Niko Syrjälä

Videon käsittely sekä kuljetus AXI –väylällä FPGA-SoC –järjestelmäpiirillä

Tekniikan ja innovaatiojohtamisen yksikkö
Informaatiotekniikan koulutusohjelma
Diplomityö
Automaatiotekniikka

Vaasa 2021

University of Vaasa
School of technology and innovation
Author: Niko Syrjälä
The topic of the thesis: Handling and transportation of video via AXI in a FPGA-SoC
Thesis supervisor: Professor Jarmo Alander
Instructor: D.Sc.Tech Petri Välisuo
Major: Automation Technology
Year of completing the thesis: 2021 **Pages:** 58

ABSTRACT

The thesis investigates the pipeline design of image processing with an FPGA-SoC (Field Programmable Gate Array System on a Chip) device. The aim is to investigate whether it makes sense to first convert image data into RGB (Red-Green-Blue) matrices, for which it is possible to perform signal processing with filters on an FPGA-SoC. What kind of pipeline structure could be used for solving matrix equations, and would this be a suitable exercise for students to study pipelining techniques? The aim is to answer these questions by implementing a pipeline raw design, which includes the necessary blocks and explains why they are needed and providing a flow chart of the principle of the algorithm.

The introduction presents how the market has developed significantly over FPGA-SoC life cycle. Substitutes for ASICs, (Application Specific Integrated Circuit), which were originally designed for, have grown into independent entities which lower NRE –costs (Non-recurring engineering) allow FPGA-SoC's to take over new industries. By examining their history, we get to see that they are futuristic solution, and things that may seem impossible to achieve now may be universal and well-known solutions in the future. This serves as a motivation to study and become more familiar with the design of FPGA-SoC applications.

In pipeline design, we examine what kind of blocks a possible pipeline may include and consider what the function of these blocks is. After, which a traditional CPU alternative solution is shown, to express how long the execution of the same algorithms take in CPU and how FPGA-SoC's parallel performance increases the efficiency, because it is capable of performing multiple functions simultaneously due to its tailor-made architecture.

Finally, there is a general discussion about the future of the industry and what pros and cons were found in the solutions and how the hardware manufacturer's own programming platform for FPGA-SoC's can make the designer's life easier when hardware-based language skills are not exemplary.

KEYWORDS: AXI, FPGA-SoC, IP, image data, Zybo Z7, pipeline

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Niko Syrjälä
Tutkielman nimi: Videon käsittely sekä kuljetus AXI –väylällä FPGA-SoC –järjestelmäpiirillä
Ohjaajan nimi: Professori Jarmo Alander
Valvojan nimi: TkT Petri Välisuo
Tutkinto: Diplomi-insinööri
Oppiaine: Automaatiotekniikka
Tutkielman valmistumisvuosi: 2021 **Sivumäärä:** 58

TIIVISTELMÄ

Diplomityössä tutkitaan kuvadatan liukuhihnan suunnittelua FPGA-SoC –laitteella. Tavoitteena on tutkia, onko järkevää kuvadata muokata ensin perinteisiksi RGB –matriiseiksi, joille voidaan mahdollisesti suorittaa suotimilla signaalinkäsittelyä FPGA-SoC:lla. Millainen liukuhihnarakenne mahdollisesti tällä halutulla matriisiratkaisulla olisi sekä sopisiko tämä esimerkiksi opiskelijoille tehtäväksi harjoitukseksi tutkia liukuhihnoitustekniikoita. Näihin kysymyksiin pyritään vastaamaan toteuttamalla liukuhihna, jossa otetaan mukaan tarvittavia lohkoja sekä kerrotaan, miksi niitä tarvitaan sekä esitetään vuokaavio algoritmin periaatteesta.

Johdannossa esitellään kuinka FPGA-SoC:n markkinat ovat kehittyneet merkittävästi tuotteen elinkaaren aikana. Alunperin alusta loppuun suunniteltujen ASIC –piirien korvaajista on kasvanut itsenäisiä kokonaisuuksia, joiden matalammat alustavat kustannukset antavat FPGA-SoC:lle mahdollisuuksia vallata uusia toimialoja. Perehtymällä siihen miten FPGA-SoC:n historia on lähtenyt käyntiin, saadaan motivaatio siitä, että FPGA on tulevaisuuden ratkaisu, joka tulee muuntautumaan tulevaisuuden haasteisiin ja asiat mitkä voivat vaikuttaa nyt mahdottomilta saavuttaa FPGA-SoC:n avulla voivat olla yleismaallisia ja tunnettuja ratkaisuja tulevaisuudessa. Tämä toimii motivaationa tutkia ja perehtyä syvemmin FPGA-SoC –sovellusten suunnitteluun.

Liukuhihnan suunnittelussa tutkitaan millaisilla lohkoilla voisi prosessin liukuhihnoituksen toteuttaa sekä katsotaan mikä näiden lohkojen toiminta tässä liukuhihnassa on. Liukuhihnan läpikäynnin jälkeen esitetään perinteinen suorittimella suoritettava vaihtoehtoratkaisu, jossa näytetään kuinka pitkäksi perinteisellä tavalla suoritettuna alkavat algoritmit venyä ja kuinka FPGA-SoC:n rinnakkaisuuden ansiosta saadaan tehokkuutta kasvatettua perinteisen sekventiaalisen suorittamisen sijaan, kun FPGA-SoC pystyy lohkorakenteensa ansiosta suorittamaan useita toimintoja samanaikaisesti.

Lopuksi pohditaan yleisesti tulevaisuutta alalle sekä mitä hyviä ja huonoja puolia ratkaisuista löydettiin ja kuinka laitevalmistajan omalla ohjelmointialustalla, jolla FPGA-SoC voidaan hallita, saadaan helpotettua suunnittelijan elämää, kun laitteistopohjaisten kielten osaaminen ei ole samalla tasolla kuin korkeamman tason ohjelmointikielten osaaminen.

AVAINSANAT: AXI, FPGA-SoC, IP, kuvadata, ZYBO Z7, liukuhihna

ESIPUHE

Omistettu äidille ja isälle, jotka ovat mahdollistaneet unelmieni seuraamisen. Aina siitä lähtien, kun pieni lippalakkipäinen poika lähti kävelemään koulutietä ovat kouluvarusteet olleet ajantasaisia kynistä lukiokirjoihin, tuettu ainevalintoja sekä annettu neuvoja mihin panostaa voimavaroja sekä istutettu asenne, että mikään ei pysäytä, jos on tahtoa saavuttaa haluamansa. Sekä kuinka he ovat tukeneet läpi yliopiston aina tarvittaessa, vaikka ruokkimalla päivittäin omien rahojen ollessa vappurientojen ansiosta huvenneet loppuun. Kuin myös ovat tukeneet maailmani sekä maailmankatsomukseni avartamista mahdollistamalla ulkomaille sijoittuneet luokkaretkeni sekä ikimuistoisen Cern –tutkimuskeskus matkani abiturienttivuotenani toverieni kanssa. Sitä vaivannäköä mitä minun elämäni onnistumisen ja onnen luomiseen on käytetty, ei pysty kaikkea tässä listaamaan, mutta kaikesta siitä olen kiitollinen syvällä sydämessäni.

Vaikeuksien kautta voittoon. Kiitos.

Vaasa maaliskuu 2021

Niko Syrjälä

SISÄLLYS

ABSTRACT 2

TIIVISTELMÄ 3

ESIPUHE 4

SISÄLLYS 5

KUVALUETTELO 7

TAULUKKOLUETTELO 8

ALGORITMILUETTELO 8

1 JOHDANTO 11

1.1 FPGA-SoC:n kehityksen historiaa 12

1.2 Työn tarkoitus 15

1.3 Rajaus 15

2 FPGA-SYSTEM ON A CHIP 16

2.1 Esimerkkilaite sekä -ohjelmisto 19

2.1.1 Kuvadatan tiedostomuoto 22

3 LIUKUHIHNAN SUUNNITTELU 24

3.1 Liukuhihna 27

3.2 Kuvadatan alkukäsittely 27

3.3 VDMA 28

3.4 IP –lohkot liukuhihnoitukseen 28

3.5 ARM –suorittimen C++ –algoritmia 31

3.6 Suorittimeen nojaava ohjelma 33

3.6.1 Vaihtoehto ohjelman läpileikkaus 34

3.7 Vivado 37

3.8 Asennus FGPA:lle 38

4 TESTAUS JA TULOKSET 40

4.1 Videokuvan muutoksen tarkastelu mittareilla 43

4.2 Parannusideoita 46

5 POHDINTA SEKÄ YHTEENVETO 48

5.1 Johtopäätökset 49

LÄHDELUETTELO 51

LIITEET 55

Liite 1. Osa YLÖSAJO –ohjelmasta 55

Liite 2 . Osa FILTTERIAJO –ohjelmasta 56

KUVALUETTELO

Kuva 1. Kuvaaja, joka arvioi piirin suunnittelun keskiarvohintaa Xilinx ja Gartnerin tarjoaman datan avulla (Trimberger, 2015:328). 17

Kuva 2. 1980-luvun puolivälistä 2000-luvun ensimmäisen vuosikymmenen puoliväliin 20 vuoden aikana tapahtunut suoritintehon kasvu verrokkipisteenä käytettiin VAX-11/780 suorittimeen (Martin, 2012:1). 18

Kuva 3. Esimerkki lohkorakennearkkitehtuuri FPGA-SoC:lle (Trimberger, 2015:320). 19

Kuva 4. ZYBO Z7 –piiri kuvattuna ylhäältä päin. 21

Kuva 5. Kuvadatan käsittelyprosessin vaiheet FPGA-SoC –järjestelmäpiirillä. 25

Kuva 6. Laatikkokaavio kokonaisjärjestelmästä. 26

Kuva 7. Liukuhihnan visualisointi. 29

Kuva 8. Vuokaavio suorittimelle kirjoitetun FILTTERIAJO:n toimintaperiaatteesta. 34

Kuva 9. Tarvittavat PNG –kuvan otsikkotiedot, joita tarvitaan vaihtoehtoisen lähestymistavan toteuttamisessa. 35

Kuva 10. Laatikkokaavio ohjelman luomisen etenemisestä. 38

Kuva 11. Esikäsittelemätön RAW –videokuva. 41

Kuva 12. Käyttäjä vilkuttaa kameralle sekä nappaa kuvan näytöllä pyörivästä videokuvasta. 42

Kuva 13. Käsittelemättömän videokuvan histogrammi. 43

Kuva 14. Käsitellyn videokuvan histogrammi. 44

Kuva 15. RGB –arvojen hakutaulu muokkaamattomalle kuvadatalle. 45

Kuva 16. RGB –arvojen hakutaulu muokatulle kuvadatalle. 45

TAULUKKOLUETTELO

Taulukko 1. ZYBOn lohkojen sisältöä (Digilent 2020). 16

Taulukko 2. ZYBOn teknisiä tietoja valmistajan (Digilent 2020) mukaan. 22

Taulukko 3. Resurssien käyttöaste ZYBO Z7-10 laitteella ajon aikana. 41

ALGORITMILUETTELO

Algoritmi 1. Pseudokoodi YLÖSAJO ohjelmasta FPGA-SoC:lle 32

Algoritmi 2. Pseudokoodia ohjelmalle FILTTERIAJO. 36

SYMBOLI- JA LYHENNELUETTELO

ADC	Analog-to-digital converter
AES	Advanced Encryption Standard
AMBA	Advanced Microcontroller Bus Architecture
AP SoC	All Programmable System on a chip
API	Application programming interface
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BMP	Bitmap image file
CFA	Color filter array
CLB	Configurable logic block
CMT	Clock Management Tile
CPU	Central Processing Unit
CSI	Camera Serial Interface
DAC	Digital-to-analog converter
DSP IP	Digital Signal Processing Intellectual Property
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPIO	General Purpose I/O
GPU	Graphic Processing Unit
HDMI	High-Definition Multimedia Interface
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group –standard

LED	Light-emitting diode
LUT	Lookup table
NRE	Non-recurring engineering
PC	Personal Computer
PNG	Portable Network Graphics
RGB	Red-Green-Blue
SDK	Software Development Kit
SoC	System on a Chip
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VDMA	Video Direct Memory Access
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit

1 JOHDANTO

Vuonna 2020 Vaasan yliopiston professorin Jarmo Alanderin kanssa käytyjen keskustelun seurauksena ja innoittamana nousi esille tämä diplomityön aihe. Diplomityön tuloksena syntyvää ohjelmointikoodia pystytään soveltamaan kuvadatan käsittelyyn FPGA-SoC–sovellutuksissa (Field-Programmable Gate Array System on a Chip). Työssä on käytetty C/C++ -ohjelmointikieltä, joka valikoitui toteutuskieleksi, koska se on tekijälle tuttu, jolloin niille tyypillisten operaatioiden kuten dynaamisten muistinvarausten suorittaminen sekä manuaalisen roskien kerääminen onnistuu, mikä on työn suorituksen kannalta merkittävää, koska matriisien ja osamatriisien muodostaminen on oltava muokattavissa syötettävän kuvadatan tai osamatriiseille suoritettavien operandien vaatimille rajauksille. Matriisit ovat kaksiulotteisia taulukkoja, joihin on varastoitu kuvadatalle merkittävää tietoa, kuten pikselien väri.

Kuvadataa voidaan syöttää FPGA-SoC–piirille esimerkiksi USB–yhteyden (Universal Serial Bus) kautta tai sisääntuloväylien kautta laitteistokohtaisesti. Kuvadata voidaan muuttaa pienemmiksi segmenteiksi käsittelyä varten ja tähän on valittu esitettäväksi kaksi tapaa: toisessa kuvan purkaminen kaksiulotteiseksi RGB–matriisiksi (Red-Green-Blue) ennen syöttöä piirin käsiteltäväksi C–pohjaisella tavallisella PC –ohjelmalla (Personal Computer) sekä toisena keinona itsenäinen FPGA-SoC rakenne, jossa suoraan kameralta siirretään dataa FPGA-SoC:n käsiteltäväksi C++ –pohjaisella ohjelmalla. Tämä mahdollistaa kuvan monipuolisen käsittelyn, koska kuva on muutettu helpommin käsiteltäväksi RGB–matriisiksi. Matriisien dataa voidaan siten syöttää halutusti FPGA-SoC–piirille ja sille ohjelmoidulle ohjelmalle käsiteltäväksi minkä ansiosta matriisille voidaan suorittaa erilaisia tehtäviä rinnakkaisesti. Lisäksi kuvan hajottaminen matriiseiksi mahdollistaa erilaisten tarkkuustasojen valitsemisen, koska käsiteltävä koko on käyttäjän valittavissa sekä voidaan muodostaa ennen ja jälkeen –kuvavertailuja, koska alkuperäinen kuvamatriisi on säilytettävissä ja ulossyötettävissä. Tämän ansiosta voidaan myös tarkkailla mahdollisten suotimien toimivuutta, kun alkuperäinen ja käsitelty kuvamatriisi ovat tallessa sekä tiedostomuoto on muutettavissa.

Diplomityössä syötetään liukuhihnan läpi videokuvaa FPGA-SoC –järjestelmäpiirillä (System on a Chip, kokonainen systeemi sijoitettuna yhdelle mikropiirille) suoraa kameralta sekä käsiteltynä. Liukuhihnan rakennetta tarkastellaan ja miten kuvadata muuttuu liukuhihnassa matkalla kameralta näytölle. Kahden eri videokuvan vertailuun käytetään muun muassa histogrammi ja tutkitaan miten käytetyt suotimet vaikuttavat siihen sekä oliko se tarpeen. Liukuhihnassa olevista aritmeettisista operaatioista on vastuussa FPGA-SoC ja sen ARM –suoritinta käytetään ohjelman käynnistämiseen sekä videokuvan alustamiseen.

1.1 FPGA-SoC:n kehityksen historiaa

FPGA esiteltiin markkinoille 1980-luvun puolivälissä mukaan lukien Xilinxin toimesta kilpailemaan ASIC–laitevalmistajien (Application Specific Integrated Circuit) laitteistojen kanssa. Näistä vaatimattomista olosuhteista ja tarpeista tulisi FPGA muuntautumaan monipuoliseksi järjestelmäpiiriksi 2000- ja 2010-luvuilla, jolla olisi mahdollista yhtäaikaisesti suorittaa ohjelmiston sekä laitteen ohjelmointia, mikä takaisi FPGA:lla suoritettavien toimien rajojen sijaitsevan ihmismielen kyvykkyydessä keksiä uusia käyttötarkoituksia FPGA:lle tulevaisuudessa (Trimberber, 2015:328-330).

FPGA syntyi tarpeesta saada halvempia laitteita pienempiä laite-eriä varten, koska ASIC–järjestelmien suunnittelu muutamia laitteita varten oli kalliimpi ratkaisu kuin FPGA:n, jonka tuottaminen per laite oli kalliimpi ratkaisu pidemmällä aikavälillä kuin vastaavan ASIC –ratkaisun, mutta FPGA:n valmistamiseksi tarvittiin pienempi NRE –investointi (non-recurring engineering), termi viittaa suunnitteluun ja testaamiseen tuotesuunnittelussa kerran käytettävään kustannukseen, joten pienempien määrien markkinoilla FPGA nousi varteenotettavaksi vaihtoehdoksi. (Trimberger, 2015:319-322).

Siirryttäessä 1990-luvulle FPGA:n markkinaosuudet lähtivät nousuun nopeammin valimoiden huomatessa FPGA:n hyödyt prosessiteollisuudessaan. Markkinaosuuksien noustessa sekä FPGA–laitteiden maineen kiihtyessä syntyi tarve investointiin käytettävien rahavirtojen nousun hillitsemisestä. Laitteiden monimutkaistuessa asiakkaiden tarpeiden

muuttuessa sekä asiakkaiden vaatiessa yhä monipuolisempia ominaisuuksia olivat kehityskulut lähteneet nousuun, jonka takia suurimmat rahavirrat ohjattiin fyysiseen kehitykseen, minkä takia markkinoille saapui yritysten ulkopuolisia ohjelmistotyökalujentarjoajia, jotka uhkasivat kasvattaa FPGA:n käyttöönottokustannuksia, kun ohjelmistot eivät enää olleet laitevalmistajien hallinnassa. FPGA –yritysten keskuudessa syntyi pelkoa asiakaskadosta, joten yritykset suorittivat toimenpiteitä saadakseen pidettyä kasvavat markkinat hallinnassaan, ettei FPGA:n kehitys hiipuisi tai jopa loppuisi kustannusten hallitsemattoman kasvun ja asiakaskadon seurauksena. (Trimberger, 2015:323-326).

Vuosituhannen taitteen jälkeen FPGA:lle asetettaville vaatimuksille tapahtui mullistuksia, joiden saavuttaminen merkitsi uusia aluevaltauksia FPGA:lle, kuten uusien komponenttien kehityksen tuoma suoritettavien tehtävien määrän ja monimutkaisuuden kasvaminen. 2000–luvulla FPGA oli muuttunut pelkistä porteista ja niiden välisistä yhteyksistä koostuvasta kokonaisuudesta uudenlaiseksi monipuoliseksi SoC –järjestelmäksi, jossa prosessoreiden, muistin sekä kellosyklien hallinnan merkitys kasvoi. Toisin kuin 90-luvulla, jolloin kustannuksia pyrittiin pitämään aisoissa sekä tehokkuuden ja nopeuden kasvamista pidettiin pääprioriteetteinä. Lohkomäärien ja erilaisten komponenttien lukumäärän kasvaessa kustannukset kasvoivat sekä suoritusnopeus heikentyi, kun resursseja tuli jakaa mikropiirin lohkojen välillä. (Trimberger, 2015:325-326).

Yritysten kuten Alteran, jonka Intel ilmoitti lehdistötiedotteella hankkineensa 2015 ja sulautti itseensä (Intel, 2015 *Intel Acquisition of Altera*), ja Xilinxin etsiessä ratkaisuja optimin rakenteen saavuttamiseksi päätyivät molemmat yritykset valitsemaan ratkaisun, jossa käyttämättömien lohkojen jääminen piirille oli hyväksyttävä, jotta yritykset pystyivät tuottamaan geneerisiä laitekatalogeja, jotteivat tuotantokulut päätyisi ikuiseen nousujohteeseen, lisäksi yritykset uskoivat sen auttavan laitteiden pysyvän varteenotettavana ratkaisuna lähitulevaisuudessa. Kun koko kapasiteetti ei ole välittömästi käytössä, kehittäjille jää vapaus päivittää tulevaisuudessa olemassa olevaa laitetta ottamaan käyttöön hyödyntämättömiä lohkoja käyttöiän pidentämiseksi. Lisäksi väylien sekä yhteyksien standardoiminen ulkoisten sekä sisäisten komponenttien välillä oli tärkeä tavoite, kun FPGA muuntautui järjestelmäpiiriksi, mikä avasi tietoliikenneyrityksille mahdollisuuden

korvata kalliimpia järjestelmiä, koska FPGA-SoC:lla saavutettiin yhtä suuria tai suurempia datatietueiden leveyksiä sekä käsittelyn liukuhihnoitusta. (Trimberger, 2015:325-328).

FPGA-SoC –laitteiden kehitys on tapahtunut alle 40 vuoden aikana ja niiden pääkäyttökohde ja -tarkoitus ovat muuttuneet useamman kerran uusien asiakkaiden ja kehittäjien keksiessä mahdollisuuksia tälle monipuoliselle kokonaisuudelle. Kehityksen suunta vaikuttaisi olleen hukassa alkuvaiheilla, kun yritettiin vallata alaa kaikkialta ja löytää parasta mahdollista markkinarakoa, joten vuosikymmenten kuluessa markkina ei lähtenyt samanlaiseen lentoon automaatioprosessien kanssa. Minkä takia uskon, että yleisen kehityssuunnan löydyttyä alkaa markkinan merkittävä kasvu. Grand View Researchin tutkimuksen (2020) mukaan oli komponenttipuolella pelkästään prosessorien markkina 83 miljardia Yhdysvaltojen dollaria, vaikka siitä poistaisi yksityishenkilöiden käytössä olevat kulutustuoteprosessorit, jotka ovat itsessään suuremmat markkinat tällä hetkellä kuin FPGA-SoC –markkina, jonka voidaan arvioida olevan 10-20 miljardia, kun arvioidaan Intelin suorittamaa suoraa Alteran käteisostoa hintapremiumilla, joten FPGA-SoC –markkinoilla on hyvin kasvuvaraa tulevaisuudessa.

2000- ja 2010-luvulla kasvua sai aikaan mikroprosessorin sekä muistin lisääminen FPGA –piirille mukaan, jolloin saatiin korkean tason ohjelmointikielien edut tuotua mukaan ohjelmoitavan logiikan kanssa samalle sivulle. Lisäksi Intel –yhtiön astuminen markkinoille mukaan hankkimalla Alteran tuo markkinoille paljon potentiaalia Intelin tietotaidon sekä investointikyvyn mukana. Uskon hankinnan johtavan 2020- ja 2030-luvuilla kulta-aikaan, kun kehitysrahaa ja intressejä on syydetty markkinoiden kasvattamiseen useamman vuoden ajan 2010-luvun puolella ja näiden vuosien tuotokset alkavat tuottamaan hedelmää 2020-luvulla. Vaikken vahvasti uskokaan FPGA-SoC –laitteiden nousevan tärkeiksi kuluttajatuotteissa voivat alan yritykset vallata yrityskuluttajien markkinoilla perinteisemmän elektroniikkaporttipiirien markkinat isoilta osin, kun 5G-verkkkojen rakennus on edennyt ja yhä useampi tuotantolaitos haluaa kytkeä tuotantonsa laitteet etähallittavaksi IoT:n avulla ja haluavat lisätä automaatioprosessiensa osuutta koko tuotantoprosessista sekä saada mukaan nopeampia sekä ”älykkäämpiä” laitteisto- ja ohjelmistokokonaisuuksia.

1.2 Työn tarkoitus

Tämän diplomityön tarkoituksena on tutkia mahdollisuutta syöttää kuvadataa käsiteltäväksi FPGA-SoC –piirille signaalinkäsittelyä varten kuten erilaisten kuvasuotimien käyttämistä. Kuvadataa voidaan kameravalmistajien SDK–pakettien (Software Development Kit) sisältämien ohjelmointirajapintoja (Application programming Interface, API) avulla syöttää C/C++–kielisillä ohjelmilla eri AXI –väyliä (Advanced eXtensible Interface) käyttäen läpi FPGA-SoC –järjestelmäpiirin. Tässä työssä perehdytään C++–kielisen ohjelman luomiseen, jota voidaan tulevaisuudessa jatkojalostaa käytettäväksi FPGA-SoC–piirien ohjelmoinnin osana. Työssä perehdytään näihin kysymyksiin:

- Miten kuvadataa kannattaa käsitellä FPGA-SoC:lla?
- Miten gammakorjaaminen ja bayer-filtteri vaikuttaa kuvanlaatuun?
- Onko kuvankäsittely FPGA-SoC:lla perusteltua?

1.3 Rajaus

Diplomityön tarkoituksena on saada syötettyä C/C++ –kielisen ohjelman välityksellä kuvadataa FPGA-SoC –piirille käsittelyä varten, joten työ keskittyy siihen, miten kuvadata on muokattu käsittelyä varten eikä siihen mitä muutoksia kuten esimerkiksi suotimia varsinaiselle kuvalle tai sen sisältämälle informaatiolle suoritetaan laskentapiirin avulla. Diplomityössä käytetään yksinkertaista suodinta ohjelman toimivuuden varmistamiseksi.

2 FPGA-SYSTEM ON A CHIP

FPGA-SoC –järjestelmäpiirien laitekokoonpano on muuttunut merkittävästi vuosikymmenten saatossa ja mukaan on tullut SoC –järjestelmäpiirin edellyttämät komponentit: suoritin eli CPU, ulkoinen tallennustila sekä keskusmuisti. Tästä eteenpäin FPGA viittaa koko järjestelmäpiiriin ja suoritin sen ARM –prosessoriin. Tavallisesti lisäksi on piirillä asennettuna erilaisia lisäkomponentteja, kuten näytönohjain eli GPU sekä signaalinkäsittely-yksiköitä sekä DA-muunnin (DAC), jolla digitaalinen signaali muutetaan analogiseen muotoon ja A/D-muunnin (ADC), jolla signaalia voidaan muuttaa analogisesta muodosta digitaaliseen (Ranta, 2012:11,13).

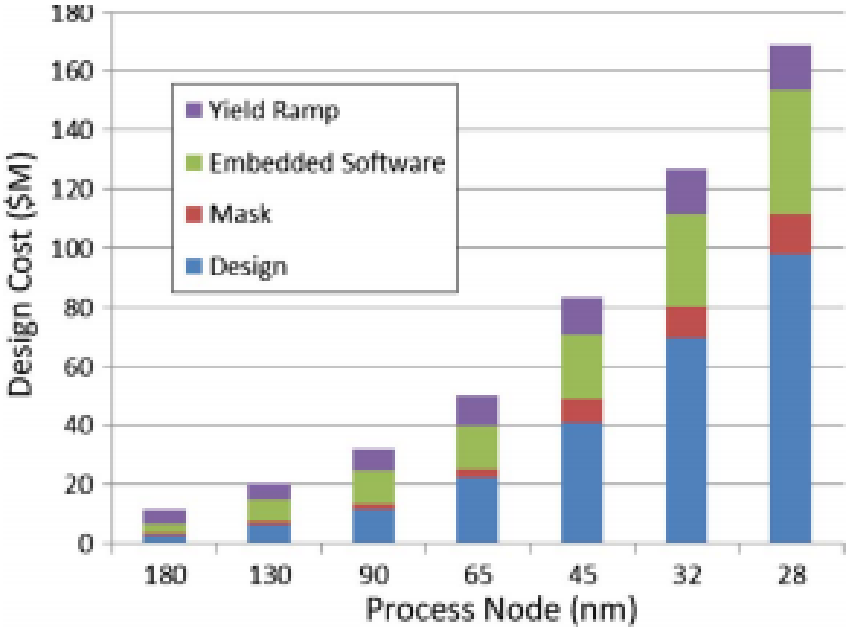
FPGA-SoC:n rinnakkaisen toiminnan mahdollistaa sen sisältämät muut osat, joita ei perinteisiltä järjestelmäpiireiltä löydy, kuten määriteltävät logiikkalohkot (CLB), jotka sisältävät logiikkakomponentteja, joista esimerkkeinä voidaan mainita LUT –hakutaulut (Lookup table) sekä erilaiset kiikut kuten D- sekä T-kiikut pientä muistiarvon säilyttämistä varten, jotka ovat kytkettynä toisiinsa ja yleensä kykenevät itsenäisesti suorittamaan yksinkertaisia toimintoja sekä sisältävät hieman muistia kiikkujen muodossa.Digilentin tuottama ZYBO Z7 FPGA-SoC–järjestelmäpiiri sisältää näistä osista useimmat, kuten taulukko 1 esittää.

Taulukko 1. ZYBOn lohkojen sisältöä (Digilent 2020).

Komponentit	*Lukumäärä*
LUT –hakutaulut	17,600
Kiikut	35,200
CMT	2

Kuvassa 1 kuvataan Xilinxin sekä Gartnerin keräämää yleisdataa, jolla Trimberger perustelee artikkelissaan kustannuksien nousua osastoittain, kuinka prosessin pieneminen sekä kuinka ohjelmiston monimutkaistuminen sekä piirin suunnittelu ovat vaatineet kasvavissa määrin resursseja yrityksiltä ajalta, jonka aikana FPGA siirtyi pelkästä ASIC –

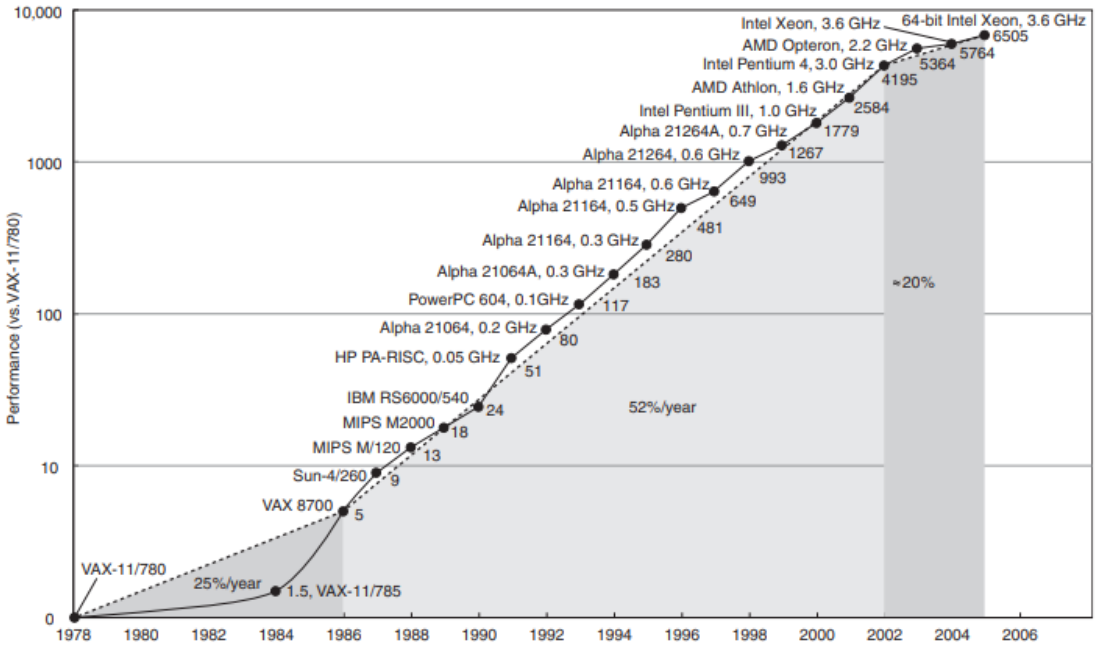
piirien korvaajasta monipuoliseksi yleislaitteeksi, kun ASIC –laitevalmistajat eivät päässeet voitolle ennen tuotteen käyttöönottoa sekä usein sekään ei riittänyt kulujen kattamiseen, kun taas FPGA saatiin nopeammin käyttöön halvemmilla aloituskuluilla (Trimberger, 2015:319).



Kuva 1. Kuvaaja, joka arvioi piirin suunnittelun keskiarvohintaa Xilinx ja Gartnerin tarjoaman datan avulla (Trimberger, 2015:328).

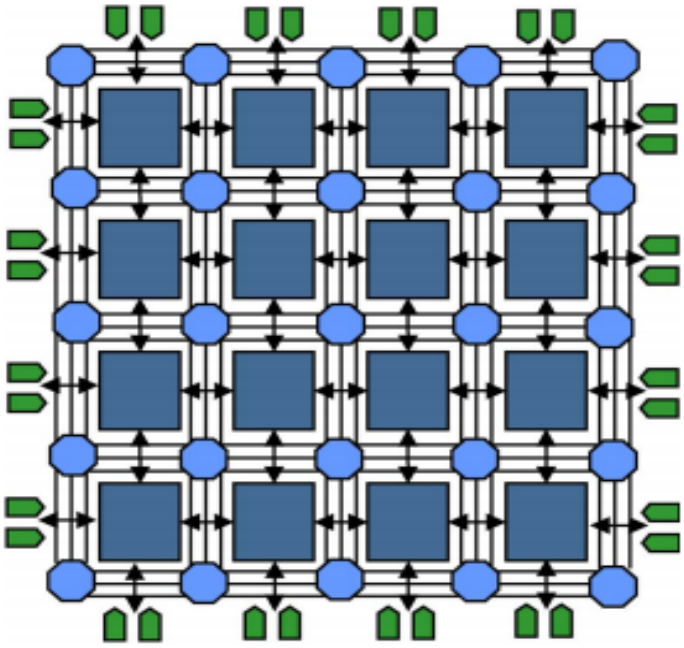
FPGA:n perustarkoitus sekä idea on säilynyt muuttumattomana alusta asti kuten esimerkiksi muiden perinteisten elektroniikkalaitteiden esimerkkinä tietokoneiden kanssa, joiden laskentateho on kasvanut huomattavasti vuosikymmenten aikana. Kuvassa 2 on esiteltynä kahden vuosikymmenen aikana tapahtunutta kehitystä PC –tietokoneiden suorittimissa. Laskentatehon voimakkuus FPGA-SoC –laitteella perustuu massiiviseen rinnakkaisuuteen, jossa jokaisen kellosyklin aikana suoritetaan kaikki ohjelmanosat toisin kuin suorittimilla ja mikroprosessoreilla, joissa kellosyklin aikana suoritetaan ohjelma kerrallaan edeten järjestelmällisesti eteenpäin. Koska tietokoneiden suorittimien teho on kasvanut huomattavasti ja niiden teho ei perustu rinnakkaiseen toimintaan, voidaan FPGA-SoC:n laskentatehokkuuden voivan kasvaa vähintään yhtä tehokkaasti Tämän mahdollistaa FPGA-SoC:n piirin lohkorakenne, jonka ansiosta ohjelman eri osaset käyttävät omia

lohkojaan, jonka ansiosta jokainen suoritetaan samanaikaisesti kellon tahdissa. (Ranta, 2012:22).



Kuva 2. 1980-luvun puolivälistä 2000-luvun ensimmäisen vuosikymmenen puoliväliin 20 vuoden aikana tapahtunut suoritintehon kasvu verrokkipisteenä käytettiin VAX-11/780 suorittimeen (Martin, 2012:1).

Kuvassa 3 esitellään Trimbergerin toimesta esimerkkiarkkitehtuuri FPGA:n lohkojärjestelylle. Siinä on kuvattuna 4x4 –lohkomatriisi, jossa siniset lohkot ovat erilaisia toimintalohkoja, kuten esimerkiksi binäärilukuja laskevia laskentalohkoja (kertolaskuyksikkö), jotka koostuvat logiikkaporteista, ja kuten muistilohkoista, joissa voidaan kevyt muisti suorittaa kiikuilla. Siniset ympyrät sijoitettuna lohkojen kulmiin kuvaavat ohjelmoitavia kytkimiä, viivat eri komponenttien välillä kuvaavat kytkentöjä piirillä eri komponenttien välillä, joita voidaan hallita esimerkiksi kytkimillä. Ulkokehälle sijoitetut vihreät elementit kuvaavat I/O –väyliä. (Trimberger, 2015:320-321). Näillä väylillä voidaan syöttää kuvadata Pcam –väylällä lohkoille käsiteltäväksi sekä suorittaa datan ulossyöttö HDMI –portin (High-Definition Multimedia Interface) avulla monitorille.



Kuva 3. Esimerkki lohkorakennearkkitehtuuri FPGA-SoC:lle (Trimberger, 2015:320). Siniset ympyrät ovat kytkimiä, siniset lohkot laskentalohkoja, vihreät nuolet I/O –väyliä sekä viivat ja mustat nuolet yhteyksiä näiden välillä.

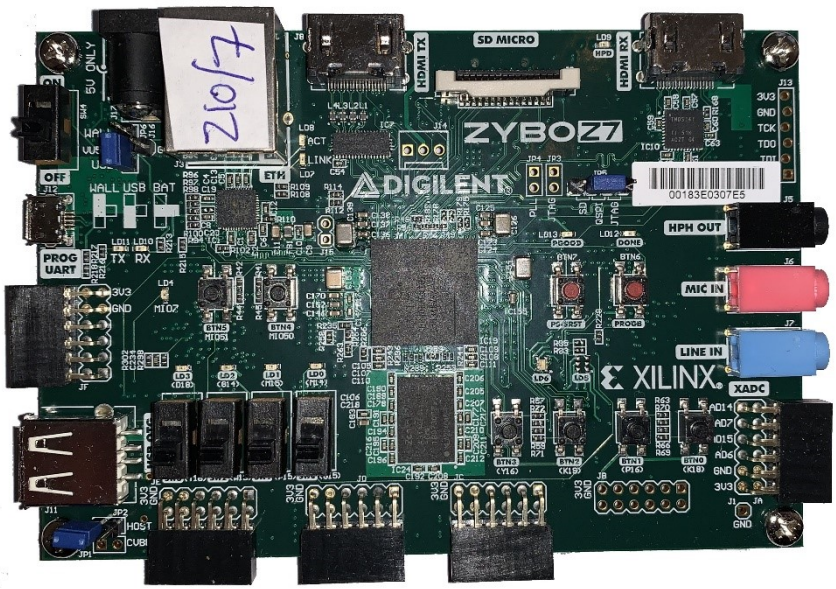
2.1 Esimerkkilaite sekä -ohjelmisto

Esimerkki FPGA-SoC–laitteistona esitellään Digilent –valmistajan valmistamaa ZYBO Z7-10 –laitetta. Digilent –yhtiö ilmoittaa kotisivuillaan olevansa National Instrumentsin tytäryhtiö (Digilent, 10/2020). Laite on suunniteltu olemaan käyttövalmis piiri, jota voidaan käyttää pohjana erilaisten sulautettujen järjestelmien ohjelmistojen testaamiselle sekä digitaalisten piirien mallintamiselle. Laitteessa on 2010-luvun tuotteelle oletettavia monipuolisia komponentteja sekä sisään- ja ulostuloväyläkytkentämahdollisuuksia, kuten taulukosta 1 on havaittavissa muutamia perustietoja teknisistä ominaisuuksista. Lisäksi laitevalmistajan tuotesivusto luettelee tärkeimpiä ominaisuuksia:

- USB-UART kytkettävyys (Universal Asynchronous Receiver Transmitter)

- USB-JTAG –ohjelmointipiiri (Joint Test Action Group), mikä mahdollistaa ohjelmoinnin suoraan USB–väylän välityksellä
- 4 liukukytkintä
- 5 yksiväristä LEDiä (Light-emitting diode) käytettäväksi
- 6 painettavaa kytkentä
- 2 RGB LEDiä
- 40 sisääntulo sekä ulostuloväylää FPGA-SoC puolelle
- 8 mikroprosessorin sisääntulo sekä ulostuloväylää
- Käytettävissä 5V voimalähteellä tai USB–väylän kautta

Laitteiston kanssa voidaan käyttää esimerkiksi ohjelmistoa Vivado Desing Suitea. VDS:n tarjoaa Xilinx-yhtiö kotisivuillaan ladattavaksi lisensoimiensa laitteidensa ajamiseksi. Diplomityön suoritushetkellä 2020 lokakuussa käytössä on versio 2019.1. Tämä ohjelmoimiseksi tarjoaa valmistaja ilmaisen Vivado Webpack –kokonaisuuden, ilmaisuus on voimassa lataushetkellä lokakuussa 2020, jota voidaan käyttää kirjoitetun ohjelman ajamiseen FPGA-SoC –laitteistolla. (Xilinx, 10/2020). Vivadolla on monia etuja perinteisiin ohjelmointiympäristöihin, koska se tarjoaa automaattisen laitteisto-ohjelmointikielten automaattikirjoittamisen sekä rakenteiden suunnittelun. Lisäksi ohjelmalla pystytään tuottamaan FPGA-SoC –piirin ohjelmointiin tarvittava binääritiedosto (bitsream) ja tuomaan FPGA-SoC:n laitteiston kuvaus SDK:n käytettäväksi. Vivadon SDK puolella luodaan sitten tarvittavat C/C++ -ohjelmat ohjelmoijan halujen mukaan. Tässä vaiheessa voidaan luoda suotimet sekä luoda ohjelma esimerkiksi kuvadatan resoluutiolle.



Kuva 4. ZYBO Z7 –piiri kuvattuna ylhäältä päin.

Ohjelmiston käyttöönottoon löytyy kattava askel askeleelta läpikäynti ZYBOn valmistajan (Digilent, inc 10/2020) kotisivuilta sekä DesingSpark –sivustolta löytyy kattava käyttäjän luoma aloittelijan projekti (awong, 6/2017), jonka avulla saatiin ZYBOn kanssa keskusteleva ohjelmisto asennettua henkilökohtaiselle tietokoneelle, joten laitteistoon tutustuminen on käyttäjäystävällistä ja ketterää erilaisille toimintaryhmille ja -ympäristöille, kuten ammattikorkeakouluille sekä yliopistoille. Tällä varmistettiin laitteiston toimivuus ennen kuin siirryttiin varsinaisen kuvadatan liikkumisen kuvaavan liukuhihnan suunnitteluun.

Taulukko 2. ZYBOn teknisiä tietoja valmistajan (Digilent 2020) mukaan.

Komponentit	*Ominaisuudet*
Mitat	88 mm x 122 mm
Piiri	Xilinx Zynq XC7Z010-1CLG400C
Prosessori	ARM Cortex-AR9 667MHz dual-core
FGPA	Xilinx 7000-sarja
RAM-muisti	1 GB DDR3L
Internet-yhteydet	Ethernet
Liitettävyys	MIPI CSI-2, Pcam, Pmod, HDMI in/out

Taulukosta 2 on listattuna komponentteja valmistajan mukaan, josta näemme työn suorittamiselle olennaista tietoa, kuten paljonko muistia on käytettävissä, mikä piirisarja on käytössä sekä millaisia liitettävyyksiä on tarjolla. Niitä käytössä monipuolisesti kuten MIPI CSI-2 kameraliitäntä, jolla voidaan liittää kamera kiinni ja yhdistää suoraan piirin kanssa ilman, että FPGA-SoC –piirille lisätään lisälaitteita. Tällä varmistetaan myös, ettei kameralta syötettävä kuvadata pääse muutosten takia menettämään dataansa, jota tarvitaan tarkkuuden ja resoluution ylläpitämiseen. Kompressio sekä kuvadatan muuttuminen matkanvarrella voivat johtaa roskadatan ilmenemiseen (Haines, Chuang; 1992:4-5). Lisäksi erittäin positiivisena lisänä on täysikokoinen HDMI –liitäntä, jolla saadaan korkealaatuista kuvaa monitorille, joka kykenee 24 bitin (8 bitittä punaiselle, 8 bittiä vihreälle ja 8 bittiä siniselle) HDMI –yhteyden tarjoamaan kuvanlaatuun.

2.1.1 Kuvadatan tiedostomuoto

Kameralta tulevasta kuvadatasta voidaan tallentaa yksittäisiä ruutuja Windows –pohjaisilla järjestelmillä esimerkiksi helposti käsiteltävissä olevaan formaattiin: PNG (Portable Network Graphics). Vaihtoehtoisesti voidaan käyttää myös BMP –kuvaformaattia (bitmap), joka on kompressoimaton. Kuvaformaattia yleisesti kuvataan joukoksi pikselitaulukoita ja se sisältää joukon taulukoita, joissa pikseleitä kuvaa taulukon soluun asetettu kokonaisluku (Muller, VanRype; 1996:8).

PNG –tiedostot soveltuvat tosimaailman sovellutuksiin, koska kuvaa ei kompressoida vaan kaikki tieto on tallessa käsittelyä varten toisin kuin esimerkiksi JPEG –tiedostomuodossa, joka on optimoitu pakkautumaan pienempään muotoon mikä myös Nasan suorittaman tutkimuksen mukaan (Haines, Chuang; 1992) asettuu ongelmaksi tietyille sovellutuksille, koska JPEG -muotoinen kuvadata alkaa menettämään dataansa ja alkaa vaurioitumaan esimerkiksi liiallisen käsittelyn sekä kompression suhteen asettuessa suuremmaksi kuin 1:120. Tämä asettuu PNG –kuvadatan ongelmaksi, koska se pitää sisällään kaiken tiedon, joten tiedostokoot voivat olla todella suuria ja hitaita käsiteltäviä perinteisille PC –tietokoneille (Muller, VanRype; 1996:72-74). Douglas W. Cromeyn julkaistussa teoksessa (Digital Images Are Data: And Should Be Treated as Such, 2012) esitetään, että tieteellisen tiedon yhtenä luottamuksen säilymisen ehtona on, ettei kuvadatan käsittelyyn liittymissä tehtävissä alkuperäinen kuvadata pääse korruptoitumaan ja täten tuloskuvan data voidaan toistaa tarvittaessa, jotta voidaan todistaa tutkimuksen oikeellisuus sekä vältetään väärinkäytökset.

Osamatriiseja voidaan käsitellä rinnakkain tai jonossa FPGA:lla sekä ulostuloa varten kasattava kuva voidaan muodostaa jälkikäteen osamatriisien avulla rinnakkain suoritettujen tulosmatriisien käyttäen. Vaihtoehtoisesti voidaan häviöttömän tallennuksen ansiosta myös jatkuvasti päivittää kesken suoritteen tulosmatriisia, jolloin prosessin keskeytyessä virheeseen tulosmatriisi jää vajaaksi, mutta rinnakkaisen suorituksen keskeytyessä jää tulosmatriisi kokoamatta, jos osamatriisien käsittelyä ei ole suoritettu loppuun.

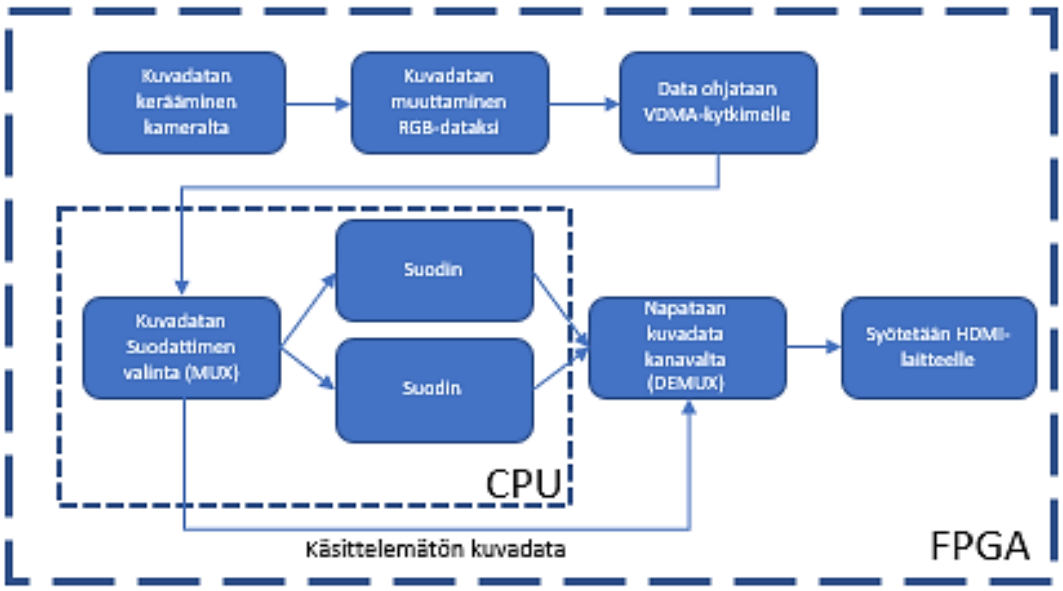
Tietoliikenneverkossa kasvavan datamäärän aiheuttama kuorman purkuun tarvitaan nopeampia erikoislaitteita perinteisten PC –tietokoneiden tilalle, kuten esimerkiksi FPGA-SoC. Tämän avulla voidaan raskas työtaakka suorittaa FPGA:lla ja suorittaa kuvien jälkianalyysi ihmisten valvonnassa erillisillä valvontatietokoneilla, jolloin FPGA-SoC:lle ei tarvitse asettaa raskaita taustaohjelmia sekä ajureita, joita täytyy ajaa taustalla toiminnan varmistamiseksi, mitä käyttöliittymät PC –tietokoneissa tarvitsevat. FPGA-SoC:n oma ARM –mikroprosessori kykenee tehokkaasti pyörittämään valmista ohjelmaa, FPGA:n laskentapiirin itse tehdessä halutut raskaammat toimet.

3 LIUKUHIHNAN SUUNNITTELU

FPGA-SoC:n algoritmin suunnittelussa tulee pitää mielessä mahdollisia esiin tulevia ongelmia digitaalisen signaalinkäsittelyn tarvitsemista resursseista sekä niiden rajoitteista ja erilaisia nopean suunnittelun alueaiheita. Monet liittyvät suunniteltuun rakenteeseen ja algoritmin sekä arkkitehtuurin rajoitteisiin sekä kellonkäyttöön, jotta saadaan optimaalinen piirirakenne (Wang, Chen; 2019:7-9). Siksi monet kehityspiirien tarjoajat ovat luoneet kirjastoja, joissa on valmiiksi suunniteltuja rakenteita rajoitteineen käytettäväksi, jotta käyttäjän on helpompi aloittaa projektin käynnistäminen ottamalla kirjastoja käyttöön ja lähteä työstämään niitä.

Tässä työssä käytetään apuna Digilentin tarjoamaa Zybo Z7-10 AP SoC –laitetta (All Programmable System on a chip), joka perustuu Xilinx 7000-sarjaan, joten piirinsuunnittelu jätetään suorittamatta ja voidaan suunnitella pelkkää algoritmia. Algoritmin suunnittelua vaikeuttaa se, että kaikkien kamerasensorien kanssa laitteet eivät ole helposti yhteensovitettavia, jos portit eivät ole identtisiä (Wang, Chen; 2019:8-9). Jotkut laitevalmistajat, kuten Xillinx, tarjoavat valmiiksi listoja eri laitteista, joiden yhteensopivuudesta valmistaja on varma.

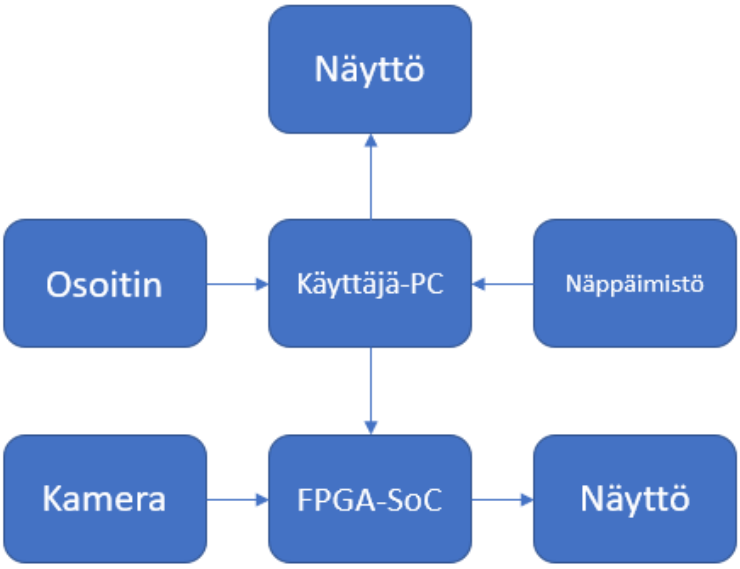
Xilinx 7000-sarjan AMBA (Advanced Microcontroller Bus Architecture) protokolla on Harvard arkkitehtuurin pohjalta rakennettu modernisoitu järjestelmä, jossa yhdistellään 32- ja 64-bitin AXI –väyliä eri komponenttien välisten mestari/orjasuhteiden muodostamisessa. Xilinxin IP –lohkot tukevat näistä 32-bittisiä väyliä oheislaitteiden ja prosessoriyksiköiden välillä. Lisäksi nämä väylät ovat tiukasti paritettuna itsenäisinä käyttöliittyminä, jotta luku- sekä kirjoitusvaiheet onnistuvat rinnakkain. IP –lohkot käyttävät AXI4 –käyttöliittymiä, joilla pystytään AXI4 virtauskirjoitukseen, jolla mestari pystyy siirtämään dataa suoraan orjan käyttöliittymään, mikä tukee videostriimaamista suoraan monitorilaitteille (Acasandrei, Barriga; 2015), koska datan virtauskirjoituksesta puuttuu perinteiset kirjoitus/lukuvaiheet sekä osoitteiden haut. (Xilinx, 2012:24).



Kuva 5. Kuvadatan käsittelyprosessin vaiheet FPGA-SoC –järjestelmäpiirillä.

Suunnittelun tukena voidaan käyttää DSP IP –lohkoja. Nämä IP –lohkot voivat olla tarjolla useista lähteistä kuten valmistajilta itseltään tai kolmannelta osapuolelta kuten erikoistuneelta ohjelmistoyritykseltä tai myös vapaajulkaisu lähteistä, kuten yksityisten henkilöiden ohjelmointialan keskustelufoorumeilta tai esimerkiksi GitHub –alustalta. Hyväksi todettuja valmiita lohkoja otetaan usein myös valmistajan sivuille esille esimerkkiprojekteiksi sekä pohjiksi oman projektin vauhdittamiseksi. DSP IP –lohkot voidaan jakaa erilaisiin ryhmiin, joista yleisimmät voisi nimetä ohjelmistotason sekä operatiivisentason suoritteiksi. (Unnikrishnan, Madhavan; 2015:1128-1129).

DSP komponentit tuovat kuvakäsittelyn elementtejä FPGA-SoC kehitykseen kustannustehokkaasti mitä siltä on puuttunut kehityskaarensa alkupäässä. Tämän ansiosta FPGA-SoC on nousemassa varteenotettavaksi laiteratkaisuksi myös digitaalisen signaalinkäsittelyn piirissä. Systeemitason design ohjelmat, kuten Vivado, ovat yksinkertaistamassa tätä kehityskaarta, kun se luo lohkotason rakenteista valmiit HDL –laitteistokieliset ratkaisut, jolla saadaan rauta ohjelmoitua kivuttomammin verrattuna siihen, että ilman niitä tulisi FPGA-SoC:n kehittäjän olla ohjelmistokehittämisen ammattilaisen lisäksi myös laitteisto-ohjelmointikielien ammattilainen.



Kuva 6. Laatikkokaavio kokonaisjärjestelmästä, jossa nuolet kuvaavat eri komponenttien vaikutusta toisiinsa.

Sen takia voidaan muita ohjelmistoja tuoda FPGA-SoC:n kehitykseen suoraa mukaan kuten MATLAB (Unnikrishnan, Madhavan; 2015:1132). Algoritmeja sekä laitteita suunnitellessa tulee ottaa etukäteen huomioon, mitä projektilta halutaan, jotta saavutetaan haluttu tehokkuus sekä nopeus. Vaihtoehtoisesti voidaan siirtyä alusta alkaen dynaamiseen malliin, jossa otetaan algoritmiin elementtejä molemmista ohjelmointikoulukunnista: sekä täysrinnakkais- että täyssarjaliikenne koulukunnista. Tätä nimitetään semirinnakkaiseksi lähestymistavaksi (Hegarty ja muut; 2006:7).

Algoritmin alussa suoritetaan kamerasensorille alustaminen, jotta kuvadataa saadaan virtaamaan FPGA-SoC –laitteelle. Sen jälkeen suoritetaan tarkemmin kuvadatan prosessoinnin liukuhihna, joka käydään kappaleessa 3.1 tarkemmin läpi. Tämän prosessin suoritus suoritetaan silmukkarakenteella, jossa algoritmia suoritetaan jatkuvasti uudelleen ja uudelleen. Tavoitteena on, että algoritmin toimiessa on kuvadataa kolmesta eri vaiheesta yhtäaikaisesti liukuhihnassa matkalla eteenpäin. Ensimmäinen on kuvadatamatriisi, joka on lähdössä kameralta matkaan kohti FPGA-SoC –piirin työmuistia ja toisena on kuvadatamatriisi, joka on käsittelyvaiheessa liukuhihnan sisällä suodinosiossa. Kolmas kuvadatamatriisi on matkalla näytölle tulostusta varten (Hegarty ja muut; 2006:2,6).

3.1 Liukuhihna

Liukuhihna koostuu kuvakäsittelyn algoritmeista. Liukuhihnaa käytetään kamerasensorin tuottaman kuvadatan kuljettamisen sekä käsittelyn mallintamiseen laitteella (Sharif ja muut, 2021:2-3). Liukuhihnan muodostavat signaalinkäsittelyn eri vaiheet sekä AXI –väylillä yhdistetyt komponentit, joihin myös käytettävät suotimet gammakorjaaminen sekä kohinanhallinta kuuluvat (Jiang ja muut, 2016). Liukuhihnoitus on olennainen osa FPGA-SoC:n prosessin suunnittelua, koska FPGA-SoC –piirin eri osat ovat luotuja suorittamaan matalan tason aritmetiikkaa, joten liukuhihnan suunnittelulla saavutetaan nopeampaa sekä rinnakkaista suorittamista. Mitä enemmän rinnakkaisuutta on saavutettu aikaiseksi, sitä nopeammaksi signaalinkäsittely muuttuu FPGA-SoC:lla.

Liukuhihnan muuttaminen lennosta voi muuttaa merkittävästi liukuhihnan suorittamiseen tarvittavia kytkentöjä ja lohkoja, joten suunnittelu kannattaa suorittaa etukäteen tarpeeksi perusteellisesti, että tältä vältyttäisiin (Jiang, ja muut; 2016:1). Tätä rakennetta auttaa FPGA-SoC –laitteiden runsas rekistereiden määrä. Rekistereillä on monipuolinen rooli liukuhihnoituksen vaiheessa, koska eri aritmeettiset operaatiot erotetaan toisistaan rekistereillä, joilla saadaan erotettujen matemaattisten operaatioiden väliin syötettyä myös muualta saatua dataa, jonka ansiosta kyetään leventämään liukuhihnaa, jolla saavutetaan haettua rinnakkaisuutta eri operaatioiden suorittamisen kanssa, jotta algoritmin suorittaminen ei hidasta ja suoritusnopeus säilyy algoritmin monimutkaisuuden kasvaessa. (Bailey, 2019:2-4).

3.2 Kuvadatan alkukäsittely

Kuvadata tulee ohjata eri lohkoille muokattavaksi, jotta se saadaan ulostulon kanssa yhteensopivaksi kuvadataksi, joka tässä tapauksessa tulee olemaan 24 bittiä, 8 bittiä per väri toisin kuin sensorilta tuleva RAW16 tai RAW24. Muutokset suoritetaan IP –lohkoilla. Näitä lohkoja voidaan luoda Vivadon IP Integrator –toiminnallisuudella, johon voi käyttää apuna IP Catalog –listaa, josta voidaan poimia haluttuja IP –lohkoja ja muokata niitä, kuten testauksessa käytettävät lohkot: gammakorjaus sekä bayer-suodin, tutummin CFA

(color filter array). CFA avulla uusi muokattu kuvadatan muoto väritetään demosaicing –tekniikalla, kun suodin päästää lävitseen aina yhden RGB –väridatasta ja loput kaksi lasketaan interpolaation avulla (Bae, 2020:2) ja lopuksi suoritetaan gammakorjaus, jotta saadaan kontrastieroja tasoitettua kamerasensorin sekä monitorin väliltä sekä paranneltua valotusta.

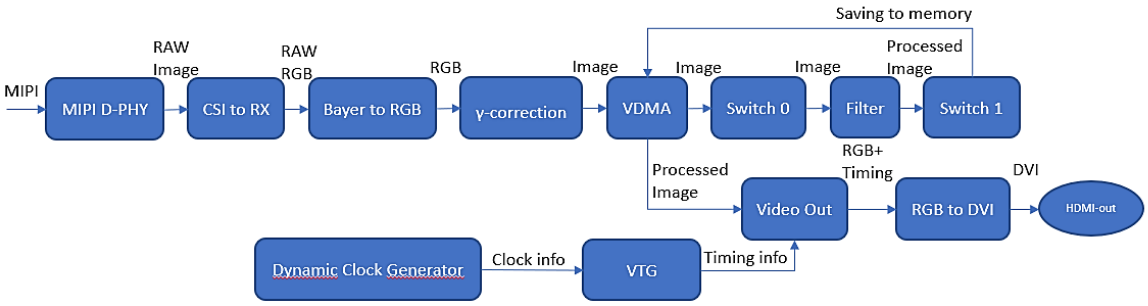
3.3 VDMA

Kuvadata ohjataan VDMA –lohkolle (Video Direct Memory Access), jonka tehtävä tulee olemaan CSI-2 rajapinnan kautta haetun kuvadatan sekä kellosignaalin hakeminen ja tämän tiedon tallentaminen muistiin kuvapuskureiden avulla, josta tätä dataa saadaan haettua liukuhihnan seuraavassa vaiheessa muokkausta varten sekä tallettaa aina uusin kellotieto sensorilta sekä muunneltu kuvadata. Lisäksi tulostettavaksi menevä kuvadata haetaan VDMA:n avulla muistista.

3.4 IP –lohkot liukuhihnoitukseen

Käytetyt IP–lohkot liukuhihnan suorittamiseksi on listattuna seuraavassa luettelossa. Nämä ovat tarjolla Zybo Z7 laitteella Vivadon IP katalogin tarjonnasta. Näistä IP –lohkojen AXI4-Stream Switch 0 ja 1 väliin voidaan sijoittaa halutut suotimet, joiden ohjaamiseen voidaan ottaa käyttöön valmiina lohkoja yhdistämällä niiden ja Switchien väyläyhteydet toisiinsa tai vaihtoehtoisesti kirjoittaa suorittimella suoritettavia suodattimia, joihin kirjoitetaan omat C++ –operaatiot kunhan kuvadata siirretään FPGA:lta CPU:lle ja muodostetaan niiden välille väylän avulla yhteys, jolla saadaan lennosta vaihdettua erilaisia suotimia toisiin tai vaihtoehtoisesti suorittamaan yhteen nivottuja suotimia samanaikaisesti yhdelle videokuvalle. Valmistajalla on tarjolla muutamia erilaisia valmiita IP –lohkoja erilaisia signaalinkäsittely toimille kuten vaikka reunojen havainnointiin tai värien invertoimiseen. Tästä kuva ohjataan sitten tässä liukuhihnassa multiplikserille tai vaihtoehtoisesti demultiplekserille (Switch 0), jos halutaan suorittaa vaihtoehtoisia suo-

dattimia tai ohjata kuva suotimien ohi, jos halutaan tarkkailla esikäsiteltyä kuvadataa ilman lisäsuotimia, kuten reunahavainnointia, toiselle demux/mux –kytkimelle (Switch 1). Tämän jälkeen liukuhihnassa on kuvadataa valmiina siirrettäväksi muistiin.



Kuva 7. Liukuhihnan visualisointi.

Kuvassa 7 on näkyvissä liukuhihnan rakennetta. Liukuhihna on kuvaus, miten kuvadata kulkee sekä mistä lohkosta mihin AXI–väylät kulkevat ja mitä eri vaiheissa ja väleissä suoritetaan, kuten gammakorjaamista sekä RAW –muotoisen kuvadatan suotiminen bayer-suotimella (Jiang, ja muut; 2016:9).

Kuvadataa kerätään MIPI CSI-2 fyysisen D-PHY –rajapinnan avulla, joka on MIPI Alliancen kehityksessä. Tämän CSI-2 käyttöliittymän avulla pystytään MIPI Alliancen mukaan tukemaan myös tulevaisuuden resoluutioita ja se tukee tämän hetken 1080p (Full HD), 2160p (4K) sekä 4320p 8K –kuvatarkkuuksia. D-PHY –rajapinnalla kyetään tuottamaan jokaiselle pikselille RAW-10 tai RAW-24 bitin värisyvyys. Nopeutta nelikaistaisella D-PHY –väylällä on 18 Gbps (gigabittiä per sekunti) (MIPI, 2021). Seuraavassa luettelossa käydään nämä lohkot läpi sekä mikä niiden päätoimi rakenteessa on.

- MIPI_D_PHY_RX: Tämä on lohko, joka hoitaa fyysisen tason rajapinnan sisääntuloväylän datajonosta eteenpäin syötettävän datan muokkaamalla kuvadatan bittisarjaa keräämällä datarakenteet yhteen ja syöttää tuloksena syntyneen datan eteenpäin.

- MIPI_CSI_2_RX: Tässä lohkossa saatu kuvadata muokataan RAW RGB –matriiseiksi, jotka syötetään eteenpäin käyttäen hyväksi AXI-Stream väylää. Tämän lohkon tuote on esiaste siitä kuvadatasta, joka tulee suotimien käsiteltäväksi myöhemmin liukuhihnassa.

- AXI_BayerToRGB: Tämä lohko muokkaa halutun datan tavanomaiseen RGB muotoon, joka on standardoitu 32 bittinen muoto, jossa 30 bittiä jaetaan värien kesken sekä viimeiset kaksi bittiä on varattu toppaussoluille matriisin muodostusta varten.

- AXI_GammaCorrection: Gamman korjauksella saadaan kuvadata muutettua 8 bitin syvyiseksi jokaiselle värille, jolloin kokonaiskooksi saadaan 24 bittiä, joka on samalla se muoto, joka syötetään myös HDMI –väylältä näyttöpäätteelle.

- AXI_Video Direct Memory Access: Tällä lohkolla muodostetaan prosessorille käytettäväksi kuvapuskureita kolme kappaletta. Tämän lohkon kuvapuskureiden määrän säätämisen avulla saadaan määriteltyä millä virkistystaajuudella ulostulon kuvadata näkyy monitorilla. Tavallisimmat virkistystaajuudet tälle ovat 30 Hz sekä 60 Hz. DMA suorittaa sekä muistilohkojen että muistiosoitteiden hallintaa siten, ettei suorittimen tarvitsee suorittaa itse luku– tai kirjoituskäsky. Tämän lohkon datana toimii yksi ruutu videokuvadatasta, joita sitten yhdistetään halutun virkistystaajuuden saavuttamiseksi. (Harvey, 1991:12-17).

- AXI4-Stream Switch 0: Tällä lohkolla oikeaan resoluutioon käsitelty kuvadata siirretään halutuille signaalinkäsittelylohkoille, jotka voivat sisältää esimerkiksi histogrammeja, rajatunnistamista, mediaaniarvojen hakemista ja implementoimista riippuen mitä on valittu haluttavan. Esimerkiksi vaihtamalla tämän switchin muxin ja switch 1 demuxin voidaan suorittaa rinnakkain useita suotimia yhdelle kuvadatavirralle ja muodostaa näistä rinnakkaisista kuvavirroista päällekkäisen kuvavirran, jolla on useita eri suotimia.

- AXI4-Stream Switch 1: Tällä lohkolla valitaan mikä kuvadatan virroista viedään eteenpäin. Toisin sanoen valitaan minkä suotimen läpivirtaavan signaalikäsitellyn kuvavirran halutaan tulevan lähetetyksi eteenpäin kohti monitoria. Switch –lohkojen välissä on suodattimien paikka.

- Dynamic Clock Generator: Tämän lohkon tehtävänä on muodostaa haluttu kellotaajuus HDMI –ulostuloportille. Kellotaajuuden määrittämiseen käytetään prosessorilta saatavaa virkistystaajuutta sekä resoluutiota, jotka on määritetty VDMA –lohkon avulla.

- Video Timing Controller: Tällä lohkolla luodaan videokuvan synkronointisignaalit, joita haluttu ulostulomuoto tarvitsee toimiakseen, kuten esimerkiksi VSync tai HSync, joilla suoritetaan kuvadatan vertikaalista tai horisontaalista synkronointia monitorin virkistystaajuuden mukaiseksi, jotta kuvadata ei repeile tai pirstaloidu epäselväksi eikä siihen ilmesty siihen kuulumattomia partikkeleita, joita voisi epähuomiossa tulkita merkitykselliseksi kuvadataksi.

- AXI4-Stream to Video Out: Tällä lohkolla kirjoitetaan lopulliseen muotoon, jotta se voidaan viimeisessä lohkossa kääntää HDMI:n tarvitsemaan muotoon. Tämä lohko saa switch 1:ltä signaalikäsitellyn kuvadatan sekä VTG –lohkolta tarvittavat signaalitiedot, jotka sitten sidotaan yhteen.

- RGB to DVI: Tämä lohko muuttaa saadun kuvadatan näytettäväksi yhdeksi bittijonoiseksi kuvadataksi, jonka sitten ulostuloon kytketty monitori viimein pääsee näyttämään.

3.5 ARM –suorittimen C++ –algoritmia

Alla on pseudokoodia C++ –ohjelmasta, YLÖSAJO, jonka suorituksen aloituksen yhteydessä ottaa tarvittavat komponentit sekä muistin käyttöön. Tällä saadaan kuvadatan virtaus liukuhihnassa käyntiin ja muisti käyttöön sekä pääohjelman osa, jolla ARM –suoritin

suorittaa videokuvan laatu- ja suodinvalintaa. Ohjelmassa käydään läpi muistin aktivoiminen sekä kameran tuottaman datan kuvalliset ominaisuudet, jotka voivat olla esimerkiksi Full HD 1920x1080 60Hz resoluutiolla.

Tähän ominaisuuteen vaikuttaa Intelin ja Adoben, 2015, luoman 4K –editoinnin rautaohjeistuksen mukaan paljonko käsittelevällä laitteella on tehoa, koska mitä tarkemmaksi kuva halutaan tai vaihtoehtoisesti sen kuvataajuutta halutaan kasvattaa, tarvitaan myös laitteistolta kykyä käsitellä kasvavaa datamäärää, joka kulkee väylillä sekä tarvitsee työmuistia käsittelyä varten. Ohjekirja kertoo, että 4K –kuvalaatu vaatii nelinkertaisen tallennustilan verrattuna Full HD –kuvanlaatuun, mikä johtaa lisääntyneeseen tehon tarpeeseen, kun pikseleiden määrä on kasvanut.

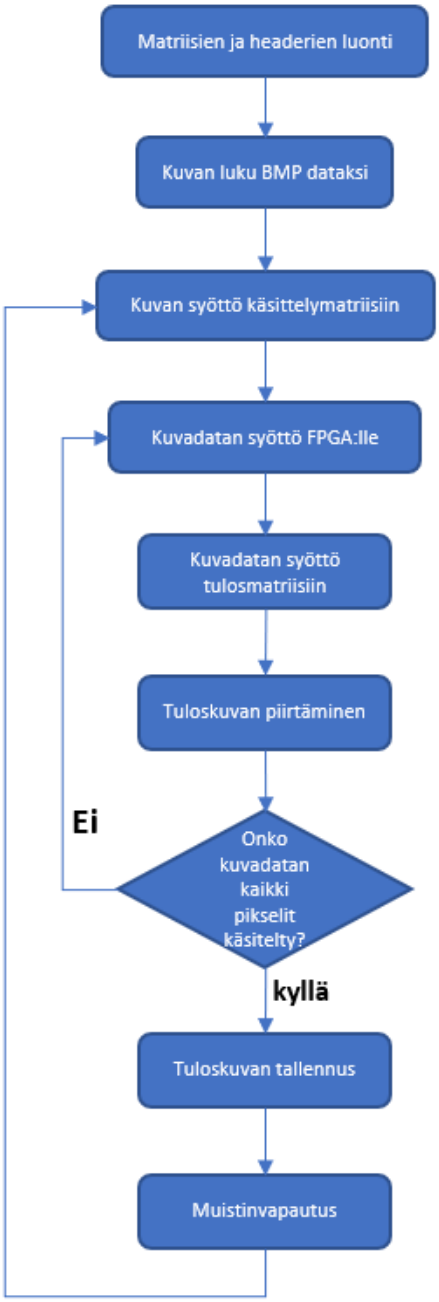
```
Metodi Liukuhihnan alustus
  Nollaa liukuhihna
  Määritä VDMA:lle ajoitustiedot sekä osoitteet
  Määritä gammasuodin
  Käynnistä kamera
  Resetoi ulostulon käsittelijä
  Aseta resoluution tiedot ulostulon controllerille
  Käynnistä ulostulosyöttö

Pääohjelma
  Alusta muisti ja resetoi FPGA
  Alusta keskeyttäjäkäsittelijä
  Alusta kameran pinni (GPIO) ja IIC -käsittelijät
    sekä kytke keskeyttäjään
  Alusta kytkimet Switch 0 ja 1
  Alusta kamera IIC- ja GPIO –käsittelijöiden tiedolla
  Alusta VDMA ja kytke keskeyttäjään
  Alusta DCG sekä VTG
  Suorita metodi liukuhihnan alustus resoluution
    parametreillä
```

Algoritmi 1. Pseudokoodi YLÖSAJO ohjelmasta FPGA-SoC:lle

Liitteessä 1 on ohjelma C++ –toteutuksen päärakenteesta, joka esitellään algoritmissa 1., jolla luodaan tarvittavat väyläyhteydet kuvadatan kuljettamiseksi FPGA-SoC:n läpi näyttölaitteelle eli käynnistetään prosessit, portit sekä liukuhihnan muistinhallinta.

3.6 Suorittimeen nojaava ohjelma

Kuvassa 8 on mallinnettuna vuokaavio algoritmista, jonka pohjalle voidaan rakentaa suoritinpohjaisen RGB-matriisin luoja sekä suotimien käyttö. Tämä ohjelma ei käytä kuvankäsittelyyn FPGA-SoC:n tarjoamia komponentteja vaan suorittimella ohjelmoituja silmukoita. Algoritmin voisi korvata tarjolla olevien kirjastojen funktioilla, mutta se on avattuna tässä kappaleessa, jotta lukija voi nähdä kuinka yksinkertaisen bittimatriisin luominen vaatii useamman sata riviä koodi. Sama logiikka on mallinnetun FPGA-SoC ratkaisun sisällä optimoituna sekä useamman kerran testattuna valmistajien sekä harrastajien toimesta, jolloin voidaan keskittyä olennaisiin digitaalisen kuvankäsittelyn teemoihin pelkän suorittavan osan hallitsemisen sijaan, kunhan ohjelmoija hallitsee tarvittavat kirjastot ja niiden toiminnallisuuden sekä mitä niillä voidaan tehdä.

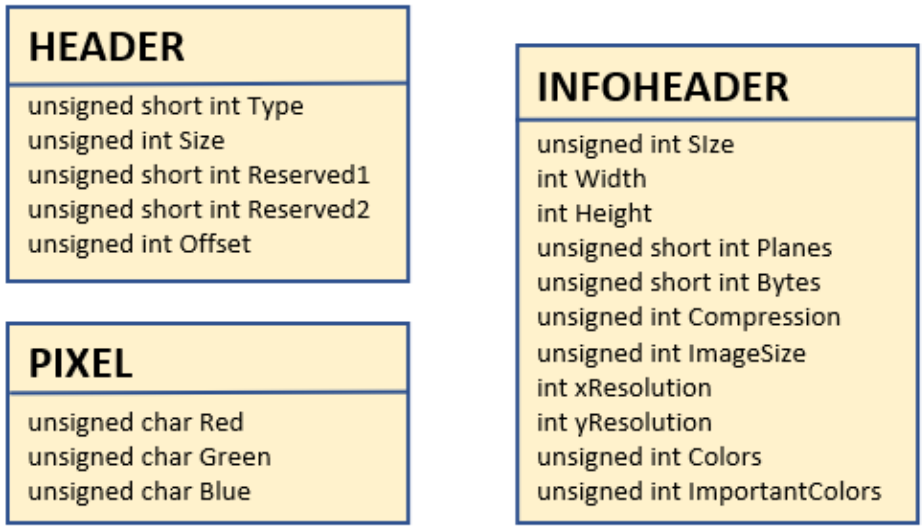


Kuva 8. Vuokaavio suorittimelle kirjoitetun FILTTERIAJO:n toimintaperiaatteesta.

3.6.1 Vaihtoehto ohjelman läpileikkaus

Tässä kappaleessa käydään suurimittaisesti läpi käsin kirjoitettuna ohjelman FILTTERIAJO matriisien muodostus sekä suotimien käyttäminen. Tämän osion tarkoitus on toimia esimerkkinä, kuinka FPGA-SoC:lla saavutetaan nopeutta, kun algoritmin toteutus on

rinnakkainen toisinkuin tässä perinteisessä suoritinmallissa, jossa ohjelman rivit suoritetaan sekventiaalisesti ja funktio funktiolta, jolloin ohjelman suorituksen nopeus riippuu enemmän suorittavan laitteen yksittäisen suorittimen laskentatehosta ja sen optimointi perustuu suoritettavien käskyjen minimointiin sekä miten tietorakenteet on tehty. Jos suoritetaan operaatioita, joiden suoritusaika on tiedossa, on suorittimella niiden kokonaisaika operaatioiden kokonaisluku kerrottuna suoritusajalla, kun vertailukohtana FPGA:n lohkorakenteen ansiosta voidaan kaikki operaatiot suorittaa yhden käskyjakson aikana. (Treece, 2017). Kuvassa 9 on kuvattuna tämän vaihtoehtoisen toteutuksen tarvitsemat tietorakenteet sekä sen attribuutit.



Kuva 9. Tarvittavat PNG –kuvan otsikkotiedot, joita tarvitaan vaihtoehtoisen lähestymistavan toteuttamisessa.

Osoittimien määrittäminen sekä tarvittavien muuttujien alustaminen on päätetty suorittaa heti metodin alussa, jotta jokaisen uuden ajokerran jälkeen muuttujat ovat varmasti nollattuna eikä muistiin jäänyttä roskaa prosessoida. Tällä tavalla tämän pääohjelman saisi muutettua myös erilliseksi funktioksi osaksi isompaa ohjelmaa, josta sitä voitaisiin kutsua yhtenä aliohjelmana. Ohjelman FILTTERIAJO pseudokoodi esitys alla:

```
Tarvittavien kirjastojen käyttöönotto
Metodi kuvadatan syöttäminen matriisiin
  Luo tietorakenteet sekä niiden osoittimet
  Alusta tarvittavat muuttujat
  Varaa muistia rakenteille
  Lue rakenteiden tietuiden määrät kuvadatasta
  Kirjoita ulostulolle tietuiden määrät kuvadatalta
  Varaa muistia kaksiulotteiseen matriisiin
  FOR jokaiselle matriisin riville
    FOR jokaiselle matriisin sarakkeelle
      Lue kuvadatan rivin ja sarakkeen solun tieto
      Kirjoita data matriisin soluun
    ENDFOR
  ENDFOR
  FOR jokaiselle matriisin riville
    FOR jokaiselle matriisin sarakkeelle
        Suorita suodin operaatio solulle
        Kirjoita uusi tieto ulostulolematriisiin
    ENDFOR
  ENDFOR
Suorita muistinvapautusoperaatiot
```

Algoritmi 2. Pseudokoodia ohjelmalle FILTTERIAJO.

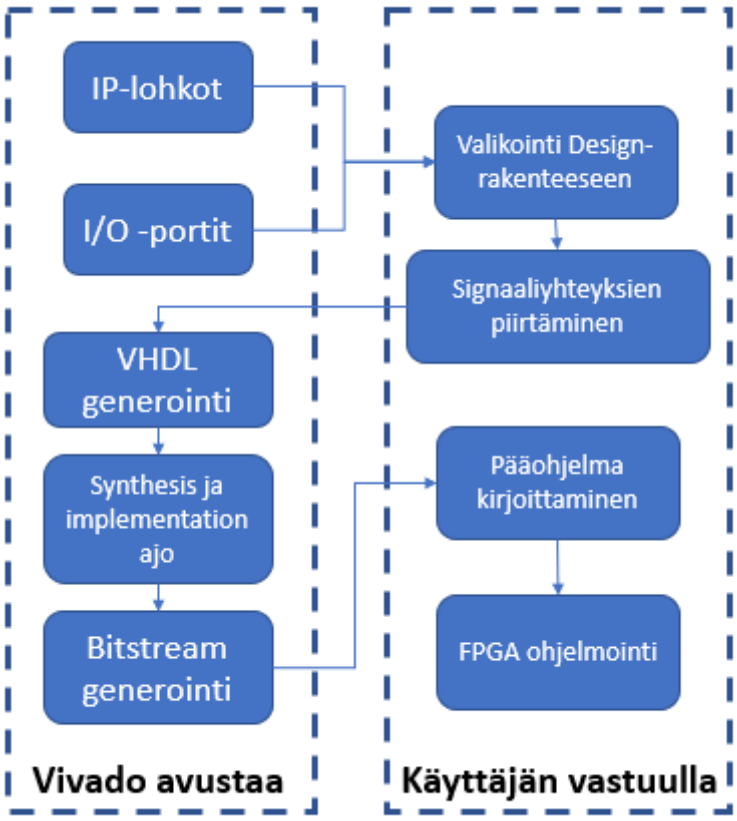
C–kielen, kuten monen muunkin ohjelmointikielen, yksi ominaisuus on sen kyky varata muistia dynaamisesti, jolloin matriisien koko kyetään muuttamaan manuaalisesti käsin lisäämällä tarvittavat muistinvaraus ja -vapautus –funktiot ohjelmaan, jolloin roskienkeruulle ei ole tarvetta. Nollilla täyttäminen on tarpeellista kokoaan muuttavalle matriisille, jotta data jakautuu matriisissa oikein esimerkiksi osamatriisien arvojen keskiarvon kautta, jotta esimerkiksi kokoaan kasvattanut matriisi ei pidä sisällään pelkkiä tyhjiä soluja lisätyissä sarakkeissa ja riveissä, tai ettei pienennetty matriisi menetä dataansa. Muistinvarauksen jälkeen suoritetaan operaatiot header – sekä infoheader –rakenteille, jotta tallentamista varten luotavalle uudelle tiedostolle saadaan sama tietorakenne kuin käsiteltävä kuva.

Matriisi muodostetaan seuraavaksi ja täytetään tallennetulla käsittelemättömällä kuvadatalla. Suorittamalla operaatio tässä, voidaan kaikki matriisin ja pikselin käsittelyoperaatiot pitää lähekkäin. Tässä vaiheessa matriisin sisältämälle kuvadatalle voidaan suorittaa matemaattisia operaatioita, jonka jälkeen voidaan käsitellyn matriisin data kirjoittaa ulossyötettäväksi kuvamuodoksi toiselle tiedostolle. Tämän jälkeen suoritetaan muistinvapautukset. Tämä ohjelma on pääpiirteiltään esitettynä liitteessä 2 ja tässä työssä se on näytteenä, kuinka ohjelma alkaa kasvaa ja koodiarkkitehtuuri monimutkaistuu, kun sen suoritukseen ei saada apua FPGA-SoC:ltä. Arkkitehtuurista on jätetty osia pois ja käsitelty vain tärkeimpiä vaiheita, joita CImg –luokan funktiot suorittavat.

3.7 Vivado

Kuvassa 10 käydään läpi prosessia, kuinka Vivadolla luodaan projekti. Siinä on lajiteltuna käyttäjän vastuualueet sekä missä ohjelmisto auttaa eteenpäin. Lohkojen ja porttien lisäämisen jälkeen, luodaan yhteydet niiden välillä piirtomekanismilla. Vaikka lohkot ja yhteydet ovat käyttäjän määritettävissä, voidaan lohkojen yhdistämisen jälkeen käyttää ohjelmiston automaattista piirtoavustinta, joka lisää tarvittaessa lohkot, jotka käyttäjältä voivat puuttua, mutta tarvitaan ohjelman rakentamiseen. Kun rakenne on valmis, voidaan rakenne varmistaa ja generoida, jolloin ohjelma rakentaa valmiiksi käyttäjälle VHDL –kieliset rakenteet sekä signaalien muodostukset, mitä käyttäjä on manuaalisesti asettanut lohkojen väliin käsin.

Ohjelman luomisen seuraava vaihe on synteesin ja implementaatio ajojen jälkeen siirtää projekti Vivadon SDK –ohjelmalle, jossa käyttäjä suorittaa pääohjelman kirjoittamisen suorittimen ohjaamiseen sekä esimerkiksi kirjoittaa käyttäjälle tekstikäyttöliittymä, jolla ohjata esimerkiksi eri suotimien valintaa. FPGA-SoC:n ohjelmointi ja ajo suoritetaan viimeisenä toimenpiteenä SDK:lla. Ajon aikana voidaan tätä tekstikäyttöliittymää ohjata SDK:n avulla halutessa ilman kuvadatan näytölle syötön keskeyttämistä. Koska Vivado määrittää mitä komponentteja tarvitsee muistiin sekä kirjoittaa VHDL –ohjelman, käyttäjän työmäärä vähenee.



Kuva 10. Laatikkokaavio ohjelman luomisen etenemisestä.

3.8 Asennus FGPA:lle

SDK:lla kirjoitetun C++ –ohjelman valmistumisen jälkeen suoritetaan asennus FPGA-SoC –laitteelle. Tämän suorittamiseen käytetään Vivado SDK –ohjelmaa, jolla asentaminen FPGA-SoC:lle on tehty valmiiksi operaatioksi, jolla USB-UART –siltaa käyttäen SDK ohjelmoi käyttäjän PC:lta suoraan tavallisen USB –väylän kautta FPGA-SoC:lle. Vivado ilmoittaa raportissaan, jotta virrantarve ylittää 0,5A, tulee FPGA-SoC:n tehonlähde olla ulkoinen 5V tasajännite hakkuritehonlähde, joka kykenee tuottamaan enemmän (Digilent 2020). Tämä johtuu siitä syystä, ettei käyttäjän PC:ssä oleva USB2.0 –portti pysty kuin 2,5W ulostuloon 5V jännitteellä, joten maksimivirta täten on 0,5A (Cadence PCB Solutions, 2021). Tasajänniteteholähteen kytkentään on valittuna 5V/3A/15W –ominaisuudet valittu NPS-hakkuriteholähde DC-2.1P –pistokkeella, joka löytyy myös Zybo Z7-10 –laitteelta. FPGA-SoC:lla tulee vaihtaa voimalähdevalitsin USB:n pinniltä

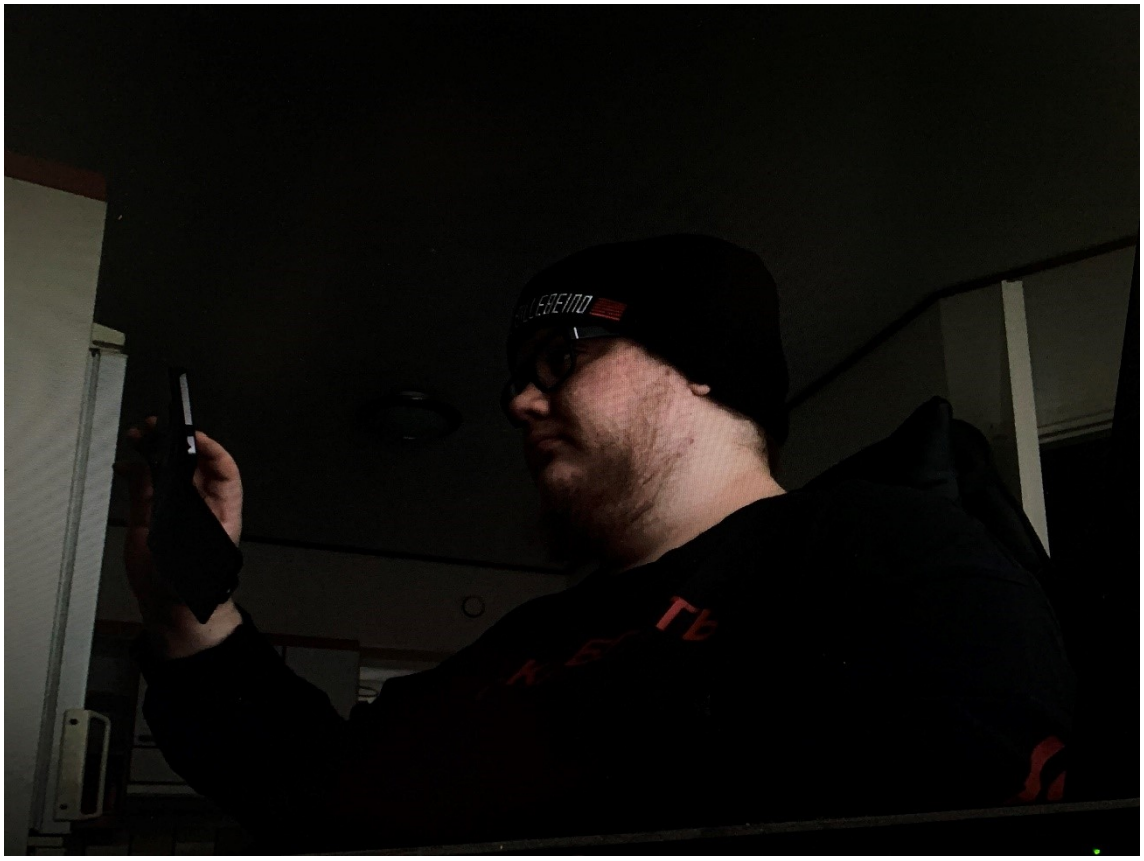
pinniin WALL siirtämällä jumpperia (JP7), jolloin FPGA-SoC ottaa tarvitsevansa tehon seinäpistokkeeseen asennetulta teholähteeltä eikä USB –väylältä, jolla se on yhdistettynä tietokoneeseen (Digilent, 2016:4-5).

Vivadolla FPGA-SoC:n koodatun ohjelman testaamiseksi käyttäjän PC:n avulla tulee myös siirtää toinen hyppyjohdin (JP5) vakioasetuksesta, missä se yhdistää QSPI Boot –liitinnastat, joka valitsee muistiin tehtaalla asennetun demon, joka näyttää muuttuvaa värikuvaa niin, että hyppyjohdin on JTAG Boot –asennossa mikä mahdollistaa ohjelmoimisen suoraan USB–väylän kautta. Zybo Z7-10 sisältämä micro-USB –portti sisältää UART sekä JTAG toiminnallisuudet, joiden avulla käyttäjän PC –tietokone hoitaa kommunikoinnin FPGA-SoC –laitteen kanssa luodakseen itsenäiset keskusteluyhteydet. UART –kanavalla mahdollistetaan keskustelu FPGA-SoC:n kanssa Windows –käyttöjärjestelmän COM -porttien kautta. Valmistaja on asettanut tälle yhteydenpidolle vakioparametrit: 115200 siirtonopeus, 1 lopetusbitti, ei pariteettibittiä, 8 bitin merkkijonolla. Tällä tavalla ajettuna kyetään ohjelmaa ajamaan suoraan Vivadon SDK:n avulla, vaikka rivi kerrallaan sekä suorittaa debuggaamista ja ajaa nopeasti uutta ohjelmaversiota sisään FPGA-SoC:lle tai antaa ohjelman suorittaa itseään saatuaan ohjelmiston käyttäjän tietokoneelta. (Digilent, 2016:3-13).

4 TESTAUS JA TULOKSET

Testaaminen suoritetaan vertailemalla kahta erilaista videokuvaa, jotka tulevat liukuhihnasta läpi. Ensimmäinen on suoraan sensorilta saatava videokuva ja toisena videokuvana toimii kahdella suotimella käsitelty kuvadata, jotta nähdään, onko sensorilta tulevan kuvadatan muokkaaminen tarpeen ja mitä eroa näillä kahdella videokuvalla on. Suotimina toimivat gammakorjaaminen sekä CFA, joilla yhdellä sensorilla varustettu järjestelmä saa parannettua värikylläisyyttään sekä valoisuuden astetta kuvan eri osissa. (Bae, 2020:1-3).

Kuvadatan käsittelyn edellyttämiseksi tulee aiheelliseksi myös varmistaa riittävätkö FPGA:n komponenttien muisti alkuvarauksen yhteydessä, kun livekuvan käsittely pitää sisällään useita erilaisia suotimia. Taulukossa 3 on esitettynä Vivadon Desing Suiten tarjoama raportti FPGA-SoC:n sisään– ja ulostuloväylien käytöstä, sekä kuinka paljon eri muistikomponentteja sekä hakutauluja on käytössä. Jos tämän raportin arvot ylittävät 100%, ei rakennetta voidaan suorittaa ja kuten taulukosta näemme jää käyttöaste näistä komponenteista jää kaikissa alle 45%. Koska esikäsittelyssä on jo käytössä kaksi suodinta, kuvassa 11 esimerkkinä käsittelemätön videostriimi, johon gammakorjaaminen sekä CFA bayer suoritetaan mistä tulos on nähtävissä kuvassa 12, voidaan todeta, että jäljellä on vielä mahdollisuus lisätä useampi kevyt suodin ennen kuin raja tulee vastaan. Ilman tätä käsittelyä syötetään käsittelemätöntä RAW –videokuvaa näytölle mikä on hyvin tummaa, vaikka sisältää kaiken mahdollisen tiedon kameralta ja tarkemmalla tarkastelulla nähdään, että yksityiskohdat ovat kunnossa, mutta ilman gammakorjaamista kuva on tumma.



Kuva 11. Esikäsittelemätön RAW –videokuva.

Taulukko 3. Resurssien käyttöaste ZYBO Z7-10 laitteella ajon aikana.

Komponentti	Käytössä	Käytettävissä	Käyttöaste-%
LUT	7768	17600	44,14
LUTRAM	291	6000	4,85
FF	10707	35200	30,42
IO	23	100	23,00
BRAM	10,50	60	17,50
BUFG	4	32	12,50
MMCM	2	2	100

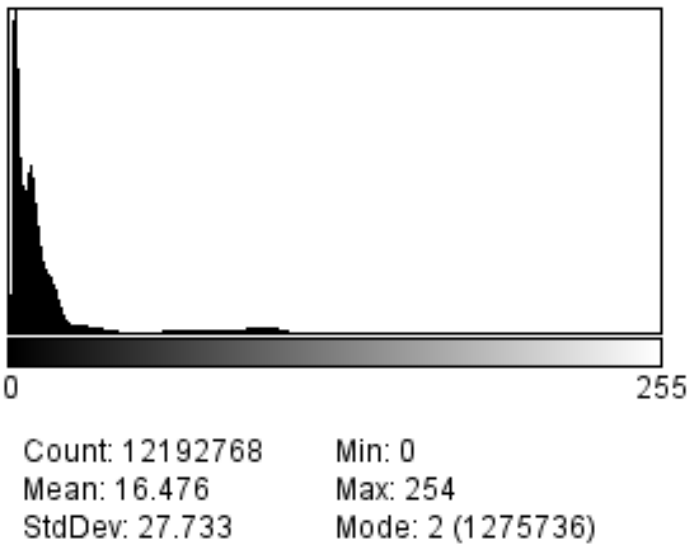
Kun asennus FPGA-SoC:lle on suoritettuna, voidaan FPGA-SoC kytkeä HDMI –kaapelilla näyttöön, jotta näemme, toimiiko ohjelma ja että kuvadata siirtyy kameralta FPGA-

SoC:n kautta suoraan näytölle ja näyttää esikäsittelyn jälkeen halutulta. Kuvassa 12 on käyttäjän ottama kuva kesken suorituksen näyttölaitteelta ja kuten kuvasta näemme, on testauslaitteena oleva kehityslauta siirtänyt kameran tuottaman kuvadatan liukuhihnansa läpi ja tuottanut sen näytölle. Kuvasta käyttäjä näkee, ettei siinä ole merkittäviä väri- tai kontrastivirheitä, joten ohjelman ajon voi tulkita olevan onnistunut sekä implementoitujen kahden suotimen olleen hyödyllisiä, koska kuva on paljon käyttäjäystävällisempi. Kuvadataan laitettiin lisää valoa gammakorjaamalla sekä eheytettiin CFA:n avulla värikarttaa, kun valon määrä kasvaa, tästä menetetään hieman yksityiskohtia, mutta kuvanlaatu on yleistasoltaan hyvä. Tästä voimme päätyä johtopäätökseen, että myös liukuhihna on paikallaan onnistuneesti, kun kuvadata virtaa FPGA-SoC:n läpi tahdotulla tavalla eikä käsittelemättömänä RAW –videokuvana, jolle ei ole suoritettuna mitään alustavia toimia kuten kuvassa 11.

Kuva 12. Käyttäjä vilkuttaa kameralle sekä nappaa kuvan näytöllä pyörivästä videokuvasta.

4.1 Videokuvan muutoksen tarkastelu mittareilla

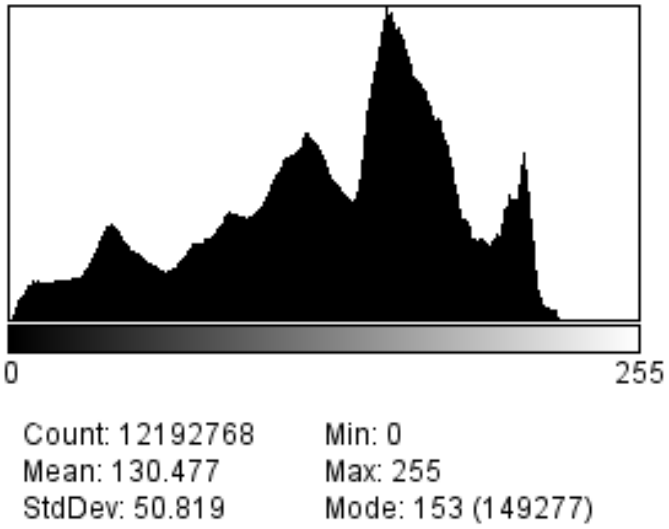
Videokuvien vertailun avuksi otetaan histogrammi sekä RGB -arvojen hakutaulukko, jotta näemme muutoksen visuaalisilla kaavioilla. Kaavioita ja taulukoita voidaan tuottaa esimerkiksi ImageJ –ohjelmalla. Digikuvan artikkelin, 2018, mukaan histogrammin avulla voidaan tarkastaa, onko kuvan valottaminen onnistunut. Histogrammissa on x-akselilla eri valotuksen arvot, jossa suurempi luku tarkoittaa vaaleampaa ja maksimi arvona on täysin valkoinen ja y-akselilla on valotusarvon volyymi ja tavoitteena on saada huippupiikki sijoittumaan mahdollisimman keskelle ns. normaali päiväolosuhteissa, ja massan sijoittumista laidoille kutsutaan ali- sekä ylivalottuneeksi histogrammiksi. Kuvassa 13 sekä 14 esitellään histogrammeja käsittelemättömälle sekä käsitellylle videokuvalle ja niistä voidaan tehdä heti havainto, että käsittelemätön RAW –muotoisena tuleva videokuva sensorilta on todella alivalottunut mikä on siitä hyvin nähtävissä.



Kuva 13. Käsittelemättömän videokuvan histogrammi.

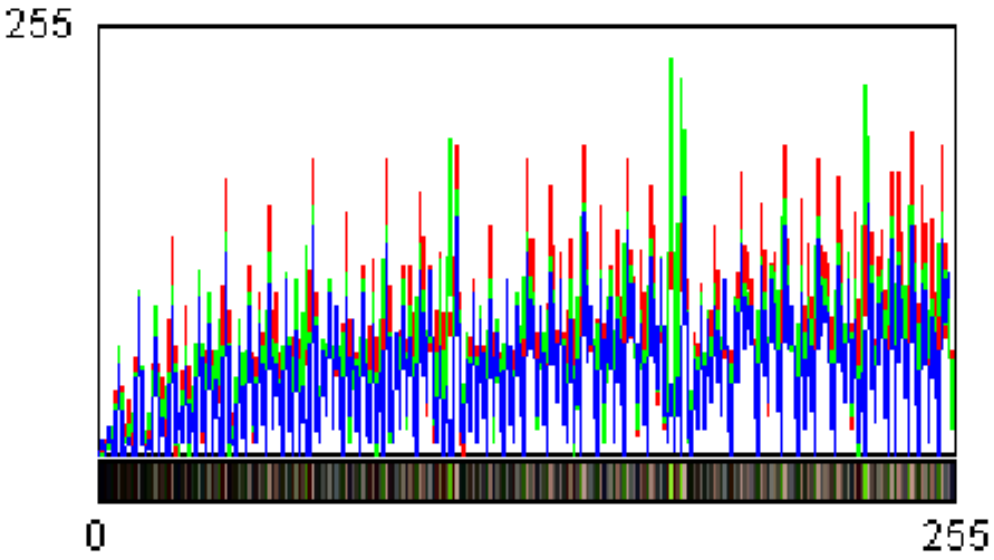
Gammakorjaamalla videokuvaa saadaan histogrammin arvot levittäytymään suuremmalle arvovälille. Nämä videokuvan kaappaukset on otettu mahdollisimman samankaltaisista luonnon valaistusolosuhteista, jotta lisävalaistus ei vaikuttaisi liikaa tulkintaan. Kasvattamalla gammakorjauksen suuruutta, huomataan kuvasta 14 arvojen hajautuminen

akselille tasaisemmin ja nyt korkein piikki asettuu lähemmäs akselin keskikohtaa. Digikuvan artikkelin mukaan tämä on tavoiteltava tila tuloskuvalle ja valotuksen voidaan tulkita olevan onnistunut. Gammakorjaamisen huono puoli on artikkelin mukaan, kuinka sitä joudutaan muuttamaan valotusolosuhteiden muuttuessa ja tulee ottaa huomioon mitkä ovat olosuhteet kuvaushetkellä, jotta histogrammin tulkinta tehdään oikein. Siinä tapauksessa, jos videokuvan kuvaaminen tapahtuisi jatkuvasti ympärivuorokautisesti tulisi huomioida yön ja keskipäivän vaikutukset tarvittavaan valotuksen korjaamiseen, jotta videokuva ei muutu tunnistamattomaksi.



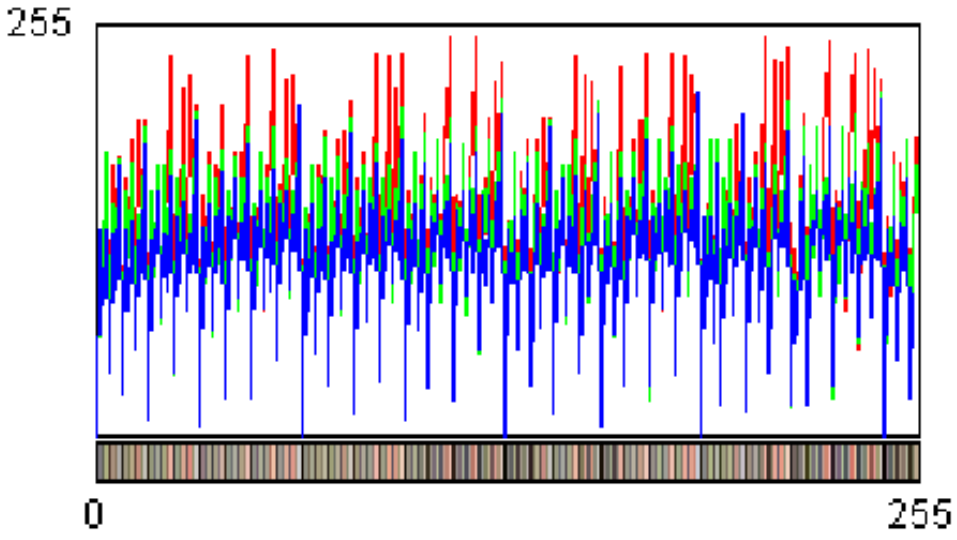
Kuva 14. Käsitellyn videokuvan histogrammi.

Toiseksi vertailukohteeksi on otettu RGB –arvojen hakutaulu, jossa näemme demosaicing –prosessin vaikutuksen värimaailmaan. Kuvassa 15 on taulu muokkaamattomasta kuvadatasta, jossa näyttää osan värimaailmasta olevan kadoksissa, koska käyrien y-akseli arvot ovat painottuneet hyvin alas ja videokuvasta tuntuu ns. puuttuvan värikylläisyyttä, kuten myös kuvasta 11 on nähtävissä.



Kuva 15. RGB –arvojen hakutaulu muokkaamattomalle kuvadatalle.

Seuraavaksi tarkastellaan kuvaa 16 minkä hakutaulu antaa käsitellyn kuvadatan vastaavat arvot. Tässä bayer filtterin käsittelyn jälkeen on ns. rekonstruktioitu värimaailmaa ja nyt on nähtävissä pidemmät hännät sekä huiput kolmelle eri värille ja keskiarvo sijoittuu lähemmäs arvotaulukon keskustaa eikä alareunaa. Tätä tukee myös kuva 12 esimerkki, jossa laajempi väriskaala on silmin havaittavissa.



Kuva 16. RGB –arvojen hakutaulu muokatulle kuvadatalle.

Näiden taulukoiden sekä esimerkkikuvien avulla voidaan havaita, että testausolosuhteissa käytetyt filtterit tulivat tarpeeseen ja kuvateknisesti paransivat kuvan laatua ja ymmärrettävyyttä, kun valoisuus jakautui tasaisemmin sekä väriskaala oli laajempi kuin pelkällä RAW –muotoisella kuvadatalla, mikä sisälsi kaiken kameran kaappaaman tiedon ja tarkimmat yksityiskohdat, mutta oli vaikeatulkintainen alivalotuksen takia.

4.2 Parannusideoita

Tähän pohjaan olisi mahdollista käyttäjän lisätä mahdollisia suotimia, kuten mustavalkosuotimia sekä reunahavainnointia. Tämä vaatii Vivadolla liukuhihnan rakenteen muuttamista, jolloin ensin valitaan mihin väliin halutut suotimet sijoitetaan, jotta matriisien sisältämä kuvadata on ensin käsitelty helposti muokattavaan RGB –matriisi muotoon ennen suotimia, joten Switch –kytkimien väli on luonnollinen vaihtoehto sekä tarvittavien muistitietoväylien muodostaminen. Näiden toiminnallisuuksien suorittaminen JTAG –väylän avulla mahdollistaa, että FPGA-SoC voi olla kytkettynä koko ajan käyttäjän tietokoneessa, joten uudelleenohjelmointi sekä käyttöönotto voidaan suorittaa, kun ohjelma on päivitetty ja tarkastettu sekä uudelleen alustettu, jos on otettu käyttöön uusia lohkoja.

Koska Vivado hoitaa muistin jakamisen, voidaan olettaa, että algoritmissa on parantamisen varaa optimoimalla esimerkiksi muistinkäyttöä. Garcia ja muut, 2019 (s. 4-11), ovat tutkineet artikkelissaan, kuinka Vivadon määrittelemää muistilohkojen jakotapaa voidaan tehostaa muistinjakamisalgoritmeilla. Heidän tapauksessaan saavutettiin 30% parannuksen tallentamisen tehokkuudessa, jolloin vapaita muistilohkoja jäi enemmän käyttäjälle käytettäväksi. Koska muistinkäytön optimointi tästä työstä puuttuu, olisi se varteenotettava kehitysmahdollisuus, kun tätä algoritmia ja liukuhihnaa ruvetaan monipuolistamineen ja kasvattamaan, jotteivat laitteen muistinhallintarajat tule välittömästi vastaan.

Lisäksi tässä työssä ei olla keskitytty energiatehokkuuden maksimointiin, mikä on noussut yhdeksi pääteemaksi 2020-luvulla ilmastonmuutoksen kiihtyessä sekä yritysten sekä yksityishenkilöiden pyrkiessä pienentämään energiankäyttökulujaan vähentääkseen maksu- sekä ilmastotaakkaansa (Price ja muut; 2010:17-18), mutta rinnakkainen laskenta

tuottaa itsessään energiatehokkuuden kasvua. Edellisessä kappaleessa esitelty muistinhallinnan tehostaminen tuo myös säästöä energiankulutukseen, kun vähemmän muistilohkoja tarvitaan ohjelman suoritukseen. Garcia ja muut, 2019 (s.12-16), suorittivat algoritminsa uudelleen muistinhallinnan tehostamisen jälkeen nähdäkseen, miten se vaikutti kulutukseen ja saavuttivat HD–kuvalaatuisella kuvalla $0.219W \div 0.085W \times 100\% = 38.81\%$ säästön energiankulutukseen. Tietysti tämä luku ei ole suoraan johdannollinen ja verrattavissa mitä voitaisiin saavuttaa myös muissa algoritmeissa, mutta antaa suuntaa siitä, ettei Vivado:n automaattisesti ohjaama rakenne ole optimaalisin tai energiatehokkain ratkaisu ja tähän voisi panostaa tulevaisuudessa.

Siqqiqui ja muut, 2019 (s.8-12), käyvät artikkelissaan läpi, kuinka ottamalla käyttöön FPGA-SoC:n ARM:n lisäksi ohjelmoimalla FPGA:n ohjelmoitavaa logiikkaa ja luomalla niiden avulla pehmeitä suorittimia, jotka ovat mikroprosessoreita, joilla on valmiiksi ohjelmoitu tehtävä. Näillä lisäytimillä pyritään hakemaan liukuhihnan eri osien suorittamisen kiihdyttämistä, jos digitaaliseen signaalinkäsittely prosessiin on sisällytetty paljon työtä vaativia prosesseja. Työssään he saivat näillä pehmeillä suorittimilla omassa algoritmissään parhaimmillaan 50% säästön suhteutettuna, jos käytössä olisi vain yksi suorittava ydin. Vertailukohtana heillä toimi Vivado:n suorittama kuvadatan oletuskulkutapa, joka on tässä työssä oletusarvoisena kulkuväylänä, niin yhdistämällä edellisessä kappaleessa esitettyjä muistinhallintaa sekä prosessin kiihdyttämistä pehmeillä ytimillä voisi tätäk työtä tulevaisuudessa lähteä optimoimaan energiaa säästävämpään suuntaan.

5 POHDINTA SEKÄ YHTEENVETO

Työssä esitetty C++ –toteutus on toimiva ratkaisu, jolla saadaan kuvadataa käsiteltyä erikseen aseteltavilla suotimilla käyttäjän tahdon mukaan ohjelmoimalla ja tarkastelemalla liukuhihnaa. Liukuhihna on osa kuvadatan käsittelyä, minkä rakenne vaikuttaa liukuhihnan tehokkuuteen. Lisäksi valitut kaksi suodinta paransivat histogrammin sekä väritasapainon mukaan kuvanlaatua, jolloin niiden sijoittaminen liukuhihnan osaksi on perusteltua. Vivado Design Suite –ohjelmisto sopii nopeaan testaamiseen ja opiskelijatason ohjelmoimiselle, koska Xillinx tarjoaa videoita sekä harjoittelumalleja, joita seuraamalla käydään ohjelman käyttöä perusteellisesti. Tosin myös tässä ohjelmointialustassa on ongelmia, kuten Design Suite itse varoittaa kuinka ohjelman vuosittain julkaistavat pääversiot eroavat merkittävästi edellisen vuoden ohjelmaversiosta, joten vanhalla versiolla tehdyt FPGA-SoC ohjelmat ja designit eivät välttämättä toimi yksi yhteen uudemman version sekä erilaisten laitteiden kanssa.

Yhtenä vaihtoehtona esiteltiin C –kielinen ohjelma, jonka voi suorittaa myös suoraan suorittimen avulla sekä muutoksin myös FPGA-SoC:n mikroprosessorin avulla käsin asennettuna, mutta käytännöllisyydessä se häviää edellisessä kappaleessa esitetylle ratkaisulle, koska tässä ratkaisussa suoritettaisiin lohkorakenteilla saatavaa hyötyä vasta CPU:lla, mikä taas ei hyödynnä FPGA-SoC:n mahdollisuuksia, kun lohkojen välisen liikenteen sijaan kuljettaisiin tietoa CPU:lta FPGA:lle useamman kerran. Design Suite:n luo kaiken käyttäjälle valmiiksi kuten kytkennät lohkojen välillä sekä niiden sekvensoinnin sekä kirjoittaa VHDL –ohjelman valmiiksi, jolloin käyttäjä pääsee nopeammin keskittymään mahdollisiin suotimiin sekä näkee mitä rakenteita ohjelman suorittamiseen tarvitaan. Lisäksi prosessin suunnittelun liukuhihnoitus –ajattelua käyttäen sekä kehityslaudan valmistajan IP –lohkojen tarjontaa hyväksikäyttäen auttaa valitsemaan mitä käyttäjä voi käyttää eri prosessien suorittamiseen, jottei niitä tarvitse erikseen lähteä luomaan alusta asti, joten käyttäjä pääsee nopeammin keskittymään main –metodin luomiseen ja millaista ohjelmaa hän FPGA-SoC –piirillä suorittaa, jolloin käyttäjä voisi lennosta vaihtaa käskyillä mitä suodinrakenteita haluaisi ottaa käyttöön.

5.1 Johtopäätökset

Johdannossa esitettiin kolme kysymystä kuvadatan käsittelyyn sekä FPGA-SoC liukuhihnoitukseen ja onko tämä perusteltua, joihin toivottiin saatavan vastaus. Näihin kysymyksiin on mielestäni vastattu työn aikana ja tässä esitetään vielä kootusti lyhyesti vastaus tutkimuskysymyksiin sekä millainen johtopäätös siitä on vedetty.

- Miksi kuvadataa kannattaa käsitellä FPGA-SoC:lla?

Kuvadataa kyetään syöttämään FPGA-SoC –piirin käsiteltäväksi sen monipuolisen lohkorakenteen sekä I/O –väylien monipuolisuuden ansiosta erilaisilta kameralaitteilta. Kuvadatan käsittely on tehokasta, kun se suoritetaan rinnakkaisesti FPGA-SoC:lla ja ARM –suorittimen tehtäväksi jätetään suotimien valinta, jolloin aikaa ei hukata syöttelemällä kuvadataa edestakaisin mikroprosessorin ja FPGA:n välillä eikä ohjelmisto koodiarkkitehtuuri kasva liian monimutkaisiksi sisäisten silmukoiden solmuiksi.

- Miten gammakorjaaminen ja bayer-filtteri vaikuttaa kuvanlaatuun?

Tuloksissa käytiin läpi histogrammin sekä väritasapainotaulukon avulla läpi, miten nämä kaksi filtteriä vaikuttivat videokuvaan. Muutos oli silminnähtävä sekä taulukoin todistettu. Histogrammi näytti valotuksen tasapainottuneen sekä värimaailman olleen tasapainoisempi sekä sisältävän laajemman skaalan värialueilta.

- Onko kuvankäsittely FPGA-SoC:lla perusteltua?

Se on perusteltua, koska rinnakkaisuuden avulla saavutetaan energiatehokkuuden kasvua minkä saavuttaminen tietokoneilla vaatii monimutkaisempia ja tehokkaampia prosessoreita, kun eri vaiheita ei kyetä ajamaan rinnakkaisesti yhtä massiivisesti kuin FPGA-SoC –järjestelmäpiirillä pystytään.

Työ oli haastava kokonaisuus, joka vaati paljon lisää vapaaehtoista lisäopiskelua, jotta tekijä oppi käyttämään Vivadon ohjelmistoja ja kuinka kameralta tuleva kuvadata saa-

daan visuaalisesti paremmaksi. Katson tavoitteen tulleen saavutetuksi ja lopullisen johtopäätöksen olevan, että FPGA-SoC tulee nostamaan päätään tulevaisuudessa myös kuvadatan sekä signaalien käsittelyssä ja niiden opiskeluun sekä tutkimiseen pitää käyttää enemmän ja enemmän voimavaroja opiskelijoiden kurssitasolta lähtien, koska FPGA-SoC –laitteiden hallinnalla voi olla merkittäviä vaikutuksia työllistymiseen, jos markkinaosuudet kasvavat ja FPGA-SoC murtautuu käytännölliseksi vaihtoehdoksi uusille teollisuuden aloille sekä kasvattaa rooliaan tietoliikenneverkkojemme infrastruktuurissa.

Järkevällä liukuhihnan rakentamisella saadaan tehokkuutta ja sekä etua perinteiseen tietokoneeseen verrattuna, kun rinnakkaisuutta saadaan kasvatettua. Suunnittelusovellusten kehittyessä ja monipuolistuessa saadaan lohkorakenteita luotua nopeammin, jolloin niiden kehittämiseen käytettävinä olleita voimavaroja vapautuu muuhun ohjelmistokehittämiseen sen sijaan, että käsin kirjoitettaisiin eri lohkojen yhteen kytkennät sekä ulos- ja sisääntulosignaalit. Tämän työn laite sekä sen ohjelmoimiseen käytettävä ohjelmistoalusta olisi hyvä alusta opiskelijoille pehmeämpään alkuun FPGA-SoC –laitteiden kanssa, kun rautatason ohjelmointia näkee ensin käytännössä mikä voi helpottaa asioiden yhdistämistä ajatustasolla FPGA-SoC:n rautatason perusasioiden oppimisen jälkeen opiskelun alkuvaiheilla.

LÄHDELUETTELO

Acasandrei, Laurentiu; Barriga, Angel; 2015. *Open Library of IP Module Interfaces for AMBA Bus.* DOI: https://doi.org/10.1142/9789813142725-0022.

awong, käyttäjä RS Components yhtiön DesingSpark –palvelussa. *Getting Started with Xilinx Zynq, All Programmable System-On-Chip (SoC)* (6/2017). Verkkoartikkeli noudettu: https://www.rs-online.com/designspark/getting-started-with-xilinx-zynq-all-programmable-soc.

Bae, Tae Wuk, 2020. *Image-quality metric system for color filter array evaluation*. DOI: https://doi.org/10.1371/journal.pone.0232583.

Bailey, Donald G; 2019. *Image Processing Using FPGAs.* Doi: https://doi.org/10.3390/jimaging5050053.

Cadence PCB Solutions. 2021. *What are the Maximum Power Output and Data Transfer Rates for the USB Standards?* Noudettu: https://resources.pcb.cadence.com/blog/2020-what-are-the-maximum-power-output-and-data-transfer-rates-for-the-usb-standards.

Cromey, Douglas W.; 24.8.2012. *Digital Images Are Data: And Should Be Treated as Such.* Noudettu: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4210356/. Doi: https://link.springer.com/protocol/10.1007%2F978-1-62703-056-4_1.

Dennis M. Ritchie. 1993. *The development of the C language*. Noudettu https://dl.acm.org/doi/10.1145/155360.155580.

Digikuva, 2018. *Näin toimii histogrammi.* Noudettu: https://digi-kuva.fi/valokuvaustekniikka/histogrammit/nain-toimii-histogrammi.

Digilent. 2020. *Installing Vivado (Legacy)*. Noudettu: https://reference.digilentinc.com/learn/software/tutorials/vivado-install-guide/start?redirect=1.

Digilent. 2020. *ZYBO Z7*. Noudettu: https://reference.digilentinc.com/reference/programmable-logic/ZYBO-z7/start.

Franz, Kaitlyn; Digilent 2020. *History of the FPGA*. Noudettu https://blog.digilentinc.com/history-of-the-fpga/.

Garcia, Paula; Bhowmik, Deepayan; Stewart, Robert; Michaelson, Greg; Wallace, Andrew; 2019. *Optimized Memory Allocation and Power Minimization for FPGA-based Image Processing*. DOI: https://doi.org/10.3390/jimaging5010007.

Grand View Research. 5.2020. *Microprocessor Market Size, Share & Trends Analysis Report By Technology (CISC, RISC, Superscalar, DSP), By Application, By Region, And Segment Forecasts, 2020-2027*. Noudettu: https://www.grandviewresearch.com/industry-analysis/microprocessor-market.

Haines, Richard F.; Chuang, Sherry L.; 1.7.1992. *The Effects of Video Compression on Acceptability of Images for Monitoring Life Sciences Experiments (NASA Technical Paper 3239)*. Noudettu: https://ntrs.nasa.gov/api/citations/19920024689/downloads/19920024689.pdf.

Harvey, A. F. ja henkilökunta; 1991. *DMA Fundamentals on Various PC Platforms.* Noudettu: http://cires1.colorado.edu/jimenez-group/QAMSResources/Docs/DMAFundamentals.pdf.

Hegarty, James; Brunhaver, John; DeVito, Zachary; Ragan-Kelley, Jonathan; Cohen, Noy; Bell, Steven; Vasilyev, Artem; Horowitz, Mark; Hanrahan, Pat. 2014. *Darkroom: Compiling High-Level Image Processing Code into Hardware Pi*pelines. Doi: https://doi.org/10.1145/2601097.2601174.

Intel, Adobe; 2015. *Hardware Performance Guide: Serious 4K Editing.* Noudettu: https://www.intel.com/content/dam/www/public/us/en/documents/guides/workstation-adobe-4k-guide.pdf.

Intel. 28.12.2015. *Intel Acquisition of Altera*. Noudettu: https://simplecore.intel.com/newsroom/wp-content/uploads/sites/11/2016/03/Intel-to-Acquire-Altera-Press-Release.pdf.

Jiang Haomiao; Tian, Qiyuan; Farrell, Joyce; Wandell, Brian. 2016. *Learning the image processing pipeline.* Noudettu: https://arxiv.org/pdf/1605.09336.pdf.

Martin, Milo. 4.2012, UPenn. *History of Processor Performance*. Noudettu: http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf.

MIPI CAMERA SERIAL INTERFACE 2. MIPI Alliance. 2021. CSI-2 portin kehittäjän kotisivut. Noudettu: https://www.mipi.org/specifications/csi-2.

Murray, James D.; VanRype, William. 1996. *Encyclopedia of graphics file formats*. Julkaisija: Bonn ; Sebastapol, CA : O'Reilly & Associates. Noudettu: https://archive.org/details/mac_Graphics_File_Formats_Second_Edition_1996/page/n47/mode/2up?q=bmp.

Price, Lynn; Wang, Xuejun; Yun, Jiang; 2010. *The Challenge of Reducing Energy Consumption of the Top-1000 Largest Industrial Enterprises in China.* Noudettu: https://china.lbl.gov/sites/all/files/ep-top1000-challengenov-2010.pdf.

Ranta, Jukka. 2012. *The current state of FPGA technology in the nuclear domain*. Noudettu https://www.vttresearch.com/sites/default/files/pdf/technology/2012/T10.pdf.

Sharif, S. M. A; Naqvi, Rizwan Ali; Biswas, Mithun. 2021. *Beyond Joint Demoisaicking and Denoising: An Image Processing Pipeline for a Pixel-bin Image Sensor.* Noudettu: https://arxiv.org/pdf/2104.09398.pdf.

Siddiqui, Fahad; Amiri, Sam; Minhas, Umar Ibrahim; Deng, Tiantai; Woods, Roger; Rafferty, Karen; Crookes, Daniel; 2019. *FPGA-Based Processor Acceleration for Image Processing Applications*. DOI: https://doi.org/10.3390/jimaging5010016.

Treece, Brandon; 2017. *CPU or FPGA for image processing: Which is best?* Noudettu: https://www.vision-systems.com/embedded/article/16737656/cpu-or-fpga-for-image-processing-which-is-best.

Trimberge, Stephen M.; 2015. *Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology*. Noudettu https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7086413.

Wang, Heng; Chen, Xinrui; 2019. *Development and Optimization Design of Digital Logic device based on FPGA.* Doi: doi:10.1088/1742-6596/1345/6/062051.

Vivado Desing Suite. Xilinx, Noudettu: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html

Zynq Architecture. Xilinx, 2012. Xilinxin arkkitehtuuri powerpoint –esitelmä. Noudettu: http://www.ioe.nchu.edu.tw/Pic/CourseItem/4468_20_Zynq_Architecture.pdf.

LIITEET

Liite 1. Osa YLÖSAJO –ohjelmasta

```
#define IRPT_CTL_DEVID  XPAR_PS7_SCUGIC_0_DEVICE_ID
#define GPIO_DEVID  XPAR_PS7_GPIO_0_DEVICE_ID
#define GPIO_IRPT_ID    XPAR_PS7_GPIO_0_INTR
#define CAM_I2C_DEVID   XPAR_PS7_I2C_0_DEVICE_ID
#define CAM_I2C_IRPT_ID XPAR_PS7_I2C_0_INTR
#define VDMA_DEVID  XPAR_AXIVDMA_0_DEVICE_ID
#define VDMA_MM2S_IRPT_ID   XPAR_FABRIC_AXI_VDMA_0_MM2S_INTROUT_INTR
#define VDMA_S2MM_IRPT_ID   XPAR_FABRIC_AXI_VDMA_0_S2MM_INTROUT_INTR
#define CAM_I2C_SCLK_RATE   100000

#define DDR_BASE_ADDR   XPAR_DDR_MEM_BASEADDR
#define MEM_BASE_ADDR   (DDR_BASE_ADDR + 0x0A000000)

#define GAMMA_BASE_ADDR XPAR_AXI_GAMMACORRECTION_0_BASEADDR

void pipeline_mode(AXI_VDMA<ScuGicInterruptController>&
vdma_driver, KAMERA& cam, VideoOutput& vid, Resolution res,
KAMERA_cfg::mode_t mode){
//inputin liukuhihnan väylän muodostaminen
{
vdma_driver.resetWrite();
MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEADDR,
CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEADDR,
CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
cam.reset();
}

{
vdma_driver.configureWrite(timing[static_cast<int>(res)].h_active,
timing[static_cast<int>(res)].v_active);
Xil_Out32(GAMMA_BASE_ADDR, 3); // Gammakorjaaminen
cam.init();
}

{
vdma_driver.enableWrite();
MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEADDR,
CR_OFFSET, CR_ENABLE_MASK);
MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEADDR,
CR_OFFSET, CR_ENABLE_MASK);
cam.set_mode(mode);
cam.set_awb(KAMERA_cfg::awb_t::AWB_ADVANCED);
}

//outputin liukuhihnan väylän muodostaminen
{
vid.reset();
```

```
vdma_driver.resetRead();
}

{
vid.configure(res);
vdma_driver.configureRead(timing[static_cast<int>(res)].h_active,
timing[static_cast<int>(res)].v_active);
}

{
vid.enable();
vdma_driver.enableRead();
}}

int main(){
init_platform();

ScuGicInterruptController irpt_ctl(IRPT_CTL_DEVID);
PS_GPIO<ScuGicInterruptController> gpio_driver(GPIO_DEVID, irpt_ctl,
GPIO_IRPT_ID);
PS_IIC<ScuGicInterruptController> iic_driver(CAM_I2C_DEVID, irpt_ctl,
CAM_I2C_IRPT_ID, 100000);

SWITCH_CTL axis_switch_ctl(src_switch, dst_switch, sw_gpio,
XPAR_AXIS_SWITCH_0_NUM_MI, 0);

KAMERA cam(iic_driver, gpio_driver);
AXI_VDMA<ScuGicInterruptController> vdma_driver(VDMA_DEVID,
MEM_BASE_ADDR, irpt_ctl,
VDMA_MM2S_IRPT_ID,VDMA_S2MM_IRPT_ID);
VideoOutput vid(XPAR_VTC_0_DEVICE_ID, XPAR_VIDEO_DYNCLK_DEVICE_ID);

pipeline_mode(vdma_driver, cam, vid, Resolution::R1920_1080_60_PP,
KAMERA_cfg::mode_t::MODE_1080P_1920_1080_30fps);

xil_printf("Video init done.\r\n");}
```

Liite 2 . Osa FILTTERIAJO –ohjelmasta

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

// Headerin rakenne
typedef struct
{
    unsigned short int Type;
    unsigned int Size;
    unsigned short int Reserved1, Reserved2;
    unsigned int Offset;
}HEADER;
```

```
// Infoheaderin rakenne
typedef struct
{
    unsigned int Size;
    int Width, Height;
    unsigned short int Planes;
    unsigned short int Bytes;
    unsigned int Compression;
    unsigned int ImageSize;
    int xResolution, yResolution;
    unsigned int Colors;
    unsigned int ImportantColors;
}INFOHEADER;

// Pikselien rakenne
typedef struct
{
    unsigned char Red, Green, Blue;
}PIXEL;

// Osoittimien määrittäminen
HEADER *pHeader;
INFOHEADER *pInfoHeader;
PIXEL *pPixel, **pImage;

// Muuttujien alustaminen
int i, j;

// Dynaaminen muistinvaraus datan tallentamista ja lukua varten
pHeader=(HEADER*)malloc(sizeof(HEADER));
pInfoHeader=(INFOHEADER*)malloc(sizeof(INFOHEADER ));
pPixel=(PIXEL*)malloc(sizeof(PIXEL));

// Luetaan header-rakenteen tietueiden määrä
fread(&pHeader->Type,sizeof(pHeader->Type),1,inFile);
fread(&pHeader->Size,sizeof(pHeader->Size),1,inFile);
fread(&pHeader->Reserved1,sizeof(pHeader->Reserved1),1,inFile);
fread(&pHeader->Reserved2,sizeof(pHeader->Reserved2),1,inFile);
fread(&pHeader->Offset,sizeof(pHeader->Offset),1,inFile);

// Ulostulotiedosto muutetaan header-rakenteen mukaiseksi
fwrite(&pHeader->Type,sizeof(pHeader->Type),1,outFile);
fwrite(&pHeader->Size,sizeof(pHeader->Size),1,outFile);
fwrite(&pHeader->Reserved1,sizeof(pHeader->Reserved1),1,outFile);
fwrite(&pHeader->Reserved2,sizeof(pHeader->Reserved2),1,outFile);
fwrite(&pHeader->Offset,sizeof(pHeader->Offset),1,outFile);

// Luetaan infoheader-rakenteen tietuiden määrä
fread(&pInfoHeader->Size,sizeof(pInfoHeader->Size), 1,inFile);
fread(&pInfoHeader->Width,sizeof(pInfoHeader->Width),1,inFile);
fread(&pInfoHeader->Height,sizeof(pInfoHeader->Height),1,inFile);
fread(&pInfoHeader->Planes,sizeof(pInfoHeader->Planes),1,inFile);
fread(&pInfoHeader->Bytes,sizeof(pInfoHeader->Bytes),1,inFile);
fread(&pInfoHeader->Compression,sizeof(pInfoHeader->
Compression),1,inFile);
fread(&pInfoHeader->ImageSize,sizeof(pInfoHeader->
ImageSize),1,inFile);
fread(&pInfoHeader->xResolution,sizeof(pInfoHeader->
```

```
xResolution),1,inFile);
fread(&pInfoHeader->yResolution,sizeof(pInfoHeader->
yResolution),1,inFile);
fread(&pInfoHeader->Colors,sizeof(pInfoHeader->Colors),1,inFile);
fread(&pInfoHeader->ImportantColors, sizeof(pInfoHeader->
ImportantColors),1,inFile);

// Ulostulotiedosto muutetaan infoheader-rakenteen mukaiseksi
fwrite(&pInfoHeader->Size,sizeof(pInfoHeader->Size),1,outFile);
fwrite(&pInfoHeader->Width,sizeof(pInfoHeader->Width),1,outFile);
fwrite(&pInfoHeader->Height,sizeof(pInfoHeader->Height),1,outFile);
fwrite(&pInfoHeader->Planes,sizeof(pInfoHeader->Planes),1,outFile);
fwrite(&pInfoHeader->Bytes,sizeof(pInfoHeader->Bytes),1,outFile);
fwrite(&pInfoHeader->Compression,sizeof(pInfoHeader->
Compression),1,outFile);
fwrite(&pInfoHeader->ImageSize,sizeof(pInfoHeader->
ImageSize),1,outFile);
fwrite(&pInfoHeader->xResolution,sizeof(pInfoHeader->
xResolution),1,outFile);
fwrite(&pInfoHeader->yResolution,sizeof(pInfoHeader->
yResolution),1,outFile);
fwrite(&pInfoHeader->Colors,sizeof(pInfoHeader->Colors),1,outFile);
fwrite(&pInfoHeader->ImportantColors,sizeof(pInfoHeader->
ImportantColors),1,outFile);

// Dynaaminen muistinvaraus kaksiulotteista matriisia varten
pImage=(PIXEL**)malloc(sizeof(PIXEL*)*pInfoHeader->Height);
for(i=0;i<pInfoHeader->Height;i++)
{
pImage[i]=(PIXEL*)malloc(sizeof(PIXEL)*pInfoHeader->Width);
}

// Kuvadata matriisiin
for(i=0;i<pInfoHeader->Height;i++)
{
    for(j=0;j<pInfoHeader->Width;j++)
    {
    fread(&pImage[i][j].Red,sizeof(pPixel->Red),1,inFile);
    fread(&pImage[i][j].Green,sizeof(pPixel->Green),1,inFile);
    fread(&pImage[i][j].Blue,sizeof(pPixel->Blue), 1, inFile);
    }
}

// Tiedostojen sulkeminen ja muistinvapautus.
fclose(inFile);
fclose(outFile);
free(pHeader);
free(pInfoHeader);
free(pPixel);
free(pImage);
```