DIPLOMITYÖ

MIKKO TÖRNQVIST

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

Mikko Törnqvist

X.264 Videokoodekin toteutus korkean tason synteesityökaluja käyttäen

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 25.5.2011

Työn valvoja

Professori Kari Halonen

Työn ohjaaja

TkT Lauri Koskinen

AALTO-YLIOPISTO **DIPLOMITYÖN TIIVISTELMÄ**
SÄHKÖTEKNIIKAN KORKEAKOULU

Tekijä:	Mikko Törnqvist
Työn nimi:	X.264 videokoodekin toteutus korkean tason synteesillä
Päivämäärä:	25.5.2011
Sivumäärä:	84
Laitos:	Mikro- ja nanotekniikan laitos
Professuuri:	S-87 Piiritekniikka
Työn valvoja:	Professori Kari Halonen
Työn ohjaaja:	TkT Lauri Koskinen

Digitaalisten piirien suunnittelu on jatkuvasti muuttumassa monimutkaisemmaksi. Perinteinen suunnittelu VHDL-ohjelmointikielellä on osoittautumassa liian hitaaksi ja tarvitaan korvaavia menetelmiä algoritmien nopeaan toteutukseen.

Tässä diplomityössä on tutkittu pystyisikö Mentor Graphicsin Catapult C-ohjelmalla tähän nopeampaan suunnitteluun. On tutkittu lisäksi millaisia ominaisuuksia X.264 –videokoodekista voitaisiin sisällyttää lopulliseen kannettavalla laitteella toteutettavaksi aiottuun tuotteeseen. Videokoodaukseen perehtymättömille on kuitenkin aluksi kuvattu yleisluonteisesti X.264-videokoodekin toimintaa.

Käytössä olevan Catapult C-ohjelman opiskelijaversion teknologiakirjastoilla FPGA-piirissä toimivan X.264 videokoodekin luominen kuitenkin epäonnistui. Ohjelmisto ei pystynyt luomaan testattavaksi laitteeksi valitun FPGA-piirin kanssa yhteensopivaa VHDL-kuvausta. Voitiin kuitenkin todeta, että paremmilla teknologiakirjastoilla ohjelmisto on lupaava: VHDL-kieltä tuntematon pystyy hyvin nopeasti rakentamaan algoritmeista valmiin lopputuotteen. Sen avulla pystyy myös luomaan samaan aikaan fyysisen tason laitekuvauksen ja ohjelmistotason simulaatiomallin, joten toiminnallisuuden varmistaminen on helppoa ja yksinkertaista.

Avainsanat: X.264, videokoodaus, MPEG-4, VHDL, Catapult

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE MASTER'S THESIS

Author :	Mikko Törnqvist
Name of the thesis:	High-Level Synthesis and Implementation of the X.26 Video Codec
Date:	25.5.2011
Number of pages:	84
Department:	Department of Micro- and Nanosciences
Professorship:	S-87 Electronic Circuit Design
Supervisor:	Professor Kari Halonen
Instructor:	D.Sc. (Tech.) Lauri Koskinen

The design process of digital circuits is becoming increasingly complex. Designing of digital circuits has traditionally employed HDL descriptions written manually using a design language such as VHDL. However, this method is proving to be too slow and alternative methods are required for rapid implementation of algorithms on hardware.

This master's thesis investigates whether Catapult C software by Mentor Graphics, Inc. is a fit tool for this task. Furthermore, it investigates what kind of properties of the X.264 video codec could be included in the end product, which was estimated to have the resources of a PDA. In the beginning of this thesis the operation of X.264 video codec is described for the benefit of those not familiar with video codecs.

As a result of this investigation, it was established that the Catapult Student version and its technology libraries were unable to create a working X.264 codec implementation on a FPGA. The software's technology libraries were unfit for creating a VHDL description compatible with the chosen FPGA model. However, it was clearly demonstrated that with better technology libraries the software shows a great deal of promise: a user with no knowledge of VHDL is capable of rapidly implementing various algorithms on a final end product. Catapult C also enables co-design of both software and hardware, which makes simulation and verification of the algorithms simple and easy.

Keywords: X.264, Video encoding, MPEG-4, VHDL, Catapult

Alkulause

Tämä diplomityö on tehty Aalto-yliopiston sähkötekniikan korkeakoulun mikro- ja nanotekniikan laitoksen piiritekniikan laboratoriossa osana laboratorion projektia Mentor Graphicsin Catapult-ohjelman käytettävyyden tutkimukseksi.

Kiitän työn valvojaa prosessori Kari Halosta mahdollisuudesta työskennellä piiritekniikan laboratoriossa. Erityisesti haluan kiittää työn ohjaaja TkT Lauri Koskista kärsivällisyydestä, jota hän on osoittanut työni aikataulun venymistä kohtaan.

Kiitokset tämän työn edistämisestä kuuluvat myös DI Jakub Groniczille, DI Kari Kokkiselle, DI Vesa Turuselle ja tekn.yo. Mika Pulkkiselle, jotka osaltaan auttoivat FPGA-piirien käyttämisessä.

Espoossa 25.5.2011

Mikko Törnqvist

Sisällysluettelo

Symboli- ja lyhenneluettelo

DCT	(Discrete Cosine Transform) Diskreetti kosinimuunnos
Ennustajaliike-vektori	(Motion vector predictor) Etukäteen laskettu liikevektori, jota käytetään uuden liikevektorin etsinnässä.
IDCT	(Inverse Discrete Cosine Transform) Käänteinen Diskreetti kosinimuunnos
Kuva	(engl. frame) Tietystä määrästä pikseleitä koostuva joukko, joka näytetään kerralla videon katsojalle. Tavallisesti sekuntiin videota mahtuu 25 kuvaa. Kuvasta käytetään myös englanninkielisestä termistä frame suoraan käännettyä termiä kehys.
Kvantisointi-parametri	Kvantisointiparametri (qp) määrittelee miten pikseliarvot kvantisoidaan. Mitä suurempi kvantisointiparametri, sen huonolaatuisempaa mutta pienempään tilaan mahtuvaa videokuvaa saadaan.
Liikevektori	(Motion vector, mv) Vektoriarvo, joka osoittaa kuinka paljon tietyn kuvan makrolohko on liikkunut kuvasta toiseen
Liikkeen-estimointi	(Motion estimation) Liikkeenestimoinnissa etsitään parhaiten koodattavaa makrolohkoa vastaava makrolohko referenssikuvasta.
Liikkeen-kompensaatio	(Motion compensation) Liikkeenkompensoinnissa vähennetään liikkeenestimoinnissa löydetty parhaiten vastaava makrolohko enkoodattavasta makrolohkosta. Tällöin syntyy ns. residuaalimakrolohko.

Makrolohko	H.264 -standardissa 16x16 pikselistä koostuva alue kuvassa
Pikseli	(engl. pixel, videokoodekeissa myös pel) on kuvan pienin yksittäinen osa. Se määrittää pienellä rajallisella alueella kuvan värin.
Täyspikseli Alipikseli Puolipikseli Neljännes-pikseli	Täyspikselit ovat suoraan kuvasta määriteltyjä pikseleitä. Puolipikselit saadaan painotettuna keskiarvona ympäröivistä täyspikseleistä sekä neljännespikselit puolestaan keskiarvona ympäröivistä puolipikseleistä ja täyspikseleistä. Puolipikseleistä ja neljännespikseleistä käytetään yhdessä nimitystä alipikseli. Niitä tarvitaan usein parempaan liikkeenestimointiin, sillä videossa liike ei yleensä noudata tarkasti pikselirajoja.
QCIF-video	176x144 pikselin videokuva
RGB	RGB-väritilassa väri määritellään näytöllä esitettävänä punaisen (R:red), vihreän (G:green) ja sinisen (B:blue) kanavan arvona (kukin kanava saa 24-bittisessä RGB-esityksessä jonkun arvon väliltä 0 - 255). Tässä tapauksessa musta pikseli on arvoltaan (0,0,0) ja valkoinen (255,255,255).
SQCIF-video	128x96 pikselin videokuva
SAD	(Sum of absolute differences) Absoluuttisten virheiden summa
SATD	(Sum of absolute transformed differences) Muunnettujen absoluuttisten erojen summa
SSD	(Sum of squared differences) Erojen neliöiden summa
Viipale	(engl. slice) Peräkkäisten makrolohkojen joukko kuvassa, joita käsitellään yhtenä yksikkönä videokoodauksessa. Niitä voi olla yhdessä kuvassa useita, mutta tässä diplomityössä on yksinkertaistuksen vuoksi oletettu yhteen kuvaan vain yksi viipale.

YUV	YUV-väritilassa väri määritellään luminanssin Y ja kahden krominanssiarvon avulla. Tämän tilan etuna RGB-väritilaan on se, että ihmissilmä on vähemmän herkkä krominanssiarvojen muutoksille jolloin krominanssiarvot voidaan yleensä tallentaa pienemmällä näytemäärällä ja näin säästää tilaa.

1. **Johdanto**

Digitaalisten piirien suunnittelu on muuttumassa yhä monimutkaisemmaksi. Samalla valmiiden tuotteiden elinkaari on yhä lyhyempi, joten suunnittelusta on tulossa yhä lyhytjännitteisempää ja hankalampaa. Tarvitaan nopeita työkaluja, jotka pystyvät muuttamaan suunnittelijan ideat nopeasti valmiiksi tuotteiksi.

1990-luvulla nopeat työkalut tarkoittivat VHDL-kehitystyökaluja. VHDL-kielessä kuvataan miten tietyn rekisterin arvo muuttuu toiseksi tietyllä kellojaksolla. Näiden työkalujen ansiosta ei enää tarvinnut välittää transistorien toimintojen yksityiskohdista ja ominaisuuksista digitaalipiireissä, sillä kehitystyökalut pystyivät varmistamaan automaattisesti digitaalipiirin toiminnan. Tämä nopeutti valtavasti suunnittelua verrattuna aikoihin, jolloin suunnittelijan piti piirtää transistorikaaviota itse.

Nykyään VHDL-kehitystyökalutkaan eivät tahdo olla riittävän nopeita. Jos esimerkiksi halutaan muuttaa ohjelmassa kokonaisten funktioiden toimintaa, se ei käy riittävän nopeasti VHDL-kuvauskielellä. Suunnittelija joutuu pilkkomaan ensin funktion kellojakson pituisiin osiin, joissa kussakin voidaan muuttaa tiettyjen rekisterien arvoja. Tämä manuaalinen muutosprosessi on yksinkertaisesti liian hidasta nykyajan monimutkaisille suunnitteluprojekteille.

Ratkaisuksi eri ohjelmistovalmistajat ovat tarjoamassa omia ratkaisujaan. FPGA-piirien valmistajat ovat luoneet omia kehitystyökalujaan, joilla pystyy graafisesti suunnittelemalla saamaan tietyn idean muutettua nopeasti toteuttamiskelpoiseksi. Esimerkiksi tunnettu FPGA-piirien valmistaja Altera on luonut oman ohjelmistonsa nimeltä SoPC Builder.

Useat muut yritykset ovat luoneet ohjelmistoja, joilla voidaan muuntaa perinteisiä ohjelmistokielillä tehtyjä ohjelmia laitteistokuvauskielille kuten VHDL:ksi tai Verilog:ksi. Näiden ohjelmistojen etuna on se, että funktioita voi muuttaa hyvin nopeasti ja saada aikaan toimiva ratkaisu. Haittapuolena on tietenkin se, että suunnittelijalta piilossa ohjelmisto joutuu tekemään monimutkaisia ratkaisuja ja päättämään toteutuksen yksityiskohdista. Siten suunnittelijan täytyy olla hyvin tarkkana, että valmis ohjelma todella toteuttaa hänen aikomansa ideat ja toteutus pysyy kellojakson sekä muiden rajoitteiden asettamissa rajoissa.

Tässä diplomityössä testattiin erästä näistä C-ohjelmointikieleen perustuvista ohjelmistoista, Mentor Graphicsin Catapult C-ohjelmistoa. Tarkoituksena oli testata miten PC:llä ohjelmistona ajettavaksi suunnitellun X.264 -videokoodekin muunto laitteistopohjaiseksi toteutukseksi onnistuisi. Lisäksi tutkittiin millaisia ominaisuuksia X.264 –videokoodekista voitaisiin sisällyttää lopulliseen kannettavalla laitteella toteutettavaksi aiottuun tuotteeseen.

Diplomityön aluksi käsitellään siksi ensin videokoodauksen perusteita johdannoksi niihin perehtymättömille. Sen jälkeen esitellään tarkemmin valitun X.264 –videokoodekin erityisominaisuuksia ja sen suoritusarvoja. Myöhemmin esitellään mitä kehitystyökaluja käytettiin ja lopuksi kerrotaan miten valmiin toteutuksen testaus FPGA-piirillä onnistui.

2. **Videokoodauksen perusteita**

Videokoodausta käytetään nykyään monenlaisissa laitteissa. Esimerkiksi kannettavissa tietokoneissa, kameroissa ja matkapuhelimissa on hyvin usein jonkintasoinen videokamera ja luonnollisesti siten myös videokoodauksen toteutus joko laitteistona tai ohjelmistona.

Matkapuhelimissa toteutus on usein laitteistopohjainen vaikkapa erillisenä mikropiirinä. Koska matkapuhelimissa on nykyisin useita muitakin ominaisuuksia videokoodauksen lisäksi, tämän videokoodauksen toteuttavan mikropiirin suunnittelusta vastaa useimmiten alihankkija. Alihankkija joutuu toteuttamaan mikropiirinsä usein melko kireässä aikataulussa ja jatkuvasti muuttuvien spesifikaatioiden paineessa. Siksi videokoodauksen toteuttaminen käy hyvin esimerkiksi projektista, jossa tarvittaisiin korkean tason synteesityökaluja jatkuvasti muuttuvien algoritmien nopeaan toteuttamiseen fyysisenä laitteena.

Videokoodaukseen voidaan kuitenkin käyttää monenlaisia videokoodekkeja. Tässä työssä on perehdytty X.264-videokoodekkiin, joka perustuu makrolohkopohjaiseen liikkeenestimoinnin. Videokoodekit ovat kuitenkin muuttumassa yhä monimutkaisemmiksi laskentatehon kasvaessa – tulevaisuudessa käytettäneen hahmopohjaista liikkeenestimointia ja muita kehittyneempiä menetelmiä. Suunnittelu vaatii siis nyt ja tulevaisuudessa avukseen työkaluja, joilla voidaan arvioida riittääkö laskentateho valitun videokoodekin kaikkien ominaisuuksien toteuttamiseen.

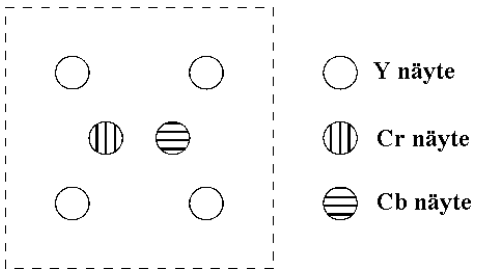
Kaikki videokoodekit kuitenkin perustuvat samoihin perusvaatimuksiin. Pikseleistä koostuvat videoinformaatio on pakattava mahdollisimman pieneen tilaan annettujen laatuvaatimusten rajoissa. Videokoodekkeihin perehtymättömien avuksi diplomityön tässä kappaleessa on siksi käyty läpi videokoodauksen yleisiä perusteita.

2.1. Kuva ja pikseli

Videosignaali koostuu kuvista, joita esitetään nopeassa järjestyksessä (tavallisimmin 25-30 kpl/sekunnissa). Videosignaalin yhteydessä voidaan käyttää myös nimitystä kehys (englanniksi frame) puhuttaessa kuvista. Yksittäinen kuva puolestaan koostuu pikseleistä, joiden määrän perusteella kuvalla on tietty leveys ja korkeus. Esimerkiksi QCIF-videosignaali on 176 pikseliä leveä ja 144 pikseliä korkea. Pikseliä voidaan pitää värinäytteenä kuvan tietyssä osassa, se siis määrittää minkä väriseltä kuvan tietyn rajallisen alueen pitäisi näyttää. Väri voidaan esittää esim. RGB-arvojen avulla, jolloin väri määritellään näytöllä esitettävänä punaisen (R:red), vihreän (G:green) ja sinisen (B:blue) kanavan arvona (kukin kanava saa 24-bittisessä RGB-esityksessä jonkun arvon väliltä 0 -255). Tässä tapauksessa musta pikseli on arvoltaan (0,0,0) ja valkoinen (255,255,255).

2.2. YUV-esitys ja YUV 4:2:0 näytteistys

Ihmisen näköjärjestelmä on suunnilleen yhtä herkkä jokaiselle RGB-komponentille, joten jos käytetään RGB-väriesitystä, pitää kukin RGB-komponenteista tallentaa samalla tarkkuudella. Väri voidaan kuitenkin esittää muinakin kuin RGB-esityksenä, esimerkiksi voidaan käyttää luminanssi- ja krominanssiarvojen yhdistelmää (YUV). Tällä esitysmuodolla on se etu, että ihmissilmä on paljon herkempi luminanssille kuin krominanssiarvoille. Krominanssiarvoja ei siis tarvitse tallentaa yhtä suurta määrää kuin luminanssiarvoja ja näin voidaan säästää tallennettavan datan määrässä. Tavallisesti käytetään YUV 4:2:0 näytteistystä, jossa tallennetaan vain yksi punakrominanssin Cr-arvo (U) ja yksi sinikrominanssin Cb-arvo (V) kutakin neljää luminanssin arvoa (Y) kohden. Krominanssiarvoja on näin puolet luminanssiarvojen leveydestä ja puolet niiden korkeudesta.



Kuva 1: YUV 4:2:0 näytteistys (laadittu kirjallisuuden [1] perusteella)

2.3. Makrolohkot ja alilohkot

Edellä esitettyjä pikseleitä on myös tapana yhdistellä isommiksi, esimerkiksi 16x16 pikselin ryhmiksi, joita kutsutaan makrolohkoiksi. Tällainen ryhmittely makrolohkoiksi on tarpeen, jotta videokoodekissa voidaan vertailla sopivan kokoisia alueita toisiinsa. Jos vastaavuus on hyvä, tarvitsee alkuperäisestä makrolohkosta ehkä tallentaa vain tieto vertailtavan makrolohkon suhteellisesta sijainnista alkuperäiseen makrolohkoon nähden (ns. liikevektori). Alkuperäisiä pikseliarvoja ei siis tarvitsisi laisinkaan tallentaa, ja tallennettavan datan määrä vähenisi merkittävästi.

H.264 – videokoodekissa ja siihen perustuvassa X.264-videokoodekissa on päädytty käyttämään 16x16 pikselin makrolohkoja, mutta muutkin valinnat ovat mahdollisia. Käytettäessä isoja makrolohkoja voisi siis mahdollisesti kuitata monta pikseliä vain yhdellä liikevektorilla. Toisaalta isoilla makrolohkoilla voi olla vaikeaa löytää hyvää vastaavuutta toiseen isoon makrolohkoon, koska liike eri kuvien välillä harvoin tapahtuu täsmällisesti yhdestä makrolohkosta toiseen. Parempi vastaavuus voidaan saada jakamalla iso makrolohko useampaan pieneen alilohkoon ja tallentamalla kustakin oma liikevektorinsa. Esim. 16x16 makrolohko voidaan jakaa kahteen 16x8 pikselin alilohkoon tai neljään 8x8 pikselin alilohkoon. Videokoodekin tehtävänä on päättää millainen jako kannattaa.

2.4. Videosignaalin laadun arviointi

Videosignaalin laatua voidaan mitata sekä subjektiivisesti että objektiivisesti. Objektiivisessa arvioinnissa koodataan ja dekoodataan signaali videokoodekin standardin mukaisesti ja vertaillaan sitä sitten laskennallisesti alkuperäiseen signaaliin. Usein käytetään huippuarvoa signaalikohina-suhteesta (peak signal-to-noise ratio, PSNR).

Objektiivinen laadunmittaus laskennallisesti ei kuitenkaan välttämättä korreloi kovinkaan hyvin testiyleisön subjektiiviseen kokemukseen videosignaalin laadusta [1]. Siksi on kehitetty myös subjektiivisia laadunmittauksen testausmenetelmiä. Niissä on kuitenkin ehkä jopa objektiivisia laadunmittausmenetelmiä suurempia ongelmia: koehenkilön näkökokemukseen vaikuttaa hänen mielialansa, ympäristönsä ja jopa muut aikaisemmin nähdyt videosignaalit [1]. Näiden seikkojen vuoksi tällä hetkellä parhaimpana käytettävissä olevana laadunarviointimenetelmänä pidetään jonkinlaista subjektiivisten ja objektiivisten laadunarviointimenetelmien yhdistelmää [2].

Laadun arvioinnissa on luonnollisesti otettava huomioon myös koodatun videosignaalin bittinopeus. Nostamalla bittinopeutta epärealistisiin lukemiin saadaan luonnollisesti todella hyvälaatuista videosignaalia, mutta sen tallentaminen ja lähettäminen etenkin langattomissa tietoliikenneverkoissa muodostuu epäkäytännölliseksi. Siksi tässäkin diplomityössä pyritään korostamaan laadun (PSNR) ja bittinopeuden suhdetta, eikä niinkään ko. arvoja erikseen.

2.4.1 PSNR

Huippuarvo signaalikohina-suhteesta (peak signal-to-noise ratio, PSNR) on yleisesti käytetty objektiivinen mitta. Se lasketaan vertailtavan kuvan ja referenssikuvan pikseliarvojen erojen neliöiden summan keskiarvon (mean squared error, MSE) avulla:

$$\boldsymbol{MSE = \frac{1}{mn}\sum_{x=0}^{m-1}\sum_{y=0}^{n-1}\left[A_{x,y} - B_{x,y}\right]^2} \quad \textbf{(1)}$$

missä $A_{x,y}$ on vertailtavan kuvan m x n -näytematriisin A alkio ja $B_{x,y}$ referenssikuvan mxn näytematriisin B alkio.

Itse PSNR lasketaan sitten edelleen seuraavasti:

$$\boldsymbol{PSNR_{dB} = 10log_{10}\frac{\left(2^{bits}-1\right)^2}{MSE}} \quad \textbf{(2)}$$

missä *bits* on kuvan pikseleissä käytetty bittimäärä ja 2^{bits}*-1* siten suurin mahdollinen pikseliarvo. Tyypillisessä videosignaalissa PSNR on yleensä noin 20-40 dB [3]. Yllä olevat kaavat perustuvat kirjallisuuteen [1].

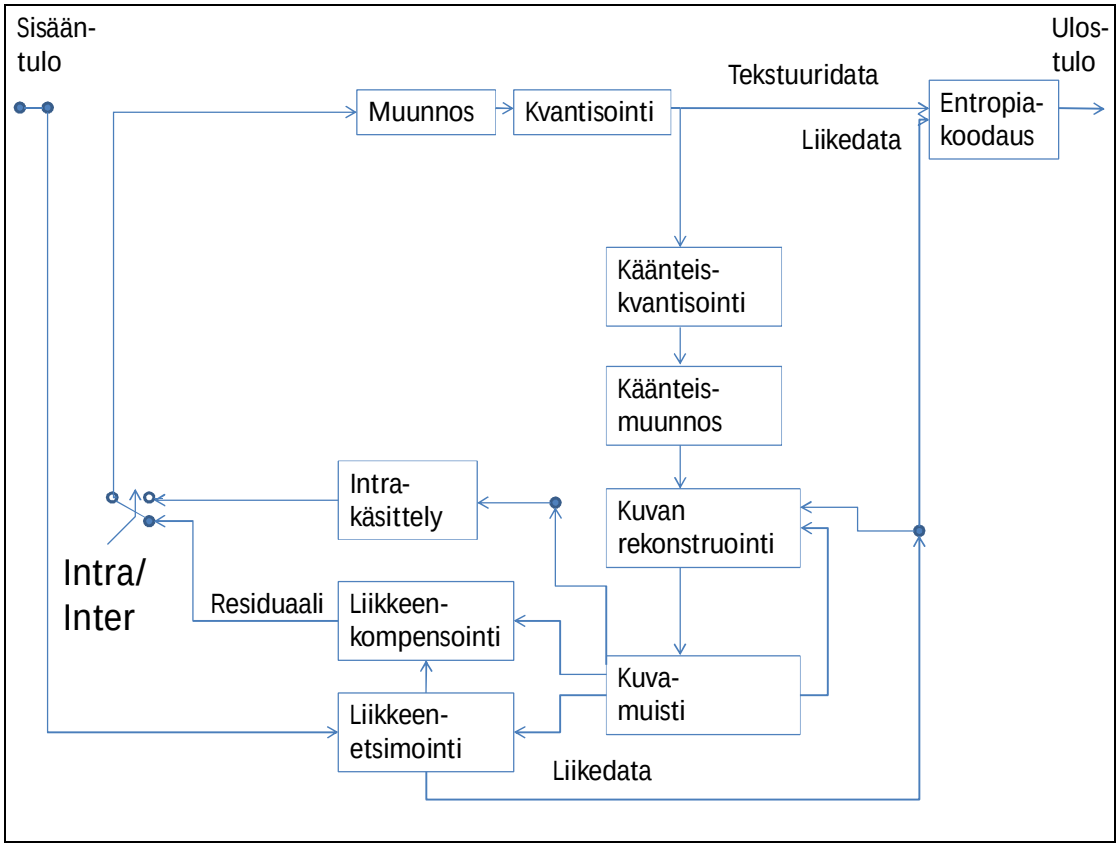
Hyväkään PSNR ei kuitenkaan välttämättä takaa ihmisen katselukokemuksen kannalta hyvää videosignaalia [2]. Tämä johtuu siitä, että ihmissilmä on paljon herkempi paikallisille häiriöille kuin koko kuvaan sekoittuneille häiriölle. Lisäksi tiettyjen kohteiden, kuten ihmishahmojen, häiriöt ovat paljon kriittisempiä subjektiivisen kokemuksen kannalta kuin liikkumattoman taustakuvan häiriöt [2].

Toistaiseksi ei ole kuitenkaan kehitetty muitakaan objektiivisia laadun laskentamenetelmiä, jotka olisivat saavuttaneet yhtä laajaa hyväksyntää kuin PSNR [2].

2.5. Videoenkoodaaja

Jokaisen videokoodekin määritelmään on sisällytettävä määritelmä kahdesta prosessista: enkoodaajasta, joka koodaa raakavideodatan koodekin määrittämään muotoon sekä dekoodaajasta, joka purkaa koodekin määrittämässä muodossa olevan videoinformaation esittämiskelpoiseksi. Seuraavissa kappaleissa tutkitaan näiden prosessien yleisiä ominaisuuksia. Videoenkoodaajan ja videodekoodaajan kaaviokuvat perustuvat kirjallisuuteen [4].

Yleinen videoenkoodaajan perusrakenne on seuraavanlainen:



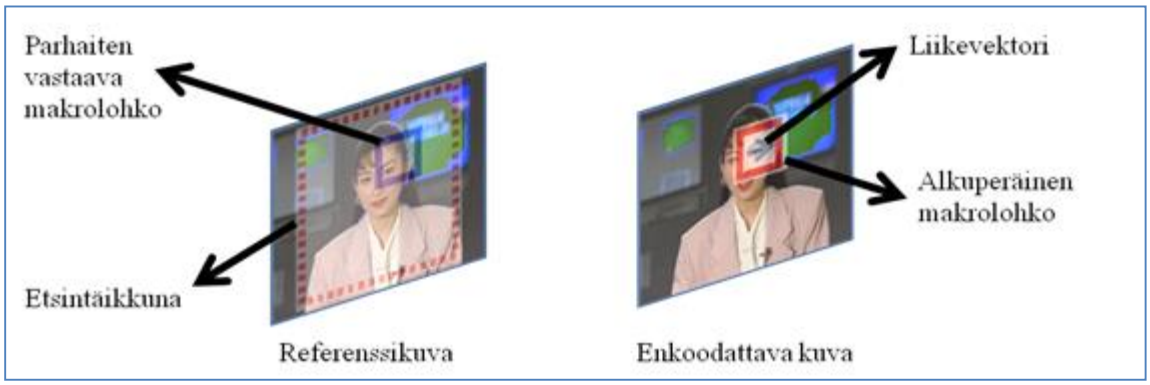
Kuva 2: Yleisen enkoodaajan perusrakenne (laadittu kirjallisuuden [4] perusteella)

Yllä olevassa kuvassa käsitteet intra ja inter viittaavat intra- ja inter-kuviin. Intra-kuviksi (I) kutsutaan niitä, joissa esiintyy vain I-makrolohkoja, eli sellaisia makrolohkoja joiden käsittelyssä ei ole käytetty ko. kuvan ulkopuolisia referenssikuvia. Inter-kuvissa sen sijaan voi olla inter-makrolohkoja, jotka käyttävät referenssikuvia.

2.5.1 Liikkeenestimointi, liikkeenkompensointi ja intra-käsittely

Liikkeenestimoinnissa etsitään koodattavan inter-kuvan kullekin makrolohkolle (yleensä 16x16 pikseliä) sitä mahdollisimman hyvin vastaava makrolohko referenssikuvasta. Etsintää jatketaan kunnes saavutetaan kulloinkin käytössä olevan etsintätavan määrittämät rajat etsinnälle. Tätä rajojen määrittämää aluetta nimitetään etsintäikkunaksi. Referenssikuva puolestaan on aiemmin enkoodattu kuva, joka voi sijoittua lopullisen videon esitysjärjestyksessä joko parhaillaan koodattavaa kuvaa ennen tai sen jälkeen [1].

Haun tuloksena tallennetaan myös ns. liikevektori, joka kertoo eron parhaiten vastaavan makrolohkon ja enkoodattavan makrolohkon suhteellisten sijaintien välillä. Esimerkiksi voidaan havaita, että jokin makrolohko näyttää liikkuneen kolme pikseliä edellisestä kuvasta ylöspäin, jolloin liikevektoriksi tulee kolmea pikseliä vastaava luku. Liikevektoreita käytetään myöhemmin erikseen koodattuna liikedatana. Tätä etsintäprosessia on havainnollistettu alla olevassa kuvassa.



Kuva 3: Liikkeenestimointi

Liikkeenkompensoinnissa vähennetään liikkeenestimoinnissa löydetty parhaiten vastaava makrolohko enkoodattavasta makrolohkosta. Tällöin syntyy ns. residuaalimakrolohko (tekstuuridata).

Intra-kuvissa ei käytetä inter-kuvien tapaan liikkeenkompensointia vaan ns. intra-kuvien käsittelyä. Itse käsittely riippuu paljon itse koodekista, mutta useimmiten etsitään jonkinlaista korrelaatiota kuvansisäisesti ja pyritään vähentämään kuvansisäistä redundanssia vaikkapa liikkeenestimoinnin tapaisilla keinoilla.

2.5.2 Hyvyysluvut

Makrolohkojen välisen vastaavuuden mittaamiseen käytetään hyvyyslukuja. Niitä ovat muun muassa absoluuttisten erojen summa (Sum of absolute differences, SAD), erojen neliöiden summa (Sum of squared differences, SSD) sekä muunnettujen absoluuttisten erojen summa (Sum of absolute transformed diffences, SATD). Määritelmät perustuvat kirjallisuuteen [1].

2.5.2.1 **SAD**

Absoluuttisten erojen summa (SAD) on hyvin yleisesti videokoodekeissa käytetty mittari eri makrolohkojen ja alilohkojen välisen vastaavuuden selvittämiseksi. Se lasketaan yleensä kunkin alilohkon luminanssiarvoista ja tarvittaessa krominanssiarvoille erikseen. Esimerkiksi voidaan laskea NxM pikselin kokoisten alilohkojen luminanssiarvojen välinen SAD:

$$\boldsymbol{SAD = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} \left| Y_{x,y}^{A} - Y_{x,y}^{B} \right|} \quad \textbf{(3)}$$

missä $Y_{x,y}^{A}$ on alilohkon A luminanssin Y arvo rivillä y sarakkeessa x ja $Y_{x,y}^{B}$ alilohkon B arvo omassa vastaavassa kohdassaan.

SAD voidaan toteuttaa hyvin nopeasti rinnakkaiskäskyinä. Esimerkiksi Intelin eräissä prosessoreissa on jopa SSE-käskykannassa oma käskynsä sen laskemiseen. SAD-hyvyysluku ei kuitenkaan vastaa kovin hyvin ihmisen näköjärjestelmän havaintoja eri makrolohkojen vastaavuudesta, joten joskus voidaan käyttää myös ensin karkeaa etsintää SAD:lla ja sen jälkeen tarkempaa etsintää SSD- tai SATD-menetelmällä.

2.5.2.2 **SSD**

Erojen neliöiden summa (Sum of squared differences, SSD) määritellään hyvin samaan tapaan SAD:n kanssa, erotuksena on vain että absoluuttisten erojen sijaan käytetään erojen neliöitä. Esimerkiksi SSD samoille NxM alilohkojen luminanssiarvoille on:

$$\boldsymbol{SSD = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} \left(Y_{x,y}^{A} - Y_{x,y}^{B} \right)^{2}} \quad \textbf{(4)}$$

SAD:hen verrattuna tämä menetelmä on laskennallisesti melko raskas: yksinkertaisen vähennyslaskun lisäksi täytyy lisäksi suorittaa vielä kertolasku. Laskentaan tarvittavien kellojaksojen määrä siis moninkertaistuu.

2.5.2.3 **SATD**

Muunnettujen absoluuttisten erojen summassa (Sum of absolute transformed diffences, SATD) lasketut erot $Y^{A}_{x,y} - Y^{B}_{x,y}$ muunnetaan ensin sopivalla muunnoksella, yleensä Hadamard-muunnoksella, ennen niiden summaamista. Hadamard-muunnosta käytettäessä SATD saadaan seuraavasti NxN pikselin kokoiselle alilohkon luminanssiarvoille:

$$\boldsymbol{SATD = c\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}|H \cdot D \cdot H^{T}|} \qquad \textbf{(5)}$$

missä eromatriisin *D* alkio $D_{x,y} = Y^{A}_{x,y} - Y^{B}_{x,y}$ ja Hadamard -muunnosmatriisin H alkio rivillä x kohdassa y on $H_{x,y} = (-1)^{x\hat{}y}$. Tässä merkinnällä x^y tarkoitetaan bittikohtaista pistetuloa (Esimerkiksi jos $x=2_{10}=10_2$ ja $y=3_{10}=11_2$, saadaan x^y = 1*1 + 0*1=1. Huomaa, että indeksointi tässä yhteydestä lähtee nollasta; matriisin ensimmäinen alkio on $H_{0,0}$). H^T on vastaavasti Hadamard-muunnos – matriisin transpoosi sekä termi c Hadamard-muunnoksen normalisointitermi, joka saadaan seuraavasti: $c = \frac{1}{\sqrt{n}}$, missä n on muunnettavan alilohkon koko (leveys tai korkeus, joiden tulee olla yhtä suuria).

Hadamard-muunnos on määritelty vain neliömatriisina, joten esim. 16x8-alilohkoa ei voi muuntaa kerralla. X.264 – videokoodekissa käytetään yleisemmin muunnosta 4x4 alilohkoissa (N=4) ja yhdistetään tulokset sitten koko alilohkolle. Tällöin 4x4 Hadamard-muunnosmatriisi on:

$$H = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

SATD:n vastaavuus ihmisen näköjärjestelmän kanssa on parempi kuin SAD:n, mutta vastaavasti se on laskennallisesti myös raskaampi. Kirjallisuudessa [5] on vertailtu SAD:n ja SATD:n tuloksia eri testivideoilla. Samassa kirjallisuuslähteessä on myös esitetty parannuksia SATD-algoritmiin laskentatapaan, jotta sen laskenta-aikaa saataisiin parannettua.

Kirjallisuudessa [5] esitettyjen tulosten mukaan kaikki SATD-laskentamenetelmät parantavat SAD-menetelmään verrattuna luminanssin PSNR:ää (katso kappale 2.4.1) maksimissaan vain noin prosentin bittinopeuden pysyessä melkein vakiona. Laskenta-aika perinteisillä menetelmillä jopa lähes kolminkertaistuu ja uudella menetelmälläkin jopa lähes kaksinkertaistuu.

2.5.3 Muunnos

Kuvan muunnoksessa muunnetaan liikkeenkompensoinnissa laskettu residuaalikuva sopivaan muunnostasoon, esimerkiksi DCT-tasoon. Tällä pyritään vähentämään näytteiden keskinäisriippuvuutta ja lisäämään kompaktiutta. Erilaisten muunnosten valintaa rajoittaa se, että muunnokselle pitää löytää käänteisoperaatio. Lisäksi sen pitää olla toteutettavissa reaalimaailman prosessoreilla eli muunnosten pitää olla toteuttavissa rajallisen tarkkuuden luvuilla ja suhteellisen vähillä resursseilla. Tyypillisiä muunnoksia ovat esimerkiksi DCT-muunnos ja sen approksimaatiot. Voidaan myös käyttää erilaisia Wavelet-muunnoksia [1]. Käänteismuunnos tehdään ko. operaatioiden etukäteen määrittämällä tavalla, esim. DCT-käänteismuunnoksena.

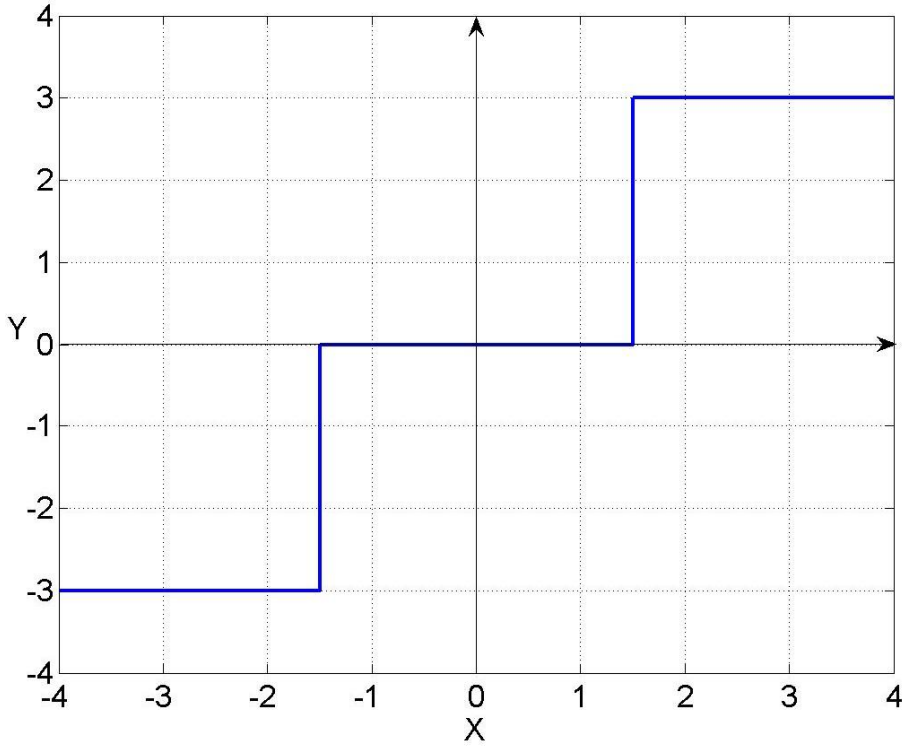
2.5.4 Kvantisointi, kuvan rekonstruointi ja kuvamuisti

Kvantisoinnissa muunnetaan signaali, jolla on tietty määrittelyväli X sellaiseksi signaaliksi, jolla on pienempi määrittelyväli Y. Kvantisoitu signaali voidaan siten esittää pienemmällä bittimäärällä kuin alkuperäinen signaali. Tyypillisesti kvantisointi suoritetaan seuraavaan tapaan:

$$\boldsymbol{Y = QP * round\left(\frac{X}{QP}\right)} \qquad \textbf{(6)}$$

missä *QP* on kvantisointiparametri, *X* alkuperäinen signaali ja *Y* kvantisoitu signaali. *Round-funktiolla* viitataan tässä funktioon, joka pyöristää luvun lähimpään kokonaislukuun.

Esimerkiksi arvolla QP=3 X:n määrittelyvälillä $X \in \{-4, -3, \ldots, 3, 4\}$ saadaan seuraavan näköinen kvantisointi:



Kuva 4: Peruskvantisointiesimerkki

Käänteiskvantisointi suoritetaan luonnollisesti käänteisenä prosessina, eli saatu kvantisoitu signaali puretaan sovituiksi signaaliarvoiksi.

Käänteiskvantisoinnin ja käänteismuunnoksen jälkeen tulokseksi jää kuvan residuaali, joka on jonkin verran vääristynyt alkuperäisestä residuaalista. Käyttämällä tätä residuaalia, tietoa sen liikevektoreista ja referenssikuvasta voidaan rekonstruoida uusi referenssikuva. Uusi referenssikuva tallennetaan sitten kuvamuistiin, joka palvelee puolestaan liikkeenestimointia ja liikkeenkompensointia.

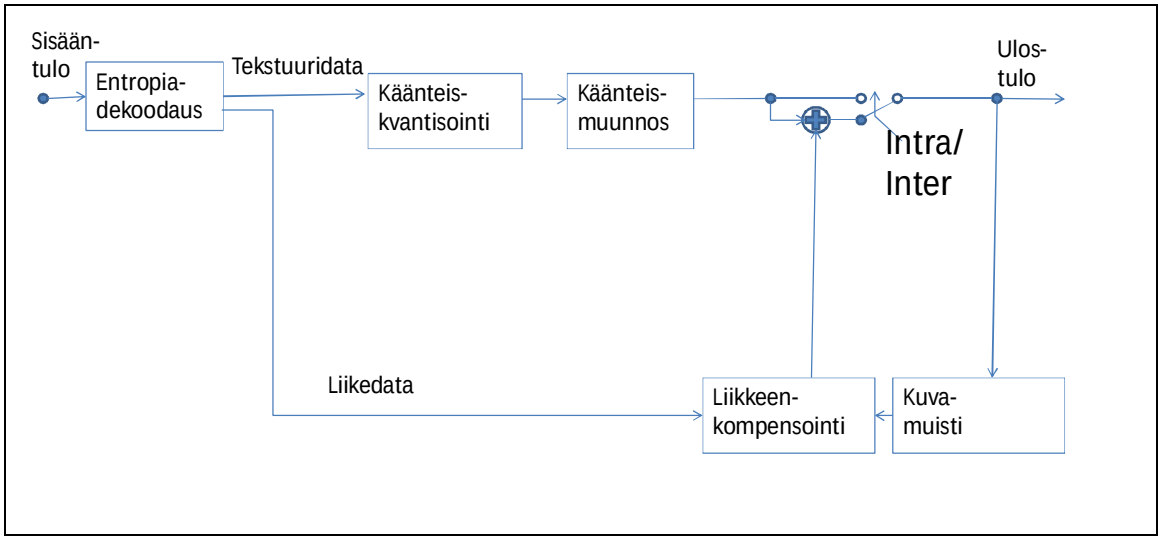
2.5.5 Entropiakoodaus

Entropiakoodauksessa koodattu data muunnetaan sopivaksi koodisarjaksi videon tallennusta tai lähetystä varten. Koodattu data voi sisältää tietoa residuaaleista (tekstuuridataa), liikevektoreita (liikedataa), kuvien otsikkotietoja tai muuta kontrollidataa. Entropiakoodauksessa käytetään yleensä hyväksi vaihtuvamittaista koodausta ja aritmeettista koodausta [1].

Vaihtuvamittaisessa koodauksessa muunnetaan datassa esiintyvät arvot niiden esiintymistiheyden mukaan sopivanpituisiksi koodisanoiksi: usein esiintyvälle arvolle lyhyt koodisana ja harvoin esiintyvälle arvolle pidempi. Esimerkiksi data {7,7,7,3,3,3,5,5,16} voitaisiin koodata binäärikoodein seuraavasti {7=>0, 3=>01,5=>001, 16=>0001}. Näin säästetään tilassa, kun data-arvoja ei tarvitse kokonaisuudessaan lähettää vaan ainoastaan lyhyemmät koodisanat. Nämä koodisanat lähetetään eteenpäin ja vastaavasti vastaanottava dekoodaaja purkaa ne takaisin data-arvoiksi.

2.6. Videodekoodaaja

Yleinen videodekoodaajan perusrakenne on esitetty alla olevassa kirjallisuudesta [4] peräisin olevassa kuvassa. Videodekoodaajan tehtävänä on siis purkaa videoenkoodaajasta saatu data sellaiseen muotoon, että sen voisi esittää kohdeyleisölle esim. tietokoneen monitorilla.



Kuva 5: Yleisen videodekoodaajan perusrakenne (perustuu kirjallisuuteen [4])

Videodekoodaaja käyttää tähän edellisessä kappaleessa esitettyjen prosessien käänteisoperaatiota: ensiksi purkaa entropiakoodauksen, jolloin saadaan selville kunkin kuvan tekstuuridata (residuaalit) ja liikevektorit (liikedata). Residuaalidata tosin pitää ensin käänteiskvantisoida ja käänteismuuntaa.

Liikevektoreita puolestaan hyödynnetään liikkeenkompensoinnissa, joka niiden avulla laskee referenssikuvasta liikevektoreiden avulla alustavan kuvan. Kun alustavaan kuvaan lisätään vielä käänteismuunnettu residuaali, saadaan valmis kuva.

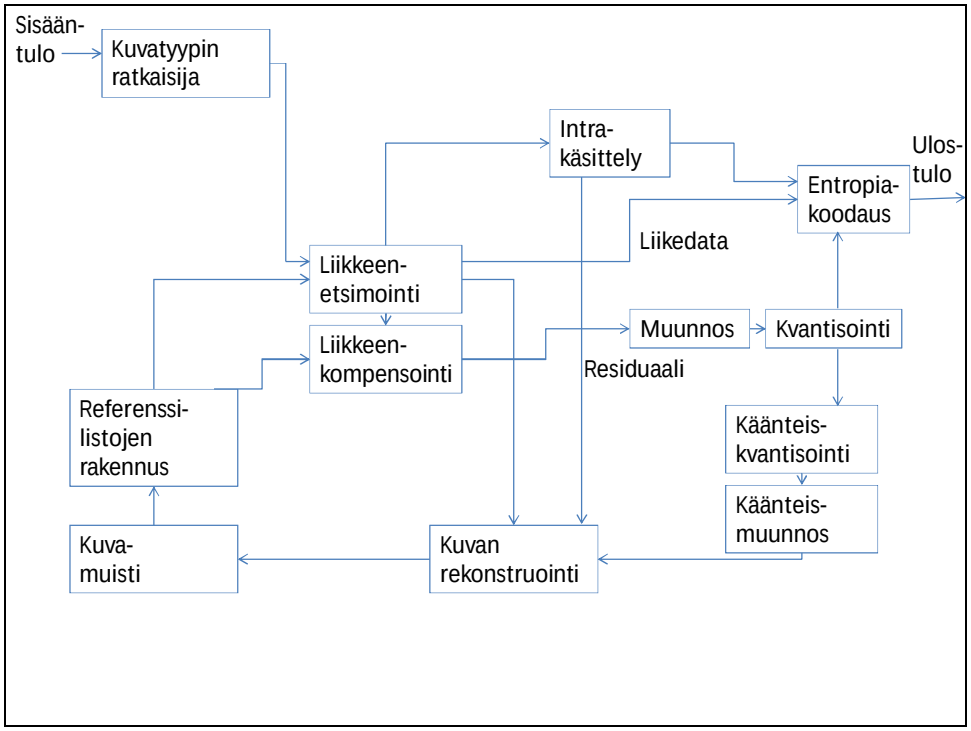
3. X.264 Videokoodekin toiminta

X.264 videokoodekin on tarkoitus toteuttaa ITU-T:n standardissa H.264 [6] määritetty videokoodaus. H.264 –standardissa ei kuitenkaan ole tiukasti määritelty miten videokoodaus toteutetaan, vaan toteutukselle annetaan laajat vapaudet käytettävissä olevien resurssirajoitusten mukaisesti.

H.264-standardille on olemassa myös suoraan standardiin perustuva mallitoteutus [7], mutta ko. toteutus on hyvin vaikeaselkoinen ja siitä on hankala erottaa yksittäisiä toimintaosioita. Siksi tässä diplomityössä on päädytty käyttämään referenssitoteutuksena H.264-standardiin perustuvaa X.264-videokoodekkia [8], jonka koodi on huomattavasti H.264-mallitoteutusta lukijaystävällisempää.

X.264-videokoodekinkaan valinta ei ollut täysin ongelmatonta, sillä siinä lähtökohtana on ollut luoda videoenkoodaaja moderneille PC-koneille ohjelmistototeutuksena. Lähtökohtaisesti siis oletetaan, että käytettävissä on melko tehokas PC-kone, jolla on resursseja ja runsaasti aikaa laskeskella monimutkaisia algoritmeja. Tässä diplomityössä taas päämääränä oli luoda resursseiltaan kännykkää vastaavalle laitteelle laitteistopohjainen videokoodekin toteutus. Siksi tässä diplomityössä on jouduttu karsimaan runsaasti myös X.264-videokoodekin ominaisuuksia eikä niitä kaikkia siis esitellä tässä kappaleessa. Videokoodekin toiminnan kuvaus perustuu sen julkisesti saatavilla olevaan lähdekoodiin [8].

X.264 – videokoodekin mukaisen enkoodaajan toimintaa voidaan kuvata alla olevan kuvan avulla:



Kuva 6: X.264 video-enkoodajan rakenne

Ensisilmäyksellä se saattaa vaikuttaa hyvin erilaiselta yleisen enkoodaajan perusrakenteeseen (Kuva 2) verrattuna, mutta ero kuvien välillä johtuu vain X.264 – videokoodekin pienten yksilöllisten ominaisuuksien korostamisesta. Seuraavissa kappaleissa kuvataan näitä pieniä eroja.

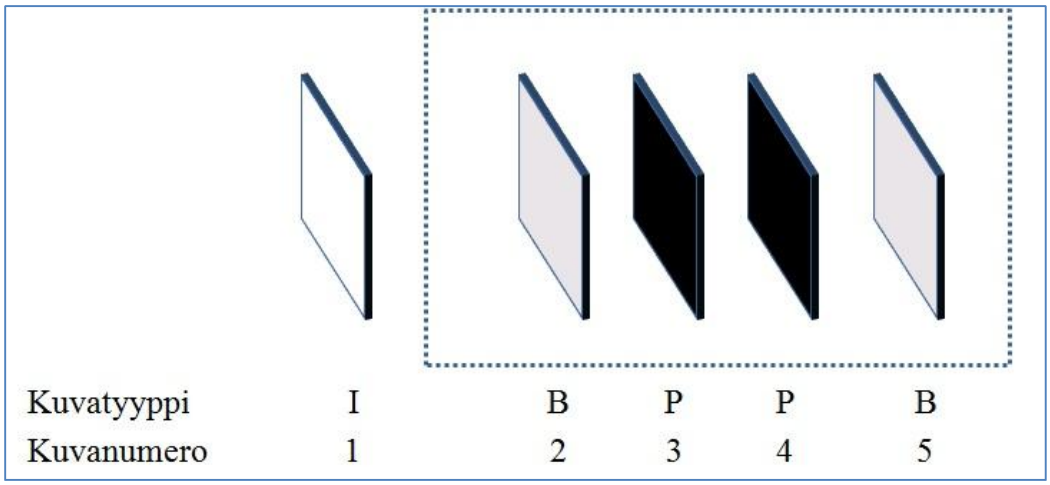
3.1. Kuvatyypin valinta

X.264 – videokoodekissa päätetään aluksi minkä tyyppiseksi kuva pitäisi asettaa: Intra-kuvaksi (I) vai inter-kuvaksi (P tai B). Intra-kuvassa on vain intra (I)-makrolohkoja, joiden koodaamisessa ei käytetä referenssinä muita kuvia. Inter-kuvissa puolestaan on myös P- ja B-makrolohkoja sekä lisäksi niissä voi olla myös mukana I-makrolohkoja P- ja B-makrolohkojen seassa.

Ero P-makrolohkojen ja B-makrolohkojen välillä on siinä, että P-makrolohkoissa käytetään yhdessä makrolohkossa vain yhtä referenssikuvaa kerrallaan laskettaessa liikevektoreita, kun taas B-makrolohkossa referenssikuvia on kaksi.

P-kuva koostuu I-makrolohkoista ja P-makrolohkoista. Huomaa, että eri P-makrolohkot voivat käyttää eri referenssikuvia, jolloin puhutaan referenssilistasta 0. B-kuvat puolestaan voivat sisältää I-, P- ja B-makrolohkoja. B-makrolohkoissa voi olla samaan aikaan referenssejä kahteen eri kuvaan. B-kuvissa näitä kahden tyyppisiä referenssikuvia voi olla useita, jolloin puhutaan referenssilistoista 0 ja 1.

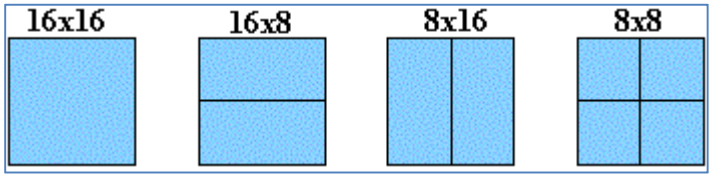
Alla olevassa kuvassa on esitetty tyypillinen videosekvenssi, jossa on I-, P- ja B-kuvia. I-kuva toimii yhteisenä referenssikuvana myöhemmille videokuville ja B-kuvat käyttävät myös P-kuvia referenssikuvina.



Kuva 7: Tyypillinen videosekvenssi

3.2. Liikkeenestimointi

X.264 –videokoodekissa voidaan isompi 16x16 makrolohko jakaa tarvittaessa osiin ja laskea jokaiselle osalle oma liikevektorinsa. Yleisimmät makrolohkojaot on kuvattu alla olevassa kuvassa.



Kuva 8: Yleisimpiä makrolohkojakoja

Pienempiäkin makrolohkoja voi käyttää, mutta X.264:n kehittäjien testien mukaan niiden tuottama lievä laadun paraneminen ei vastaa niiden vaatimaa ylimääräistä laskentatehoa [8]. Makrolohkojaossa jakoa pienempiin makrolohkoihin harkitaan hyvin tarkkaan: yhdellä liikevektorilla tallennettavaa on vähemmän, mutta toisaalta silloin koodattava kuva ja referenssikuva eivät vastaa ehkä kovin hyvin toisiaan ja tallennettava jäännös (residuaali) on isompi. Etsintä aloitetaan 16x16 – makrolohkoista.

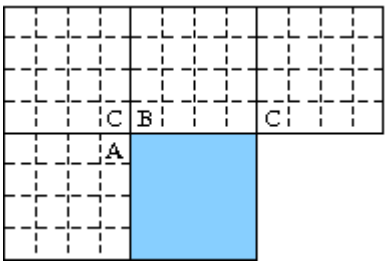
16x16 inter-makrolohkojen tutkinta

Analysoinnissa tutkitaan ensin löytyisikö hyvä vastaavuus nyt koodattavan kuvan 16x16 makrolohkon ja toisen, jo aiemmin koodatun referenssikuvan makrolohkojen välillä. Tätä nimitetään jatkossa 16x16-etsinnäksi inter-makrolohkoista (vastaavasti voidaan tehdä myös haku intra-makrolohkoista, eli samassa kuvassa jo aiemmin koodatuista makrolohkoista).

Etsintä aloitetaan tutkimalla ennustajaliikevektoreita eli aiemmista etsintöjen tuloksista saatuja liikevektoreita, jotka voivat ohjata etsintää nopeammin oikeille jäljille.

Ennustajaliikevektori mvp

16x16 – makrolohkoilla ennustajaliikevektori mvp:n lataamista on havainnollistettu alla olevassa kuvassa. Siinä on esitetty parhaillaan käsittelyssä oleva 16x16 makrolohko ja neljä viereistä 16x16 makrolohkoa. Viereiset 16x16 makrolohkot on myös pilkottu kuutentoista pienempään 4x4 pikselin alilohkoon, joista kullakin on oma liikevektorinsa.



Kuva 9: 16x16-etsinnän mvp

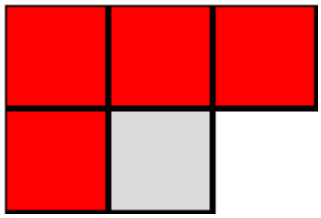
Ennustajaliikevektorin kandidaatit haetaan yllä olevassa kuvassa seuraavasti:

1. kandidaatti **A** on vasemmanpuoleisen 16x16 makrolohkon yläoikealla olevan **4x4** lohkon liikevektori
2. kandidaatti **B** on yläpuolisen 16x16 makrolohkon vasemmalla alhaalla olevan **4x4** lohkon liikevektori
3. kandidaatti **C** on vasemmalla ylhäällä olevan 16x16 makrolohkon oikealla alhaalla olevan **4x4** lohkon liikevektori. Jos vasemmalla ylhäällä ei ole 16x16 makrolohkoa, käytetään oikealla ylhäällä olevaa makrolohkoa (katso kuva)

Jos vertailtavaa makrolohkoa ei ole, nollataan kyseinen kandidaatti. Ennustajaliikevektori mvp lasketaan sitten mediaanina kandidaateista A, B ja C.

Ennustajaliikevektorit mvc

Ennustajaliikevektoreiksi **mvc** tallennetaan osa viereisten 16x16 -makrolohkojen **16x16-etsinnässä saaduista** liikevektoreista (katso Kuva 10: 16x16-etsinnän mvc). Käytettävät makrolohkot ovat suoraan vasemmalla, ylhäällä, ylävasemmalla ja yläoikealla sijaitsevat 16x16- makrolohkot. Jos ko. asemissa ei ole makrolohkoja, jätetään se asema yksinkertaisesti väliin.



Kuva 10: 16x16-etsinnän mvc

Aloitusarvot ennustajaliikevektoreista mvp ja mvc

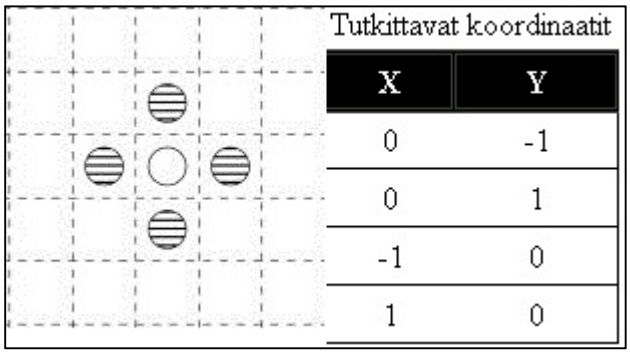
Aluksi verrataan nykyistä laskettavaa 16x16 -makrolohkoa ennustajaliikevektorien mvp ja mvc osoittamiin kohtaan. Laskenta suoritetaan laskemalla SAD (katso kappale 2.5.2.1) pikseli kerrallaan. Aloituskohtaa korjataan sitten sen ennustajaliikevektorin mukaisesti, joka antaa pienimmän SAD-luvun.

3.2.1 Liikevektorin etsintätilat

Aloitusarvon asettamisen jälkeen voidaan etsiä sopivaa liikevektoria usealla eri metodilla, joista ensimmäinen on timanttietsintä.

Timanttietsintä (DIA)

Timanttietsinnässä lasketaan vertailukuvan vertailukohtien (Kuva 11: Timanttietsintä, merkitty :lla) ja nykyisen 16x16 makrolohkon välinen SAD pikseli kerrallaan. Uudeksi aloituskohdaksi asetetaan se vertailukohta, jolla on pienin SAD. Etsintää jatketaan kunnes saavutetaan etsintäikkunan rajat tai huomataan, että origo antaa paremman SAD-luvun kuin mikään vertailukohdista.

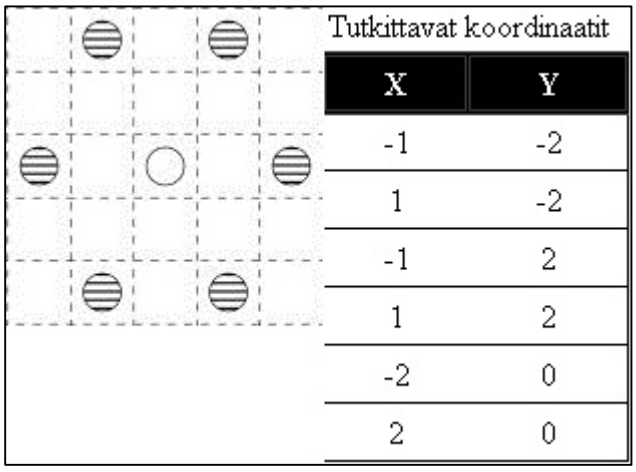


Tutkittavat koordinaatit	
X	Y
0	-1
0	1
-1	0
1	0

Kuva 11: Timanttietsintä

Heksadiagonaalietsintä (HEX)

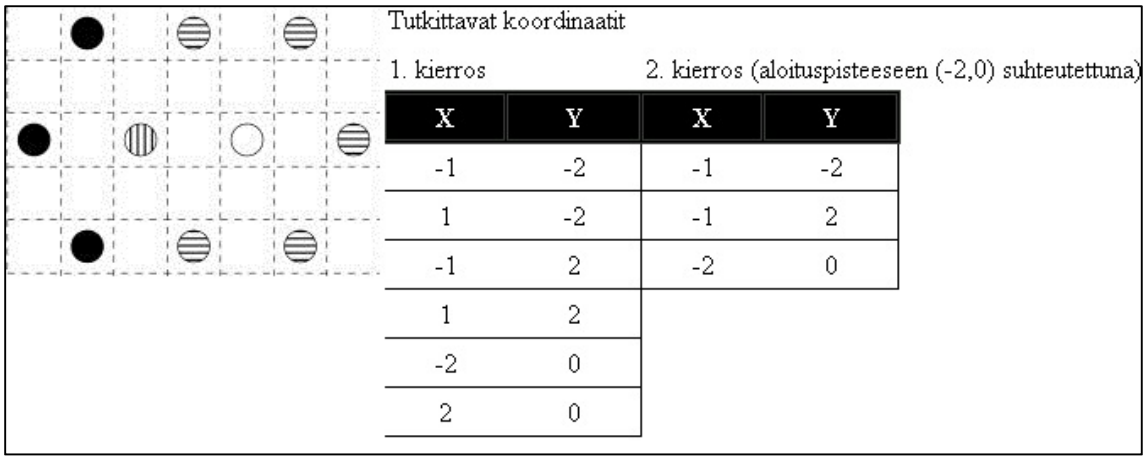
Heksadiagonaalietsinnässä aloitetaan vertailemalla aloituskohtaa kuvan osoittamiin vertailukohtiin. Sen jälkeen valitaan parhaan SAD-luvun kohta uudeksi aloituskohdaksi ja suoritetaan puolikasheksagonaalietsintä (koska osa vertailukohdista oli jo laskettu aloituskierroksella).



Tutkittavat koordinaatit	
X	Y
-1	-2
1	-2
-1	2
1	2
-2	0
2	0

Kuva 12: Heksadiagonaalietsintä, vaihe 1

Esimerkiksi (merkitään aloituskohtaa ○:lla, parhaimman SAD-luvun kohtaa ◍:lla, ensimmäisen kierroksen vertailukohtaa ≡:lla, toisen kierroksella laskettavia kohtia ●:lla) jos ensimmäisellä kierroksella paras vertailukohta on koordinaatti (-2,0), laskenta etenee seuraavasti:

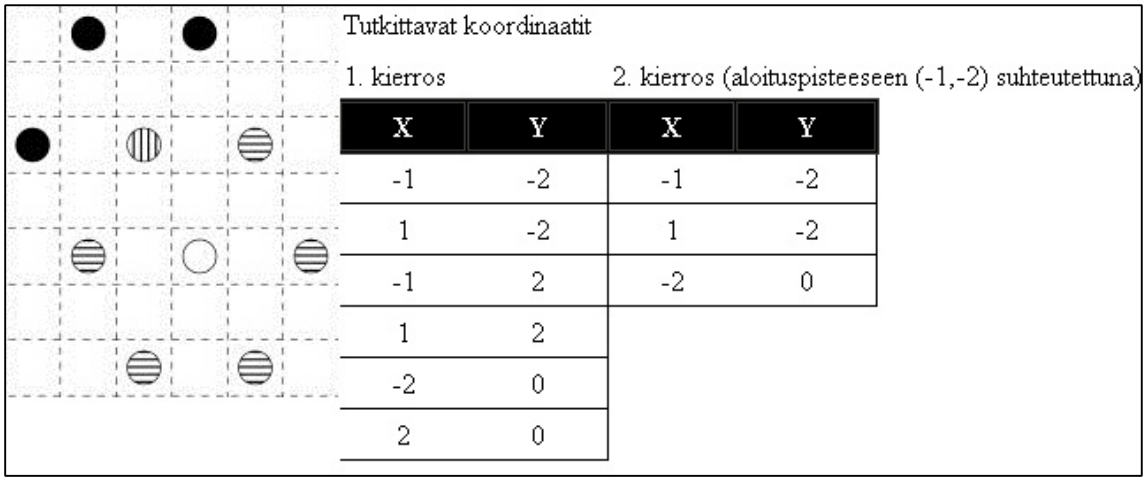


Tutkittavat koordinaatit

1. kierros		2. kierros (aloituspisteeseen (-2,0) suhteutettuna)	
X	Y	X	Y
-1	-2	-1	-2
1	-2	-1	2
-1	2	-2	0
1	2		
-2	0		
2	0		

Kuva 13: Heksadiagonaalietsintä, vaihe 2 (esimerkki 1)

Tai jos ensimmäisellä kierroksella paras vertailukohta on koordinaatti (-1,-2) laskenta etenee seuraavasti:

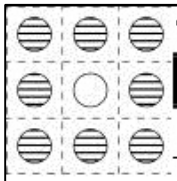


Tutkittavat koordinaatit

1. kierros		2. kierros (aloituspisteeseen (-1,-2) suhteutettuna)	
X	Y	X	Y
-1	-2	-1	-2
1	-2	1	-2
-1	2	-2	0
1	2		
-2	0		
2	0		

Kuva 14: Heksadiagonaalietsintä, vaihe 2 (esimerkki 2)

Puoliheksagonaalietsintää jatketaan kunnes saavutetaan etsintäikkunan rajat tai huomataan, että origo antaa paremman SAD-luvun kuin mikään vertailukohdista. Tämän jälkeen suoritetaan täydentävä neliöhaku:



Tutkittavat koordinaatit	
X	Y
0	-1
0	1
-1	0
1	0
-1	-1
-1	1
1	-1

Kuva 15: Heksadiagonaalietsinnän vaihe 3

Täydentävää neliöhakua ei suoriteta rekursiivisesti, vaan parhaimman SAD-luvun valittu piste valitaan suoraan.

Muut täyspikselietsintätilat

X.264 -koodekissa on toteutettu myös kaksi muuta täyspikselietsintätilaa: täyshaku (Exhaustive search, ESA) ja muunnettu täyshaku (Transformed Exhaustive Search, TESA). Ne ovat kuitenkin laskennallisesti hyvin raskaita, eivätkä testivideoilla tehtyjen simulointien mukaan (katso kappale 4: X.264-videokoodekin tulosmittauksia) paranna laatu/bittinopeus –suhdetta.

Vektorin tallennuskustannus

Etsintätilojen (DIA, HEX jne.) suorittamisen jälkeen SAD-lukuun lisätään vektorin tallentamisen kustannus. Näin voidaan arvioida onko järkevää tallentaa liikevektori vai pitäisikö se vain jättää nollaksi. Vektorin tallennuskustannus lasketaan seuraavasti:

$$\boldsymbol{\lambda = 2^{\frac{qp}{6}-2}} \quad \textbf{(7)}$$

$$\boldsymbol{cost(mv) = \lambda(log_2(mv + 1) * 2 + 0.718 + f(mv)) + 0.5} \quad \textbf{(8)}$$

missä parametri *qp* on kvantisointiparametri, *mv* on tallennettava liikevektori (neljännespikselin tarkkuudella), *cost(mv)* vektorintallennuskustannus ja *f(mv)* on seuraavasti määritelty funktio: f(0)=0, muulloin f(mv)=1. Nämä kaavat perustuvat X.264 –videokoodekin lähdekoodiin [8].

Alilohkoetsintätilat

Edellä kuvatun 16x16-etsinnän jälkeen seuraavaksi on vuorossa 8x8-etsintä. Siinä ennustajaliikevektoriksi mvc asetetaan 16x16 -etsinnässä tulokseksi saatu liikevektori. Ennustajaliikevektori mvp alustetaan kuten 16x16 –etsinnässä (Kuva 9), paitsi että tarkastellaan kulloisenkin 8x8-makrolohkon läheisten 4x4 –alilohkojen liikevektoreita.

Etsintä jatkuu sitten kuten 16x16 -etsinnässä, eli lasketaan ensin kullekin 8x8-alilohkolle ennustajaliikevektorien osoittama aloituskohta. Sitten jatketaan säädetyllä täyspikselietsinnällä (DIA, HEX jne.), kunnes saadaan kullekin 8x8-alilohkolle oma liikevektori. SAD-luvut lasketaan siis luonnollisesti vertailemalla kahden 8x8-lohkon arvoja eikä 16x16-lohkojen arvoja. Lopuksi lasketaan SAD-lukujen arvot ja liikevektorien kustannukset yhteen.

16x8 ja 8x16- etsintää ei suoriteta automaattisesti, vaan ainoastaan, jos 8x8 antaa huomattavasti parempia tuloksia kuin 16x16 -etsintä. Vertailu suoritetaan vertaamalla koko 8x8-etsinnän yhteenlaskettujen kustannuksia 16x16-etsintään, johon on lisätty niiden kahden 8x8-alilohkon vektorien tallennuskustannukset, jotka kuuluvat 16x8 -alilohkoon tai 8x16-alilohkoon.

Jos kynnysarvot alittuvat, suoritetaan ensin 16x8 -etsinnät. Ennustajaliikevektori mvp alustetaan kuten 16x16 –etsinnässä (Kuva 9), paitsi että tarkastellaan kulloisenkin 16x8-makrolohkon läheisten 4x4 –alilohkojen liikevektoreita.

Ennustajaliikevektoreita mvc on kolme. Ensimmäinen niistä on 16x16 -etsinnästä saatu tulos ja kaksi muuta ovat kulloisenkin etsinnän alueella olevien 8x8-alilohkojen etsintöjen tulokset.

Etsintä jatkuu sitten kuten 16x16 -etsinnässä, eli lasketaan ensin kullekin 16x8-alilohkolle ennustajaliikevektorien osoittama aloituskohta. Sitten jatketaan säädetyllä täyspikselietsinnällä (DIA, HEX jne.), kunnes saadaan kullekin 16x8-alilohkolle oma liikevektori. 8x16-etsintä suoritetaan vastaavasti, paitsi ennustajaliikevektori mvp alustetaan kulloisenkin 8x16-makrolohkojen läheisten 4x4-alilohkojen liikevektoreista.

Oikeanlainen makrolohkojako päätetään kaiken kaikkiaan siis seuraavasti:
Oletusarvona makrolohkojaolle on 16x16-lohkojako. Ensin tehdään 16x16-etsintä, sitten 8x8-etsinnät. Jos 8x8-etsinnät tuottavat paremman tuloksen kuin 16x16-etsintä, niin asetetaan makrolohkojaoksi 8x8. Jos kynnysarvot 16x8 -etsinnälle ja 8x16 -etsinnälle alittuvat, niin tehdään myös ne. Jos ne tuottavat paremman vertailuarvon, asetetaan makrolohkojaoksi 16x8 tai 8x16.

I- ja B-kuvien käsittely

Edellä oleva teksti on koskenut lähinnä yksinkertaista tilannetta, jossa on käytettävissä vain yksi referenssikuva koodattavan kuvan lisäksi. Sekin muodostaa laskennallisesti valtavan haasteen kannettaville laitteille, joten intra (I)- ja B-kuvien käsittely on tarkoituksella jätetty käsittelemättä tässä diplomityössä.

3.2.2 Alipikseliliikkeenarviointi

Useinkaan videossa esiintyvät liikkuvat kohteet eivät liiku täsmälleen pikselirajojen mukaisesti. Siten laskemalla myös alipikselimuutoksia voidaan parantaa huomattavasti koodattavan videon laatu/bittinopeus-suhdetta (katso tarkemmin kappale 4).

Alipikselien laskenta on esitetty kirjallisuudessa [1]. Yksinkertaistettuna puolipikselit saadaan painotettuna keskiarvona viereisistä täyspikseliarvoista ja neljännespikselit kahdesta viereisestä arvosta (puolipikseleistä tai täyspikseleistä).

Alipikselietsintätilat

Tärkeimmät alipikselietsintätilat ovat:

0 Tutkitaan vain täyspikseleitä

1 Tutkitaan yksi kierros puolipikseleitä ja yksi kierros neljännespikseleitä kokopikselietsinnän päätteeksi. Hyvyyslukuna käytetään SAD:tä sekä puolipikselien että neljännespikselien tutkinnassa.

2 Alipikseleitä tutkitaan jokaisen lohkon (16x16, 8x8 jne.) tutkimisen päätteeksi erikseen ja lisäksi vielä kokopikselietsinnän päätteeksi. Kuitenkin kerralla tutkitaan vain yksi neljännespikselikierros. Hyvyyslukuna käytetään SATD:tä.

Simuloitujen tulosten mukaan muilla etsintätiloilla ei juuri saavuteta parannusta laatu/bittinopeus-suhteeseen (katso tarkemmin kappale 4). Suurin osa parannuksesta saavutetaan itse asiassa jo siirryttäessä etsintätilasta 0 etsintätilaan 1.

Alipikselietsintä

Puolipikselietsinnän aluksi käytetään samaa täyspikselietsinnässä käytettyä ennustajaliikevektoria mvp ja päivitetään tarpeen vaatiessa aloituskohtaa. Toisin kuin täyspikselietsinnässä, tällä kertaa otetaan huomioon myös mvp:n alipikselikomponentti. Puolipikselietsintä suoritetaan vertaamalla kulloistakin makrolohkoa puolipikselilaskennassa saatuun vertailulohkoon käyttäen timanttietsintää. Etsintää jatketaan niin monta kierrosta kuin kussakin alietsintätilassa on säädetty. Neljännespikselietsintä tapahtuu vastaavasti timanttietsinnällä jatkaen niin monta kierrosta kuin kussakin alietsintätilassa on säädetty.

3.3. Residuaalin laskenta, muunnos ja kvantisointi

Oikean makrolohkojaon ja liikevektorin laskemisen jälkeen on laskettava jäännöskuva eli residuaali. Se lasketaan vähentämällä ensin pikseli kerrallaan referenssimakrolohkosta lasketun liikevektorin osoittama makrolohko. Residuaaliin sovelletaan sen jälkeen DCT-muunnoksen approksimaatiota (kirjallisuudessa [1], sivu 190 tai kaava (9)) kuhunkin 4x4-alilohkoon kerrallaan. Näin voidaan lopullisen datan tallennuksessa käyttää hyväksi pikselien arvojen välistä redundanssia.

$$\boldsymbol{Y} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{2} & \mathbf{1} & \mathbf{-1} & \mathbf{-2} \\ \mathbf{1} & \mathbf{-1} & \mathbf{-1} & \mathbf{1} \\ \mathbf{1} & \mathbf{-2} & \mathbf{2} & \mathbf{-1} \end{bmatrix} \boldsymbol{X} \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{-1} & \mathbf{-2} \\ \mathbf{1} & \mathbf{-1} & \mathbf{-1} & \mathbf{2} \\ \mathbf{1} & \mathbf{-2} & \mathbf{1} & \mathbf{-1} \end{bmatrix} \qquad \textbf{(9)}$$

missä *X* on muunnettava 4x4 näytteen matriisi ja *Y* muunnettu matriisi.

DCT-muunnoksen approksimaation jälkeen tehdään tulosten kvantisointi, jolla pyritään edelleen tehostamaan laatu/bittinopeuden suhdetta. Kvantisointi on esitelty tarkemmin kirjallisuudessa [1].

4. X.264-videokoodekin tulosmittauksia

X.264 – videokoodekki on rakennettu hyvin joustavaksi. Sen **tietokoneella käännetylle ohjelmistototeutukselle** voi antaa komentorivillä monenlaisia parametrejä, joiden aiheuttamaa muutosta kuvan laadussa (PSNR) ja bittinopeudessa voidaan tarkkailla. Esimerkiksi voidaan säätää käytetäänkö laisinkaan B-kuvia, montaako referenssikuvaa käytetään ja käytetäänkö krominanssiarvoja laisinkaan vertailulaskennassa.

Tulosmittausten tarkoituksena oli kaiken kaikkiaan selvittää miten päästäisiin kannettavalle laitteelle siedettävillä laskentavaatimuksilla kohtuulliseen laatuun ja bittinopeuteen. Tutkittavia parametrejä kertyi siksi todella monta, joten oli tarpeen rajoittaa jo etukäteen niiden määrää. Jo etukäteen voitiin olettaa, että B-kuvien laskenta voisi olla kannettavalle laitteelle liian raskasta. Samoin perustein referenssikuvien määrää rajoitettiin vain yhteen. Entropiakoodauksen **CABAC-optio** (Context-based Adaptive Binary Arithmetic Coding), **painotettu ennustaminen P-kuvissa** sekä **Trellis-kvantisointi** kytkettiin pois, koska ne olisivat myös olleet liian raskaita toteutettavaksi.

X.264-videokoodekissa oli myös yksi kriittinen ominaisuus, jolle ei löytynyt valmista komentorivin parametria, nimittäin I-makrolohkojen käytön esto P-kuvissa. Tutkimalla koodia voitiin havaita, että tämä lisäsi todella paljon laskentaa, joten oli tarpeen muokata alkuperäistä X.264-videokoodekkia siten että I-makrolohkojen käyttö saataisiin tarvittaessa estettyä.

Vertailtavien parametrien määrän vuoksi oli tarpeen myös jakaa vertailuparametrit useampaan eri kategoriaan. Ensimmäiseksi oli tarkoitus tutkia oliko eri kokopikselietsintätiloilla (timanttietsintä, heksagonaalietsintä jne.) vaikutusta laatuun ja bittinopeuteen. Toisessa kategoriassa tarkastellaan millaista vaikutusta alipikselietsintätiloilla on samoihin ominaisuuksiin. Lisäksi testattiin muutama vähämerkityksisempää parametria, kuten vaihtelua 8x8 DCT approksimaation ja 4x4 DCT approksimaation välillä sekä krominanssiarvojen käyttöä liikevektorietsinnässä.

Testivideoiksi oli valittu kahdeksan Xiph.org:n [9] vapaaseen julkiseen käyttöön tarjoamaan YUV-koodattua QCIF-kokoista (176x144 pikseliä) testivideota: akiyo_qcif.y4m, bowing_qcif.y4m, carphone_qcif.y4m, crew_qcif_15fps.y4m, foreman_qcif.y4m, hall_monitor_qcif.y4m, harbour_qcif_15fps.y4m ja soccer_qcif_15fps.y4m. Nämä testivideot oli valittu videossa esiintyvän liikkeen määrän mukaan edustamaan mahdollisimman erilaisia testivideoita. Esimerkiksi testivideossa akiyo_qcif.y4m on mahdollisimman vähän liikettä: vain videossa esiintyvän naisen pää liikkuu ja koko muu osa videokuvasta ei muutu kuvasta toiseen. Testivideossa bowing_qcif.y4m puolestaan videossa esiintyvä mieshenkilön koko vartalo liikkuu, mutta taustakuva pysyy vakiona. Toista ääripäätä edustaa testivideo soccer_qcif_15fps.y4m, missä sekä tausta että siinä esiintyvät henkilöt liikkuvat.

4.1. Kokopikselietsinätilat

Tässä kokeessa käytettiin seuraavia X.264-koodekin asetuksia:

Parametri	Komentorivi	Käytetyt arvot	Lisätietoa parametristä
Kokopikselietsintätila	`--me`	Dia,hex,umh,esa,tesa	
Alipikselietsinätilat	`--subme`	0-7	
Käytettävien referenssikuvien määrä	`--ref`	1	Muut vaihtoehdot laskennallisesti liian raskaita
Käytettyjen B-kuvien määrä	`--bframes`	0	Muut vaihtoehdot laskennallisesti liian raskaita
Painotettu ennustaminen P-kuvissa	`--weightp`	0	Muut vaihtoehdot laskennallisesti liian raskaita
8x8 DCT-tilan käyttämättömyys	`--no-8x8dct`	8x8 DCT käytössä	Jos 8x8 näytteen DCT-muunnosta ei käytetä, käytössä on koko ajan 4x4 DCT

Krominanssiarvojen käyttämättömyys liikevektorietsinnässä	`--no-chroma-me`	Krominanssiarvoja ei käytetä	
Viipaleiden määrä kuvassa	`--slices`	1	Muut vaihtoehdot laskennallisesti liian raskaita
Kvantisointiparametri	`--qp`	24,40	
CABAC:n poiskytkentä	`--no-cabac`	CABAC pois päältä	Laskennallisesti liian raskas
I-makrolahkojen käyttö P-kuvissa	`--no-imbs`	I-makrolohkoja ei käytetty	
Trellis	`--trellis`	Ei käytössä	Laskennallisesti liian raskas
Kuvien määrä	`--frames`	300	

Taulukko 1: Täyspikselietsinnän parametrit

Tulokset kokopikselietsinnästä käyttäen alipikselietsintätila nollaa on esitetty liitteenä olevassa kuvassa (Liite 1). Alipikselietsintätila nolla siis tarkoittaa tilaa, jossa alipikselietsintää ei käytetä laisinkaan.

Liitteenä olevasta kuvasta voidaan havaita, että kuvanlaadun (PSNR) ja bittinopeuden (kbps) suhde ei muutu juuri laisinkaan, vaikka kokopikselietsinätilaa vaihdettaisiin. Itse asiassa laskennallisesti yksinkertaisimmat tilat, eli timanttietsintä ja heksagonaalietsintä, tuottavat joskus jopa monimutkaisempia tiloja parempia tuloksia.

Varmuuden vuoksi samaa asiaa on tutkittu myös alipikselietsintätilassa 1 (yksi puolipikselikierros ja yksi neljännespikselikierros täyspikselietsinnän jälkeen, katso Liite 2 sekä alipikselietsintätilalla 2 (yksi neljännespikselikierros täyspikselietsinnän jälkeen ja yksi puolipikselietsintä jokaisen partition jaon jälkeen, katso Liite 3)). Niistäkin voidaan havaita, että eroa täyspikselietsintätilojen välillä ei juuri ole.

Lisäksi on pyritty varmistamaan, ettei kvantisointiparametrin asettaminen vaikuta tuloksiin. Seuraaviin testeihin asetettiin kvantisointiparametriksi (qp) 40, joka tuottaa PSNR-arvoltaan noin 30 dB tasoista videokuvaa eli langattomiin lähetyksiin suhteellisen kelvollista kuvaa. Niiden tulokset on esitetty liitteenä olevissa kuvissa (Liite 4, Liite 5 ja Liite 6). Niistä voidaan havaita jo pieniä eroja täyspikselietsintätilojen välillä, mutta niissäkin yksinkertaisimmat tilat eli timanttietsintä ja heksagonaalietsintä tuottavat parhaimpia tuloksia.

Näiden tulosten perusteella on siis turha käyttää monimutkaisempia täyspikselietsintätiloja näissä olosuhteissa.

4.2. Alipikselietsintätilat

Seuraavaksi oli tarkoitus selvittää mitä vaikutusta eri alipikselietsintätiloilla on kuvanlaatuun (PSNR) ja bittinopeuteen. Käytetyt parametrit olivat samoja kuin täyspikselietsinnässä (katso yllä oleva Taulukko 1).

Tulokset alipikselietsintöjen vertailusta kolmella eri testivideolla on esitetty liitteenä olevissa kuvissa (Liite 7, Liite 8 ja Liite 9). Näistä nähdään selvästi, että siirryttäessä alipikselietsintätilasta 0 tilaan 1 saadaan huomattava parannus kuvanlaadun (PSNR) ja bittinopeuden (kbps) suhteeseen. Siirryttäessä muissa tiloista pykälän ylöspäin parannus ei ole enää niin merkittävä.

Erojen tarkemmaksi selvittämiseksi laskettiin myös keskiarvot parannuksille laatu/bittinopeus-suhteille tiloista toiseen. Keskiarvot laskettiin siis kaikkien seitsemän testivideon kaikkien eri täysetsintäpikselitilojen kesken (kaikkiaan 35 eri arvon keskiarvo). Tulokset olivat seuraavat:

Keskiarvo parannuksista laatu/bittinopeus-suhteessa	0->1	1->2	2->3	3->4	4->5	5->6	6->7
	31,54 %	4,61 %	1,27 %	0,97 %	0,10 %	-2,69 %	0,00 %

Taulukko 2: Alipikselietsintätilojen vertailua, qp=24

Yllä olevasta taulukosta (Taulukko 2) havaitaan, että kannettavallakin laitteella kannattaa toteuttaa alipikselietsintätila 1, jos suinkin laskentateho sallii sen. Muista alipikselietsintätiloista saatava hyöty ei sen sijaan ole enää kovin merkittävä. Varmuuden vuoksi vastaavat luvut laskettiin myös kvantisointiparametrillä qp=40 saaduista tuloksista:

Keskiarvo parannuksista laatu/bittinopeus-suhteessa	0->1	1->2	2->3	3->4	4->5	5->6	6->7
	13,94 %	-3,05 %	1,33 %	-0,34 %	0,52 %	10,01 %	0,00 %

Taulukko 3: Alipikselietsintätilojen vertailua, qp=40

Ne osoittavat vielä paremmin miten langattomissa verkoissa vaatimattomalla laadulla (PSNR alle 30 dB) ei kannata toteuttaa muita alipikselietsintätiloja kuin 1.

4.3. 8x8 DCT approksimaation käyttö

X.264 –videokoodekissa on annettu mahdollisuus käyttää myös 8x8 näytematriisin DCT-approksimaatiota yhdessä 4x4 näytematriisin DCT-approksimaation kanssa. Kokeessa käytettiin muuten samoja parametrejä kuin täyspikselietsinnässä (Taulukko 1) paitsi osassa kokeesta käytettiin myös 8x8 DCT:tä ja osassa ei.

Tarkoituksena oli siis mitata miten laadun (PSNR) ja bittinopeuden (kbps) suhde muuttuu siirryttäessä 8x8 DCT approksimaation sallivasta tilasta vain 4x4 DCT approksimaatiota käyttävään tilaan. Tuloksena laskettiin keskiarvot kaikista laadun täyspikselietsintätiloista (5 kpl), alipikselietsintätiloista (8 kpl) ja testivideoista (8 kpl), yhteensä siis keskiarvo 320 arvosta. Tulokset olivat seuraavat:

kvantisointiparametri qp	qp=24	qp=40
Keskiarvo parannuksista laatu/bittinopeus-suhteessa siirryttäessä käyttämään 8x8 DCT-approksimaatiota	0,36%	-3,10 %

Taulukko 4 : 8x8 DCT approksimaation käytön tuloksia

Yllä olevan taulukon (Taulukko 4) mukaan ei ole osoitettavissa, että 8x8 DCT approksimaation käyttö parantaisi laatua. On myös laskennallisesti hankalaa tarkistaa kannattaisiko sitä käyttää, joten se on järkevää jättää kannettavasta laitteesta pois.

4.4. Krominanssiarvojen käyttö liikevektorien laskennassa

Tässä kokeessa käytettiin muuten samoja parametrejä kuin täyspikselietsinnässä (Taulukko 1) paitsi osassa kokeesta käytettiin myös krominanssiarvoja liikevektorien laskennassa ja osassa ei.

Tarkoituksena oli siis mitata miten laadun (PSNR) ja bittinopeuden (kbps) suhde muuttuu siirryttäessä krominanssiarvoja liikevektorin etsinnässä käyttämättömästä tilasta niitä käyttävään tilaan. Tuloksena laskettiin keskiarvot kaikista täyspikselietsintätiloista (5 kpl) ja testivideoista (8 kpl), yhteensä siis keskiarvo 40 arvosta. Tulokset olivat seuraavat:

Alipikselietsintätila	0	1	2	3	4	5	6	7	Keski-arvo
Keskiarvo parannuksista laatu/bittinopeus-suhteessa siirryttäessä käyttämään krominanssiarvoja	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,78 %	0,18 %	0,18 %	0,14 %

Taulukko 5: Krominanssiarvon testauksen parametrit

Yllä olevan taulukon mukaan vain alipikselietsintätiloilla 5 ja siitä ylöspäin krominanssiarvoilla oli merkitystä. Siten on järkevää jättää kannettavasta laitteesta krominanssiarvojen käyttö pois.

4.5. I-makrolohkojen käyttö P-kuvissa

Tässä kokeessa käytettiin muuten samoja parametrejä kuin täyspikselietsinnässä (Taulukko 1) paitsi osassa kokeesta käytettiin myös I-makrolohkoja P-kuvissa ja osassa ei.

Tarkoituksena oli siis mitata miten laadun (PSNR) ja bittinopeuden (kbps) suhde muuttuu siirryttäessä P-kuvissa I-makrolohkoja käyttämättömästä tilasta niitä käyttävään tilaan. Tuloksena laskettiin keskiarvot kaikista täyspikselietsintätiloista (5 kpl) ja testivideoista (8 kpl), yhteensä siis keskiarvo 40 arvosta. Tulokset olivat seuraavat:

Alipikselietsintätila	0	1	2	3	4	5	6	7	Keski-arvo
Keskiarvo parannuksista laatu/bittinopeus-suhteessa siirryttäessä käyttämään I-makrolohkoja	0.94 %	-0.06 %	-0.34 %	-0.37 %	-0.33 %	-0.29 %	0.86 %	0.51 %	0.11 %

Taulukko 6: I-makrolohkojen testauksen tulokset

Yllä olevista tuloksista näkee, että I-makrolohkojen käytöllä ei ole juurikaan merkitystä. Testauksessa niitä esiintyikin vain noin yhden makrolohkon verran keskimäärin kuudessatoista kuvassa. X.264 –videokoodekissa laskettiin kuitenkin melko monimutkaisesti tarvitaanko niitä. Siten kyseisen laskennan jättäminen pois kannettavasta laitteesta kannattaa.

5. X.264 – videokoodekin toteutus Catapult-C:llä

Catapult C-ohjelmalle syötetään laitekuvaus Mentor Graphics-yhtiön Catapult C-ohjelmointikielellä, ja ohjelmisto tuottaa siitä VHDL-kuvauksen. VHDL-kuvauksen toimivuus voidaan varmistaa joko saman yhtiön Modelsim-ohjelmalla tai muilla VHDL-simulointiohjelmilla. Tämän kappaleen kuvat ovat kuvankaappauksia Catapult C-ohjelmiston vuoden 2009 yliopistoversiosta, jollei toisin mainita.

Catapult C-ohjelmointikieli perustuu C/C++ -ohjelmointikieleen, joten ennen Catapult C-ohjelmointikieleen perehtymistä olisi suositeltava tuntea C/C++ -ohjelmointikielen perusteet. Eräs erittäin helppolukuinen ja kenelle tahansa ohjelmointia harrastamattomallekin sopiva peruskirja [10] kattanee hyvin tämän tarpeen.

C++ -ohjelmointikielessä ei kuitenkaan yleensä tarvitse kiinnittää kovin paljon huomiota erilaisten tietorakenteiden rajoitteisiin, kuten kvantisoinnin tuottamiin virheisiin ja ylivuotoihin. Lisäksi yleensä ei ole tarpeellista määritellä bittien tarkkuudella kuinka iso jokin tietorakenne on. Catapult C-ohjelmointikielessä nämä piirteet ovat kuitenkin oleellisia, jotta voitaisiin varmistaa reaalimaailmassa toimivan laitteen syntetisoituminen.

5.1. Catapult-C:n tietorakenteet

Catapult C-ohjelmiston valmistanut Mentor Graphics on laatinut suunnittelijaa varten ”Algorithmic C™” – tietotyypit, jotka mahdollistavat syntetisoituvuuden. Ne myös nopeuttavat laitteen simulointia Modelsim-ohjelmistolla. Taulukossa esiintyvä parametri W ilmoittaa luvun sananleveyden ja I kokonaislukuosan sananleveyden.

Tyyppi	Kuvaus	Määrittelyalue
Ac_int<W,false>	Etumerkitön kokonaisluku	$0 – 2^{W}-1$
Ac_int<W,true>	Etumerkillinen kokonaisluku	$– 2^{W-1}$ $- 2^{W-1} - 1$
Ac_fixed<W,I,false>	Etumerkitön kiinteän pilkun luku	$0 – (1 – 2^{–W})2^{I}$
Ac_fixed<W,I,true>	Etumerkillinen kiinteän pilkun luku	$(-0.5)\ 2^{I} – (0.5 - 2^{–W})2^{I)}$

Taulukko 7 : Algorithmic C -tietotyypit

Esimerkkejä:
Ac_int<4,false> muuttujan_nimi määrittelisi etumerkittömän kokonaisluvun, jossa on neljä bittiä (siis bbbb, esimerkiksi $0100=0100_2=4_{10}$)
Ac_fixed<4,0,false> muuttujan_nimi määrittelisi etumerkittömän kiinteän pilkun luvun, jossa on neljä bittiä murtuluvuille (siis .bbbb, esimerkiksi $.0100=.0100_2=0{,}25_{10}$)
Ac_fixed<7,4,false> muuttujan_nimi määrittelisi etumerkittömän kiinteän pilkun luvun, jossa on kolme bittiä kokonaisluvuille ja neljä bittiä murtuluvuille (siis bbb.bbbb, esimerkiksi $010.0100=010.0100_2=4{,}25_{10}$)

Käytännössä nämä määritykset on hyvä tehdä sopivassa otsikkotiedostossa (header file) esimerkiksi seuraavaan tyyliin:

```
#include ”ac_fixed.h”
typedef ac_int<5,false> i_type
typedef ac_int<8,false> p_type
```

Näin voidaan varsinaisessa tiedostossa käyttää muuttujien määrittelyissä sopivissa kohdissa tyyppejä i_type ja p_type, jotka on optimoitu sisältämään vain tarvittavan operaation tuloksen tallentamiseen vaadittava minimaalinen bittimäärä. Jos huomataan myöhemmin, että tarvitaankin lisää bittejä, tarvitsee vain muuttaa otsikkotiedoston määritelmiä.

Kvantisointi ja ylivuoto

Edellä esitetyissä ac_int –tyyppisissä kokonaislukurakenteissa ei kuitenkaan ole mahdollista määritellä kvantisoinnin tai ylivuodon hallintaa, vaan esimerkiksi kvantisoinnissa vähiten merkitsevän bitin oikean puoleiset bitit vain hylätään. Sen sijaan ac_fixed –tietotyypeissä voidaan määrittää erikseen kvantisointimoodi (AC_TRN mode) ja ylivuotomoodi (AC_SAT mode). Catapult – ohjelmisto huolehtii näiden moodien toteutuksesta lisäämällä tarvittava määrä ylimääräisiä komponentteja VHDL-kuvaukseen.

Väylärakenteet

Catapult C –ohjelmointikieli mahdollistaa myös monimutkaisten laitteiden mallintamisen useina erillisinä hierarkkisina lohkoina. Esimerkiksi voi olla tarpeen prosessoida dataa useammalla eri kellotaajuudella, jolloin suunnittelu laite täytyy jakaa useampaan lohkoon. Näiden lohkojen välisten väylien mallintamiseen voidaan käyttää erillisiä ”AC Channel” – tietorakenteita.

Niitä voidaan käyttää seuraavanlaisella määrittelyllä:

```
#include ”ac_fixed.h” // otsikkotiedoston mukaanotto

// väylien määrittely funktion määrittelyssä
Void funktio (ac_channel <int> &data_in, ac_channel <int > &data_out);
```

Väylien käyttö funktion sisällä tapahtuu seuraavasti:

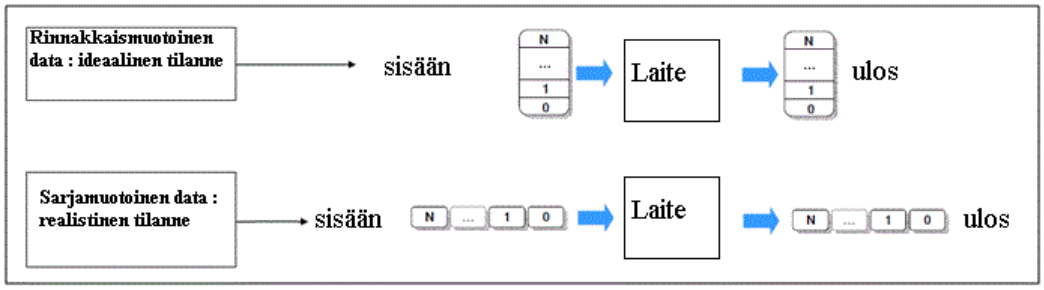
```
muuttuja = data_in(read); // lukuoperaatio
data_out.write(arvo); // kirjoitusoperaatio
```

Operaatio .read() luo automaattisen tarkistusrakenteen, joka tarkistaa onko dataa saatavilla luettavaksi. Mikäli suunnittelija haluaa itse päättää tästä kontrollirakenteesta, hän voi käyttää myös operaatiota .nb_read() .

5.2. Lohkoperiaate ja funktiokuvaukset

Kuten VHDL-kuvauksissa myös Catapult C-kuvauksissa ohjelmakoodista syntetisoidaan toiminnallisia lohkoja, joilla on omat väylänsä ja kontrollirakenteensa. VHDL-kuvauksissa näiden lohkojen portit on kuitenkin kuvattu eksplisiittisesti, kun taas Catapult-ohjelmisto joutuu päättelemään Catapult C-koodin funktiokuvausten ja itse funktioiden sisällön perusteella tarvittavat portit ja väylät.

Siksi ohjelmoitaessa Catapult C-ohjelmointikielellä on pohdittava tarkkaan miten dataa oikein käsitellään funktiokuvauksessa: esimerkiksi luetaanko tietyn muistirakenteen sisältöä sarja- vai rinnakkaismuotoisesti. Asiasta on esitetty havainnollinen kuva kirjallisuudessa [11]:



Kuva 16: Sarja- ja rinnakkaismuotoiset väylät (perustuu kirjallisuuteen [11])

Jos halutaan käyttää sarjamuotoista väylää, tietorakenteista täytyy hakea tietoa yksi kerrallaan, esimerkiksi näin:

```
for(i=0;i<64;i++)
  tietorakenne[i] = i + 3;
```

Vaihtoehtoisesti voidaan käyttää eksplisiittisesti ac_channel – väylämäärityksiä, jotka pakottavat sarjamuotoiseen tiedonsiirtoon eri lohkojen välillä.

5.2.1 Ongelmatapauksia ja aloittelijoiden virheitä

Edellisissä kappaleissa kuvatut määritelmät löytyvät hyvin selostettuna Catapult C-ohjelmiston manuaaleista. Kuitenkin on myös olemassa tiettyjä tietääkseni dokumentoimattomia piirteitä, jotka voivat aiheuttaa aloittelevalle suunnittelijalle päänvaivaa. Tulevien suunnittelijoiden työn helpottamiseksi lienee hyvä kirjata näitä kompastuskiviä. Osa tosin on melko triviaaleja ja saattaa koskea vain käyttämääni Catapult C-ohjelmiston vuoden 2009 yliopistolisenssi – versiota.

Tapaus 1: Tietorakenteen koon pakollinen määrittely

Perinteisessä C/C++ -tyylisessä ohjelmoinnissa funktiolle annetaan vain osoitin taulukkoon, ei välttämättä tietoa taulukon koosta. Siis tyyliin:

```
#pragma hls_design top
void design(m_type *memory, ac_channel<m_type> &output_channel)
```

jota sitten kutsutaan testipenkistä tyyliin :

```
m_type memory[SIZE];
ac_channel<m_type> output_channel;
CCS_DESIGN(design)(memory,output_channel);
```

Jos määritettelytiedostossa own_defs.h määritellään muistin kooksi (#define SIZE) yli 1024, Catapult valittaa

```
# Error: design.cpp(4): Unable to reduce array size for variable 'memory', currently 1024 words (CIN-84)
```

Pienemmillä muisteilla Catapult osaa päätellä muistin koon oikein ja voidaan käyttää pelkkiä osoittimia taulukkoon.

Tapaus 2: Tietorakenteen koon asettaminen väärin

Suuremmilla taulukoilla taulukon koko pitää siis asettaa eksplisiittisesti tyyliin :
void design(m_type memory[SIZE], ac_channel<m_type> &output_channel)
Jos taulukon koon asettaa väärin, esim.
void design(m_type memory[SIZE-100], ac_channel<m_type> &output_channel)

ja sitten kuitenkin tekee operaatioita tuon alueen ulkopuolella, esim.

```
c_type i;
for(i=0;i<SIZE;i++)
    memory[i] = i;

// Write the results to an output channel
for(i=0;i<CMP_SIZE;i++)
    output_channel.write(memory[i]);
```

niin tämä johtaa ongelmiin.

Jos testipenkissä muistin koko on asetettu oikein
m_type memory[SIZE];
niin Catapultin C++ testipenkki ei huomaa ongelmaa ja Catapult tuottaa RTL-koodin ongelmitta. Virhe huomataan vasta Modelsimissä :
Error: Simulation FAILED @ 3018 ns

Joissain tapauksissa Modelsim vain kaatuu mystisesti jättäen viestin :
** Fatal: (SIGSEGV) Bad pointer access. Closing vsimk.
** Fatal: vsimk is exiting with code 211.

Tapaus 3: Tietorakenteen koon asettaminen liian pieneksi

Jos muisti alustetaan alun perinkin väärin pienemmäksi kuin oli tarkoitettu, näyttää siltä että scc_verify_top voi joutua C++ testipenkissä ikuiseen looppiin.

Asetetaan siis esim.
#define SIZE 20
#define CMP_SIZE 50
Alustetaan muisti testipenkissä kuten yllä
m_type memory[SIZE];

Suoritetaan sama testi pääohjelmassa (CCS_DESIGN), nyt vain CMP_SIZE on suurempi kuin SIZE.

```
for(i=0;i<SIZE;i++)
    memory[i] = i;

// Write the results to an output channel
for(i=0;i<CMP_SIZE;i++)
    output_channel.write(memory[i]);
```

Ja verrataan sitä testipenkissä:
stdc_design(results_ref); // tehdään STD-C:llä vastaavat operaatiot kuin Catapult-C:llä.

```
// Read the results from the output channel
for(i=0;i<CMP_SIZE;i++) {
    results[i] = output_channel.read();
}
errors = 0;
// Compare results
for(i=0;i<CMP_SIZE;i++) {
    if (results_ref[i] != results[i].to_int())
        errors++;
}
```

5.3. Catapult-C – kehitystyökalut

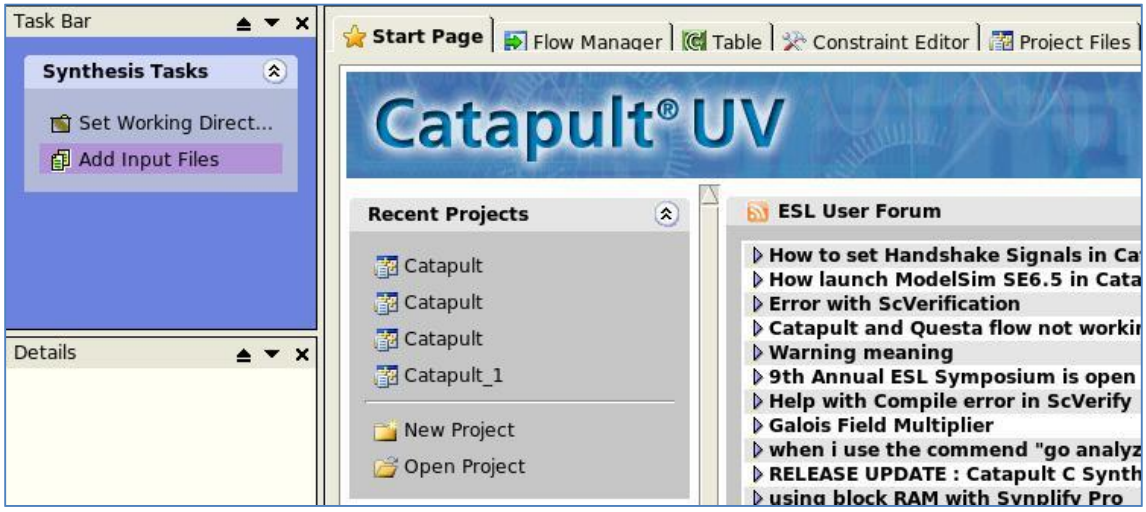
Edellisessä luvussa käsiteltiin muutamia käytännön ongelmia, joita suunnittelija saattaa kohdata ryhtyessään käyttämään tätä uutta korkean tason Catapult-C – ohjelmointikieltä. Päästyään alkukommelluksista ja projektin laajetessa suunnittelija voi kuitenkin törmätä melko nopeasti C++ -tyylisiin ongelmiin projektin hallinnasta. Ainakin vuoden 2009 Catapult C –ohjelmiston opiskelijaversiossa esimerkiksi oletetaan, että suunnittelijalla olisi toimivaa Catapult C-koodia syntetisoitavaksi. Itse ohjelmisto ei tarjonnut juurikaan kehitystyökaluja itse ohjelmointikoodin tuottamiseksi.

Onneksi voidaan käyttää valmiita C++ -ohjelmointikielen tuottamiseen kehitettyjä työkaluja, kunhan muistetaan edellisessä kappaleessa esitetyt erot Catapult C-ohjelmointikielen ja C++-ohjelmointikielen välillä. Linux/Unix – ympäristössä suosittelen gcc:tä, gdb:tä ja doxygeniä. Tarjolla on kuitenkin lukuisia muitakin työkaluja, kuten Microsoftin Windows-ympäristöön kehitetyt Visual C++-työkalut. Suositeltavaa on laatia myös rinnakkaisversio koodista puhtaana C++-toteutuksena, jolla varmistetaan testipenkissä ohjelman toiminta ennen Catapult C-ohjelmiston käyttöä.

5.3.1 Catapult

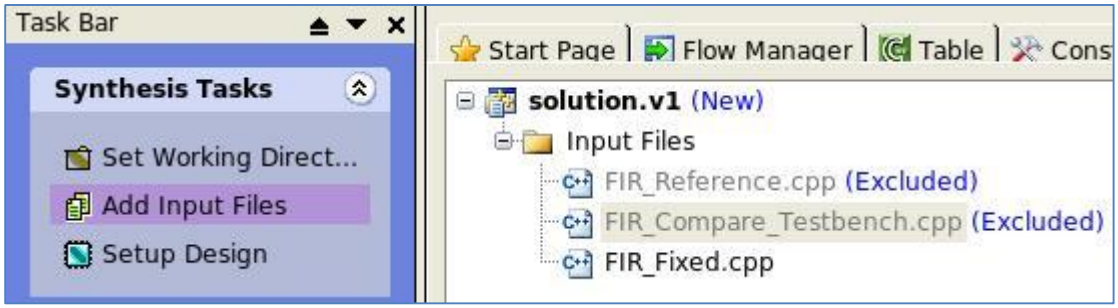
Kun haluttu ohjelmakoodi on saatu toimimaan halutunlaisesti C++-kääntäjän tuottamana ohjelmana, on aika simuloida sen toimintaa varsinaisella Catapult-ohjelmalla.

Käyttämässäni Catapult-ohjelmassa ohjelman päänäyttö näytti tältä:



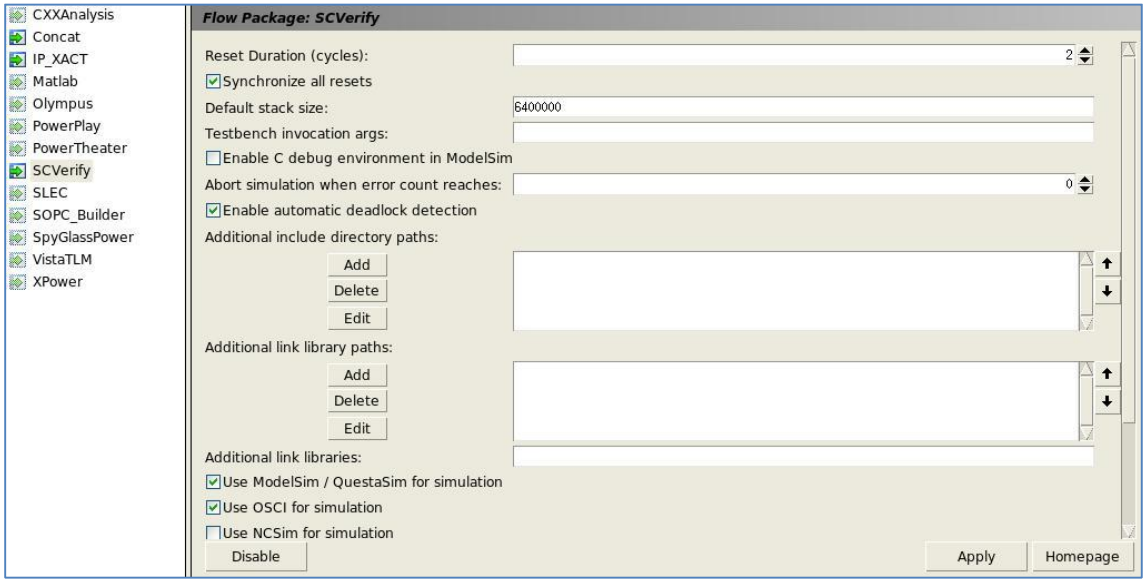
Kuva 17: Catapult-ohjelman päänäyttö

Valitsemalla ’Add input Files’ voidaan lisätä halutut ohjelmatiedostot:



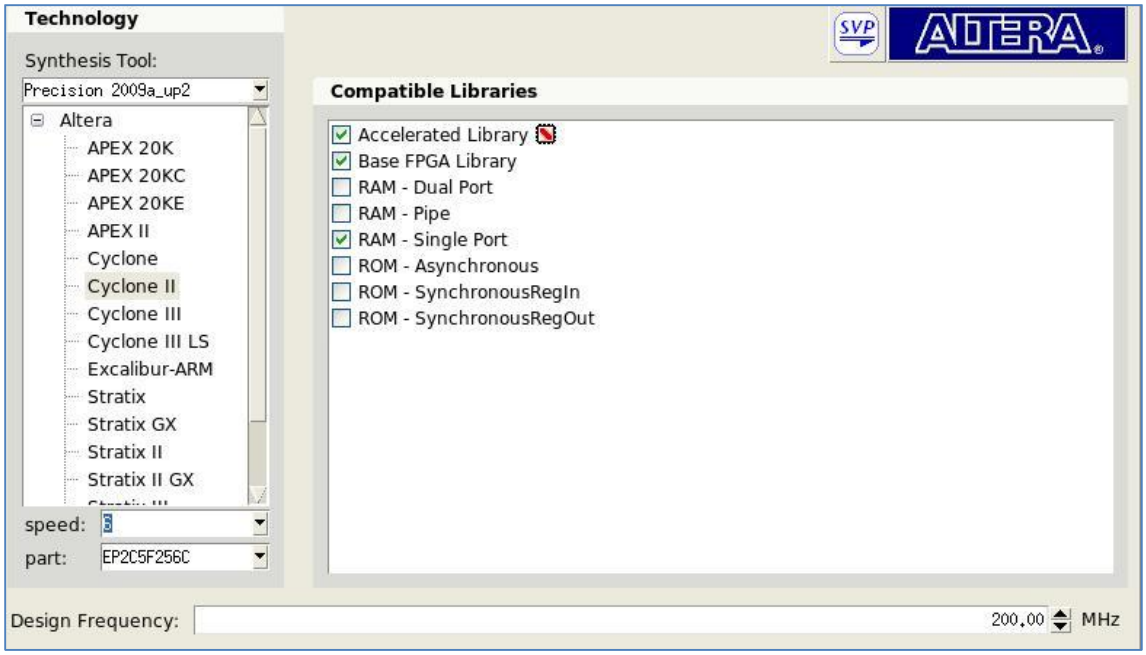
Kuva 18: Valitut ohjelmatiedostot

Käännettäväksi on valittu kaikki hakemiston .cpp-kooditiedostot, joista varsinaiseen testaukseen on valittu vain FIR_Fixed.cpp, joka sisältää varsinaisen Catapult C-ohjelman. Muut ohjelmatiedostot toimivat sen testipenkkinä. Ennen kääntämistä on myös syytä varmistaa, että Catapult-ohjelmiston SCVerify toiminta on valittu:



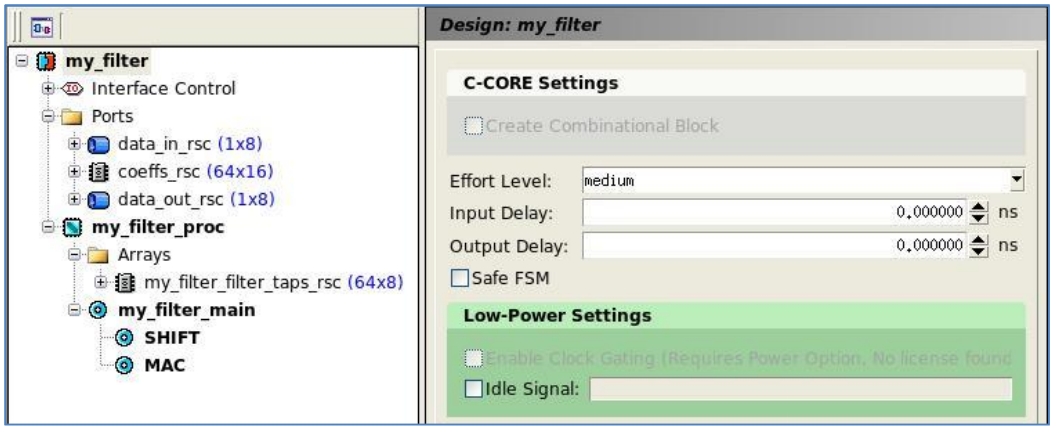
Kuva 19: SCVerify-näkymä

Kun tämä vaihe on suoritettu, valitaan ”Synthesis Tasks”-kohdasta Setup Design, joka kääntää koodin alustavasti. Sen jälkeen pitää valita haluttu CMOS-teknologia ja sopiva kellotaajuus:



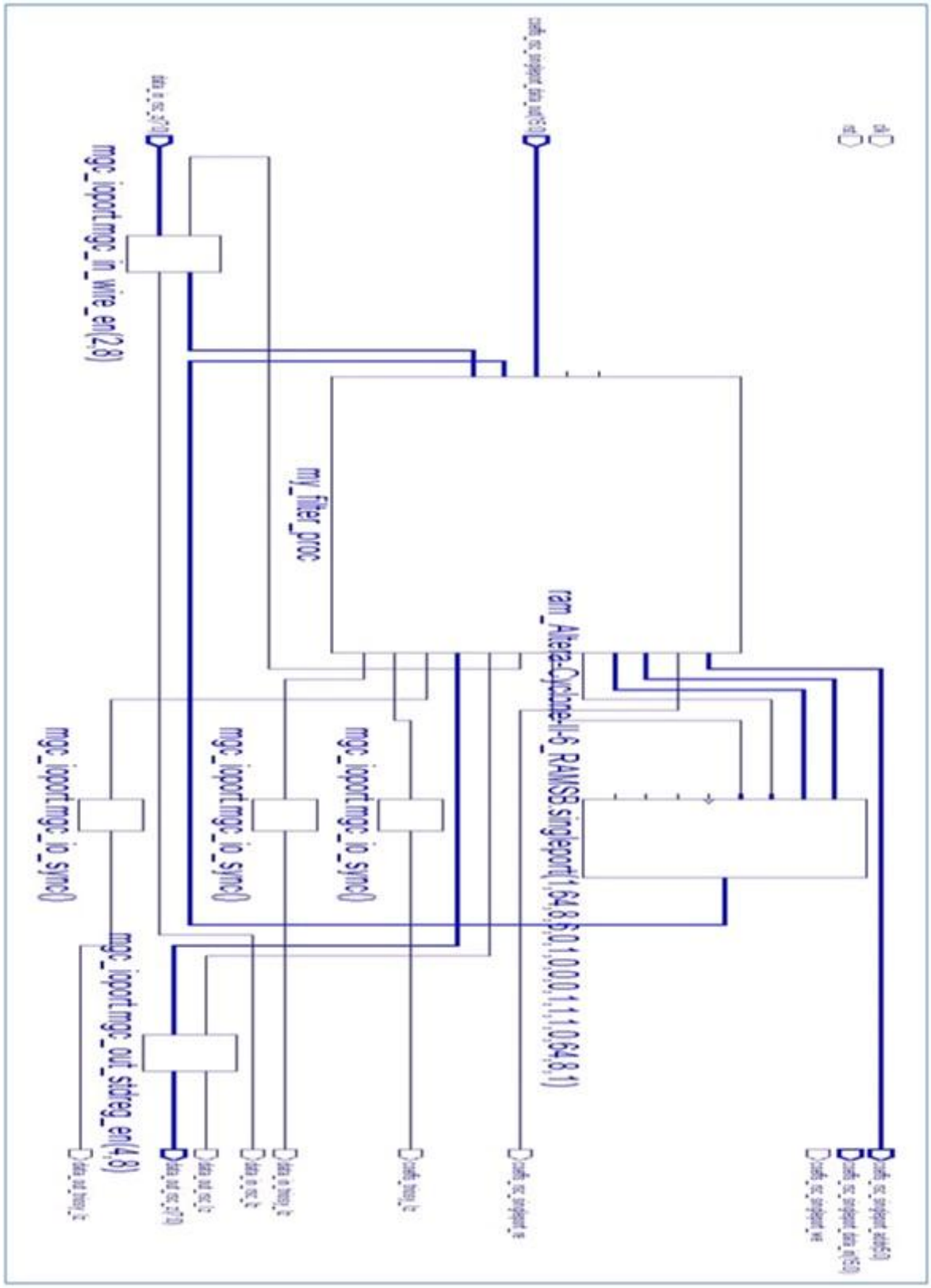
Kuva 20: Teknologian valinta

Yllä olevassa kuvassa (Kuva 20) on valittu malliksi Cyclone II FPGA-piiri ja kellotaajuus 200 MHz. Seuraavaksi valitaan kohta ’Architecture constraints’, jolloin Catapult C –ohjelma tutkii ohjelmatiedostoa arvioiden tarvittavat porttirakenteet sekä suoritettavat silmukat ohjelmatiedostossa. Esimerkkiohjelman tapauksessa näkymä oli tällainen:



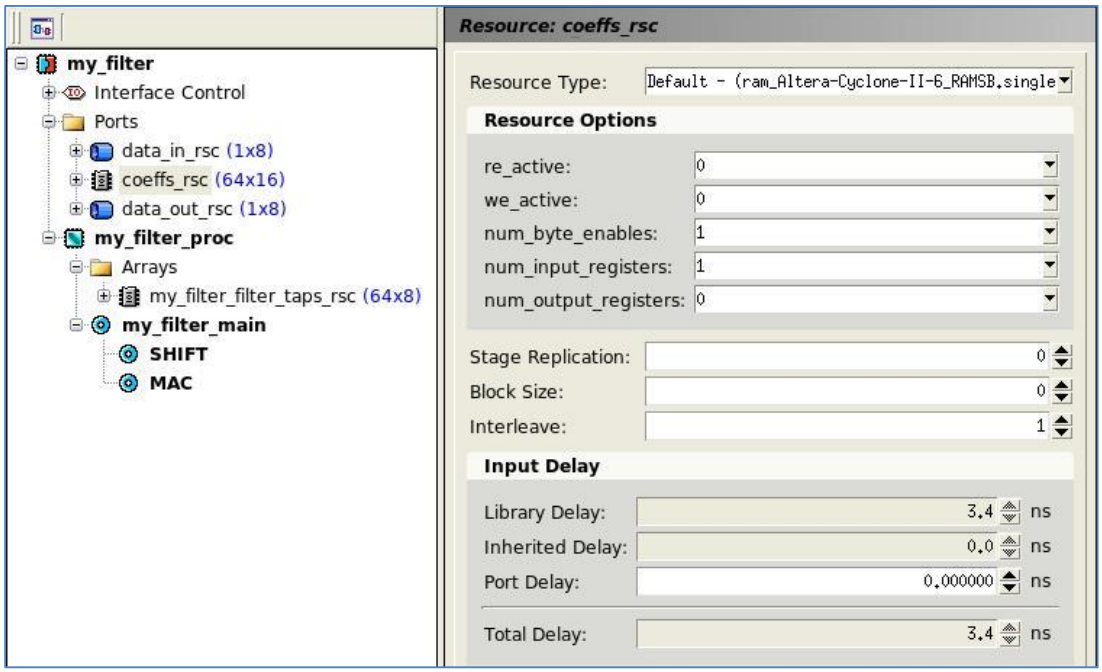
Kuva 21: Architecture Constraints -päänäkymä

Yllä olevan kuvan (Kuva 21) näkymästä erottuu myös miten kukin Catapult C-ohjelmakoodin funktiomäärittelyn argumentti on muutettu omaksi resurssikseen. Nämä resurssit edustavat todellisia komponentteja valmiissa toteutuksessa (Kuva 22), joten niiden ominaisuuksien säätäminen muuttaa oleellisesti toteutuksen suorituskykyä.



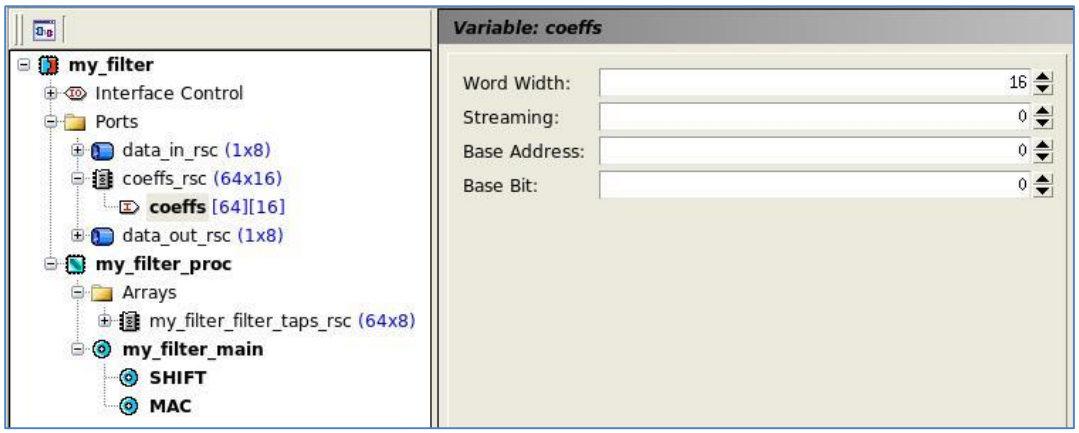
Kuva 22: komponenttinäkymä

Päänäkymästä erottuu myös miten Catapult C-funktioiden argumentit on muutettu omiksi I/O-porteikseen. Valitsemalla jonkun näistä porteista voimme tarkastella tarkemmin sen ominaisuuksia. Esimerkkinä on sisääntuloväylä, coeffs_rsc:



Kuva 23: Resurssiominaisuudet #1

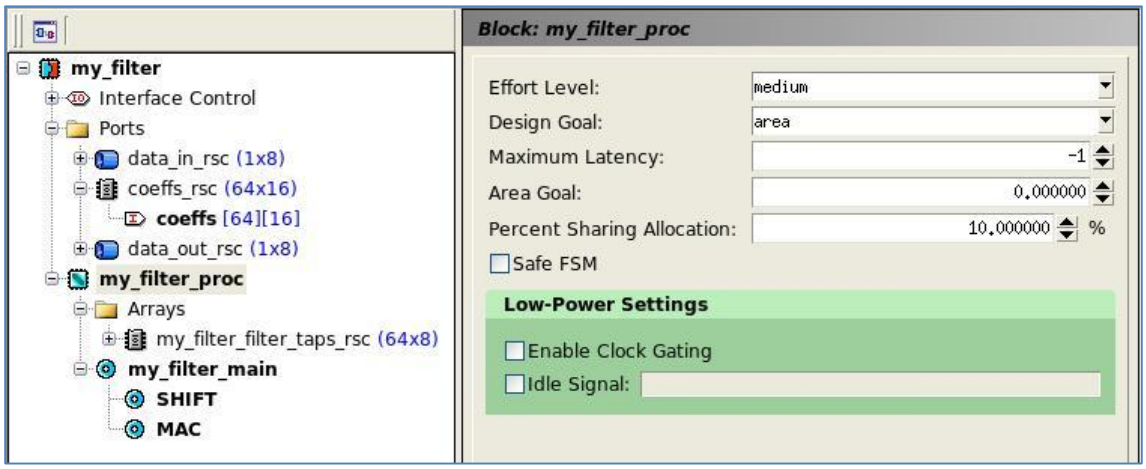
Yllä olevassa kuvassa (Kuva 23) voidaan määritellä tarkemmin resurssin kontrollilohkon sekä väylien ominaisuuksia. Valitsemalla näkymän numero 2 saadaan näkyviin myös sen sisäiset ominaisuudet, kuten muistikamman sananleveys tms.:



Kuva 24: Resurssiominaisuudet #2

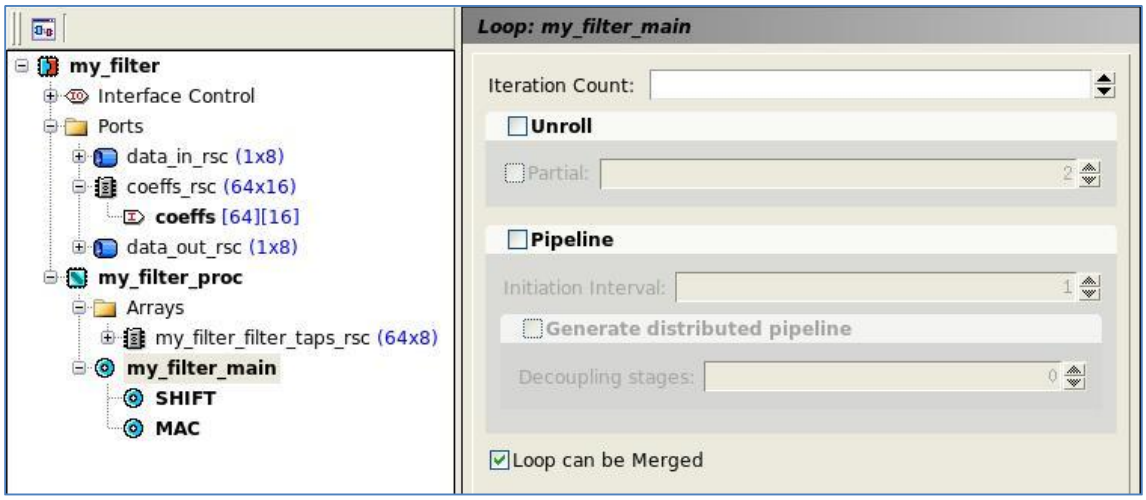
Muuttamalla näissä kuvissa (Kuva 23 ja Kuva 24) esiintyviä parametrejä voidaan joskus muuttaa ratkaisevastikin laitteen optimoitumista. Aloitteleva suunnittelija voi kuitenkin hämmentyä äkkiseltään valikkojen määrästä, jolloin monta oleellista muuttujaa voi jäädä huomiotta. Siksi kannattaa tarkastella rauhassa valikkoja ja pohtia miten niillä kannattaisi optimoida oman laitteensa toimintaa.

Valitsemalla pääprosessi (päälohko) saadaan seuraavanlainen näkymä:



Kuva 25: Pääprosessin (my_filter_proc) ominaisuuksia

Yllä olevassa kuvassa (Kuva 25) voidaan säätää yleisiä suunnitteluominaisuuksia, eli esimerkiksi pyritäänkö minimoimaan toteutuksen pinta-alaa vai latenssia. Siinä voidaan myös säätää maksimaaliset pinta-alat ja latenssit, joiden rajoissa toteutuksen on vähintään oltava. Lisäksi voidaan säätää kuinka paljon tietokoneen resursseja käytetään suunnitteluun sekä paljonko kellojaksosta varataan tilakoneen kontrollilogiikalle (sharing allocation).



Kuva 26: Pääfunktion ominaisuuksia

Yllä olevassa kuvassa (Kuva 26) erottuu miten pääfunktion (eli päälohkon) eri toiminnalliset alilohkot on nimetty kukin erikseen. Suunnittelija voi myös itse nimetä näitä lohkoja haluamallaan tavalla, esimerkiksi näin:

```
MAC:for (int i=0;i<16;i++)
```

Näin voidaan yksilöityjen nimien avulla erottaa selkeästi mistä toiminnallisesta lohkosta on kyse ja vaikuttaa sen ominaisuuksiin. Säädettävinä on yllä olevassa kuvassa (Kuva 26) neljä ominaisuutta: iteraatioiden lukumäärä, rinnakkaistaminen (unrolling), liukuhihnan käyttö ja silmukoiden yhdistäminen.

5.3.1.1 **Iteraatioiden lukumäärä**

Kaikki nämä ominaisuudet vaativat sellaista tarkkaavaisuutta ja harkintaa suunnittelijalta, johon tavallinen C++-ohjelmoija ei yleensä ole tottunut. Esimerkiksi on otettava huomioon, että Catapult-ohjelmisto ei pysty toteuttamaan rakennetta optimaalisesti, jos ei ole tiedossa täsmällisesti kuinka monta kertaa kukin silmukkarakenne voidaan suorittaa. Siis esimerkiksi seuraavanlainen perinteinen C++-ohjelmasekvenssi tuottaisi toteuttamisongelmia:

```
Void funktio (Int a[256], int &n, int &tulos) {
        Int acc=0;
        For (int i=0;;i++) {
           Acc+=a[i];
           If(i==n) break;}
        Result = acc;
}
```

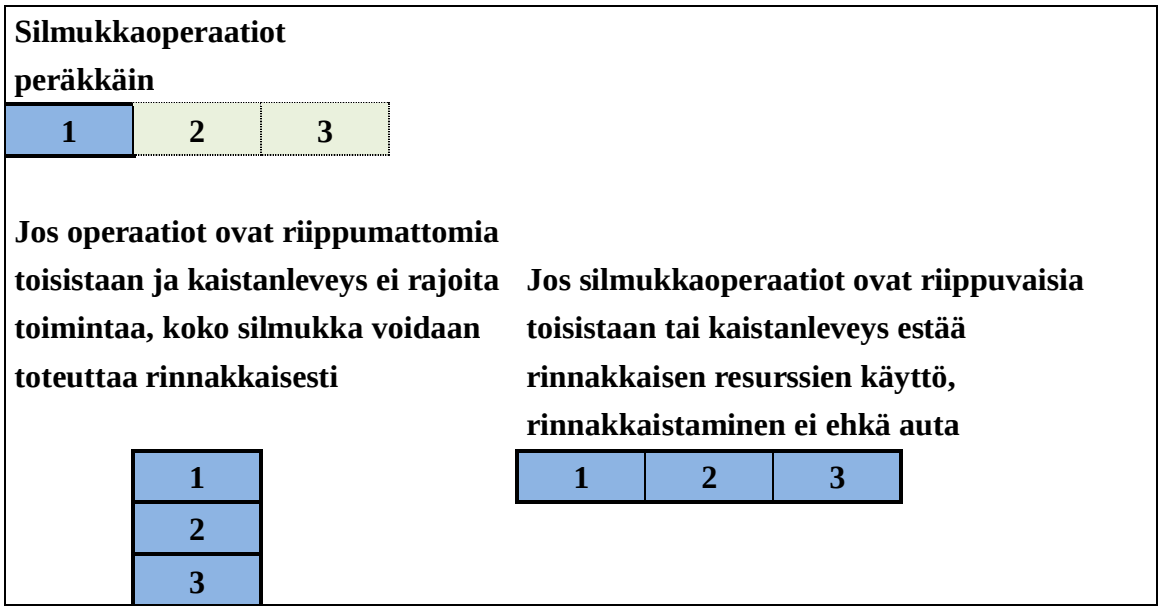
Yllä olevassa ohjelmasekvenssissä silmukan iteraatiomäärä riippuu siis ulkoisesta muuttujasta n, jonka arvo voi olla mitä tahansa int-tyyppinen kokonaisluku. Vaikkapa 32-bittisellä tietokoneella sen arvo voisi olla siis maksimissaan 2^{31}-1= 2147483647 eli Catapult C-ohjelmisto täytyy varautua suorittamaan kyseinen silmukka jopa yli 2 miljardia kertaa. Tämä tuottaisi luonnollisesti absurdeja syntetisointituloksia. Siksi on parempi asettaa eksplisiittisesti jokin maksimiraja iteraatioille, esimerkiksi näin:

```
Void funktio (Int a[256], int &n, int &tulos) {
        Int acc=0;
        For (int i=0;i<256;i++) {
           Acc+=a[i];
           If(i==n) break;
          }
        Result = acc;
}
```

Näin voidaan varmistaa, että silmukka suoritetaan maksimissaan 256 kertaa. Iteraatiomäärä voidaan myös asettaa Catapult-ohjelmiston valikoista, mutta suunnittelijan lienee parempi tottua ottamaan se jo alun alkaen huomioon ohjelmoinnissaan huolimattomuusvirheiden välttämiseksi.

5.3.1.2 **Rinnakkaistaminen**

Toinen kriittisistä ominaisuuksista oli rinnakkaistaminen (unrolling). Se mahdollistaa useampien silmukkaoperaatioiden laskemisen rinnakkain, mikäli se on mahdollista annettujen resurssien rajoissa (Kuva 27). Tämä yleensä parantaa laitteen latenssiarvoa ja suorituskykyä. Suunnittelijan on kuitenkin oltava tarkkana, että silmukkaoperaatiot eivät ole toisistaan riippuvaisia tai käytä samoja resursseja – muuten Catapult-ohjelmisto saattaa joutua luomaan erittäin raskaan kontrollirakenteen, mikä johtaa epäoptimaalisiin toteutuksiin.

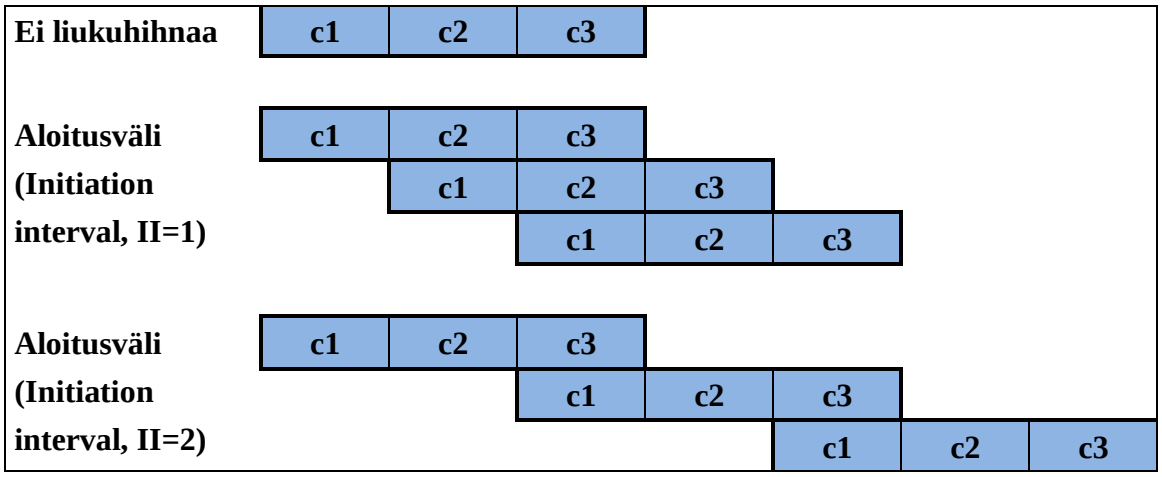


Kuva 27: Rinnakkaisoperaatiot (perustuu kirjallisuuteen [11])

Rinnakkaistamista pitää siis käyttää harkiten. Kannattaa aloittaa yksinkertaisimmista sisimmistä silmukoista jotka eivät juuri käytä rajallisia resursseja. Jos sisempää silmukkaa ei ole rinnakkaistettu, ei kannata myöskään rinnakkaistaa ulommaisempaa silmukkaa.

5.3.1.3 **Liukuhihnan käyttö**

Liukuhihnassa ideana on aloittaa seuraava silmukka-iteraatio ennen kuin edellisen iteraation tulos on valmistunut. Toisin kuin rinnakkaistamisessa, liukuhihna ei lisää laitteeseen rinnakkaista laskentakomponentteja vaan ainoastaan rekisterielementtejä, jotka tallentavat tuloksia. Siten laitteen kokonaispinta-ala ei kasva yhtä paljon rinnakkaistamisessa, mutta silti laitteen suoritusnopeus kasvaa.



Kuva 28: Liukuhihna (perustuu kirjallisuuteen [11])

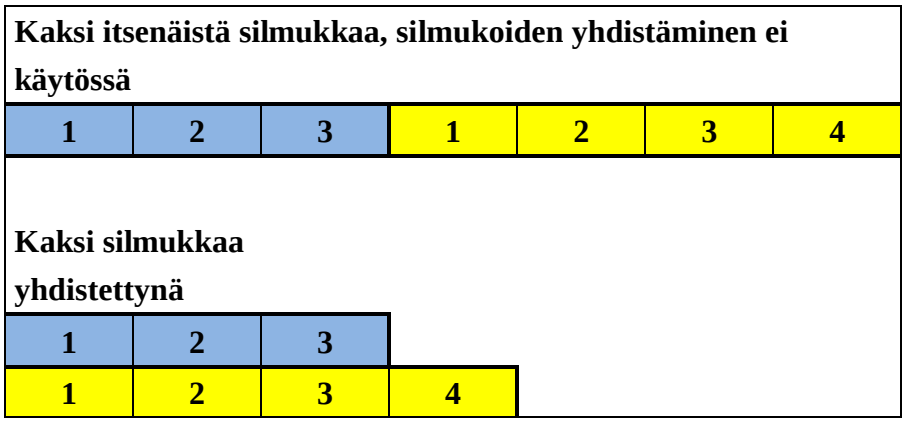
Yllä olevassa kuvassa (Kuva 28) laitteen toiminta koostuu siis kolmesta operaatiosta, jota nimitetään Catapult C-ohjelmointikielen mukaisesti C-askeliksi c1, c2 ja c3. Jos operaatioiden välillä ei ole riippuvuutta, voidaan uusi operaatio aloittaa joka kellojaksolla, jolloin aloitusväliksi (Initation Interval) voidaan säätää 1. Aloitusväliksi voidaan asettaa myös muita kokonaislukuja tarpeen mukaan.

Liukuhihnan käyttö lisää yleensä myös rekistereiden määrä ja siten myös viiveitä, jotka liittyvät niiden käyttöön. Käytetty teknologia voi siis rajoittaa liukuhihnan käyttöä, jos käytetty kellojakso ei anna varaa näiden rekistereiden käytölle. Suunnittelija joutuu siis usein pohtimaan tarkasti miten annetun teknologian rajoissa pystyisi mahdollisimman järkevästi toteuttamaan operaationsa – painotetaanko suorituskykyä vai komponenttimäärää.

Aiemmassa kuvassa (Kuva 26) oli myös mahdollisuus luoda hajautettu liukuhihna. Keskitetyssä liukuhihnassa koko liukuhihnaa hallitsee yksittäinen tilakone (FSM), joten yksittäisiä liukuhihnavaiheita ei voi viivästää tarvittaessa. Jos näitä viivästyksiä kuitenkin tarvitaan, voidaan käyttää hajautettua liukuhihnaa. Se tosin voi kasvattaa laitteen pinta-alaa huomattavasti, jopa 20%.

5.3.1.4 Silmukoiden yhdistäminen

Edelle esitetyssä kuvassa (Kuva 26) näkyi myös valikossa mahdollisuus silmukoiden yhdistämiseen. Se mahdollistaa usean itsenäisen prosessin suorittamisen rinnakkain, jolloin iteraatioiden määräksi tulee luonnollisesti pisimmän silmukan iteraatioiden määrä.

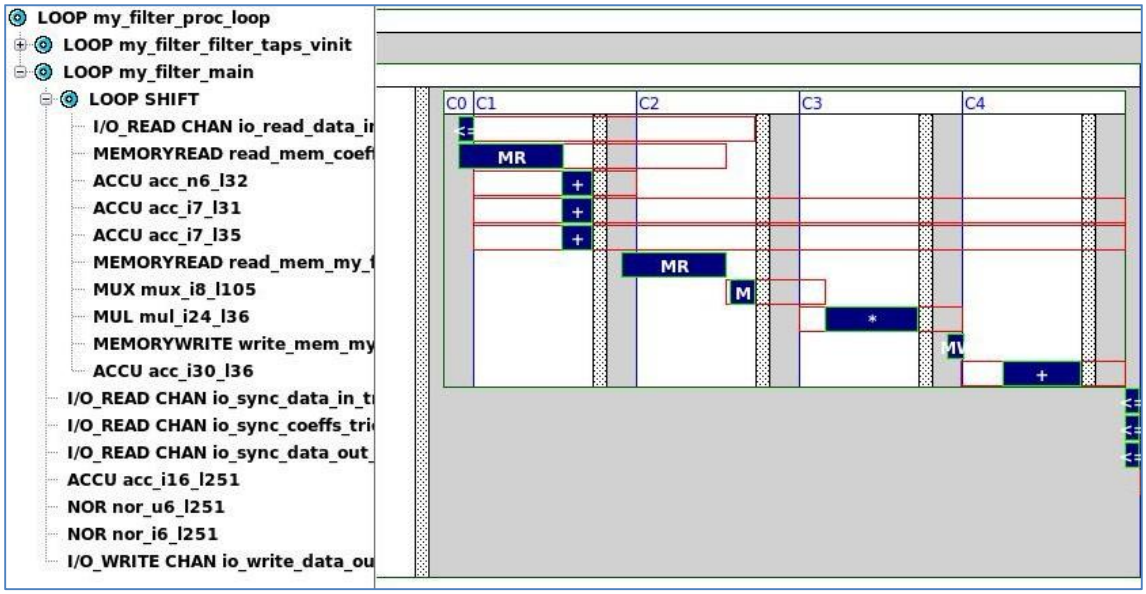


Kuva 29: Silmukoiden yhdistäminen (perustuu kirjallisuuteen [11])

Silmukoiden yhdistäminen on oletusasetuksena, mutta I/O-resurssit saattavat estää sen. Se myös toimii vain yksinkertaisilla rakenteilla ja yleisesti ottaen suunnittelijan on usein tehokkaampaa yhdistää operaatiot itse manuaalisesti suoraan koodissa.

5.3.1.5 **Ajoitus**

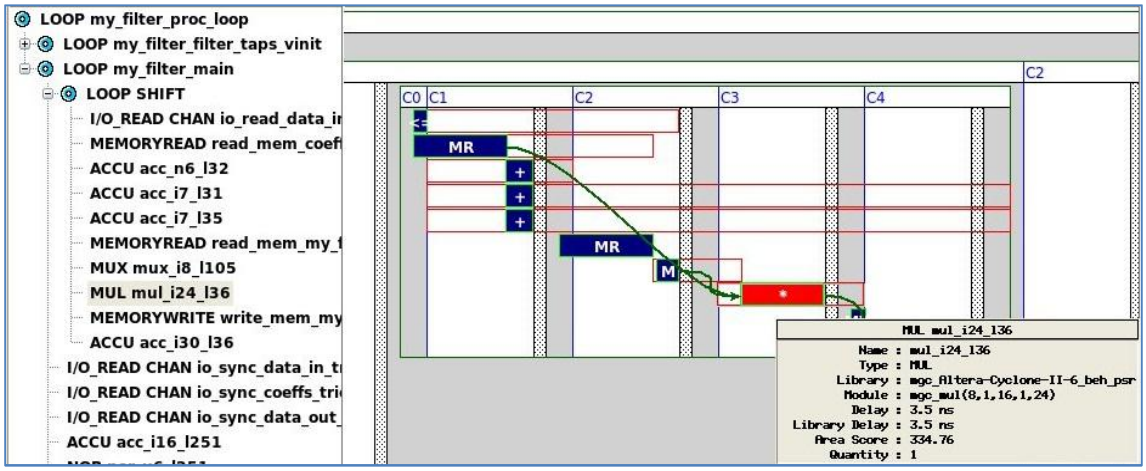
Kun edellisessä luvussa kuvatut arkkitehtuuriset rajoitukset on asetettu, voidaan edetä ajoitusosioon. Sen päänäyttö esimerkkiprojektissa näyttää seuraavalta:



Kuva 30: Catapult-ajoitus

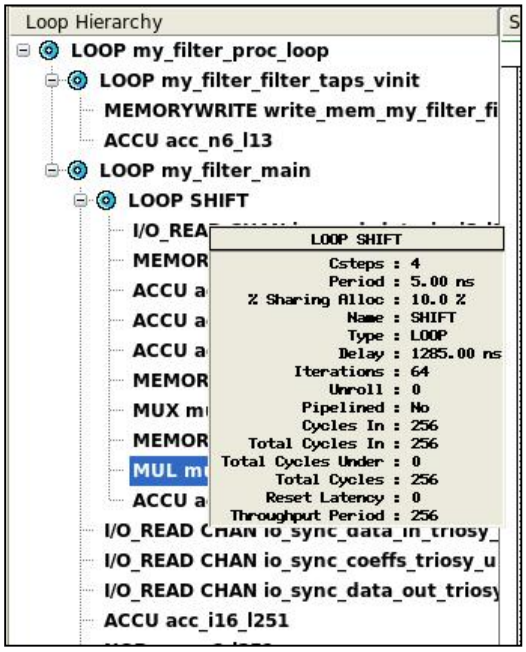
Yllä olevassa kuvassa (Kuva 30) on avattu auki koko prosessin ja sen eri funktion Ganttin kaavio. Kukin funktio on jaettu omiin C-askeleihinsa C0, C1 ja C2 jne. kuten aiemmassa liukuhihnakuvassa (Kuva 28). Samasta kuvasta on myös erotettavissa äärimmäisenä oikealla yleinen ajankäytön profiili (runtime profile), josta selviää kuinka suuri osuus ajasta menee kunkin funktion (lohkon) käytössä. Tässä tapauksessa huomataan, että suodintappien alustus (my_filter_filter_taps_vinit) vie vain pienen osan, 325 ns, ja suurimman osan ajasta (1285 ns) vie SHIFT-funktio.

Kolmanneksi on huomattava tyhjät kehykset väritetyllä alueella kuvattujen toimintojen ympärillä. Ne kuvaavat sallittuja sijoitusmahdollisuuksia ko. operaatioille, eli toisin sanoen kuinka paljon niitä voitaisiin aikaistaa tai viivästää jotta ne ehtisivät valmiiksi myöhempiä, niistä riippuvia operaatioita varten. Valitsemalla yksittäinen operaatio (tässä tapauksessa kerronta-operaatio MUL) saadaan seuraavanlainen kuva:



Kuva 31: Yksittäisen operaation ominaisuuksia

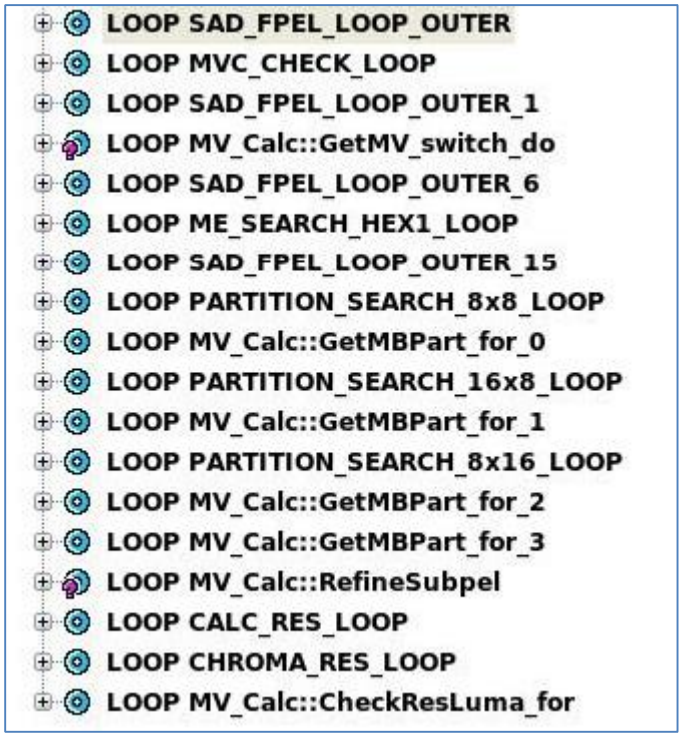
Yllä olevasta kuvasta (Kuva 31) havaitaan, että ko. operaation kesto on 3,5 ns ja ko. kerrontayksikkö veisi n. 335 μm^2 tilaa valitulla prosessilla. Lisäksi vihreät nuolet kuvaavat mistä operaatioista kerronta-operaatio on riippuvainen ja siitä lähtevä nuoli kuvaa mitkä tulevat operaatiot ovat siitä riippuvia. Siirtämällä kursoria SHIFT-funktion päälle saadaan seuraavat kuva:



Kuva 32: Funktion ominaisuudet ajoituksessa

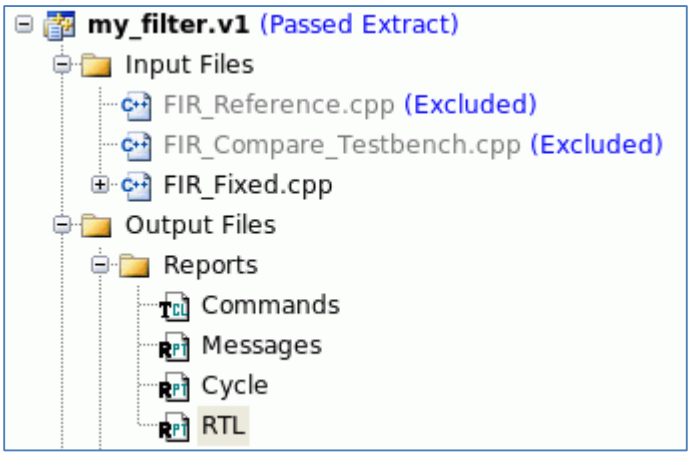
Yllä olevasta kuvasta (Kuva 32) havaitaan, että se koostuu neljästä C-askeleesta, joiden kunkin pituus on 5 ns, eli yksi kellojakso valitulla 200 MHz kellotaajuudella. Cycles In –osuus ilmoittaa monellako kellojaksolla luetaan dataa ja Cycles Under-ilmoittaa montako kellojaksoa sitä käsitellään. Iteraatioiden määrä on 64 eli yhteensä käytetään 256 kellojaksoa. Lisäksi selviää, että rinnakkaistamista tai liukuhihnaa ei ole käytetty, joten voidaan olettaa näiden ominaisuuksien paranevan huomattavasti niiden käytöllä. Ajoituksen tarkastelu antaakin yleensä huomattavasti lisätietoa siitä miten laitetta voisi parantaa.

Valitsemalla ajoitusikkunassa vaihtoehto ”View Sorted->By type” voidaan kaikki silmukat saada mukavasti peräkkäin näkyviin kuten alla olevassa kuvassa. Sen perusteella voidaan arvioida kätevästi miten paljon kellojaksoja kussakin vaiheessa Catapult C-ohjelmointikoodia kuluu.



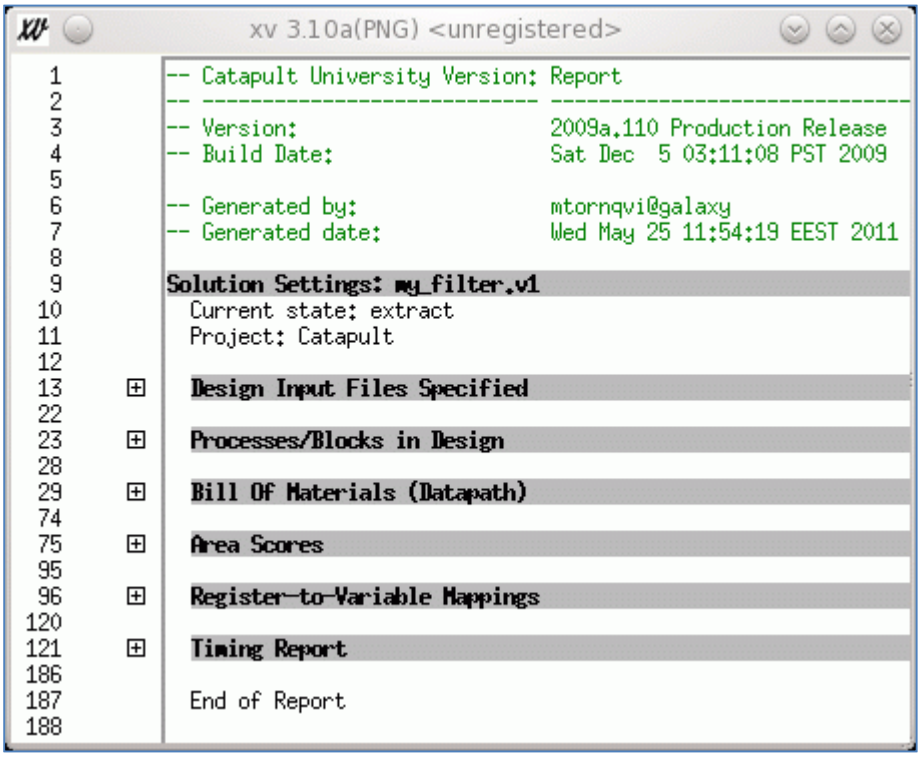
Kuva 33: Catapultin silmukkainäkymä

Ajoituksen tarkastelun jälkeen valitaan yleensä RTL-tiedoston luonti (Generate RTL). Kun se on valmis, voidaan tarkastella RTL-raporttia tarkemmin:



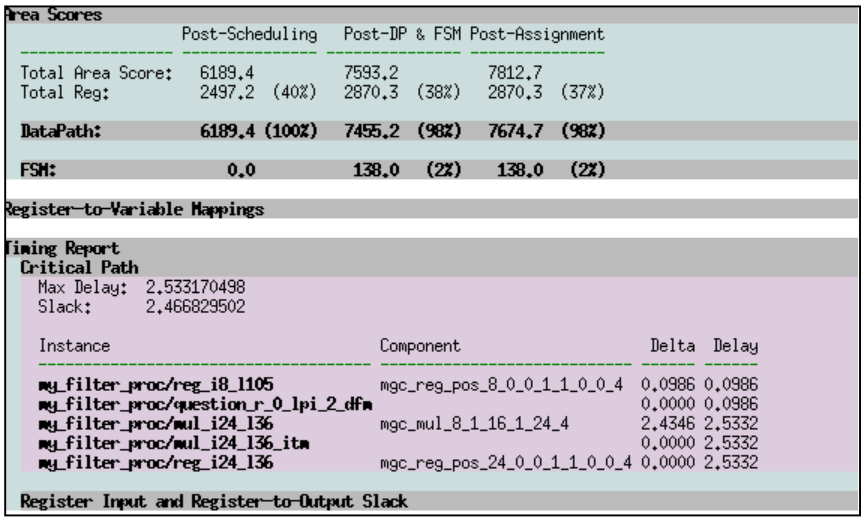
Kuva 34: RTL-raportin valinta

Valitsemalla yllä olevassa kuvassa (Kuva 34) RTL-raportin saadaan näkyviin seuraava kuva:



Kuva 35: RTL-raportti

Yllä olevassa kuvassa (Kuva 35) mielenkiintoisia kohtia ovat mm. ajoitusraportti (Timing Report), kokonaisraportti pinta-alasta (Area Scores) ja kokonaisraportti komponenteista (Bill of Materials). Näistä kaksi ensimmäistä antavat esimerkkitapauksessa seuraavat tiedot:



Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	6189.4	7593.2	7812.7
Total Reg:	2497.2 (40%)	2870.3 (38%)	2870.3 (37%)
DataPath:	**6189.4 (100%)**	**7455.2 (98%)**	**7674.7 (98%)**
FSM:	**0.0**	**138.0 (2%)**	**138.0 (2%)**

Register-to-Variable Mappings

Timing Report

Critical Path

Max Delay: 2.533170498

Slack: 2.466829502

Instance	Component	Delta	Delay
my_filter_proc/reg_i8_1105	mgc_reg_pos_8_0_0_1_1_0_0_4	0.0986	0.0986
my_filter_proc/question_r_0_lpi_2_dfm		0.0000	0.0986
my_filter_proc/mul_i24_136	mgc_mul_8_1_16_1_24_4	2.4346	2.5332
my_filter_proc/mul_i24_136_itm		0.0000	2.5332
my_filter_proc/reg_i24_136	mgc_reg_pos_24_0_0_1_1_0_0_4	0.0000	2.5332

Register Input and Register-to-Output Slack

Kuva 36: Ajoitusraportti

Yllä olevan kuvan (Kuva 36) ala-osassa näkyy kriittisen polun ajoitus. Ylä-osassa näkyy puolestaan mielenkiintoisesti miten pinta-alan arviot vaihtelevat merkittävästi ajoituksen jälkeisestä tilanteesta lopullisiin RTL-tiedostoista ekstraktoituihin arvioihin.

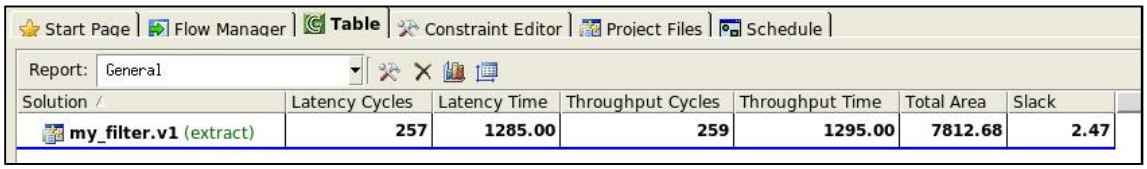
Kokonaisraportti komponenteista antaa puolestaan seuraavanlaisen kuvan:

Bill Of Materials (Datapath)

Component Name	Area Score	Delay	Post Alloc	Post Assign
[Lib: mgc_ioport]				
mgc_in_wire_en(2,8)	0.000	0.000	1	1
mgc_io_sync()	0.000	0.000	3	3
mgc_out_stdreg_en(4,8)	0.000	0.000	1	1
[Lib: mgc_sample-090nm_beh_dc]				
mgc_add(15,1,1,0,16,4)	141.969	1.079	1	1
mgc_add(30,0,24,1,30,3)	802.896	0.989	0	1
mgc_add(30,0,24,1,30,4)	522.215	2.778	1	0
mgc_add(6,0,1,1,6,4)	98.300	0.268	1	1
mgc_add(6,0,1,1,7,4)	102.578	0.766	2	2
mgc_and(1,2,4)	3.131	0.031	0	7
mgc_and(1,4,4)	6.298	0.067	0	2
mgc_and(30,2,4)	93.937	0.031	0	1
mgc_and(6,2,4)	18.787	0.031	0	2
mgc_and(8,3,4)	37.717	0.054	0	1
mgc_mul(8,1,16,1,24,4)	2652.370	2.435	1	1
mgc_mux(1,1,2,4)	5.716	0.133	0	2
mgc_mux(16,1,2,4)	91.453	0.133	0	1
mgc_mux(30,1,2,4)	171.475	0.133	0	1
mgc_mux(6,1,2,4)	34.295	0.133	0	5
mgc_mux(8,1,2,4)	45.727	0.133	1	2
mgc_mux1hot(6,3,4)	52.358	0.134	0	1
mgc_nand(1,8,4)	14.011	0.095	0	1
mgc_nor(1,2,4)	2.209	0.019	0	12
mgc_nor(1,3,4)	3.801	0.050	0	1
mgc_nor(1,6,4)	8.577	0.092	0	1
mgc_nor(1,8,4)	11.760	0.108	0	3
mgc_nor(6,2,4)	13.256	0.019	2	2
mgc_not(1,1)	2.822	0.010	0	9
mgc_or(1,2,4)	3.118	0.048	0	2
mgc_or(1,6,4)	9.083	0.100	0	2
mgc_or(6,2,4)	18.710	0.048	0	4
mgc_reg_pos(1,0,0,1,1,0,0,4)	18.776	0.099	0	6
mgc_reg_pos(16,0,0,1,1,0,0,4)	300.412	0.099	0	2
mgc_reg_pos(24,0,0,1,1,0,0,4)	450.618	0.099	0	1
mgc_reg_pos(30,0,0,1,1,0,0,4)	563.273	0.099	0	1
mgc_reg_pos(6,0,0,1,1,0,0,4)	112.655	0.099	0	5
mgc_reg_pos(8,0,0,1,1,0,0,4)	150.206	0.099	0	3
[Lib: ram_sample-090nm-dualport_beh_dc]				
RAM_dualRW(1,64,8,6,0,1,0,0,0,1,1,8,64	0.000	0.010	1	1
RAM_dualRW(3,64,16,6,0,1,0,0,0,1,1,16,	0.000	0.010	1	0
TOTAL AREA (After Assignment):	7674.684	12.000		

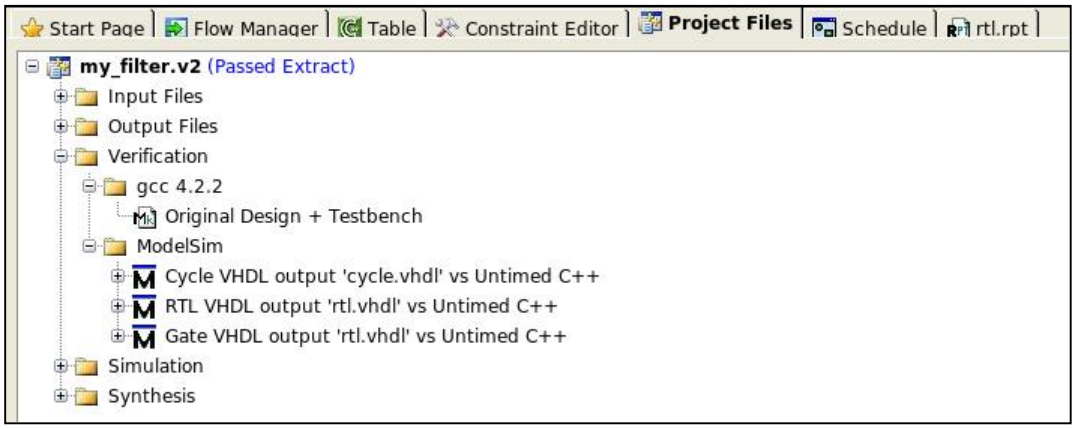
Kuva 37: Kokonaisraportti materiaaleista

Siitä havaitaan mielenkiintoisesti miten yksittäinen kertoja-yksikkö (mgc_mul) vie yksinäänkin lähes kolmanneksen pinta-alasta. RTL-raportin lisäksi voidaan valita myös tulosten yleinen tarkastelu kohdasta ”Table”:



Kuva 38: Taulukko tuloksista

Siitä nähdään tietoja latenssista, suorituskyvystä ja kokonaispinta-alasta. Lisäksi on tieto ns. slack-ajasta eli siitä kuinka paljon kriittisellä polulla jää ylimääräistä aikaa sen jälkeen kun prosessit on suoritettu annetussa kellojakson pituudessa. Tulosten tarkastelun jälkeen voidaan siirtyä verifiointiin:

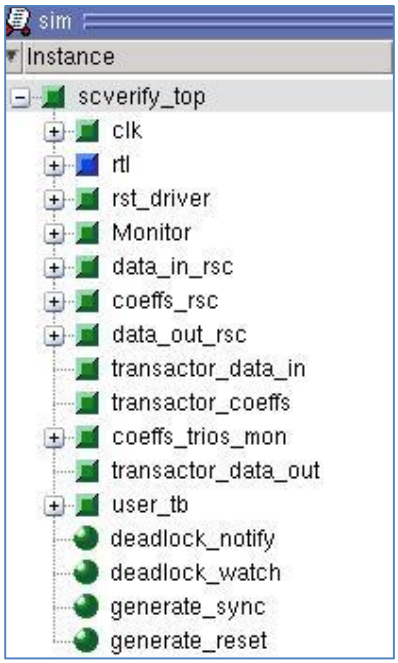


Kuva 39: Verifiointi

Jos SCVerify oli valittuna aiemman kuvan (Kuva 19) osoittamalla tavalla, voidaan klikkaamalla valita ”Original Design + Testbench”, joka suorittaa Catapult C-lähdekoodiin manuaalisesti koodatun testipenkin. Klikkaamalla puolestaan kohtaa Modelsim->RTL VHDL Output ’rtl.vhdl’ vs Untimed C++ voidaan simuloida toteutuksen ajoitusta.

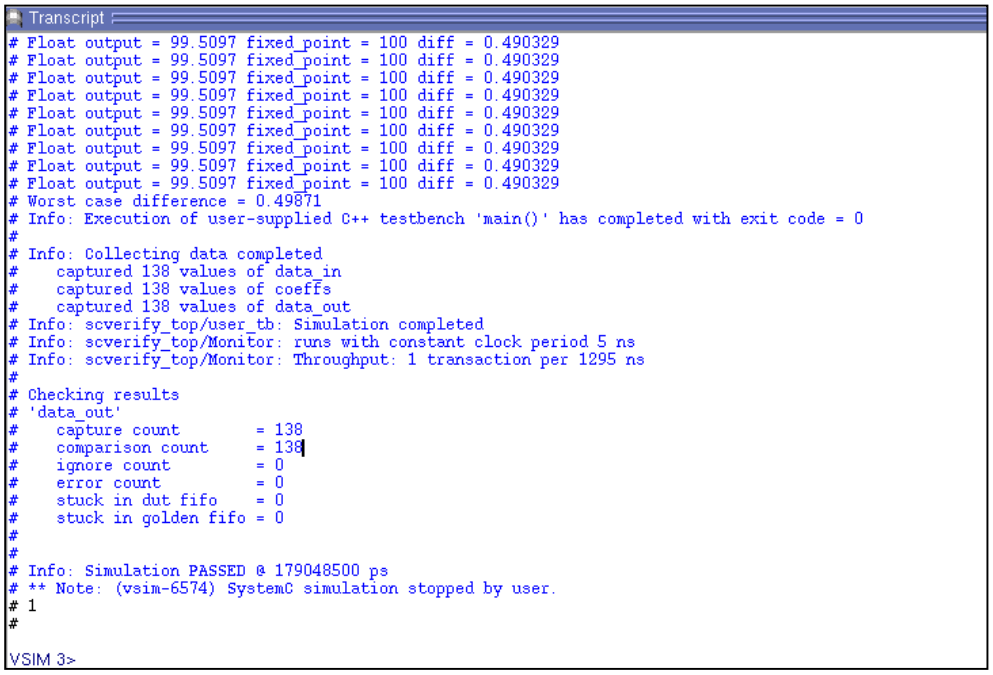
5.3.2 Modelsim

Modelsim-simulaatiossa saadaan näkyviin seuraavanlainen näkymä:



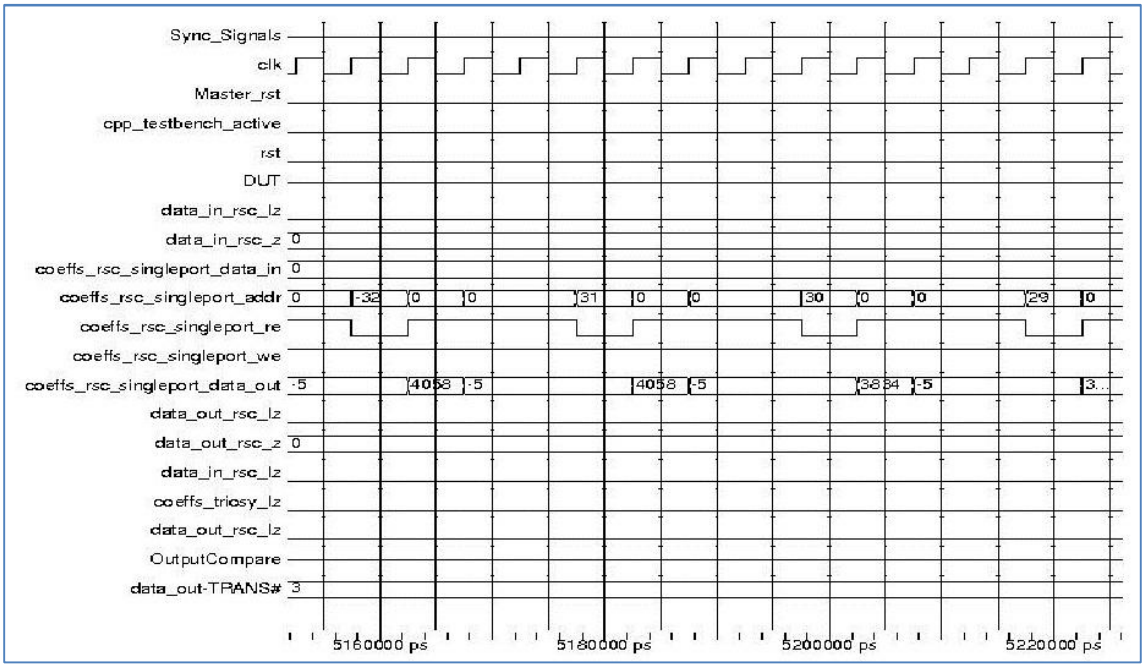
Kuva 40: Modelsim-päänäkymä

Yllä olevasta kuvasta (Kuva 40) näkyy mm. kellosignaali (clk) ja sovelluksen resursseja (data_in_rsc,coeffs_rsc, data_out_rsc). Valitsemalla simulate->Run->Run –all voidaan testata simulaatiolla sovelluksen toimivuutta:



Kuva 41: Verifiointi Modelsimillä

Yllä olevasta kuvasta (Kuva 41) nähdään, että esimerkkisovellus on todettu toimivaksi simuloinnissa. Modelsim-simulointi onkin erittäin tarpeellinen työkalu, sillä pelkkä testaus Catapult C-testipenkeillä ei pysty varmistamaan, että sovelluksen ajoitus todella toimisi käytännössä. Modelsim-simulaatiolla voidaan tutkia tuloksia aaltomuodossa:



Kuva 42: Modelsim aaltomuoto-ikkuna

5.4. Valmis VHDL-kuvaus

Catapult -ohjelma kääntää valmiin ohjelman tiivistetyksi VHDL-kuvaukseksi concat_rtl.vhdl, joka sijoitetaan suunnitteluhakemiston alaiseen hakemistoon Catapult/<pääfunktion_nimi.version_numero>. Esimerkiksi, jos pääfunktion nimi olisi mb_calc, VHDL-kuvaus olisi hakemistossa Catapult/mb_calc.v1/concat_rtl.vhdl. Pääfunktio määritellään sijoittamalla seuraava direktiivi juuri ennen pääfunktiota:
#pragma hls_design top

VHDL-kuvaustiedosto concat_rtl.vhdl sisältää pääfunktion määrittelyn seuraavaan tyyliin:

```
ENTITY pääfunktio IS
  PORT(
....
);
END pääfunktio;
```

Porttikuvauksen osiossa on tietorakenteille määritelty omat väylänsä. Esimerkiksi tietorakenteelle sram oli määritelty seuraavat väylät:

```
sram_singleport_data_in : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
sram_singleport_addr : OUT STD_LOGIC_VECTOR (14 DOWNTO 0);
sram_singleport_re : OUT STD_LOGIC;
sram_singleport_we : OUT STD_LOGIC;
sram_singleport_data_out : IN STD_LOGIC_VECTOR (15 DOWNTO 0)
```

eli datan sisään- ja ulostuloväylä, lukuoperaation salliva väylä sekä kirjoitusoperaation salliva väylä. Jos tietorakenteita halutaan yhdistää yhdeksi muistirakenteeksi, se pitää tehdä seuraavanlaisella käskyllä itse koodissa:

```
#pragma resource sram: variables = "mv res_frame cmp_frame cur_frame"
```

AC_CHANNEL-kanavien avulla määritettyihin portteihin määritellään väylät Catapultin luomassa VHDL-koodissa seuraavaan tyyliin:

```
kanavan_nimi_z : OUT STD_LOGIC;
kanavan_nimi_lz : OUT STD_LOGIC;
```

missä _z-päätteinen väylä on varsinainen väylä ja _lz –päätteinen väylä on Modelsimin simulointitarkoituksiin operaatioiden etenemisestä ilmoittava väylä. Käyttäjä voi luoda oman ylätason VHDL-kuvauksensa ja siirtää PORT MAP-käskyn avulla ko. väylät ylätason kuvaukseen tai käyttää Catapultin luomaa V HDL-kuvausta sellaisenaan.

5.5. Syntetisointituloksia Catapultista

Toteutin Catapult C:llä seuraavat X.264-videokoodekin ominaisuudet, jotka arvelin ohjelmistopuolen simulointitulosten perusteella (kappale 4) lopputuotteessa pystyttävän toteuttamaan tai joista olisi huomattavaa hyötyä tuotettavan videokuvan laadun ja bittinopeuden suhteelle.

Ominaisuus	Huomioita
Kuvan koko	176x144 pikseliä (QCIF), muutettavissa
Täyspikselietsintä	Timanttietsintä ja heksadiagonaalietsintä
Makrolohkojaosta päättäminen	16x16, 16x8, 8x16 ja 8x8-makrolohkojaot
Alipikselietsintä	Yksi puolipikselikierros ja neljännespikselikierros
Residuaalin laskenta	DCT-approksimaatio ja kvantisointi

Taulukko 8: Catapult-C:llä toteutetut ominaisuudet X.264-videokoodekista

Tarkoituksena oli, että ohjelma hakisi lopputuotteen muistista parhaillaan enkoodattavan kuvan ja referenssikuvan. Lopputuloksena muistiin tallennettaisiin residuaalikuva ja liikevektorit. Catapultissa oli laboratoriossa käytettävissä olevaa FPGA-piiriä (katso tarkemmin kappale 6) varten omat generointimenunsa, joten VHDL-tulokset riippuvat myös kyseisen laitteen ominaisuuksista.

Kokeilin ensin millaisella kellotaajuudella tuotteen saadaan toimimaan. Osoittautui, että muistin osoiteväylä muodostaa pullonkaulan kellotaajuuden asettamisessa. Catapultin RTL-raportin mukaan tarvitaan vähintään noin 32 nanosekunnin kellojakso, jotta muistin osoiteväylää pystytään käyttämään. FPGA-piirillä tehtyjen kokeiden perusteella (katso kappale 6) arvelin 40 nanosekunnin kellojakson (25 megahertsin kellotaajuuden) antavan riittävästi marginaalia. Marginaalia tarvittaan sillä Catapultin ajoitukset ovat muilla ohjelmilla tehtyjen kokeiden perusteella osoittautuneet vain suuntaa-antaviksi.

Loput kokeista tehtiin siis 25 megahertsin kellotaajuudella. Tarkastelin ensin täysin optimoimattoman ratkaisun tuloksia, joita on esitetty alla olevassa taulukossa.

Ominaisuus	Kellojaksoja	Osuus
Täyspikselietsintä/ makrolohko	24914	75,0 %
Alipikselietsintä / makrolohko	57	0,2 %
Residuaalin laskenta/makrolohko	320	1,0 %
SKIP-laskenta / makrolohko	500	1,5 %
Muuta toimintaa / makrolohko	7429	22,4 %
Yhden makrolohkon käsittely yhteensä	33220	100,0 %

Taulukko 9: Optimoimattoman Catapult-ratkaisun suoritusarvoja #1

Yllä olevassa taulukossa alipikselietsintä vie huomattavan vähän kellojaksoja, koska alipikselietsinnässä tehtiin vain yksi puolipikselikierros ja neljännespikselikierros etukäteen valmiiksi lasketuista neljännes- ja puolipikseleistä. Niiden laskenta kuten myös referenssikuvien rekonstruointi oletetaan tehtävän erikseen toteutettavassa X.264-videokoodekin dekoodaajassa. Muuhun toimintaan menee huomattavan paljon kellojaksoja, sillä FOR-silmukoiden nimeämiskeinolla ei pystynyt täydellisesti seuraamaan laskennan etenemistä eri ohjelmakoodin osissa. Muitakaan keinoja seurantaan ei kuitenkaan ollut käytettävissä. Muuhun toimintaan sisältyy myös Catapultin C-koodin kanavien lukua tms. toimintaa, joka jäi silmukoiden ulkopuolelle ja jonka osuutta laskennasta ei voitu tarkemmin täsmentää.

Yllä olevassa taulukossa mainittu SKIP-laskenta puolestaan on residuaalin laskennan tulosten tarkastelua. Jos huomataan, että makrolohkon residuaali vastaa tarpeeksi hyvin nollaa, voidaan merkitä koko makrolohko SKIP-tyyppiseksi eli siitä ei tarvitse tallentaa yhtään residuaalidataa.

Seuraavaksi tarkasteltiin millaiseen videodatan käsittelyyn näillä optimoimattomilta suoritusluvuilla pystytään 25 megahertsin kellotaajuudella. Tulokset on laskettu yllä olevasta taulukosta kertomalla yhden makrolohkon käsittelyyn kuuluva kellojaksojen määrä kunkin videokuvatyypin sisältämällä makrolohkojen määrällä.

Ominaisuus	Kellojaksoja	Aika (s)
QCIF-kuvan käsittely yhteensä	3288780	0,13
QCIF videokuvan sekunnin (25 kuvaa) käsittely	82219500	3,29
SQCIF-kuvan käsittely yhteensä	1594560	0,06
SQCIF videokuvan sekunnin käsittely	39864000	1,59
64x48-pikselin kuvan käsittely yhteensä	398640	0,02
64x48-pikselin videokuvan sekunnin käsittely	9966000	0,40

Taulukko 10: Optimoimattoman Catapult-ratkaisun suoritusarvoja #2

Tuloksista havaitaan, että optimoimattomalla ratkaisulla pystytään käsittelemään reaaliaikaisesti vain todella pientä videokuvaa, 64x48 pikselin videota. Tarvitaan siis selkeästi optimointia. Valitettavasti Catapult C –ohjelma ei pystynyt itse tuottamaan ajoituksen ja muistiresurssien rajoitusten vuoksi itse optimoituja ratkaisuja, joten kokeilin seuraavaksi täyspikselietsinnän rajoittamista timanttietsintään. Sillä saatiin seuraavia tuloksia:

Ominaisuus	Kellojaksoja	Osuus
Täyspikselietsintä/ makrolohko	9470	56,5 %
Alipikselietsintä / makrolohko	57	0,3 %
Residuaalin laskenta/makrolohko	320	1,9 %
SKIP-laskenta / makrolohko	500	3,0 %
Muuta toimintaa / makrolohko	6400	38,2 %
Yhden makrolohkon käsittely yhteensä	16747	100,0 %

Taulukko 11: Optimoidun Catapult-ratkaisun suoritusarvoja #1

Yllä olevan taulukon arvoista laskettiin vastaavasti suoritusarvoja eri resoluution videokuville 25 megahertsin kellotaajuudella:

Ominaisuus	Kellojaksoja	Aika (s)
QCIF-kuvan käsittely yhteensä	1657953	0,07
QCIF videokuvan sekunnin (25 kuvaa) käsittely	41448825	1,66
SQCIF-kuvan käsittely yhteensä	803856	0,03
SQCIF videokuvan sekunnin käsittely	20096400	0,80
64x48-pikselin kuvan käsittely yhteensä	200964	0,01
64x48-pikselin videokuvan sekunnin käsittely	5024100	0,20

Taulukko 12: Optimoidun Catapult-ratkaisun suoritusarvoja #2

Yllä olevasta taulukosta huomataan, että kyseisellä ohjelmakoodilla pystytään käsittelemään reaaliaikaisesti SQCIF (128x96 pikseliä) –videokuvaa. Se on riittävän hyvätasoista kuvaa kannettavalle laitteelle, joten seuraavaksi siirryttiin testaamaan toimintaa FPGA-piirillä.

6. **Simulointi FPGA-piirillä**

Valmista Catapult C –koodia simulointiin Altera -yhtiön valmistamalla Cyclone II FPGA-piirillä (piirisarja EP2C35F672C6, katso Liite 10), jota varten Catapult-ohjelmassa oli omat VHDL-generointimenut. Tässä kappaleessa esiintyvät kuvat ovat kuvankaappauksia Quartus II –ohjelmistosta, jollei toisin mainita.

FGPA-piirin tarkemmat ominaisuudet olivat seuraavanlaiset:

Ominaisuus	Arvo
Logiikkasoluja	33216
Maksimitaajuus	500 MHz
Prosessiteknologia	90 nm
Sisään- ja ulostulopinnejä	672
Nopeuskerroin	6
SRAM-muistia	512 kilotavua
Oskillaattorit	50 MHz ja 27 MHz

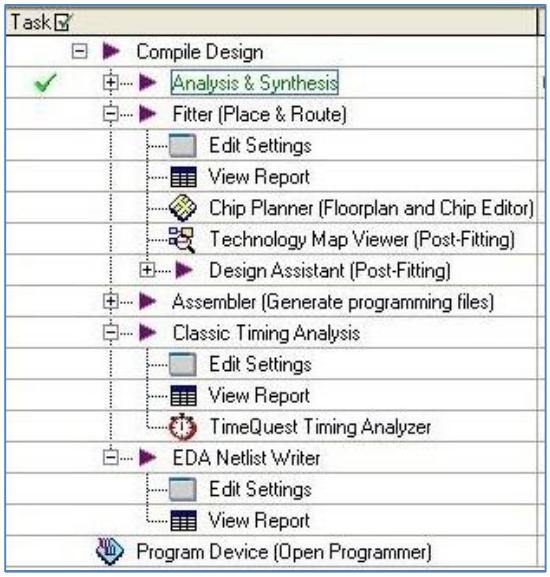
Taulukko 13 : Cyclone II FPGA-piirin (piirisarja EP2C35F672C6) ominaisuuksia

Alteran FPGA-piireissä nopeuskertoimella viitataan aikaan, joka kuluu makrosolun läpi kulkemiseen. Nopeuskerroin 6 tarkoittaa, että makrosolun läpikulkeminen kestää kuusi nanosekuntia. Tulo- ja lähtöpinnit taas mahdollistivat datan syöttämisen ja lukemisen fyysisesti FPGA-piiriltä. Osa pinneistä tosin on varattu FPGA-piirin ledeille, painonapeille, muisteille ja muille sisäisille laitteille, joten niitä ei voi käyttää ulostulona. Oskillaattorit toimivat FPGA-piirin sisäisinä kelloina, joihin toteutus voidaan kytkeä.

.

6.1. Quartus II-ohjelmisto ja logiikkasolujen määrä

Edellisessä kappaleessa kuvattu Catapultin tuottama VHDL-kuvaus muutettiin seuraavaksi FPGA-piirivalmistajan Alteran Quartus II-ohjelmistolla FPGA-piirille sopivaksi ohjelmointitiedostoksi. Quartus II-ohjelmistossa asetettiin ensin haluttu FPGA-malli ja simulointivaihtoehdot. Seuraavaksi suoritettiin analysointi ja synteesi-vaihe:



Kuva 43 : Quartus II-ohjelmiston menut

Analysoinnin jälkeen asetettiin väylät asettaa bitti kerrallaan oikeisiin FPGA-piirin pinneihin (Kuva 44). Määrityksen pystyi onneksi tallentamaan omaan .csv-tiedostoonsa, joten pinnejä ei tarvinnut joka kerta määritellä uudelleen.

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	altera_reserved_tck	Input				3.3-V LVTTL (defa...	
2	altera_reserved_tdi	Input				3.3-V LVTTL (defa...	
3	altera_reserved_tdo	Output				3.3-V LVTTL (defa...	
4	altera_reserved_tms	Input				3.3-V LVTTL (defa...	
5	auto_stp_trigger_...	Output				3.3-V LVTTL (defa...	
6	clk	Input	PIN_N2	2	B2_N1	3.3-V LVTTL (defa...	
7	debug_trigger	Output				3.3-V LVTTL (defa...	
8	fpga_sram_addr[14]	Output	PIN_W10	8	B8_N1	3.3-V LVTTL (defa...	
9	fpga_sram_addr[13]	Output	PIN_W8	8	B8_N1	3.3-V LVTTL (defa...	
10	fpga_sram_addr[12]	Output	PIN_AC7	8	B8_N1	3.3-V LVTTL (defa...	
11	fpga_sram_addr[11]	Output	PIN_V9	8	B8_N1	3.3-V LVTTL (defa...	
12	fpga_sram_addr[10]	Output	PIN_V10	8	B8_N1	3.3-V LVTTL (defa...	

Kuva 44 : FGPA-pinnien asetus

Synteesin yhteydessä saatiin myös karkea arvio siitä kuinka monta logiikkasolua toteutus tarvitsisi FPGA-piirissä (Kuva 45). Tämä luku muuttui vielä jonkin verran lopullisessa tuloksessa, mutta synteesivaiheen arviosta sai selville ainakin logiikkasolujen määrän suuruusluokan.

Flow Status	Successful - Wed May 11 15:03:00 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Full Version
Revision Name	toplevel
Top-level Entity Name	toplevel
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	N/A
Total logic elements	9,075
Total combinational functions	7,005
Dedicated logic registers	4,636
Total registers	4636
Total pins	41
Total virtual pins	0
Total memory bits	178,688
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Kuva 45: FPGA-logiikkasolujen määrä

Heti aluksi selvisi, että aiottuja X.264 –videokoodekin ominaisuuksia ei pystynyt toteuttamaan, sillä aiotulla resoluutiolla 176x144 (QCIF) ja vain täyspikselietsintätilaa käyttäen logiikkasoluja olisi vaadittu 80840. Pienentämällä resoluutiota 128x96 pikseliin (SQCIF) tarvittiin edelleen 75325 logiikkasolua ja resoluutiolla 64x48 pikselillä edelleen 67707 logiikkasolua. Oli siis selvää, että Catapult loi ohjelmakoodista täysin FPGA-piirille optimoimattoman ratkaisun. Oli karsittava X.264-videokoodekin ominaisuuksia huomattavasti, jotta laite ylipäänsä mahtuisi FPGA-piiriin. Seuraavaksi kokeilin äärimmäisen yksinkertaista ratkaisua, joka laski äärimmäisen pienellä resoluutiolla 64x48 jokaiselle 16x16 makrolohkolle vain yhden liikevektorin ja jätti residuaalikuvan laskemisen sikseen.

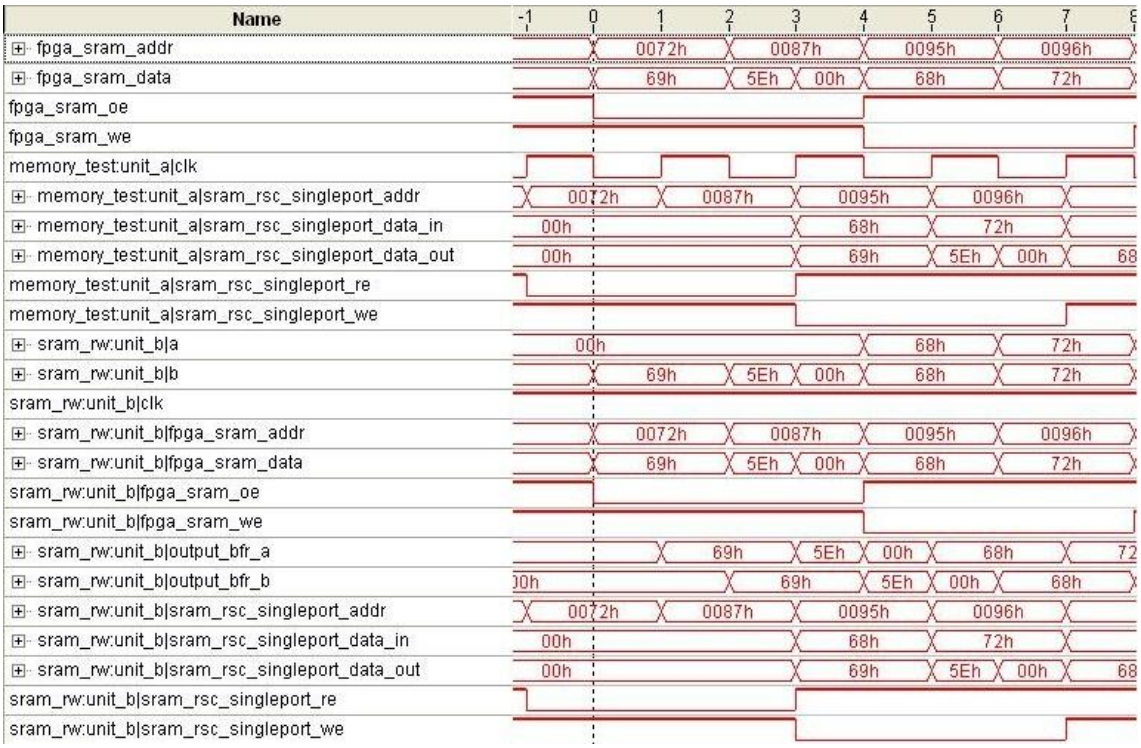
Catapult C-ohjelmisto ei kuitenkaan onnistunut tuottamaan toimivaa VHDL-koodia tästä X.264-videokoodekista. Catapultin generoima VHDL-koodi FPGA-piirin SRAM-muistin käsittelyyn sisälsi väärät väylämääritykset ja SRAM-muistin luku- ja kirjoitusoperaation ajoitukset olivat virheellisiä. Siksi käytössä olleen opiskelijaversion teknologiakirjastoilla toimivan videokoodekin tuottaminen oli mahdotonta.

6.2. Väylämääritykset ja VHDL-välittäjämoduuli

Catapultin VHDL-määrittelyssä SRAM-muistin käsittelyyn luotiin sekä data in- että data out-väylä, eikä millään ohjelmiston optiolla asiaa saanut korjattua. Jouduin siis laatimaan itse moduulin joka toimisi puskurina Catapultin VHDL-koodin ja FPGA:n SRAM-muistin välillä (Liite 11).

Välittäjämoduuli välittää Catapultin VHDL-moduulin Read Enable (Output Enable, OE) ja Write Enable (WE) signaalit sekä sovittaa data_in- ja data_out – väylät yhteiselle data-väylälle. Sovitus tapahtuu kolmitilapuskurin avulla: lukuoperaation jälkeen data-väylä siirretään korkeaimpedanssiseen tilaan, jotta kirjoitusoperaatio toimisi moitteettomasti ja väylä toimisi varmasti halutunsuuntaisesti.

Kolmitilapuskurin käytön vuoksi piti myös tahdistaa Catapultin VHDL-moduuli käyttämään puolta pienempää kellotaajuutta kuin välittäjämoduuli, jotta kirjoitus- ja lukuoperaatiota tapahtuu Catapultin VHDL-moduulin määräämillä kellojaksoilla. Ohessa on esimerkki SRAM-muistin ajoituksista:



Kuva 46: SRAM-muistioperaation ajoitus

Toteutus koostuu kahdesta moduulista: unit_a (Catapultin luoma VHDL-moduuli) ja unit_b (välittäjämoduuli SRAM-muistille). Neljä ylintä väylää fpga_sram_{addr,data,oe,we} ovat toteutuksen yhteisiä väyliä, sen jälkeen on kuusi Catapultin luomaa väylää ja lopuksi välittäjämoduulin väyliä.

Kuvassa on kaksi lukuoperaatiota osoitteista 72 ja 87 sekä kaksi kirjoitusoperaatiota osoitteisiin 95 ja 96. Voidaan havaita, että Catapultin VHDL-koodissa lukuoperaatio osoitteesta 72 käynnistyy aikaindeksillä -1 ja valmis data (69) on valmiina data_out väylällä kahden kellojakson päästä kuten Catapultin VHDL-koodi vaatii. Catapultin lukuoperaation ajoitus piti valitettavasti selvittää yrityksen ja erehdyksen kautta. Kirjoitusoperaatiot olivat onneksi helpompia: sekä osoite että data voitiin lukea puskuriin, josta ne seuraavalla kellojaksolla syötettiin eteenpäin.

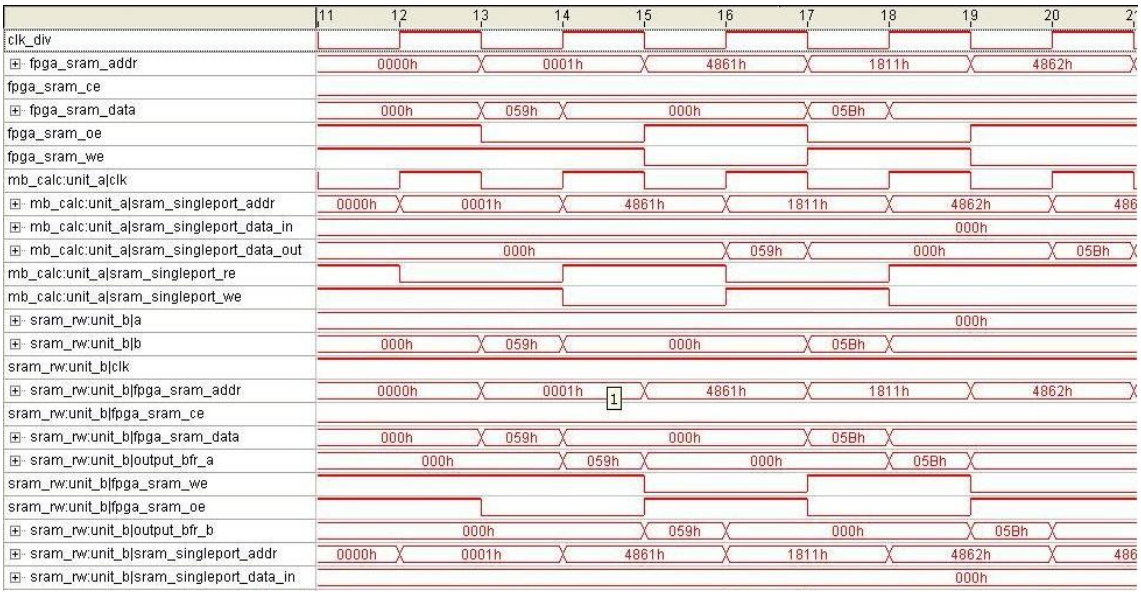
6.3. SRAM-muistin ajoitusongelmat

Välittäjämoduulin avulla oli siis mahdollista saada yksittäisiä kirjoitus- ja lukuoperaatiota toimimaan, mutta ne eivät kuitenkaan taanneet kumulatiivisten operaatioiden onnistumista. Esimerkiksi laskettaessa kahden eri makrolohkon välistä SAD-arvoa voitiin todeta, ettei kumulatiivisesti päivittyvän välimuuttujan ”sum” arvo päivittynyt oikein.

Testattava koodi oli siis seuraavanlainen:

```
SAD_FPEL_LOOP:for( y = cur_y_min; y < cur_y_max; y++ )
   {
     pix2 = pix2_base_y;
     for( x = cur_x_min; x < cur_x_max; x++ )
       {
         debug_vector[0] = pix1[x];
         debug_vector[1] = *pix2;
         debug_vector[2] = sum;
         if (pix1[x] > *pix2)
           sum = sum + pix1[x] - *pix2;
         else
           sum = sum + *pix2 - pix1[x];
         // Debugging with Quartus
         debug_vector[3] = sum;
…
```

Makrolohkoon 1 osoittaa siis osoitin pix1 ja makrolohkoon 2 osoitin pix2. Välillä yritetään debug_vector – muistirakenteen avulla selvittää mitä VHDL-koodi on saanut aikaiseksi. Quartus II – ohjelman Signaltap-simuloinnin tulokset olivat seuraavanlaisia:



Kuva 47: Kumulatiivisten muistioperaatioiden epäonnistuminen

Kuvassa 49 toiseksi ylin väylä fpga_sram_addr määrittelee siis luettavan tai kirjoitettavan osoitteen SRAM-muistissa ja fpga_sram_data sinne kirjoitettavan tai sieltä luetun datan arvon. Huomataan, että ensimmäisestä makrolohkosta (osoite 0001) ja toisesta makrolohkosta (osoite 1811) voidaan kyllä lukea dataa (arvot 059 ja 05B), mutta ne eivät ole tarpeeksi nopeasti käytössä seuraavaa operaatiota varten. Nimittäin yritettäessä kirjoittaa käskyllä debug_vector[0] = pix1[x] osoitteeseen 4861 dataa ei ohjelma olekaan vielä ehtinyt saada valmiiksi osoittimen pix1[x] välittämää dataa osoitteessa 0001. Siten kumulatiivisten välimuuttujien, kuten ”sum”, käyttö epäonnistuu.

Kaiken kaikkiaan voidaan todeta, että käytettävissä olevilla teknologiakirjastoilla Catapult ei yksinkertaisesti sovellu nopeaan tuotekehitykseen. Sen toimintaa joutuu paikkaamaan niin paljon itse kirjoitetuilla VHDL-moduuleilla, että tulisi nopeammaksi kirjoittaa itse toimiva VHDL-koodi.

7. **Johtopäätökset**

Diplomityössä on kuvattu laajasti videokoodauksen perusteita. Modernin videokoodauksen tehtävänä on pakata suuri määrä dataa mahdollisimman pieneen tilaan. Tässä tarvitaan monimutkaisia algoritmejä, joilla pystytään arvioimaan mikä osuus videodatasta on oleellista ja mitä voidaan hylätä. Tarvitaan myös algoritmejä, joilla voidaan pakata oleellinen data tiedonsiirtoa varten mahdollisimman pieneen tilaan.

Erityisen tarkasteluun otettiin X.264-videokoodekki. Siinä käytetään 16x16 pikselin makrolohkoja, joilla kullekin etsitään oma liikevektorinsa. Isot makrolohkot voidaan pilkkoa myös pienempiin osiinsa ja etsiä kullekin omat liikevektorinsa. Lisäksi videokoodekissa voidaan käyttää myös alipikselietsintätiloja, jotka vaativat runsaasti enemmän laskentatehoa mutta myös parantavat huomattavasti bittinopeuden ja laadun suhdetta.

Näiden kaikkien ominaisuuksien toteuttaminen perinteisellä VHDL-ohjelmoinnilla olisi ollut hyvin monimutkaista ja työlästä. Tässä diplomityössä haluttiin siksi testata pystyttäisiinkö Mentor Graphicsin Catapult C-ohjelmalla tätä suunnitteluaikaa lyhentämään ja rakentamaan algoritmit nopeasti Catapult C-ohjelmointikieltä käyttäen.

Valitettavasti osoittautui, ettei laboratorion käytettävissä olevilla teknologiakirjastoilla tähän pystytty. Oli mahdotonta saada algoritmien toimintaa mallinnettua FPGA-piirillä, koska suunnitteluohjelma ei muunnosta VHDL-koodiin tehdessään ottanut tarpeeksi hyvin huomioon FPGA-piirien parametrejä.

Nopean suunnittelun tarve on kuitenkin väistämätön. Tulevaisuudessa myös piiritekniikan laboratoriossa tutkittaneen muita ohjelmistoja, joilla tämä prosessi onnistuu.

Lähdeluettelo

[1] Richardson, I., H.264 and MPEG-4 video compression : video coding for next generation multimedia, Wiley, 2003

[2] Winkler, S., Mohandas, P., The Evolution of Video Quality Measurement: From PSNR to Hybrid Metrics," Broadcasting, IEEE Transactions on , vol.54, no.3, s.660-668,
DOI: 10.1109/TBC.2008.2000733
Saatavissa:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4550731&isnumber=4604792

[3] Amer, A., Dubois, E.; , Fast and reliable structure-oriented video noise estimation, Circuits and Systems for Video Technology, IEEE Transactions on , vol.15, nro.1, s. 113-118,
DOI:10.1109/TCSVT.2004.837017
Saatavissa:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1377366&isnumber=30064

[4] Koskinen, L., Analog Parallel Processor Solutions for Video Encoding, Otamedia, 2005

[5] Zhu Ce, Xiong Bing, Transform-Exempted Calculation of Sum of Absolute Hadamard Transformed Differences, ircuits and Systems for Video Technology, IEEE Transactions on , vol.19, nro.8, s.1183-1188,
DOI: 10.1109/TCSVT.2009.2020264
Saatavissa:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4811987&isnumber=5208486

[6] ITU-R. H.264: Advanced video coding for generic audiovisual services, Vitattu 1.5.2011. Saatavissa : http://www.itu.int/rec/T-REC-H.264.

[7] ITU-R: H.264/AVC Reference Software Encoder, Viitattu 1.5.2011.
Saatavissa :http://iphome.hhi.de/suehring/tml/doc/lenc/html/index.html

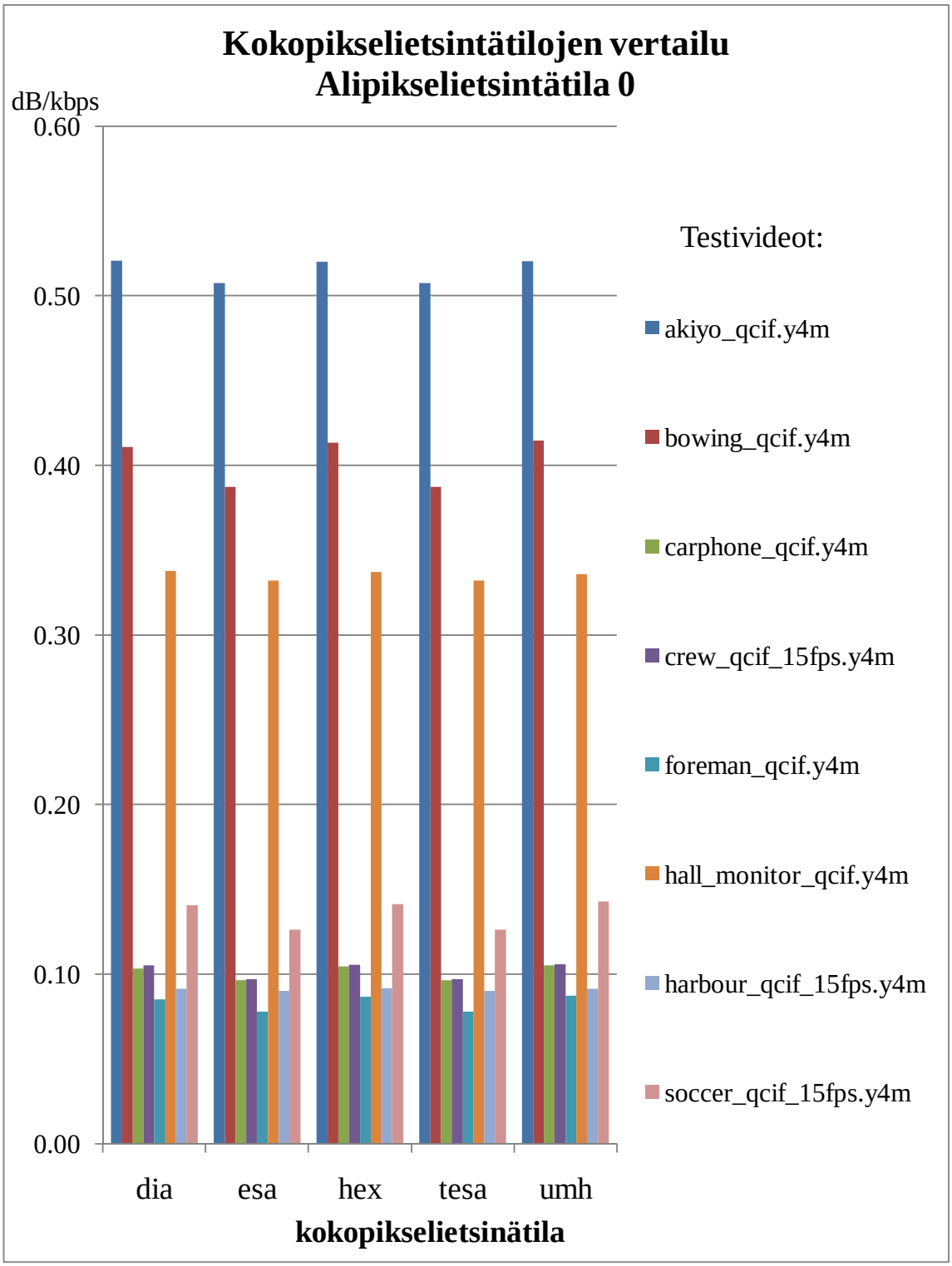
[8] X.264 Video Encoder, Viitattu 1.5.2011.
Saatavissa : http://www.videolan.org/developers/x264.htm

[9] Xiph.org, Viitattu 1.5.2011.
Saatavissa :http://media.xiph.org/video/derf/

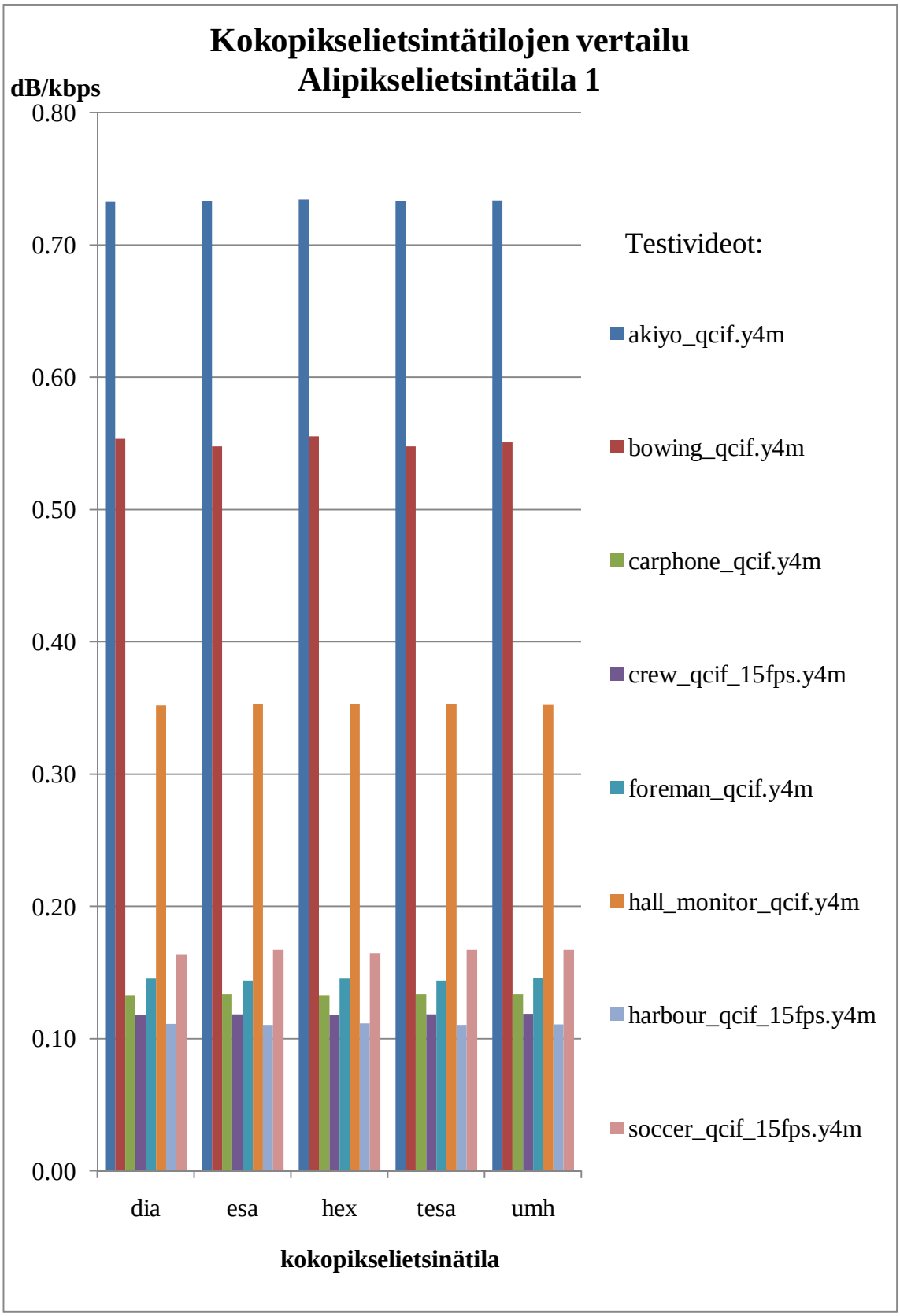
[10] Miller Lawrence H., Quilici Alexander E., The Joy of C, John Wiley & Sons, 1997

[11] Clubb S., Catapult Basic Training, Mentor Graphics, 2009

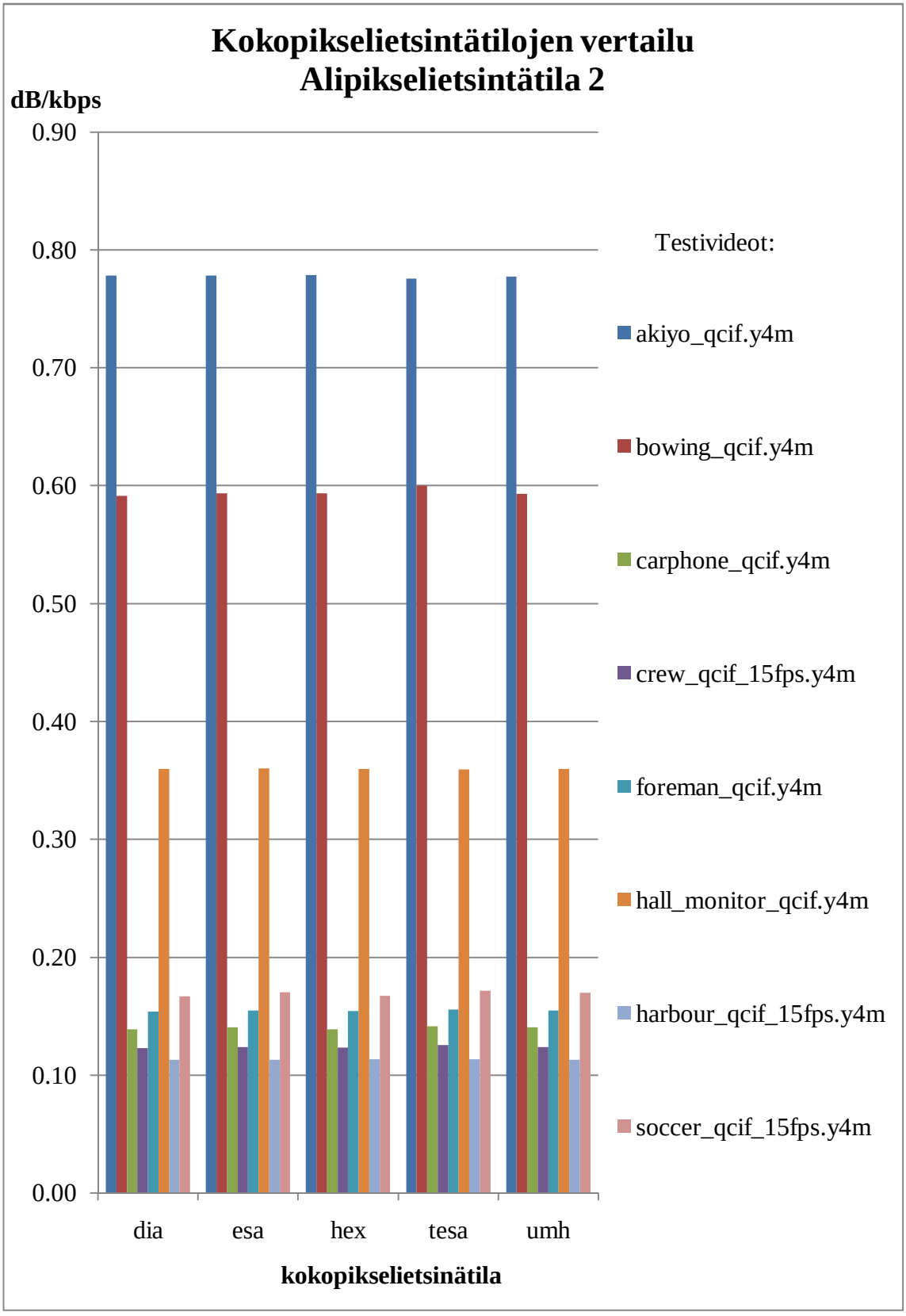
LIITTEET



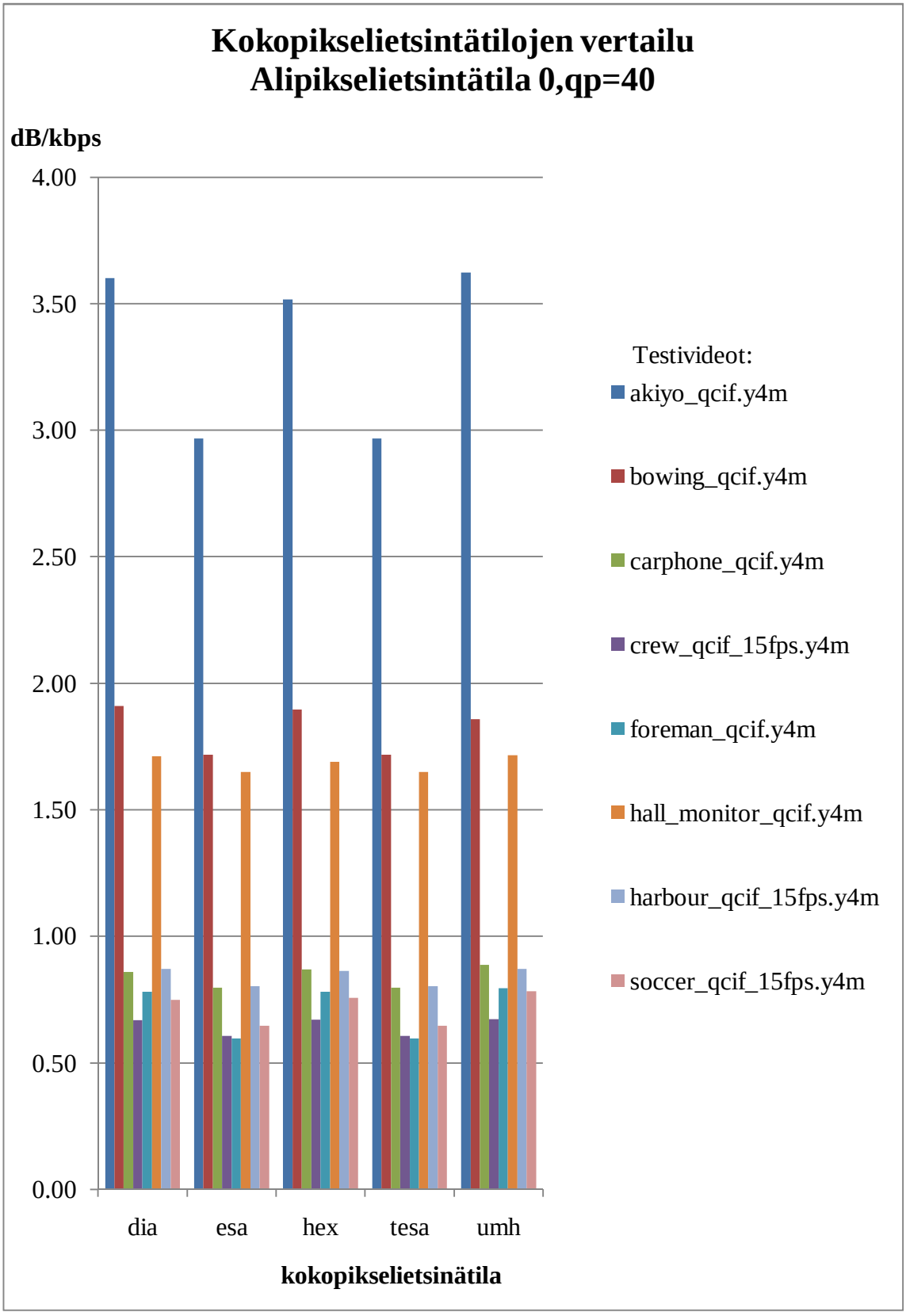
Liite 1 : Kokopikselietsintätilojen vertailu, alipikselietsintätila 0



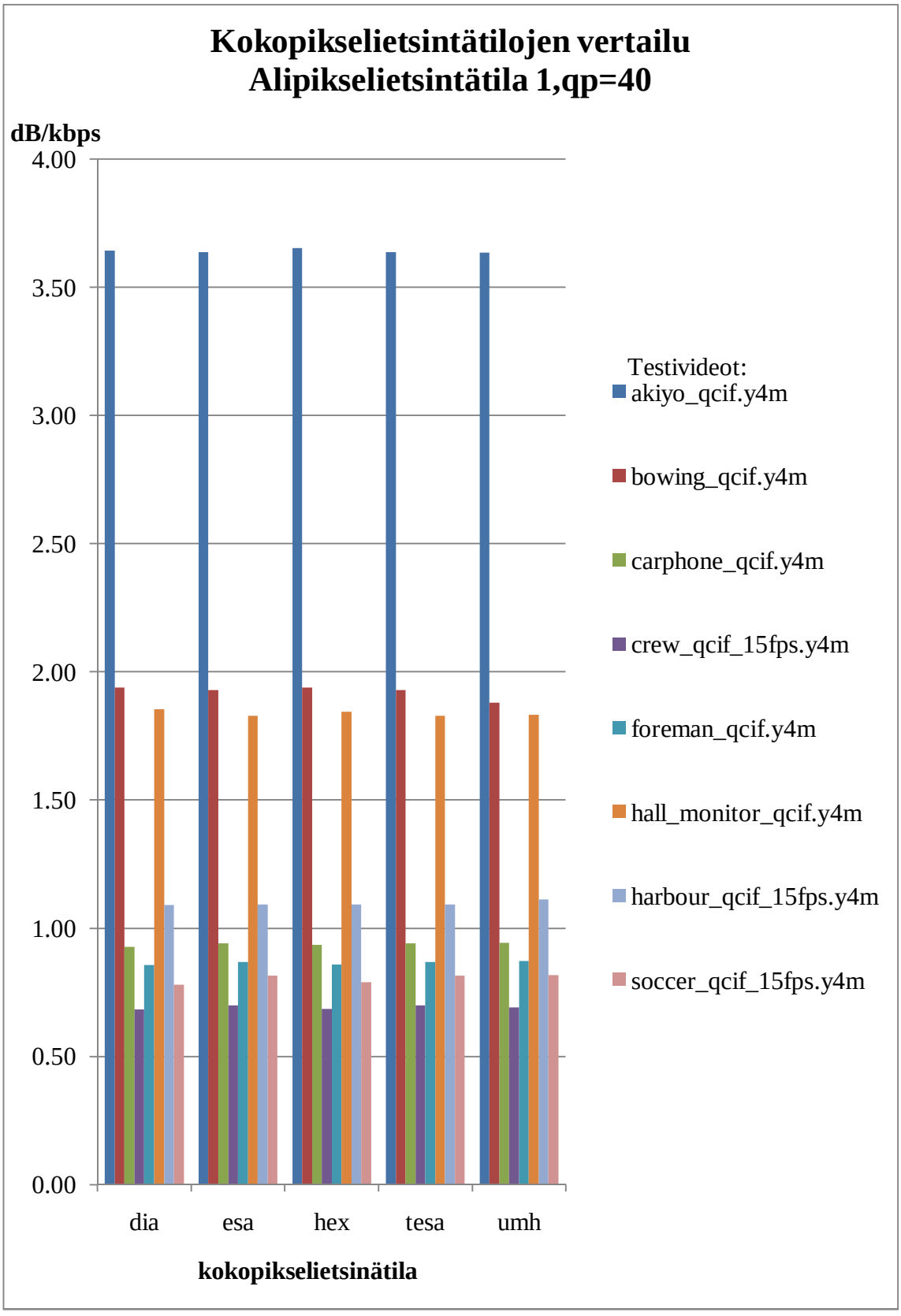
Liite 2 : Kokopikselietsinätilojen vertailu, alipikselietsinätila 1



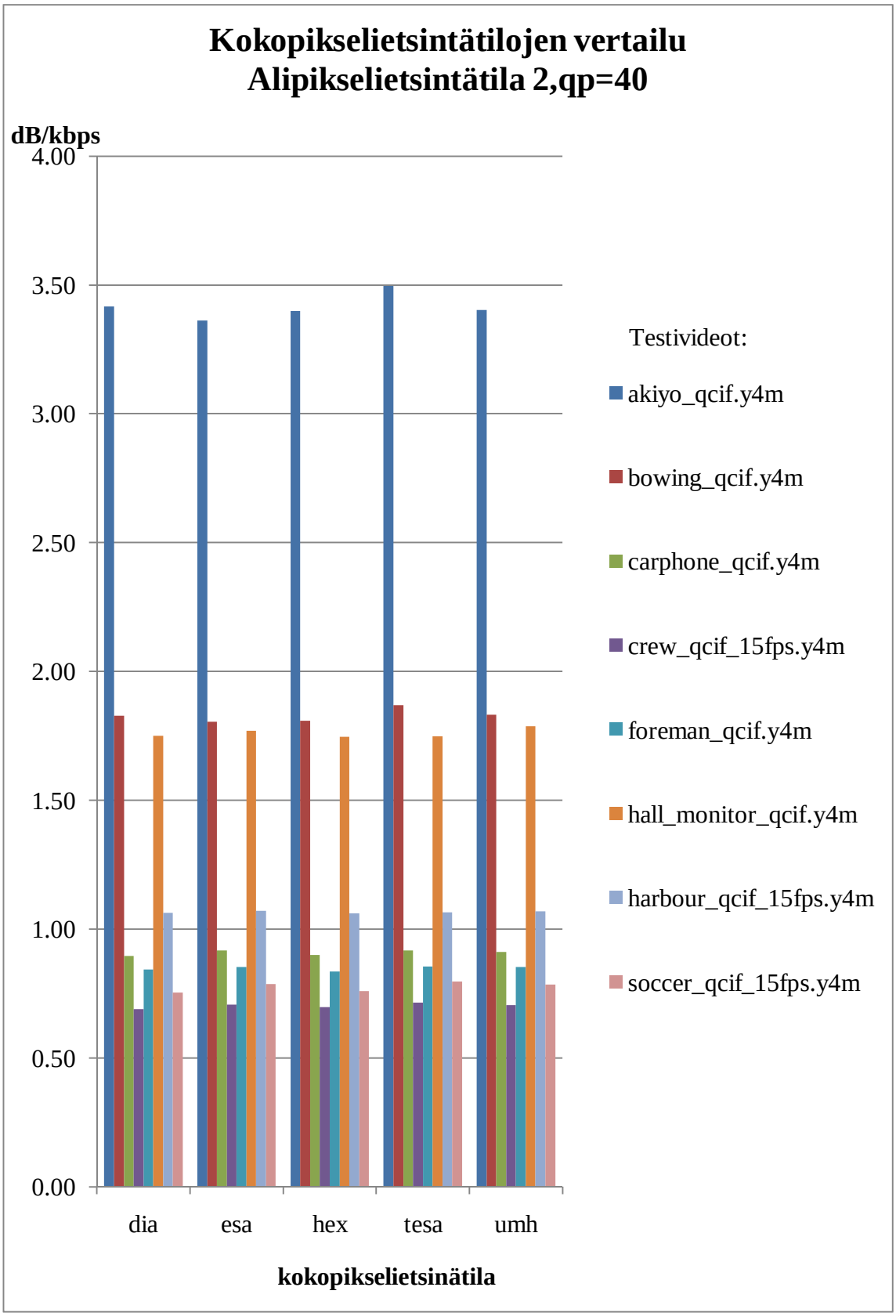
Liite 3 : Kokopikselietsinätilojen vertailu, alipikselietsinätila 2



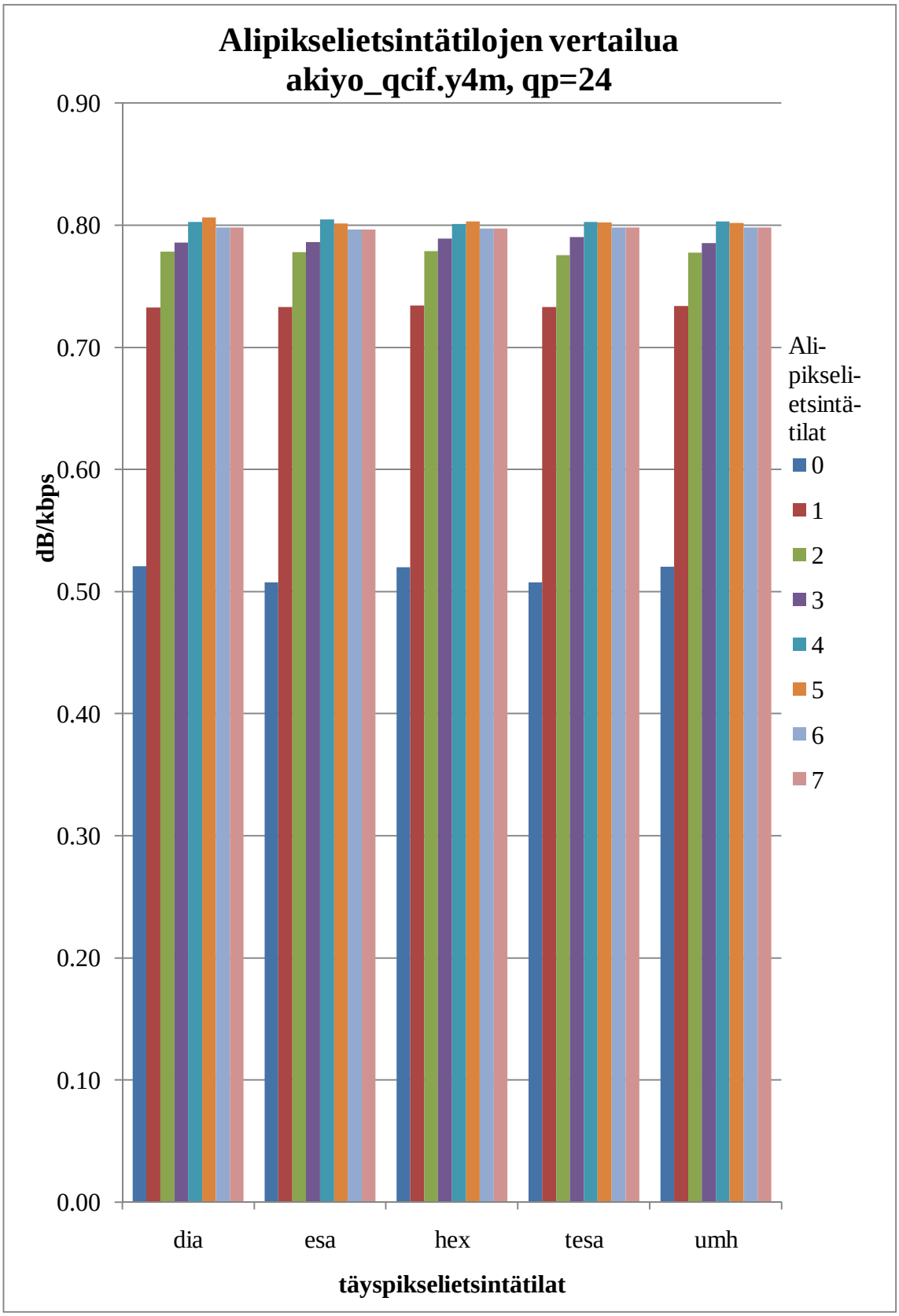
Liite 4: Kokopikselietsinätilojen vertailu, alipikselietsinätila 0, qp=40



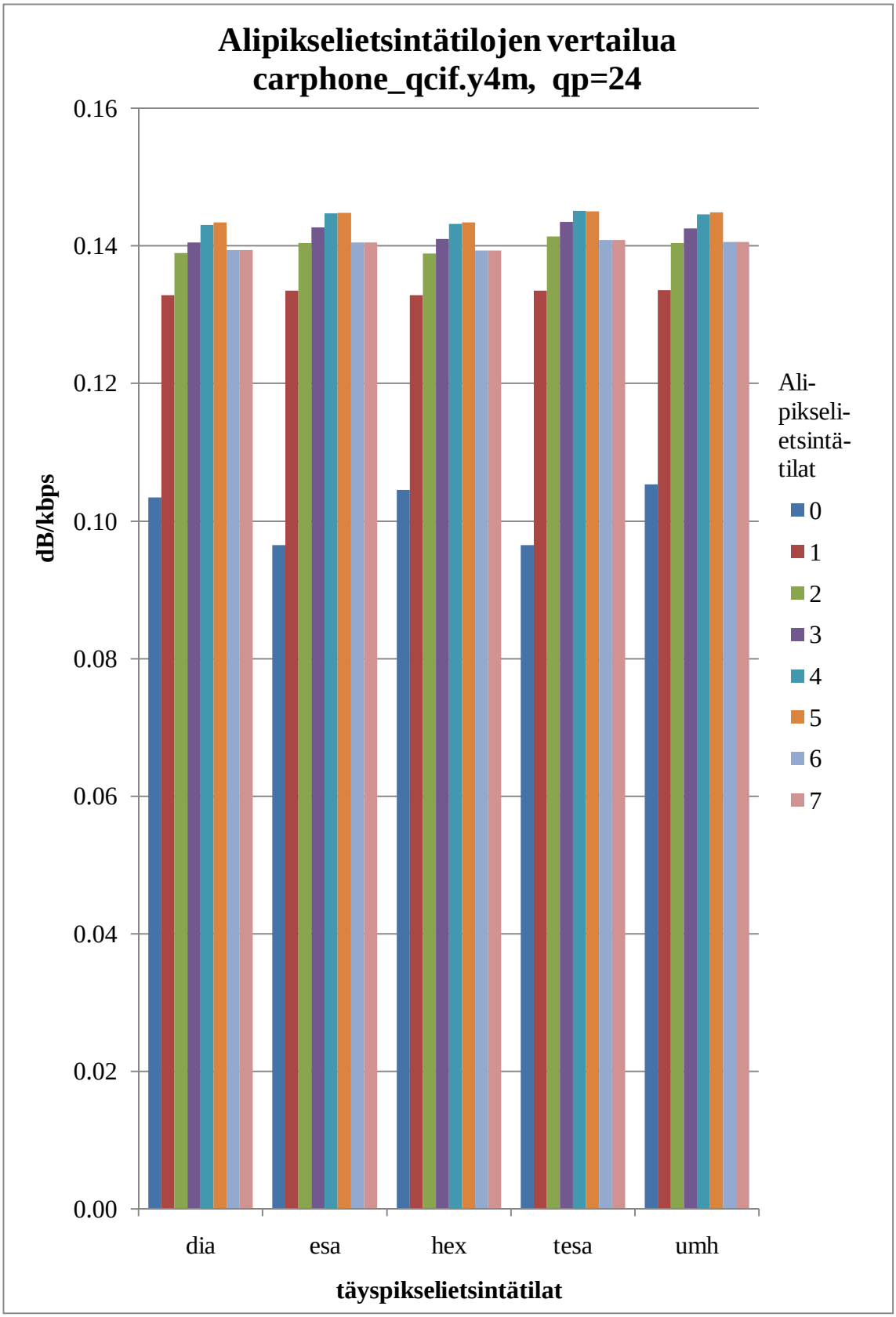
Liite 5: Kokopikselietsinätilojen vertailu, alipikselietsinätila 1, qp=40



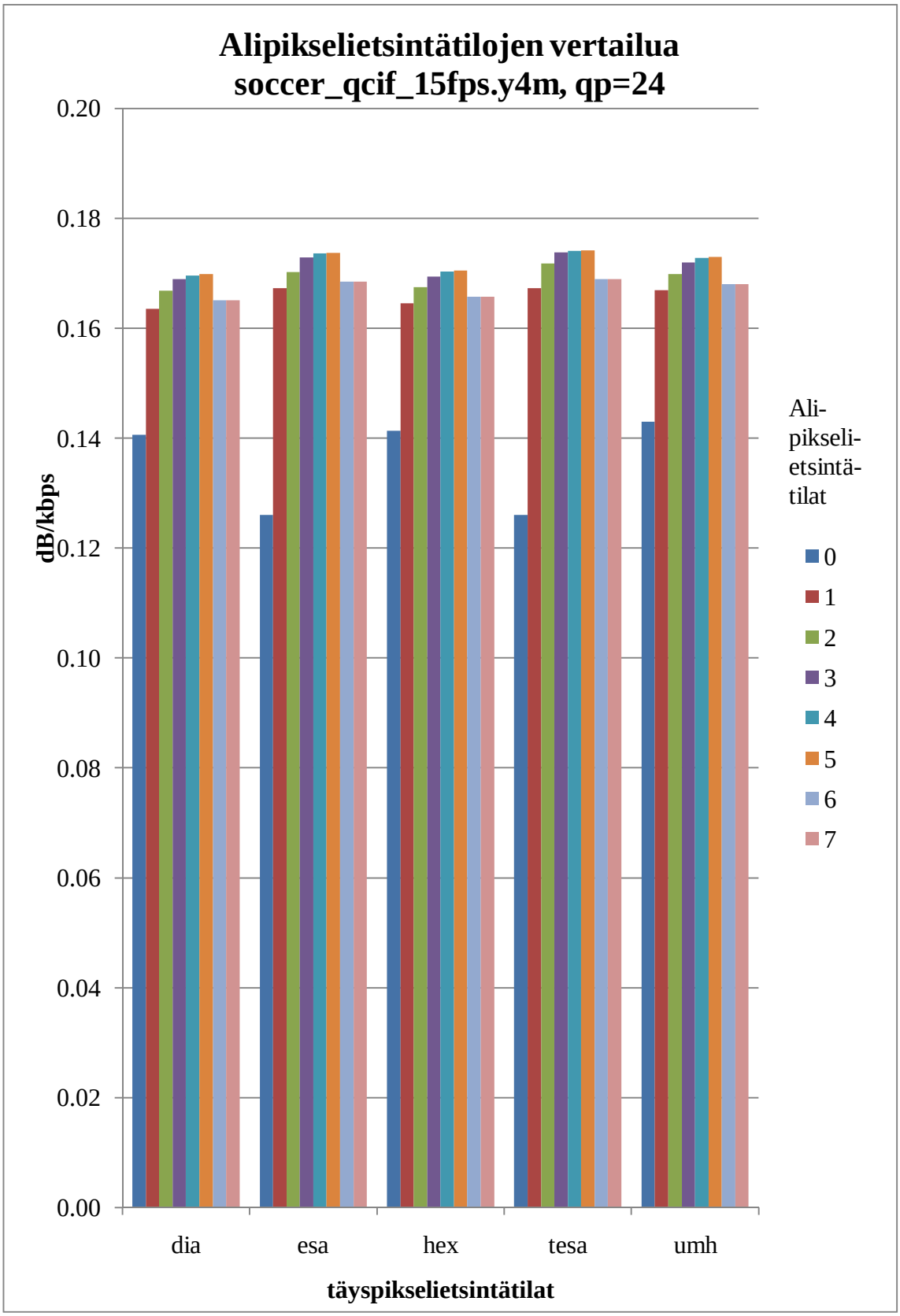
Liite 6: Kokopikselietsinätilojen vertailu, alipikselietsinätila 2, qp=40



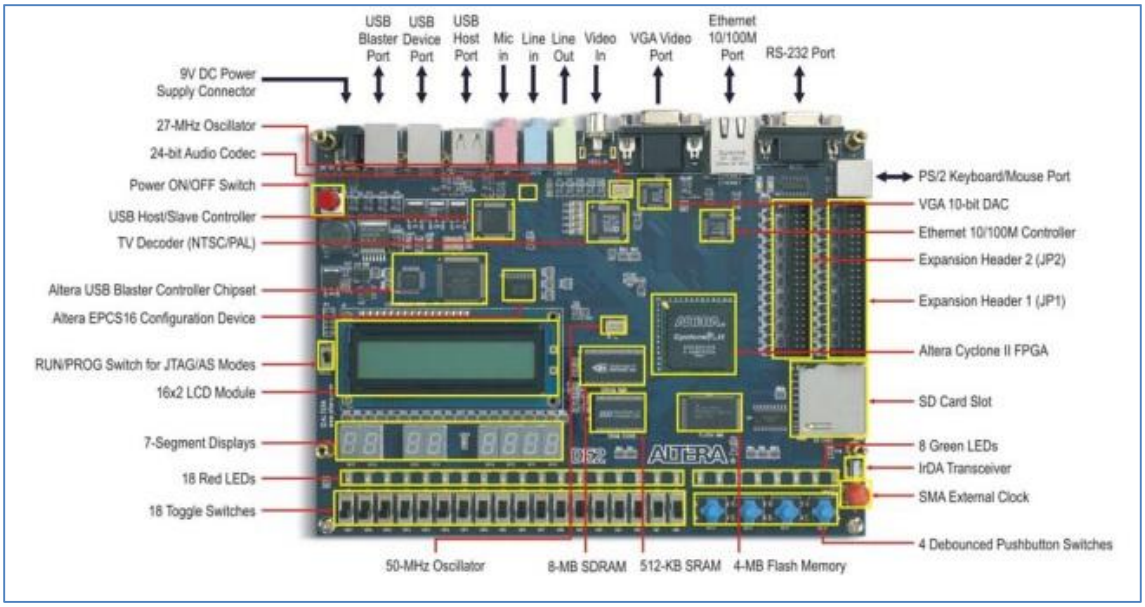
Liite 7: Alipikselietsintätilojen vertailua, qp=24, akiyo_qcif.y4m



Liite 8: Alipikselietsintätilojen vertailua, qp=24, carphone_qcif.y4m



Liite 9: Alipikselietsintätilojen vertailua, qp=24, soccer_qcif_15fps.y4m



Liite 10 : Simuloinnissa käytetty Alteran FPGA-levy

Liite 11 : : FPGA-piirin SRAM-muistin käsittelyyn luotu VHDL-moduuli

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY sram_rw IS
  GENERIC
      (
      k : integer;  -- address bus width
      w : integer  -- data bus width
  );
  PORT(
    clk : IN STD_LOGIC;
    -- Resources created by Catapult program
    -- data_in to the FPGA
    sram_singleport_data_in : IN STD_LOGIC_VECTOR (w-1 DOWNTO
0);
    -- data_out from the FPGA
    sram_singleport_data_out: OUT STD_LOGIC_VECTOR (w-1 DOWNTO
0);

    sram_singleport_re : IN STD_LOGIC;  -- Output Enable
    sram_singleport_we : IN STD_LOGIC;  -- Output Enable
    sram_singleport_addr : IN STD_LOGIC_VECTOR (k-1 DOWNTO 0);

    -- Signals going to FPGA SRAM unit
    fpga_sram_data : INOUT STD_LOGIC_VECTOR (w-1 DOWNTO 0);
    fpga_sram_addr : OUT STD_LOGIC_VECTOR (k-1 DOWNTO 0);
    fpga_sram_oe : OUT STD_LOGIC;  -- Output Enable
    fpga_sram_we : OUT STD_LOGIC;  -- Write enable
    fpga_sram_ce : OUT STD_LOGIC;  -- Chip enable
    fpga_sram_lb : OUT STD_LOGIC;  -- Lower byte
    fpga_sram_ub : OUT STD_LOGIC  -- Upper byte

    );
```

```
end sram_rw;

architecture v1 of sram_rw is
SIGNAL  a  : STD_LOGIC_VECTOR (w-1 DOWNTO 0);  -- DFF that
stores value from input
SIGNAL  b  : STD_LOGIC_VECTOR (w-1 DOWNTO 0);  -- DFF that
stores feedback value
SIGNAL output_bfr_a : STD_LOGIC_VECTOR (w-1 DOWNTO 0);
SIGNAL output_bfr_b : STD_LOGIC_VECTOR (w-1 DOWNTO 0);
SIGNAL re_delayed : STD_LOGIC;

begin  -- v1

   -- general signals
   fpga_sram_ce <= '0';    -- chip enable
   fpga_sram_lb <= '0';    -- low byte
   fpga_sram_ub <= '0';    -- high byte


   -- Creates the flipflops
   FLIPFLOPS:PROCESS(clk)
   BEGIN
   IF clk = '1' AND clk'EVENT THEN
      a <= sram_singleport_data_in;

      output_bfr_a <= b;
      output_bfr_b <= output_bfr_a;
      sram_singleport_data_out <= output_bfr_b;
      fpga_sram_addr <= sram_singleport_addr;
      fpga_sram_we <= sram_singleport_we;
      fpga_sram_oe <= sram_singleport_re;
      END IF;
   END PROCESS FLIPFLOPS;

   RW:PROCESS (sram_singleport_re, fpga_sram_data,a)
   BEGIN
```

```
        -- Reading process
        IF( sram_singleport_re = '0') THEN
            fpga_sram_data <= (others => 'Z');
            b <= fpga_sram_data;
        -- Writing process
        ELSE
            fpga_sram_data <= a;
            b <= fpga_sram_data;
        END IF;
    END PROCESS RW;


end v1;
```