taipogi viena sudėtingiausių algoritmo dalių, suvartojanti daugiausiai galios. Siūlomas galios ir greičio problemos sprendimas – DCT algoritmas vienlustėje sistemoje.

Šioje dalyje aprašomos pagrindinės DCT transformacijos matematinės savybės, ištirtos dvimačio DCT bei IDCT architektūros, jų pritaikomumas aparatinėje realizacijoje. Taip pat aptartas papildomos aparatūros kaip pakartotinio panaudojimo komponento realizacija. Apžvelgiamos sistemų aprašymo kalbos ir papildomi paketai. Projekto kūrimui pasirinkta VHDL kalba.

# 3 Projektinė dalis

## *3.1 Tyrimui pasirinkti DCT ir IDCT algoritmai*

Iš anksčiau minėtų 2-D transformacijų skaičiavimo metodų reikia pasirinkti tinkamus aparatinei realizacijai. Ir eilučių/stulpelių metodui ir tiesioginiam 2-D skaičiavimui reikalingas panašus aritmetinių veiksmų ir registrų skaičius, tačiau tiesioginis metodas turi keletą trūkumų – sudėtingesnį valdymą ir sudėtingesnę duomenų perdavimo schemą. Dėl nuoseklumo ir paprastumo, tyrimui pasirinkti eilučių/stulpelių algoritmai. 3.1 lentelėje pateikti populiariausiems vienmačiams algoritmams atlikti 8 skaičių sekai transformuoti reikalingų matematinių operacijų skaičiaus palyginimai.

*3.1 lentelė. Algoritmų palyginimas*

| | **Chen** | **Wang** | **Lee** | **Vetterli** | **Suehiro** | **Hou** | **Loeffler** | **BinDCT** |
|---|---|---|---|---|---|---|---|---|
| Daugybos | 16 | 13 | 12 | 12 | 12 | 12 | 11 | 0 |
| Sumos | 26 | 29 | 29 | 29 | 29 | 29 | 29 | 30 |

Darbui pasirinkti 3 algoritmai:

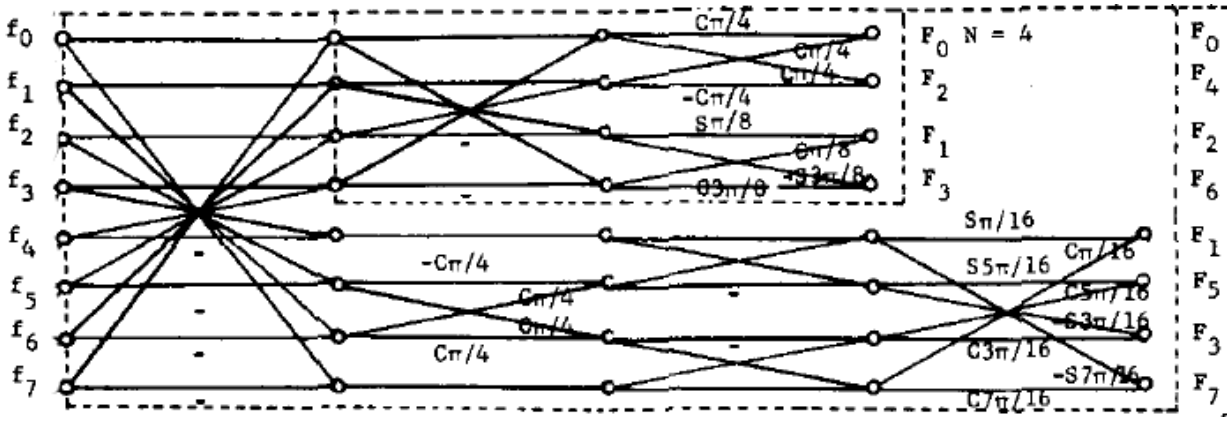
- W.Chen algoritmas – pirmas pasaulyje „greitasis“ DCT algoritmas [28].
- Loeffler algoritmas – teoriškai mažiausią daugybos operacijų skaičių turintis algoritmas [29]. Viena iš šio algoritmo versijų naudojama JPEG standarte.
- BinDCT – vienas naujausių DCT algoritmų, nenaudojantis daugybos operacijų [30].

### 3.1.1 W. Chen algoritmas

Šis algoritmas – pirmasis viešai publikuotas greito DCT ir IDCT skaičiavimo algoritmas [28]. Jis sukūrė pagrindus tokių algoritmų optimizavimui. Aštuonių skaičių ilgio vektoriui transformuoti algoritmas naudoja 16 daugybos ir 26 sudėties operacijas.

Šio algoritmo apibendrintoji forma gali atlikti N ilgio vektoriaus transformaciją, kai $N = 2^m$ , $m \geq 2$. N ilgio vektoriui algoritmas naudoja $(3N/2)(\log_2 N - 1) + 2$ sudėties ir $N*\log_2 N - 3N/2 + 4$ daugybos operacijų. Toks operacijų skaičius yra apie šešis kartus mažesnis, nei DCT skaičiavimas, naudojant greitąja Furje transformacija paremtus algoritmus [28].

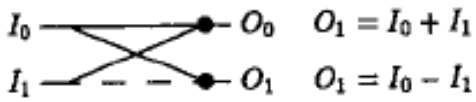
Signalų perdavimo grafas kai N = 4, 8 pavaizduotas 3.1 pav. Šitaip pavaizduotą algoritmą patogu realizuoti aparatinėmis ir programinėmis priemonėmis.




*3.1 pav. Chen algoritmo signalų perdavimo grafas [28].*

Iš signalų perėjimo grafo matyti, kad algoritmas suskaidytas į 4 etapus. Visi algoritmo etapai turi būti skaičiuojami nuosekliai, nes duomenys priklauso nuo prieš tai buvusio etapo. Skaičiavimai kiekvieno etapo viduje gali būti atliekami lygiagrečiai. 2 etape algoritmas skyla į dvi dalis: su lyginiais koeficientais ir su nelyginiais koeficientais. Lyginių koeficientų dalis tampa paprasta keturių taškų DCT.

Algoritmą sudaro „drugelio“ (butterfly) ir daugybos operacijos. Daugybos operacijų vietose parodytos konstantų, iš kurių dauginamas signalas, reikšmės. „Drugelio“ operacija pavaizduota 3.2 pav.



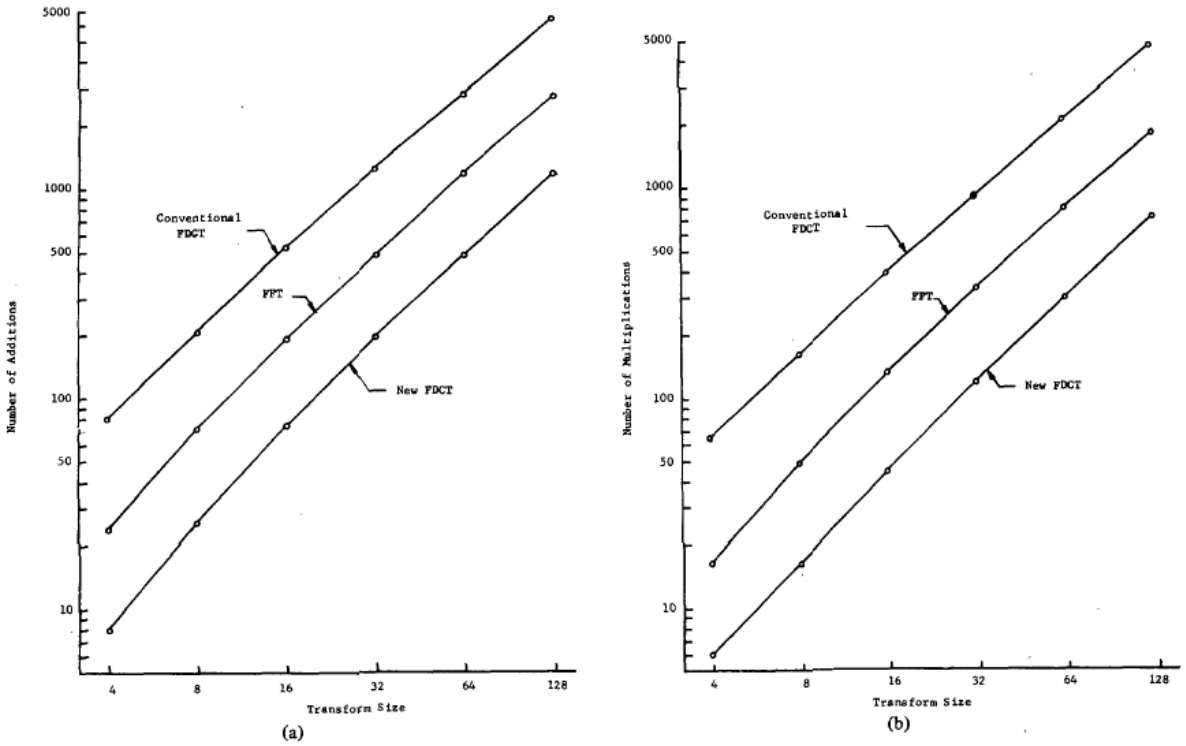

*3.2 pav. „Drugelio“ operacijos žymėjimas ir paaiškinimas*

Iš 3.1 pav. matyti, kad norint atlikti 8 taškų transformaciją atmintyje reikia saugoti 13 konstantų:

- `cos(π/4);`
- `-cos(π/4);`
- `sin(π/8);`
- `-sin(3π/8);`
- `sin(3π/8);`
- `sin(π/16);`
- `cos(π/16);`
- `sin(5π/16);`
- `cos(5π/16);`
- `-sin(3π/16);`
- `cos(3π/16);`
- `-sin(7π/16);`
- `cos(7π/16);`

3.1 pav. pavaizduotas signalų grafas yra dvikryptis [28]. Tai reiškia, jog atliekant veiksmus iš kairės į dešinę vektorius [f] transformuojamas į kosinuso funkcijos koeficientus [F]

(tiesioginė diskrečioji kosinusinė transformacija), atliekant veiksmus iš dešinės į kairę, kosinuso funkcijos koeficientai [F] transformuojami į pradinį vektorių [f] (atvirkštinė diskrečioji kosinusinė transformacija).




*3.3 pav. Matematinių operacijų kiekio palyginimas [28]. a – sudėčių, b – daugybų skaičius*

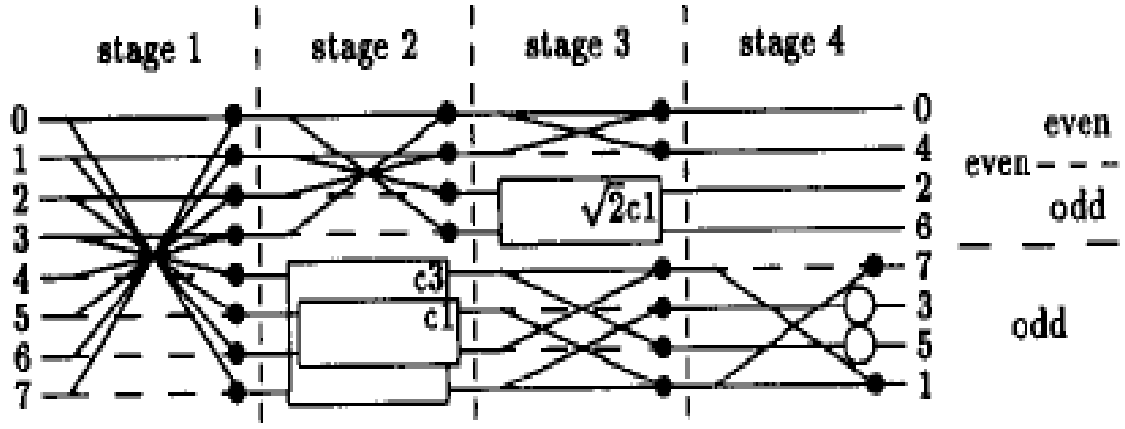
3.3 pav. pavaizduotas matematinių operacijų kiekio palyginimas tarp įprastinės DCT transformacijos, greitosios Furje transformacijos bei Chen algoritmo. Iš grafikų matyti, kad Chen pasiūlytas algoritmas naudoja mažiausiai matematinių operacijų.

### 3.1.2 Loeffler algoritmas

Šis eilučių/stulpelių algoritmas pasirinktas, nes naudoja 11 daugybos ir 29 sudėties operacijas, atlikti vienmatei DCT 8 skaičių ilgio sekai. Viena iš šio algoritmo versijų naudojama JPEG kodavimo standarte.

Algoritme daugybos operacijų skaičius sumažintas iki teoriškai mažiausio skaičiaus, nepadidinant sudėties operacijų skaičiaus [29]. Tai vienas naujausių „greitųjų“ DCT skaičiavimo algoritmų.

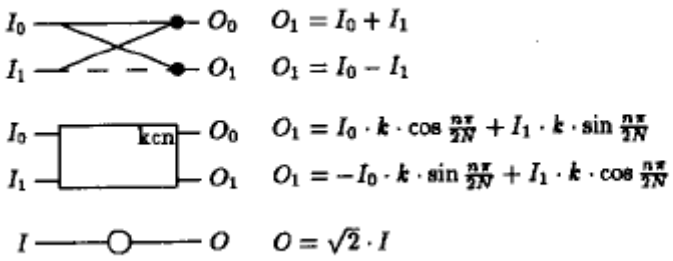
Algoritmas skirstomas į 4 etapus, kuriuose atliekamos atitinkamos operacijos. Šios operacijos pavaizduotos 3.4 pav.




*3.4 pav. Loeffler algoritmo signalų perdavimo grafas [29]*

Kaip ir W. Chen algoritme skaičiavimai kiekvieno etapo viduje gali būti atliekami lygiagrečiai. 2 etape algoritmas skyla į dvi dalis: su lyginiais koeficientais ir su nelyginiais koeficientais. Lyginių koeficientų dalis tampa keturių taškų DCT.

Algoritmą sudarantys skaičiavimo blokai paaiškinti 3.5 pav.




*3.5 pav. Loeffler algoritmą sudarantys blokai [29]*

Pirmajam blokui apskaičiuoti reikia dviejų sudėties operacijų. Antrasis blokas gali būti apskaičiuotas naudojant tris daugybos ir tris sudėties operacijas, vietoje keturių daugybos ir dviejų sudėties operacijų. Tam reikia pritaikyti (12) ir (13) formules.

$$y_0 = a \cdot x_0 + b \cdot x_1 = (b-a) \cdot x_1 + a \cdot (x_0 + x_1) \quad (12)$$

$$y_1 = -b \cdot x_0 + a \cdot x_1 = -(a+b) \cdot x_0 + a \cdot (x_0 + x_1) \quad (13)$$

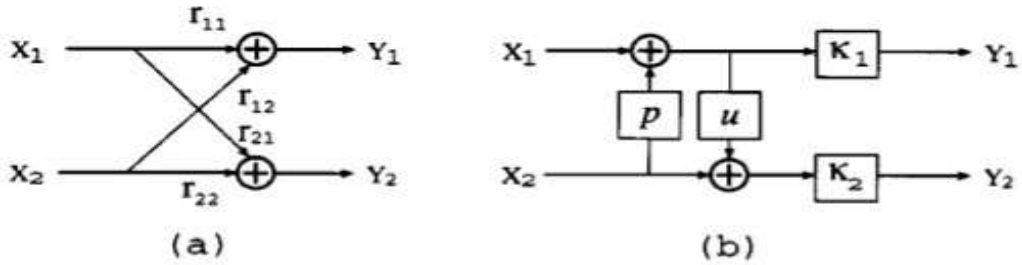
Kaip matosi iš 3.4 ir 3.5 pav., šiam algoritmui atmintyje reikia saugoti tik 7 konstantas:

- `cos(π/16);`
- `sin(π/16);`
- `cos(3π/16);`
- `sin(3π/16);`
- $\sqrt{2}$ `*cos(6π/16);`
- $\sqrt{2}$ `*sin(6π/16);`
- $\sqrt{2}$ .

Maksimalus signalo kelias šiame algoritme yra dvi daugybos ir keturios sudėties operacijos.

### 3.1.3 BinDCT algoritmas

BinDCT yra vienas naujausių greitosios DCT ir IDCT skaičiavimo algoritmų, kuris nenaudoja daugybos operacijų [30]. Chen bei Loeffler algoritmuose DCT realizuojama naudojant dvimatės erdvės pasukimus (daugybas iš konstantos) ir „drugelio“ operacijas. Šis naujas algoritmas plokštumos pasukimo operacijas pakeičia keletu dvinarių reikšmių, kurios vadinamos pakėlimo žingsniais (3.6 pav.). Šie pakėlimo žingsniai turi tokį formatą $\frac{a}{2^b}$ , kur *a* yra sveikasis skaičius, o *b* – natūralusis. Tokius pakėlimo žingsnius nesunku realizuoti aparatinėmis priemonėmis naudojant poslinkio ir sudėties operacijas, kurios atliekamos žymiai paprasčiau, nei daugyba. Algoritmui galima pritaikyti bet kokį DCT signalų perėjimo grafą.




*3.6 pav. Bendras plokštumos pasukimas (a), pakėlimo žingsnių seka (b) [30]*

Pakėlimo koeficientus galima apskaičiuoti naudojantis šiomis formulėmis:

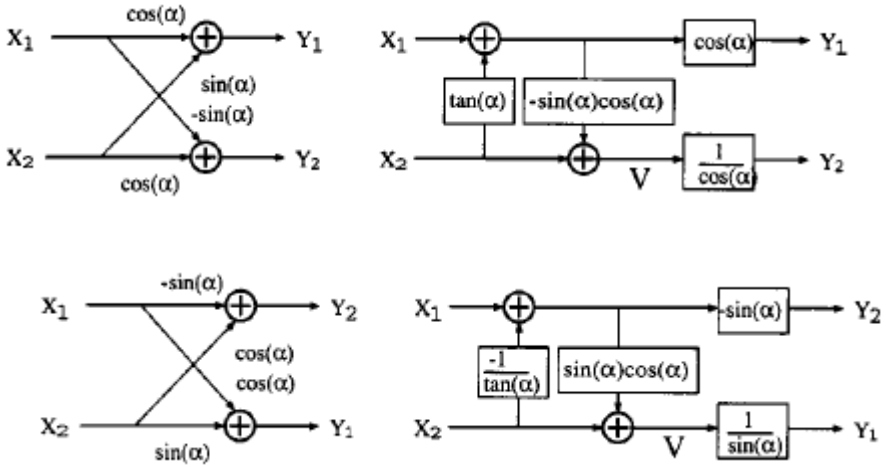
$$p = \frac{r_{12}}{r_{11}} \tag{14}$$

$$u = \frac{r_{11} r_{21}}{r_{11} r_{22} - r_{21} r_{12}} \tag{15}$$

$$K_1 = r_{11} \tag{16}$$

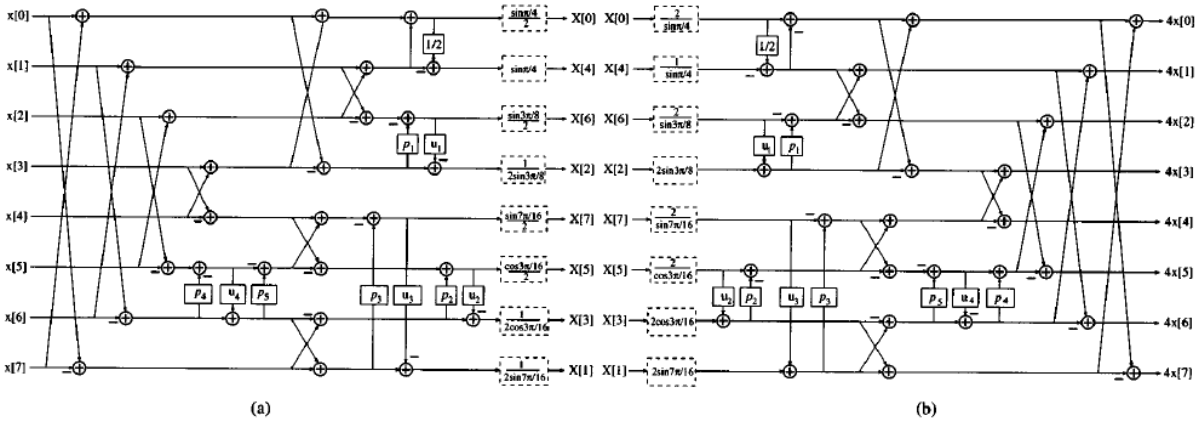
$$K_2 = \frac{r_{11} r_{22} - r_{21} r_{12}}{r_{11}} \tag{17}$$

Pritaikius (14) – (17) formules plokštumos pasukimo operacijoms, gaunama pakėlimo žingsnių schema, pavaizduota 3.7 pav.




*3.7 pav. Pakėlimo žingsnių schema [30]*

Kadangi Loeffler algoritmas turi teoriškai mažiausią daugybos operacijų skaičių, ir yra naudojamas standartiniame JPEG algoritme, jis pasirinktas kaip šio BinDCT algoritmo pagrindas. Pritaikius 3.7 pav. pateiktas schemas Loeffler algoritmo grafui (3.4 pav.), gaunamas BinDCT algoritmo signalų perėjimo grafas, pavaizduotas 3.8 pav.




*3.8 pav. BinDCT signalų perdavimo grafas. a- DCT algoritmas, b – IDCT algoritmas [30]*

Skaičiuojant pakėlimo žingsnius (14) – (17) formulėmis, būtina pasirinkti *p* ir *u* reikšmių konfigūracijas. Nuo šio pasirinkimo priklauso algoritmo tikslumas. Algoritmo autoriai pasiūlė tris galimas pakėlimo žingsnių konfigūracijas kurios skiriasi savo tikslumu. Reikšmės pateiktos 3.2 lentelėje.

*3.2 lentelė. Algoritmo konfigūracijos [32]*

| | **A** | **B** | **C** |
|---|---|---|---|
| $p_1$ | 3/8 | 3/8 | 3/8 |
| $u_1$ | 3/8 | 3/8 | 3/8 |
| $p_2$ | 7/8 | 1/2 | 7/8 |
| $u_2$ | 1/2 | 1/8 | 1/2 |
| $p_3$ | 3/16 | 1/8 | 1/8 |
| $u_3$ | 3/16 | 0 | 0 |
| $p_4$ | 7/16 | 0 | 0 |
| $u_4$ | 11/16 | 3/8 | 3/8 |
| $p_5$ | 3/8 | 5/8 | 5/8 |

Pritaikius šitokį daugybos būdą Loeffler algoritmui (3.4 pav.), gaunamas greitasis dvimatės diskrečiosios transformacijos algoritmas, naudojantis tik sveikųjų skaičių sudėties ir postūmio operacijas.

Tokį signalų perdavimo grafą patogu tiesiogiai realizuoti programinėmis ar aparatinėmis priemonėmis, visiškai nenaudojant daugybos operacijų. Šios operacijos pakeičiamos loginio poslinkio ir sudėties operacijomis tokiu būdu: $\frac{3}{8}x=\frac{1}{8}x+\frac{2}{8}x=\frac{1}{8}x+\frac{1}{4}x=\frac{x}{2^3}+\frac{x}{2^2}$ , tai aparatūroje atliekama kintamąjį *x* paslinkus 3 bitus į dešinę ir pridėjus x, paslinktą per 2 bitus į dešinę.

Analogiškai skaidomi ir kiti koeficientai.

## *3.2 Tyrimo metodika*

Šiame darbe tyrimą pradedame matematiniu algoritmų modeliavimu. Įsitikiname, ar jie veikia teisingai ir atitinka JPEG kodavimo standartą. Išsiaiškiname algoritmo kokybę, t.y.: jo įnešamų į vaizdą klaidų kiekį ir įtaką regimai vaizdo kokybei. Toliau pateikiame priemones aparatūros aprašui VHDL kalba modeliuoti ir sintezuoti. Taip pat aprašomos FPGA matricos, jų panaudojimas tyrime. Aptariamas algoritmų pritaikymas FPGA matricoms, specifinių įrenginių panaudojimas ir poreikis.

### 3.2.1 Matematinis modeliavimas

Siekiant ištirti ir surasti tinkamus algoritmus aparatinei realizacijai, atliekamas matematinis modeliavimas MATLAB aplinkoje.

MATLAB (angl. **mat**rix **lab**oratory) tai matematinio skaičiavimo aplinka, paremta Matlab

programavimo kalba. MATLAB naudojimas įgalina atlikti veiksmus su matricomis, braižyti funkcijas bei duomenų diagramas, realizuoti algoritmus ir bendrauti su kitomis programomis, parašytomis kitomis kalbomis, tokiomis kaip C, C++ ar Fortran. MATLAB aplinkoje programuojama MATLAB kalba, kuri palaiko funkcijas, kintamuosius, vektorių tipus bei klases [31].

Matematinio modeliavimo metu siekiama išsiaiškinti algoritmu kodavimo tikslumą, kai skaičiavimams naudojamos tik sveikųjų skaičių operacijos. Algoritmus realizuoti ir modeliuoti MATLAB kalba yra paprasčiau nei VHDL ar kitomis aparatūros projektavimo kalbomis. Modeliuojant algoritmus nereikia modeliuoti papildomos algoritmo valdymo įrangos. MATLAB modeliai leidžia lengvai keisti algoritmų parametrus, panaudojimą bei testus ir suteikia patogias statistinės analizės priemones. Be to, matematinių modelių modeliavimas yra daug spartesnis nei aparatinių modelių [32].

Aukščiau išvardinti matematinių modelių privalumai leidžia ženkliai sumažinti tyrimo laiką, nes galima greitai ir paprastai atmesti algoritmus, netinkamus sveikųjų skaičių aritmetikai. Dėl šių priežasčių kiekvieną DCT ir IDCT algoritmą aprašysime MATLAB modeliu prieš realizuodami jį VHDL kalba.

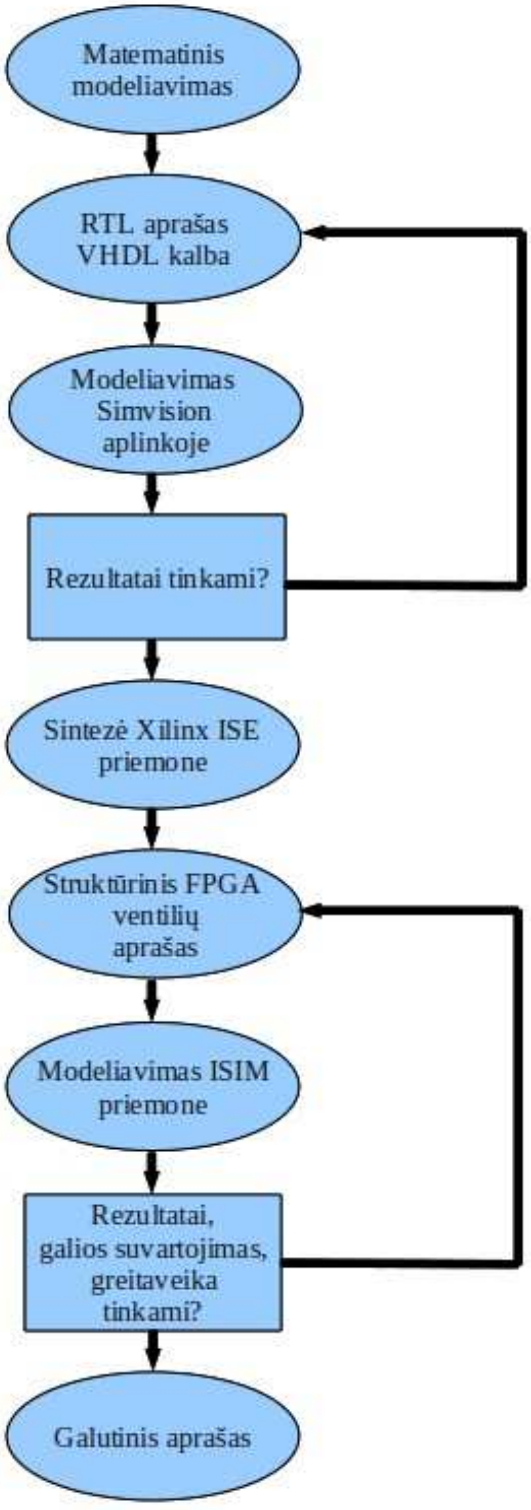
Skaičiuojant kodavimo kokybę, naudojamas maksimalus signalo ir triukšmo santykis (Peak Signal to Noise Ratio – PSNR). Šis dydis ganėtinai tiksliai aprašo vaizdo kokybę. Jis lengviausiai išreiškiamas per vidutinį kvadratinį nuokrypį (Mean Square Error – MSE), kuris dviems vienspalviams vaizdams I ir K, kur vienas vaizdas yra kito aproksimacija su triukšmu, aprašomas (18) bei (19) formulėmis:

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left[ I(i,j) - K(i,j) \right]^2 \tag{18}$$

$$PSNR = 10\log_{10}\left( \frac{MAX_I^2}{MSE} \right) \tag{19}$$

Čia $MAX_I$ yra maksimali galima vaizdo taško reikšmė. Jei vaizdo taškai koduojami 8 bitais, ši reikšmė yra 255. Dažniausiai DCT algoritmais apdorotiems vaizdams PSNR reikšmės svyruoja apie 30 – 50 decibelų (dB). Didesnis PSNR santykis reiškia geresnę vaizdo kokybę [33].

### 3.2.2 Projektavimo eiga




*3.9 pav. Projekto eiga*

Galutinis šio projekto tikslas yra specialiosios paskirties vienlustė sistema – ASIC. Tokia sistema būtų sudaryta iš mūsų ištirtų ir tinkamiausiais išrinktų algoritmų ir jiems valdyti reikalingos įrangos. Kitas žingsnis po matematinio modeliavimo yra DCT bei IDCT algoritmų

ir papildomos atminties bei valdymo aparatūros RTL (register transfer level) lygio aprašas VHDL aparatūros aprašymo kalba. Toks kalbos aprašas gali būti sintezuojamas į struktūrinį aparatūros modelį. Kitas svarbus žingsnis ASIC projektavime yra DCT ir IDCT aprašų pritaikymas FPGA matricoms. FPGA matricos nėra tinkamos energiją taupantiems prietaisams, tačiau praktikoje yra naudojamos kaip patikimas sistemų testavimo būdas. Tyrimai rodo, kad ASIC DCT algoritmo realizacija eikvotų apie 1/100-tąją dalį elektros energijos, suvartojamos bendros paskirties procesoriaus, atliekančio tą pačią funkciją [34]. 3.9 pav. pavaizduota projektavimo eiga DCT ir IDCT algoritmų realizacijai. Matematinis modeliavimas ir RTL modeliavimas leidžia testuoti pasirinktą algoritmą su norimais įvedimo duomenimis, paduodant duomenis įvedimo srautui ir skaitant išėjimo srautą. Toks testavimas reikalingas, norint patikrinti teisingą matematinio MATLAB ir aparatinio VHDL modelio veikimą. Testavimas taip pat leidžia patikrinti papildomos įrangos veikimą, stebėti priderinamų ar išmetamų iš projekto komponentų įtaką galutiniam rezultatui. Matematinis modeliavimas leidžia apskaičiuoti algoritmo įvedamas klaidas ir koduotų vaizdų PSNR santykį su originalais, kuris yra pagrindinis vaizdo kodavimo kokybės matas.

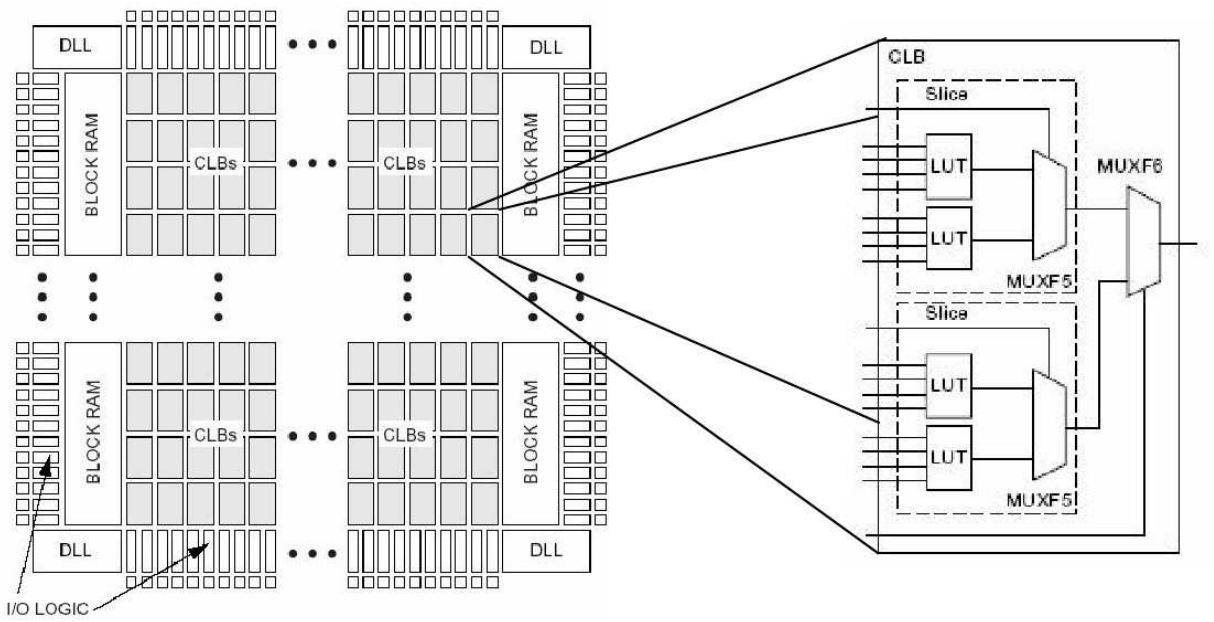
### 3.2.3 FPGA matricos

FPGA (angl. Field-programmable Gate Array) – tai mikroschemos, kurias jau po gamybos, konfigūruoja vartotojas. Tokių matricų konfigūracija dažniausiai aprašoma sistemų aprašymo kalba, pvz. VHDL, lygiai taip pat kaip specializuotos paskirties mikroschemų (ASIC). FPGA matricos naudojamos realizuoti bet kokiai funkcijai, kurią galėtų atlikti ASIC. Kadangi FPGA duoda galimybę pakeisti funkcionalumą po gamybos, vartotojui panorėjus atlikti dalinį ar visišką perkonfigūravimą be papildomų gamybos išlaidų, lyginant su ASIC, šios matricos teikia daug privalumų projektavimo procese, ir yra tarpinis žingsnis į ASIC kūrimą [35][36].

FPGA prototipų kūrimas, dar vadinamas ASIC ar vienlusčių sistemų prototipų kūrimu, yra metodas testuoti vienlustės sistemos funkcionalumui, dar nepradėjus sistemos gamybos. Toks aparatūros testavimo metodas, taip pat programinės ir aparatinės įrangos suderinamumo testavimas tapo tendencija. Vienlustės sistemos testavimas FPGA matricoje yra patikimas būdas įsitikinti veikimo teisingumu. Toks metodas daug patikimesnis, nei remtis vien modeliavimo rezultatais, kurie negali užtikrinti norimo tikslumo, teisingumo ir greičio [37].

FPGA matrica susideda iš programuojamos logikos elementų, vadinamų loginiais blokais, įėjimo ir išėjimo blokų ir hierarchijos konfigūruojamų sujungimų, leidžiančių šiuos blokus įvairiai sujungti tarpusavyje norima konfigūracija. Skirtingi blokų sujungimai gali atlikti

sudėtingas kombinacines užduotis, arba būti naudojami kaip paprasti loginiai elementai, tokie kaip AND ar OR. Dauguma FPGA matricų turi atminties elementus, nuo paprastų ventilių iki RAM (angl. Random Access Memory – operatyvinės atminties) elementų. Kai kurios matricos turi analoginių funkcijų, tokių kaip išėjimų galios valdymas ar skaitmeniniai – analoginiai bei analoginiai - skaitmeniniai keitikliai, kurie šioms plokštėms leidžia veikti kaip tikrai vienlustei sistemai [38]. FPGA matricos struktūrinis vaizdas pateiktas 3.10 pav.




*3.10 pav. FPGA matricos struktūrinis vaizdas*

FPGA yra lėtesnės, naudojančios daugiau energijos ir mažiau funkcionalios nei ASIC. Tyrimai rodo, kad projektui, realizuotam FPGA, vidutiniškai reikia 29 kartų daugiau ploto, 87 kartus daugiau statinės galios. Jo darbo greitis vidutiniškai 4,6 kartus mažesnis nei projekto, realizuoto ASIC [39].Pagrindinis FPGA privalumas yra galimybė perprogramuoti logiką, taip ištaisant klaidas. Tai leidžia greičiau pristatyti produktą į rinką (shorter time to market), ir sumažina negrįžtamus projektavimo kaštus. Taipogi gamintojas turi trečią pasirinkimą – kurti savo projektą naudojant FPGA, ir gamybai atiduoti tik galutinę produkto versiją, kai ji jau nebekeičiama [40].

## 3.3 Papildoma aparatūra dvimačiam algoritmui

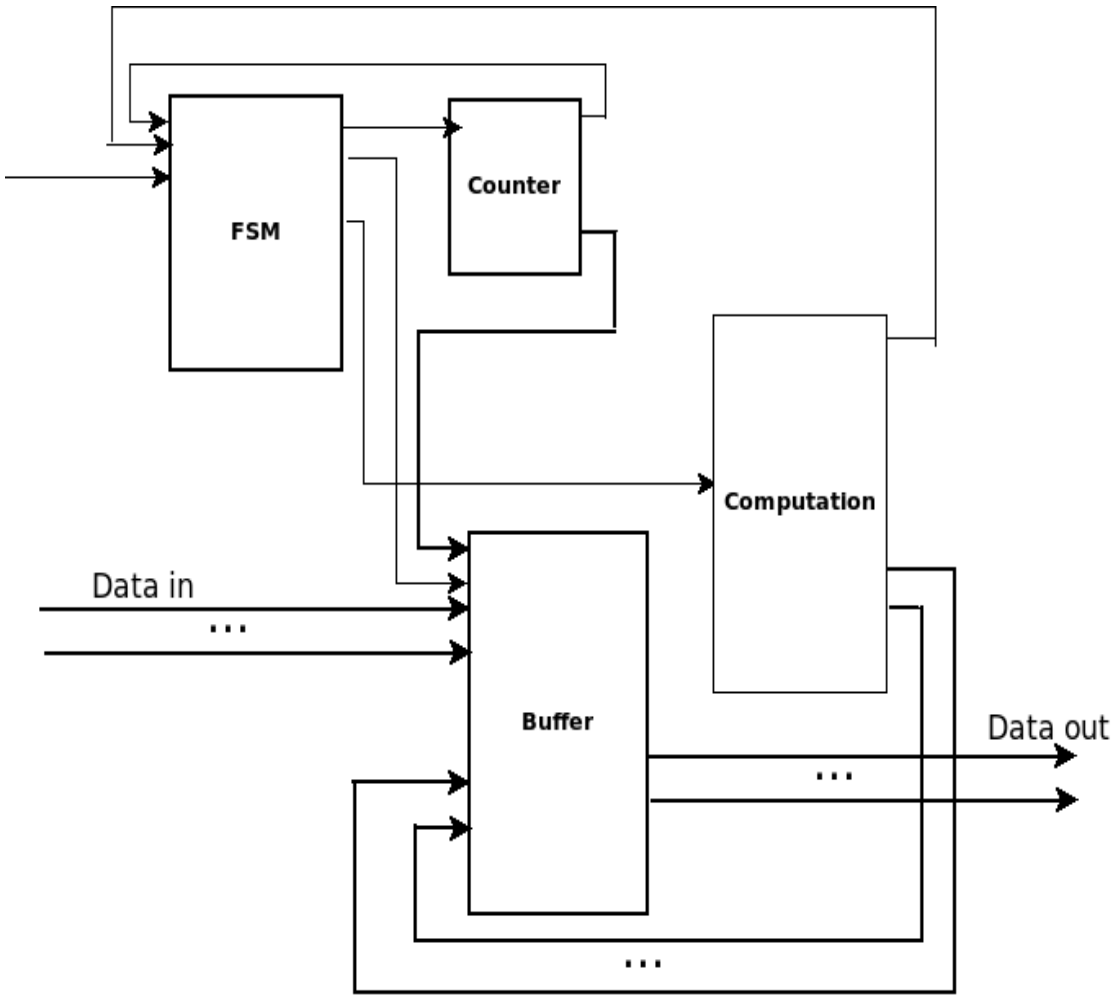
Eilučių/stulpelių algoritmai atliekami keturiais žingsniais:

- Atliekama vienmatė DCT ar IDCT kiekvienai įėjimo matricos eilutei, rezultatas išsaugojamas tarpinėje atmintyje.
- Matrica transponuojama.
- Dar kartą atliekama vienmatė DCT visoms transponuotoms matricos eilutėms.

- Matrica vėl transponuojama ir išduodama į išėjimus.

  Šiems veiksmams atlikti būtina papildoma aparatūra. Projektuojant šį algoritmą aparatūroje, paminėtus veiksmus galima supaprastinti. Tai pagreitina algoritmo veikimą, sumažina ploto perviršį. Praktiškai aparatūroje atliekama veiksmų seka yra tokia:
- Atliekama vienmatė DCT ar IDCT kiekvienai įėjimo matricos eilutei, rezultatas išsaugojamas tarpinėje atmintyje. Prieš tai ten buvusi informacija prarandama, nes nebėra reikalinga.
- Antrą kartą DCT ar IDCT blokui įėjime paduodami matricos stulpeliai, todėl nebereikalingas transponavimas
- Atliekama DCT ar IDCT matricos stulpeliams, rezultatai išsaugomi toje pačioje atmintyje. Prieš tai ten buvusi informacija prarandama, nes nebėra reikalinga.
- Į išėjimus atiduodami matricos stulpeliai, o išėjimo duomenys saugomi teste (interpretuojami) kaip eilutės. Tokiu būdu nereikia papildomo matricos transponavimo.

Remiantis šias žingsniais, sukurtas DCT bei IDCT algoritmų valdymo aparatūros aprašas VHDL kalba. Ši aparatūra leidžia įdiegti patį DCT ar IDCT algoritmą kaip nepriklausomą standartinį bloką, taigi visiems algoritmams realizuoti galima panaudoti tą pačia papildomą įrangą. Blokinė šios aparatūros aukščiausio hierarchijos lygmens schema pateikiama 3.11 pav. Kiekvienas schemos blokas aprašytas detaliau.



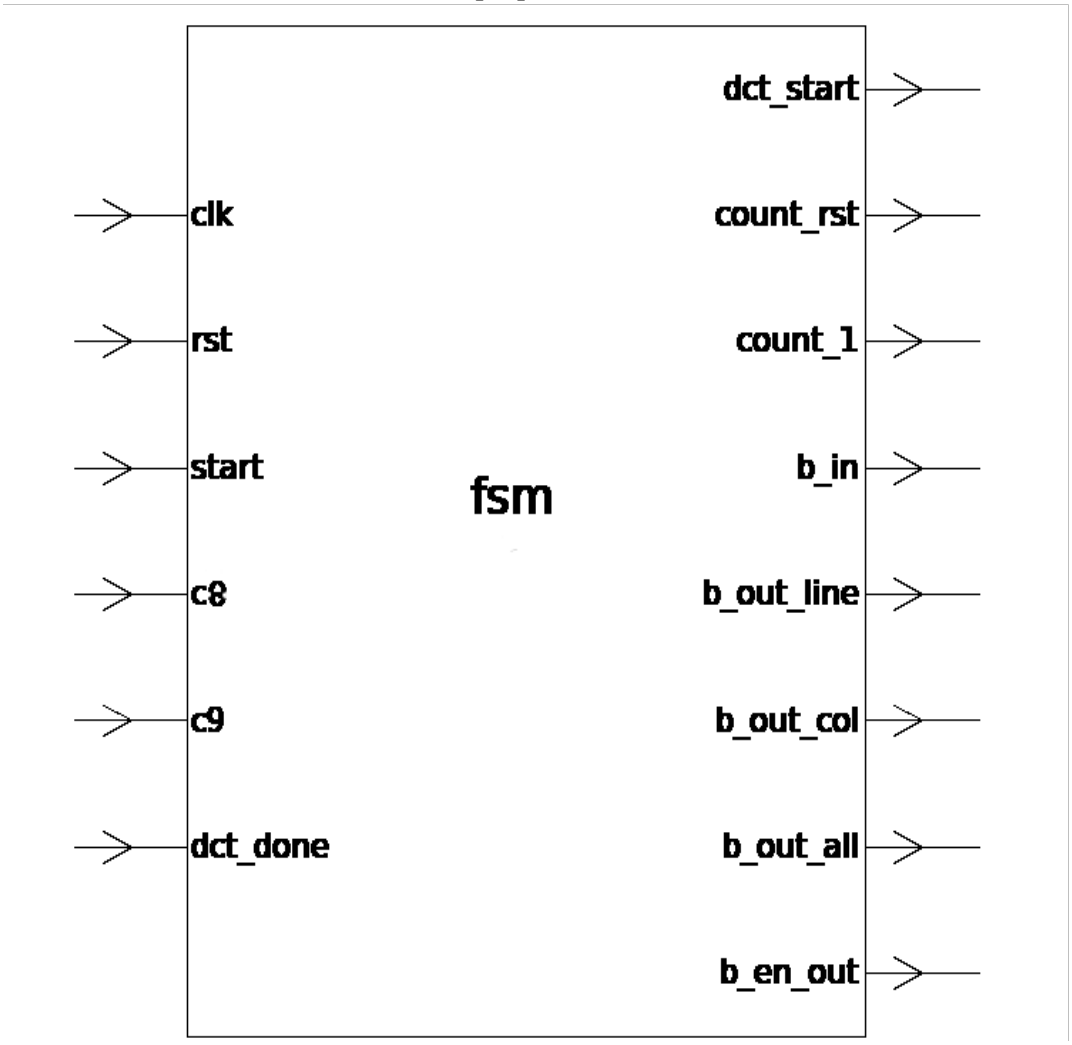

*3.11 pav. Aukščiausiojo hierarchijos lygmens schema*

### 3.3.1 Valdymo automatas (FSM)

Valdantysis baigtinis būsenų automatas – matematinė abstrakcija, naudojama skaitmeninės logikos valdymui. Jo elgseną aprašo baigtinis būsenų skaičius, perėjimų tarp būsenų sąlygos ir veiksmai, atliekami kiekvienoje būsenoje. Automatą aprašo perėjimų tarp būsenų grafas, nurodantis, kokias sąlygas reikia įvykdyti pereinant iš vienos būsenos į kitą. Kiekvienoje būsenoje automatas privalo įvykdyti tam tikrus veiksmus. Perėjimai priklauso nuo esamos būsenos ir įėjimo signalų - toks būsenų automatas vadinamas Mili automatu. Kiekvienoje būsenoje išduodami tuo metu reikalingi valdymo signalai. Darbą automatas pradeda pradinėje būsenoje ir eina per būsenas pagal perėjimų sąlygas. Galinės būsenos priėjimas reiškia sėkmingą darbo pabaigą. Šiame modelyje automato galinė būsena sutampa su pradine, t.y.: baigęs darbą automatas grįžta į pradinę būseną ir vėl pasiruošia darbui. Baigtiniai automatai išsprendžia daugelį su valdymu susijusių problemų.

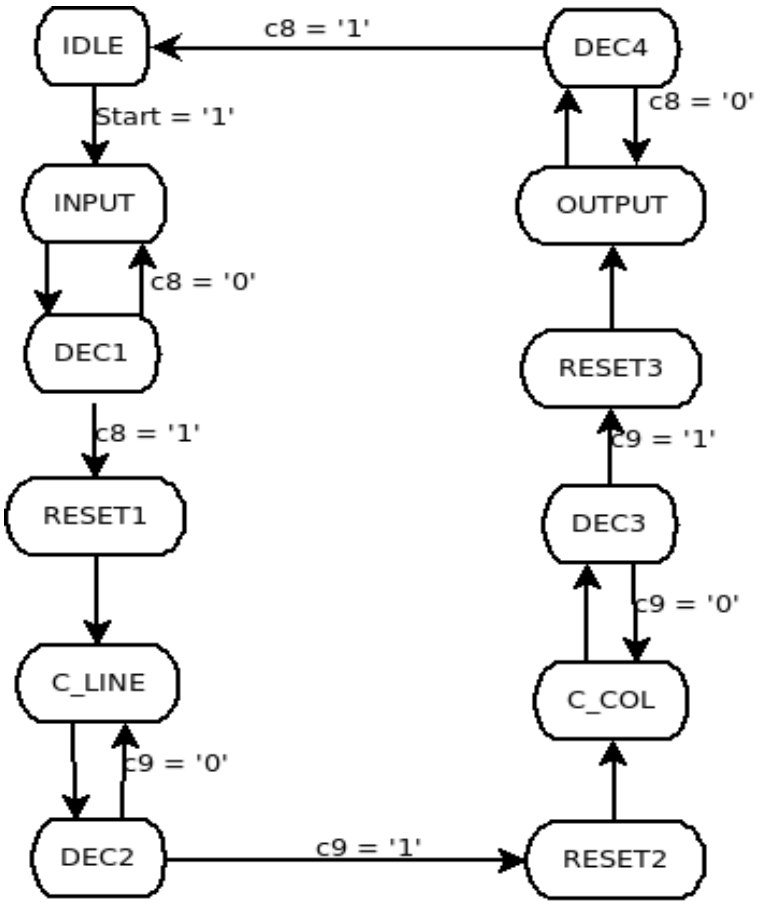
Esamą būseną lemia prieš tai buvusi būsena, todėl automatas turi savo vidinę atmintį,

atspindinčią, kokias būsenas perėjo sistema iki esamos būsenos. Perėjimai yra veiksmai, kuriuos būtina atlikti tam tikru laiko momentu [41].




*3.12 pav. Baigtinio automato sąsaja*

Baigtinis automatas, kurio sąsaja pavaizduota 3.12 pav. yra pagrindinis algoritmo valdymo įrenginys. Vykdant būsenas žemiau pateikta tvarka, suformuojami atitinkami valdymo signalai kitiems įrenginiams. Būsenų perėjimo grafas pavaizduotas 3.13 pav. Grafe pavaizduotų būsenų vykdymas leidžia automatui, priklausomai nuo įėjimo signalų nuosekliai išduoti tuo metu reikalingus valdymo signalus ir taip valdyti kitų įrenginių darbą bei duomenų perdavimą tarp jų.




*3.13 pav. Valdančiojo baigtinio automato būsenų perėjimo grafas*

Valdančiojo automato įėjimo ir išėjimo signalai pateikti 3.3 lentelėje.

*3.3 lentelė. Valdančiojo automato signalai*

| Pavadinimas | Tipas | Funkcija |
|---|---|---|
| `clk` | Įėjimas | Sinchronizacijos signalas |
| `rst` | Įėjimas | Nustatymo į pradinę būseną signalas |
| `c8` | Įėjimas | Išduoda skaitliukas, suskaičiavęs iki 8 |
| `c9` | Įėjimas | Išduoda skaitliukas, suskaičiavęs iki 9 |
| `Start` | Įėjimas | Pradeda algoritmo vykdymą |
| `dct_done` | Įėjimas | Nurodo, algoritmo pabaigą |
| `dct_start` | Išėjimas | Pradeda 1-D DCT operaciją |
| `count_rst` | Išėjimas | Skaitliuko nustatymo į pradinę būseną signalas |
| `Count_1` | Išėjimas | Skaitliuko didinimo signalas |
| `b_in` | Išėjimas | Nurodymas buferiui vykdyti duomenų skaitymą |
| `b_out_line` | Išėjimas | Nurodymas buferiui išduoti matricos eilutes |
| `b_out_col` | Išėjimas | Nurodymas buferiui išduoti matricos stulpelius |
| `b_out_all` | Išėjimas | Nurodymas buferiui išduoti galutinį rezultatą |
| `b_en_out` | Išėjimas | Prijungia išėjimus prie buferio |

Norint įvykdyti dvimate DCT ir IDCT transformaciją reikia atlikti tokius veiksmus:

1. Įvesti 8x8 duomenų matricą.
2. Atlikti vienmatę DCT ar IDCT matricos eilutėms.
3. Atlikti vienmatę transformaciją stulpeliams su duomenimis iš 2 žingsnio.
4. Išvesti gautus rezultatus.

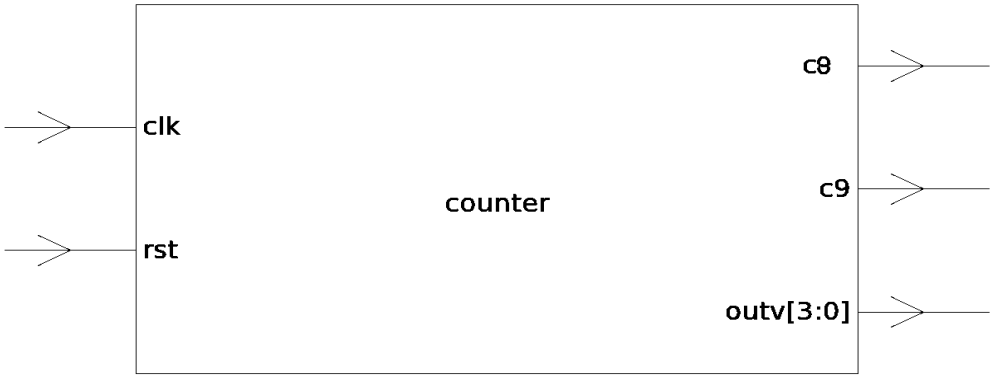
Automatas darbą pradeda IDLE būsenoje. Kai gaunamas *start* signalas automatas pereina į INPUT būseną, kurioje priimami duomenys (1 žingsnis). DEC1 būsenoje sprendžiama ar baigtas duomenų įvedimas. Jei *c8* signalas lygus 1, tai reiškia, kad įvesta visa 8x8 matrica. Pereinama į RESET1 būseną, kurioje į pradinę būseną nustatomas skaitliukas. Pereinama į C_LINE būseną, kurioje atliekama skaičiavimai eilutėms (2 žingsnis). Skaičiavimo rezultatai rašomi į duomenų vietą atmintyje. Pasibaigus 2 žingsniui pereinama į C_COL būseną, kurioje atliekami skaičiavimai su stulpeliais (3 žingsnis), rezultatą išsaugant duomenų vietoje. Baigus šį žingsnį atmintyje saugomi duomenys yra 2-D DCT transformacija įvestiems duomenims. Automatas pereina į OUTPUT būseną, kurioje rezultatai išduodami į išėjimus (4 žingsnis).

Atliekant šiuos veiksmus, kiekvienoje būsenoje išduodami valdymo signalai, kurie pavaizduoti 3.4 lentelėje.

*3.4 lentelė. Valdančiojo automato išduodamų signalų reikšmės*

| | IDLE | INPUT | DEC1 | RESET1 | C_LINE | DEC2 | RESET2 | C_COL | DEC3 | RESET3 | OUTPUT | DEC4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b_en | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_out_line | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_out_col | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| b_out_all | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| b_en_out | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| dct_start | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| count_rst | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| count_1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

### 3.3.2 Skaitliukas



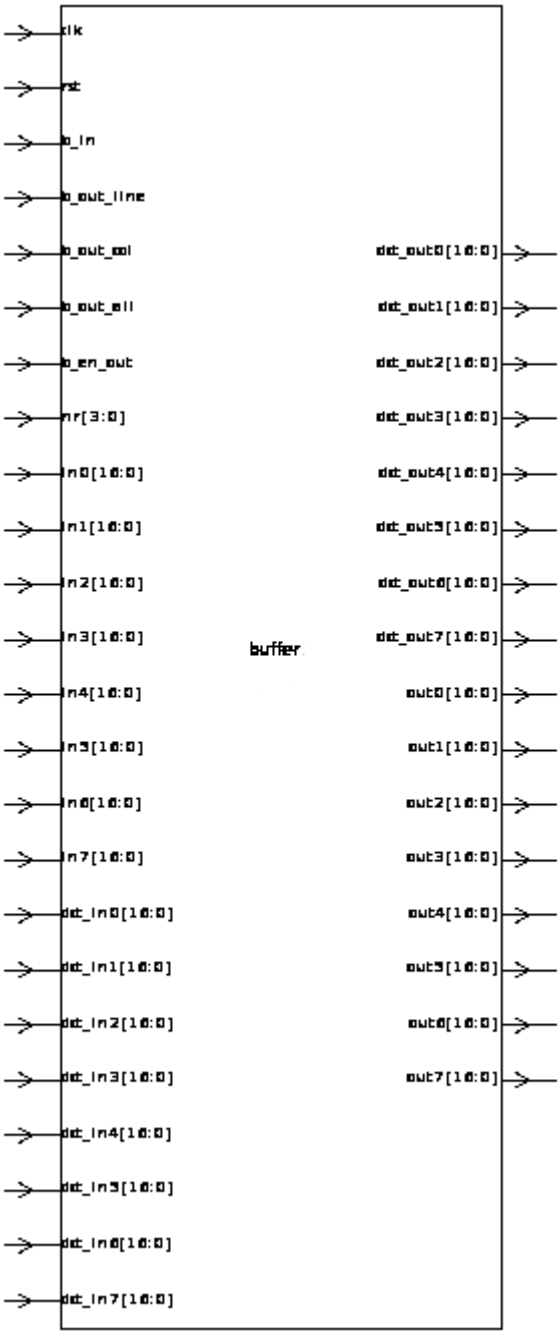

*3.14 pav. Skaitliuko sąsaja*

Skaitliukas yra įrenginys, skaičiuojantis kiek kartų įvyko tam tikras įvykis. 3.14 pav. pavaizduotas skaitliukas skaičiuoja skaitymo ir rašymo operacijas. Jis padidėja vienetu, kai gauna priekinio fronto signalą `clk` įėjime. Skaičiavimo rezultatas reikalingas tam, kad nustatyti, kada baigėsi atitinkama skaitymo arba rašymo operacija. Skaitliuko įėjimo ir išėjimo signalai pavaizduoti 3.5 lentelėje.

*3.5 lentelė. Skaitliuko signalai*

| **Pavadinimas** | **Tipas** | **Funkcija** |
|---|---|---|
| `clk` | Įėjimas | Sinchronizacijos signalas |
| `rst` | Įėjimas | Nustatymo į pradinę būseną signalas |
| `c8` | Išėjimas | Išduodamas suskaičiavus iki 8 |
| `c9` | Išėjimas | Išduodamas suskaičiavus iki 9 |
| `Outv[3:0]` | Išėjimas | Skaičiavimo rezultatas |

### 3.3.3 Buferis




*3.15 pav. Buferio sąsaja*

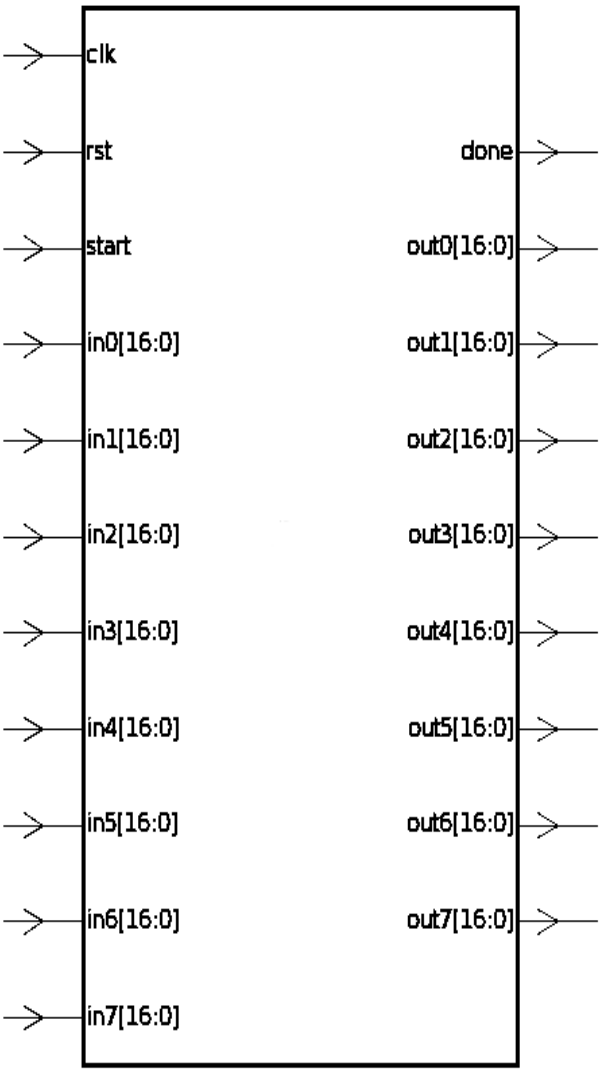
Buferis yra atmintis, sauganti tarpinių skaičiavimų duomenis ir galutinį rezultatą. Buferyje saugomos reikšmės gaunamos iš išorinių įėjimų arba iš DCT skaičiavimo bloko, ir rašomos į išorinius išėjimus arba į DCT bloką, priklausomai nuo valdymo automato siunčiamų signalų, kurie nurodyti 3.6 lentelėje. Buferio sąsaja su kitais blokais pavaizduota 3.15 pav. Ši atmintis turi dvi 16 bitų pločio įėjimo ir išėjimo magistrales. Pirmoji pora: `in0 - in8` bei `out0 - out8`– naudojama duomenims iš išorės paimti. Antroji pora: `dct_in0 -`

`dct_in8` bei `dct_out0 - dct_out8` – vidiniai sujungimai su DCT skaičiavimo bloku. Šios magistralės skirtos į DCT paduoti tarpinius skaičiavimo rezultatus bei išsaugoti gautą transformuotą rezultatą. Įėjimo ir išėjimo magistralės valdomos valdančiojo automato signalais. Priklausomai nuo atliekamos operacijos buferiui nurodoma įrašyti įeinančius duomenis į matricos eilutes arba stulpelius, arba išduoti duomenis iš matricos eilučių ar stulpelių arba skaityti duomenis iš išorės ar rašyti juos į išėjimus.

*3.6 lentelė. Buferio valdymo signalai*

| Pavadinimas | Tipas | Funkcija |
|---|---|---|
| `clk` | Įėjimas | Sinchronizacijos signalas |
| `rst` | Įėjimas | Nustatymo į pradinę būseną signalas |
| `b_in` | Įėjimas | Nurodymas įvesti pradinius duomenis |
| `b_out_line` | Įėjimas | Nurodymas išduoti eilutę į DCT bloką |
| `b_out_col` | Įėjimas | Nurodymas išduoti stulpelį į DCT bloką |
| `b_out_all` | Įėjimas | Nurodymas išduoti rezultatus |
| `b_out_en` | Įėjimas | Išėjimų įjungimas |
| `in0[16:0] .. in7[16:0]` | Įėjimas | Duomenų įvedimo magistralės |
| `dct_in0[16:0] .. dct_in7[16:0]` | Įėjimas | Duomenų iš DCT bloko įvedimo magistralės |
| `dct_out0[16:0] .. dct_out7[16:0]` | Išėjimas | Duomenų į DCT bloką išvedimo magistralės |
| `out0[16:0] .. out7[16:0]` | Išėjimas | Rezultatų išvedimo magistralės |

### 3.3.4 Skaičiavimo blokas




*3.16 pav. Skaičiavimo bloko sąsaja*

Pasirinkti algoritmai yra realizuojami skaičiavimo bloke. Šiame bloke gali būti realizuotas bet kuris vienmatės DCT ar IDCT skaičiavimo algoritmas. Pagrindinis reikalavimas – tinkamai realizuota sąsaja su valdymo aparatūra. Bloko sąsajos signalai pavaizduoti 3.16 pav. ir paaiškinti 3.7 lentelėje.

*3.7 lentelė. Skaičiavimo bloko signalai*

| Pavadinimas | Tipas | Funkcija |
|---|---|---|
| `clk` | Įėjimas | Sinchronizacijos signalas |
| `rst` | Įėjimas | Nustatymo į pradinę būseną signalas |
| `start` | Įėjimas | Nurodymas pradėti skaičiavimą |
| `done` | Išėjimas | Skaičiavimų pabaigos signalas |
| `in0 [16:0].. in7[16:0]` | Įėjimas | Duomenų įvedimo magistralės |
| `out0[16:0]... out7[16:0]` | Išėjimas | Rezultatų išvedimo magistralės |

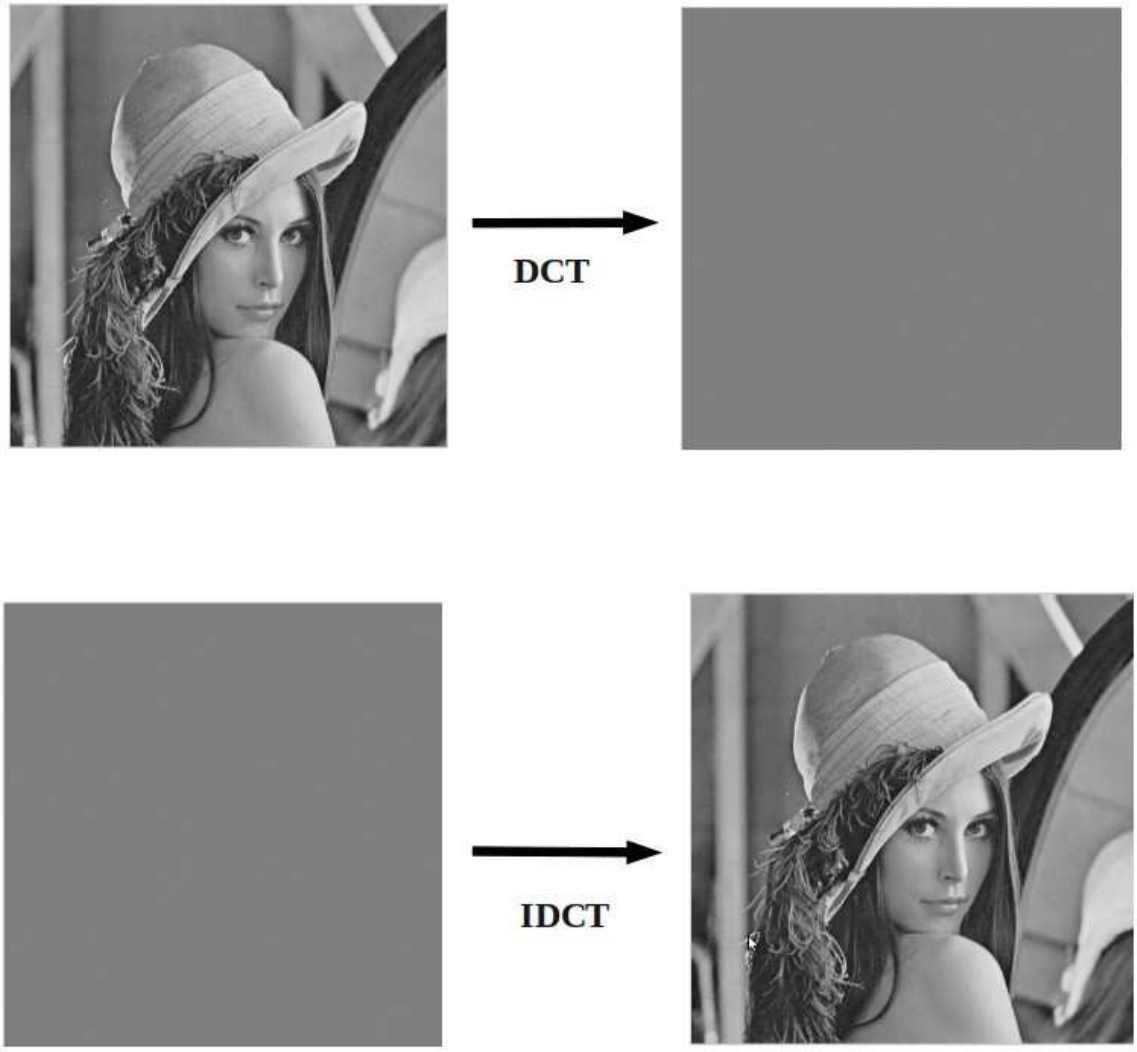
## 3.4 Projektinės dalies išvados

Šioje dalyje aprašyti tolesniam tyrimui pasirinkti 3 DCT ir IDCT algoritmai, jų signalų perdavimo grafai. Aptartos naudojamos konstantos ir kitos savybės, svarbios aparatinėje realizacijoje. Visi pasirinkti algoritmai naudoja daug mažiau matematinių operacijų nei tiesioginis DCT taikymas. Juos patogu realizuoti aparatiškai.

Aprašyta tyrimo metodika, susidedanti iš dviejų pagrindinių dalių: matematinio modeliavimo ir aparatinio modeliavimo. Tyrimo eigoje sukurti papildomi komponentai, valdantys dvimačio DCT algoritmo skaičiavimą eilučių/stulpelių metodu. Šioje dalyje kiekvieno komponento veikimas, funkcija bei signalai aprašyti atskirai. Algoritmų kokybė bus vertinama pagal kodavimo netikslumų skaičių, lyginant koduotą vaizdą su originalu. Vaizdo kodavimo kokybės matu pasirinktas maksimalus signalo ir triukšmo santykis (PSNR) tarp koduoto vaizdo ir originalo.

# 4 Tyrimo dalis

Tyrimo metu pasirinkti paveikslėliai skaidomi į 8x8 blokus ir kiekvienam iš blokų atliekama 2D-DCT transformacija. Pritaikius 2D-IDCT transformaciją gaunamas dekoduotas vaizdas. Šie veiksmai atliekami matematiniu ir aparatiniu algoritmo modeliu ir, siekiant algoritmus palyginti vienodomis sąlygomis, taikomi visiems algoritmams. Užkodavus vaizdą DCT transformacija jo peržiūrėti neįmanoma. Norint paveikslėlį peržiūrėti, būtina jį atkoduoti IDCT transformacija. Kodavimo ir dekodavimo pavyzdžiai pavaizduoti 4.1 pav.

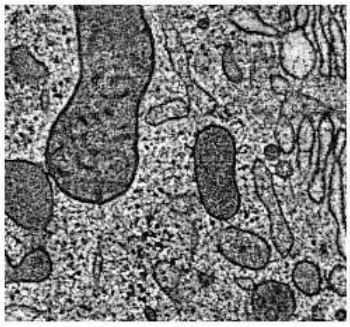



*4.1 pav. Kodavimas DCT ir dekodavimas IDCT algoritmu*

Algoritmų bandymai atliekami su 3 testiniais paveikslėliais, pavaizduotais 4.2 pav. Po kodavimo ir dekodavimo skaičiuojamas originalo ir rezultate gauto vaizdo maksimalus signalo/triukšmo santykis (PSNR). Tie patys paveikslėliai naudojami visų algoritmų testavime,

siekiant palyginti rezultatus.


Lena.bmp


Cell.bmp


Fruit.bmp

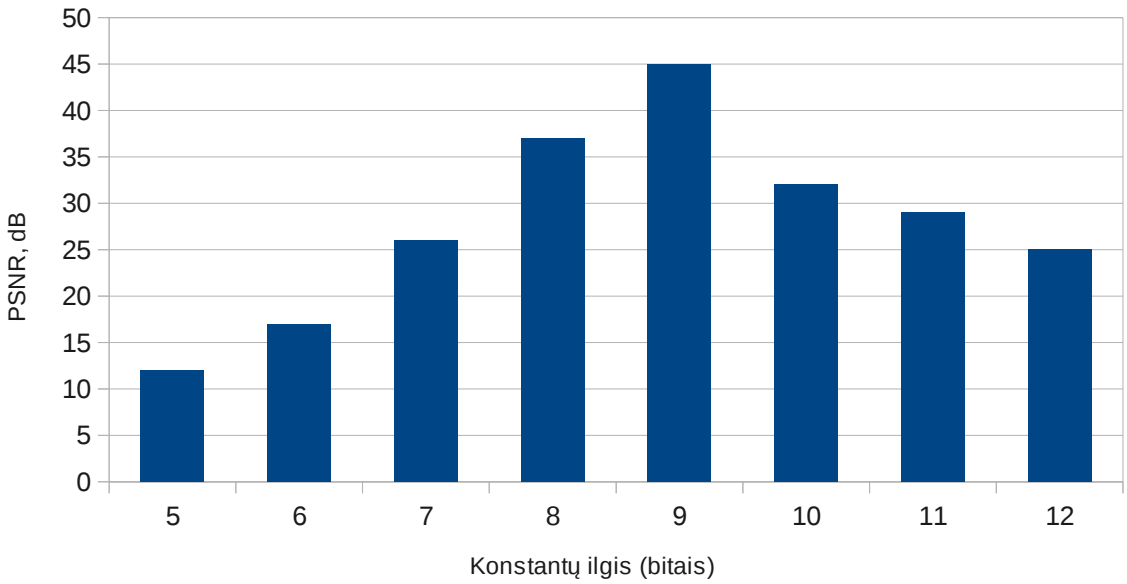
*4.2 pav. Testavimui naudojami paveikslėliai*

Dvejetainių failų skaitymas VHDL kalboje nėra standartizuotas. Norint aparatūros modeliui duoti paveikslėlį, reikia panaudoti papildomą išankstinį duomenų apdorojimą (pre-processing). Šiam etapui panaudojamas MATLAB įrankis. Juo vaizdo informacijos failas skaidomas į 8x8 vaizdo taškų dydžio matricas. Vaizdo taškai užkoduojami dešimtainėje sistemoje 8 bitų ilgio skaičiais be ženklo. Skaičių matricos sudėliojamos nuosekliai, viena eilute ir surašomos į tekstinį failą. Tekstinį failą galima skaityti aparatūros modelio testavimo įranga. Gauti aparatinio modeliavimo rezultatai surašomi į tekstinį failą. Panaudojamas rezultato apdorojimas (post-processing), kad atstatyti matomą vaizdą. Tam vėl panaudotas MATLAB įrankis, skaitantis tekstinį failą, sudėliojantis skaičių matricas reikiama tvarka ir dešimtainius skaičius paverčiantis vaizdo taškais. Tokį apdorotą vaizdą galima peržiūrėti bei skaičiuoti jo PSNR santykį suoriginalu.

## 4.1 Chen algoritmo matematinis modeliavimas

Algoritmo grafas, pavaizduotas 3.1 pav. gali būti tiesiogiai transformuojamas į matematinį ir aparatinį aprašą, tačiau jame pavaizduoti slankaus kablelio skaičiavimai yra per daug sudėtingi, kad juos efektyviai atlikti aparatinėmis priemonėmis. Matematikos atlikimui aparatūroje dažniausiai tenka apsiriboti sveikųjų skaičių operacijų naudojimu. Tam algoritme naudojamos trupmeninės konstantos keičiamos į proporcingai didesnius sveikuosius skaičius. Tai atliekama padauginant konstantas iš tam tikro ilgio skaičiaus (n = 5,6...12), o atlikus veiksmus rezultatą vėl padalinant iš to paties skaičiaus. Nuo pasirinkto skaičiaus ilgio priklausys gautos konstantos ilgis ir tikslumas, taigi ir realizuoto algoritmo tikslumas. Pasirinkus didesnio ilgio konstantas tikslumas padidėja, nes išsaugoma daugiau skaitmenų po kablelio, tačiau reikia didinti skaičius saugančios ir apdorojančios aparatūros kiekį, sudėtingėja ir ilgėja tokių skaičių perdavimas, operacijos su jais, didėja ir skaičių perpildos tikimybė. Sveikųjų skičių naudojimas yra pagrindinė DCT algoritmo kokybės praradimo priežastis.

Siekiant išsiaiškinti konstantų ilgio įtaką kokybei, atliekamas papildomas matematinis modeliavimas. Jo metu algoritmas modeliuojamas su skirtingais konstantų ilgiais. Gauti kodavimo tikslumo rezultatai pateikti 4.3 pav.




*4.3 Pav. PSNR priklausomybė nuo konstantų ilgio*

Siekiant kiek įmanoma sumažinti aparatūros plotą ir galią, Chen algoritmui pasirinktas 9 bitų konstantų ilgis. Matematinis modeliavimas parodė, kad naudojant ilgesnes nei 9 bitų konstantas atsiranda perpilda 32 bitų kintamuosiuose, todėl smarkiai sumažėja algoritmo tikslumas. Skaičiavimų pabaigoje reikalinga papildoma korekcija, norint gauti teisingus rezultatus.

Chen algoritmui reikia šių konstantų:

- `cos(π/4)`
- `sin(π/4)`
- `sin(π/8)`
- `cos(π/8)`
- `cos(3π/8)`
- `sin(3π/8)`
- `sin(π/16)`
- `cos(π/16)`
- `sin(7π/16)`
- `cos(3π/16)`
- `sin(3π/16)`

Modelyje šios konstantos dauginamos iš $2^9$. Tokiu būdu gaunamos sveikosios konstantų reikšmės su kuriomis atliekamas tyrimas. Daugybą iš šių skaičių patogu atlikti aparatiškai,

atliekant aritmetinį postūmį į kairę per 9 bitus. Pakeitus konstantas sveikaisiais skaičiais, kai kurių reikšmės pasidaro vienodos, taigi algoritmui pakanka 7 konstantų.

Modelis panaudotas paveikslėlio kodavimui taikant eilučių/stulpelių metodą. Tyrimui naudojami 3 testavimui pasirinkti paveikslėliai Paveikslėliai užkoduojamai Chen DCT algoritmo matematiniu modeliu, tada atkoduojami Chen IDCT algoritmo matematiniu. Gauti algoritmo matematinio modeliavimo rezultatai pateikti 4.1 lentelėje.

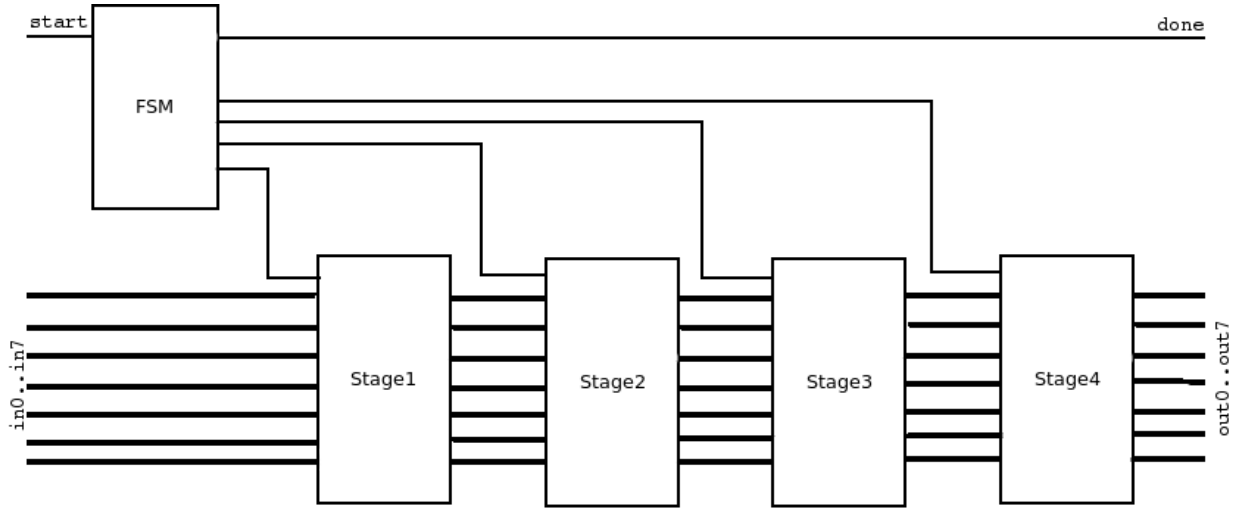
*4.1 lentelė. Chen algoritmo matematinio modeliavimo rezultatai*

| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 45.3 |
| Fruit.bmp | 45.96 |
| Cell.bmp | 35.24 |

Iš lentelės matyti, kad algoritmas praranda labai nedaug vaizdo kokybės. Cell.bmp paveikslėlio kodavimo rezultatai prastesni, nes paveikslėlis turi didžiausią detalumą, mažą kiekį vienodo šviesumo blokų t. y.: mažą koreliaciją. Kaip jau rašyta analitinėje darbo dalyje, tokius vaizdus DCT transformacija apdoroja prasčiau. Koduojant DCT algoritmu, smulkios detalės bei staigūs ryškumo perėjimai praranda daugiau kokybės nei vienodos spalvos blokai.

## 4.2 Chen algoritmo aparatinė realizacija

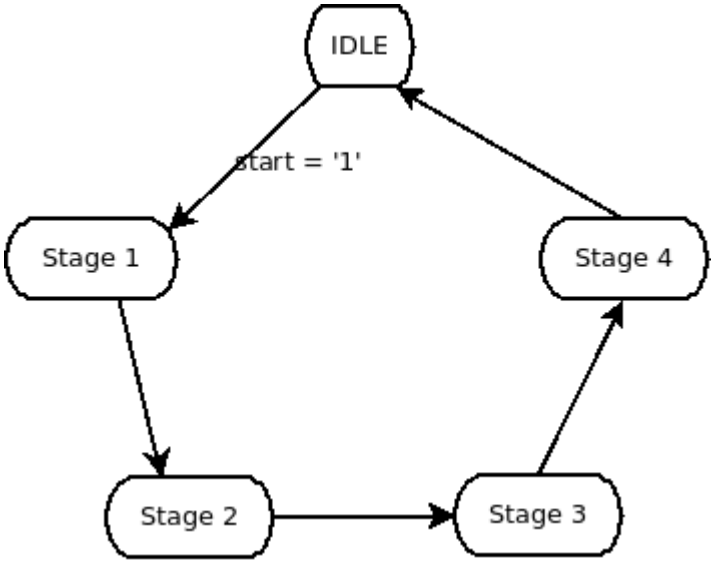
Po matematinio modeliavimo sukuriamas Chen algoritmo modelis VHDL sistemų aprašymo kalba. Vienamatis Chen DCT algoritmas realizuotas taip, kad jo sąsaja sutaptų su 3.7 lentelėje nurodytais signalais. Tai leidžia prijungti algoritmą prie valdančios aparatūros ir skaičiuoti dvimatę DCT transformaciją. Šios realizacijos blokinė schema pavaizduota 4.4 pav.




*4.4 pav. Chen DCT aparatinės realizacijos blokinė schema*

Iš 3.1 pav. Matyti, kad algoritmas susideda iš 4 etapų. DCT skaičiavimo procesui valdyti

naudojamas baigtinis Mūro automatas. Automatas pradeda darbą atėjus *start* signalui ir nuosekliai eina per visus keturis algoritmo skaičiavimo etapus, kurie automate atitinka būsenas. Paskutinėje būsenoje išduodamas *done* signalas. Šis signalas reiškia vienmačio DCT algoritmo darbo pabaigą. Automato būsenų perėjimo grafas pavaizduotas 4.5 pav.

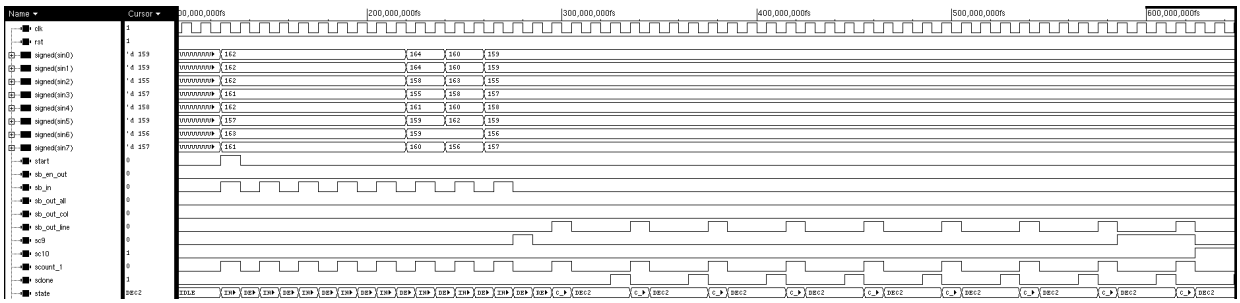



*4.5 pav. Algoritmo valdymo automato būsenų grafas*

### 4.2.1 Laikinės diagramos

Laikinėmis diagramomis 4.6 – 4.8 pav. Parodyta, kaip veikia valdančioji įranga, realizuodama 2-D DCT algoritmą.

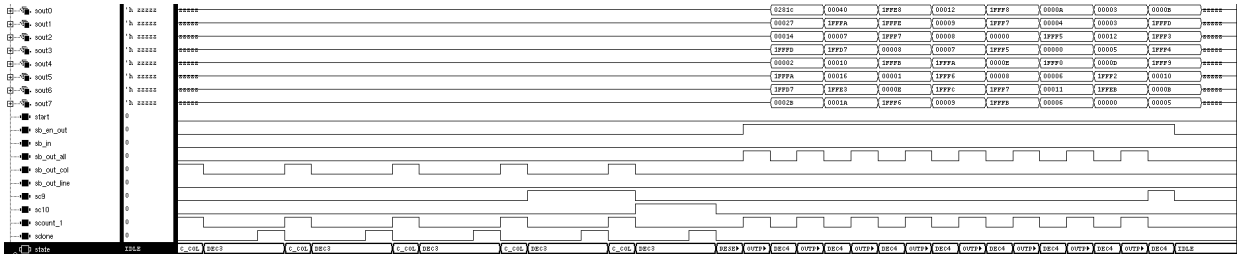
4.6 pav. pavaizduota įėjimų skaitymo operacija, valdantieji signalai bei valdančiojo automato veikimas skaitant duomenis. *Start* signalas žymi darbo pradžią. Po jo seka 8 *b_in* buferio valdymo signalai. Šių signalų kiekį nulemia skaitliukas. Signalai nurodo priimti duomenis iš išorės. Įėjimu per `in0 - in8` magistralę tuo metu paduodami duomenys. Automatui perėjus į sekančią būseną, sukuriami b_out_line signalai. Šie signalai žymi buferyje saugomos matricos eilučių padavimą DCT kodavimui. STATE kintamasis diagramoje vaizduoja valdančiojo automato esamą būseną



*4.6 pav. Įėjimų skaitymas ir valdančiojo automato veikimas*

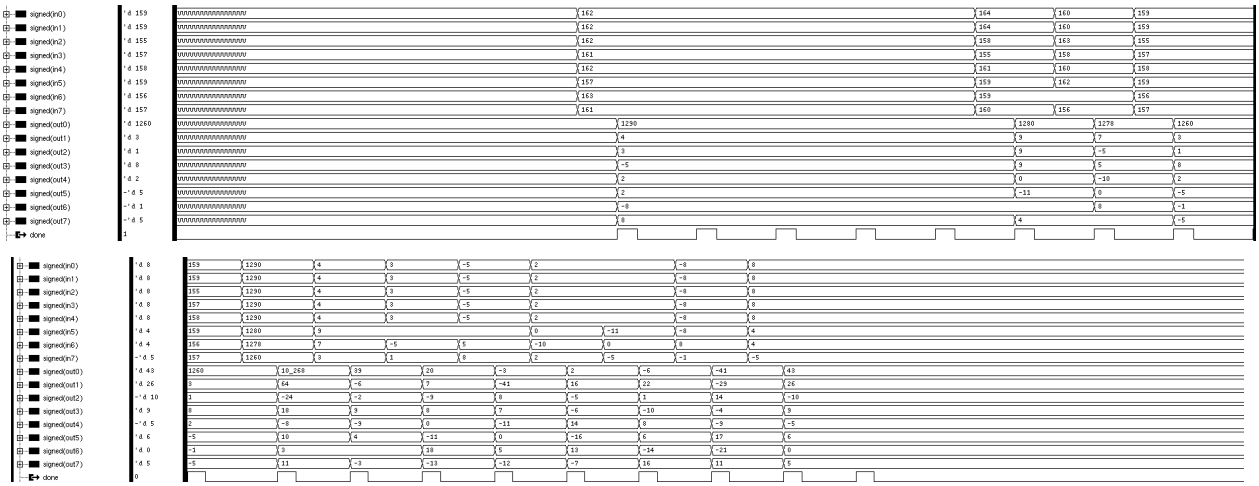
4.7 pav. pavaizduotas rezultatų rašymas, valdymo signalai ir valdančiojo automato

veikimas rašymo metu. Atlikus DCT ar IDCT kodavimą, automatas suformuoja 8 *b_out_all* buferio valdymo signalus. Buferis atiduoda suskaičiuotus rezultatus per `out0 - out8` magistralę. Tuo metu, kai rezultatai neišduodami, išėjimai yra atjungti nuo aparatūros (Z būsenoje). Tai pagerina šios magistralės suderinamumą su kita aparatūra, priimančia duomenis: kai duomenys nesiunčiami, magistralė gali būti naudojama kitais tikslais. Taipogi tai palengvina duomenų skaitymą testavimo įranga.



*4.7 pav. Rezultatų rašymas ir valdančiojo automato veikimas*

4.8 pav. vaizduojamas skaičiavimo bloko veikimas. Blokas paima duomenis iš `in0 – in7` įėjimų, atliekamas vienmatės DCT skaičiavimas ir į `out0 – out7` magistralę išduodami rezultatai. Kiekvieną kartą atlikus skaičiavimus ir gavus rezultatą išduodamas `done` signalas, žymintis darbo pabaigą. Kadangi dvimatei DCT atlikti reikia 16 vienmatės DCT operacijų, į DCT algoritmą iš buferio paduodamos 8 matricos eilutės ir 8 stulpeliai. Iš diagramos matyti, kaip skaičiavimai atliekami 16 kartų.



*4.8 pav. DCT bloko veikimas, atliekant dvimatę DCT operaciją*

### 4.2.2 Rezultatai

Modeliavimas atliktas VHDL RTL lygio ventilių aprašui (Register Transfer Level – RTL). Naudojant Cadence modeliavimo įrangą. Sintezė atlikta naudojant Synopsys programinę įrangą. Testavimui panaudoti tie patys paveikslėliai, kaip ir matematiniame modeliavime (4.2 pav.).

Atlikus kodavimą ir dekodavimą, išmatuojamas originalo ir rezultato vaizdų PSNR santykis.

Modeliavimo metu siekiama ištirti Chen DCT ir Chen IDCT algoritmų tikslumą. Tokiam tyrimui reikalingi kokybės neprarandantys DCT ir IDCT algoritmai. Norint apskaičiuoti Chen DCT algoritmo tikslumą, reikia užkodavus vaizdą Chen DCT algoritmu, dekodavimą atlikti kokybės neprarandančiu algoritmu. Taip rezultate gaunamos tik Chen DCT algoritmo įnešamos skaičiavimo klaidos. Tiriant Chen IDCT algoritmą, veiksmas atliekamas priešingai: užkoduojama kokybės neprarandančiu algoritmu, o atkoduojama Chen IDCT algoritmu. Taip rezultatuose matomos Chen IDCT algoritmo įnešamos klaidos. Tyrimui naudojami tikslūs matematiniai DCT bei IDCT algoritmai realizuojami MATLAB paketu, tiesiogiai įgyvendinant analitinėje darbo dalyje pateiktas (3) ir (4) formules.

Pirmiausia atliekamas vaizdo kodavimas Chen DCT algoritmu ir dekodavimas tiksliu IDCT algoritmu. Šis matavimas nustatyto Chen DCT algoritmo kodavimo kokybę. Rezultatai pateikti 4.2 lentelėje.

*4.2 lentelė. Chen DCT algoritmo kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 46.65 |
| Fruit.bmp | 47.67 |
| Cell.bmp | 35.3 |

Chen IDCT algoritmo kokybė tikrinama paveikslėlius užkoduojant tiksliu DCT algoritmu ir atkoduojant Chen IDCT algoritmu. Taip išmatuojamas Chen IDCT algoritmo kodavimo tikslumas. Rezultatai pateikiami 4.3 lentelėje.

*4.3 lentelė. Chen IDCT algoritmo kodavimo kokybės rezultatai*

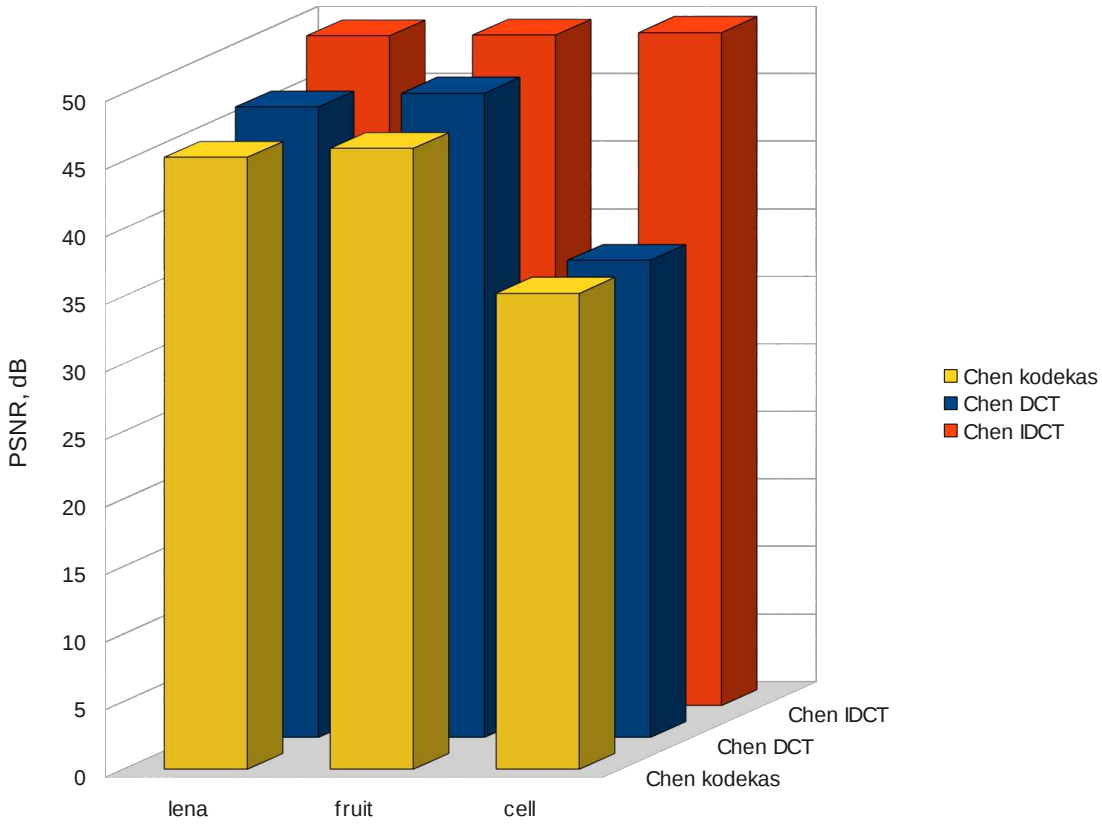
| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 49.57 |
| Fruit.bmp | 49.62 |
| Cell.bmp | 49.8 |

Sekančiame matavime matuojamas visos kodavimo – dekodavimo sistemos – kodeko, paremto Chen algoritmu tikslumas. Jo kokybė matuojama užkoduojant paveikslėlius Chen DCT algoritmu ir atkoduojant Chen IDCT algoritmu. Rezultatai pateikiami 4.4 lentelėje

*4.4 lentelė. Chen algoritmo kodeko kodavimo kokybės rezultatai*

| Paveikslėlis | PSNR, dB |
|---|---|
| Lena.bmp | 45.3 |
| Fruit.bmp | 45.96 |
| Cell.bmp | 35.24 |

Apibendrinti kokybės modeliavimo rezultatai grafiškai pateikti 4.9 pav.




*4.9 pav. Chen algoritmo kokybės rezultatai*

Iš grafiko matyti, kad klaidos kodavime atsiranda DCT algoritme. IDCT algoritmas įneša santykinai mažai kodavimo klaidų. Žmogaus rega pradeda pastebėti pasikeitimus ir kodavimo klaidas, jei PSNR santykis mažesnis nei 30 dB. Apie 50 dB PSNR santykį galima laikyti tiksliu vaizdo atkūrimu. Taigi viso kodeko kodavimo kokybė yra labai gera.

Siekiant palyginti lusto plotą, suvartojamą galią ir greitaveiką, Visos algoritmų realizacijos sintezuotos į FPGA matricą.

Chen DCT realizacija taip pat patikrinta FPGA matricoje. Sintezei ir RTL kodo testavimui panaudota Xilinx Spartan-3AN matrica. Dvimačio Chen DCT procesoriaus sintezės statistika

pavaizduota 4.5 lentelėje.

*4.5 lentelė  FPGA sintezės statistika Chen 2-D DCT*

| Naudojama logika | Išnaudota | Prieinama | Panauda |
|---|---|---|---|
| Trigeriai | 1905 | 22528 | 8.00 % |
| 4 įėjimų LUT blokai | 5341 | 22528 | 23.00 % |
| Užimtos dalys | 3005 | 11264 | 26.00 % |
| Įvesties/išvesties taškai | 275 | 375 | 73.00 % |
| 18X18 Daugintuvai | 30 | 32 | 93.00 % |

Iš lentelės matyti, kad algoritmui su papildoma aparatūra sunaudojama apie ketvirtis visų FPGA matricoje esančių blokų. Kadangi DCT algoritmo veikimo veikimui reikia dauginti 32 bitų ilgio skaičius, o matricoje esantys daugintuvai yra 18 bitų pločio, kai kurioms daugybos operacijoms sunaudojama po 2 daugintuvus. Taigi algoritmo realizacijai sunaudojami beveik visi matricoje esantys aparatiniai daugintuvai.

FPGA matricoje sintezuojamas ir dvimatis Chen IDCT procesorius. Sintezės statistika pavaizduota 4.6 lentelėje.

*4.6 lentelė. FPGA sintezės statistika Chen 2-D IDCT*

| Naudojama logika | Išnaudota | Prieinama | Panauda |
|---|---|---|---|
| Trigeriai | 1899 | 22528 | 8.00 % |
| 4 įėjimų LUT blokai | 5149 | 22528 | 22.00 % |
| Užimtos dalys | 2839 | 11264 | 25.00 % |
| Įvesties/išvesties taškai | 275 | 375 | 73.00 % |
| 18X18 Daugintuvai | 22 | 32 | 68.00 % |

Iš lentelės matyti, kad Chen IDCT algoritmas sunaudoja kiek mažiau aparatinių resursų. Pastebėtina, kad sunaudojama daug mažiau aparatinių daugintuvų.

## 4.3 Loeffler algoritmo matematinis modeliavimas

Loeffler algoritmo grafas, pavaizduotas 3.4 pav., taip pat gali būti tiesiogiai transformuojamas į matematinį ir aparatinį aprašą. Realizuojant šį algoritmą taip pat būtina išvengti sudėtingo slankaus kablelio skaičių naudojimo. Atliekamas papildomas tyrimas ir matematinis modeliavimas, kuriuo norima išsiaiškinti konstantų ilgio įtaką algoritmo kodavimo kokybei.

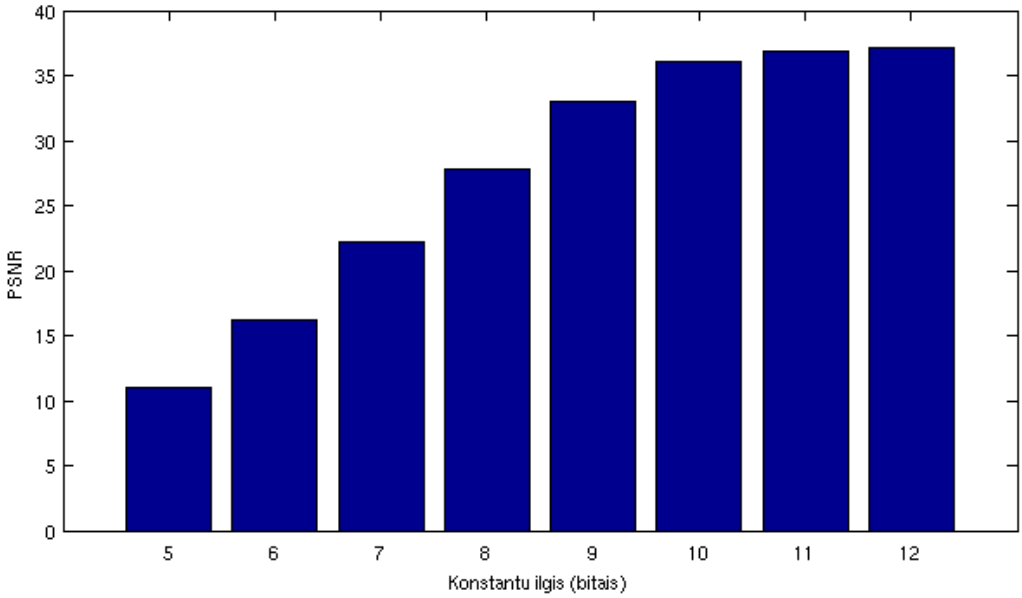
Loeffler algoritmui reikia šių konstantų:

- $\cos(\pi/16) \approx 0.9807$
- $\sin(\pi/16) \approx 0.1950$

- $\cos(3\pi/16) \approx 0.8314$
- $\sin(3\pi/16) \approx 0.5555$
- $\sqrt{2} * \cos(6\pi/16) \approx 0.5411$
- $\sqrt{2} * \sin(6\pi/16) \approx 1.3065$
- $\sqrt{2} \approx 1.4142$

Atliekant tyrimą, modelyje šios konstantos dauginamos iš $2^n$ (n = 5, 6, 7, 8, 9, 10, 11, 12). Tokiu būdu gaunamos sveikosios konstantų reikšmės su kuriomis testuojama algoritmo kodavimo kokybė. Daugybą iš tokių skaičių patogu atlikti aparatiškai, atliekant aritmetinį postūmį į kairę per *n* bitų.

Paveikslėlio kodavimui panaudojus skirtingus konstantų ilgius ir pritaikius atvirkštinę transformaciją galima apskaičiuoti sveikųjų skaičių konstantų naudojimo įtaką kokybei. Apskaičiuota PSNR priklausomybė nuo konstantų ilgio *n* pateikiama 4.10 pav.




*4.10 pav. PSNR priklausomybė nuo konstantų ilgio*

Atlikus šį testavimą, paveikslėlio kodavimui pasirinktas 10 bitų konstantų ilgis. Matematinis modeliavimas parodė, kad tokio ilgio konstantų naudojimas duoda patenkinamą kokybę su kiek įmanoma mažesniu konstantų ilgiu. Didinant konstantų ilgį kodavimo kokybė didėja nežymiai, o rezultatui reikia taikyti papildomas korekcijas, reikalaujančias matematinių skaičiavimų. Taigi modelyje visos konstantos dauginamos iš $2^{10}$ . Tokiu būdu gaunamos sveikosios konstantų reikšmės su kuriomis atliekamas kodavimas ir aparatinė algoritmo realizacija. Šis modelis panaudotas testinių paveikslėlių kodavimui taikant eilučių/stulpelių metodą. Loeffler algoritmo matematinio modelio kodavimo kokybės rezultatai pateikti 4.7 lentelėje.

*4.7 lentelė Loeffler algoritmo matematinio modeliavimo rezultatai*

| Paveikslėlis | PSNR, dB |
|---|---|
| Lena.bmp | 37.53 |
| Cell.bmp | 24.55 |
| Fruit.bmp | 35.82 |

Iš lentelės matyti, kad matematinis algoritmo modelis kokybe kiek nusileidžia Chen algoritmui. Cell.bmp paveikslėlio kodavimo kokybės rezultatai prasčiausi, nes paveikslėlis yra didžiausio detalumo, turi daug staigių šviesumo perėjimu. Tokius paveikslėlius DCT algoritmai koduoja prasčiau, nei paveikslėlius, turinčius daug vienodo šviesumo blokų ar tolygių šviesumo perėjimų. Toks kokybės sumažėjimas žmogaus akiai nėra matomas, arba matomas labai nežymiai.

## *4.4 Loeffler algoritmo aparatinė realizacija*

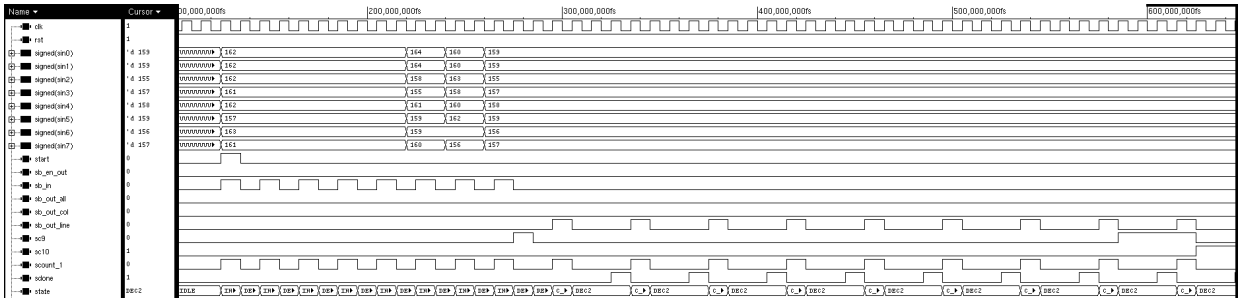
Po matematinio modeliavimo, pasirinkus 10 bitų konstantų ilgį, algoritmas aprašomas VHDL sistemų aprašymo kalba. Vienamatis Loeffler DCT algoritmas realizuojamas taip, kad jo sąsaja sutaptų su 3.7 lentelėje nurodytais signalais. Tai leidžia prijungti algoritmą prie valdančios aparatūros ir skaičiuoti dvimatę DCT transformaciją.

Iš 3.6 pav. pavaizduoto Loeffler algoritmo grafo matyti, kad algoritmas susideda iš 4 etapų. Jis realizuojamas tokiu pačiu metodu, kaip ir prieš tai aprašytasis Chen algoritmas, pavaizduotas 4.4 pav. Skaičiavimo procesui valdyti kaip ir Chen algoritme naudojamas baigtinis automatas, kurio būsenos atitinka algoritmo etapus. Automatas pradeda darbą atėjus *start* valdymo signalui ir nuosekliai eina per visas keturias būsenas. Paskutinėje būsenoje išduodamas *done* signalas, reiškiantis skaičiavimo proceso pabaigą. Šio automato perėjimų grafas sutampa su Chen algoritmo valdymui skirtu automatu, ir yra pavaizduotas 4.5 pav.

### 4.4.1 Laikinės diagramos

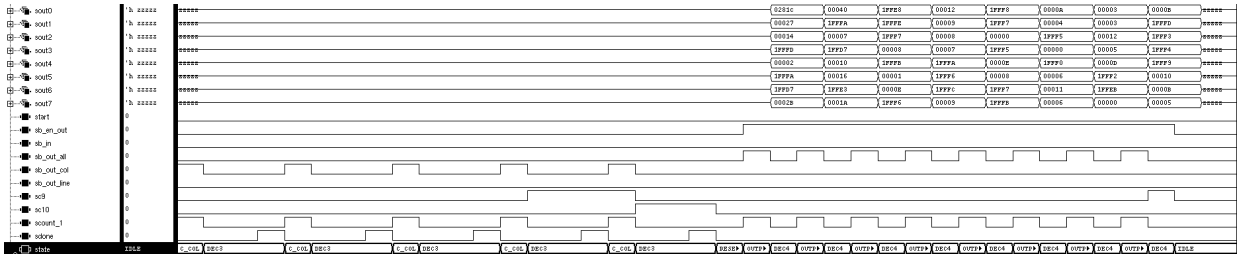
Laikinėmis diagramomis 4.11 – 4.13 pav. parodyta, kaip veikia valdančioji įranga, atlikdama 2-D Loeffler DCT algoritmą.

4.11 pav. pavaizduota įėjimų skaitymo operacija, valdantieji signalai bei valdančiojo automato veikimas skaitant duomenis. STATE kintamasis diagramoje vaizduoja valdančiojo automato esamą būseną. Išduodami 8 *b_in* signalai, kurie nurodo buferiui skaityti duomenis iš `in0 – in8` magistralės. Magistrale tuo metu paduodami įvedimo duomenys. Surašius šiuos duomenis į buferį, pradedama 16 kartų atlikinėti vienmatė DCT operacija.



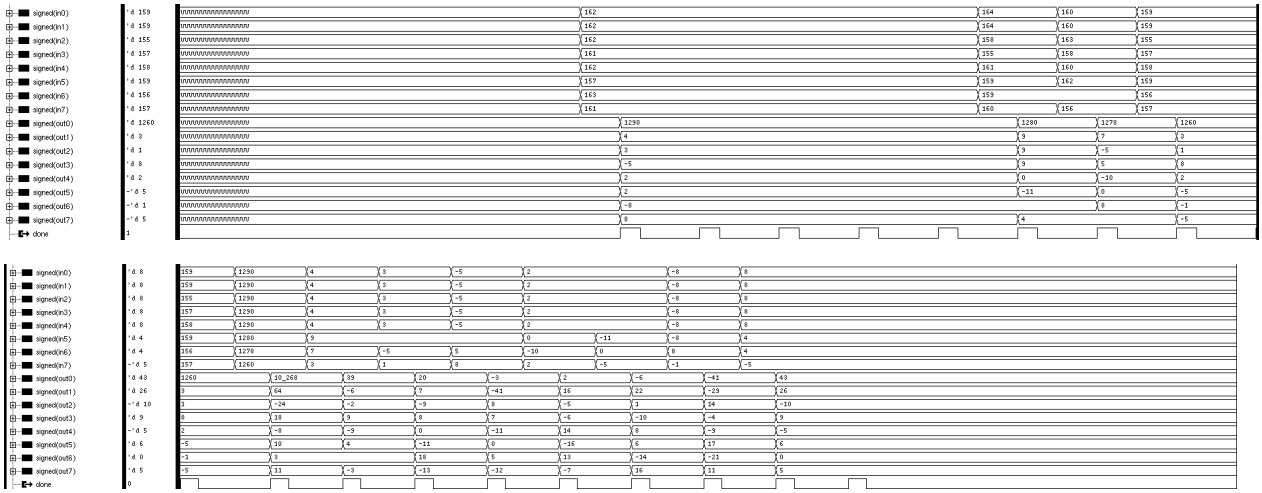
*4.11 pav. Įėjimų skaitymas ir valdančiojo automato veikimas*

4.12 pav. pavaizduotas rezultatų rašymas, valdymo signalai ir valdančiojo automato veikimas rašymo metu. Baigus dvimatės DCT skaičiavimą, automatas išduoda *b_out_all* valdymo signalus, kuriais buferiui nurodoma išduoti duomenis į išorę per `out0 – out8` magistralę. Kol neduodami šie signalai ir magistralėje nėra duomenų, ji būna atjungta nuo įrangos (Z būsenoje).



*4.12 pav. Rezultatų rašymas ir valdančiojo automato veikimas*

4.13 pav. vaizduojamas skaičiavimo bloko veikimas. Blokas paima duomenis iš `in0 – in7` įėjimų, atlieka vienmatės DCT skaičiavimą ir gražinęs rezultatą, išduoda *done* signalą, reiškiantį darbo pabaigą. Kadangi dvimatei DCT atlikti reikia 16 vienmatės DCT operacijų, iš diagramos matyti, kaip skaičiavimai atliekami 16 kartų. Duomenų padavimą, rezultatų saugojimą ir skaičiavimo pradžią valdo baigtinis valdymo automatas.



*4.13 pav. DCT bloko veikimas, atliekant dvimatę DCT operaciją*

### 4.4.2 Rezultatai

Aparatinio aprašo testavimui panaudoti tie patys paveikslėliai, kaip ir matematiniame modeliavime (4.2 pav.). Atlikus kodavimą ir dekodavimą, matuojamas originalo ir rezultato vaizdų PSNR santykis.

Kaip ir Chen algoritmo atveju, siekiama atskirai ištirti Loeffler DCT bei Loeffler IDCT algoritmo kodavimo kokybę, tuomet viso Loeffler algoritmu paremto kodeko kodavimo kokybę. Tam reikalingi tikslūs ir kokybės neprarandantys DCT bei IDCT algoritmai. Siekiant ištirti Loeffler DCT algoritmo tikslumą, atliekamas vaizdo kodavimas Loeffler DCT algoritmu ir dekodavimas tiksliu IDCT algoritmu. Tokiu būdu rezultatuose matomos klaidos, įnešamos tik Loeffler DCT algoritmo. Tai leidžia išmatuoti Loeffler DCT algoritmo kodavimo kokybę. Rezultatai pateikti 4.8 lentelėje.

*4.8 lentelė. Loeffler DCT algoritmo kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 35.78 |
| Fruit.bmp | 34.13 |
| Cell.bmp | 23.38 |

Iš lentelės matyti, kad dėl aparatinių daugintuvų apvalinimo gauti kiek prastesni rezultatai, nei matematinio modeliavimo metu.

Loeffler IDCT algoritmo kokybė tikrinama paveikslėlius užkoduojant tiksliu DCT algoritmu ir atkoduojant Loeffler IDCT algoritmu. Taip rezultate gaunamos klaidos, įnešamos tik Loeffler IDCT algoritmo, ir tokiu būdu galima išmatuoti šio algoritmo kodavimo tikslumą.

Rezultatai pateikiami 4.9 lentelėje.

*4.9 lentelė. Loeffler IDCT algoritmo kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 51.45 |
| Fruit.bmp | 51.56 |
| Cell.bmp | 51.68 |

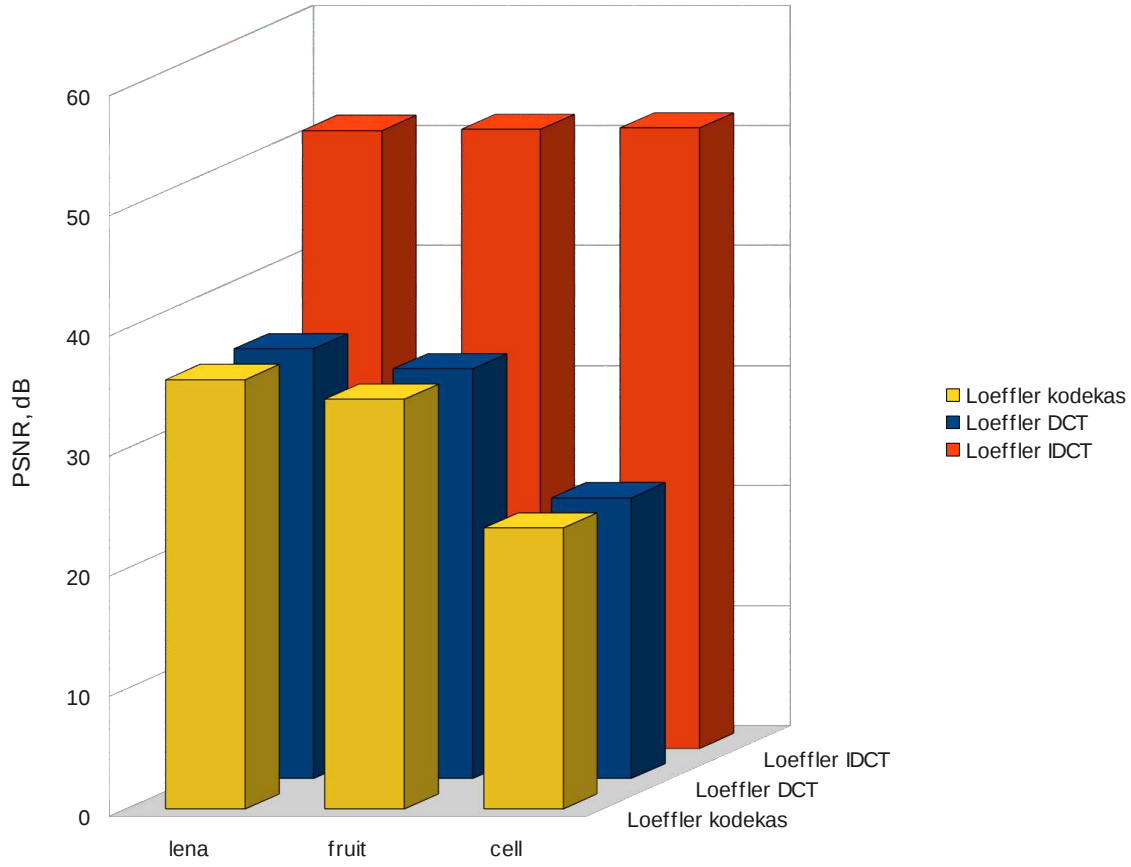
Iš lentelės matyti, kad Loeffler IDCT algoritmas veikia praktiškai neprarasdamas kokybės. Kokybės praradimas atsiranda tik dėl sveikų skaičių naudojimo.

Viso Loeffler kodeko veikimas testuojamas užkoduojant paveikslėlius Loeffler DCT algoritmu ir atkoduojant Loeffler IDCT algoritmu. Rezultatai pateikiami 4.10 lentelėje

*4.10 lentelė. Loeffler algoritmo kodeko kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR** |
|---|---|
| Lena.bmp | 35.74 |
| Fruit.bmp | 34.11 |
| Cell.bmp | 23.38 |

Apibendrinti kokybės modeliavimo rezultatai pateikti 4.14 pav.




*4.14 pav. Loeffler algoritmo kokybės rezultatai*

Kaip matyti iš grafiko, kodeko kodavimo kokybę nulemia Loeffler DCT algoritmo kodavimo kokybė. Toks kokybės pablogėjimas nėra aiškiai matomas plika akimi, tačiau cell.bmp paveikslėlio kodavimo rezultatai prasti, jau matomi vaizde atsiradę netikslumai.

Realizacija taip pat patikrinta FPGA matricoje. Sintezei ir RTL kodo testavimui panaudota Xilinx Spartan-3AN matrica. Dvimačio Loeffler DCT procesoriaus sintezės statistika pavaizduota 4.11 lentelėje.

*4.11 lentelė. FPGA sintezės statistika Loeffler 2-D DCT*

| **Naudojama logika** | **Išnaudota** | **Prieinama** | **Panauda** |
|---|---|---|---|
| Trigeriai | 1818 | 22528 | 8% |
| 4 įėjimų LUT blokai | 5561 | 22528 | 24% |
| Užimtos dalys | 3193 | 11264 | 28% |
| Įvesties/išvesties taškai | 275 | 375 | 73% |
| 18X18 Daugintuvai | 22 | 32 | 68% |

Kaip matyti iš lentelės, algoritmas su papildoma aparatūra užima apie ketvirtį visos FPGA

matricos blokų. Nors algoritmas turi tik 11 daugybų, sunaudojami 22 aparatiniai daugintuvai. Taip yra todėl, kad algoritme dauginami 32 bitų skaičiai. Aparatiniai daugintuvai yra 18 skilčių, todėl vienai daugybai panaudojami 2 daugintuvai.

FPGA matricoje sintezuojamas ir dvimatis Loeffler IDCT procesorius. Šio algoritmo ir papildomos įrangos sintezės statistika pavaizduota 4.12 lentelėje.

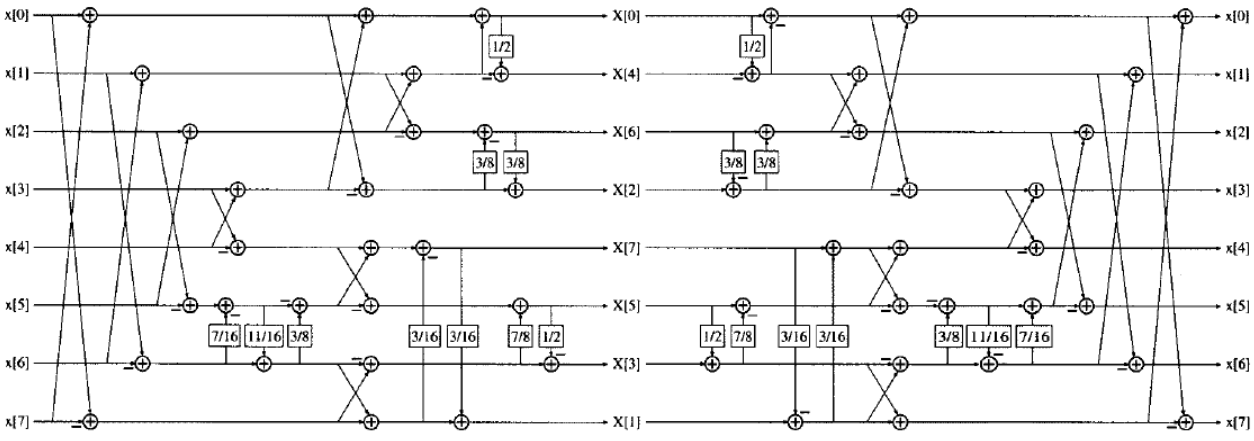
*4.12 lentelė. FPGA sintezės statistika Loeffler 2-D IDCT*

| Naudojama logika | Išnaudota | Prieinama | Panauda |
|---|---|---|---|
| Trigeriai | 1822 | 22528 | 8 % |
| 4 įėjimų LUT blokai | 5775 | 22528 | 25 % |
| Užimtos dalys | 3222 | 11264 | 28 % |
| Įvesties/išvesties taškai | 275 | 375 | 73 % |
| 18X18 Daugintuvai | 22 | 32 | 68 % |

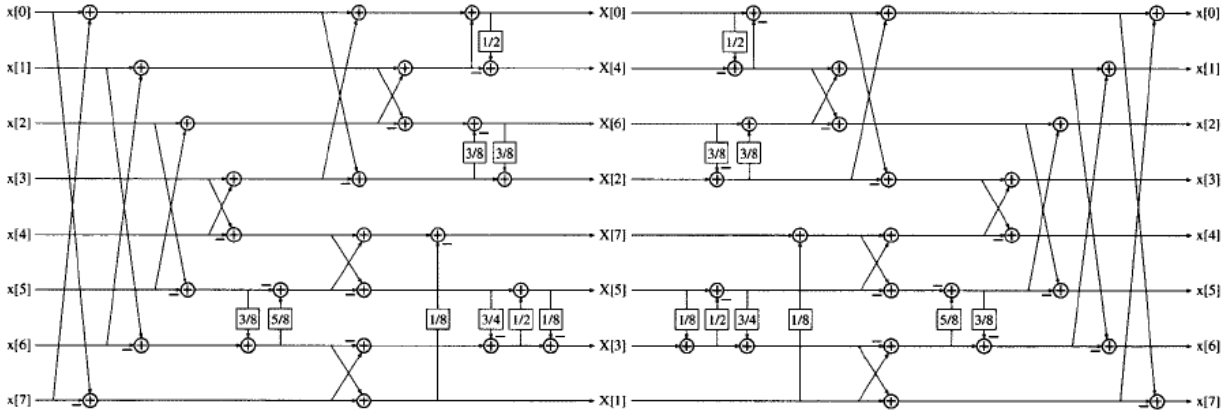
## 4.5 BinDCT algoritmo matematinis modeliavimas

BinDCT aprašomas kaip metodas, pakeičiantis daugybos veiksmus pakėlimo žingsniais. Tikslingiausia šiam metodui naudoti Loeffler algoritmo signalų perėjimo grafą, nes šis turi mažiausiai matematinių operacijų. Algoritmo autoriai taip pat siūlo šį grafą kaip algoritmo pagrindą [10]. Pritaikius 3.8 pav. pavaizduotam Loeffler algoritmo grafui 3.2 lentelės *p* ir *u* reikšmės, gaunamos 3 galimos grafo kombinacijos:

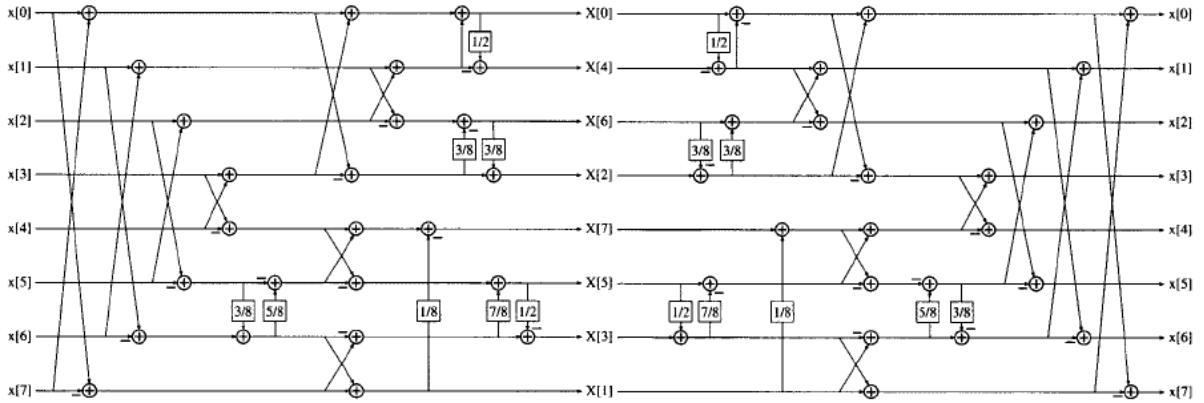
A




B




C




*4.15 pav. Algoritmo versijos (A, B, C). Kairėje – DCT algoritmas, dešinėje – IDCT algoritmas*

Siekiant išsiaiškinti, kuri iš šių versijų tinkamiausia aparatinei realizacijai, atliekamas papildomas matematinis modeliavimas, kuriame palyginama algoritmų kodavimo kokybė ir jiems atlikti reikalingų matematinių operacijų skaičius. Atlikus šį matematinį modeliavimą su visomis trimis algoritmų versijomis (4.15 pav.), apskaičiuotas kiekvienos versijos kodavimo PSNR santykis lena.bmp paveikslėliui. Rezultatai, kartu su algoritmų matematinių operacijų skaičiaus palyginimu, pateiktas 4.13 lentelėje.

*4.13 lentelė. BinDCT versijų matematinių operacijų skaičius*

| **Versija** | **Sudėtys** | **Postūmiai** | **PSNR, dB** |
|---|---|---|---|
| A | 36 | 19 | 49.47 |
| B | 31 | 14 | 49.47 |
| C | 30 | 13 | 48.13 |

Aparatinei realizacijai pasirinktas C algoritmo variantas. Nors šio varianto kokybė kiek prastesnė, ji labai nežymiai skiriasi nuo kitų variantų kokybės, tuo tarpu ši versija turi mažiausią kiekį sudėties ir poslinkio operacijų, todėl turėtų turėti didžiausią greitaveiką, būti taupiausia

galiai bei lusto plotui.

## *4.6 BinDCT algoritmo aparatinė realizacija*

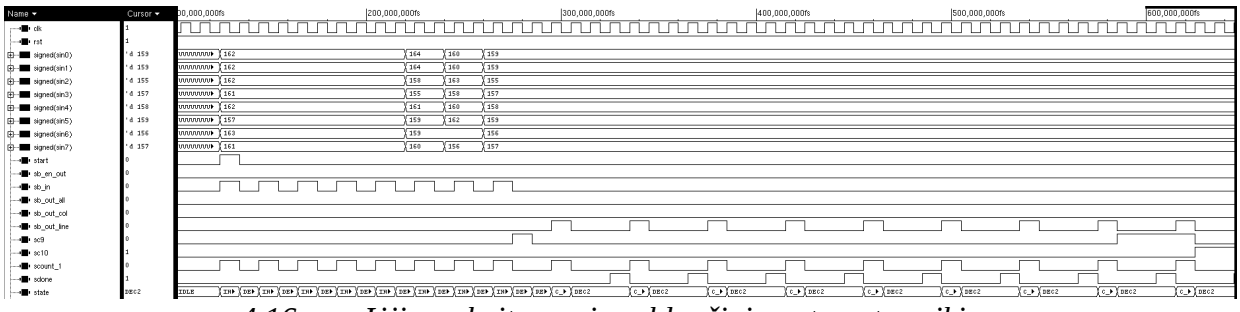
Po matematinio modeliavimo BinDCT algoritmui pritaikius Loeffler algoritmo signalų perėjimo grafą ir po matematinio modeliavimo pasirinkus C algoritmo variantą, šis algoritmas aprašomas VHDL sistemų aprašymo kalba. Vienamatis BinDCT algoritmas realizuotas taip, kad jo sąsaja sutaptų su 3.7 lentelėje nurodytais signalais. Tai leidžia prijungti algoritmą prie valdančios aparatūros , pavaizduotos 3.11 pav. ir skaičiuoti dvimatę DCT transformaciją. Schema realizuojama tokiu pat metodu, kaip Chen ir Loeffler algoritmai, aprašyti ankščiau. Algoritmo blokinė schema sutampa su Chen algoritmo, kuri pavaizduota 4.4 pav.

Iš 4.15 pav. matyti, kad šis algoritmas susideda iš 4 etapų, kurie realizuoti kaip valdančiojo baigtinio automato būsenos. Automatas pradeda darbą atėjus *start* signalui ir nuosekliai eina per visas keturias būsenas. Kiekvienoje būsenoje skaičiavimų blokas atlieka etapui reikalingus skaičiavimus. Paskutinėje būsenoje išduodamas *done* signalas. Automato perėjimo grafas pavaizduotas 4.5 pav.

### 4.6.1 Laikinės diagramos

Laikinėmis diagramomis 4.16 - 4.18 pav. parodyta, kaip veikia valdančioji įranga, atlikdama 2-D BinDCT algoritmą.

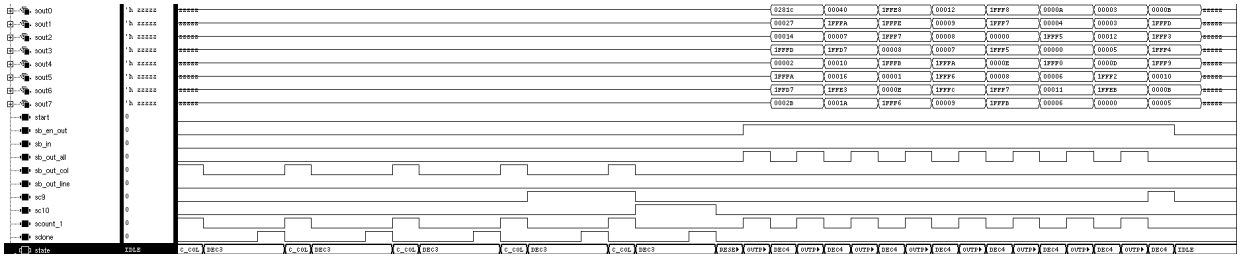
4.16 pav. pavaizduota įėjimų skaitymo operacija, valdantieji signalai bei valdančiojo automato veikimas skaitant duomenis. STATE kintamasis diagramoje vaizduoja valdančiojo automato esamą būseną. Išduodami 8 *b_in* signalai, kurie nurodo buferiui skaityti duomenis iš `in0 – in8` magistralės. Magistrale tuo metu paduodami įvedimo duomenys. Surašius šiuos duomenis į buferį, pradedama 16 kartų atlikinėti vienmatė DCT operacija.



*4.16 pav. Įėjimų skaitymas ir valdančiojo automato veikimas*

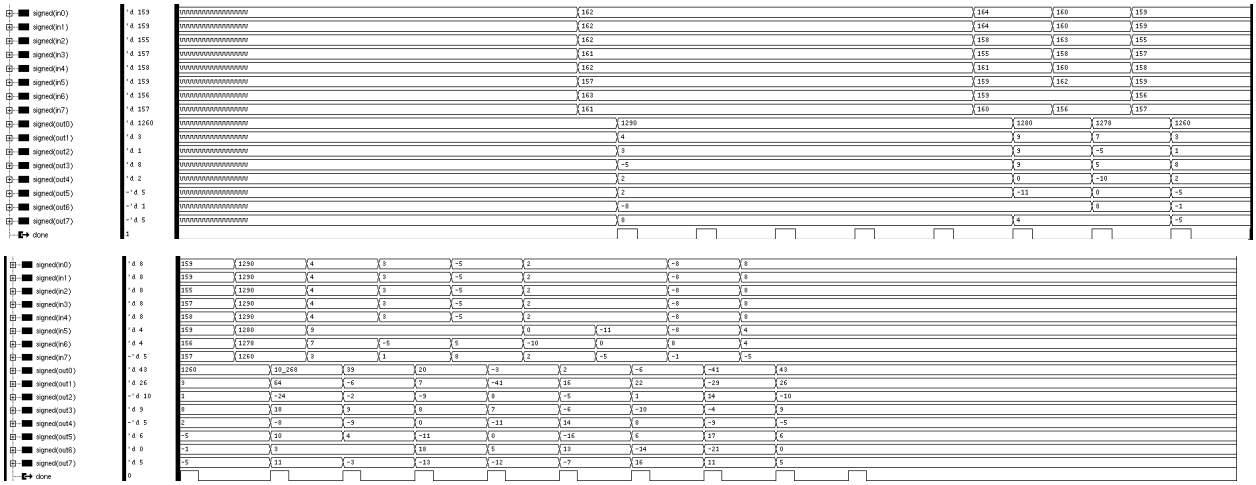
4.17 pav. pavaizduotas rezultatų rašymas, valdymo signalai ir valdančiojo automato veikimas rašymo metu. Baigus dvimatės DCT skaičiavimą, automatas išduoda *b_out_all* valdymo signalus, kuriais buferiui nurodoma išduoti duomenis į išorę per `out0 – out8`

magistralę. Kol neduodami šie signalai ir magistralėje nėra duomenų, ji būna atjungta nuo įrangos (Z būsenoje).



*4.17 pav. Rezultatų rašymas ir valdančiojo automato veikimas*

4.18 pav. vaizduojamas skaičiavimo bloko veikimas. Blokas paima duomenis iš `in0 – in7` įėjimų, atlieka vienmatės DCT skaičiavimą ir gražinęs rezultatą, išduoda *done* signalą, reiškiantį darbo pabaigą. Kadangi dvimatei DCT atlikti reikia 16 vienmatės DCT operacijų, iš diagramos matyti, kaip skaičiavimai atliekami 16 kartų. Duomenų padavimą, rezultatų saugojimą ir skaičiavimo pradžią valdo baigtinis valdymo automatas.



*4.18 pav. DCT bloko veikimas, atliekant dvimatę DCT operaciją*

### 4.6.2 Rezultatai

Modeliuojant aparatūros aprašą siekiama ištirti BinDCT ir BinIDCT algoritmų tikslumą. Tam naudojami tikslūs matematiniai DCT bei IDCT algoritmai. Norint gauti BinDCT algoritmo kodavimo kokybės rezultatus, kodavimas atliekamas BinDCT algoritmu, o atkodavimas - tiksliu IDCT algoritmu. Rezutate gaunama tik BinDCT algoritmo įnešamos kodavimo klaidos. Apskaičiavus rezutato ir pradinio vaizdo PSNR santykį, galima išmatuoti BinDCT algoritmo kodavimo kokybę. Šio matavimo rezultatai pateikti 4.14 lentelėje.

*4.14 lentelė. BinDCT algoritmo kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR, dB** |
|---|---|
| Lena.bmp | 31.63 |
| Fruit.bmp | 30.58 |
| Cell.bmp | 19.92 |

BinIDCT algoritmo kokybė tikrinama atvirkčiai nei BinDCT - paveikslėlius užkoduojant tiksliu DCT algoritmu ir atkoduojant BinIDCT algoritmu. Taip rezultate matomos klaidos, įnešamos tik BinIDCT algoritmo. Tokiu būdu galima išmatuoti BinIDCT algoritmo kodavimo tikslumą. Šio matavimo rezultatai pateikiami 4.15 lentelėje.

*4.15 lentelė .BinIDCT algoritmo kodavimo kokybės rezultatai*

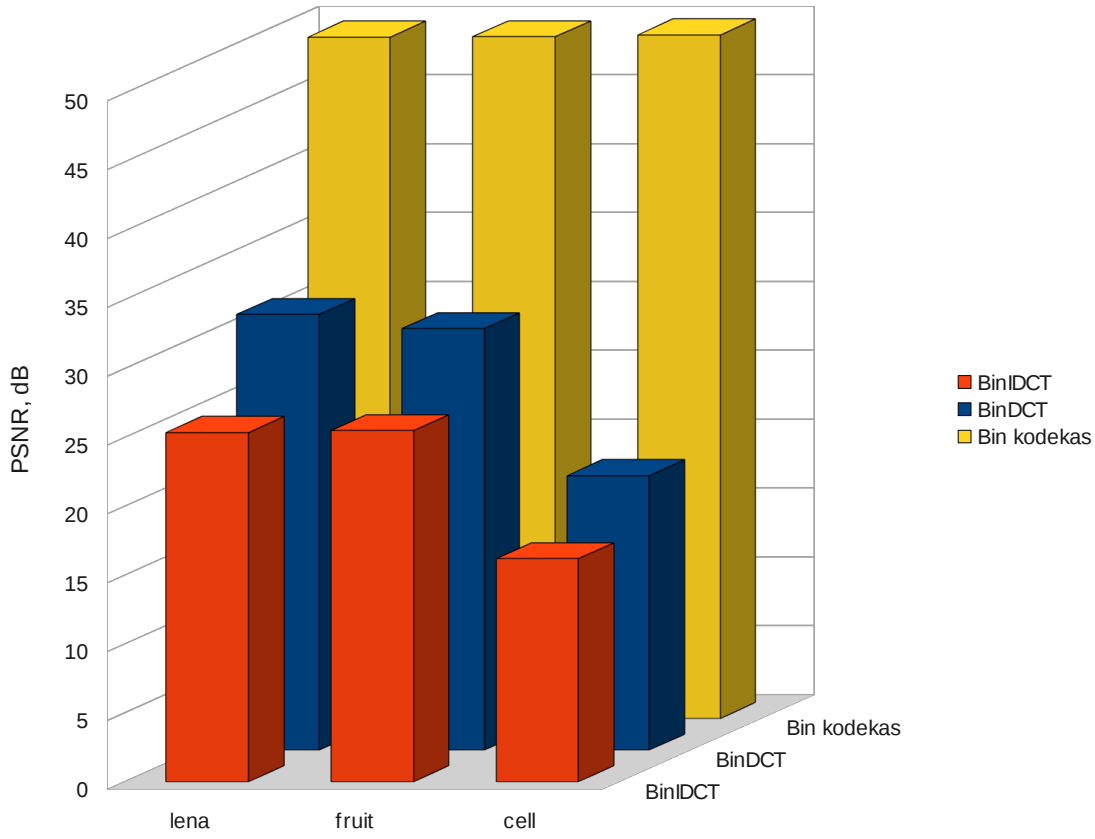
| **Paveikslėlis** | **PSNR,dB** |
|---|---|
| Lena.bmp | 25.36 |
| Fruit.bmp | 25.48 |
| Cell.bmp | 16.18 |

Atlikus atskirų algoritmų kokybės matavimą, siekiama ištirti viso kodeko, paremto Bin algoritmu veikimas. Testavimas atliekamas užkoduojant paveikslėlius BinDCT algoritmu ir atkoduojant BinIDCT algoritmu. BinDCT kodeko tyrimo rezultatai pateikiami 4.16 lentelėje.

*4.16 lentelė. Bin algoritmo kodeko kodavimo kokybės rezultatai*

| **Paveikslėlis** | **PSNR** |
|---|---|
| Lena.bmp | 49.47 |
| Fruit.bmp | 49.5 |
| Cell.bmp | 49.63 |

Apibendrinti kokybės modeliavimo rezultatai pateikti 4.19 pav.




*4.19 pav. Bin algoritmo kokybės rezultatai*

Nors šis algoritmas netinkamas kokybės neprarandančiam kodavimui, skaičiavimo tikslumu jis nenusileidžia kitiems greitiesiems DCT algoritmams, naudojamiems daugumoje šiandieninių įrenginių. Kaip matyti, algoritmas praranda nemažai vaizdo kokybės. Kodavimo metu atsiranda triukšmai, plika akimi matomi dekoduotuose vaizduose.

Pastebėtina, kad algoritmo kodeko veikimas pasižymi itin gera kodavimo kokybe. Tai nutinka, kadangi algoritmo autorių sukurtoje pakėlimo žingsnių schemoje ir pakėlimo konstantose gaunami koeficientai. Norint kad ši transformacija atitiktų DCT formulę, šie koeficientai turi būti kompensuojami. Būtent dėl šių koeficientų atskirai skaičiuojama BinDCT bei BinIDCT algoritmų kokybė nėra labai gera. Naudojant šį algoritmą kaip kodeką, t. y. tuo pačiu algoritmu atliekant vaizdo kodavimą ir dekodavimą, koeficientai kompensuojami, ir algoritmo kokybės rezultatai tampa labai geri. Galima teigti, kad BinDCT algoritmu paremtas kodekas vaizdo kokybės kodavimo metu nepraranda.

Realizacija taip pat patikrinta FPGA matricoje. Sintezei ir RTL kodo testavimui

panaudota Xilinx Spartan-3AN matrica. Dvimačio BinDCT procesoriaus sintezės statistika pavaizduota 4.17 lentelėje.

*4.17 lentelė. FPGA sintezės statistika 2-D BinDCT*

| Naudojama logika | Išnaudota | Prieinama | Panauda |
|---|---|---|---|
| Trigeriai | 1917 | 22528 | 8 % |
| 4 įėjimų LUT blokai | 4905 | 22528 | 21 % |
| Užimtos dalys | 2757 | 11264 | 24 % |
| Įvesties/išvesties taškai | 275 | 375 | 73 % |
| 18X18 Daugintuvai | 0 | 32 | 0 % |

Kaip matyti iš FPGA sintezės statistikos, algoritmas užima apie penktadalį matricos blokų. Aparatiniai daugintuvai šiam algoritmui nenaudojami.

Dvimačio BinIDCT procesoriaus sintezės statistika pavaizduota 4.18 lentelėje.

*4.18 lentelė. FPGA sintezės statistika 2-D BinIDCT*

| Naudojama logika | Išnaudota | Prieinama | Panauda |
|---|---|---|---|
| Trigeriai | 1885 | 22528 | 8 % |
| 4 įėjimų LUT blokai | 5066 | 22528 | 22 % |
| Užimtos dalys | 2809 | 11264 | 24 % |
| Įvesties/išvesties taškai | 275 | 375 | 73 % |
| 18X18 Daugintuvai | 0 | 32 | 0 % |

Kaip matyti iš BinIDCT sintezės rezultatų, BinIDCT algoritmas identiškas BinDCT algoritmui. Aparatiniai daugintuvai jam taip pat nenaudojami.

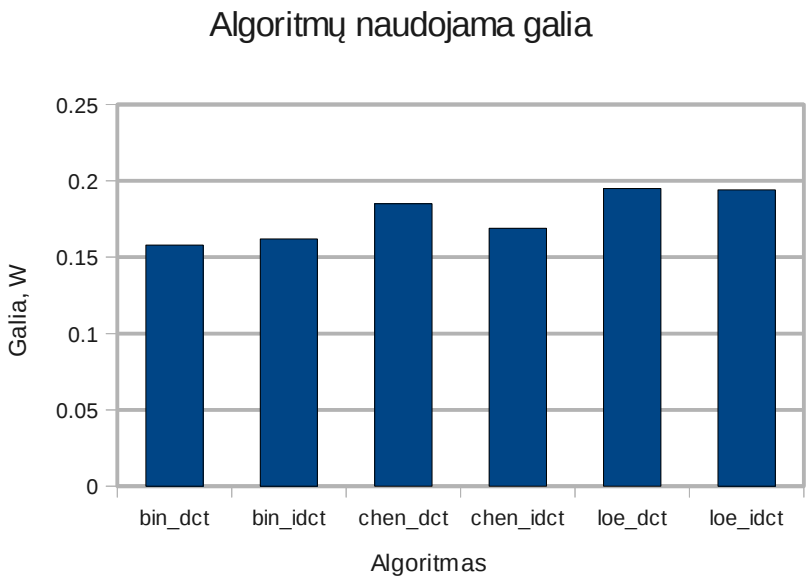
## 4.7 FPGA galios ir greitaveikos palyginimas

Projektuojant mobiliems įrenginiams skirtas skaičiavimo sistemas labai svarbus veiksnys yra galios suvartojimas. Xilinx kompanija pateikia programinę įrangą, kuri gali tiksliai apskaičiuoti sintezuotos schemos naudojamą galią. FPGA naudojama galia skirstoma į statinę ir dinaminę. Statinė galia, tai energijos suvartojimas, nepriklausantis nuo schemos veikimo ir taktinio dažnio. Dinaminė galia atsiranda, kai sistemą sudarantys tranzistoriai pereina iš vienos būsenos į kitą. Kuo didesnis sistemos taktinis dažnis, tuo daugiau dinaminės galios suvartojama. 4.19 lentelėje pateikiamas tiriamų algoritmų galios suvartojimas, kai jų taktinis dažnis 50 MHz.

*4.19 lentelė. FPGA galios suvartojimas*

| **Algoritmas** | **Galia, W** |
|---|---|
| BinDCT | 0.158 |
| BinIDCT | 0.162 |
| Chen DCT | 0.185 |
| Chen IDCT | 0.169 |
| Loeffler DCT | 0.195 |
| Loeffler IDCT | 0.194 |

Grafinė šių duomenų išraiška pateikta 4.20 pav.




*4.20 pav. Skirtingų algoritmų galios suvartojimas*

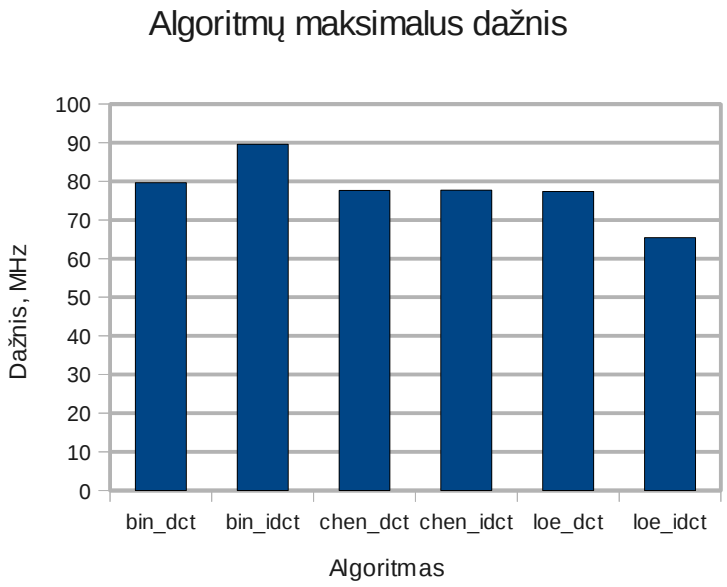
Nors algoritmų vartojama galia skiriasi nežymiai, tačiau Bin DCT/IDCT algoritmai sunaudoja mažiausiai galios. Toks rezultatas yra logiškas, kadangi algoritmas nenaudoja aparatinių daugintuvų ir reikalauja mažiausiai aparatinės įrangos. kaip tik šis algoritmas yra geriausiai pritaikytas aparatiniai realizacijai.

Kitas labai svarbus šių algoritmų parametras yra greitaveika. Kuo greičiau gali veikti algoritmas, tuo daugiau duomenų juo galima apdoroti per tam tikrą laiko tarpą. Xilinx kompanija pateikia programinius įrankius, kurie gali įvertinti signalo kelią, komponentų vėlinimą ir kitus parametrus ir taip nustatyti maksimalų galimą taktinį dažnį. 4.20 lentelėje pateikti greitaveikos duomenys, gauti naudojant šiuos įrankius.

*4.20 lentelė. Algoritmų greitaveika*

| **Algoritmas** | **Dažnis, MHz** | **Periodas, ns** |
|---|---|---|
| BinDCT | 79.6 | 12.56 |
| BinIDCT | 89.5 | 11.16 |
| Chen DCT | 77.6 | 12.88 |
| Chen IDCT | 77.7 | 12.86 |
| Loeffler DCT | 77.3 | 12.93 |
| Loeffler IDCT | 65.4 | 15.28 |

4.20 lentelėje Pateiktas algoritmų maksimalus taktinis dažnis atvaizduotas grafiškai 4.21 pav. Duomenys rodo, kad lėčiausiai veikia Loeffler IDCT algoritmas. Tokį rezultatą galėjo lemti sudėtinga kombinacinė logika, naudojama duomenų srautų valdymui.




*4.21 pav. Algoritmų maksimalus taktinis dažnis*

Lėčiausio – Loeffler IDCT algoritmo maksimalus taktinis dažnis Xilinx Spartan-3AN FPGA matricoje yra 65MHz. Tokia greitaveika leidžia apdoroti apie 350 512x512 vaizdo taškų turinčių nespalvotų vaizdų per sekundę. Tai atitinka apie 15 HD kokybės (1920x1080) spalvotus video kadrus per sekunde. Likę algoritmai atitinkamai spartesni.

## *4.8 Tyrimo dalies išvados*

Tyrimo metu sukurti visų trijų pasirinktų algoritmų matematiniai ir VHDL modeliai. Gauti matematinio modeliavimo ir aparatūros aprašo modeliavimo kokybės rezultatai. Matematinio modeliavimo rezultatai parodė algoritmų tikslumo priklausomybę nuo naudojamų

konstantų ilgio bei algoritmų konfigūracijos. Aparatiniu modeliavimu išsiaiškinta, kiekvieno algoritmo kokybės rezultatai bei šiais algoritmais paremtų kodekų kokybės rezultatai. Aparatinio ir matematinio modelių kodavimo kokybės skirtumai yra maži, todėl modelių aprašai yra tinkami tiek FPGA, tiek ASIC gamybai. Atlikus sintezę į FPGA matricą, gauti algoritmų ploto, galios bei greitaveikos rezultatai.

# 5 Išvados

DCT ir IDCT transformacija labai svarbi skaitmeninės informacijos kodavime. Jos pritaikymo sritys siekia vaizdo, garso apdorojimą, signalų filtravimą, kalbos atpažinimo ir kitas sritis. Aparatinė šio algoritmo realizacija padeda sutaupyti galios ir sutrumpinti skaičiavimo laiką. Tiesioginis algoritmo taikymas aparatūrai netinka, todėl naudojami greitieji DCT/IDCT algoritmai, kurie sumažina matematinių operacijų skaičių.

Informacijos paieška parodė, kad yra publikuota daug greitųjų DCT/IDCT skaičiavimo algoritmų, kurie tinkami aparatinei realizacijai.

Tyrimo metu buvo sukurti matematiniai ir aparatiniai Loeffler, Chen ir BinDCT algoritmo modeliai. Taip pat atlikti papildomi modeliavimai siekiant išsiaiškinti sveikųjų skaičių bei algoritmo konfigūracijų įtaką kodavimo kokybei Šių modelių tyrimas parodė, kad visi algoritmai tinkami aparatinei DCT bei IDCT realizacijai. Algoritmų kodavimo tikslumas matuotas, skaičiuojant originalaus vaizdo ir koduoto vaizdo PSNR santykį. Nustatyta, kad skaičiavimams naudojant sveikuosius skaičius atsiranda kodavimo klaidų, tačiau šis klaidų kiekis pernelyg negadina vaizdinės informacijos, todėl yra priimtinas. Žmogaus akis nepastebi atsiradusių klaidų, kai algoritmas taikomas vaizdinei informacijai koduoti. Šios klaidas būtų dar mažiau pastebimos, jei algoritmai naudojami kokybę prarandančiam vaizdo suspaudimui (pvz. JPEG formatu).

Dvimatėms DCT transformacijoms realizuoti panaudotas eilučių/stulpelių metodas, kuriam atlikti aparatinėmis priemonėmis sukurta papildoma valdymo įranga. Svarbiausias reikalavimas šiai įrangai - pakartotinis visų jos komponentų panaudojimas. Darbo metu realizuoti bendriniai sistemos komponentai, kurie tinkami visiems tyrimo metu naudotiems algoritmams. Realizuota bendrinė skaičiavimo algoritmo sąsaja, leidžianti komunikavimą tarp algoritmo ir papildomos valdymo įrangos.

Tiek algoritmų, tiek valdymo įrangos VHDL aprašas nepriklauso nuo konkrečios technologinės bibliotekos, todėl gali būti taikomas tiek FPGA, tiek ASIC gamybai.

Siekiant išsiaiškinti algoritmų galią ir greitaveika, tyrimo metu aprašas buvo sintezuojamas į Xilinx Spartan-3AN FPGA matricą. Kaip ir tikėtasi, Bin DCT/IDCT algoritmai pasižymi geriausiomis techninėmis savybėmis, nes yra specialiai pritaikyti aparatinei realizacijai. Deja šių algoritmų rezultatas nėra visiškai tiksli DCT transformacija. Nepaisant to, Bin algoritmai pasižymi tokiu pačiu energijos suspaudimo lygiu kaip ir tikslioji transformacija, taigi yra tinkami įvairių signalų filtrų realizavimui.

Loeffler algoritmas, kurio modifikacija naudojama JPEG standarte, pasižymi mažiausia greitaveika ir didžiausia suvartojama galia, tačiau šios savybės nedaug skiriasi nuo Chen algoritmo. Abiejų šių algoritmų privalumas - jų duodami rezultatai atitinka tiksliuosius algoritmus, nepaisant to, kad atsiranda nedidelės klaidos dėl sveikųjų skaičių aritmetikos naudojimo.

Tyrimas parodė, kad diskrečioji kosinusinė transformacija labai svarbi šiuolaikiniame informacijos apdorojime, todėl neišvengiamai reikalingi ir kuriami procesoriai galintys skaičiuoti šią transformaciją aparatiškai. Darbo metu gauti rezultatai gali padėti išsirinkti tinkamiausią algoritmą konkrečiai aparatinei realizacijai.

# Literatūra


1. G. Yeap, „Practical low power VLSI design“,. Kluwer Academic Publishers 1998.

2. Rao and Yip, „Discrete cosine transformation. algorithms, advantages, applications“, 1990.

3. P. Master, „ACM augments DSPs for MPEG-4“, EE Times, [interaktyvus] Nov. 6, 2000. [Žiūrėta 2011 gegužės 10 d] Prieiga per internetą: <http://www.eet.com/story/OEG20001106S0037>.

4. G. Strang, „The Discrete Cosine Transform“, SIAM Review, Volume 41, Number 1, pp. 135-147, 1999.

5. R. J. Clark, „Transform Coding of Images“, New York: Academic Press, 1985.

6. N. Ahmed, T. Natarajan, K. R. Rao, „Discrete Cosine Transform“, IEEE Trans. Computers, 90-93, 1974.

7. JPEG. „Joint Photographic Experts Group“ JPEG Homepage [interaktyvus]. 2011 [Žiūrėta 2011 gegužės 10 d] Prieiga per internetą: <www.jpeg.org>.

8. B. William, Pennebaker, Joan L. Mitchell, „JPEG still image data compression standard (3rd ed.)“. Springer. p. 291. 1993.

9. I. Bauermann, E. Steinbacj, „Further Lossless Compression of JPEG Images“, Proc. of Picture Coding Symposium (PCS 2004), San Francisco, USA, December 15–17, 2004.

10. M. J. Narasimha, A. M. Peterson, „On the computation of the discrete cosine transform“, IEEE Trans. Commun. 26 (6), p. 934–936, 1978.

11. N. August, „On the Low Power Design of DCT and IDCT for Low Bit Rate Video Codecs“, Blacksburg, VA, 2001.

12. E. Scopa, A. Leone, R. Guerrieri, G. Baccarani, „A 2-D DCT low-power architecture for H.261 Coders“, 1995 International Conference on Acoustics, Speech, and Signal Processing, vol. 5, pp. 3271-3274, May 1995.

13. E. Feig, S. Winograd, „Fast algorithms for the discrete cosine transform“, IEEE Transactions on Signal Processing, vol. 40, pp. 2174-2193, Sep. 1992.

14. Y. P. Lee, T. H. Chen, L. G. Chen, M. J. Chen, C. W. Ku, „A cost-effective architecture for 8x8 two-dimensional DCT/IDCT using direct method“, IEEE Transactions on Circuits and Systems for Video Technology, vol. 7, pp. 459-467, Jul. 1997.

15. L. G. Chen, J. Y. Jiu, H. C. Chang, Y. P. Lee, C. W. Ku, „A low power 2-D DCT chip design using direct 2-D algorithm“, Proceedings of the ASP-DAC '98. Asia and South Pacific Design Automation Conference 1998, pp. 145-150, Feb. 1998.

16. K. W. Shin, H. W. Jeon, Y. S. Kang, „An efficient VLSI implementation of vector-radix 2-D DCT using mesh-connected 2-D array“, 1994 IEEE International Symposium on Circuits and Systems, 1994, vol. 4, pp 47-50, Jun. 1994.

17. Y. M. Huang, J. L. Wu, C. T. Hsu, „A refined fast 2-D discrete cosine transform algorithm with regular butterfly structure“, IEEE Transactions on Consumer Electronics, vol. 44, pp. 376-383, May 1998.

18. T. S. Chang, C. S. Kung, C. W. Jen, „A simple processor core design for DCT/IDCT“, IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, pp. 439-447, Apr. 2000.

19. Liu, „Vector-radix DCT/IDCT implementation for MPEG DSP“, Proceedings of 3rd International Conference on Signal Processing 1996, vol. 1, pp. 641-644, Oct. 1996.

20. V. Srinivasan, K. J. R. Liu, „VLSI design of high-speed time-recursive 2-D DCT/IDCT processor for video applications“, IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, pp. 87-96, Feb. 1996.

21. S. F. Hsiao, W. R. Shiue, „New hardware-efficient algorithm and architecture for the computation of 2-D DCT on a linear systolic array“, 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, Proceedings, vol. 6, pp. 3517-3520 Mar. 1999.

22. Lengwehasatit, Ortega, „Distortion/decoding time tradeoffs in software DCT-based image coding”, 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 4, pp. 2725-2728, Apr. 1997.

23. Girod, Stuhlmuller, „A content-dependant fast DCT for low bit-rate video coding”, Proceedings of the 1998 International Conference on Image Processing“, 1998, vol. 3, pp. 80-84, Oct. 1998.

24. Vytautas Štuikys, Ole Olsen, Giedrius Ziberkas, „Building of VHDL Reusable Components for DSP- oriented Architectures“, Informatica, 2000.

25. R. R. Schaller, „Moore's law: past, present, and future“, IEEE Spectrum Volume 34 Issue 6, June 1997

26. V. Jusas, E. Bareiša, R. Šeinauskas, „Skaitmeninių sistemų projektavimas VHDL kalba : vadovėlis“, Kauno technologijos universitetas. Kaunas : Technologija, 1997. 208 p.

27. UMBC universitetas [interaktyvus], Computer Science and Electrical Engineering 2011 [Žiūrėta 2010 gruodžio 15 d.] Prieiga per internetą: <http://www.cs.umbc.edu/portal/help/VHDL/stdpkg.html>.

28. W. H. Chen, C. H. Smith, S. Fralick, „A Fast Computational Algorithm for the Discrete Cosine Transform“, IEEE Trans. on Communications 25, 1004–1009, Sep 1977.

29. C. Loeffler, A. Lightenberg, G. S. Moschytz, „Practical fast 1-D DCT algorithms with 11 Multiplications“, Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP ’89), P. 988–991. 1989.

30. J.Liang, T. D. Tran, „Fast Multiplierless Approximations of the DCT With the Lifting

Scheme“, IEEE Trans. On Signal Processing 49 (12), P. 3032–3044. 2001.

31. Altius Directory [interaktyvus] „MATLAB programming language“ 2010 [Žiūrėta 2011 gegužės 10 d] Prieiga per internetą: <http://www.altiusdirectory.com/Computers/matlab-programming-language.php>.

32. Mathworks Inc. [interaktyvus] „MATLAB technical documentation“. 2011 [Žiūrėta 2011 kovo 20 d.] Prieiga per internetą: <http://www.mathworks.com/help/techdoc/index.html>.

33. Q. Huynh-Thu, M. Ghanbari, „Ipswich Scope of validity of PSNR in image/video quality assessment“, Psytechnics Ltd, June 19, 2008.

34. T. S. Chang, C. S. Kung, C. W. Jen, „A simple processor core design for DCT/IDCT“, IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, pp. 439-447, Apr. 2000.

35. Wiśniewski, Remigiusz, „Synthesis of compositional microprogram control units for programmable devices“, University of Zielona Góra, Zielona Góra, p. 153. 2009.

36. University of Toronto. Computer Engineering Research Group. The FPGA Place-and-Route Challenge [interaktyvus] 2007 [Žiūrėta 2011 sausio 5 d.] Prieiga per internetą: <http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html>.

37. Altium Corp. FPGA SI Tutorial - Simulating the Reflection Characteristics. [interaktyvus] 2005 [Žiūrėta 2010 spalio 12 d.] Prieiga per internetą: <http://wiki.altium.com/display/ADOH/FPGA+SI+Tutorial+-+Simulating+the+Reflection+Characteristics>.

38. M. Hee K. Phoon, M. Yap, C. K. Chai, „A Highly Compatible Architecture Design for Optimum FPGA to Structured-ASIC“, 2005.

39. NASA Office of Logic Design. Drive Strength of Actel FPGAs. [interaktyvus] 2007 [Žiūrėta 2010 spalio 12 d.] Prieiga per internetą:

<http://klabs.org/richcontent/fpga_content/DesignNotes/signal_quality/actel_drive_strength/index.htm>.

40. I. Kuon, J. Rose, „Measuring the Gap Between FPGAs and ASICs “, . IEEE Transactionss on Computer-aided Design of Integrated Circuits and Systems, vol 26, No. 2, Feb. 2007.

41. F. Wagner, „Modeling Software with Finite State Machines: A Practical Approach“, Auerbach Publications, 2006.

*ELECTRONICS AND ELECTRICAL ENGINEERING*

*ISSN 1392 – 1215* *2011. Nr. ()*

*ELEKTRONIKA IR ELEKTROTECHNIKA*

*ELECTRONICS*

*T 170*

*ELEKTRONIKA*

# A 2-D DCT hardware codec based on Loeffler algorithm

**I. Martišius, D. Birvinskas, V. Jusas, Ž. Tamoševičius**

*Software Engineering Department, Kaunas University of Technology*
*Studentų St. 50-404, Kaunas, Lithuania*
*e-mail: vacius.jusas@ktu.lt*

## Introduction

Digital picture and video applications are becoming a inseparable part of our everyday lives. Still image and video devices such as cell phones and cameras rely on codecs (coders and decoders) for image coding, processing and compression. These codecs must meet the ever-growing demands for power dissipation and operation speed. Currently the most popular standards of compression are JPEG for image, MPEG-1, 2, 4 and H263 for video [1]. All of these standards use the discrete cosine transform (DCT) and the inverse discrete cosine transform (IDCT) as a basis of their operation. DCT and the IDCT are some of the most computationally intensive blocks, therefore creating performance bottlenecks in picture and video data compression algorithms [2]. The computational complexity of the DCT in the coder surpasses than of any other unit, consuming 21% of the total computations [3, 4]. The IDCT also incurs the largest computational cost in the decoder. The complexity of these systems leads to large power dissipation and a fast, lossless yet energy efficient hardware codec is required. Implementing DCT/IDCT, as an ASIC is a design solution, which meets the real time processing requirements [2].

Since it's introduction, numerous fast algorithms have been proposed to calculate the DCT and IDCT. This paper focuses on the fast Loeffler algorithm and it's hardware applications.

## Discrete cosine transform

The DCT transforms data from the spatial domain to the spatial frequency domain. It decorrelates the image data, which is typically highly correlated for small areas of the image. Heavy correlation provides much redundant information, whereas just a few pieces of uncorrelated information can represent the same data much more efficiently. Through the DCT transform, the energy of an image sample is packed into just a few spatial frequency coefficients. The DCT typically operates on 8x8 pixel blocks of image data. For an NxN block of samples *x(m,n)*, the two dimensional (2-D) DCT and IDCT are formulated in

$$Y(k,l)=\frac{2}{N}\alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}x(m,n)\cos\left(\frac{(2m+1)\pi k}{2N}\right)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (1)$$

$$x(m,n)=\frac{2}{N}\alpha(k)\alpha(l)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}Y(k,l)\cos\left(\frac{(2m+1)\pi k}{2N}\right)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (2)$$

where $\alpha(0=\frac{1}{\sqrt{2}}$, $\alpha(j) = 1$ for $j \neq 0$. The straightforward implementations of DCT and IDCT require $N^4$ multiplications for an NxN block, so they are rarely implemented in hardware [5].

Beside that, computation of DCT/IDCT need floating-point multiplications, which is slow both in hardware and in software implementations. To achieve better operating speeds, coefficients can be scaled and approximated by integers so that floating-point multiplications can be replaced by integer multiplications.

DCT and IDCT are highly computational intensive, which creates prerequisites for performance bottlenecks in systems utilizing them. To overcome this problem, a number of algorithms have been proposed for more efficient computations of these transforms.

The measure of algorithm effectiveness is the number of mathematical operations needed to perform it. The number of multiplications is extremely important, since it is a relatively complex and power consuming operation, to be performed in hardware. The number of computations can be reduced from $N^4$ to $2Nlog_2N$ using fast DCT algorithms. Many one-dimensional algorithms were published over the years for N data points where $N = 2^m$, $m \geq 2$ [6, 7, 8]. These algorithms are based on the sparse factorizations of the DCT matrix and many of them are recursive. The theoretical lower bound on the number of multiplications required for 1-D 8-point DCT has proven to be 11. In this sense, the method proposed by Loeffler, with 11 multiplications and 29 additions, is the most efficient solution. That's why it was chosen for this implementation.

Loeffler algorithm is combined out of stages. The stages of the algorithm numbered 1 to 4 are parts that have to be executed in serial mode due to the data dependency. In stage 2, the algorithm splits into two parts. One for the even coefficients, the other for the odd

ones. The even part is a 4 point DCT, again separated in even and odd parts in stage 3. The rounding block in Fig. 1 signifies a multiplication by a constant.

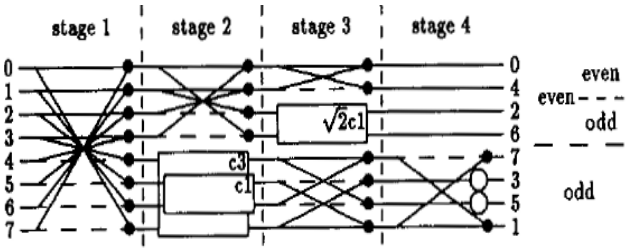



**Fig. 1.** 8-point Loeffler DCT algorithm [9]

Each stage must be executed in series and can not be evaluates in parallel because of data dependencies. However, calculations inside one stage can be parallelized. We clarify the building blocks of the algorithm in Fig. 2.

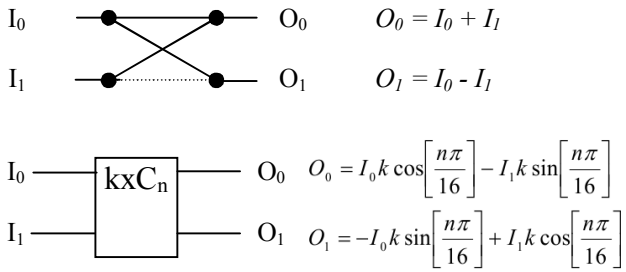



**Fig. 2**. The "butterfly" and the rotation block with their associated equations [9]

The rotation block can be calculated using 3 multiplications and 3 additions instead of 4 multiplications and 2 additions.

The two dimensional DCT transform equation (1) can be expressed as,

$$Y(k,l)=\frac{2}{N}\alpha(k)\alpha(l)\sum_{n=0}^{N-1}\cos\left(\frac{(2m+1)\pi k}{2N}\right)\sum_{m=0}^{N-1}x(m,n)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (3)$$

This property, known as separability, has the advantage that 2D DCT can be computed in two steps by successive 1-D operations on rows and columns. The idea of a so-called row-column approach and is graphically illustrated in Fig. 3. It is the most popular approach for computing the 2D DCT/IDCT. It result in simple regular implementations that are less computationally efficient than direct 2D implementations.

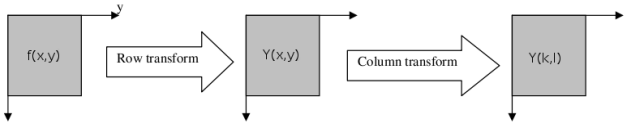



**Fig. 3.** Row-column approach for 2D-DCT/IDCT

**Algorithm modelling**

As we can see from the graph in Fig. 1, only 7 constant values need to be stored in the memory for Loeffler algorithm to function.

The algorithm signal-flow graph (Fig. 1) can be directly translated into a hardware implementation. However, it still uses floating-point multiplication which is too complex for hardware. The workaround for this problem is constant scaling. All constant can be multiplied by a factor of 2, so the floating point is eliminated, and the results are divided accordingly. In hardware applications, this kind of multiplication and division is easily done by shifting bits left or right. The choice of fixed-point constants is arbitrary. Using more bits gives better accuracy, because a more exact value of the constant is retained but with increased risk of overflow and surplus hardware register logic for storing and processing data.

To measure the quality of coding, the peak signal-to-noise ratio (PSNR) was calculated. It is most easily defined via the mean squared error (MSE) which for two m×n monochrome images I and K, where one of the images is considered a noisy approximation of the other, is defined as:

$$MSE=\frac{1}{64}\sum_{i=0}^{7}\sum_{j=0}^{7}[I(i,j)-K(i,j)]^2 \quad (5)$$

The PSNR is defined as:

$$PSNR=10\log_{10}\left(\frac{MAX_I^2}{MSE}\right) \quad (6)$$

Here, $MAX_I$ is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255 [10]. Typical values for the PSNR in lossy image and video compression are between 30 and 50 dB.

Mathematical modelling showed that the best choice for quality vs. size is a constant length of 10 bits. The codec using 10 bit constants is lossy because of fixed-point arithmetic. We show the different images used to test the algorithm Fig. 4.

Lena.bmp Cell.bmp Fruit.bmp

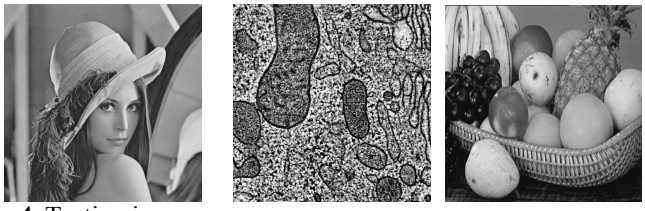


**Fig. 4**. Testing images

The PSNR measured for each of them are shown in Table 1.

**Table 1**. Mathematical modelling results

| Picture | PSNR |
|---|---|
| Lena.bmp | 33.6577 |
| Cell.bmp | 33.2790 |
| Fruit.bmp | 33.7290 |

Using a constant length of less than 10 bits leads to a lossy algorithm, which adds gross errors in the encoding of the image, which, when decoded, show up as visible relics in the picture. Although reducing constant bit size has a downside of reducing image quality, it also

greatly reduces the area and power consumption of the finished schematic, there for it might still be used in applications, where image quality is a secondary requirement, and power dissipation is a key, like portable video systems or cellular phones.

**Hardware implementation**

Hardware model of 2D-DCT is implemented using VHDL language. The system is divided into the following blocks:

- computation unit;
- data buffer;
- counter;
- general control unit.

These blocks are shown in Fig. 5.

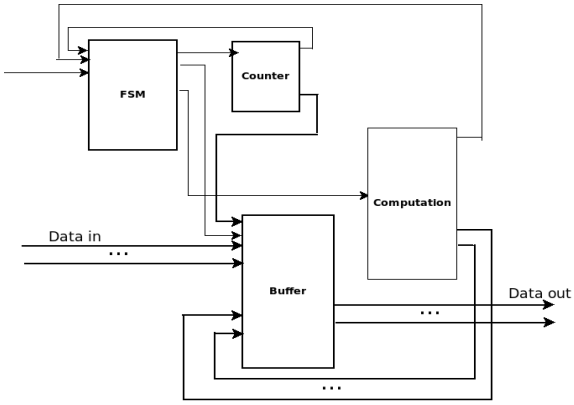



**Fig. 5.** Top level schematic

The computation unit performs one dimensional DCT or IDCT operation. This block has a generic interface, suitable for both DCT and IDCT algorithms. Interface has 8 data input buses and 8 output data buses. Beside data buses, the block uses *start* and *done* signals, for control and data flow simplification. This design allows reuse of all the system components for other DCT and IDCT computation algorithms.

For encoding, the system computes a DCT for a one-dimensional 8-point input sequence using Loeffler algorithm. This algorithm is directly translated from Fig. 1 and adapted to use fixed-point arithmetic with 10 bits constant length.

For decoding data DCT block is replaced with IDCT. Inverse transform may be computed by executing the stages backwards. A codec using any other 8-point 1D-DCT/IDCT computation algorithm can be implemented via reusing the components of this system.

Data buffer is the most complex block of the system. It has 4 functions:

1. Input and storing an 8x8 matrix,
2. Output lines of matrix to DCT processor and replace data lines with the result;
3. Output columns of the matrix to the DCT processor and replace data line with the result;
4. Output of the data matrix.

When 1D-DCT processor computes a result of any given 8-point input sequence the input data is no longer needed and can be overwritten with the computation result. This reduces memory size, resulting in smaller area and power consumption, however this method increases interconnect network.

General control unit is implemented as a finite state machine. It is used to control data flow between components.

**Implementation results**

We simulated VHDL register transfer level (RTL) code using Cadence simulation and modelling tools. Synthesis was performed using Synopsys tools. Since file formats for binary files, read by VHDL are not standardized, we used a pre/post-processor to convert binary images files to text format and vice versa.

Images used for testing where the same as in mathematical modelling. Results of synthesized gate level circuit simulation are shown in Table 2

**Table 2**. Gate level simulation results

| Picture | PSNR |
|---|---|
| Lena.bmp | 31.5951 |
| Cell.bmp | 30.8324 |
| Fruit.bmp | 31.8590 |

We show differences in PSNR for mathematical modelling and gate level circuit modelling of each image in Fig. 6.

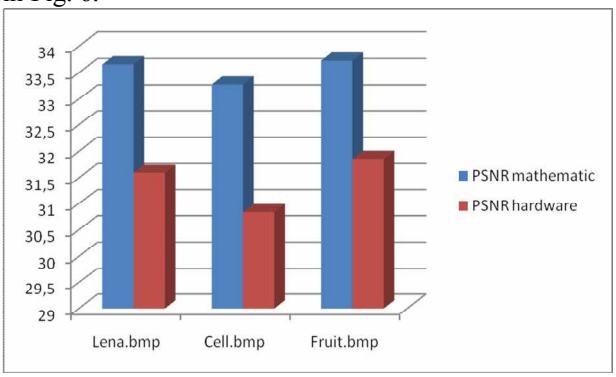



**Fig. 6**. Difference in PSNR

Implementation is also proven to work on FPGA matrices. A Xilinx Spartan-3AN was used to synthesize and test the RTL code. Utilization summary for the device is shown in Table 3.

**Table 3**. FPGA device utilization

| Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Slice Flip Flops | 426 | 22,528 | 1% |
| 4 input LUTs | 2,280 | 22,528 | 10% |
| Occupied Slices | 1,309 | 11,264 | 11% |
| Bonded IOBs | 274 | 375 | 73% |
| MULT18X18SIOs | 22 | 32 | 68% |

## Conclusions

In this paper, we reported a hardware implementation scheme and results of a Loeffler algorithm based 2-D discrete cosine transform codec. This codec performs a full 2-D DCT encoding and 2-D IDCT decoding. It is fully functional, capable of being used in any hardware coding algorithm. The Loeffler algorithm is one of the newest and most effective fast DCT algorithms for hardware implementation. 2-D DCT/IDCT was computed by applying 1-D operations on matrix using a row/column approach. Mathematical and gate-level modelling of the algorithm has shown that fixed-point arithmetic and constant approximations add errors to the results of the algorithm.

For a hardware implementation, additional data buffers and control logic is used to perform 2-D transforms. Since hardware implementation is less accurate than mathematic modelling, more errors occur in the computational results. However, differences are small and cannot be directly observed. This codec should provide enough quality for low resolution image compression or processing algorithms at high speed.

## References


1. **Atitallah A. B., Kadionik P., Ghozzi F., Nouel P., Masmoudi N., Marchegay P.** Optimization and implementation on FPGA of the DCT/IDCT algorithm. IEEE International Conference on Acoustics // Speech and Signal Processing (ICASSP '06), 2006. – P. 928–931.
2. **Bukhari K. Z., Kuzmanov G. K., Vassiliadis S.** DCT and IDCT implementations on different FPGA technologies // Proceedings of the 13th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC'02), Veldhoven, The Netherlands, November 2002. – P. 232–235.
3. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** On the Enrichment of Functional Delay Fault Tests // Information Technology and Control. – No. 38(3), 2009. – P. 208–216.
4. **Bareiša E., Bieliauskas P., Jusas V., Targamadzė A., Motiejūnas L., Šeinauskas R.** Factor of Randomness in Functional Delay Test Generation for Full Scan Circuits // Electronics and Electrical Engineering. - Kaunas: Technologija, 2010. – No. 9(105). – P. 39–42.
5. **Narasimha M. J., Peterson A. M.** On the computation of the discrete cosine transform // IEEE Trans. Commun. 26 (6), 1978. – P. 934–936.
6. **Chen W.-H., Smith C. H., Fralick S.** A Fast Computational Algorithm for the Discrete Cosine Transform // IEEE Trans. on Communications 25, Sep 1977. – P. 1004–1009.
7. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** On the Enrichment of Static Functional Test // Electronics and Electrical Engineering. - Kaunas: Technologija, 2009. – No. 3(91). – P. 9–14.
8. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** The non-scan delay test enrichment based on random generated long test sequences // Information Technology and Control. – No. 39(4), 2010. – P. 251–256.
9. **Loeffler C., Lightenberg A., Moschytz G. S.** Practical fast 1-D DCT algorithms with 11 Multiplications // Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP '89), 1989. – P. 988–991.
10. **August N. J., Dong Sam Ha.** Low Power Design of DCT and IDCT for Low Bit Rate Video Codecs // IEEE transactions on multimedia, vol 6, No 3, June 2004. – P. 414–422.


Received 2011.03.03

**I. Martišius, D. Birvinskas, V. Jusas, Ž. Tamoševičius. 2-D DCT kodeko realizacija, naudojant Loeffler algoritmą // Elektronika ir elektrotechnika. - Kaunas: Technologija, 2011. - Nr. (). - P..**
Straipsnyje aprašomas aparatinis dvimatės kosinusinės transformacijos (DCT) kodeko realizacija. Šis kodekas atlieka pilną 2-D DCT kodavimą ir dekodavimą (2-D IDCT). Jis pilnai veikiantis ir pritaikomas visuose aparatiniuose kodavimo algoritmuose. Loeffler algoritmas yra vienas efektyviausių ir naujausių greitųjų DCT algoritmų. Dvimatė DCT/IDCT skaičiuojama pritaikant vienmatės DCT algoritmą eilučių/ stulpelių metodu. Atliekant dvimates DCT operacijas, panaudojama papildoma atmintis ir valdymo logika. Aparatinės realizacijos skaičiavimo tikslumas mažesnis nei matematinio modeliavimo, tačiau šie skirtumai nedideli ir nematomi plika akimi.. Il. 6, lent. 3, bibl. 8 (lietuvių kalba; santraukos lietuvių ir anglų k.).

**I. Martišius, D. Birvinskas, V. Jusas, Ž. Tamoševičius. A 2-D DCT hardware codec based on Loeffler algorithm // Electronics and Electrical Engineering. - Kaunas: Technologija, 2011. – N0. (). - P..**
In this paper, we report a hardware implementation scheme and results of a Loeffler algorithm based 2-D discrete cosine transform codec. This codec performs a full 2-D DCT encoding and 2-D IDCT decoding. It is fully functional, capable of being used in any hardware coding algorithm. The Loeffler algorithm is one of the newest and most effective fast DCT algorithms for hardware implementation. 2-D DCT/IDCT was computed by applying 1-D operations on matrix using a row/column approach. For a hardware implementation, additional data buffers and control logic is used to perform 2-D transforms. Since hardware implementation is less accurate than mathematical modelling, more errors occur in the computational results. However, differences are small and cannot be directly observed. Ill. 6, tabl. 3, ref. 8 (English, Abstracts in Lithuanian and English).

# APARATNĖ DVIMATĖS DCT IR IDCT REALIZACIJA, NAUDOJANT BINDCT ALGORITMĄ

**Darius Birvinskas[1], Ignas Martišius[2]**

*Kauno technologijos universitetas, Programų inžinerijos katedra, Studentų 50 – 406, Kaunas, Lietuva, [1]b.darius@nkm.lt, [2]ignas_martisius@inbox.com*

**Santrauka (abstract)**. Straipsnyje aprašoma aparatinė dvimatės kosinusinės transformacijos (DCT) realizacija. Šis skaičiavimo metodas atlieka pilną 2-D DCT kodavimą ir dekodavimą (2-D IDCT). Įrenginyje naudojamas Bindct algoritmas yra vienas efektyviausių ir naujausių greitųjų DCT algoritmų, apskaičiuojantis transformaciją be daugybos veiksmų naudojimo.

**Raktiniai žodžiai:** DCT, IDCT, BinDCT, transformacijos, aparatinė

## 1 Įžanga

Skaitmeninis vaizdo ir garso apdorojimas tampa neatskiriama kasdienybės dalimi. Šiuos kodavimus vis dažniau atlieka mobilūs įrenginiai, tokie kaip fotoaparatai, kameros, mobilieji telefonai. Tokiuose įrenginiuose kodavimui, apdorojimui ir suspaudimui naudojami kodekai (kodavimo – dekodavimo algoritmai). Šie kodekai turi atitikti vis didėjančius greitaveikos ir galios suvartojimo reikalavimus. Šiuo metu populiariausi vaizdo suspaudimo standartai yra JPEG - paveikslėliams, bei MPEG-1,2,4 bei H263 – video informacijai. Visi šie standartai remiasi diskrečiąja kosinusine transformacija (Discrete Cosine Transform- DCT) bei atvirkštine DCT (IDCT). DCT bei IDCT yra sudėtingiausi ir daugiausiai skaičiavimų atliekantys blokai algoritme, todėl nuo jų greitaveikos priklauso viso kodavimo greitis [1]. Matematiškai DCT ir IDCT yra patys sudėtingiausi algoritmo blokas, sunaudojantis apie 21% visų matematinių skaičiavimų. Didelis šių sistemų sudėtingumas lemia didelį galios suvartojimą, todėl siekiama sukurti greitą, kuo mažiau koduojamos informacijos prarandantį tačiau taupų galiai kodeką. Šios problemos sprendimas – vienlustė sistema, atitinkanti galios bei greičio reikalavimus.

## 2 Diskrečioji kosinusinė transformacija

DCT yra daugumos vaizdinės informacijos spaudimo algoritmų pagrindas. Kadangi tai visiškai grįžtama transformacija, ją galima naudoti ir prarandančioms ir neprarandančioms kokybės spaudimo technologijoms. DCT yra ypatingas Furje transformacijos atvejis, kur sinusinė dedamoji yra pašalinama. Vaizdinės informacijos kodavime naudojama dvimatė (2-D) diskrečioji kosinusinė transformacija, kurioje dydžių įvertinimas vyksta du kartus [1]. 2-D DCT įėjimas yra NxN vaizdo taškų matrica. Po transformacijos gaunama NxN dažnių erdvės komponenčių matrica. Dvimatę tiesioginę ir atvirkštinę diskrečią kosinusinę transformaciją galima aprašyti šiomis formulėmis:

$$Y(k,l) = \frac{2}{N}\alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(m,n)\cos\left(\frac{(2m+1)\pi k}{2N}\right)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (1)$$

$$x(m,n) = \frac{2}{N}\alpha(k)\alpha(l)\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} Y(k,l)\cos\left(\frac{(2m+1)\pi k}{2N}\right)\cos\left(\frac{(2n+1)\pi l}{2N}\right) \quad (2)$$

kur $\alpha(0) = \frac{1}{\sqrt{2}}$ ir α(j) = 1 kai j ≠ 0.

DCT algoritmo efektyvumo matas yra daugybos operacijų skaičius. Tiesioginė DCT realizacija naudoja $N^4$ daugybos operacijų NxN įvedimo blokui, todėl yra retai naudojama. Dauguma realizacijų naudoja greituosius algoritmus, kurie sumažina daugybos operacijų skaičių iki *2Nlog2N* [2]. Šie algoritmai pagreitina DCT, sumažindami atliekamų skaičiavimų kiekį.

Šiuo metu sugalvota daug greitųjų vienmatės (1-D) transformacijos algoritmų kurie koduoja N ilgio skaičių seką, kur N = 2m, m ≥ 2. Matematiškai įrodyta, kad mažiausias įmanomas daugybos operacijų keikis reikalingas 8 skaičių sekai transformuoti yra 11 [2]. Tokį skaičių daugybos operacijų naudoja C. Loeffler pasiūlytas algoritmas, laikomas pačiu efektyviausiu.

Dvimatę transformaciją galima atlikti du kartus pritaikant vienmatę DCT/IDCT transformaciją. Pirmą karta transformacija taikoma įėjimo bloko eilutėms, antra kartą – stulpeliams. Toks skaičiavimo metodas vadinamas eilučių/stulpelių metodu ir pasižymi lengvai valdomu duomenų srautu bei paprasta struktūra, todėl dažnai taikomas aparatinėse realizacijose [1].

## 3 BinDCT algoritmas

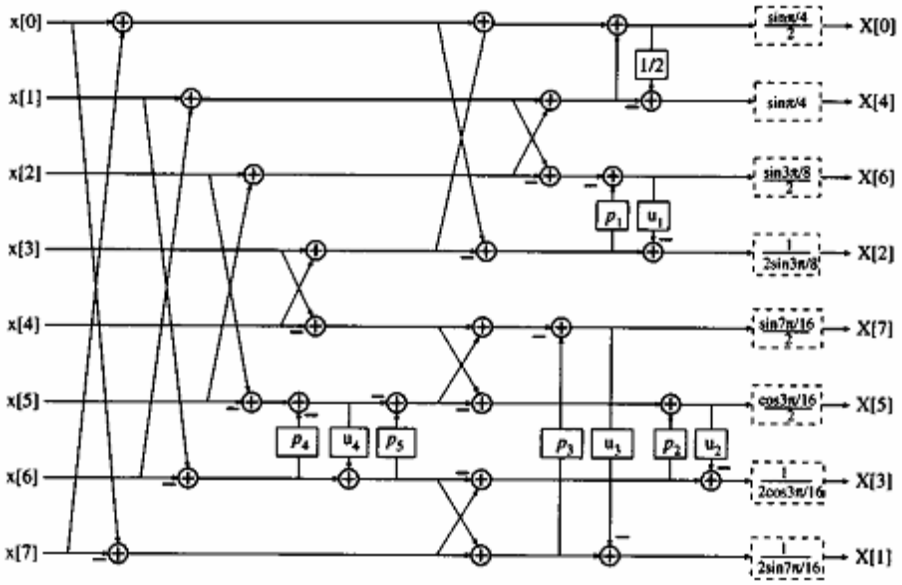
Visi DCT/IDCT skaičiavimo algoritmai naudoja slankaus kablelio aritmetiką. Aparatinė tokios aritmetikos realizacija užima didelį lusto plotą, suvartoja daug galios ir yra lėtesnė palyginus su sveikųjų skaičių aritmetika. Dėl šių priežasčių projektuojant aparatinius skaičiavimo įrenginius stengiamasi išvengti slankaus kablelio operacijų. BinDCT yra vienas naujausių greitosios DCT ir IDCT skaičiavimo algoritmų. Šio algoritmo pranašumas – galimybė apskaičiuoti DCT nenaudojant daugybos veiksmų [3]. Šis algoritmas plokštumos pasukimą pakeičia keletu dvinarių reikšmių, kurios vadinamos pakėlimo žingsniais. Pakėlimo žingsniai turi tokį formatą $\frac{a}{2^b}$, kur $a$ yra sveikasis skaičius, o $b$ – natūralusis. Tokį pakėlimo žingsnį nesunku realizuoti aparatinėmis priemonėmis naudojant vien poslinkio ir sudėties operacijas. Algoritmo autoriai siūlo tris galimas pakėlimo žingsnių kombinacijas kurios skiriasi savo tikslumu [4].

Pritaikius šitokį „daugybos“ būdą Loeffler algoritmui ir pasirinkus tinkamas pakėlimo žingsnių kombinaciją, gaunamas greitasis dvimatės diskrečiosios transformacijos algoritmas, naudojantis tik sveikųjų skaičių sudėties ir postūmio operacijas. Šio algoritmo signalų perdavimo grafas pavaizduotas 1 pav. , galimos konstantų variacijos parodytos 1 letelėje.

**Lentelė Nr. 1. Algoritmo konfigūracijos [4]**

| Konstantos | A | B | C |
|---|---|---|---|
| $p_1$ | 3/8 | 3/8 | 3/8 |
| $u_1$ | 3/8 | 3/8 | 3/8 |
| $p_2$ | 7/8 | 1/2 | 7/8 |
| $u_2$ | 1/2 | 1/8 | 1/2 |
| $p_3$ | 3/16 | 1/8 | 1/8 |
| $u_3$ | 3/16 | 0 | 0 |
| $p_4$ | 7/16 | 0 | 0 |
| $u_4$ | 11/16 | 3/8 | 3/8 |
| $p_5$ | 3/8 | 5/8 | 5/8 |

Kaip matyti iš 1 pav, algoritmui nereikalingas kosinuso reikšmių skaičiavimas, reikšmių lentelės ar konstantos. Šis grafas yra dvikryptis. Tai reiškia, jog atliekant veiksmus iš kairės į dešinę vektorius [x] transformuojamas į kosinuso koeficientus [X] (tiesioginė diskrečioji kosinusinė transformacija), atliekant veiksmus iš dešinės į kairę, kosinuso koeficientai [X] transformuojami į pradinį vektorių [x] (atvirkštinė diskrečioji kosinusinė transformacija) [2]. Tokį signalų perdavimo grafą patogu realizuoti programinėmis ar aparatinėmis priemonėmis.




**1 pav. binDCT algoritmo signalų perdavimo grafas [3]**

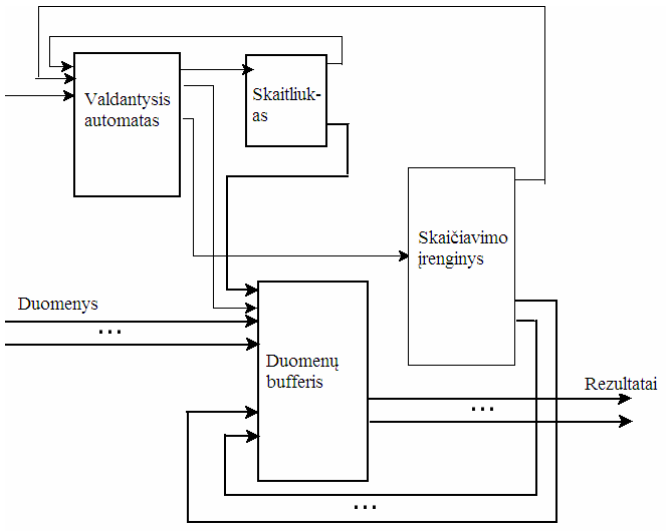
## 4 Aparatinė realizacija

Dvimatę DCT ir IDCT galima apskaičiuoti atskirai taikant vienmatį algoritmą 8x8 matricos eilutėms ir stulpeliams. Tokiam 2-D DCT skaičiavimui reikia realizuoti ne tik patį skaičiavimo algoritmą, bet ir papildomą valdymo įrangą [5].

Dvimatės DCT/IDCT skaičiavimo procesorių sudaro 4 blokai: Valdymo automatas, skaitliukas, duomenų buferis ir skaičiavimo įrenginys. Skaičiavimo įrenginys atlieka vienmatę transformaciją 8 skaičių eilutei. Šis įrenginys turi bendrinę sąsają, kuri tinkama tiek DCT tiek IDCT skaičiavimo algoritmams. Tai reiškia, jog abiem atvejais kitus procesoriaus blokus galima panaudoti be pakeitimų.

Valdymo automato perėjimai priklauso nuo esamos būsenos ir įėjimo signalų. Toks būsenų automatas vadinamas Mili automatu. Kiekvienoje būsenoje išduodami tuo metu reikalingi valdymo signalai. Darbą automatas pradeda pradinėje būsenoje ir eina per būsenas pagal perėjimų sąlygas. Skaitliukas skaičiuoja nuo 0 iki 9, ir išduoda signalus, nuo kurių priklauso valdančiojo automato perėjimai.

Duomenų buferis saugo vieną 8x8 duomenų matricą. Atlikus transformaciją, rezultatai yra įrašomi į duomenų vietą, tokiu būdu sumažinat loginių ventilių skaičių procesoriuje. Valdymo automatas ir skaitliukas valdo duomenų srautus tarp duomenų buferio ir skaičiavimo įrenginio, bei procesoriaus išorės. Procesoriaus aparatinis aprašas realizuotas VHDL projektavimo kalba. Mūsų realizuoto procesoriaus blokinė diagrama pavaizduota 2 pav.




**2 pav. Skaičiavimo procesoriaus blokinė diagrama**

## 5 Modeliavimo rezultatai

Skaičiuojant kodavimo kokybę, naudojamas maksimalus signalo ir triukšmo santykis (Peak Signal to Noise Ratio – PSNR). Jis lengviausiai aprašomas vidutiniu kvadratiniu nuokrypiu (Mean Square Error – MSE), kuris dviems vienspalviams vaizdams I ir K, kur vienas vaizdas yra kito aproksimacija su triukšmu, aprašomas 7 bei 8 formulėmis:

$$MSE = \frac{1}{64}\sum_{i=0}^{7}\sum_{j=0}^{7}\left[I(i,j) - K(i,j)\right]^2 \tag{7}$$

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{8}$$

kur $MAX_I$ yra maksimali galima vaizdo taško reikšmė. Jei vaizdo taškai koduojami 8 bitais, ši reikšmė yra 255. Dažniausiai suspaustiems vaizdams PSNR reikšmės yra apie 30 – 50 decibelų (dB). Didesnis PSNR santykis reiškia geresnę kokybę.

Atlikus matematinį modeliavimą su visomis trimis algoritmų versijomis, apskaičiuotas kiekvieno įnešamas PSNR. Rezultatai, kartu su algoritmų matematinių operacijų skaičiaus palyginimu, pateiktas 2 lentelėje. Aparatinei realizacijai pasirinktas C algoritmo variantas. Šis turi mažiausią matematinių operacijų skaičių, ir duoda panašius kodavimo rezultatus kaip kiti variantai.

**Lentelė Nr. 2. Modeliavimo rezultatai**

| Versija | Sudėtys | Postūmiai | PSNR |
|---|---|---|---|
| A | 36 | 19 | 49.93 |
| B | 31 | 14 | 49.47 |
| C | 30 | 13 | 49.45 |

Aparatinio modelio testavimui pasirinkti trys 512x512 taškų dydžio monochromatiniai paveikslėliai. Kiekvienas paveikslėlis buvo užkoduotas ir atkoduotas naudojant VHDL kalba aprašytą procesoriaus modelį. Kiekvienu atveju paskaičiuojamas maksimalus signalo ir triukšmo santykis, lyginant su originaliu paveikslėliu. Skaičiavimo rezultatai patiekti 3 lentelėje.

**Lentelė Nr. 3. Modeliavimo rezultatai**

| Paveikslėlis | PSNR |
|---|---|
| Lena.bmp | 49.47 |
| Cell.bmp | 49.63 |
| Fruit.bmp | 49.55 |

Nors šis algoritmas netinkamas kokybės neprarandančiam kodavimui, kokybe jis nenusileidžia kitiems greitiesiems DCT algoritmams, naudojamiems daugumoje šiandieninių įrenginių. Kadangi algoritme nenaudojamos konstantos ir daugybos operacijos, šis algoritmas itin taupus galiai ir užima labai mažai lusto ploto, taip pat yra itin greitas. Atlikus testavimą Xilinx Spartan 3AN-700 FPGA matricoje gautas 0,158 W galios suvartojimas bei maksimalus 67Mhz dažnis, kuris leidžia apdoroti apie 350 512x512 vaizdo taškų turinčių vaizdų per sekundę. Šitoks DCT/IDCT procesorius gali pasiekti greitaveiką, reikalingą HD video informacijos apdorojimui.

## 6 Išvados

Straipsnyje aprašėme aparatinę BinDCT algoritmu paremtą dvimatės kosinusinės transformacijos schemą. BinDCT – vienas naujausių ir greičiausių sparčiųjų DCT algoritmų, taip pat reikalaujantis mažiausiai matematinių operacijų. Sukurta schema atlieka dvimatį DCT kodavimą, panaudoja vienmatės DCT algoritmą efektyviu eilučių/stulpelių metodu. Šis kodekas tinkamas žemos raiškos vaizdų kodavimui ir apdorojimui dideliais greičiais ir turėtų atitikti ateityje prietaisams keliamus galios bei greitaveikos reikalavimus.

## Literatūros sąrašas


[1] **Narasimha M. J., Peterson A. M.** On the computation of the discrete cosine transform // IEEE Trans. Commun. 26 (6), 1978. – P. 934–936.

[2] **Loeffler C., Lightenberg A., Moschytz G. S.** Practical fast 1-D DCT algorithms with 11 Multiplications // Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP ’89), 1989. – P. 988–991.

[3] **Liang J., Tran T.D.** Fast Multiplierless Approximations of the DCT With the Lifting Scheme // IEEE Trans. On Signal Processing 49 (12), 2001. – P. 3032–3044.

[4] **Tran T. D.** "The BinDCT: fast multiplierless approximation of the DCT". IEEE Signal Processing Letters, vol. 7, no. 6, pp. 141-144, jun. 2000.

[5] **August N. J., Dong Sam Ha.** Low Power Design of DCT and IDCT for Low Bit Rate Video Codecs // IEEE transactions on multimedia, vol 6, No 3, June 2004. – P. 414–422.


## A BinDCT algorithm-based 2-D DCT and IDCT hardware architecture


In this article we propose a 2-D DCT and IDCT codec architecture based on the BinDCT algorithm. This codec is used to calculate a full 2-D coding and decoding, utilizing a row/column 1-D DCT approach . The BinDCT algorithm is one of the newest and most effective fast DCT algorithm, calculating the DCT without the use of multiplications or constants. The codec is capable of meeting the speed and power requirements for future picture and video processing devices.

# Priedas Nr. 2. Algoritmų VHDL modeliai

```
::::::::::::::
bindct/bindct.vhd
::::::::::::::
--  This is VHDL implementation of Bin DCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

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

entity bindct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end bindct;

architecture behave of bindct is

type state_type is (IDLE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;

begin
process1: process (clk,rst)
```

```
variable x0 : signed(31 downto 0);
variable x1 : signed(31 downto 0);
variable x2 : signed(31 downto 0);
variable x3 : signed(31 downto 0);
variable x4 : signed(31 downto 0);
variable x5 : signed(31 downto 0);
variable x6 : signed(31 downto 0);
variable x7 : signed(31 downto 0);
variable x8 : signed(31 downto 0);
variable x9 : signed(31 downto 0);
variable x10 : signed(31 downto 0);
variable x11 : signed(31 downto 0);

begin
      if (rst ='0') then
            state <= IDLE;
      elsif (clk='1' and clk'event) then
            case state is
                  when IDLE =>
                        if start ='1' then
                              state <= STAGE1;
                        else
                              x0 := X"00000000";
                              x1 := X"00000000";
                              x2 := X"00000000";
                              x3 := X"00000000";
                              x4 := X"00000000";
                              x5 := X"00000000";
                              x6 := X"00000000";
                              x7 := X"00000000";
                              x8 := X"00000000";
                              x9 := X"00000000";
                              x10 := X"00000000";
                              x11 := X"00000000";
                              done <= '0';
                              state <= IDLE;
                        end if;
                  when STAGE1 =>
                        x0 := resize(in0 + in7, x0'length);
                        x1 := resize(in1 + in6, x1'length);
                        x2 := resize(in2 + in5, x2'length);
                        x3 := resize(in3 + in4, x3'length);
                        x4 := resize(in3 - in4, x4'length);
                        x5 := resize(in2 - in5, x5'length);
                        x6 := resize(in1 - in6, x6'length);
                        x7 := resize(in0 - in7, x7'length);
                        done <= '0';
                        state <= STAGE2;
                  when STAGE2 =>
                        x8 := x0 + x3;
                        x9 := x1 + x2;
                        x10 := x1 - x2;
                        x11 := x0 - x3;

                        x2 := x6 + (x5(31 downto 3) + x5(31 downto 2));
--x(3) = x(7) + (x(6) * 3/8);
                        x3 := (x2(31 downto 3) + x2(31 downto 1)) - x5;
--x(4) = (x(3) * 5/8) - x(6);
                        x1 := x4 + x3;
```

```
                              x6 := x4 - x3;
                              x3 := x7 - x2;
                              x7 := x7 + x2;
                              done <= '0';
                              state <= STAGE3;
                        when STAGE3 =>
                              x0 := x8 + x9;
                              x2 := (x11(31 downto 3) + x11(31 downto 2)) - x10;
      --x(3) = (x(12) * 3/8) - x(11);
                              x4 := x7(31 downto 3) - x1;
            --x(5) = (x(8) * 1/8) - x(2);
                              x5 := x6 + (x3(31 downto 3) + x3(31 downto 2) + x3(31
downto 1)); --x(6) = x(7) + (x(4) * 7/8);
                              done <= '0';
                              state <= STAGE4;
                        when STAGE4 =>
                              out0 <= resize(x0, out0'length);
                              out4 <= resize((x0(31 downto 1) - x9), out4'length);
      --output(5) = (x(1) * 1/2) - x(10);
                              out6 <= resize(x2, out6'length);
                              out2 <= resize((x11 + (x2(31 downto 3) + x2(31 downto
2))), out2'length); --output(3) = x(12) + (x(3) * 3/8);
                              out7 <= resize(x4, out7'length);
                              out5 <= resize(x5, out5'length);
                              out3 <= resize((x3 - x5(31 downto 1)), out3'length);
      --output(4) = x(4) - (x(6) * 1/2);
                              out1 <= resize(x7, out1'length);
                              done <= '1';
                              state <= IDLE;
                        when others =>
                              done <= '0';
                              state <= IDLE;
                  end case;
            end if;
end process process1;

end behave;

::::::::::::::
binidct/binidct.vhd
::::::::::::::
--  This is VHDL implementation of Bin IDCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

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

```
use ieee.numeric_std.all;

entity binidct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end binidct;

architecture behave of binidct is

type state_type is (IDLE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;



begin
process1: process (clk,rst)
      variable x0 : signed(31 downto 0);
      variable x1 : signed(31 downto 0);
      variable x2 : signed(31 downto 0);
      variable x3 : signed(31 downto 0);
      variable x4 : signed(31 downto 0);
      variable x5 : signed(31 downto 0);
      variable x6 : signed(31 downto 0);
      variable x7 : signed(31 downto 0);
      variable x8 : signed(31 downto 0);
      variable x9 : signed(31 downto 0);
      variable x10 : signed(31 downto 0);

      begin
            if (rst ='0') then
                  state <= IDLE;
            elsif (clk='1' and clk'event) then
                  case state is
                        when IDLE =>
                              if start ='1' then
                                    done <= '0';
                                    state <= STAGE4;
                              else
```

```
                                    x0 := X"00000000";
                                    x1 := X"00000000";
                                    x2 := X"00000000";
                                    x3 := X"00000000";
                                    x4 := X"00000000";
                                    x5 := X"00000000";
                                    x6 := X"00000000";
                                    x7 := X"00000000";
                                    x8 := X"00000000";
                                    x9 := X"00000000";
                                    x10 := X"00000000";
                                    done <= '0';
                                    state <= IDLE;
                              end if;
                        when STAGE4 =>
                              x0 := resize(in0, x0'length);
                              x1 := resize((in0(16 downto 1) - in4), x1'length);
      --x(2) = (x(1) * 1/2) - input(5);
                              x2 := resize(in6, x2'length);
                              x3 := resize((in2 - (in6(16 downto 3) + in6(16 downto
2))), x3'length); --x(4) = input(3)-(x(3) * 3/8); x2 = in6
                              x4 := resize(in7, x4'length);
                              x5 := resize(in5, x5'length);
                              x6 := resize((in5(16 downto 1) + in3), x6'length);
      --x(7) = (x(6) * 1/2) + input(4);  x5 = in5
                              x7 := resize(in1, x7'length);
                              done <= '0';
                              state <= STAGE3;
                        when STAGE3 =>
                              x9 := x1;
                              x8 := x0 - x9;
                              x10 := (x3(31 downto 3) + x3(31 downto 2)) - x2;
      --x(11) = (x(4) * 3/8) - x(3);
                              x4 := x7(31 downto 3) - x4;
                              x5 := x5 - (x6(31 downto 3) + x6(31 downto 2) + x6(31
downto 1)); --x(6) = x(6) - (x(7) * 7/8);
                              done <= '0';
                              state <= STAGE2;
                        when STAGE2 =>
                              x0 := x8 + x3;          --x8(31 downto 1) + x4(31
downto 1);
                              x0 := x0(31) & x0(31 downto 1);
                              x1 := x9 + x10;
                              x1 := x1(31) & x1(31 downto 1);
                              x2 := x9 - x10;
                              x2 := x2(31) & x2(31 downto 1);
                              x3 := x8 - x3;
                              x3 := x3(31) & x3(31 downto 1);
                              x8 := x7 - x6;
                              x8 := x8(31) & x8(31 downto 1);
                              x9 := x4 - x5;
                              x9 := x9(31) & x9(31 downto 1);

                              x4 := x4 - x9;
                              x7 := x7 - x8;
                              x5 := (x8(31 downto 3) + x8(31 downto 1)) - x9;
--x(6) = (x(9) * 5/8) - x(10);
                              x6 := x8 - (x5(31 downto 3) + x5(31 downto 2));--x(7)
= x(9) - (x(6) *3/8);
```

```
                              done <= '0';
                              state <= STAGE1;
                        when STAGE1 =>
                              out0 <= resize((x0(31 downto 1) + x7(31 downto 1)),
out0'length);
                              out1 <= resize((x1(31 downto 1) + x6(31 downto 1)),
out1'length);
                              out2 <= resize((x2(31 downto 1) + x5(31 downto 1)),
out2'length);
                              out3 <= resize((x3(31 downto 1) + x4(31 downto 1)),
out3'length);
                              out4 <= resize((x3(31 downto 1) - x4(31 downto 1)),
out4'length);
                              out5 <= resize((x2(31 downto 1) - x5(31 downto 1)),
out5'length);
                              out6 <= resize((x1(31 downto 1) - x6(31 downto 1)),
out6'length);
                              out7 <= resize((x0(31 downto 1) - x7(31 downto 1)),
out7'length);
                              done <= '1';
                              state <= IDLE;
                        when others =>
                              done <= '0';
                              state <= IDLE;
                  end case;
            end if;
end process process1;

end behave;
::::::::::::::
chen_dct/chen_const.vhd
::::::::::::::
--  This is VHDL file with constants for Chen DCT/IDCT algorithm
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

package chen_const is
      -- konstantos padaugintos is 2^10
      -- c14->s14, c18->s38, c116->s716
      constant s14 : integer:= 362; --s14 = sin(pi/4);
      constant s38 : integer:= 473;--s38 = sin(3*pi/8);
      constant s18 : integer:= 196; --s18 = sin(pi/8);
      constant s116 : integer:= 100;--s116 = sin(pi/16);
      constant s716 : integer:= 502;--s716 = sin(7*pi/16);
      constant c316 : integer:= 426;--c316 = cos(3 * pi/16);
      constant s316 : integer:= 284;--s316 = sin(3 * pi/16);
```

```
end chen_const;

::::::::::::::
chen_dct/chen_dct.vhd
::::::::::::::
--  This is VHDL implementation of Chen DCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.chen_const.all;

entity chen_dct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end chen_dct;

architecture behave of chen_dct is

type state_type is (IDLE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;

begin
process1: process (clk,rst)
```

```
	variable x0 : signed(31 downto 0);
	variable x1 : signed(31 downto 0);
	variable x2 : signed(31 downto 0);
	variable x3 : signed(31 downto 0);
	variable x4 : signed(31 downto 0);
	variable x5 : signed(31 downto 0);
	variable x6 : signed(31 downto 0);
	variable x7 : signed(31 downto 0);
	variable x8 : signed(31 downto 0);
	variable x9 : signed(31 downto 0);
	variable x10 : signed(31 downto 0);
	variable x11 : signed(31 downto 0);

	variable temp : signed(31 downto 0);

	begin
		if (rst ='0') then
			state <= IDLE;
		elsif (clk='1' and clk'event) then
			case state is
				when IDLE =>
					if start ='1' then
						state <= STAGE1;
					else
						x0 := X"00000000";
						x1 := X"00000000";
						x2 := X"00000000";
						x3 := X"00000000";
						x4 := X"00000000";
						x5 := X"00000000";
						x6 := X"00000000";
						x7 := X"00000000";
						x8 := X"00000000";
						x9 := X"00000000";
						x10 := X"00000000";
						x11 := X"00000000";
						done <= '0';
						state <= IDLE;
					end if;
				when STAGE1 =>
					x0 := resize(in0 + in7, x0'length);
					x1 := resize(in1 + in6, x1'length);
					x2 := resize(in2 + in5, x2'length);
					x3 := resize(in3 + in4, x3'length);
					x4 := resize(in3 - in4, x4'length);
					x5 := resize(in2 - in5, x5'length);
					x6 := resize(in1 - in6, x6'length);
					x7 := resize(in0 - in7, x7'length);
					done <= '0';
					state <= STAGE2;
				when STAGE2 =>
					x8 := x0 + x3;
					x9 := x1 + x2;
					x10 := x1 - x2;
					x11 := x0 - x3;

					x0 := x4(31) & x4(21 downto 0) & "000000000";
					x1 := resize((x6 - x5) * to_signed(s14, 10),
x1'length); -- reikia padauginti is -2
```

```
                              x2 := resize((x6 + x5) * to_signed(s14, 10),
x2'length); -- reikia padauginti is -2
                              --x1 := resize((x6 * to_signed(s14, 10)) - (x5*
to_signed(s14, 10)) , x1'length);
                              --x2 := resize((x6 * to_signed(s14, 10)) + (x5*
to_signed(s14, 10)) , x2'length);
                              x3 := x7(31) & x7(21 downto 0) & "000000000";
                              state <= STAGE3;
                        when STAGE3 =>
                              x4 := x0 + x1;
                              x5 := x0 - x1;
                              x6 := x3 - x2;
                              x7 := x3 + x2;
                              x0 := resize((x8 + x9) * to_signed(s14, 10),
x0'length);
                              x1 := resize((x8 - x9) * to_signed(s14, 10),
x1'length);
                              x2 := resize((x10 * to_signed(s18,10)) + (x11 *
to_signed(s38, 10)), x2'length);
                              x3 := resize((x11 * to_signed(s18,10)) - (x10 *
to_signed(s38, 10)), x3'length);
                              done <= '0';
                              state <= STAGE4;
                        when STAGE4 =>
                              out0 <= x0(31) & x0(24 downto 9);
                              out4 <= x1(31) & x1(24 downto 9);
                              out2 <= x2(31) & x2(24 downto 9);
                              out6 <= x3(31) & x3(24 downto 9);

                              temp := resize((x4 * to_signed(s116, 10)) + (x7 *
to_signed(s716, 10)), temp'length);
                              out1 <= temp(31) & temp(31) & temp(31) & temp(31
downto 18) ;

                              temp := resize((x5 * to_signed(c316, 10)) + (x6 *
to_signed(s316, 10)), temp'length);
                              out5 <= temp(31) & temp(31) & temp(31) & temp(31
downto 18);

                              temp := resize((x6 * to_signed(c316, 10)) - (x5 *
to_signed(c316, 10)), temp'length);
                              out3 <= temp(31) & temp(31) & temp(31) & temp(31
downto 18);

                              temp := resize((x7 * to_signed(s116, 10)) - (x4 *
to_signed(s716, 10)), temp'length);
                              out7 <= temp(31) & temp(31) & temp(31) & temp(31
downto 18);


                              done <= '1';
                              state <= IDLE;
                        when others =>
                              done <= '0';
                              state <= IDLE;
                  end case;
            end if;
end process process1;
```

```
end behave;

::::::::::::::
chen_idct/chen_idct.vhd
::::::::::::::
--  This is VHDL implementation of Chen IDCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.chen_const.all;

entity chen_idct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end chen_idct;

architecture behave of chen_idct is

type state_type is (IDLE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;
```

```
begin
process1: process (clk,rst)
      variable x0 : signed(31 downto 0);
      variable x1 : signed(31 downto 0);
      variable x2 : signed(31 downto 0);
      variable x3 : signed(31 downto 0);
      variable x4 : signed(31 downto 0);
      variable x5 : signed(31 downto 0);
      variable x6 : signed(31 downto 0);
      variable x7 : signed(31 downto 0);
      variable x8 : signed(31 downto 0);
      variable x9 : signed(31 downto 0);
      variable x10 : signed(31 downto 0);
      variable x11 : signed(31 downto 0);
      --variable x12 : signed(31 downto 0);

      begin
            if (rst ='0') then
                  state <= IDLE;
            elsif (clk='1' and clk'event) then
                  case state is
                        when IDLE =>
                              if start ='1' then
                                    done <= '0';
                                    state <= STAGE4;
                              else
                                    x0 := X"00000000";
                                    x1 := X"00000000";
                                    x2 := X"00000000";
                                    x3 := X"00000000";
                                    x4 := X"00000000";
                                    x5 := X"00000000";
                                    x6 := X"00000000";
                                    x7 := X"00000000";
                                    x8 := X"00000000";
                                    x9 := X"00000000";
                                    x10 := X"00000000";
                                    x11 := X"00000000";
                                    done <= '0';
                                    state <= IDLE;
                              end if;
                        when STAGE4 =>
                              x0 := resize(in0, x0'length);
                              x1 := resize(in4, x1'length);
                              x2 := resize(in2, x2'length);
                              x3 := resize(in6, x3'length);

                              x4 := resize((in1 * to_signed(s116, 11)) - (in7 *
to_signed(s716, 11)), x4'length);
                              x5 := resize((in5 * to_signed(c316, 11)) - (in3 *
to_signed(s316, 11)), x5'length);
                              x6 := resize((in3 * to_signed(c316, 11)) + (in5 *
to_signed(s316, 11)), x6'length);
                              x7 := resize((in7 * to_signed(s116, 11)) + (in1 *
to_signed(s716, 11)), x4'length);
                              done <= '0';
                              state <= STAGE3;
                        when STAGE3 =>
                              x8 := resize((x0 + x1) * to_signed(s14,11),
```

```
x8'length);
                              x9 := resize((x0 - x1) * to_signed(s14,11),
x9'length);
                              x10 := resize((x2 * to_signed(s18, 11)) - (x3 *
to_signed(s38, 11)), x10'length);
                              x11 := resize((x3 * to_signed(s18, 11)) + (x2 *
to_signed(s38, 11)), x11'length);

                              x0 := resize(x4(31 downto 1) - x5(31 downto 1),
x1'length);
                              x1 := resize(x7(31 downto 1) - x6(31 downto 1),
x2'length);
                              x4 := resize(x4(31 downto 1) + x5(31 downto 1),
x3'length);
                              x7 := resize(x7(31 downto 1) + x6(31 downto 1),
x4'length);
                              done <= '0';
                              state <= STAGE2;
                        when STAGE2 =>
                              x2 := x9 - x10;
                              x2 := x2(31) & x2(22 downto 0) & "00000000";
                              x3 := resize((x8(31 downto 1) - x11(31 downto 1)),
x4'length);

                              x5 := resize((x1 - x0) * to_signed(s14,11),
x5'length);
                              x6 := resize((x1 + x0) * to_signed(s14,11),
x5'length);

                              x0 := resize((x8(31 downto 1) + x11(31 downto 1)),
x0'length);
                              x1 := x9 + x10;
                              x1 := x1(31) & x1(22 downto 0) & "00000000";
                              done <= '0';
                              state <= STAGE1;
                        when STAGE1 =>
                              --out0 <= resize((x0(31) & x0(26 downto 12)) +
(x7(31) & x7(26 downto 12)), out0'length);
                              --out7 <= resize((x0(31) & x0(26 downto 12)) -
(x7(31) & x7(26 downto 12)), out1'length);
                              --out3 <= resize((x3(31) & x3(26 downto 12)) +
(x4(31) & x4(26 downto 12)), out2'length);
                              --out4 <= resize((x3(31) & x3(26 downto 12)) -
(x4(31) & x4(26 downto 12)), out3'length);
                              --out1 <= resize((x1(31) & x1(26 downto 12)) +
(x6(31) & x6(26 downto 12)), out4'length);
                              --out6 <= resize((x1(31) & x1(26 downto 12)) -
(x6(31) & x6(26 downto 12)), out5'length);
                              --out2 <= resize((x2(31) & x2(26 downto 12)) +
(x5(31) & x5(26 downto 12)), out6'length);
                              --out5 <= resize((x2(31) & x2(26 downto 12)) -
(x5(31) & x5(26 downto 12)), out7'length);
                              x11 := x0 + x7;
                              out0 <= x11(31) & x11(25 downto 10);

                              x11 := x0 - x7;
                              out7 <= x11(31) & x11(25 downto 10);

                              x10 := x3 + x4;
```

```
                              out3 <= x10(31) & x10(25 downto 10);

                              x10 := x3 - x4;
                              out4 <= x10(31) & x10(25 downto 10);

                              x9 := x1 + x6;
                              out1 <= x9(31) & x9(31) & x9(31) & x9(31) & x9(31) &
x9(30 downto 19);

                              x9 := x1 - x6;
                              out6 <= x9(31) & x9(31) & x9(31) & x9(31) & x9(31) &
x9(30 downto 19);

                              x8 := x2 + x5;
                              out2 <= x8(31) & x8(31) & x8(31) & x8(31) & x8(31) &
x8(30 downto 19);

                              x8 := x2 - x5;
                              out5 <= x8(31) & x8(31) & x8(31) & x8(31) & x8(31) &
x8(30 downto 19);

                              done <= '1';
                              state <= IDLE;
                         when others =>
                              done <= '0';
                              state <= IDLE;
                    end case;
               end if;
end process process1;

end behave;
::::::::::::::
loe_dct/loe_dct_const.vhd
::::::::::::::
--  This is VHDL file with constants for Loeffler DCT/IDCT algorithm
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

package loe_dct_const is

      constant c1 : integer:=1004; -- cos(pi/16)<<10 */
      constant s1 : integer:=200; -- sin(pi/16)<<10 */
      constant c3 : integer:=851; -- cos(3pi/16)<<10 */
      constant s3 : integer:=569; -- sin(3pi/16)<<10 */
      constant r2c6 : integer:=554; -- sqrt(2)*cos(6pi/16)<<10 */
      constant r2s6 : integer:=1337;
      constant r2 : integer:=181;   -- sqrt(2)<<7 */
```

```
end loe_dct_const;
::::::::::::::
loe_dct/dct.vhd
::::::::::::::
--  This is VHDL implementation of Loeffler DCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.loe_dct_const.all;

entity dct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end dct;

architecture behave of dct is

type state_type is (IDLE, RESIZE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;

begin
process1: process (clk,rst)
```

```
variable x0 : signed(31 downto 0);
variable x1 : signed(31 downto 0);
variable x2 : signed(31 downto 0);
variable x3 : signed(31 downto 0);
variable x4 : signed(31 downto 0);
variable x5 : signed(31 downto 0);
variable x6 : signed(31 downto 0);
variable x7 : signed(31 downto 0);
variable x8 : signed(31 downto 0);
variable x9 : signed(31 downto 0);
variable x10 : signed(31 downto 0);
variable x11 : signed(31 downto 0);

variable temp : signed(31 downto 0);

begin
      if (rst ='0') then
            state <= IDLE;
      elsif (clk='1' and clk'event) then
            case state is
                  when IDLE =>
                        if start ='1' then
                              state <= RESIZE;
                        else
                              x0 := X"00000000";
                              x1 := X"00000000";
                              x2 := X"00000000";
                              x3 := X"00000000";
                              x4 := X"00000000";
                              x5 := X"00000000";
                              x6 := X"00000000";
                              x7 := X"00000000";
                              x8 := X"00000000";
                              x9 := X"00000000";
                              x10 := X"00000000";
                              x11 := X"00000000";
                              done <= '0';
                              state <= IDLE;
                        end if;
                  when RESIZE =>
                        x0 := resize (signed (in0), x0'length);
                        x1 := resize (signed (in1), x1'length);
                        x2 := resize (signed (in2), x2'length);
                        x3 := resize (signed (in3), x3'length);
                        x4 := resize (signed (in4), x4'length);
                        x5 := resize (signed (in5), x5'length);
                        x6 := resize (signed (in6), x6'length);
                        x7 := resize (signed (in7), x7'length);
                        x8 := "00000000000000000000000000000000";
                        state <= STAGE1;
                  when STAGE1 =>
                        x8:=x7+x0; x0:=x0-x7;  x7:=x1+x6; x1:=x1-x6;
                        x6:=x2+x5; x2:=x2-x5;  x5:=x3+x4; x3:=x3-x4;
                        done <= '0';
                        state <= STAGE2;
                  when STAGE2 =>
                        x4:=resize(x8+x5, x4'length);
                        x8:=resize(x8-x5, x8'length);
                        x5:=resize(x7+x6, x5'length);
```

```
                              x7:=resize(x7-x6, x7'length);
                              x6:=resize(to_signed(c1,12) * (x1+x2), x6'length);
--20bitu
                              x2:=resize(to_signed((-s1-c1),12) * x2+x6,
x2'length);    --32bitas
                              x1:=resize(to_signed((s1-c1),12) * x1+x6, x1'length);
                              x6:=resize(to_signed(c3,12) * (x0+x3), x6'length);
--20bitu
                              x3:=resize(to_signed((-s3-c3),12) * x3+x6,
x3'length); --32bitai
                              x0:=resize(to_signed((s3-c3),12) * x0+x6, x0'length);
      --32bitai
                              state <= STAGE3;
                        when STAGE3 =>
                              x6:=x4+x5;
                              x4:=x4-x5;
                              x5:=resize(to_signed(r2c6,13) * x7+x8, x5'length);
                              x7:=resize(to_signed(-r2s6-r2c6,14) * x7+x5,
x7'length);
                              x8:=resize(to_signed(r2s6-r2c6,14) * x8+x5,
x8'length);
                              x5:=x0+x2;
                              x0:=x0-x2;
                              x2:=x3+x1;
                              x3:=x3-x1;
                              done <= '0';
                              state <= STAGE4;
                        when STAGE4 =>
                              out0 <= x6(16 downto 0);
                              --out0.write(x6);

                              out4 <= x4(16 downto 0);
                                          --out4.write(x4);

                              temp := x8 + 512;
                              --temp <= "0000000000" & temp(31 downto 10);
                              --out2 <= temp(16 downto 0);
                              out2 <= temp(31) & temp(25 downto 10);
                                          --out2.write((x8+512)>>10);

                              temp := x7 + 512;
                              out6 <= temp(31) & temp(25 downto 10);
                                          --out6.write((x7+512)>>10);

                              temp := x2 - x5 + 512;
                              out7 <= temp(31) & temp(25 downto 10);
                                          --out7.write((x2-x5+512)>>10)

                              temp := x2 + x5 + 512;
                              out1 <= temp(31) & temp(25 downto 10);
                                          --out1.write((x2+x5+512)>>10);

                              temp := resize(x3 * to_signed (r2, 12),32) + 65536;
                              if temp(31) = '0' then
                                    temp := "00000000000000000" & temp(30 downto
17);
                              else
                                    temp := "11111111111111111" & temp(30 downto
17);
```

```
                              end if;
                              out3 <= temp(16 downto 0);

                                    --out3.write((x3*r2+65536)>>17);

                              temp := resize(x0 * to_signed(r2, 12), 32) + 65536;
                              if (temp(31) = '0') then
                                    temp := "0000000000000000" & temp(30 downto
17);
                              else
                                    temp := "1111111111111111" & temp(30 downto
17);
                              end if;
                              out5 <= temp(16 downto 0);
                              done <= '1';
                              state <= IDLE;
                        when others =>
                              done <= '0';
                              state <= IDLE;
                  end case;
            end if;
end process process1;

end behave;

::::::::::::::
loe_idct/idct.vhd
::::::::::::::
--  This is VHDL implementation of Loeffler IDCT algorithm.
--  Copyright (C) 2011  Darius Birvinskas, Ignas Martišius
--
--  This program is free software: you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.loe_idct_const.all;

entity idct is
      port(
      clk : in std_logic;
      rst : in std_logic;
      start : in std_logic;
      in0 : in signed(16 downto 0);
      in1 : in signed(16 downto 0);
      in2 : in signed(16 downto 0);
      in3 : in signed(16 downto 0);
      in4 : in signed(16 downto 0);
```

```
      in5 : in signed(16 downto 0);
      in6 : in signed(16 downto 0);
      in7 : in signed(16 downto 0);

      done : out std_logic;
      out0 : out signed(16 downto 0);
      out1 : out signed(16 downto 0);
      out2 : out signed(16 downto 0);
      out3 : out signed(16 downto 0);
      out4 : out signed(16 downto 0);
      out5 : out signed(16 downto 0);
      out6 : out signed(16 downto 0);
      out7 : out signed(16 downto 0)
      );
end idct;

architecture behave of idct is

type state_type is (IDLE, RESIZE, STAGE1, STAGE2, STAGE3, STAGE4);
signal state: state_type ;

begin
process1: process (clk,rst)
      variable x0 : signed(31 downto 0);
      variable x1 : signed(31 downto 0);
      variable x2 : signed(31 downto 0);
      variable x3 : signed(31 downto 0);
      variable x4 : signed(31 downto 0);
      variable x5 : signed(31 downto 0);
      variable x6 : signed(31 downto 0);
      variable x7 : signed(31 downto 0);
      variable x8 : signed(31 downto 0);
      variable x9 : signed(31 downto 0);
      variable x10 : signed(31 downto 0);
      variable x11 : signed(31 downto 0);

      variable temp : signed(31 downto 0);

      begin
            if (rst ='0') then
                  state <= IDLE;
            elsif (clk='1' and clk'event) then
                  case state is
                        when IDLE =>
                              if start ='1' then
                                    state <= RESIZE;
                              else
                                    x0 := X"00000000";
                                    x1 := X"00000000";
                                    x2 := X"00000000";
                                    x3 := X"00000000";
                                    x4 := X"00000000";
                                    x5 := X"00000000";
                                    x6 := X"00000000";
                                    x7 := X"00000000";
                                    x8 := X"00000000";
                                    x9 := X"00000000";
                                    x10 := X"00000000";
                                    x11 := X"00000000";
```

```
                        done <= '0';
                        state <= IDLE;
                  end if;
            when RESIZE =>
                  x0 := resize (signed (in0), x0'length);
                  x1 := resize (signed (in1), x1'length);
                  x2 := resize (signed (in2), x2'length);
                  x3 := resize (signed (in3), x3'length);
                  x4 := resize (signed (in4), x4'length);
                  x5 := resize (signed (in5), x5'length);
                  x6 := resize (signed (in6), x6'length);
                  x7 := resize (signed (in7), x7'length);
                  x8 := "00000000000000000000000000000000";
                  state <= STAGE1;
            when STAGE1 =>
                  x0 := x0(31) & x0(21 downto 0) & "000000000";

                  x1 := x1(31) & x1(23 downto 0) & "0000000";

                  --x2 := resize (signed (in2), x2'length);

                  x3 := resize (x3 * r2, x3'length);
                  x4 := x4(31) & x4(21 downto 0) & "000000000";

                  x5 := resize (x5 * r2, x5'length);
                  --x6 := resize (signed (in6), x6'length);

                  x7 := x7(31) & x7(23 downto 0) & "0000000";

                  x8 := x7 + x1;
                  x1 := x1 - x7;
                  done <= '0';
                  state <= STAGE2;
            when STAGE2 =>
                  x7:=x0+x4;
                  x0:=x0-x4;
                  x4:=x1+x5;
                  x1:=x1-x5;
                  x5:=x3+x8;
                  x8:=x8-x3;
                  x3:=resize(to_signed(r2c6,13) * (x2+x6), x3'length);
                  x6:=resize(x3 + (to_signed(-r2c6-r2s6,14) * x6),
x6'length);
                  x2:=resize(x3 + (to_signed(-r2c6+r2s6,14) * x2),
x2'length);
                  done <= '0';
                  state <= STAGE3;
            when STAGE3 =>
                  x3:=resize(x7+x2, x3'length);
                  x7:=resize(x7-x2, x7'length);
                  x2:=resize(x0+x6, x2'length);
                  x0:=resize(x0-x6, x0'length);
                  x6:=resize(to_signed(c3,12) * (x4+x5), x6'length);
--20bitu

                  x5:= resize(x6 + (to_signed((-c3-s3),12) * x5),
x5'length);
                  if (x5(31) = '0') then
                        x5:= "0000000" & x5(30 downto 6);
```

```
                        else
                              x5:= "1111111" & x5(30 downto 6);
                        end if;

                        x4:= resize(x6 + (to_signed((-c3+s3),12) * x4),
x4'length);
                        if (x4(31) = '0') then
                              x4:= "0000000" & x4(30 downto 6);
                        else
                              x4:= "1111111" & x4(30 downto 6);
                        end if;

                        x6:=resize(to_signed(c1,12) * (x1+x8), x6'length);

                        x1:= resize(x6 + (to_signed((-c1-s1),12) * x1),
x1'length);
                        if (x1(31) = '0') then
                              x1:= "0000000" & x1(30 downto 6);
                        else
                              x1:= "1111111" & x1(30 downto 6);
                        end if;

                        x8:= resize(x6 + (to_signed((-c1+s1),12) * x8),
x8'length);
                        if (x8(31) = '0') then
                              x8:= "0000000" & x8(30 downto 6);
                        else
                              x8:= "1111111" & x8(30 downto 6);
                        end if;
                        done <= '0';
                        state <= STAGE4;
                  when STAGE4 =>
                        x7:= x7 + 512;
                        x2:= x2 + 512;
                        x0:= x0 + 512;
                        x3:= x3 + 512;

                        temp := x3 + x4;
                        out0 <= temp(31) & temp(27 downto 12);

                        temp := x2 + x8;
                        out1 <= temp(31) & temp(27 downto 12);

                        temp := x0 + x1;
                        out2 <= temp(31) & temp(27 downto 12);

                        temp := x7 + x5;
                        out3 <= temp(31) & temp(27 downto 12);

                        temp := x7 - x5;
                        out4 <= temp(31) & temp(27 downto 12);

                        temp := x0 - x1;
                        out5 <= temp(31) & temp(27 downto 12);

                        temp := x2 - x8;
                        out6 <= temp(31) & temp(27 downto 12);

                        temp := x3 - x4;
```

```
                              out7 <= temp(31) & temp(27 downto 12);

                              done <= '1';
                              state <= IDLE;
                         when others =>
                              done <= '0';
                              state <= IDLE;
                    end case;
               end if;
end process process1;

end behave;
```