LAPPEENRANNAN TEKNILLINEN YLIOPISTO
SÄHKÖTEKNIIKAN OSASTO

FPGA:lle sulautetulla mikroprosessorilla toteutettu sähkökäytön säätöjärjestelmä

Diplomityön aihe on hyväksytty Lappeenrannan teknillisen yliopiston Sähkötekniikan osaston osastoneuvoston kokouksessa 12.1.2005.

Työn valvoja ja tarkastaja: Professori Olli Pyrhönen
Työn ohjaaja ja tarkastaja: TkT Julius Luukko

Lappeenrannassa 2.5.2005

Aki Penttinen
Punkkerikatu 7 B 22
53850 Lappeenranta
p. +35844-0739102

TIIVISTELMÄ

Tekijä: Penttinen, Aki

Nimi: **FPGA:lle sulautetulla mikroprosessorilla toteutettu sähkökäytön säätöjärjestelmä**

Osasto: Sähkötekniikan osasto

Vuosi: 2005

Paikka: Lappeenranta

Diplomityö. Lappeenrannan teknillinen yliopisto. 81 sivua, 22 kuvaa, 24 taulukkoa ja 2 liitettä.

Tarkastajat: Professori Olli Pyrhönen ja TkT Julius Luukko

Hakusanat: FPGA, sulautettu järjestelmä, MicroBlaze, RTOS, uC/OS-II, sähkökäyttö, taajuudenmuuttaja

Sähkökäyttöjen ohjauselektroniikka koostuu yleensä hyvin monista eri komponenteista, jolloin järjestelmän toimivuutta haittaavat esimerkiksi piirilevyvetoihin kytkeytyvät häiriöt. Tässä diplomityössä selvitetään, kuinka sähkökäyttöjen ohjausjärjestelmä voidaan toteuttaa FPGA-piirillä ja sille sulautetulla mikroprosessorilla. Tällöin koko ohjausjärjestelmä toteutetaan käyttäen vain yhtä mikropiiriä. FPGA-piirit ovat ohjelmoitavia logiikkapiirejä, joiden koko ja nopeus ovat kasvaneet riittävän suuriksi tällaisiin järjestelmiin. FPGA-piirille voidaan toteuttaa sulautettu prosessori kahdella tapaa, laitteistolohkona tai käyttäen piirillä olevaa logiikkaa. Piirillä olevaa logiikkaa käyttäen saadaan monia etuja, kuten järjestelmän helppo päivittäminen ja ominaisuuksien muuttaminen.

Xilinx tarjoaa FPGA-piireilleen ohjelmistopohjaisen MicroBlaze-nimisen prosessoriytimen, joka tehokkuutensa ansiosta soveltuu hyvin käytettäväksi monimutkaisissa järjestelmissä kuten tietoliikenne-, sulautetuissa ja kuluttajamarkkinoiden laitteissa. Jotta prosessoria voitaisiin käyttää mahdollisimman tehokkaasti, tutustutaan myös reaaliaikakäyttöjärjestelmän tarjoamiin etuihin ja sovelletaan tälläistä järjestelmää yksinkertaisen oikosulkumoottorin ohjausjärjestelmän toteuttamiseen.

ABSTRACT

Author: Penttinen, Aki

Subject **Control system for electic drives implemented with FPGA embedded microprocessor**

Department: Electrical Engineering

Year: 2005

Place: Lappeenranta

Master's thesis. Lappeenranta University Of Technology. 81 pages, 22 figures, 24 tables and 2 appendices.

Examiners: Professor Olli Pyrhönen and D.Sc. Julius Luukko

Keywords: FPGA, embedded system, MicroBlaze, RTOS, uC/OS-II, electric drive, frequency converter

Electric drive control electronics are build with many components when at the same time it is exposed to electrical interference. This master's thesis analyzes the possibility to use FPGA's software IP (Intellectual Property) embedded microprocessor (especially Xilinx's Microblaze) in electrical motor control systems. The idea is to implement the controlling electronics into one single chip. FPGA is a field programmable gate array chip which can be programmed many times. Their capacity and speed has increased enough to use in this kind of systems. It is possible to implement a processor core to the FPGA in two different ways. When the processor is fixed into silicon it is called hardware IP-processor and when it is done using FPGA's logical resources it is called software IP-processor. With software IP-processor there are many advantages compared to hardware IP-processors, like easily changeable processor's peripherals and memory, easy to update and upgrade.

Xilinx provides very powerful software IP-processor core called MicroBlaze. It is powerful enough to be used for building complex systems for the networking, telecommunication, data communication, embedded and consumer markets. Processor's resources can be used more efficiently by using realtime operating systems (RTOS). The use of an RTOS with MicroBlaze also been examined. Finally a simple control system prototype are build to drive a electric motor.

Alkusanat

Tämä työ liittyy Tekes:in ÄLY2000 teknologiaohjelmaan kuuluvaan hankkeeseen ”Tehoelektroniikkalaitteen modulaarinen ohjaustekniikka” (TEMO), jonka tavoitteena oli kehittää tehokas logiikkapiireihin (erityisesti FPGA-piireihin) perustuva toteutusmalli tehoelektroniikkalaitteen ohjausjärjestelmälle. Tutkimushanke toteutettiin pääosaltaan Lappeenrannan teknillisen yliopiston (LTY) sähkötekniikan osaston säätötekniikan ja sovelletun elektroniikan laboratorioissa. Hankkeessa yhteistyöyrityksinä oli mukana Kemppi Oy ja Vacon Oyj.

Haluan kiittää työni tarkastajia ja ohjaajia erittäin mielenkiintoisesta aiheesta sekä asiantuntevasta ja kannustavasta ohjaamisesta. Työkavereitani kiitän mielenkiintoisesta ja rennosta ilmapiiristä sekä kavereitani opiskeluajan sähellyksistä. Suuret kiitokset tuesta myös vanhemmilleni sekä erityisen lämmin kiitos kihlatulleni Kaisa Kaivolalle.

Sisällysluettelo

KÄYTETYT MERKINNÄT JA LYHENTEET

MERKINNÄT JA LYHENTEET

ABI	Application Binary Interface
ALU	Arithmetic Logic Unit, aritmeettislooginen yksikkö
ANSI C	American National Standards Institute, Standard X3.159-1989
ASIC	Application Specific Integrated Circuit
BDM	Background Debug Mode
BFM	Bus Functional Model, väylän toiminnallinen malli
BIP	Break In Progress
BRAM	Block SelectRAM
CCLK	The Configuration Clock Pin
CF	Compact Flash, muistityyppi
CLB	Configurable Logic Block, logiikkalohko
DCM	Digital Clock Manager, digitaalinen kellonhallinta
DDR	Double Data Rate
DMA	Direct Memory Access, muistin suorasaanti
DMIPS	Dhrystone Million Instruction Per Second
DSP	Digital Signal Processor, digitaalinen signaaliprosessori
DTC	Direct Torque Control, suora käämivuosäätö
EDIF	Xilinx Primitive Netlist
EDK	The Embedded Development Kit, kehitysympäristö
ELF	Executable and Link Format
EMC	External Memory Controller
EST	Embedded System Tools, sulautetun järjestelmän työkalut
FAA	Federal Aviation Administration
FF	FlipFlop, kiikku
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
FSL	Fast Simplex Link
GDB	GNU Project Debugger
GNU	”GNU's Not Unix”

GPL	General Public License
GPR	General Purpose Register, yleiskäyttöinen rekisteri
GTS	Global 3-state signal, yleinen kolmetilainen signaali
GWE	Global Write Enable
HW	Hardware, laitteisto
IE	Interrupt Enable, keskeytykset sallittu
IEEE	Institute of Electrical and Electronics Engineering
IOB	Input/Output Block
IP	Intellectual Property
IPIC	Intellectual Property Interconnect
IPIF	Intellectual Property Interface
ISC	In-System Configurable
ISR	Interrupt Service Routine, keskeytysaliohjelma
JTAG	Boundary-scan
LE	Logic Element, Alteran logiikka-elementti
LGPL	GNU Lesser General Public License
LIFO	Last In, First Out
LMB	Local Memory Bus, paikallinen muistiväylä
LTY	Lappeenrannan teknillinen yliopisto
LUT	Look-up Table, hakutaulukko
MAC	Multiply-accumulate
MDM	Microprocessor Debug Module
MHS	Microprocessor Hardware Specification
MMU	Memory Management Unit, muistinhallintayksikkö
MPM	Multi-Package Module
MSR	Machine Status Register, prosessorin tilarekisteri
MSS	Microprocessor Software Specification
MULT_AND	Multiplicate and AND
NGC	Xilinx Netlist tiedostomuoto
OPB	On-Chip Peripheral Bus
PC	Program Counter, ohjelmlaskuri
POR	Power On Reset

PROM	Programmable Read-Only Memory, uudelleenohjelmoitava pysyväismuisti
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory, pysyväismuisti
RTCA	Radio Technical Commission For Aeronautics
RTOS	Real-Time Operating System, reaaliaikakäyttöjärjestelmä
sbss	Small Block Started By Symbol
SelectRAM	Block SelectRAM (BRAM)
SDA	Small Data Area, pieni data-alue
SDRAM	Synchronous Dynamic Random Access Memory
SOP	Sum Of Product, tulojen summa
SRAM	Static Random Access Memory
SystemACE	System Advanced Configuration Environment
SW	Software, ohjelmisto
TAP	Test Access Port
TBUF	3-State Buffer
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TEKES	Tekniikan kehittämiskeskus
TEMO	Tehoelektroniikkalaitteen modulaarinen ohjaustekniikka
TLB	Translation look-aside buffer
TMS	Test Mode Select
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, laitteistokuvauskieli
V_{CCAUX}	Auxiliary Supply Voltage
V_{CCINT}	Internal Supply Voltage
V_{CCO}	Output Driver Supply Voltage
WCET	Worst Case Execution Time, pahimman tapauksen suoritusaika
XMD	Xilinx Microprocessor Debug
XOR	Exlusive-OR
XPS	Xilinx Platform Studio

SUUREET

f	taajuus	[Hz]
I	virta	[A]
P	teho	[W]
U	jännite	[V]

1 JOHDANTO

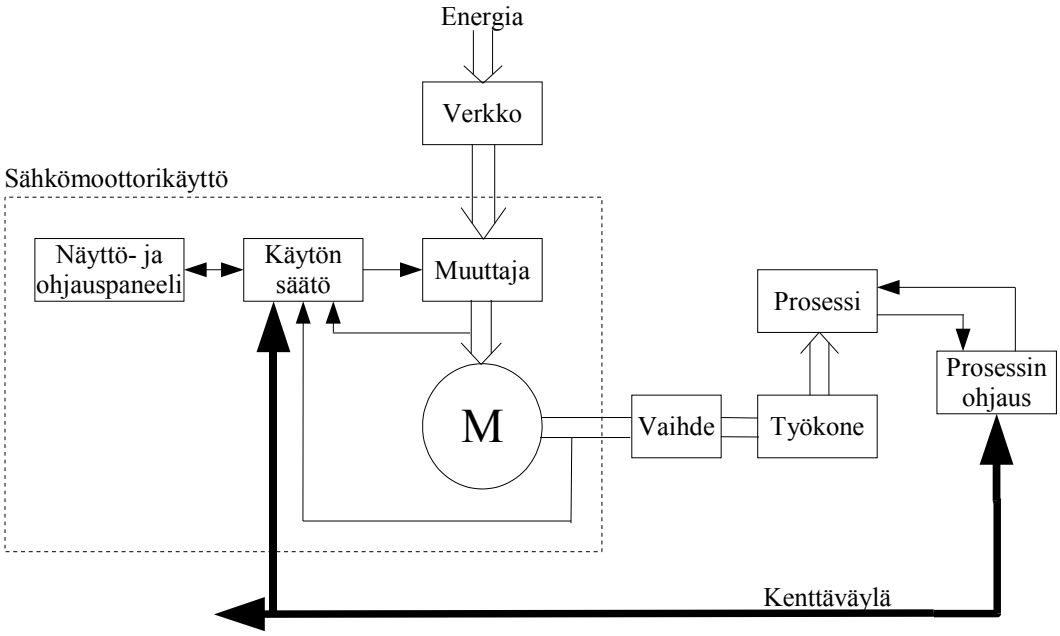
Tyypillinen sähkökäytön säätöjärjestelmä koostuu monista toiminnallisista lohkoista ja niiden toteuttamiseen onkin perinteisesti käytetty paljon eri komponentteja. Esimerkiksi pelkkä säätö on toteutettu käyttäen mikroprosessoria tai digitaalista signaaliprosessoria (DSP) sekä kaikkein nopeimpiin suorituksiin ASIC-piirejä. Eräänä vaihtoehtona on käyttää FPGA-piirille (Field Programmable Gate Array) ohjelmistopohjaisella IP-lohkolla (Intellectual Property) toteutettua sulautettua mikroprosessoria (erityisesti Xilinx:n MicroBlaze:a) yhdessä samalla piirillä toimivan sähkökäytön tehoelektroniikan ohjauksen kanssa. Etuna tästä yhdellä piirillä toteutetusta ratkaisusta on esimerkiksi se, että vältytään komponenttien välisiltä piirilevyvedoilta. Piirilevyvedokset keräävät helposti sähkömagneettisia häiriöitä, jotka voivat hidastaa tai pahimmillaan jopa sekoittaa koko järjestelmän toiminnan. Yhdellä mikropiirillä toteutetulla järjestelmällä on etuna häiriösietoisuuden lisäksi myös nopeuden kasvu, jolloin säätöalgoritmeja voidaan parannella entisestään sekä niihin voidaan lisätä uusia ominaisuuksia. Myös koko säätöjärjestelmän rakenteen muuttaminen on helppoa FPGA-piirin uudelleen ohjelmoitavan luonteen vuoksi.

Toteutusalustana on ohjelmoitava logiikkapiiri eli FPGA, jonka koko ja nopeus on kasvanut riittävän suureksi käytettäväksi tällaisiin järjestelmiin sekä piirien hinta on laskenut kuluttajamarkkinoillekin riittävän alhaiseksi. FPGA-piirille voidaan muun logiikan lisäksi liittää myös prosessori. Prosessori voidaan liittää kahdella tavalla, laitteistopohjaisella tai ohjelmistopohjaisella IP-lohkolla. Laitteistopohjaisella IP-lohkolla prosessori toteutetaan kiinteästi FPGA-piirille muun logiikan lisäksi. Ohjelmistopohjainen prosessori toteutetaan taas käyttäen FPGA-piirin logiikkaa prosessorin toteutukseen, jolloin etuna laitteistopohjaiseen toteutukseen verrattuna on sen joustavuus. Ohjelmistopohjaisen prosessorin muita etuja ovat mm. järjestelmän helppo muunneltavuus, jolloin prosessorin ominaisuuksista voidaan ottaa käyttöön vain tarvittavat osat sekä päivittäminen uusien vaatimusten mukaiseksi on helpompaa. Yleisesti FPGA-piirille liitetyllä prosessorilla on etuna perinteiseen prosessorijärjestelmään verrattuna se, että siihen voidaan helpommin lisätä omia lohkoja, joilla voidaan esimerkiksi lisätä ohjelmiston suoritusnopeutta suorittamalla aikaa vaativat operaatiot rinnakkaisesti laitteistolla.

Laitteistoalustan lisäksi sulautetussa järjestelmässä on suuressa osassa siinä suoritettava ohjelmisto sekä näiden yhteenliittäminen. Tehokkaan ohjelmiston avulla sulautetun prosessorijärjestelmän resurssit voidaan ottaa paremmin käyttöön. Reaaliaikakäyttöjärjestelmä on ratkaisu, jonka avulla prosessorin resursseja voidaan helpommin hyödyntää useampien tehtävien kesken. Se tarjoaa myös työkalut prosessorissa olevien ja siihen liitettävien resurssien jakamiseen. Näiden resurssien hallintapalveluiden ansiosta prosessorilla suoritettavat samanaikaiset tehtävät voivat käyttää esimerkiksi sarjaporttia tai muistia dynaamisesti sotkematta samalla muiden tehtävien toimintaa. Aikakriittisessä sovelluksessa ennen käyttöjärjestelmän käyttöönottoa on kuitenkin hyvä tietää kuinka pitkään eri käyttöjärjestelmän palveluiden suorittaminen kestää. Tässä työssä tarkastellaan myös μC/OS-II reaaliaikaydintä, joka tarjoaa hyvän siirrettävyyden eri laitteistoalustoille, keskeyttävän reaaliaikaisuuden, hyvin mukautuvaisen sekä moniajavan ytimen mikroprosessoreille ja mikrokontrollereille. [1]

1.2 Perinteinen sähkökäyttö

Sähkömoottorikäyttö eli lyhyemmin sähkökäyttö, on syöttöverkon ja prosessin välinen energiamuunnin, joka muuntaa sähköenergiaa mekaaniseksi työksi. Tätä muunnosprosessia voidaan hallita muuttajan avulla, joka perustuu puolijohdekytkimillä toteutettuihin tasasuuntaajiin, vaihtosuuntaajiin ja tasasähkökatkojiin. Sähkömoottorikäyttö jakautuu energiaa siirtävään sähkömekaaniseen osaan ja sitä ohjaavaan säätöosaan. Kuva 1 esittää tällaisen sähkömoottorikäytön perusjärjestelmän, jossa säätöosa ohjaa sähkömoottoria taajuudenmuuttajan avulla. Sähkömoottori taas liikuttaa vaihteen kautta työkonetta, joka muuttaa sähkömoottorin tuottaman mekaanisen työn hyötytyöksi prosessiin. Ympyrä sulkeutuu, kun säätöosa saa kenttäväylää pitkin prosessilta takaisinkytkentäarvon haluttua toimintaa varten.



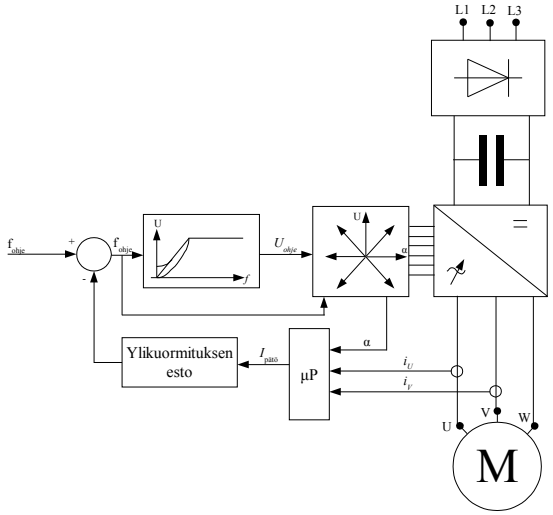
Kuva 1. Perinteinen sähkömoottorikäyttö osana sähkökäyttöjärjestelmää. [2]

Sähkömoottorikäytöistä yksinkertainen ja teollisuuden yleisimmin käyttämä sähkömoottorityyppi on epätahtikoneisiin kuuluva oikosulkumoottori. Epätahtikoneille on tyypillistä, että nimensä mukaisesti niiden pyörimisnopeus poikkeaa kuormitettuna tahtinopeudesta. Tätä nopeuseroa kutsutaan jättämäksi ja se on tarpeen, koska vain näin saadaan oikosuljettuun häkkikäämitykseen indusoitua vääntömomenttia muodostava virta. Oikosulkumoottorin yksinkertainen ja dynamiikkavaatimuksiltaan vähäisiin käyttöihin soveltuva säätö on skalaarisäätö. [2]

1.2 Käytön säätö

Sähkökäytöissä tavoitteena on asennon, nopeuden tai vääntömomentin ohjaaminen halutulla tavalla. Tähän liittyen lisäehtona on koneen jännitteen eli itse asiassa käämivuon pitäminen haluttuna kussakin toimintatilassa. Oikosulkumoottorin säätötapoja ovat skalaarisäätö, vektorisäätö ja suora vääntömomenttisäätö. Vaikka skalaarisäätö on varhaisimpien taajuusmuuttajien säätöperiaate, niin sillä voidaan toteuttaa kohtuullisen hyvin koko nopeusalueella toimiva käyttö. Kuitenkin on huomioitava se, että koneen ohjaus perustuu tällöin pysyvän tilan malleihin, jotka pätevät tarkasti vain nopeuden ja kuormituksen ollessa vakioita. Eli mitä nopeampia muutoksia vääntömomentissa halutaan, sitä suuremmat ovat skalaarisäädön tekemät

dynaamiset virheet. Skalaarisäätö on perusluonteeltaan taajuussäätö eli ohjaussuureina ovat moottorin taajuus, jännite ja niiden korjaukset virtamittauksilla. Kuvassa 2 on esitetty pyörimisnopeusohjeen saavan skalaarisäätöisen taajuudenmuuttajan ohjauksen peruskytkentä. Kuvan oikeassa reunassa ovat taajuudenmuuttajan osat eli verkkosilta, välipiiri ja vaihtosuuntaaja. Vaihtosuuntaajaa ohjaa modulaattori, joka saa ohjearvonsa säädöltä. [3]



Kuva 2. Erään skaalarisäätöisen taajuudenmuuttajan ohjauksen peruskytkentä. f_{ohje} on pyörimisnopeuden ohjearvo, f_{ohje} on skalaarisäädön ohjearvo ja U_{ohje} on vektorimodulaattorin jänniteohje. L_1, L_2 ja L_3 ovat verkon vaiheet ja U, V sekä W ovat moottorin vaiheet. Moottorin vaiheista mittaukset i_U ja i_V, sekä modulaattorista jännitevektori α menevät mikroprossorille. Mikroprosessorin tuottama $I_{pätö}$ ohje kulkee ylikuormituksen eston kautta takaisin säädölle. [4]

Skalaarisäätöä paremmin muutostilanteissa ohjaava vektorisäätö perustuu vuoden 1970 alussa Blaschken esittelemään kiertokenttäkoneiden vektorisäätöideaan. Perusluonteeltaan vektorisäätö on vääntömomenttisäätö, jolle pyörimisnopeussäätö antaa ohjearvon. Vakiovuoalueella toimittaessa vektorisäätö on varsin hyvä menetelmä ja suurimpana haittana on ainoastaan sen herkkyys roottoriresistanssin, magnetointi-induktanssin ja roottorin hajainduktanssin arvoissa oleville virheille. Mikäli niiden arvot eivät ole oikein, estimoitu roottorivuo poikkeaa oikeasta sekä suuruudeltaan, että suunnaltaan. [3]

Suora vääntömomenttisäätö perustuu Depenbrock:n ja Takahashi:n 1980-luvun puolessavälissä esittämiin ideoihin, ja joihin ABB:n vuonna 1994 markkinoille tuoma

DTC-taajuudenmuuttajakin (Direct Torque Control) perustuu. Edellisistä roottorin käämivuon estimointiin perustuvista säätöperiaatteista poiketen DTC:n periaate on estimoida staattorin käämivuo moottorin staattorijännitteen ja -virran avulla. Käytännössä DTC:n ydin on toteutettu DSP-prosessorilla, jonka toiminta jakautuu eri aikatasoihin siten, että nopeimmalla 25 μs aikatasolla määritetään kytkimien ohjaus. DTC:n vaatima optimikytkentälogiikka on yleensä toteutettu käyttäen ASIC-piirejä (Application Specified Integrated Circuit), joiden tehtävänä on valita laskettujen vääntömomentin ja käämivuon muutostarpeiden perusteella aina parhaat hetkelliset kytkinasennot. DSP:llä suoritettavia aikatasoja ovat esimerkiksi 100 μs, 200 μs, 500 μs, 1 ms jne., joille sijoitetaan tehtäviä tärkeysjärjestyksessä. Oikosulkukoneen suora vääntömomenttisäätö on yleiskäyttöinen ja periaatteessa hyvin yksinkertainen moottorin säätömenetelmä, joka soveltuu kaikenlaisiin tehtäviin. [3]

1.3 TEMO-hanke

Tämä työ liittyy Tekes:in ÄLY2000-teknologiaohjelmaan kuuluvaan hankkeeseen ”Tehoelektroniikkalaitteen modulaarinen ohjaustekniikka” (TEMO), jonka tavoitteena on kehittää tehokas logiikkapiireihin (erityisesti FPGA-piireihin) perustuva toteutusmalli tehoelektroniikkalaitteen ohjausjärjestelmälle. Toteutusmallin tavoitteena on ottaa huomioon tekninen tehokkuus, kustannustehokkuus, testattavuus ja ylläpidettävyys. Tavoitteena on myös kehittää uusia suunnittelumenetelmiä tehoelektroniikkalaitteen ohjausjärjestelmien toteuttamiseen ja ottaa käyttöön olemassa olevia ohjelmoitavien piirien suunnittelutyökaluja.

Esimerkkisovelluksina hankkeessa toteutetaan kaksi prototyyppiä joista toinen on tässä työssä esitelty taajuudenmuuttaja ja toinen on hitsauskoneen ohjausjärjestelmä. Hankkeen tuottamia julkaisuja ovat mm. kesällä 2005 konferensseihin hyväksytyt artikkelit, joista toinen esittelee FPGA-piirillä olevien IP-lohkojen välisen väylärakenteen ja toinen tarkastelee taajuusmuuttajan säätöjärjestelmän toteuttamista yhdellä FPGA-piirillä [5],[6]. Hankkeessa on hyödynnetty myös VHDL:n testaukseen kehitettyä reaaliaikaista dSPACE-alustaista sähkömoottorin emulaattoria [7].

Edellä mainittu IP-lohkojen välinen väylärakenne on nimeltään OKITO. OKITO-väylän

debuggaukseen on myös kehitetty sulautetulla prosessorilla suoritettava debuggaustyökalu, jolla voidaan seurata väylällä olevia laitteita. Debuggaustyökalu on tosin tarkoitettu vain sovellustason debuggaukseen eli ajoitusten tai erittäin nopeiden ilmiöiden seuraamiseen se ei sovellu. [8]

1.4 Työn määrittely ja tavoitteet

Työn lähtökohtana on toteuttaa sähkökäytön säätöjärjestelmä FPGA-piirille sulautetulla mikroprosessorilla. Ensimmäisenä on tutustuttava FPGA-piiriin ja mitä se tarvitsee toimiakseen. Tarkastellaan FPGA-piirin resursseja sekä kuinka resurssien kulutusta voidaan yksinkertaisesti etukäteen arvioida. Tutustutaan myös sulautettuihin ohjelmistoprosessoreihin ja niistä lähemmin Xilinx:n MicroBlaze-prosessoriin sekä hiukan ohjelmistoprosessorijärjestelmän kehitys- ja testausmetodiikkaan. Tarkastellaan MicroBlaze-prosessorin aritmeettisten operaatioiden kestoa kokonais- ja liukuluvuilla sekä trigonometrisiä operaatioita.

Tutustutaan mitä hyötyä reaaliaikakäyttöjärjestelmästä on monimutkaisen sulautetun prosessorijärjestelmän hallintaan ja mitä palveluita sillä on tarjota. Ensin on kuitenkin tarkasteltava mitä lisälaitteita ja asetuksia prosessori tarvitsee toimiakseen ja kuinka reaaliaikakäyttöjärjestelmää voidaan prosessorilla suorittaa. Tehdään reaaliaikakäyttöjärjestelmistä μC/OS-II:lle suorituskykymittauksia, joista voidaan päätellä mitä palveluita aikakriittisissä sovelluksissa on tehokkainta käyttää. Viimeisenä toteutetaan vielä prototyyppi, jolla voidaan testata yksinkertaista skalaarisäätöistä moottorinohjausjärjestelmää.

2 SULAUTETUN FPGA-JÄRJESTELMÄN RAKENNE

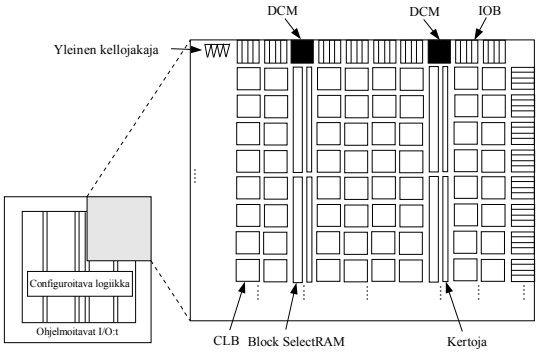
Älykkään taajuudenmuuttajan toteuttaminen vaatii moottorisäädön lisäksi mm. tietoliikenne-, diagnostiikka-, automaatio- ja käyttöliittymätoimintojen sisällyttämisen järjestelmään. Kokonaisuudessaan taajuusmuuttajassa on siis hyvin monipuoliset sekä vaativat ominaisuudet. Kaikkien näiden ominaisuuksien täyttäminen tarvitsee siten monipuolisen laitealustan, riittävästi muistia ja nopeutta. Tässä kappaleessa tutustutaan tällaisen järjestelmän kehittämiseen ja testaamiseen tarvittaviin työkaluihin. Otetaan huomioon myös reaaliaikakäyttöjärjestelmän tarjoamat tekniikat sekä edut, joiden avulla voidaan helpommin kehittää ja hallinnoida koko järjestelmää. Ohjelmistoa täytyy pystyä päivittämään myös käyttöönoton jälkeen, joten selvitetään kuinka ohjelmistoa päästään muuttamaan kenttäolosuhteissa. Ensimmäiseksi tutustutaan kuitenkin järjestelmän toteuttamiseen käytettävään piiriarkkitehtuuriin, FPGA-piirin resursseihin ja FPGA-piirille sulautettuihin prosessoreihin. Prosessoreista tutustutaan vielä tarkemmin MicroBlaze ohjelmisto-IP-prosessorin ominaisuuksiin ja tutkitaan, riittävätkö ne taajuusmuuttajan ohjaukseen sekä tarvitaanko FPGA-piirin lisäksi muita ulkoisia piirejä.

2.1 FPGA-piiri

Järjestelmän kehitykseen ja testaamiseen käytetty Memec Design:n FPGA-kehitys- ja testauslevy DS-DB-V2MB1000 sisältää Xilinx:n Virtex-II (XC2V1000) FPGA-piirin. Tälle FPGA-piirille kehitysympäristö tarjoaa 100 MHz ja 24 MHz nopeuksiset kellosignaalit. Tarkastellaan seuraavaksi hiukan tätä kehitysympäristön sisältämää FPGA-piiriä sekä verrataan sitä kahteen muuhun Xilinx:n tarjoamaan piiriin.

FPGA-piirin pääelementtejä ovat logiikkalohkot (CLB, Configurable Logic Block) joista varsinainen logiikka muodostetaan. Muistielementteinä piirillä on Block SelectRAM-lohkot (BRAM) ja piiriltä löytyy myös pelkästään kertolaskuihin käytettäviä kertojayksiköitä. Kaikkia toiminnallisia lohkoja ohjaa kellojakaja, jonka avulla logiikka saadaan toimimaan yhtenäisesti. Digitaalisilla kellonhallintayksiköillä (DCM, Digital Clock Manager) voidaan muodostaa syötetystä kellosignaalista haluttu kellotaajuus. Liityntöjä ulkomaailmaan hoitavat IOB-lohkot. Kuvassa 3 on Virtex-II FPGA-piiriarkkitehtuuri, josta näkyy kuinka ulkoiset liitynnät (IOB), konfiguroitavat

logiikkalohkot, BRAM-lohkot ja kertojayksiköt sijoittuvat piirille. [9]



Kuva 3. Virtex-II FPGA-piiriarkkitehtuurin rakenne, josta nähdään logiikkalohkojen (CLB), ulkoisten liityntälohkojen (IOB), BRAM-lohkojen ja kertojayksiköiden sijoittuminen piirillä [9]

DS-DB-V2MB1000 kehitys- ja testauslevyn FPGA-piiri sisältää 40x32 (1280) konfiguroitavaa logiikkalohkoa. Näitä logiikkalohkoja käytetään rakennettaessa kombinatorista ja synkronista logiikkaa. Yksi CLB-elementti sisältää 4 samanlaista siivua (slice), joista jokainen siivu sisältää kaksi neljätuloista funktio-generaattoria, ylivuotologiikan, aritmeettisia logiikkalohkoja, leveitä funktioiden ottovalitsimia ja kaksi muistielementtiä. Aritmeettisissa logiikkalohkoissa on XOR-tyyppinen logiikkaportti, joka mahdollistaa 2-bittisen kokosummaimen (full adder) toteuttamisen yhdessä siivussa. Lisäksi aritmeettisissa logiikkalohkoissa on kertolaskujen tehostamiseen AND-tyyppinen logiikkaportti (MULT_AND). Jokainen neljätuloinen funktio-generaattori voidaan ohjelmoida neljätuloiseksi hakutauluksi (LUT, Look-Up Table), 16-bittiseksi jaetuksi muistiksi (Distributed SelectRAM), tulojen summaketjuksi (SOP, Sum Of Product) tai 16-bittiseksi siirtorekisterielementiksi. Jokaisessa CLB-lohkossa on myös kaksi 3-tilaista ajuria (TBUF) piirin sisäisiä väyliä varten ja joita CLB-lohkojen siivut pääsevät käyttämään valintamatriisin (switch matrix) kautta. Taulukkoon 1 on koottu edellä esitellyt yhden CLB-lohkon tarjoamat resurssit.

Taulukko 1. Virtex-II FPGA-piirin CLB-lohkon resurssit. [9]

Siivut	LUT	Flip-Flops	MULT_AND	Aritmetiikka & Carry-ketjut	SOP ketjut	Distributed SelectRAM	Siirtorekisterit	TBUF
4	8	8	8	2	2	128 bittiä	128 bittiä	2

Kertojalohkoja XC2V1000-piirillä on 40 kappaletta ja ne ovat 18x18-bittisiä etumerkillisiä 2:n komplementtia käyttäviä lohkoja. Nämä kertojalohkot voidaan

yhdistää 18 kb:n muistilohkojen kanssa tai käyttää yksistään. Ne on optimoitu suurille nopeuksille ja kuluttavat vain vähän tehoa verrattuna 18x1-bittisille siivuista tehdyille kertojille. Piirillä on myös 40 kappaletta 18 kb:n, eli yhteensä 720 kb, BRAM-muistilohkoja. BRAM-lohkot ovat suositeltavampi tapa rakentaa muistia, kuin käyttää CLB-lohkojen jaettua muistielementtiominaisuutta.

Piiri tarjoaa myös laajan valikoiman tehokkaita kellosignaalin käsittelyominaisuuksia DCM-lohkojen avulla. DCM-lohkoja voidaan käyttää digitaalisina viivelinjoina mahdollistamaan luotettavan ja tarkan kellosignaalin vaiheen sekä taajuuden käsittelyn. Ne tarjoavat myös digitaalisen takaisinkytkentäjärjestelmän, jolla voidaan kompensoida lämpötilan ja jännitteen vaihteluiden aiheuttamat kellon virheet.

Taulukkoon 2 on koottu edellä olevat Virtex-II XC2V1000 FPGA-piirin esitetyt ominaisuudet sekä vertailun vuoksi myös Virtex-II XC2V4000 ja Spartan-3 XC3S1000:n vastaavat arvot. Taulukon järjestelmälohkojen määrä vastaa FPGA-piirin resurssien toteuttamista digitaalisilla peruslogiikkalohkoilla. Taulukosta nähdään, että XC2V4000 on noin neljä kertaa suurempi kuin XC2V1000, kun taas XC3S1000 vastaa järjestelmälohkojen osalta XC2V1000:sta, mutta muilta osin ominaisuudet jäävät vähäisemmiksi.

Taulukko 2. Virtex-II FPGA-piirien XC2V1000 ja XC2V1000 sekä Spartan-3 XC3S1000 tarjoamat resurssit. [9]

		CLB (1 CLB = 4 siivua = Enintään 128 bits)				SelectRAM Lohkoa			
Piiri	Järjestelmä-lohkoa	Kenttä rivi x sarake	Siivua	Enintään jaettua RAM:ia (kb)	Kertoja lohkoja	18 kb lohkoa	Enintään RAM:ia (kb)	DCMs	Max I/O:t
XC2V1000	1M	40x32	5120	160	40	40	720	8	432
XC2V4000	4M	80x72	23040	720	120	120	2160	12	912
XC3S1000	1M	48x40	7680	120	24	24	432	4	391

2.1.1 Block SelectRAM

Jokainen Block SelectRAM-lohko on 18 kb:n kokoinen kaksiporttinen muistielementti. Nämä muistielementit voidaan kellottaa erikseen sekä ohjata synkronisilla porteilla, jotka liittyvät yhteiseen muistialueeseen. Molemmat portit ovat toiminnoiltaan täysin samanlaisia. Lohkot tukevat erilaisia muistinasetusjärjestelyjä, kuten yksi- ja

kaksiporttista muistia sekä monia data/osoite-muotoja. [9]

2.1.1.1 Yksiporttinen

Yksiporttisena käytettäessä SelectRAM-lohkolla on pääsy 18 kb:n muistialueelle seuraavilla konfiguraatioilla: 2K x 9-bittiä, 1K x 18-bittiä tai 512 x 36-bittiä. 16 kb:n muistialueelle muistilohkolla on pääsy vastaavasti konfiguraatioilla: 16K x 1-bittiä, 8K x 2-bittiä tai 4K x 4-bittiä. Etu käytettäessä 9-bittistä, 18-bittistä tai 36-bittistä muistinleveyttä on mahdollisuus tallentaa pariteettibitti jokaiselle kahdeksalle bitille. Pariteettibittiä ei kuitenkaan muodosteta automaattisesti vaan se on muodostettava ja tarkastettava omalla erillisellä logiikalla. [9]

2.1.1.2 Kaksiporttinen

Kaksiporttisena käytettäessä kummallakin SelectRAM-lohkon portilla on pääsy yhteiseen 18 kb:n muistiin. Kaksiporttisena muistinleveydet voidaan konfiguroida itsenäisesti ja se tarjoaa mahdollisuuden valmiiseen väyläleveyden muutokseen. Jos molemmat portit on asetettu joko 2K x 9-bittiseksi, 1K x 18-bittiseksi tai 512 x 36-bittiseksi, niin 18 kb:n lohko on molempien A- ja B-portin käytettävissä. Jos molemmat portit on konfiguroitu joko 16K x 1-bittiseksi, 8K x 2-bittiseksi tai 4K x 4-bittiseksi, niin 16 kb:n lohko on molempien A- ja B-portin käytettävissä. Kaikilla muilla konfiguraatioilla tulos on, että toisella portilla on pääsy 18 kb:n muistilohkoon ja toisella portilla on pääsy 16 kb:n alalohkoon. [9]

2.1.2 FPGA-piirin konfigurointi ja päivittäminen

FPGA-piiri ei säilytä ohjelmoitua logiikkaa siten, että ne säilyisivät myös virrankatkaisun jälkeen. Tämän vuoksi FPGA-piirin täytyy aina käynnistyksen yhteydessä lukea ohjelmointitiedot jostain ulkopuoliselta datalähteeltä. Esimerkiksi Virtex-II XC2V1000 mallissa konfigurointibittejä on 4,082,592 kappaletta ja niiden konfigurointi voidaan tehdä viidellä tavalla:

- Orja-sarjamuoto
- Isäntä-sarjamuoto
- Orja SelectMAP
- Isäntä SelectMAP

- Boundary-Scan (IEEE 1532) [40]

Näissä viidessä konfigurointititavassa käytettävät FPGA-piirin liitynnät esitellään taulukossa 3. Konfigurointitapa valitaan FPGA:n siihen varatuilla pinneillä M2, M1 ja M0. Lisäksi HSWAP_EN-pinnillä valitaan, käytetäänkö ylösvetovastuksia I/O-pinneissä ohjelmoinnin aikana. Oletuksena HSWAP_EN on sisäisellä vastuksella kytketty ylös, jolloin ylösvetovastukset eivät ole käytössä. Muita varattuja pinnejä ovat CCLK (The Configuration Clock Pin), DONE, PROG_B ja boundary-scan pinnit: TDI, TDO, TMS ja TCK. Konfigurointitavasta riippuen CCLK voi olla FPGA:lta ohjattavissa (output) tai luettavissa (input). Pysyvyysvalinnalla voidaan vielä valita, pysyykö konfigurointitivaiheessa käytetyt pinnit ohjelmoinnin jälkeenkin varattuina vai vapautetaanko ne yleiseen käyttöön. Pysyvyysvalintaan eivät kuulu kuitenkaan CCLK, PROG_B, DONE ja boundary-scan pinnit. Tästä pysyvyysvalinnasta on hyötyä esimerkiksi tilanteissa, jolloin piiri on vain osin ohjelmoitu tai se halutaan uudelleenohjelmoida kokonaan käytön aikana. [9]

Taulukko 3. FPGA-piirin konfigurointiin varatut pinnit, niiden suunta ja selvitys mihin pinnejä käytetään. [9]

Pinni	Suunta	Selvitys
CCLK	I/O	Konfigurointikello. Lähtömuotoa käytetään isäntämuodossa ja tuloa orjamuodossa.
PROG_B	I	Alhaalla aktiivinen sykroninen logiikan resetointi.
DONE	I/O	Kaksisuuntainen signaali jota lähtömuodossa käytetään ilmoittamaan konfigurointiprosessin valmistumisesta. Tulomuodossa tilan alhaalla pitämisellä voidaan viivästää käynnistyssekvenssiä.
M0, M1, M2	I	Konfigurointimoodin valinta.
HSWAP_EN	I	Konfiguroinnissa ylösvetovastusten käytön valinta.
CS_B	I	SelectMAP-moodissa tämä signaali on alhaalla aktiivinen piirin valintasignaali.
RDWR_B	I	SelectMAP-moodissa tämä signaali on alhaalla aktiivinen kirjoitustoiminnon ilmoitussignaali
BUSY/DOUT	O	SelectMAP-moodissa BUSY-signaali säätelee configurointidatan siirtonopeutta. Sarja-moodissa DOUT tarjoaa datalahdön ketjun seuraaville piireille.
DIN	I	Sarja-moodissa datan tulo
TCK	I	Boundary scan kello
TDI	I	Boundary scan tulodata.
TDO	O	Boundary scan lähtödata.
TMS	I	Boundary scan testimoodin valinta.

Orja-sarjamuodossa FPGA-piiri vastaanottaa ohjelmointitiedot esim. sarjamuotoiselta

PROM-piiriltä (Programmable Read-Only Memory). Tällä menetelmällä voidaan ohjelmoida myös useampi FPGA-piiri ketjutetusti samalla ohjelmointitiedolla eli kun yksi piiri valmistuu, niin se alkaa D_{OUT}-pinnin kautta syöttää dataa taas seuraavalle piirille.

Isäntä-sarjamuodossa CCLK-pinni toimii lähtönä eli FPGA-piiri ohjaa konfigurointikellopulssia. Tällöin esimerkiksi sarjamuotoinen PROM-piiri syöttää CCLK:n ohjaamana ohjelmointiedot sarjamuodossa FPGA:n D_{IN}-pinniin. Tässä muodossa onnistuu myös edelliseen tapaan FPGA-piirien ketjuttaminen.

SelectMAP-muoto on nopein, jolloin ohjelmointitieto siirretään rinnakkaisesti tavun levyisenä. FPGA-piiri vastaanottaa datavirtaa BUSY-lipulla ohjaten. Ulkoisen lähteen täytyy pitää dataa väylällä niin kauan, kuin BUSY-signaali pysyy ylhäällä. Ulkoinen lähde tarjoaa CCLK:n, alhaalla aktiivisen piirinvalintasignaalin (CS_B) ja takaisinkirjoitussignaalin (RDWR_B). Jos RDWR_B-signaali on asetettu, ohjelmointiedot luetaan FPGA:lta takaisinkirjoitusoperaation mukaisesti. Ohjelmoinnin jälkeen rinnakkaissiirtoon käytetyt pinnit voidaan tarvittaessa vapauttaa vapaaseen käyttöön. Tässä konfigurointimuodossa voidaan useampi FPGA-piiri ohjelmoida ja käynnistää rinnakkain samanaikaisesti. Myös yksittäisten piirien ohjelmointi rinnakkaisesti kytketyssä väylässä onnistuu käyttäen CS_B-pinniä.

Isäntä SelectMAP-muoto on vastaava kuin edellinen, mutta FPGA-piiri ohjaa CCLK-signaalia.

Boundary-scan-muodossa FPGA-piirin konfigurointiin käytetään siihen varattuja pinnejä eli ohjelmointi tapahtuu täysin IEEE 1149.1 [39] standardin mukaisen TAP-portin (Test Access Port) mukaisesti. Boundary-scan on yleinen mikropiirien testausmenetelmä, jossa mikropiirit verkotetaan samaan testausketjuun. Testauksen lisäksi tämän menetelmän avulla voidaan myös ohjelmoida piirejä eli Virtex-II piirien boundary-scan ohjelmointi on yhteensopiva IEEE 1149.1 standardin sekä IEEE 1532 [40] standardin mukaisten ISC (In-System Configurable) laitteiden kanssa. IEEE 1532 standardi on alaspäin yhteensopiva IEEE 1149.1-1993 TAP-portin ja tilakoneen kanssa. ISC laitteiden on tarkoitus olla ohjelmoitavia, uudelleenohjelmoitavia tai testattavissa

piirilevyllä fyysisesti sekä loogisesti. Ohjelmointi tämän boundary-scan-portin kautta on aina mahdollista ja riippumatonta ohjelmointimuodosta eli valittaessa boundary-scan-muoto estetään ainoastaan muiden muotojen käyttö. [9]

Taulukkoon 4 on kirjattu eri konfigurointimuotojen valintapinnien asetukset, CCLK-signaalin suunta, dataväylän leveys ja D_{OUT}-portin sarjamuotoisuus.

Taulukko 4. Virtex-II FPGA-piirin konfigurointitimuotojen asetukset, CCLK-signaalin suunta, dataväylän leveys ja D_{OUT}-portin sarjamuotoisuus. [9]

Konfigurointimuoto	M2	M1	M0	CCLK:n suunta	Datan leveys	Sarja D_{OUT}
Isäntäsarja	0	0	0	O	1	Kyllä
Orjasarja	1	1	1	I	1	Kyllä
Isäntä SelectMAP	0	1	1	O	8	Ei
Orja SelectMAP	1	1	0	I	8	Ei
Boundary Scan	1	0	1	N/A	1	Ei

2.1.2.1 Käynnistyssekvenssi

Virtex-II FPGA-piirin käynnistyssekvenssi koostuu kolmesta osasta tehojen kytkennän ja resetoimisen (POR, Power On Reset) jälkeen. POR tapahtuu, kun FPGA-piirin käyttöjännitteistä V_{CCINT} on suurempi kuin 1.2 V, V_{CCAUX} on suurempi kuin 2.5 V ja V_{CCO} on suurempi kuin 1.5 V. Nämä jännitetasot saavutettuaan siirrytään suorittamaan kolmivaiheista käynnistyssekvenssiä.

Ensimmäiseksi ohjelmointimuisti (configuration memory) tyhjennetään. Seuraavaksi ohjelmointitiedot ladataan edellisen kappaleen mukaisesti muistiin ja viimeiseksi käynnistysprosessissa logiikka aktivoidaan.

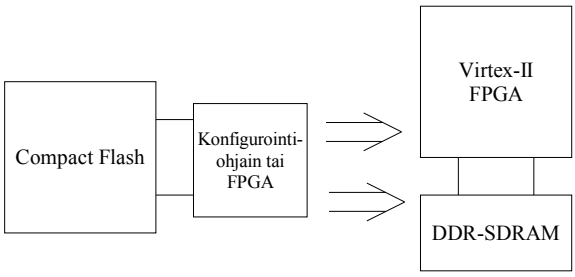
Konfigurointi aloitetaan automaattisesti virtojen kytkennän jälkeen, ellei sitä viivästetä. Viivästämistä voidaan suorittaa pitämällä ohjelmointimuistin tyhjennykseen käytettyä INIT_B-pinniä alhaalla tai asettamalla PROG_B-pinni. Peruskäynnistyssekvenssissä ensimmäisellä CCLK-pulssilla DONE-pinnin ylösnousun jälkeen vapautetaan 3-tilainen GTS-signaali (Global 3-state signal), jolloin mahdollistetaan piirin lähtöjen käyttö tarvittaessa. Seuraavalla CCLK:n kellopulssilla vapautetaan vielä yleinen kirjoitussignaali (GWE, Global Write Enable), joka sallii sisäisten muistielementtien vaihtaa tiloja logiikan ja kellopulssin mukaan. Näiden tapahtumien suhteellisia

ajoituksia voidaan muuttaa ohjelmiston konfigurointioptioilla. Lisäksi GTS- ja GWE-tapahtumat voidaan tehdä riippuviksi usean piirin DONE-pinnien ylösnousuista, jolloin voidaan suorittaa piirien synkronoitu käynnistys. [9]

2.1.2.2 Ulkoinen PROM- ja flash-piiri

FPGA-piirin ohjelmointitiedot voidaan säilyttää ulkoisella PROM- tai flash-piirillä, joka voi sisältää myös FPGA-piirillä olevan prosessorin BRAM-muistilohkojen alustuksen. Tällöin FPGA-piirin logiikka ja prosessorin ohjelmat voidaan päivittää sen sisältöä muuttamalla tai vaihtamalla koko piiri. Tällainen järjestelmä sopii käytettäväksi tosin ainoastaan silloin, kun FPGA:lle sulautetulla prosessilla ajettava ohjelmisto mahtuu kokonaisuudessaan BRAM-muistiin.

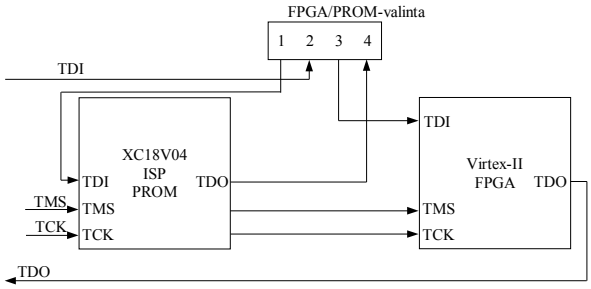
Taajuusmuuttajien muistivaatimukset ovat kuitenkin kasvaneet 1980-luvun alle 100 kt:n koosta 1990-luvun liki 500 kt:n kokoon asti [38]. Jos kehitys on jatkunut vähääkään saman suuntaisesti, niin nykyään muistia vaaditaan vähintään 1 Mt. Tällöin ulkoisen muistin käyttö on välttämätöntä ja eräänä vaihtoehtona onkin käyttää FPGA-piirin ohjelmointitietojen säilytyspaikan lisäksi prosessorilla suoritettavalle ohjelmalle omaa tallennuspaikkaa tai SystemACE-ratkaisua. SystemACE on Xilinx:n tarjoama ratkaisu, jolla voidaan suoraan alustaa FPGA-piiri ja BRAM-muisti sekä ulkoinen muisti (kuva 4).



Kuva 4. SystemACE-ratkaisulla voidaan alustaa FPGA-piiri ja BRAM-muisti sekä ulkoinen muisti kerralla.

PROM-piiriä käytettäessä, jos PROM-piiri kuuluu boundary-scan-ketjuun, sen sisältöä päästään helposti päivittämään ja muuttamaan piiriä irrottamatta. Kuvassa 5 on juuri tällainen järjestely, jossa Virtex-II FPGA-piiri ja XC8V04 PROM-piiri on liitettynä samaan boundary-scan-ketjuun. Virtex-II ja Spartan-3 FPGA-piirien kanssa

yhteensopivia PROM-piirejä (Programmable Read-Only Memory) ovat esimerkiksi Xilinx:n XC18V-sarjalaiset ja flash-piirejä ovat Xilinx:n XC-sarjalaiset.



Kuva 5. Virtex-II FPGA-piiri ja XC18V04 PROM-piiri liitettynä samaan boundary-scan-ketjuun. [13]

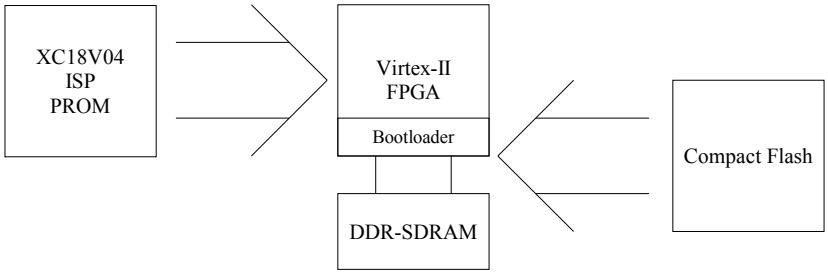
XC18V04 on sarjansa suurin PROM-muistipiiri ja pystyy tallentamaan 4,194,304 ohjelmointibittiä. Tämä muistipiiri on yhteensopiva Xilinx:n FPGA-piirien kanssa ja sopii kokonsa puolesta hyvin Virtex-II XC2V1000 piirin sekä Spartan-3 XC3S1000:n kanssa. Spartan-3 XC2S1000:ssa ohjelmoitavia bittejä on 2,223,488 kappaletta eli koko PROM-piirin kapasiteettiä ei tarvitse edes käyttää. Vastaavasti Virtex-II XC2V4000:nen tarvitsee näitä samoja muistipiirejä neljä kappaletta. Näiden piirien taataan kestävän 20,000 ohjelmointi/tyhjennys-kertaa ja datan säilyvän piirillä vähintään 20 vuotta. [11]

Flash-muistipiirit ovat kuitenkin syrjäyttäneet lähes kokonaan vanhentuneet PROM-piirit ja niiden hinta/koko-suhdekin on paljon parempi. Virtex-II ja Spartan-3 FPGA-piirien kanssa yhteensopivia flash-muistipiirejä ovat esimerkiksi Xilinx:n XCF-sarjan flash-piirit. Suuremmissa flash-piireissä on vielä Xilinx:n patentoima pakkausmenetelmä, jonka avulla on mahdollista tallentaa jopa 50 % enemmän tietoa. Taulukosta 5 näkyvät PROM- ja flash-piirien ominaisuudet. [12]

Taulukko 5. FPGA-piirin konfigurointiin tarkoitetut Xilinx:n PROM- ja flash PROM-piirit sekä niiden ominaisuudet. [10], [12]

PROM	Koko	JTAG-Ohjelmointi	Sarja-konfigurointi	Rinnakkais-konfigurointi	Pakkaus	Kotelo
XC18V512	512Kt	X	X	X		SOIC,VQFP,PLCC
XC18V01	1Mt	X	X	X		SOIC,VQFP,PLCC
XC18V02	2Mt	X	X	X		SOIC,VQFP,PLCC
XC18V03	4Mt	X	X	X		SOIC,VQFP,PLCC
Flash						
XCF01S	1Mt	X	X			VO20
XCF02S	2Mt	X	X			VO20
XCF04S	4Mt	X	X			VO20
XCF08P	8Mt	X	X	X	X	FS48
XCF16P	16Mt	X	X	X	X	FS48
XCF32P	32Mt	X	X	X	X	FS48

Jos suoritettavalle ohjelmalle käytetään omaa tallennuspaikkaa, niin ohjelmatietojen päivittäminen voidaan toteuttaa edellisten tavoin käyttäen boundary-scan-ketjua. Toisena vaihtoehtona on liittää muistipiiri FPGA-piiriin siten, että FPGA-piirillä oleva prosessori pääsee lukemaan sekä kirjoittamaan muistipiirille. Tällöin tietokoneeseen käytettyä liityntäväylää voidaan käyttää ohjelmointietojen siirtämiseen prosessorille ja prosessorilla olevalla ohjelmalla kirjoittaa se muistipiirille. Jos ohjelmaa vielä suoritettaisiin suoraan tallennuspaikastaan, niin FPGA-piirin muistisolut voitaisi vapauttaa ainoastaan data- ja välimuistiksi. ROM- ja flash-piirien hitauden vuoksi suoritettava ohjelma on kuitenkin kannattavampaa kopioida ennen suoritustaan nopeammalle dynaamiselle muistialueelle. Käytännössä tämä voidaan toteuttaa kuvan 6 mukaisesti eli FPGA-piirin logiikkaohjelmointitietojen yhteydessä alustetaan FPGA-piirin BRAM-muistisolut bootloader-ohjelmalla. Bootloader-ohjelma kopioi ROM- tai flash-piiriltä tiedot ulkoiselle muistille ja siirtyy suorittamaan varsinaista ohjelmakoodia sieltä.



Kuva 6. FPGA-piirii konfiguroidaan PROM-muistista, jolloin alustetaan samalla BRAM-muisti bootloader-ohjelmalla. Seuraavaksi bootloader-ohjelma siirtää varsinaisen suoritettavan ohjelman CF-muistilta (Compact Flash) ulkoiseen muistiin ja siirtyy ajamaan ohjelmaa sieltä.

FPGA-piirin, BRAM-muistin ja ulkoisen muistin alustamiseen käytettyjä SystemACE-menetelmiä on kolme: SystemACE CF, SystemACE MPM ja SystemACE SC. SystemACE CF käyttää erillistä CF-muistia, kun taas SystemACE MPM on yhden piirin ratkaisu. SystemACE SC on SystemACE MPM:stä uudistettu menetelmä, joka käyttää ohjelmistopohjaista ohjainratkaisua ja tarjoaa aivan samat palvelut, mutta käyttäen useampaa piiriä. Taulukkoon 6 on koottu näiden kolmen ohjelmointimenetelmän ominaisuudet, jotka seuraavaksi tarkemmin esitellään. [14]

Taulukko 6. System ACE FPGA-piirin ohjelmointiratkaisut, joiden avulla voidaan alustaa FPGA-piiri, FPGA-piirin BRAM-muistilohkot ja ulkoinen muisti. [15]

Ominaisuudet	System ACE CF	System ACE MPM	System ACE SC
Muistin koko	8Gbit asti	16Mbit, 32Mbit, 64Mbit	16Mbit, 32Mbit, 64Mbit
Komponenttien lukumäärä	2	1	3
Tilavaatimus vähintään	25 cm^2	12.25 cm^2	Valittavissa
Pakkaus	Ei	Kyllä	Kyllä
FPGA:n konfigurointi-muoto	JTAG	SelectMAP (enintään 4 FPGA:ta), Slave-Serial (Enintään 8 FPGA:ta)	SelectMAP (enintään 4 FPGA:ta), Slave-Serial (Enintään 8 FPGA:ta)
Suurin konf. nopeus	30 Mbit/sec	152 Mbit/sec	152 Mbit/sec
Suurin kokoonpanojen määrä	Rajoittamaton	8	8

SystemACE CF on hyvin joustava ja paljon tallennustilaa tarjoava kahden piirin ratkaisu, jossa voidaan käyttää CompactFlash-muisteja ja IBM MicroDrive kovalevyjä aina 8 Gt kokoon asti. Erillisen CompactFlash-muistikortin avulla järjestelmän muuttaminen ja päivittäminen on helppoa, koska sen sisältö voidaan vaihtaa joko piirin ollessa laitteessa kiinni tai irrotettuna. Muistipiirin päivittäminen sen laitteessa kiinni ollessa on mahdollista suorittaa myös verkko-yhteyden avulla. Muistikortin lisäksi tarvitaan ACE-ohjainpiiri, joka sisältää ohjelmointilogiikan monipuoliset liitännät. Tällä menetelmällä voidaan ohjelmoida monia FPGA-piirejä sisältäviä järjestelmiä boundary-scan-ketjun ansiosta sekä voidaan säilyttää useita eri ohjelmointiversioita. Muistipiirille voidaan tallentaa ohjelmointitietojen lisäksi kaikkea muutakin tietoa, kuten julkaisutietoja, versiohistoriaa, ohjeita käyttäjälle ja vastauksia usein kysyttyihin kysymyksiin. [14]

SystemACE MPM-ratkaisussa käytetään yhdelle piirille integroituja FPGA-piiriä (Virtex-E), PROM-muistipiiriä ja AMD Flash-muistipiiriä. PROM-muistipiiri toimii ohjelmointitietojen ohjaukseen käytetyn FPGA-piirin ohjelmointiin. Tämän FPGA-piirin tehtävänä on ohjelmoida varsinainen kohdejärjestelmä flash-muistista. Kohdejärjestelmän ohjelmointimahdollisuuksia ovat Slave Serial ja SelectMAP. Slave Serial-ohjelmointitavalla voidaan ohjelmoida yksi, kaksi, neljä tai kahdeksan FPGA-ketjua. SelectMAP-ohjelmointitavalla voidaan ohjelmoida vain enintään neljä eri FPGA-piiriä, mutta se on huomattavasti nopeampi rinnakkaisen tiedonsiirron ansiosta. Ohjelmointitiedot voidaan säilyttää myös pakattuna eli jos esimerkiksi Virtex-II XC2V8000 tarvitsee normaalisti 26 Mt ohjelmointitietoja, niin pakkauksen ansiosta sille riittää kuitenkin 16 Mt:n tallennustila. [14], [15]

System ACE SC on ladattava versio SystemACE MPM-ohjaimesta, joka on toteutettu käyttäen FPGA-piiriä. Tämä FPGA-piiri ohjelmoidaan käyttäen omaa ohjelmointimuistiaan. Kohdejärjestelmän ohjelmointi tapahtuu siis vastaavasti kuin SystemACE CF-ratkaisussa, mutta käyttäen vain FPGA-piiriä ohjaimena. [14]

2.2 FPGA:lle sulautetut ohjelmistoprosessorit

FPGA-piirille sulautetut ohjelmistoprosessorit ovat ominaisuuksiltaan kuin sisäiseltä

rakenteeltaankin hyvin samanlaisia kuin tavalliset prosessorit. Perusrakenteesta löytyvät niin aritmeettislooginen yksikkö (ALU), prosessorin sisäiset väylät ja kuin rekisteritkin.

FPGA-piirin resurssit ovat kuitenkin rajalliset, joten piirin logiikasta rakennetun prosessorin ydin kannattaa valita sovelluksen mukaisesti oikein. Vaikka ohjelmistoprosessori onkin hyvin mukautuvainen, niin ytimen rakennetta ei ainakaan kaupallisilla prosessoreilla ole aina välttämättä mahdollista muokata. Tämä on eräs syy siihen miksi esimerkiksi Xilinx on kehittänyt MicroBlaze:sta pienemmän mikrokontrolleritasoisen ohjelmistoprosessorin nimeltään PicoBlaze.

Muutkaan FPGA-piirien valmistajat eivät ole jättäytyneet kisasta ja tarjoavatkin Xilinx:n ohjelmistoprosessoreille kovan vastuksen. Nios-II on Alteran tehokkain ohjelmistoprosessoriperhe ja se pystyy käyttämään Alteran FPGA-piirin resurssit mahdollisimman tehokkaasti. Ohjelmistoprosessoreita on saatavana myös avoimina eli niiden lähdekoodit ovat saatavissa vapaasti muokattavaksi ja tällaisten avoimien prosessorien määrä vain kasvaa koko ajan. [16]

2.2.1 MicroBlaze

MicroBlaze on edullinen 32-bittinen prosessori Virtex™ ja Spartan™-II/3 FPGA-piireille. MicroBlaze on Harvard-tyylinen RISC-arkkitehtuurinen prosessori, jonka erilliset 32-bittiset käsky- ja data-väylät mahdollistavat ohjelman suorituksen sekä datan käsittelyn niin sisäisestä kuin ulkoisestakin muistista. Ytimen tehokkuuden ansiosta sitä voidaan käyttää hyvin monimutkaisten järjestelmien toteuttamiseen kuten tietoliikenne-, verkko-, sulautettuihin- ja kuluttajamarkkinoiden laitteisiin. FPGA-piiriltä MicroBlaze vie resursseja noin 950 logiikkasolua ja pystyy parhaimmillaan 123 DMIPS:n lukemiin 150 MHz:llä toimivalla Virtex-II Pro FPGA-piirillä. Yksi logiikkasolu koostuu hakutaulusta, kiikusta ja lisäksi pienestä määrästä lisälogiikkaa. MicroBlaze yhdistääkin kaikki ohjelmistoprosessorin joustavat ominaisuudet. [17]

2.2.2 PicoBlaze

PicoBlaze soveltuu loistavasti sulautettuihin järjestelmiin, joissa tarvitaan monimutkaisempia tilakoneita. PicoBlaze on pieni 8-bittinen ohjelmistopohjainen

mikrokontrolleri, joka on yhteensopiva Virtex™ ja Spartan™ FPGA-piirien kanssa. Kontrolleri sisältää 57 kappaletta 16- ja 18-bittisiä käskyjä, 8-32 yleiskäyttöistä 8-bittistä rekisteriä, 256 suoraan ja epäsuorasti osoitettavaa porttia sekä päälle kytkettävän (maskable) keskeytyksen. Verrattuna muihin kaupallisiin mikrokontrollereihin PicoBlaze on erittäin tehokas ja tuottaakin 100 MIPS:iä 200 MHz:n Virtex-II Pro™ FPGA-piirillä. FPGA-piirillä resurssien kulutus on hyvin vähäistä eli se tarvitsee vain 192 logiikkasolua. Esimerkiksi Spartan-3 XC3S200 FPGA-piirin resursseista PicoBlaze kuluttaa vain 5 %.

Ohjelmamuistina PicoBlaze käyttää yleensä vain yhtä BRAM-lohkoa (konfiguroitu 1Kx18-lohkoksi), joten Spartan-3 ja Virtex-II/Pro piireillä ohjelmamuistia on siten vain 1024 käskyn verran. Pienuuden vuoksi sitä voidaankin helposti liittää FPGA-piireille useita kappaleita, jolloin suuremmat tehtävät voidaan jakaa pienempiin tehtäviin tai rajata tehtävät vain toisistaan erillisiksi. Taulukossa 7 on PicoBlaze:n ominaisuudet ja nopeudet eri FPGA-piireillä. [18]

Taulukko 7. Xilinx:n ohjelmistomikrokontrollerin PicoBlaze:n ominaisuudet ja nopeudet eri FPGA-piireillä. [18]

	PicoBlaze	
Ominaisuus	Spartan-3 ja Virtex-II/Pro	Virtex-E ja Spartan-IIE
Ohjelman koko	1024 käskyä	256 käskyä
Käskysanan koko	18-bittiä	16-bittiä
Sisäinen ohjelma	Kyllä	Kyllä
8-bittisiä rekistereitä	16	16
Pinon syvyys	31	15
Koko	96 Spartan-3 siivua	76 Spartan-IIE siivua
Tehokkuus	44 MIPS (Spartan-3) ja 100 MIPS (Virtex-II PRo	37 MIPS (Spartan-IIE)

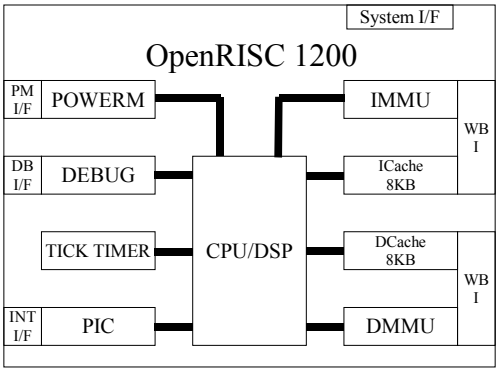
PicoBlaze on saatavana myös Roman-Jones Inc. toimesta 8051 mikrokontrolleripiiriperheeseen kuuluvan 8031 piirin kanssa yhteensopivaksi. Tämän laajennuksen avulla PicoBlaze kykenee ajamaan 8051:lle tehtyjä ohjelmia suoraan. Emuloinnin ansiosta käskykanta on täysin sama, mutta niiden suoritukseen vaadittavat kellojaksot eivät kuitenkaan täysin täsmää. [19]

2.2.3 OpenRISC

OpenRISC on projekti, jonka tarkoituksena on luoda ilmainen avoimeen lähdekoodiin perustuva 32/64-bittinen RISC/DSP-arkkitehtuurinen ohjelmistoprosessori GNU (L) GPL lisenssin alaisena. Projektin prosessorin on tarkoituksena olla yleiskäyttöinen ja monipuolinen sekä sitä on pystyttävä käyttämään eri kohdepiireillä. Tarkoituksena on myös kehittää ja käyttää avoimia prosessorin kehitystyökaluja, prosessorilla ajettava käyttöjärjestelmä, sovelluksia ja kirjastoja. Tosin OpenRISC-projekti ei estä kolmannen osapuolen käyttävän prosessoria ja työkaluja omissa kaupallisissa sovelluksissaan tai lisätä omiin sovelluskehittimiin tukea kyseiselle prosessorille.

OpenRISC 1000 on OpenRISC-projektin 32/64-bittinen load/store RISC-arkkitehtuurinen prosessoriperhe käytettäväksi esimerkiksi verkko-, sulautettuihin-, ajoneuvo-, kannettaviin- ja langattomiin sovelluksiin. OpenRISC 1000 arkkitehtuurinen prosessori, OpenRISC 1200, on 32-bittinen skalaarinen RISC-tyyppinen ja Harvard-arkkitehtuurinen prosessorin ydin. OpenRISC 1200-prosessorissa on viisitasoinen liukuhihna, virtuaalimuistituki (MMU) ja joitain DSP-ominaisuuksia.

OpenRISC 1200-prosessorin ydin koostuu useasta modulaarisesta yksiköstä kuvan 7 mukaisesti. Kuvasta keskeltä löytyy tehokas 32-bittinen CPU/DSP-ydin, joka sisältää ORBIS32-käskykannan, DSP 32x32 MAC-käskyn (Multiply-accumulate), omien käskyjen lisäämismahdollisuuden, 32 kappaletta 32-bittistä rekisteriä ja skalaarisen 5-tasoisen liukuhihnan. DSP 32x32 MAC-käsky on signaaliprosessoreiden käyttämä tehokas ratkaisu kerto- ja yhteenlaskun yhdistelmään. Liukuhihnan ansiosta suurin osa käskyistä pystytään suorittamaan yhdellä kellojaksolla eli parhaimmillaan saavutetaan 250 MIPS:n tehokkuus 250 MHz:llä. Ytimen ja keskeytysten suoritusajat ovat ennustettavia ja keskeytyksiin vastaaminen nopeaa, joten prosessori soveltuu erittäin hyvin koviin reaaliaikasovelluksiin.



Kuva 7. OpenRISC-projektin OpenRISC-1200 prosessorin ytimen rakenne. [20]

Ytimeen kuuluu L1-tason käsky- ja datavälimuistia 1-64 kb, joiden hallintaan on myös omat erityiskäyttöiset rekisterit. Ytimessä on myös muistinhallintayksikkö (MMU, Memory Management Unit), joka on jaettu käsky- ja datamuisteille erikseen Harvard-arkkitehtuurin mukaisesti. Muistinhallintayksikössä on käsky- ja datamuisteille 16-256 yksikön kokoinen suoraan muistiin viittaava (direct mapped) hash-pohjainen käännöstaulu (TLB, Translation look-aside buffer). Tehonhallintayksikön avulla ydin voi vähentää tehonkulutusta esimerkiksi ohjelmistosta käsin säädettävän kellotaajuuden, keskeytyksellä sleep-tilasta heräämisen ja eri toiminnallisten lohkojen omien kellotaajuksien avulla.

Tehokas debuggaus-yksikkö mahdollistaa haitattoman prosessorin tai koko järjestelmän debuggauksen sekä seuraamisen. Yhdistetty kellokeskeytyslaskuri mahdollistaa reaaliaikakäyttöjärjestelmän taskien aikatauluttamisen ja tarkan ajanmittauksen. Ohjelmoitava keskeytyskäsittelijä sisältää kaksi (unmaskable) keskeytyslähdettä, 20 päällekytkettävää (maskable) keskeytyslähdettä ja näille kaksi eri prioriteettitasoa. Ytimeen on mahdollista lisätä 8 omaa yksikköä, esimerkiksi liukulukuyksikkö, joita voidaan hallita erityiskäyttöisten rekistereiden tai omien käskyjen avulla. [20]

OpenRISC 1200-prosessorin perusjärjestelmä vie Xilinx:n Virtex FPGA-piiriltä 3000 siivua toimiessaan 33 MHz:llä, joten se tarvitsee resursseja enemmän kuin MicroBlaze. [21]

2.2.4 Nios-II

Nios-II on ohjelmistoprosessoriperhe Alteran Stratix II, Stratix, Cyclone II, Cyclone ja HardCopy-piiriperheille. Nios-II sisältää kolme erilaista prosessoriydintä, jotka parhaillaan tuottavat 200 DMIPS:iä ja joihin on saatavana yli 60 IP-lisälaitelohkoa. Nämä kolme prosessoria ovat nimetty seuraavasti:

- Nios II/f: Nopein
- Nios II/e: Taloudellisin
- Nios II/s: Perus

Nios II/f-ydin on suunniteltu mahdollisimman nopeaksi. Nopeutta on saatu ytimen koon kustannuksella ja se onkin noin 35 % suurempi kuin Nios-II/s-perusydin. Suoritusnopeuden parantamisessa on keskitytty kahteen asiaan: tehokkuuteen suorittaa käsky jokaisella kellojaksolla ja saada ydin toimimaan mahdollisimman suurella kellotaajuudella. Tälläinen ydin soveltuu parhaiten tehokkuutta vaativiin sekä suuria ohjelmakoodi- ja datamääriä sisältäviin sovelluksiin.

Nios-II/e-ydin on suunniteltu taloudellisimmaksi niin, että se olisi mahdollisimman pieni kooltaan. Tällä ytimellä on ainoastaan yksi suunnittelukriteeri eli pyrkiä minimoimaan resurssien käyttö kaikilla mahdollisilla tavoilla, mutta silti pyrkiä pitämään yhteensopivuus Nios II käskykannan kanssa. Laitteistoresurssien säästö on saatu aikaan suoritusnopeuden kustannuksella ja näin ytimen koko on kuitenkin saatu melkein puoleen Nios II/s-perusytimestä. Tällainen ydin soveltuu parhaiten kustannuskriittisiin sovelluksiin sekä sovelluksiin jotka tarvitsevat vain yksinkertaista ohjauslogiikkaa.

Nios II/s-ydin pyrkii toteuttamaan pienen prosessoriytimen tinkimättä merkittävästi ohjelmiston suoritustehokkuudesta. Tällainen ydin käyttää noin 25 % vähemmän logiikkaa kuin Nios-II/f-ydin, mutta samalla suoritusnopeus on kuitenkin yli 400 % suurempi kuin Nios-II/e-ytimellä. Tämä nopeus/hinta-suhde on saatu aikaan sopivilla laitteistotason ominaisuuksien poistoilla. Tämä ydin soveltuukin parhaiten hintakriittisiin ja keskiluokan tehokkuutta vaativiin sovelluksiin. [22]

Kolmen eri Nios-II-ytimen ominaisuudet on kerätty taulukkoon 8. Taulukosta nähdään edellä käydyt asiat eli laitteistotason ominaisuuksia karsimalla päästään pienempään resurssien kulutukseen, mutta samalla kärsii myös suoritusnopeus. Taulukossa kokoarvio on annettu Alteran LE (Logic Element) arvona, joka vastaa Xilinx:n logiikkasolua. Nios-II/s-ytimessä on ohjelmahypyn staattinen ennakointi, jolloin hyppyjen ennakointi tapahtuu jo ohjelman käännösvaiheessa. Nios-II/f-ytimessä on taas lisälogiikan ansiosta dynaaminen hypyn ennakointi, eli se ennakoi ohjelmahyppyjä suorituksen aikana. [22],[23]

Taulukko 8. Nios-II-ohjelmistoprosessoriperheen ytimien ominaisuudet. Prosessorin kokoarvio on annettu Alteran LE-lukuna, joka vastaa Xilinx:n logiikkasolua. [22]

Ominaisuudet		Nios II/e	Nios II/s	Nios II/f
Tehokkuus	DMIPS/MHz	0.16	0.75	1.17
	Max DMIPS	28	120	200
	f_{MAX}	150 MHz	135 MHz	135 MHz
Kokoarvio (LE)		<600	<1,300	<1,800
Liukuhihna		-	5	6
Ulkoinen muistiavaruus		2 Gt	2 Gt	2 Gt
Käskyväylä	Välimuisti	-	0.5 - 64 Kt	0.5-64 Kt
	Hypyn ennakointi	-	Staattinen	Dynaaminen
Dataväylä	Välimuisti	-	-	0.5-64 Kt
ALU	Laitteisto kertolasku	-	3 kellojaksoa	1 kellojakso
	Laitteisto jakolasku	-	-	Valinnainen
	Bittisiirto	1 kellojakso/ bitti	3:n kellojakson	yhden kellojakson
JTAG debuggaus-moduuli		Kyllä	Kyllä	Kyllä
Poikkeusten käsittely		Kyllä	Kyllä	Kyllä
Omien käskyjen tuki		256	256	256

2.3 MicroBlaze-ohjelmistoprosessorin rakenne ja ominaisuudet

Prosessorin ydin sisältää 32 kappaletta 32-bittistä yleiskäyttöistä rekisteriä ja kaksi 32-bittistä erityiskäyttöistä rekisteriä. Kolmeakymmentäkahta 32-bittistä rekisteriä kutsutaan GPR:ksi (GPR, General Purpose Register), jotka ovat numeroitu R0:sta R31:een. Osalla näistä yleiskäyttöisistä rekistereistä on tosin myös eristyiskäyttökohteita, jotka esitellään tarkemmin kappaleen 2.3.4 ABI (Application Binary Interface) yhteydessä ja taulukossa 8.

Erityiskäyttöisiä rekistereitä ovat ohjelmalaskuri (PC, Program Counter) ja prosessorin tilarekisteri (MSR, Machine Status Register). Ohjelmalaskuri sisältää 32-bittisen

osoitteen suorituksessa olevaan käskyyn. Prosessorin tilarekisteri sisältää ylivuotolipun, keskeytysten kieltämisen ja sallimisen, mahdollisuuden väylän lukitsemiselle sekä tiedon välimuistin ja FSL:n virhetoiminnasta (FSL, Fast Simplex Link). FSL on nopea väylärakenne, joka esitellään kappaleessa 2.4.2 tarkemmin.

Jo prosessorin rekisterien määrästä voidaan päätellä, että kyseessä on ns. load-store-arkkitehtuuri [23]. Load-store-arkkitehtuurissa aritmetiikka tapahtuu rekistereissä, joita on useita ja muistia käsitellään ainoastaan load- (*lb**, *lh** ja *lw**) sekä store- (*sb**, *sh**, *sw**) käskyillä. Koko prosessorin käskykanta on esitetty liittessä 1. Käskyt ovat kolmioperandisia eli esim. käsky

```
lw r1, r2, r3
```

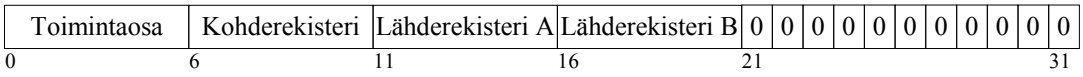
tallentaa rekisterien *R2* ja *R3* arvojen yhteenlasketun summan osoittamasta muistiosoitteesta sanan kokoisen arvon rekisteriin *R1*. Muistiin voidaan osoittaa käyttäen kolmea datakokoa, tavua (8 bittiä), puolisanaa (16 bittiä) ja sanaa (32 bittiä). Koska muisti koostuu tavun kokoisista lohkoista, osoitettaessa käytetään aina datakoon mukaista kohdistusta eli puolisanaa käytettäessä muistiosoitteen vähiten merkitsevä bitti pakotetaan nollaksi ja vastaavasti sanaa käytettäessä kaksi osoitteen vähiten merkitsevää bittiä.

MicroBlaze on Big-Endian tyyppinen prosessori, joten osoitteet ja indeksointi ovat kuvan 8 mukaisia. Kuvassa ylimmällä rivillä on tavun osoite ja sen alapuolella indeksi eli yhdessä sanassa on neljä tavua indeksoituna nollasta kolmeen. Eniten merkitsevät tavut ovat pienemmissä osoitteissa ja vähiten merkitsevät suuremmissa, joka on juuri päinvastoin kuin Little-Endian-järjestelmissä. Puolisanan ja tavun tapauksessa bittien indeksointi ja merkitsevyys menee vastaavasti kuin sanallakin.

	SANA			
Tavun osoite	n	n+1	n+2	n+3
Tavun indeksi	0	1	2	3
Tavun merkitsevyys	Eniten			Vähiten
Bitin indeksi	0			31
Bitin merkitsevyys	Eniten			Vähiten

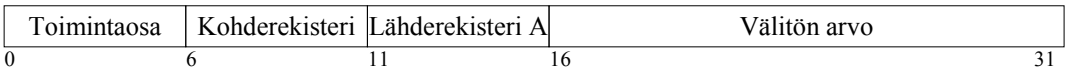
Kuva 8. MicroBlaze on Big-Endian tyyppinen prosessori, joten osoitteet ja indeksointi menevät kuvan esittämällä tavalla sanan tapauksessa. Tavun ja puolisanan kanssa tilanne on vastaava. [24]

Käskymuotoja on kaksi: muoto A ja muoto B. Muoto A sisältää toimintaosan, yhden kohde- ja kaksi lähderekisteriä. Käskyn loppuosa on nollia, koska käskysanan pituuden on aina oltava vakion pituinen. Esimerkiksi edellä esitetty muistinosoituskäsky on juuri tämän muotoinen ja kuva 9 esittää siitä yleisen muodon. Tätä muotoa käytetään rekisteri-rekisteri-käskyissä.



Kuva 9. MicroBlaze:n käskymuoto A, joka sisältää toimintaosan, yhden kohde- ja kaksi lähderekisteriä. Tätä muotoa käytetään rekisteri-rekisteri-käskyissä. [24]

Toinen käskymuoto on muoto B, jota käytetään välittömissä rekisterikäskyissä. Käsky sisältää toimintaosan, yhden kohde- ja yhden lähderekisterin sekä 16-bittisen välittömän arvon, kuten kuva 10 esittää.

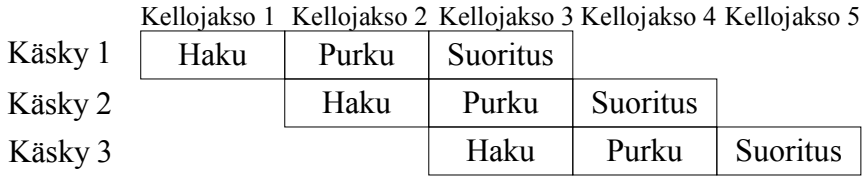


Kuva 10. MicroBlazen:n käskymuoto B, joka sisältää toimintaosan, yhden kohde- ja yhden lähderekisterin sekä 16-bittisen välittömän arvon. [24]

MicroBlaze:n ydin on Harvard-arkkitehtuurin mukainen eli sillä on erilliset liityntäyksiköt datalle ja käskyille. Väylien liityntäyksiköt ovat jaettu paikalliseksi muistiväyläksi (LMB, Local Memery Bus) ja IBM:n lisälaitteiden liityntäväyläksi (OPB, On-Chip Peripheral Bus). LMB:tä käytetään lähinnä FPGA-piirillä olevien kaksiporttisten muistilohkojen käyttämiseen ja se tarjoaa mahdollisuuden käsitellä muistia kahdella kellojaksolla. OPB-liityntä tarjoaa yhteyden niin FPGA-piirin sisäisille kuin ulkoisillekin lisälaitteille ja muisteille. Näiden lisäksi prosessorin ytimessä on mahdollisuus käyttää nopeaa väylää, joka sisältää 8 tulo- ja 8 lähtöliityntää. Nämä FSL-väylät ovat yksisuuntaisia jakamattomia kanavia lähinnä tietoliikennekäyttöön ja rinnakkaisesti toimiviin laitteistotason ohjelmistolaajennuksiin.

MicroBlaze:ssa on rinnakkainen liukuhihna, joka on jaettu kolmeen osaan: haku, purku, ja suoritus. Yleensä jokainen näistä osista suoritetaan yhdellä kellojaksolla. Vastaavasti

yhden käskyn suorittaminen kestää siten kolme kellojaksoa, jos viiveitä tai pysähdyksiä ei oteta huomioon. Liukuhihnan ansiosta kokonaissuorituksena saadaan kuitenkin parhaimmillaan jokaisella kellojaksolla suoritettua yksi käsky. Kuvassa 11 on esitetty liukuhihnan eri vaiheet sekä sen läpi kulkevien käskyjen tarvitsemat kellojaksot.



Kuva 11. MicroBlaze:ssa on rinnakkainen kolmeosainen liukuhihna, joka sisältää osat: haku, purku ja suorittaminen. Jokaisen osan suorittaminen kestää yhden kellojakson. [24]

Prosessorin käynnistyksen tai resetoimisen jälkeen ohjelman suoritus aloitetaan muistiosoitteesta 0. Ohjelmalaskuri (PC) ja prosessorin tilarekisteri (MSR) ovat tällöin alustettu oletusarvoihinsa. Prosessori voi oletuksena ottaa vastaan kolmenlaisia keskeytyksiä: keskeytykset (interrupt), poikkeukset (exception) ja katkaisut (break). Ainoaan ulkoiseen keskeytyslinjaan prosessori reagoi suorittamalla liukuhihnan suoritusvaiheessa olevan käsky loppuun ja samalla purkuvaiheeseen vaihdetaan hyppykäsky keskeytysvektorin osoitteeseen 0x10. Keskeytyksen paluuosoite, eli osoite siihen käskyyn joka oli liukuhihnan purkuvaiheessa, tallennetaan automaattisesti rekisteriin R14. Myös tulevat keskeytykset kielletään nollaamalla tilarekisteristä IE-bitti (Interrupt Enable). Keskeytyksiä ei palvella, jos katkaisu on käynnissä eli tilarekisterin BIP (Break In Progress) on asetettuna.

Poikkeuksia prosessori palvelee samalla tavoin kuin keskeytyksiäkin, mutta käyttäen rekisteriä R17 tallentamaan ennen poikkeuksen tapahtumista seuraavan suoritettavan käskyn osoite ja siirtymällä suorittamaan poikkeusvektoria osoitteesta 0x8.

Katkaisuja voi tulla ohjelmasta käsin sekä ulkoisesti päälle kytkettävältä (maskable) ja aina päällä olevalta (non-maskable) linjalta, joista ensimmäiseen reagoimista voidaan muuttaa tilarekisterin BIP bitillä. Ulkoisen katkaisun tultua prosessori tallentaa seuraavan suoritettavan käskyn osoitteen rekisteriin R16 ja siirtyy suorittamaan ohjelmakoodia katkaisuvektorin osoitteesta 0x18. Ohjelmasta *brk*- tai *brki*-konekielikäskyillä suoritetuille katkaisuille annetaan osoite, minne siirtyä sekä rekisteri,

jonne tallennetaan seuraavan suoritettavan käskyn osoite.

Luettaessa dataa, joka sijaitsee LMB-muistialueen ulkopuolella, voidaan MicroBlaze:ssa käyttää valinnaista välimuistia parantamaan suorituskykyä. Välimuistilla on seuraavanlaiset perusominaisuudet:

- Suoraan keskusmuistiin viittaavaa (direct mapped)
- Läpikirjoittava
- Käyttäjän valitsema muistialue, johon välimuistia käytetään
- Säädettävä välimuisti sekä kirjaustaulukon koko
- Mahdollisuus lukita välimuistilinjat erikseen
- Välimuisti voidaan asettaa päälle ja pois MSR-rekisteristä
- Käskyt joilla voi kirjoittaa välimuistiin
- Ei vaadi erityistä muistinhallintayksikköä ja toimii OPB-lisälaitteiden kanssa
- Muisti voidaan jakaa välimuistillisiin ja välimuistittomiin segmentteihin

MicroBlaze tarjoaa debuggausliitynnän, joka tukee JTAG-pohjaisia debuggaustyökaluja, jotka tunnetaan myös paremmin nimellä BDM-debuggerit (Background Debug Mode). BDM-debuggerien tapaan Xilinx:n mikroprosessorin debuggaustyökalu (XMD, Xilinx Microprocessor Debug) on suunniteltu liitettäväksi mikroprosessorin debuggausmoduuliin (MDM, Microprocessor Debug Module). Tämä MDM-IP-ydin toimii liityntärajapintana JTAG-portin ja Xilinx:n FPGA-piirin välillä. MDM-moduuli tukee myös useamman MicroBlaze-prosessorin yhtäaikaista debuggausta. Debuggerilla on seuraavanlaisia ominaisuuksia:

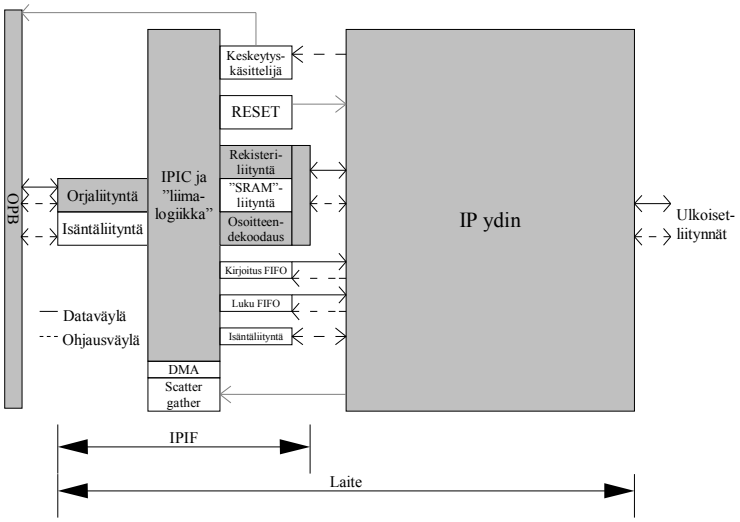
- Laitteistokeskeytyspisteiden ja katselupisteiden määrä valittavissa sekä rajoittamaton määrä ohjelmistokeskeytyspisteitä
- Mahdollista pysäyttää, resetoida ja askeltaa käsky kerrallaan
- Muistin ja kaikkien rekistereiden (myös PC ja MSR) luku ja kirjoitus
- Käsky- ja datavälimuistille kirjoittaminen

MicroBlaze:ssa on mahdollisuus käyttää laitteistokertojaa ja -jakajaa. Myös laitteistopohjaista kahdella kellojaksolla toteuttavaa useamman bitin mittaista bittisiirtoa (Barrel Shifter), voidaan käyttää nopeuttamaan bittien siirtoja ja pyörityksiä. [24] Barrel Shifter on todella tarpeellinen esimerkiksi käytettäessä kiinteän pilkun aritmetiikkaa.

2.3.1 OPB-väylä

OPB-väylä on osa IBM:n kehittämästä CoreConnect-arkkitehtuurista, joka on tarkoitettu yleiskäyttöiseksi väyläksi. OPB-väylästä Xilinx on räätälöinyt oman version helpottamaan lisälaitteiden liittämistä FPGA-piirillä olevaan prosessoriin. Väylän runsaat ominaisuudet tekevät siitä varsin tehokkaan ja näitä ominaisuuksia voidaankin käyttää FPGA-piirillä suurimmaksi osaksi. Koska kaikkia näitä ominaisuuksia ei ole aina tehokasta käyttää, Xilinx on ottanut käyttöön vain osia OPB:stä, mutta FPGA-piirien ohjelmoitavan luonteen vuoksi väylä on mahdollista ottaa käyttöön myös kaikkine ominaisuuksineen.

Omien IP-lohkojen liittämistä OPB-väylään on helpotettu IPIF:n (Intellectual Property Interface) avulla (kuva 12). IPIF toimii rajapintana OPB-väylän ja oman IP-lohkon välillä siten, että oma IP-lohko näkee vain IPIC (IP Interconnect) signaalit. IPIF on hyvin modulaarinen ja mukautuvainen, joten esimerkiksi kuvassa 12 on esitetty harmaalla yksinkertaisen orja-tyyppisen rekisteripohjaisen lisälaitteen liittämiseen tarvittavat osat ja muut ovat valinnaisia. Valinnaisia toimintoja ovat esimerkiksi keskeytyksen käyttö, IP-ytimen resetointi, FIFO-puskuri sekä DMA- ja Scatter Gather tiedonsiirtolaajennukset. [25]



Kuva 12. OPB-väylän ja IP-lohkon IPIF-rajapinnan rakenne. IPIF:n avulla omien IP-lohkojen liittäminen sulautettuun prosessoriin on helpompaa yksinkertaisemman IPIC-liityntärajapinnan ansiosta. IPIF on hyvin modulaarinen ja mukautuvainen, joten kuvassa on esitetty harmaalla tarvittavat osat rekisterityyppiselle lisälaitteelle. [25]

2.3.2 FSL-väylä

MicroBlaze sisältää kahdeksan FSL-tulo- ja kahdeksan FSL-lähtöliityntää. Näitä FSL-väyliä voidaan käyttää yksisuuntaiseen pisteestä pisteeseen toimivaan datan jatkuva-aikaiseen siirtoon. FSL-liitynnät ovat 32-bittiä leveitä ja niitä voidaan käyttää lähettämään tai vastaanottamaan niin ohjaus- kuin datasanojakin. Siirretty sana sisältää erityisen bitin, joka kertoo onko data ohjaustietoa vai dataa. MicroBlaze:ssa on myös neljä erityistä käskyä, joilla voidaan lukea dataa tai ohjaustietoa FSL-kanavilta prosessorin rekistereihin. [24]

OPB-väylän lisäksi lisälaitelohkoja voidaan liittää aikakriittisiin sovelluksiin paremmin soveltuvan FSL-väylän kautta. FSL-väylän etuna OPB-väylään verrattuna on sen yksinkertaisuus eli vältytään yleiskäyttöiseksi suunnitellun väyläprotokollan vaatimilta käsittelyajoilta. FSL-väylä soveltuu erinomaisesti ohjelmistoa nopeuttavien laitteistotason toteutusten liittämiseen ja se on jopa parempi ratkaisu kuin erityisten omien käskyjen lisääminen suoraan prosessorin ytimen käskykantaan. Koska nykyisten RISC-tyyppisten prosessoreiden suoritinyksiköissä on vain kaksi operandia ja yksi paluuarvo, niin useamman operandin ja paluuarvon vaativien tehtävien suorittamiseen tarvittaisiin useiden käskyjen sarja. Omien käskyjen lisäämisessä ytimen käskykantaan liittyy myös toinen ongelma. Jos käskyjen suorituksen kriittinen polku kulkee lisätyn käskyn lävitse, niin se voi vaikuttaa koko prosessorin suorituskykyyn. Näiden ongelmien ratkaisuksi FSL-väylän kautta voidaan tehokkaasti hyödyntää rinnakkaisesti toimivaa laitteistotason toteutusta nopeuttamaan ohjelmiston suoritusta.

FSL-väylään liitetyt lisälaitteet voivat olla joko isäntiä tai orjia. Isännäksi määritelty lisälaite ohjaa FSL-väylän data- ja ohjaussignaaleja. Vastaavasti orjaksi määritellyt lisälaitteet vain lukevat väylältä data- ja ohjaussignaaleja. MicroBlaze:n FSL-väylän luku- ja kirjoituskäskyillä voidaan siirtää tietoa FSL-väylältä prosessorin rekistereihin sekä rekistereiltä väylälle. [24]

2.3.3 Muistit

MicroBlaze voi käyttää FPGA:n SelectRAM-lohkoja nopeaan paikalliseen data- ja käskymuistiin tai välimuistina nopeuttamaan hitaamman ulkoisen muistin käsittelyä.

Valmiissa lisälaitevalikoimassa on mahdollisuus ottaa käyttöön myös ulkoisia SRAM- ja Flash-pohjaisia muisteja. EDK sisältää VHDL-kieliset lähdekoodit muistinhallintayksiköille ja niistä voidaan myös muokata muistinhallintayksiköitä muisteille, joita Xilinx ei vielä tue. [26]

Sisäisen BlockRAM-muistin käsittely LMB-väylän kautta vaatii 2 kellojaksoa niin lukemiseen kuin kirjoittamiseenkin. BlockRAM-muistista kirjoittaminen OPB-väylän kautta vaatii 3 kellojaksoa ja lukeminen 4 kellojaksoa. Ulkoisesta muistista lukeminen tarvitsee OPB-väylän kautta 5-7 kellojaksoa, joiden lisäksi viivettä voi tulla lisää vielä väylän välitysoperaatioidenkin johdosta. [11]

2.3.4 ABI (Application Binary Interface)

ABI (Application Binary Interface) määrittelee prosessorin resurssien yhteiset sovitut käyttötavat ja sen tunteminen on tärkeää etenkin konekielellä ohjelmoinnissa sekä reaaliaikakäyttöjärjestelmää siirrettäessä. Myös suoritettavan ohjelmakoodin jakamisessa eri muistialueille linkkausohjeita muokaten täytyy tietää muistinkäsittelyn yhteisistä muistin ja rekisterien käytön sopimuksista. MicroBlaze:n GNU-kääntäjätkin noudattavat juuri näitä samoja määrittelyjä.

MicroBlaze:n konekieliohjelmilla käytetyt datatyypit ovat 8-, 16- sekä 32-bittiä ja niitä nimitetään tavuksi, puolisanaksi ja sanaksi. Taulukkoon 9 on kerätty konekielellä käytetyt datatyypit, ANSI C:n vastaavat tietotyypit ja niiden koot tavuina. Taulukosta nähdään, että konekielistä 8-bittistä datatyyppiä vastaa C-kielessä char-tietotyyppi, 16-bittistä short-tietotyyppi sekä 32-bittistä int-, long int- ja enum-tietotyypit. Osoittimena voidaan käyttää 16- tai 32-bittistä tietotyyppiä, joista ensimmäistä käytetään osoittamaan pienille data-alueille (SDA, Small Data Area).

Taulukko 9. MicroBlaze:n konekielen tietotyypit, vastaavat ANSI C tietotyypit ja niiden koot tavuina. [24]

Konekielen datatyypit	Vastaavat ANSI C datatyypit	Koko (tavuina)
data8	char	1
data16	short	2
data32	int	4
data32	long int	4
data32	enum	4
data16/data32	osoitin	2/4

MicroBlaze:n yleiskäyttöisten rekisterien käyttösopimukset jakavat rekisterit kolmeen eri luokkaan: volatile, non-volatile ja dedicated. Volatile-rekistereitä käytetään tiedon väliaikaiseen talletukseen, eivätkä niiden arvot säily funktiokutsuissa. Rekisterit R3-R12 ovat juuri volatile-rekistereitä, joista rekistereitä R3 ja R4 käytetään funktion paluuarvon siirtämiseen kutsuvalle funktiolle. Rekistereitä R5-R10 käytetään parametrien siirtoon aliohjelmien välillä. Parametrien siirtoon voidaan tosin käyttää pinoakin.

Rekisterit R19-R31 ovat non-volatile tyyppisiä rekistereitä eli ne säilyttävät arvonsa funktiokutsuissa. Kutsuvan funktion täytyykin tallentaa rekisterit pinoon ja palauttaa ne, jos niitä käytetään kutsuttavassa funktiossa.

Kolmas rekisteriluokka on dedicated, joita ohjelmoijan ei tulisi käyttää mihinkään muuhun, kuin mihin ne on tarkoitettu. Dedicated-rekisterillä R0 on erityinen ominaisuus eli siihen kirjoitettaessa tieto katoaa ja rekisterin arvo onkin aina nolla. Tätä rekisterin R0 ominaisuutta voidaan käyttää esimerkiksi rekisterien nollaamiseen, koska siihen ei ole varsinaista omaa konekielikomentoa. Esimerkiksi konekielikomento

```
addk r6, r0, r0
```

nollaa rekisterin R6. Dedicated-rekistereitä R14-R17 käytetään keskeytysten, aliohjelmien, loukkujen (traps) ja poikkeuksien paluuosoitteen tallennukseen. Esimerkiksi aliohjelmia kutsuttaessa branch- tai link-käskyillä ohjelmalaskurin (PC) arvo tallennetaan rekisteriin R15. Pienen data-alueen osoittimia käytetään erityisien muistialueiden, .sdata (Small Data), .sdata2 (Small Data 2) ja .sbss (Small Block Started By Symbol) käsittelyssä 16-bittisen välittömän arvon kanssa. .sbss-muistialuetta käytetään alustamattomille datasegmenteille ja .sdata- sekä .sdata2-muistialuetta

alustetuille datasegmenteille. Esimerkiksi vakiot tallennetaan näille pienille data-alueille ja niiden lukemiseen käytetään rekisteriä R2. Toinen rekisteri pienten data-alueiden käsittelyssä on R13, jota käytetään niiden alueiden käsittelyssä, joilla voi olla molemmat luku- ja kirjoitusominaisuudet.

Rekisteriä R1 käytetään pinon osoittimena ja päivitetään aina funktiokutsuissa ja paluissa. Rekisteriä R18 käytetään konekielikäskyjen tarvitsemiin väliaikaistalletuksiin. Erityisrekistereitä, kuten ohjelmalaskuria (PC) ja prosessorin tilarekisteriä (MSR), ei voida käsitellä samalla tavoin kuin yleiskäyttöisiä, vaan niiden käsittelyyn on omat konekielikäskyt (*mts* ja *mfs*). Sovitut rekistereiden käyttötarkoitukset on koottu taulukkoon 10.

Taulukko 10. ABI (Application Binary Interface)-määritysten mukaiset rekisterien käyttötarkoitukset selityksineen. [24]

Rekisteri	Tyyppi	Käyttö	Tarkoitus
R0	Dedicated	Laitteisto	Arvo 0
R1	Dedicated	Ohjelmisto	Pino-osoitin
R2	Dedicated	Ohjelmisto	Pienen lukudata-alueen ankkuri
R3-R4	Volatile	Ohjelmisto	Paluuarvo/väliaikaiset
R5-R10	Volatile	Ohjelmisto	Parametreille/väliaikaiset
R11-R12	Volatile	Ohjelmisto	Väliaikaiset
R13	Dedicated	Ohjelmisto	Pienen kirjoitus-ja lukudata-alueen ankkuri
R14	Dedicated	Laitteisto	Keskeytysten paluuosoite
R15	Dedicated	Ohjelmisto	Aliohjelmien paluuosoite
R16	Dedicated	Laitteisto	Trapin paluuosoite (debugger)
R17	Dedicated	Laitteisto, Ohjelmisto*	Poikkeusten paluuosoite
R18	Dedicated	Ohjelmisto	Konekielikäskyjen väliaikaistellukset
R19-R31	Non-volatile	Ohjelmisto	Täytyy tallentaa funktiokutsuissa
PC	Erityis	Laitteisto	Ohjelmalaskuri
MSR	Erityis	Laitteisto	Prosessorin tilarekisteri

ABI määrittelee pinolle parametrien siirtoprotokollan, non-volatile-rekistereiden tallennuksen ja funktioiden paikallisten muuttujien tilatarpeen. Funktiot, jotka sisältävät aliohjelmakutsuja, tarvitsevat uuden pinolohkon omaan käyttöönsä. Kun ohjelma aloittaa suorituksensa, pino-osoitin osoittaa suurimpaan arvoonsa. Funktiokutsuissa pino-osoitinta vähennetään niin monen sanan verran kuin jokainen funktio vaatii. Kutsuvan funktion pino-osoite pysyy siten aina suurempana arvona verrattuna kutsuttuun funktioon.

Muistimalli jakautuu neljään erilliseen osaan: pieni data-alue, data-alue, yleinen alustamaton alue ja vakiot. Pieni data-alue on tarkoitettu julkisille, kooltaan pienille, muuttujille. Muuttujat, jotka luokitellaan pieniksi, on asetettu MicroBlaze:n C-kääntäjään (*mb-gcc*) oletuksena tavuksi, mutta sitä voidaan muuttaa kääntäjän optiolla. Pienen data-alueen koko voi olla 64 kt ja etua tästä on se, että siellä oleviin julkisiin muuttujiin voidaan osoittaa ilman `imm` käskyn käyttöä (Liite 1). Konekielikäskyä `imm` käytetään 32-bittisen muistialueen osoittamiseen muun konekielikäskyn yhdeydessä. Kaikkia muuttujia voidaan siis tällä pienellä data-alueella osoittaa käyttäen absoluuttista osoitetta.

Data-alue sisältää alustetut muuttujat, joihin voidaan osoittaa käyttäen pienen luku- ja kirjoitus-data-alueen ankkuria R13 tai käyttäen absoluuttista osoitetta riippuen kääntäjän optiosta. Yleinen alustamaton alue sisältää julkiset alustamattomat muuttujat ja sitä voidaan osoittaa samalla tavalla kuin data-alueita. Vakiot tallennetaan omalle ainoastaan lukemiseen käytetylle pienelle data-alueelle, jota voidaan osoittaa käyttäen rekisteriä R2. [24]

2.3.5 Matemaattiset operaatiot

MicroBlaze suorittaa laitteistolla kokonaislukujen yhteen- ja vähennyslaskutoimitukset, jotka ovat hyvin nopeita. C-kielellä ohjelmoitaessa oletuksena kertolaskut suoritetaan käyttäen Xilinx:n EDK:n (Embedded Development Kit) mukana tulevan GCC:n C-kirjaston [41] tarjoamaa funktiota *mulsi3_proc*, kuten myös jakolasku ja jakojäännös käyttäen funktioita *divsi3_proc* ja *modsi3_proc*. Xilinx:n EDK on kehitysympäristö sulautetuille FPGA-järjestelmille, joka tarjoaa suunnittelutyökalut ja laajan valikoiman valmiita lisälaitteita käytettäväksi MicroBlaze:n kanssa.

Kirjastofunktioilla toteutetut laskutoimitukset ovat kuitenkin aina paljon laitteistotason toteutusta hitaampia, joten laitteistolla kannattaa toteuttaa laskutoimituksista niin paljon kuin mahdollista. Tuki kertolaskun suorittamiseen laitteistolla kuuluu oletuksena MicroBlaze-ytimeen, mutta vanhemmissa kuin EDK:n 6.3i-versioissa tämä täytyy ottaa käyttöön *mb-gcc* kääntäjän optiolla *-mno-xl-soft-mul*. Myös jakolaskulle löytyy laitteistotuki, mutta se täytyy erikseen kääntää prosessorin ytimeen ennen syntetisointia.

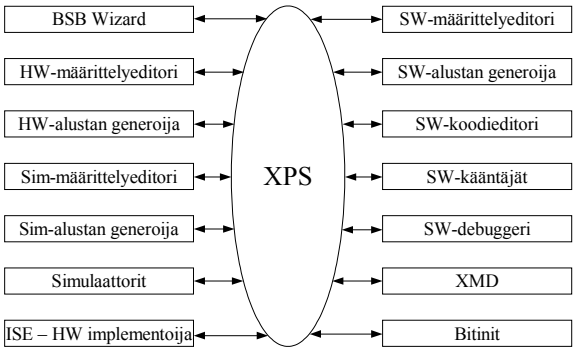
Tämän lisäksi täytyy vanhemmissa EDK:n versioissa vielä ohjelmistokääntäjääkin tiedottaa tästä optiolla *-mno-xl-soft-div*. Tuplatarkkuuden kertolasku, jakolasku ja jakojäännös suoritetaan käyttäen kirjastofunktioita *muldi3_proc*, *divdi3_proc* ja *moddi3_proc*. Erityisesti fraktionaalilukujen, eli kiinteänpilkun lukujen, kanssa suoritettavien skaalausten tulisi olla mahdollisimman nopeita. Barrel Shifter on ratkaisu tähän ongelmaan eli sen avulla voidaan useamman bitin siirto suorittaa kahdella kellojaksolla useamman yhden bitin siirron sijasta. Barrel Shifter täytyy asettaa päälle ennen syntetisointia, jonka jälkeen EDK osaa ottaa tämän käyttöön automaattisesti tai suoraan kääntäjää käytettäessä käyttämällä kääntäjän optiota *-mxl-barrel-shift.*

Kaikki liukulukujen laskutoimitukset kuten yhteenlasku, vähennyslasku, kertolasku ja jakolasku suoritetaan aina ohjelmistolla käyttäen C-kirjastofunktioita. [27]

2.4 Suunnittelu ja analyysi

Xilinx:n EDK:n tarjoamia suunnittelutyökaluja voidaan käyttää yhteistyössä myös kolmannen osapuolen työkalujen kanssa, kuten simulaattorit ja tekstieditorit. EDK sisältää XPS-kehitysympäristön (Xilinx Platform Studio), joka yhdistää kaikki työkalut yhdeksi kokonaisuudeksi. EDK version 6.3i myötä Windows ja Solaris käyttöjärjestelmien lisäksi kehitysympäristöä voidaan käyttää myös Linux Red Hat Enterprise 3.0:n kanssa. Xilinx tarjoaa tuen Linux-jakeluista juuri Red Hat Enterprise 3.0:lle, mutta rajoitettu tuki on tarjolla myös Red Hat jakelun vanhemmille versioille, kuten Red Hat Linux 8.0:lle ja 9.0:lle. Tämän uuden version myötä tukea FPGA-piirillä kulkevien signaalien tarkasteluun käytettävän Xilinx:n Chipscope Pro työkalun käyttämiselle koko järjestelmän verifioinnissa on parannettu huomattavasti sekä väylän toiminnallisen simuloinnin (BFM, Bus Functional Model Simulation) ansiosta omien lohkojen väylärajapinnan tarkastelu on mahdollista.

Xilinx:n sulautettujen järjestelmien työkalut (EST, Embedded System Tools) sisältävät prosessorin alustan räätälöintiohjelmiston, ohjelmistokehityksen työkalut, debuggaustyökalut, laitteistoajurit ja kirjastot. Perusjärjestelmän rakentamiseen käytetyt useat työkalut (kuva 13) mahdollistavat koko sulautetun järjestelmän suunnittelun alusta loppuun.



Kuva 13. Xilinx:n XPS-kehitysympäristön työkaluarkkitehtuuri, jonka useat työkalut mahdollistavat koko sulautettun järjestelmän suunnittelun alusta loppuun. [27]

Koko järjestelmän suunnittelu koostuu niin FPGA-piirin sisäisistä kuin ulkoisistakin laitteisto- ja ohjelmistokomponenteista sekä valinnaisesti myös verifiointi- ja simulointikomponenteista. Laitteistokomponentti voidaan muodostaa automaattisesti luodulle alustalle, jota muokataan vastaamaan käyttäjän vaatimuksia. Ohjelmistokomponentti sisältää vastaavasti ohjelmistoalustan, joka luodaan käyttäjän suunnitteleman ohjelmiston lisäksi työkaluja käyttäen. Verifiointikomponentti koostuu automaattisesti luoduista simulaatiomalleista määrätylle simulaattorille ja koostuu niin laitteisto- kuin ohjelmistokomponenteista. Tyypillinen sulautetun järjestelmän suunnittelu- ja verifiointi noudattaa seuraavia vaiheita:

1. Laitteistoalustan luominen.
2. Laitteistoalustan verifiointi (simulointi).
3. Ohjelmistoalustan luominen.
4. Ohjelmiston luonti.
5. Ohjelmiston verifiointi (debuggaus).
6. Järjestelmän implementointi piirille.

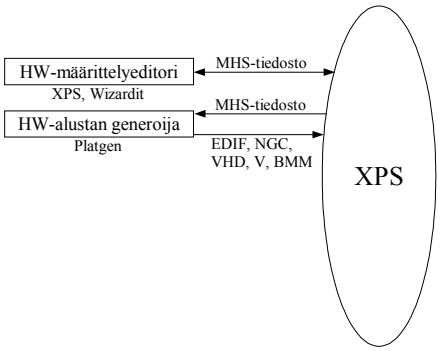
[27]

2.4.1 Laitteistopohjan luominen

Laitteistoalusta on määritelty mikroprosessorin laitteistomäärittelytiedostossa (MHS, Microprocessor Hardware Specification). Tämä tiedosto sisältää tiedot järjestelmän laitteistojen ja sulautetun prosessorin määrityksistä sekä määrittelee lisäksi järjestelmän kytkennät, järjestelmän laitteiden osoitekartat ja asetukset. Myös kaikki käyttäjän omat lisälaitteet täytyy olla tässä tiedostossa määriteltynä. Tiedosto on normaali tekstitiedosto

ja sitä voidaan editoida millä tahansa tekstieditorilla tai käyttäen EDK:n graafista työkalua. Kuvassa 14 on esitetty kuinka MHS-tiedosto liittyy laitteistopohjan luontityökaluihin.

Laitteistoalusta luodaan käyttäen laitteistoalustan luontityökalua (platgen), joka käyttää MHS-tiedoston tietoja. Platgen voi luoda verkkolistan (netlist) useammalla formaatilla (NGC, EDIF, VHD, V ja BMM) sekä mahdollistaa komponenttien lisäämisen myös ylemmällä tasolla automaattisesti luodulle laitteistoalustalle. [27]



Kuva 14. Laitteistoalustan luonnin jälkeen voidaan HW-määrittelyeditorilla käsitellä laitteistomäärittelytiedostoa (MHS), jonka tietojen mukaan HW-alustan generoija (Platgen) luo laitteistoalustan. [27]

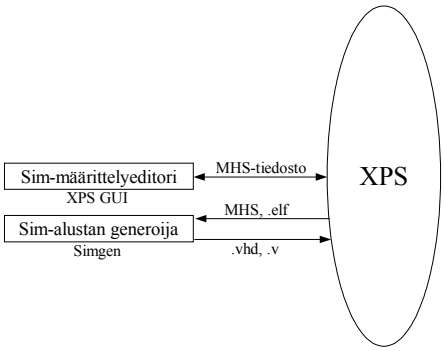
2.4.2 Laitteistoalustan verifiointi

Laitteistoalustan testaaminen voidaan suorittaa verifiointialustan avulla. Verifiointialusta perustuu laitteistoalustaan ja sen avulla voidaan määritellä simulointimalli jokaiselle prosessorille, lisälaitteille tai muille laitteistoalustan moduuleille. Simulointialustan luontityökalu (Simgen) käyttää MHS-tiedoston tietoja ja luo sen avulla määrätylle simulaattorille simulointiedostot (VHDL, Verilog tai vastaavat käännetyt mallit) ja käskytiedostot (kuva 15). Kuten laitteistoalustankin kanssa käyttäjä voi editoida näitä simulointitiedostoja lisäten muita komponentteja jo valmiiksi automaattisesti luotuun verifiointialustaan. Jos ohjelmisto on valmiina ja käytettävissä, sitä voidaan käyttää muistin alustamiseen ja suorittaa siten sitä myös verifiointialustalla. [27]

Yksittäisten komponenttien ja niiden väyläliikenteen verifiointiin voidaan käyttää tyypillisesti kahta menetelmää: luodaan testipenkki tai luodaan jo toimivaksi

osoittautunut suurempi järjestelmä, joka luo väyläliikennettä tarkasteltavalle komponentille. Näistä testipenkin luominen vaatii kaikkien liityntöjen ja testivektoreiden muodostamisen kaikilla eri väyläliikenteen mahdollisuuksilla, jolloin testipenkin luominen on hyvin suuri ja aikaa vaativa tehtävä. Toinen vaihtoehto eli kokonaisen testijärjestelmän luominen vaatii taas muiden lisäkomponenttien luomisen ja ohjelmoimisen niin, että saadaan halutut väyläliikenteet aikaiseksi ja tulkituksi. Tällaisesta kokonaisesta järjestelmästä tulee monesti hyvin suuri ja monimutkainen, jolloin testaaminen muuttuu sen takia hitaaksi ja aikaavieväksi.

Väylän toiminnallisen simuloinnin (BFM) ansiosta väylään liitettävän komponentin tarkastelu on helpointa. Tämä menetelmä on tuettu EDK:n työkaluissa ja sen avulla testivektoreiden luominen on paljon helpompaa kuin edellisillä menetelmillä. Mutta vaikka tällä menetelmällä olisikin helppo testata omien komponenttien toimintaa, niin ainoastaan toiminnalliseen simulointiin (behavioral simulation) rajoittuneena se ei kuitenkaan sovellu ajoituksista riippuvien komponenttien tarkasteluun. [28]

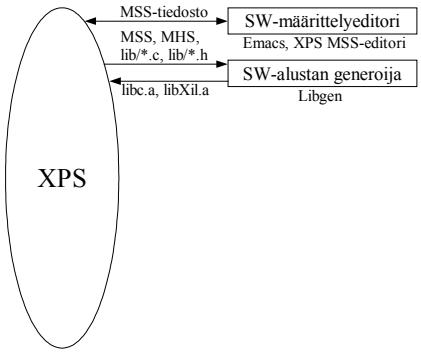


Kuva 15. Laitteistoalustan verifioinnissa simulaattorin määrittelyeditorilla määritellään simulointiasetukset MHS-tiedostoon, jota simulaattorialustan generoija (Simgen) hyödyntää tuottaessaan varsinaiset simulointitiedostot simulaattorille. [27]

2.4.3 Ohjelmistoalustan luominen

Koko MicroBlaze-järjestelmän ohjelmistoalusta on määritelty MSS-tiedostossa (Microprocessor Software Specification). Tästä tiedostosta löytyy määrittelyt lisälaitteiden laiteajurien ja kirjastojen asetusparametreista, prosessorin asetusparametreista, perus-I/O-laitteista, keskeytyskäsittelijärutiineista sekä muista ohjelmistotason ominaisuuksista. Myös MSS-tiedosto on normaali tekstitiedosto, joten

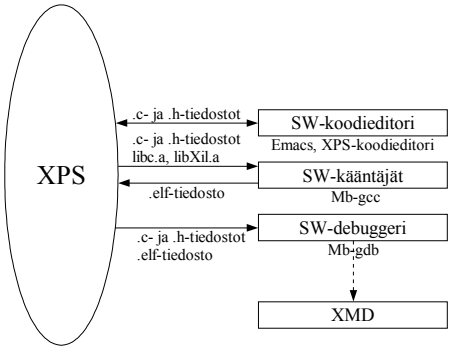
sitä voidaan muunnella normaalilla tekstieditorilla, tosin XPS-työkalut tarjoavat graafisen työkalun tämän tiedoston käsittelyyn. Kirjastojen luontityökalu (LibGen, Library Generator Tool) käyttää MSS-tiedoston määrittelyjä luodessaan ajurit, kirjastot ja keskeytyskäsittelijät (kuva 16). [27]



Kuva 16. Ohjelmistoalustan määrittelyt luodaan MSS-tiedostoon, jonka tietoja ohjelmistalustan generoija (Libgen) käyttää tuottaessaan ohjelmistokirjastot. [27]

2.4.4 Ohjelmiston luonti ja verifiointi

Ohjelmisto on koodia, jota suoritetaan laitteisto- ja ohjelmistoalustoilla. Ohjelmakoodi voi olla kirjoitettu assembler-kielellä tai käyttäen ylemmän tason ohjelmointikieliä kuten C ja C++. Kuvassa 17 on esitetty ohjelmiston luontiin ja verifiointiin käytettyjen työkalujen liittyminen XPS-järjestelmään. XPS tarjoaa koodieditorin, jolla voi käsitellä ohjelmakoodeja, mutta myös mitä tahansa muuta editoria voi käyttää. Kun ohjelmakoodi on valmis, voidaan se kääntää ja yhdistää suoritettavaksi ELF (Executable and Link Format) muotoiseksi tiedostoksi. XPS:n mukana tulee GNU-pohjaiset työkalut ohjelmakoodin kääntämiseen ja linkkaamiseen, mutta voidaan käyttää myös muita kääntäjiä, joista vain löytyy tuki käytetylle prosessorille ja laitteistoalustalle. XMD:tä ja GNU:n debuggeria (GDB) voidaan käyttää yhdessä ohjelmiston debuggaukseen. XMD tarjoaa lisäksi käskykantasimulaattorin, johon voidaan vaihtoehtoisesti GDB:llä liittyä. Tällöin ohjelmakoodia voidaan testata simulaattorissa ennen varsinaiselle laitteistolle siirtoa. [27]



Kuva 17. Ohjelmiston luomisen jälkeen ohjelmakoodit käännetään suoritettavaan muotoon mb-gcc-kääntäjällä. Suoritettavaan muotoon käännettyä tiedostoa ja ohjelmistokoodeja käyttäen XMD:tä hyödyntävällä ohjelmistodebuggerilla voidaan verifioida koko ohjelmistoa. [27]

2.5 Reaaliaikakäyttöjärjestelmä

Eräs määritelmä sulautetulle järjestelmälle on, että se on tietokonepohjainen järjestelmä, joka ei kuitenkaan näytä perustuvan tietokoneeseen. Reaaliaikaiset järjestelmät ovat yleensä sulautettuja järjestelmiä, jotka voidaan jakaa kahteen luokkaan: pehmeisiin ja koviin. Pehmeissä reaaliaikaisissa järjestelmissä tehtävät palvellaan niin nopeasti kuin mahdollista, mutta niiden ei tarvitse valmistua määrätyssä ajassa. Kovissa järjestelmissä tehtäviä täytyy palvella määrätyn ajan kuluessa sekä niiden täytyy myös valmistua määrätyssä ajassa. Useat järjestelmät, kuten esim. taajuusmuuttaja, sisältävät kummankin reaaliaikaisuuden luokkia. Osa moottorisäädön osista täytyy suorittaa kovan reaaliaikaisuuden mukaisesti, kun taas käyttöliittymätoiminnot voivat huoletta toimia pehmeän reaaliaikaisuuden mukaan. [1]

Reaaliaikakäyttöjärjestelmiä kutsutaan lyhyemmin nimellä RTOS (Real-Time Operating System). RTOS:n avulla voidaan selkeyttää ohjelmiston rakennetta, helpottaa uudelleenkäyttöä, parantaa luotettavuutta, jakaa prosessorin resurssit paremmin eri tehtävien (taskien) kesken sekä jaettujen resurssien suojaaminen onnistuu helpommin käyttöjärjestelmän palveluiden ansiosta. Taskeja voidaan kutsua myös säikeiksi (thread). Taskit ovat yksinkertaisia ohjelmia, jotka luulevat yksin omistavansa koko prosessorin. Järjestelmän suorittamat tehtävät jaetaan taskien kesken, joille määritellään tärkeysjärjestys niiden suorittamien tehtävien tärkeyden perusteella. Näin käyttöjärjestelmä osaa tärkeysjärjestyksen perusteella jakaa tehtäville suoritusaikaa oikeassa järjestyksessä.

Koska käyttöjärjestelmän palvelut vaativat kuitenkin oman suoritusaikansa, RTOS ei välttämättä sovellu aivan kaikkiin sulautettuihin järjestelmiin. Pienemmät järjestelmät kannattaakin toteuttaa ennemmin yksinkertaisella foreground/background- tai Round-Robin-menetelmällä, jolloin ei tarvita käyttöjärjestelmää turhaan viemään prosessorin suoritusaikaa. Foreground/background-menetelmässä taskeja suoritetaan peräkkäin ikuisessa silmukassa ja kriittiset osat suoritetaan keskeytyksissä. Tällaisessa järjestelmässä keskeytyksiin vastataan nopeasti, mutta taskitason tapahtumiin vastaaminen riippuu kaikkien taskien yhteenlasketuista suoritusajoista. Taskien suoritusajat eivät yleensä ole myöskään vakion pituisia, joten järjestelmästä ei tule ennustettava. Jos vielä lisäksi ohjelmakoodia jossain vaiheessa muutetaan, niin samalla muuttuu myös taskitason tapahtumiin vastaamisaikakin. [1]

Taajuusmuuttajajärjestelmä edellyttää tarkkaan tapahtuvia jaksollisia (periodisia) ja ennustettavia tapahtumia, käyttäjän ohjausrajapinnan, resurssien jakoa usean tehtävän kesken. Melkein kaikkien näiden palveluiden käyttämiseen ja hallintaan löytyy jo valmiit ratkaisut reaaliaikakäyttöjärjestelmistä. [29]

2.5.1 µC/OS-II

µC/OS-II on Jean J. Labrossen kehittämä RTOS, josta hän on kirjoittanut myös kirjan [1]. Käyttöjärjestelmä on kirjoitettu käyttäen ANSI C:tä ja ainoastaan pieni osa koodista on konekielellä. Tämä konekielellä oleva osio on vielä pyritty pitämään mahdollisimman vähäisenä ja sitä käytetään ainoastaan eri prosessoriarkkitehtuureille sovittamista varten. µC/OS-II:lle löytyykin valmiit prosessoririippuvaiset osat lukuisille prosessorialustoille, mikrokontrollereille ja digitaalisille signaaliprosessoreille, aina 8-bittisistä 64-bittisiin. Käyttöjärjestelmän siirto eri prosessorialustalle onnistuukin, kunhan vain prosessorista löytyy tuki pino-osoittimelle sekä pinoon voidaan lisätä ja poistaa prosessorin rekisterit. Ytimen ohjelmakoodit on kirjoitettu mahdollisimman selväpiirteisesti ja dokumentoidusti. Kirjassa on esitetty myös hyvin tarkasti mitä koodit tekevät ja kuinka ne toimivat yhteen. [1]

µC/OS-II on suunniteltu sulautettuihin järjestelmiin ja tämän vuoksi sitä voidaankin käyttää helposti missä vain, kunhan riittävät ohjelmistotyökalut ovat käytettävissä.

Järjestelmä on myös hyvin skaalautuva eli käyttöön otettavat palvelut voidaan valita ohjelmakoodissa käytettävien esikääntäjän ehdollisten määrittelyjen avulla hyvin vapaasti.

Käyttöjärjestelmä on täysin keskeyttävä (pre-emtive) eli aina siirrytään suorittamaan prioriteetiltaan suurinta taskia, joka on siirtynyt ”valmiina ajoon”-tilaan. Taskeja voi olla 64 kappaletta, mutta kirjassa [1] suositellaan, että näistä 8 kannattaa varata itse käyttöjärjestelmälle. Jokaisella taskilla on oma prioriteettinsa eli µC/OS-II ei voi toteuttaa Round-Robin tyylistä aikataulutusta, jossa jokainen taski saa suoritusvuoron vuorollaan peräkkäin.

Käyttöjärjestelmän palvelut ovat myös ennustettavia (deterministic) eli tiedetään aina, kuinka pitkään jonkun järjestelmän funktion tai palvelun suorittaminen kestää, eivätkä ne ole riippuvaisia taskien lukumäärästä. Ainoastaan OSTimeTick() ja muutamat tapahtumalippujen palvelut eivät ole deterministisiä. Jokainen taski tarvitsee myös oman pinonsa ja µC/OS-II:ssa niiden ei tarvitse olla samankokoisia. Taskien tarvitsemien pinojen koot voidaan arvioida taskien käyttämien muuttujien avulla tai selvittää tarkemmin esimerkiksi erityisen käyttöjärjestelmän palvelun avulla ja näin voidaan vähentää ylimääräistä muistin kulutusta.

µC/OS-II tarjoaa suuren joukon käyttöjärjestelmän palveluita kuten semaforit, ehdolliset semaforit, tapahtumaliput, viestilaatikot, viestijonot, vakio muistiosiot, taskien hallinnan, ajanhallintafunktiot ja paljon muuta. µC/OS-II perustuu µC/OS -järjestelmään, jota on käytetty sadoissa kaupallisissa tuotteissa vuodesta 1992 ja µC/OS-II sisältääkin saman ytimen sekä samoja funktioita. Kesäkuussa 2000 µC/OS-II hyväksyttiin käytettäväksi kaupallisissa ilmailualan laitteissa FAA:n (Federal Aviation Administration) toimesta, jolloin sen täytyy kattaa RTCA DO-178B [42] standardin mukaiset ehdot. Tämä standardi vaatii, että ohjelmiston toiminta voidaan osoittaa dokumentaation ja testien avulla virheettömäksi ja turvalliseksi. [1]

µC/OS-II käyttöjärjestelmästä on itse sen tekijä tehnyt prosessoririippuvaiset osat MicroBlaze:lle, joten se tarjoaakin pääosin kaikki tarvittavat palvelut valmiina tutkittavaan taajuudenmuuttajajärjestelmään [30]. Ainoastaan suora tuki jaksollisten

taskien suorittamiselle puuttuu. Jaksollinen suoritus voidaan toteuttaa esimerkiksi käyttäen laitteistolaskuria, jolloin keskeytyspalvelu signaloi jaksollisesti taskia käyttäen semaforia tai tapahtumalippuja. Tällainen toteutus ei olisi kuitenkaan kovin portattava, joten TkT Julius Luukko on tehnyt μC/OS-II:een laajennuksen, jonka avulla periodisten taskien suorittaminen voidaan toteuttaa portattavasti käyttäen käyttöjärjestelmän laajennukseen käytettyjä lisäpalveluita. [31]

2.5.1.1 Keskeytykset

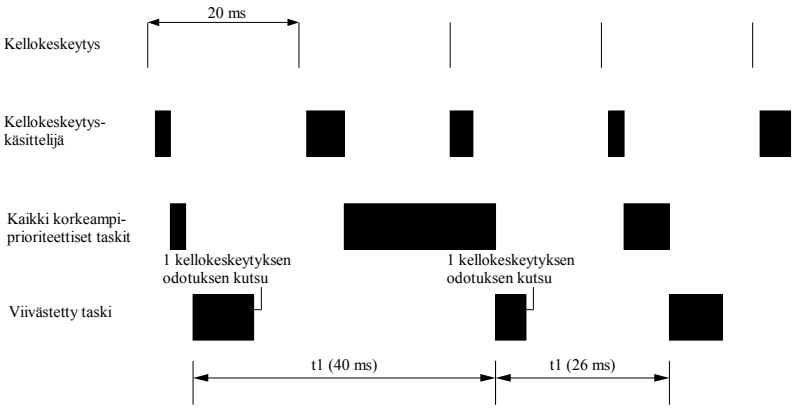
Keskeytyksen tapahduttua prosessori tallentaa kontekstinsa (esim. rekisterit) ja siirtyy suorittamaan aliohjelmaa, jota kutsutaan keskeytyskäsittelijäksi (ISR, Interrupt Service Routine). Keskeytyskäsittelijä käsittelee tapahtuman, jonka jälkeen keskeyttävä käyttöjärjestelmä, kuten μC/OS-II, siirtyy suorittamaan prioriteetiltaan suurinta taskia, joka on valmis ajoon. Keskeytykset voidaan sallia ja kieltää järjestelmän funktioilla. Reaaliaikajärjestelmissä pyritään kuitenkin olemaan kokonaan kieltämättä keskeytykset tai kieltämään keskeytykset mahdollisimman vähäksi aikaa, koska keskeytysten kieltäminen vaikuttaa keskeytysten latenssiin ja voi pahimmillaan aiheuttaa keskeytysten ohi menemisen. Yleensä prosessorit sallivat keskeytysten ketjuttamisen niin, että tärkeämpi keskeytys voi keskeyttää alemman keskeytyksen. μC/OS-II:n MicroBlaze-versio ei kuitenkaan tue vastaavaa keskeytysten ketjuttamista, koska prosessorin ainoan keskeytyslinjan keskeytykset kielletään keskeytyksen tapahduttua. Tosin keskeytyskäsittelijä voi ottaa vastaan useampia keskeytyksiä ja prosessorin keskeytyksen tapahduttua tarkistetaan keskeytyskäsittelijältä tapahtuneet keskeytykset. Tapahtuneet keskeytykset palvellaan siten määrätyssä järjestyksessä peräkkäin eikä ketjutetusti.

MicroBlaze sisältää vain yhden keskeytyssignaalin, joten keskeytysvektoreitakin on vain yksi. Kun IE-bitti prosessorin MSR-rekisterissä on asetettuna, MicroBlaze suorittaa keskeytyksen tullessa suorituksessa olevan käskyn loppuun ja siirtyy suorittamaan keskeytysvektoria. EDK:n ohjelmistoalusta tarjoaa valmiin keskeytysesikäsittelijäohjelman, johon keskeytysvektorista oletuksena siirrytään. Keskeytysesikäsittelijän tehtävänä on tallentaa ja palauttaa prosessorin rekisterit, tarkastaa keskeytyskäsittelijälohkolta mikä tai mitkä lisälaitteet ovat keskeytyksen

aiheuttaneet sekä suorittaa näiden varsinaiset keskeytyskäsittelijäaliohjelmat. μC/OS-II-käyttöjärjestelmän tarvitsee tosin tehdä keskeytyksen alussa ja lopussa muutakin, joten tämä EDK:n keskeytysesikäsittelijä ei ole riittävä käyttöjärjestelmän tarpeisiin. Käyttöjärjestelmää varten täytyy ennen käyttöjärjestelmän käynnistystä vaihtaa keskeytysvektori siirtymään käyttöjärjestelmän omaan esikäsittelijään.

Kellokeskeytys (clock tick) on keskeytys, joka tapahtuu jaksollisesti ja voidaan ajatella olevan eräänlainen järjestelmän sydänpulssi. Tämän keskeytyksen ansiosta käyttöjärjestelmä voi viivästää taskeja kellokeskeytysten tarkkuudella. Keskeytyksen väliaika on sovelluksesta riippuvainen ja normaalisti 10-200 millisekuntia. Mitä pienempi on kellokeskeytysten väliaika, sitä enemmän se aiheuttaa ylimääräistä kuormaa järjestelmään.

Käyttöjärjestelmän ydin sallii taskien viivästää toimintaansa kellokeskeytysten monikerran verran. Tarkkuus ei kuitenkaan ole tarkalleen kellokeskeytysten väli vaan riippuu kellokeskeytyskäsittelijään kuluvasta ajasta ja prioriteetiltaan korkeiden taskien suoritusajoista. Prioriteetiltaan korkeiden taskien yhtäjaksoinen suoritusaika sekä kellokeskeytyskäsittelijä eivät saisi olla missään tapauksessa yli yhtä kellokeskeytysten väliaikaa. Jos kellokeskeytysten väliaika on pienempi, viivästänyt taski ei pääse ajallaan takaisin ajoon (kuva 18). Taskien viivästäminen on siis yksi tekijä, mikä määrää kuinka lyhyeksi kellokeskeytyksen väli voidaan asettaa [1]. Yleisellä tasolla voidaan ajatella myös, että ylemmän tason taskien yhteenlaskettu yhtäjaksoinen suoritusaika ei saa olla enempää kuin mitä alemman tason taskin suoritusväli vaatii.



Kuva 18. Jos kellokeskeytyskäsittelijän ja korkeamman prioriteettisten taskien yhtäjaksoinen suoritusaika on pidempi kuin kellokeskeytysten väliaika, viivästynyt taski ei pääse ajallaan takaisin ajoon. [1]

Kellokeskeytyksen lähteenä voidaan käyttää mitä tahansa jaksollista kellosignaalia. FPGA-piirin IP-lohkojen synkronointi prosessorilla ajettavan ohjelmiston kanssa kannattaakin toteuttaa niin, että FPGA-piiriltä otetaan IP-lohkojen käyttämä kellosignaali kellokeskeytykselle. Näin FPGA-piirin IP-lohkot saadaan toimimaan synkronoidusti ohjelmiston kanssa, jolloin tiedonsiirto näiden välillä helpottuu.

2.5.1.2 Semaforit

Helpoin tapa viestiä eri taskien kesken on käyttää jaettuja muuttujia ja tietorakenteita. Näin voidaankin toimia, jos suoritettavat taskit sijaitsevat samassa muistiavaruudessa. Tällaisessa tietojen jaossa täytyy kuitenkin ottaa huomioon, että jokaisella taskilla on yksinomainen käyttömahdollisuus resurssiin, jotta tiedon korruptoitumista ei tapahtuisi. Eräs vaihtoehto tähän on kieltää aina keskeytykset, kun jaettua tietoa käsitellään, mutta käyttöjärjestelmällä on tiedon suojaamiseen myös palvelu nimeltä semafori. Semaforeja voidaan tosin käyttää muuhunkin kuin kontrolloimaan jaetun resurssin käyttöä, kuten signaloimaan tapahtuman tapahtumista ja synkronoimaan kahden taskin suoritusta. [1]

2.5.1.3 Mutex-semaforit

Ehdolliseen suoritukseen pystyviä semaforeja (mutual exclusion semaphore) kutsutaan mutexeiksi. Mutexit ovat binaarisemaforeja, mutta normaaleihin semaforeihin verrattuna ne sisältävät lisäominaisuuksia, joista eräs tärkein on prioriteetin inversion ratkaiseminen. Prioriteetti-inversio on nimitys tapahtumalle, jossa alemman prioriteetin

taski omistaa resurssin, jota korkeamman prioriteetin taski tarvitsee. Prioriteetin inversion ratkaisemisen ansiosta järjestelmä ei jää tähän jumiin, vaan ydin pääsee nostamaan alemman prioriteetin taskin prioriteetin ylemmän prioriteetin taskin kanssa samaksi. Tällä menettelyllä alemman prioriteetin taski pääsee suorittamaan tehtävänsä loppuun ja vapauttamaan resurssin.

Yleensä reaaliaikakäyttöjärjestelmissä prioriteetin inversion ratkaiseminen vaatii, että käyttöjärjestelmän ydin sallii useammalla taskilla olevan sama prioriteetti. µC/OS-II:n ydin ei kuitenkaan tällaista salli, mutta onneksi ongelma voidaan ratkaista toisellakin tavalla. Prioriteetin inversio voidaan siis ratkaista myös siten, että alemman prioriteetin taskin prioriteetti muutetaan suuremmaksi kuin suuremman prioriteetin taskin. Tämä toteutus vaatii ainoastaan sen, että taskien prioriteettia suurempi prioriteettiarvo täytyy varata tyhjäksi. [1]

2.5.1.4 Tapahtumaliput

Tapahtumalippuja käytetään synkronoimaan taskeja useiden tapahtumien mukaan. Taski voidaan valita reagoimaan niin, että jokin tapahtumista tapahtuu (OS_FLAG_WAIT_SET_ANY ja OS_FLAG_WAIT_CLR_ANY), tai kaikkien tapahtumien täytyy tapahtua (OS_FLAG_WAIT_SET_ALL ja OS_FLAG_WAIT_CLR_ALL). Yhdessä tapahtumalipussa voi olla tapahtumia 8-, 16- tai 32-kappaletta riippuen käyttöjärjestelmän ytimen asetuksista. [1]

2.5.1.5 Viestilaatikot

Viestilaatikko on käyttöjärjestelmän palvelu taskien ja keskeytysaliohjelmien viestien välittämiseen osoittimen kokoisen muuttujan avulla toiseen taskiin. Osoitinta voidaan käyttää siirtämään osoite johonkin sovelluskohtaiseen tietorakenteeseen, joka sisältää tarvittavan tiedon. Koska useampi kuin yksi taski voi vastaanottaa viestilaatikkoon laitetun viestin (OS_POST_OPT_BROADCAST), viestilaatikon yhteyteen liitetään odotuslista odottavista taskeista. Viestilaatikkoa voidaan käyttää myös binäärisemaforin tapaan eli viestilaatikossa odottava viesti voi tarkoittaa palvelun olevan vapaana ja tyhjä viestilaatikko tarkoittavan palvelun olevan jonkun muun käytössä. [1]

2.5.1.6 Viestijonot

Viestijonoilla voidaan lähettää viestilaatikoiden tapaan osoittimen kokoisia muuttujia yhteen (OS_POST_OPT_NONE) tai useampaan taskiin (OS_POST_OPT_BROADCAST). Viestijono sisältää useita viestilaatikoita eli se on viestijonotaulukko. Viestijonoon lähetetyt viestit voidaan lukea joko FIFO- tai LIFO-periaatteiden mukaisesti. LIFO-periaatteella viesti sijoitetaan viestijonoon ensimmäiseksi (OS_POST_OPT_FRONT), jolloin se saadaan ensimmäisenä luettua. [1]

2.6 Lukujen esitysmuodot ja aritmetiikka

Prosessoreiden käyttämä aritmetiikka ja lukujen esitystapa voidaan jakaa kiinteän pilkun ja liukuvan pilkun toteutuksiin. Liukuvan pilkun toteutus on yleisemmin käytetty tehokkaiden tietokoneiden kanssa, kun taas kiinteän pilkun toteutusta käytetään yleensä digitaalisissa signaaliprosessoreissa sekä mikrokontrollereissa. Kiinteän pilkun aritmetiikan toteuttaminen on paljon helpompaa ja siten sitä on helpompi käyttää myös FPGA-piireissä. Kiinteänpilkun aritmetiikka voidaan toteuttaa kokonaisuudessaan käyttäen kokonaislukuja ja kokonaislukuaritmetiikkaa.

2.6.1 Kokonaislukuaritmetiikka

Kokonaisluvut ovat yksinkertaisin tapa esittää lukuja joissa bittijono tulkitaan niin, että esimerkiksi yhteen tavuun mahtuu etumerkittömät kokonaisluvut väliltä 0-255. Etumerkillisten kokonaislukujen yksinkertaisimmassa esitysmuodossa käytetään yhtä, yleensä ylintä, bittiä ilmoittamaan onko kyseessä negatiivinen vai positiivinen luku. Tällöin esimerkiksi yhdellä tavulla voidaan esittää luvut väliltä -128 – 127. Kokonaislukuaritmetiikkaa käyttävät prosessorit ovatkin edullisempia ja yleisimmin käytettyjä säätöjärjestelmissä sekä digitaalisessa signaalinkäsittelyssä niiden yksinkertaisuuden ja nopeuden ansiosta.

MicroBlaze:ssa kokonaislukujen yhteen ja vähennyslaskuille on omat konekielikomennot, joiden suorittaminen kestää vain yhden kellojakson. Laitteistokertojan ansiosta kertolaskun konekielikäskyn suorittaminen vaatii kolme kellojaksoa. Kertolasku toteutetaan 32x32-bittisessä muodossa eli lopputulos on 64-bittinen, mutta eniten merkitsevät 32-bittiä kadotetaan. Myös jakolaskulle on oma

käskynsä, jos tuki sille asetettu ennen syntetisointia ja jolloin jakolasku voidaan suorittaa 34 kellojaksolla. [24]

2.6.2 Liukulukuaritmetiikka

Liukulukuja eli liukuvan pilkun lukuja käyttävät pääosin tavalliset tietokoneet, joiden suorittimiin on rakennettu liukulukuyksikkö hoitamaan laskutoimitukset. Prosessorit eivät suoraan osaa tehdä liukulukulaskutoimituksia ja ilman laitteistotukea ne suoritetaankin kirjastofunktioiden tarjoamilla monimutkaisilla sarjoilla kokonaislukulaskuja. Liukuluvuilla voidaan esittää desimaalilukuja, joka tapahtuu eksponenttiesityksen mukaisesti eli

$$14.5 = 145 * 10^{-1} \quad . \tag{1}$$

Näin liukuluku voidaan esittää kahden kokonaisluvun avulla eli edellisen luvun tapauksessa 145:n ja -1:n avulla. Mikroprosessori käyttää kuitenkin binaarijärjestelmää, jossa luvut esitetään IEEE-standardin 754 [31] mukaisessa muodossa. Yksinkertaisen tarkkuuden liukuluku on siis muotoa

$$(-1)^s \times 2^{(e-127)} \times (1.m)_2 \quad , \tag{2}$$

jossa *s* on merkkibitti, *e* on eksponenttiosa ja *m* on mantissa. Liukulukujen tarkkuus ja suurin mahdollinen luku riippuvat siitä, kuinka monta bittiä luku vie. Yksinkertaisen tarkkuuden liukuluvut ovat 32-bittiä pitkiä, joista yksi on merkkibitti, eksponentti 8-bittiä ja loput mantissaa. Kaksinkertaisen tarkkuuden liukuluvuilla käytetään yhtä merkkibittiä, 11-bittistä eksponenttia ja loput mantissaa. Kaksikertaisen tarkkuuden liukuluvuilla yksinkertaisen tarkkuuden yhtälön (2) bias-arvon (127) sijasta käytetään arvoa 1023. [32]

Kokonaislukuprosessorien liukulukulaskutoimitukset käyttävät kirjastofunktioita ja niiden suoritusnopeudet ovat riippuvaisia kokonaislukujen laitteistotuesta kertolaskulle, jakolaskulle ja bittien siirrolle.

2.6.3 Liukulukuaritmetiikan korvaaminen kokonaislukuaritmetiikalla

Liukulukuaritmetiikan toteuttaminen on yleensä paljon hitaampaa kuin kokonaislukuaritmetiikan varsinkin, jos laitteistotukea liukuluvuille ei ole. Kokonaisluvuilla operaatiot ovat hyvinkin nopeita, ainakin yhteen- ja vähennyslaskut, joten laskutoimitukset olisi hyvä toteuttaa tehokkaammin näitä hyödyntäen. Kokonaislukuprosessorin tapauksessa voidaan käyttää joko kiinteänpilkun lukuja eli fraktionaalilukuja, tai käyttäen skaalausta. Nämä kaksi toteutustapaa ovat hyvin lähellä toisiaan ja voidaan suorittaa samalla tavalla kokonaislukuoperaatioiden avulla. Kiinteän pilkun lukujen etuna on, että luvut ovat aina suoraan oikeassa muodossa.

2.6.3.1 Kiinteänpilkun aritmetiikka

Kiinteänpilkun luvut eli fraktionaaliluvut ovat yleisesti käytetty muoto kokonaislukuprosessorien kanssa, joissa ei ole laitteistotukea liukuluvuille, kuten yleensä digitaalisissa signaaliprosessoreissa. Aikakriittisissä järjestelmissä, joissa vaaditaan desimaaliluvuilla laskemista, täytyy liukulukulaskutoimitukset vaihtaa kiinteänpilkun laskutoimituksiksi säästämään suoritinaikaa. Kiinteänpilkun esitys on yleistys desimaaliesityksestä, jossa luku esitetään numerojonona desimaalipisteen kanssa. Pisteen vasemmalla puolella olevat numerot edustavat luvun kokonaisosaa ja pilkun vasemmalla puolella olevat luvun loppuosaa. Tietokoneen muistissa ei tosin pilkkua ole. Koska kysymyksessä on nimenomaan kiinteän pilkun esitystapa, niin pilkun paikka on kaikilla luvuilla samassa paikassa ja laskutoimituksissa tiedetään, missä se sijaitsee. [32]

Reaaliluku X esitetään fraktionaalimuodossa

$$X = (b_{-A}, \ldots, b_{-1}, b_0, b_1, \ldots, b_B)_r = \sum_{i=-A}^{B} b_i r^{-i}, 0 \le b_i \le (r-1) \quad , \tag{3}$$

jossa r on kantaluku, A kokonaisluvun muodostavien numeroiden määrä ja B fraktionaaliosan numeroiden määrä. Prosessoritoteutuksen kannalta binääriesitys on olennainen, jolloin kantaluku $r = 2$ ja binäärinumerot b_i voivat saada arvon 0 tai 1. Esimerkiksi desimaalilukua 6.25 vastaava binääriluku $(110.01)_2$ muodostetaan

seuraavasti

$$(110.01)_2 = 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} \quad . \tag{4}$$

Tärkeitä fraktionaalilukujen esitysmuotoja ovat muodot, joissa fraktionaaliosassa on nolla numeroa ($A = n$ - 1, $B = 0$) sekä muoto, jossa kokonaisosassa on nolla numeroa ($A = 0$, $B = n - 1$). [43] Ensimmäisessä muodossa esitetyt luvut esittävät kokonaislukuja välillä $0...2^n - 1$. Toisessa muodossa esitetyt luvut ovat binäärifraktionaalimuoto ja esittävät fraktionaalilukuja väliltä $0...1\text{-}2^{-n}$. Tästä jälkimmäisestä muodosta käytetään myös nimitystä Q-muoto. Q-muodossa Q-kirjaimen yhteyteen voidaan merkitä bittien lukumäärä (poislukien merkkibitti), esimerkiksi 16-bittistä lukua merkitään Q15 (15+merkkibitti). Negatiivisten lukujen esittämiseen on kolme vaihtoehtoa: merkki-itseisarvo-, yhden- ja kahdenkomplementtimuoto. Näistä jälkimmäisin on tärkein ja yleisimmin käytetyin tapa. [44]

Kertolaskuissa täytyy ottaa huomioon, että kahden b-bittisen luvun tulo on yleisesti 2b-bittinen, joten kiinteän pilkun aritmetiikassa tulos on joko katkaistava tai pyöristettävä takaisin b-bittiseksi. Tämä aiheuttaa katkaisu- tai pyöristysvirhettä, jota voidaan vähentää käyttämällä laskennassa 2b-bittisiä rekisterejä. MicroBlaze:n tapauksessa käytössä ovat 32-bittiset rekisterit, joten katkaisu- ja pyöristysvirheen minimoimiseksi lukualue kannattaa rajoittaa Q15 muotoon. Q15-muodossa 15 bitillä esitetyn luvun tarkkuus on

$$2^{-15} \approx 3.05\times 10^{-5} \quad , \tag{6}$$

joka on riittävä moniin sovelluksiin.

Yhteen- ja vähennyslaskut suoritetaan kokonaislukuaritmetiikalla ja fraktionaaliaritmetiikalla samalla tavalla, mutta kertolaskun tapauksessa lopputuloksen tulkinnassa on eriäväisyyksiä. Oleellinen ero on, että kokonaislukuaritmetiikassa tuloksen kannalta tärkeät bitit ovat oikeanpuoleisimmat ja fraktionaaliaritmetiikassa vasemmanpuoleisimmat, poislukien jatkettu merkkibitti. Näin myös kertolasku voidaan

suorittaa millä tahansa kertojalla lisättynä tarpeellisilla bittisiirroksilla.

2.6.3.2 Skaalaus

Liukuluvut voidaan myös skaalata sopivalle kokonaislukualueelle, jolloin laskutoimitukset voidaan suorittaa suoraan kokonaislukuaritmetiikkaa käyttäen. Prosessorilla toteutettaessa käytetään skaalausarvona sopivaa 2:n potenssia. [32] Skaalauskertoimen valinnassa täytyy ottaa huomioon, että kertolasku suurimmilla luvuilla mahdutaan esittämään valittavalla bittimäärällä. 32-bittisiä muuttujia käytettäessä kertolaskun bitit on silloin saatava mahtumaan 30 bittiin (30 + 2 merkkibittiä), jolloin suurin mahdollinen bittisiirrosten määrä on 15 bittiä.

Esimerkiksi jos halutaan laskea summa 0.54 + 1.68 = 2.22. jolloin luvut kerrottuna 2^{15}:lla, saadaan ne kokonaislukualueelle. Nyt skaalatuilla luvuilla kokonaisluvuiksi katkaistuina summaksi tulee 17694 + 55050 = 72774. Jos lopputulos nyt halutaan takaisin alkuperäiselle lukualueelle, on se jaettava alussa kerrotulla luvulla eli 2^{15}:lla. Tulokseksi tulee noin 2.22, johon jää pientä virhettä skaalausten kokonaislukualueelle katkaisujen vuoksi. Lukua ei tosin aina tarvitse skaalata takaisin alkuperäiselle lukualueelle, jos laskutoimituksia jatketaan tuloksen kanssa.

Kertolaskussa tarvitaan vielä yksi skaalausoperaatio lisää. Jos lasketaan esimerkiksi 0.54 x 1.68 = 0.9072, jonka tulontekijät skaalataan 2^{15}:lla, saadaan 17694 * 55050 = 974054700. Tämä tulos on vielä 2^{15} kertainen oikeaan verrattuna ja vielä siitä 2^{15} kertainen mitä se on alkuperäisellä lukualueella. Eli kertolaskun jälkeen täytyy tulos jakaa skaalauksessa käytetyllä luvulla, jotta saadaan oikea lopputulos.

Jakolaskussa tarvitaan vastaavanlainen operaatio. Esimerkiksi laskettaessa 1.5 / 0.5 = 3, jolloin luvut skaalattuna 2^{15}:lla, on vastaavasti 49152 / 16384 = 3. Tämä tulos olisi oikea alkuperäisellä lukualueella, mutta takaisinskaalauksen takia se tulkittaisiin luvuksi $9.155*10^{-5}$. Jotta tulos olisi halutunlainen, niin jakolaskussa on jaettavaa kerrottava vielä yhden kerran 2^{15}:lla. Tällöin laskutoimituksesta tulee 2^{15} * 49152 / 16384 = 98304, joka on halutulla skaalausalueella.

2.6.4 Trigonometriset funktiot

GCC:n C-kirjasto sisältää mm. seuraavat trigonometriset funktiot: sin, cos ja atan. Nämä funktiot käyttävät liukulukuja ja yleisesti ottaen ne ovat liian hitaita reaaliaikajärjestelmille. Muita nopeampia tapoja toteuttaa trigonometrisiä operaatiota ovat mm. yhdistetty digitaalinen sinioskillaattori [43], Taylorin sarjan avulla tai taulukoimalla. Taulukointi on edellisistä nopein, mutta vastaavasti se tarvitsee myös eniten muistia.

3 MITTAUKSET JA TULOKSET

Taajuusmuuttaja tarvitsee moottorisäädön lisäksi runsaasti muitakin ominaisuuksia, joten arvioidaan näiden kaikkien ominaisuuksien toteuttamiseen tarvittavien resurssien kulutusta FPGA-piirillä. Kerrotaan kuinka suorituskykymittausjärjestelmä, sekä prototyypin prosessorijärjestelmä rakennetaan. Tässä kappaleessa mitataan myös laskutoimitusten ja reaaliaikakäyttöjärjestelmän palveluiden suoritusnopeuksia suorituskykymittausjärjestelmällä, koska nykyaikaiset sähkömoottorin ohjausalgoritmit vaativat suuren määrän laskutoimituksia. Vastaavasti reaaliaikakäyttöjärjestelmää käytettäessä on hyvä tietää kuinka paljon aikaa prosessorilta kuluu taskien vaihtamiseen, keskeytyksiin vastaamiseen ja yleensä eri käyttöjärjestelmän palveluiden suorittamiseen.

3.1 IP-lohkojen tarvitsemat resurssit

IP-lohkojen tarvitsemia resursseja ei voida määrittää aivan tarkkaan, koska ne vaihtelevat esimerkiksi kohdepiirin, syntetisointiparametrien ja IP-lohkojen asetusten mukaan. Xilinx EDK:n mukana tulevien IP-lohkojen datalehdissä onkin aina arvio kulutetuista resursseista neljänä eri resurssin arvona. Näitä annettuja resursseja ovat BRAM, FF eli kiikut, LUT ja siivut. Näiden arvojen avulla resurssien kulutuksen vertailu ei ole kovin helppoa, joten parempi olisi käyttää vähäisempää arvojen määrää.

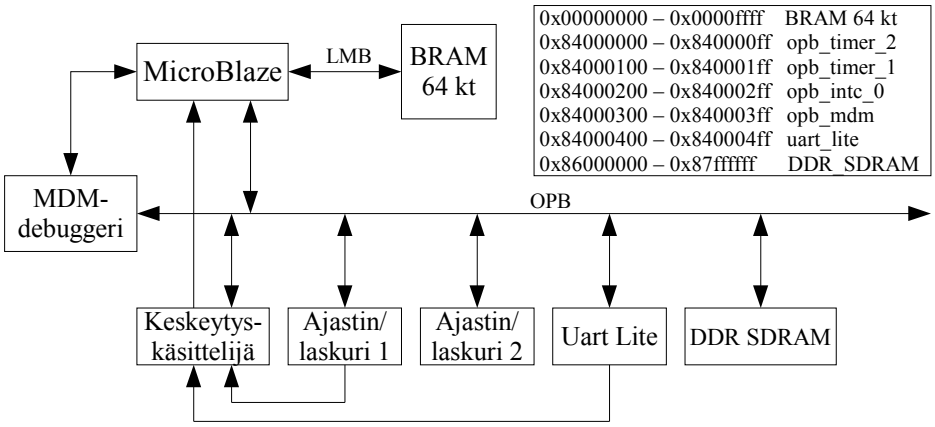
FPGA-piirin perusresursseja ovat logiikkalohkot, mutta niiden sisältämät resurssit vaihtelet paljon FPGA-piireillä. Logiikkasolut muodostuvat pääosin hakutauluista sekä pienestä määrästä muuta logiikkaa, jotka ovat eri FPGA-piirien valmistajillakin hyvin samanlaisia. Tämän vuoksi logiikkasoluja onkin kannattavampi käyttää IP-lohkojen resurssien tarvetta arvioitaessa. Xilinx:lla on kotisivuillaan IP-lohkojen arviointilaskuri, jolla voidaan arvioida koko prosessorijärjestelmän resurssien tarvetta eri lisälaitevalikoimalla. Tällä laskurilla on selvitetty taulukkoon 11 kerätyt IP-lohkojen tarvitsemat resurssit käytettäessä Virtex-II Pro FPGA-piiriä. Nämä arvot ovat tosin vastaavia Virtex-II piireilläkin. [33] Taulukossa on arviot prosessorille, OPB-väylälle, ajastin/laskurille, keskeytyskäsittelijälle, BRAM-muistin ohjaimelle, sarjaliikennekomponentille ja ulkoiselle DDR SDRAM-muistin.

Taulukko 11. Xilinx:n IP-lohkolaskurilla [33] selvitetyt eri IP-lohkojen tarvitsemat resurssit FPGA-piiriltä.

IP-lohko	Logiikkasolut
MicroBlaze	950
OPB Arbiter & Bus structure	110
OPB Timer/Counter	255
OPB Interrupt Controller (min 2)	34
OPB Uart Lite	88
OPB DDR Controller	370

3.2 Mittausjärjestelmä

Mittausjärjestelmä toteutettiin automaattisesti luodulle alustalle, jolle valittiin kellotaajuudeksi 100 MHz, sisäisen data- ja käskymuistin kooksi 64 kt ja ohjelman siirtoa sekä debuggausta varten MDM-komponentti. Lisälaitteiksi OPB-väylään valittiin sarjaliikennekomponentti Uart Lite, kaksi ajastin/laskuria ja DS-DB-V2MB1000 testaus-ja kehityslevyltä otettiin käyttöön DDR SDRAM muistia 32 Mt. Keskeytykset kytkettiin käytettäviksi sarjaliikennekomponentin ja toisen laskurin kanssa. Mittausjärjestelmä on kuvan 19 mukainen.



Kuva 19. Prosessorijärjestelmän lohkokaavio suorituskykymittauksia varten sekä lisälaitteiden sijoittuminen muistiavaruudessa. Muisti on kytketty LMB-väylän kautta prosessoriin ja kaikki lisälaitteet OPB-väylän kautta. Keskeytyksiä käyttävät sarjaliikennekomponentti ja toinen ajastin/laskureista.

Ennen järjestelmän syntetisoimista prosessorin asetuksista valittiin vielä laitteistotuki jakolaskulle ja bittisiirroksille. Syntetisointitulosten mukaan FPGA-piirin resursseja lohkot kuluttivat taulukon 12 mukaisesti. Kokonaisresursseista kulutettiin siis yhteensä 3142 hakutaulua eli 30 % piirin resursseista. Kokonaisresurssien kulutus on hiukan suurempi kuin mitä taulukon 12 yhteenlaskettu määrä, koska taulukossa ei ole otettu

huomioon esimerkiksi kuinka paljon resursseja kuluu kaikkien IP-lohkojen yhdistämiseen.

Taulukko 12. Mittausjärjestelmän IP-lohkojen kuluttamat resurssit. Prosessori vie arvioidusta huomattavasti enemmän esimerkiksi jakolaskun ja bittisiirroksen laitteistotuen vuoksi. ILMB ja DLMB ovat käsky- ja datapuolen BRAM-muistiohjaimia.

IP-lohko	Hakutaulua
MicroBlaze	1719
ILMB BRAM Controller	5
DLMB BRAM Controller	5
OPB v2.0	208
OPB Timer/Counter 1	283
OPB Timer/Counter 2	283
OPB Interrupt Controller	69
OPB MDM	119
OPB Uart Lite	95
OPB DDR Controller	284
Yhteensä	3070

Taulukon 12 resurssien kulutusarvoja verrattaessa Xilinx:n IP-laskurilla saatuihin arvioihin (taulukko 11) nähdään, että prosessorin arvio on huomattavasti pienempi kuin mitä mittausjärjestelmämme prosessori kuluttaa. Prosessorin resurssien kulutus on tosin suurempi ainakin siihen lisättyjen laitteistolaajennuksien vuoksi. Muutenkin IP-laskurin arviot ovat pääosin pienempiä kuin esimerkkijärjestelmässämme, joka voi tosin johtua ainakin eri syntetisoinnin optimointitason käytöstä. Jos taulukosta 11 lasketaan vielä koko mittausjärjestelmän resurssien kulutus, saadaan tulokseksi 2062 logiikkasolua. Koko mittausjärjestelmän resurssien kulutuksen arvio on siten huomattavasti alhaisempi kuin mittausjärjestelmän.

3.3 Suorituskykymittaukset

Seuraavaksi suoritetut suorituskykymittaukset on mitattu kellojaksoja käyttäen, koska tällöin saadaan suoraan prosessorin kellotaajuudesta riippumattomia arvoja sekä mittaukset ovat helposti toistettavissa. Kellojaksot on mitattu prosessoriin liitetyllä laskurilla, joka lisää arvoansa joka kellojaksolla. Myös tämän laskurin arvon lukemiseen tarvittava suoritusaika otetaan huomioon kalibroinnissa ennen laskutoimitusten mittausten suorittamista ja joka vähennetään aina kokonaissuoritusajasta. Kalibrointiin käytetty C-kielinen ohjelmakoodi on liitteessä 1 ja se on pseudokoodina esitettynä :

```
luetaan laskurin_alku_arvo;
luetaan laskurin_loppu_arvo;
vähennettävä_aika = laskurin_loppu_arvo  -
                    laskurin_alkuarvo;
```

Mittaus tapahtuu siten, että laskurin käynnistyksen jälkeen ja ennen mitattavaa suoritusta luetaan laskuriin aloitusarvo sekä mitattavan suorituksen jälkeen luetaan lopetusarvo. Lopetusarvosta vähennetään aloitusarvo sekä kalibroinnissa saatu arvo, jolloin saadaan ainoastaan mitattavan toimituksen kuluttamat kellojaksot. Liitteessä 1 on C-kieliset ohjelmakoodit myös mittauksen aloitukseen ja lopetukseen. Mittaus tapahtuu pseudokoodina esitettynä:

```
luetaan laskurin_alku_arvo;
mitattava_suoritus;
luetaan laskurin_loppu_arvo;
suoritus_aika = laskurin_loppu_arvo –
                laskurin_alku_arvo -
                vähennettävä_aika;
```

Laskutoimitusten suoritusnopeudet mitataan siten, että laskettavat arvot haetaan muistista rekistereihin, koska laskuoperaatiot tapahtuvat aina rekistereissä. Lopputulos tallennetaan myös rekisteriin eli mittaukseen otetaan mukaan myös tämän arvon tallentaminen takaisin muistiin. Näin mitattaessa voidaan helposti vertailla laskutoimituksiin kuluvaa suoritusaikaa eri muisteista suoritettaessa. Seuraavissa laskutoimitusten suorituskykymittauksissa onkin mittaukset suoritettu niin FPGA-piirin sisäisestä BRAM-muistista, kuin ulkoisesta 66 MHz:n kellotaajuudella toimivasta DDR-SDRAM-muistista. Ulkoisesta muistista suoritettaessa ei ole käytetty välimuistia lainkaan, jolloin saadaan hitaimman mahdollisen suorituksen tulos. Normaalisti välimuistin kanssa ohjelmakoodia suoritettaessa jäädään siten kokonaissuoritusajassa aina näiden kahden tuloksen välimaastoon.

Vielä kalibroinninkin jälkeen BRAM-muistista suoritettaessa mittaustuloksissa esiintyy yhden tai kahden kellojakson heittoa mittauskohteesta riippuen, joka johtuu osaltaan mittaustavasta ja MicroBlaze:n kolmetasoisen liukuhihnan toiminnasta. Myös käytetyn muistin tyyppi vaikuttaa mittaushäiriöön, koska ulkoisesta DDR-SDRAM-muistista suoritettaessa mittauksissa esiintyy jopa 20 kellojakson heittoa johtuen esimerkiksi

muistin virkistyksestä [34]. Hitaampaa ulkoista muistia käytettäessä sisäistä muistia voidaan käyttää välimuistina, jolloin ensimmäinen käskyjen tai muuttujien haku muistista kestää kauemmin. Ohjelmasilmukoita suoritettaessa tosin tämän ensimmäisen hakukerran jälkeen käskyt ja muuttujat löytyvät välimuistista, jolloin ne voidaan suorittaa täydellä nopeudella.

Reaaliaikajärjestelmissä, joissa vaaditaan hyvin tarkkaa ennustettavuutta, on liukuhihnan ja välimuistin käyttöä kartettu juuri niiden aiheuttaman stokastisuuden vuoksi. Liukuhihnan ja välimuistin analyysimenetelmiä on kuitenkin paljon tutkittu ja kehitetty viimeaikoina. Näitä analyysimenetelmiä ovat esimerkiksi Sebek:n teknisessä raportissa analyysimenetelmistä liukuhihnoille reaaliaikajärjestelmissä sekä Petters:n ja Färber:n julkaisussa WCET-analyysistä (Worst Case Execution Time) välimuistillisille ja liukuhihnoitetuille prosessoreille [34],[35]. Tässä työssä ei tosin perehdytä välimuistin eikä liukuhihnan aiheuttamaan stokastisuuteen muuten kuin ottamalla huomioon liukuhihnan aiheuttama pieni virhe.

3.4 Suorituskykymittaustulokset ja analysointi

Suorituskykymittaukset toteutettiin kappaleen 3.2 mukaisella prosessorijärjestelmällä ja kappaleen 3.3 mukaisella mittausmenetelmällä sekä ohjelmarakenteella. Ulkoisesta muistista suorituksia mitattaessa ohjelman linkityssääntöjä muutettiin XPS-työkalulla niin, että ohjelma on kokonaisuudessaan ulkoisen muistin alueella. Prosessorille suoritettavaan muotoon käännetty ohjelma siirrettiin MDM-lohkon avulla ulkoiselle muistille ja siirryttiin ajamaan ohjelmaa sieltä.

3.4.1 Kokonaislukuoperaatiot

Yhteen- ja vähennyslaskun suorittamiseen prosessorilta kuluu yksi kellojakso. Taulukossa 13 on mitattu edellä esitetyllä suorituskykymittaustavalla ulkoisesta ja sisäisestä muistista suoritettavia kokonaislukulaskuoperaatioita, kun laitteistotuki kertolaskulle, jakolaskulle ja bittien siirrolle on käytössä. Sisäisen muistin käsittely vaatii kaksi kellojaksoa, joten käskyjen hakeminen, kahden parametrin hakeminen ja yhden kirjoittaminen vaativat 8 kellojaksoa. Ulkoisesta muistista suoritettaessa yhteen- ja vähennyslaskut vaativat 88 kellojaksoa, josta yksi kellojakso kuluu laskutoimituksen

suorittamiseen. 87 kellojaksolla tapahtuu siis 7 muistioperaatiota, eli 4 käskyn ja 2 parametrin hakua sekä yksi tallennus, jotka ovat siten keskimäärin 12 kellojakson pituisia. Kertolaskun tapauksessa suoritusnopeus on liukuhihnan ansiosta ja mittauksen perusteella sama kuin yhteen- ja vähennyslaskuilla.

Jakolaskun tapauksessa laskutoimitukseen kuluu 34 kellojaksoa, joten muistioperaatioille jää silloin 66 kellojaksoa. Jakojäännöstä laskettaessa suoritetaan kolme seuraavaa operaatiota: jakolasku, kertolasku ja vähennyslasku, jotka vievät yhteensä 38 kellojaksoa. Muistioperaatioille jakojäännöstä suorittaessa jää siis 76 kellojaksoa.

Taulukko 13. MicroBlaze:n kokonaislukujen laskutoimitusten vaatimat kellojaksot suoritettaessa sisäisestä ja ulkoisesta muistista, kun laitteistokertoja ja jakaja on käytössä.

	Kellojaksot	
Operaatio	BRAM	DDR-SDRAM
a = b + c	8	88
a = b – c	8	88
a = b * c	10	88
a = b / c	42	100
a = b % c	45	114

Yhteenvetona edellisistä kokonaislukujen laskutoimituksista voidaan päätellä, että ulkoisen muistin käsittely vie noin 10 kellojaksoa. Tämän johdosta laskuoperaatiot ulkoisesta muistista suoritettaessa ovat siis noin kymmenen kertaa hitaampia kuin sisäisestä muistista.

3.4.2 Liukulukuoperaatiot

Float-tyyppisten liukulukulaskutoimitusten vaatimat kellojaksot on koottu taulukkoon 14, kun laitteistotuki kertolaskulle, jakolaskulle ja bittien siirrolle on käytössä. Koska MicroBlaze on kokonaislukutyyppinen prosessori, liukulukujen laskutoimitukset suoritetaan sarjalla kokonaislukuoperaatioita. Kokonaislukuoperaatiosarjan suorittava ohjelmakoodi täytyy hakea ulkoisesta muistista vielä käsky kerrallaan, jolloin jokaisen käskyn suorittaminen on noin kymmenen kertaa hitaampaa. Lopputuloksena on siis, että ulkoisesta muistista liukulukuoperaatiot vievät noin kymmenen kertaa enemmän kellojaksoja, kuin sisäisestä muistista suoritettaessa.

Taulukko 14. Float-tyyppisiä liukulukuja käytettäessä laskutoimitusten tarvitsemat kellojaksot, kun laitteistotuki jakolaskulle, kertolaskulle ja bittien siirrolle on käytössä.

	Kellojaksot	
Operaatio	BRAM	DDR-SDRAM
a = b + c	84	984
a = b – c	110	1327
a = b * c	78	841
a = b / c	278	2992

3.4.3 Kiinteänpilkun- ja skaalausmenetelmien operaatiot

Fraktionaalilukujen- ja skaalattujen kokonaislukujen laskutoimitusten tarvitsemat kellojaksot tarvittavilla bittisiirroksilla on taulukossa 15. Luvut on skaalattu käyttäen 2:n potenssia sekä jakolasku, kertolasku ja bittien siirto on toteutettu laitteistotasolla. Yhteen- ja vähennyslaskut voidaan suorittaa yhtä nopeasti kuin kokonaisluvuilla, koska bittisiirtoja ei tarvita. Kertolaskuilla ja jakolaskuilla tarvitaan kokonaislukulaskutoimituksen lisäksi bittisiirrot, jotka voidaan tosin suorittaa nopeasti käyttäen sille laitteistotukea. Laitteistotuen ansiosta kertolasku ja jakolasku tarvitsevatkin kokonaislukulaskutoimituksiin verrattuna vain yhden ylimääräisen kahden kellojakson suorituksen.

Taulukko 15. Kiinteänpilkun ja skaalattuja kokonaislukuja käytettäessä laskutoimitusten tarvitsemat kellojaksot, kun laitteistotuki jakolaskulle, kertolaskulle ja bittien siirrolle on käytössä.

	Kellojaksot	
Operaatio	BRAM	DDR-SDRAM
a = b + c	8	88
a = b – c	8	101
a = b * c	12	101
a = b / c	44	114

3.4.4 Trigonometriset funktiot

Taulukko 16 sisältää trigonometristen GNU:n C-kirjastofunktioiden suoritusajat, kun laitteistotuki kertolaskulle, jakolaskulle ja bittien siirrolle on käytössä. Operaatiot on suoritettu kulman arvoilla 0 - 2π ja mitattu niistä suurimman sekä pienimmän kellojaksojen lukumäärän. Muistina on käytetty ainoastaan nopeaa BRAM-muistia. Kellojaksot alkavat kasvaa kulman arvon kasvaessa minimiarvosta ja maksimiarvojen kohdalla kellojaksojen kasvu hidastuu. Trigonometriset funktiot käyttävät liukulukuja eikä trigonometristen funktioiden algoritmien tarkkuutta voida säätää suoritusnopeuden

parantamiseksi. GNU C-kirjaston funktiot kuluttavat ohjelmamuistia 8-25 kt riippuen siitä kuinka monta ja mitä funktiota käytetään.

Taulukko 16. Trigonometristen funktioiden tarvitsemat kellojaksot, kun käytetään GNU libc-kirjaston liukulukuja käyttäviä trigonometrisiä funktioita. Nämä tulokset ovat saatu, kun MicroBlaze-prosessorissa on käytössä laitteistotuki jakolaskulle, kertolaskulle ja bittien siirrolle.

		Kellojaksot	
	Funktio	Min	Max
Double	sin	164	15992
	cos	170	16011
	atan	114	16646
Float	sinf	1090	3448
	cosf	126	3252
	atanf	201	2959

3.4.5 Suoritusaikojen analysointi

Aritmetiikkaoperaatioiden suoritusnopeusmittausten (taulukot 13-16) perusteella nähdään ulkoisesta muistista suorituksen olevan noin kymmenen kertaa hitaampaa kuin sisäisestä muistista. Liukulukuoperaatiot (taulukko 14) ovat huomattavasti kokonaislukuoperaatioita hitaampia ja kiinteänpilkun sekä skaalattujen kokonaislukujen olevan huomattavasti nopeampi tapa käsitellä desimaalilukuja. Liukulukuoperaatiot ja trigonometriset funktiot ovat kirjastofunktioilla toteutettaessa todella hitaita.

Suorituskykymittausten perusteella suuremmat ohjelmakokonaisuudet on kannattavin jakaa sisäisen ja ulkoisen muistin kesken. Nopean sisäisen muistin vähyyden vuoksi sisäisestä muistista on kannattavin suorittaa vain ohjelmiston nopeimmat osat ja jättää hitaammat osat ulkoiseen muistiin. Ulkoista muistia käytettäessä sisäisestä muistia on kannattavaa jättää myös välimuistiksi.

3.5 µC/OS-II:n palveluiden suoritusaikojen mittaaminen

µC/OS-II reaaliaikakäyttöjärjestelmän suorituskykymittausten mittausjärjestelmä on kappaleen 3.2 mukainen, kuten edellisissäkin suorituskykymittauksissa. Mittausmenetelmä on myös kappaleen 3.3 mukainen, eli lukemalla tehtävän suorituksen alussa ja lopussa laskurin arvo sekä vähentämällä näiden erotuksesta vielä laskurin lukemiseen käytetty aika. Nyt mitataan tosin µC/OS-II reaaliaikakäyttöjärjestelmän palveluiden suoritusaikoja ja käyttäen ainoastaan sisäistä muistia. Käyttöjärjestelmän

palveluiden suorittamiseen kuluvaa aikaa kutsutaan järjestelmän aiheuttamaksi viiveeksi. Mitä suurempi järjestelmän aiheuttama viive, sitä hitaampaa on järjestelmän vastaaminen tapahtumiin. Mittaukset suoritetaan pääosin keskeytykset kiellettyinä, jotta kellokeskeytys ei pääse mittauksen väliin. Tosin palvelut, jotka käyttävät kellokeskeytyksen palveluita, tarvitsevat kellokeskeytyksen olevan sallittuna.

Käyttöjärjestelmästä mitataan kellokeskeytyksen suoritusaika samaan taskiin palattaessa ja toiseen taskiin vaihdettaessa. Käyttöjärjestelmän palveluista mitataan keskeytysten kieltämisen ja sallimisen, semaforien, mutexien, tapahtumalippujen, viestilaatikoiden ja viestijonojen suoritusaikoja.

Kellokeskeytyksen suoritusaika voidaan mitata generoimalla ohjelmasta keskeytys. Käytettävän keskeytyskäsittelijän (opb_intc v1.00.c) avulla on mahdollista suorittaa keskeytyksiä ohjelmasta käsin niin kauan kunnes se kerran estetään. Kellokeskeytyksestä palaaminen samaan, ennen keskeytystä suorituksessa olleeseen taskiin, mitattiin suoraan generoimalla vain kellokeskeytys omassa taskissaan. Korkeamman prioriteetin taskiin vaihtaminen onnistuu esimerkiksi semaforin avulla. Korkeamman prioriteetin taskissa jäädään odottamaan semaforia ja alemman prioriteetin taskissa vapautetaan semafori suoraan ilman käyttöjärjestelmän palvelukutsua OSSemPost.

Keskeytysten kieltämisen sekä sallimisen suoritusaika OS_ENTER_CRITICAL- ja OS_ENTER_CRITICAL-funtioilla voidaan mitata suoraan samassa taskissa tapahtuvilla kutsuilla.

Semaforien luontifunktion OSSemCreate, odotusfunktion OSSemPend, tarkistusfunktion OSSemAccept ja varausfunktion OSSemPost suoritusnopeuden mittaminen onnistuu samassa taskissa tapahtuvilla kutsuilla muodostaen tarvittavan tilanteen sekä asetukset ennen mittausta. Signalointi tai resurssien vapauttaminen siten, että semaforia odottava taski pääsee ajoon voidaan mitata kahden taskin avulla. Toinen taskeista jää odottamaan semaforia ja toisessa taskeista aloitetaan mittaus sekä vapautetaan semafori. Semaforin vapauttamisessa käyttöjärjestelmä huomaa, että korkeamman prioriteetin taski odottaa sitä ja vaihtaa taskien kontekstit sekä siirtyy

ajamaan semaforia odottanutta taskia. Semaforia odottanut taski pysäyttää mittauksen ja taskin vaihtoon kulunut aika saadaan mitattua.

Mutex-semaforit ovat semaforien kanssa hyvin samanlaisia, joten suorituskykymittaukset voidaan suorittaa samalla tavalla. Mutex-semaforien luontifunktion OSMutexCreate, odotusfunktion OSMutexPend, tarkistusfunktion OSMutexAccept ja varausfunktion OSMutexPost mittaaminen onnistuu samassa taskissa tapahtuvilla kutsuilla. Erikoistapausta jolloin tarvitaan prioriteetin inversion ratkaisemista ei tässä työssä mitata, koska sitä ei tarvita protolaitteiston ohjelmistossa.

Tapahtumalippujen luontifunktion OSFlagCreate, odotusfunktion OSFlagPend ja tarkistusfunktion OSFlagAccept eri asetuksilla voidaan mitata edellisten tavoin samassa taskissa tapahtuvilla kutsuilla. Myös signalointi tai resurssien vapauttamisen suorittama taskin vaihto voidaan mitata edellisten tavoin.

Viestilaatikoiden luontifuntion OSMboxCreate, odotusfunktion OSMboxPend, tarkitusfunktion, OSMboxAccept lähetysfunktion OSMboxPost ja lähetysfunktion jolla on lisäoptioita OSMboxPostOpt mittaukset voidaan suorittaa edellisten tavoin samassa taskissa tapahtuvilla kutsuilla. Signaloinnin tai resurssien vapauttamisen suorittaman taskin vaihdon mittaaminen suoritettiin käyttäen lähetysfunktiota OSMboxPost sekä OSMboxPostOpt kahdella eri asetuksella samoin tavoin kuin edellisissä mittauksissakin.

Viestijonojen luontifunktion OSQCreate, odotusfunktion OSQPend, tarkistusfunktion OSQAccept, lähetysfunktion OSQPost, viestijonon ensimmäiseksi sijoittavan lähetysfunktion OSQPostFront ja OSQPostOpt lähetysfunktion jolla on lisäoptioita mittaus voidaan suorittaa edellisten tavoin samassa taskissa tapahtuvilla kutsuilla. Signalointi tai viestinjonon lähetys, joka aiheuttaa taskien vaihdon, suoritusnopeuden mittaus suoritetaan edellisten mittausten tavoin, mutta jokaisella OSQPost-, OSQPostOpt- ja OSQPostFront-funktiolla sekä OSQPostOpt-funktion lisäoptioilla.

3.6 µC/OS-II:n palveluiden suoritusaikojen mittaustulokset ja analysointi

µC/OS-II reaaliaikakäyttöjärjestelmän suorituskykymittaukset toteutettiin kappaleen 3.2 mukaisella prosessorijärjestelmällä ja kappaleen 3.5 mukaisella ohjelmarakenteella. Mittaukset suoritettiin käyttäen ainoastaan nopeaa BRAM-muistia. Muistin vähyyden vuoksi kaikkia palveluita ei voitu ottaa käyttöön samanaikaisesti, joten mittaukset suoritettiin valitsemalla palveluita käyttöön eri ajokerroilla muistin sallimissa rajoissa.

3.6.1 Kellokeskeytys

Kellokeskeytyksen suoritus kestää mittauksen perusteella 530 kellojaksoa silloin, kun kellokeskeytyksestä palatessa palataan takaisin ennen keskeytystä suorituksessa olleeseen taskiin Jos kellokeskeytyksestä siirrytään ajamaan prioriteetiltaan korkeinta taskia, joka on valmiina ajoon, niin uuteen taskiin siirtyminen kestää 634 kellojaksoa. Taulukkoon 17 on merkitty edelliset kellokeskeytysten suoritusajat sekä laskettu kellojaksoja vastaavat suoritusajat, kun prosessoria suoritetaan 100 MHz:llä.

Taulukko 17. Kellokeskeytysten suoritusajat, kun palataan takaisin ajossa olleeseen taskiin ja kun siirrytään suuremman prioriteetin omaavaan taskiin. Kellojaksoja vastaavat suoritusajat on myös ilmoitettu, kun prosessoria suoritetaan 100MHz:llä ja käytettäessä ainoastaan sisäistä BRAM-muistia.

Kellokeskeytys	Kellojaksot	Aika (100MHz:llä)
Palaaminen takaisin samaan ajossa olleeseen taskiin	530	5.30µs
Palaaminen suurimman prioriteetin taskiin	634	6.34µs

3.6.2 Keskeytysten kieltäminen

Yksinkertaisin tapa tiedon suojaamiseen on keskeytysten kieltäminen. Taulukkoon 18 on mitattu keskeytysten kieltämiseen ja sallimiseen käytettyjen OS_ENTER_CRITICAL ()- ja OS_EXIT_CRITICAL()-funktioiden suoritusajat. µC/OS-II-käyttöjärjestelmän MicroBlaze versiossa käytetään vain kolmannen tason suojausta, joten tallennuspaikka prosessorin tilarekisterin tallentamiseen täytyy olla käytettävissä.

Taulukko 18. Keskeytysten kieltämiseen ja sallimiseen tarvittavat kellojaksot, kun käytetään käyttöjärjestelmän OS_ENTER_CRITICAL – ja OS_EXIT_CRITICAL-funktioita.

Toiminto	Kellojaksot
OS_ENTER_CRITICAL()	20
OS_EXIT_CRITICAL()	22

3.6.3 Semaforit

Semaforit ovat käyttöjärjestelmän yleisimpiä palveluita ja suoritusajat ovat sen yksinkertaisuuden vuoksi hyvin nopeita. Taskista toiseen vaihto sujuu käyttöjärjestelmän muidin palveluihin verrattaessa kaikkein nopeiten. Taulukkoon 19 on kerätty semaforien eri tilanteiden vaatimat suoritusajat.

Taulukko 19. Semaforifunktioiden vaatimat kellojaksot eri tilanteissa selityksineen sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettaessa.

Funktio	Toiminto	Kellojaksot OS_ARG_CHK_EN = 0	Aika (100MHz:llä)	Kellojaksot OS_ARG_CHK_EN = 1	Aika (100MHz:llä)	Tietoja
OSSemCreate		98	980 ns	98	980 ns	Luominen onnistui
OSSemPend	OS_NO_ERR	88	890 ns	92	920 ns	Resurssi vapaana
OSSemAccept	cnt > 1	74	740 ns	77	770 ns	Resursseja vapaana > 1
OSSemAccept	cnt = 0	73	730 ns	76	760 ns	Ei resursseja vapaana
OSSemPost	OS_NO_ERR	80	800 ns	82	820 ns	Semaforin vapauttaminen onnistui
OSSemPost & OSSemPend		431	4.31 µs	434	4.34 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto

3.6.4 Mutex-semaforit

Taulukossa 20 on mutex-semaforien suoritusajat kellojaksoissa sekä kellojaksoja vastaavat suoritusajat 100 MHz:llä selityksineen. Taulukon suoritusarvoista nähdään, että mutex-semaforit ovat hiukan hitaampia kuin tavalliset semaforit johtuen mutex-semaforien monimutkaisemmasta rakenteesta. Monimutkaisen rakenteen vuoksi myös taskin vaihtoon kuluva aika on kaikista käyttöjärjestelmän palveluista hitain.

Taulukko 20. Mutex-semaforifunktioiden vaatimat kellojaksot eri tilanteissa selityksineen sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettaessa.

Funktio	Toiminto	Kellojaksot OS_ARG_CHK_EN = 0	Aika (100MHz:llä)	Kellojaksot OS_ARG_CHK_EN = 1	Aika (100MHz:llä)	Tietoja
OSMutexCreate	OS_NO_ERR	119	1.19 µs	124	1.24 µs	Luominen onnistui
OSMutexPend	OS_NO_ERR	101	1.01 µs	104	1.04 µs	Resurssi vapaana
OSMutexAccept	OS_NO_ERR	82	820 ns	84	840 ns	Resurssi vapaana
OSMutexAccept	OS_NO_ERR	82	820 ns	84	840 ns	Resurssi ei vapaana
OSMutexPost	OS_NO_ERR	110	1.10 µs	112	1.12 µs	Resurssi vapaana
OSMutexPost & OSMutexPend		549	5.49 µs	551	5.51 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto

3.6.5 Tapahtumaliput

Taulukkoon 21 on koottu tapahtumalippujen suoritukseen kuluvat kellojaksot sekä

kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettaessa. Tapahtumaliput ovat ainoa käyttöjärjestelmän palvelu, jonka avulla voidaan signaloida taskeja useisiin tapahtumiin. Suoritusajat ovat tapahtumalipuilla eri parametreilla hyvin samanpituisia.

Taulukko 21. Tapahtumalippujen vaatimat kellojaksot eri tilanteissa selityksineen sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettessa.

Funktio	Toiminto	Kellojaksot OS_ARG_CHK_EN = 0	Aika (100MHz:llä)	Kellojaksot OS_ARG_CHK_EN = 1	Aika (100MHz:llä)	Tietoja
OSFlagCreate	OS_NO_ERR	94	940 ns	96	960 ns	Luominen onnistui
OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME						
OSFlagPend	OS_NO_ERR	139	1.39 µs	142	1.42 µs	Lippu saatavilla
OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME						
OSFlagPend	OS_NO_ERR	141	1.41 µs	144	1.44 µs	Lippu saatavilla
OS_FLAG_WAIT_CLR_ALL + OS_FLAG_CONSUME						
OSFlagPend	OS_NO_ERR	136	1.36 µs	139	1.39 µs	Lippu saatavilla
OS_FLAG_WAIT_CLR_ANY + OS_FLAG_CONSUME						
OSFlagPend	OS_NO_ERR	132	1.32 µs	139	1.39 µs	Lippu saatavilla
OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME						
OSFlagAccept	OS_NO_ERR	119	1.19 µs	135	1.35 µs	Lippu saatavilla
OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME						
OSFlagAccept	OS_NO_ERR	122	1.22 µs	125	1.25 µs	Lippu saatavilla
OS_FLAG_WAIT_CLR_ALL + OS_FLAG_CONSUME						
OSFlagAccept	OS_NO_ERR	118	1.18 µs	121	1.21 µs	Lippu saatavilla
OS_FLAG_WAIT_CLR_ANY + OS_FLAG_CONSUME						
OSFlagAccept	OS_NO_ERR	112	1.12 µs	115	1.15 µs	Lippu saatavilla
OSFlagPost & OSFlagPend		521	5.21 µs	524	5.24 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto

3.6.6 Viestilaatikot

Taulukkoon 22 on koottu viestilaatikoiden suoritukseen tarvitsemat kellojaksot sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettaessa. Viestilaatikot on kätevä tapa lähettää tietoa taskista tai keskeytysaliohjelmasta. Lähetysmahdollisuutta useammalle taskille tarvitaan myös usein sekä viestilaatikoiden käsittely on nopeaa.

Taulukko 22. Viestilaatikoiden vaatimat kellojaksot eri tilanteissa selityksineen sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettaessa.

Funktio	Toiminto	Kellojaksot OS_ARG_CHK_EN = 0	Aika (100MHz:llä)	Kellojaksot OS_ARG_CHK_EN = 1	Aika (100MHz:llä)	Tietoja
OSMboxCreate	NO_ERR	100	1.00 µs	100	1.00 µs	Luominen onnistui
OSMboxPend		94	940 ns	96	960 ns	Viesti laatikosta
OSMboxAccept	NO_ERR	72	720 ns	74	740 ns	Viesti laatikosta
OSMboxPost	OS_NO_ERR	83	830 ns	86	860 ns	Viesti lähetettiin
OSMboxPostOpt	OS_NO_ERR	87	870 ns	91	910 ns	Viestin lähettäminen onnistui
OSMboxPost & OSMboxPend		444	4.44 µs	448	4.48 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto
OS_POST_OPT_NONE						
OSMboxPostOpt & OSMboxPend		450	4.50 µs	454	4.54 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto
OS_POST_OPT_BROADCAST						
OSMboxPostOpt & OSMboxPend		456	4.56 µs	460	4.60 µs	Signalointi taskista toiseen jolloin tapahtuu kontekstin vaihto

3.6.7 Viestijonot

Taulukkoon 23 on koottu viestijonojen suoritusajat selityksineen. Viestijonojen käsittelyyn on monia funktioita ja niiden suoritusajat ovat hyvin lähellä toisiaan. Myös taskin vaihto sujuu nopeasti niin yhdelle, kuin useammallekin taskille viestiessä.

Taulukko 23. Viestijonojen vaatimat kellojaksot eri tilanteissa selityksineen sekä kellojaksoja vastaavat suoritusajat prosessoria 100 MHz:llä suoritettessa.

Funktio	Toiminto	Kellojaksot OS_ARG_CHK_EN = 0	Aika (100MHz:llä)	Kellojaksot OS_ARG_CHK_EN = 1	Aika (100MHz:llä)	Tietoja
OSQCreate	NO_ERR	170	1.70 µs	172	1.72 µs	Luominen onnistui
OSQPend	OS_NO_ERR	100	1.00 µs	109	1.09 µs	Viesti saatavana
OSQAccept	OS_NO_ERR	96	960 ns	99	990 ns	Viesti saatavana
OSQPost	OS_NO_ERR	101	1.01 µs	104	1.04 µs	Viestin lähettäminen onnistui
OSQPost & OSQPend	OS_NO_ERR	443	4.43 µs	446	4.46 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto
OS_POST_OPT_BROADCAST						
OSQPostOpt		111	1.11 µs	116	1.16 µs	Postaus onnistui
OSQPostOpt & OSQPend		456	4.56 µs	461	4.61 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto
OS_POST_OPT_NONE						
OSQPostOpt		113	1.13 µs	118	1.18 µs	Postaus onnistui
OSQPostOpt & OSQPend		450	4.50 µs	455	4.55 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto
OS_POST_OPT_FRONT						
OSQPostOpt		116	1.16 µs	121	1.21 µs	Postaus onnistui
OSQPostOpt & OSQPend		450	4.50 µs	455	4.55 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto
OS_POST_OPT_FRONT + OS_POST_OPT_BROADCAST						
OSQPostOpt		116	1.16 µs	121	1.21 µs	Postaus onnistui
OSQPostOpt & OSQPend		456	4.56 µs	461	4.61 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto
OSQPostFront	OS_NO_ERR	106	1.06 µs	109	1.09 µs	Postaus onnistui
OSQPostFront & OSQPend	OS_NO_ERR	443	4.43 µs	446	4.46 µs	Korkeimman prioriteetin taski ajoon, kontekstin vaihto

3.6.8 Suoritusaikojen analysointi

Kellokeskeytysten suoritusajoista (taulukko 17) nähdään kuinka paljon kuormitusta

jaksollinen käyttöjärjestelmän pakollinen palvelu aiheuttaa. Tarvittaessa pienin mahdollinen kellokeskeytysten väli voidaan arvioida kellokeskeytyksen ja prioriteetiltaan korkeiden taskien yhtäjaksoisista suoritusajoista.

Keskeytysten kieltämiseen ja sallimiseen ei kulu paljon kellojaksoja (taulukko 18), joten niitä voidaan käyttää yksinkertaiseen ja nopeaan jaetun datan suojaamiseen. Käyttöjärjestelmän palveluista ja etenkin käyttöjärjestelmän palveluiden suorittamien taskien vaihtojen (4 – 6 µs) suoritusajoista on hyvä tietää, että keskeytykset eivät ole kiellettynä koko aikaa, vaan vain tarvittavissa kohdissa.

Järjestelmän palveluiden suoritusaikoja (taulukot 17-23) voidaan käyttää prosessorijärjestelmän suunnittelussa, kun mietitään kuinka tehokkaan prosessorin järjestelmä vaatii sekä kuinka paljon ominaisuuksia järjestelmään voidaan sisällyttää. Suoritusaikojen avulla voidaan myös vertailla eri reaaliaikakäyttöjärjestelmien tehokkuuksia sekä ominaisuuksia.

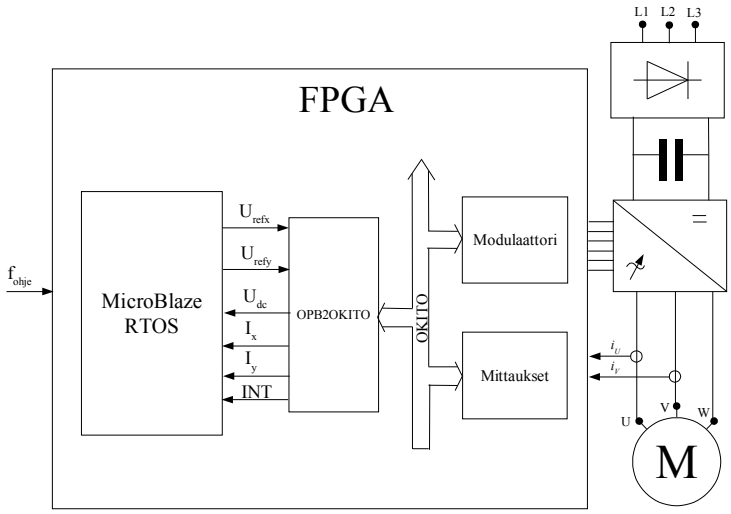
Esimerkiksi Express Logic:n ThreadX reaaliaikakäytöjärjestelmässä vapaan semaforin saaminen suoritetaan 0.3 µs:ssa, mutexin saaminen 0.4 µs:ssa ja tapahtumalipun saaminen 0.4 µs:ssa. ThreadX:ssä säikeen (thread) vaihto suoritetaan semaforilla 3 µs:ssa, mutexilla 3.6 µs:ssa ja tapahtumalipuilla 4.3 µs:ssa. [45] Verrattaessa näitä arvoja mitattuihin µC/OS-II:n (ilman argumenttien testausta, OS_ARG_CHK_EN) semaforin saamiseen 980 ns:ssa, mutexin saamiseen 1.19 µs:ssa tapahtumalipun saamiseen 1.18 – 1.41 µs:ssa sekä taskin vaihtoon semaforilla 4.31 µs:ssa, mutexilla 5.49 µs:ssa ja tapahtumalipulla 5.21 µs:ssa, huomataan ThreadX:ssa vastaaviin suorituksiin kuluvan noin 0.6 - 2 µs vähemmän aikaa. Tämä ero johtuu täysin käyttöjärjestelmien sisäisen rakenteen eroista, koska molempia suoritettiin samalla prosessorilla.

3.7 Järjestelmän testaus osana taajuudenmuuttajaa

Taajuusmuuttajan toimintojen toteuttaminen yhdellä FPGA-piirillä jaettiin niin, että muuttajaa ohjaava modulaattori, mittauslohko ja säädön kriittisin osa eli ylivirtarajoitus, toimivat omina IP-lohkoinaan. Loput säädöstä toteutettiin FPGA-piirillä olevalla

sulautetulla prosessorilla. Prosessori oli nopeusmittauksissa käytetyn prosessorijärjestelmän (kuva 19) mukainen, mutta lisättynä OPB-väylään liitetyllä OPB2OKITO-rajapinnalla ja ilman ulkoista muistia. Testausjärjestelmä on kuvan 20 mukainen, jossa on esitetty myös taajuudenmuuttajan osat. Kuvan mukaisesti FPGA-piirillä olevat lisälaitteet liitettiin yhteen OKITO-väylän [5] avulla ja OPB2OKITO-rajapinnan kautta prosessori liitettiin samaan väylään muiden piirillä olevien lisälaitteiden kanssa.

Xilinx:n tarjoamaa IPIF-rajapintaa käytettiin OPB2OKITO-rajapinnassa helpottamaan prosessorin liittämistä OKITO-väylään. EDK:n versiolla 6.1 IPIF:n käyttämiseen ei ollut valmiita työkaluja eikä sen käyttöönottaminen ollut aivan suoraviivaista. EDK:n seuraavassa 6.3i versiossa IPIF:n käyttöönottamista oli kehitelty huomattavasti, mutta samalla liityntärajapintaa oli muutettu. Uuden version myötä IPIF:n käyttöönottaminen oli helpompaa ja riittävällä IPIF-rajapinnan ja OKITO-väylän debuggauksella saatiin tarpeelliset muutokset tehtyä, jotta tiedonsiirto väylien välillä onnistui.



Kuva 20. Yhden piirin sähkökäytön säätöjärjestelmän rakenne FPGA-piirillä ja FPGA-piirille sulautetulla prosessorijärjestelmällä toteutettuna. Moottorin vaiheista mittaukset menevät suoraan FPGA-piirillä sijaitsevalle mittauslohkolle. Mittauslohkolta virta-arvot ja välipiirin jännite siirretään OPB2OKITO-liityntärajapinnan kautta prosessorille, jossa varsinainen säätöalgoritmi laskee jänniteohjeet. Jänniteohjeet siirtyvät modulaattorin kautta vaihtosuuntaajalle kuudeksi kytkinohjeeksi.

Koko järjestelmän testaamiseen käytettiin LTY:n ja Vaconin jo aiemmassa tutkimushankkeessa kehitettyä FPGA-prototyyppikorttia. FPGA:na kortilla on Xilinx:n

XC2V3000. Kortti liitettiin taajuudenmuuttajan power-korttiin ja siinä käytetyn liitynnän kautta kulkee myös mittausdata FPGA-kortille sekä kytkinohjeet power-kortille. FPGA-piirin tarvitsema 1.5 V:n jännite tehdään FPGA-kortilla. Kortilla on galvaanisesti erottamaton JTAG-liityntä, kuitulinkki ylemmältä säätöjärjestelmältä sekä nopeat kuitulinkit. Näiden lisäksi FPGA:n käyttämättömiä pinnejä on tuotu piikkirimalle laajennus- ja debuggaustarpeita varten. Ohjelmointimuistina on erillinen 4 Mb:n XC18V04 PROM-piiri [10].

Nopeassa kuitulinkissä lähetinvastaanotinmoduulina on HFBR-5911-piiri, jonka maksimitiedonsiirtonopeus on 1.5 Gt/s [36]. Maksiminopeuden FPGA-kortilla määrää FPGA:n kello. Tämä nopea kuitulinkki on tarkoitettu lähinnä tulevaisuuden tarpeisiin, joten pääasiallisena liityntänä ylempään säätöjärjestelmään toimiikin hitaampi linkki (15 Mt/s). Tässä hitaammassa linkissä käytetään lähettimenä HFBR-1528 ja vastaanottimena HFBR-2538 piirejä [37].

Prototyypissä toteutettiin moottorisäätö kokonaisuudessaan FPGA-piirillä kuvan 3 mukaisesti. Prototyypissä on seuraavanlaiset toiminnot:

- Modulaattori
- Läpisyttymis- ja ylivirtarautasuojaukset
- Skalaarisäätö
    - Nopeussäätö nopeuden estimoinnilla
    - Virtaraja
    - Välipiirin jänniteraja
    - Referenssien rampitukset
    - IR-kompensointi
    - Kentänheikennys
    - Vääntömomentin laskenta
- Käyttöliittymä
- Debuggausliityntä PC:lle
- ”Sovellusohjelmat” (näitä varten varattu ohjelmistoon tila/runko)

Tämä protolaitteisto on viiden hengen ryhmän tulos, jossa tämän työn puitteissa on osallistuttu käyttöliittymän ohjelmointiin ja debuggaukseen, debuggausliitynnän

toteuttamiseen PC:lle, OPB2OKITO-rajapinnan ja koko säätöohjelmiston debuggaukseen, prosessorin konfigurointiin, resurssien kulutuksen selvitykseen ja suooritusarvomittauksiin.

Prototyypin FPGA-piirin LUT-resursseista kulutetaan 19 % ja BRAM-resursseista 35 %. Resursseista sulautetun MicroBlaze-prosessorin osuus on 7 % ja muun VHDL-toteutuksen osuus 12 %. Prosessorilla suoritetaan µC/OS-II-reaaliaikakäyttöjärjestelmää, jonka periodisten taskien laajennuksen avulla suoritetaan säädön ohjausalgoritmeja 100 µs:n, 1 ms:n ja 100 ms:n suoritusvälein suoritettavissa taskeissa. 100 µs:n taskissa suoritetaan jännitereferenssin laskenta. 1 ms:n taskissa suoritetaan virran ja välipiirin rajoitukset, kentänheikennys, vääntömomentin estimointi ja nopeussäätö. 100 ms:n taskissa suoritetaan taajuusohjeen muutoksen rajoittaminen. Ohjelmamuistia kulutetaan 24.5 kt ja datamuistia 11.8 kt.

Prosessoria ohjelmisto kuormittaa käyttöjärjestelmän statistiikkapalvelun mukaan ajossa (moduloimassa) 15 % ja pysäyksissä (ei moduloimassa) 5 %. Statistiikkapalvelulla on käytössä oma taski, joka arvioi prosessorin kuormitusta idle-laskurin ja sen suurimman arvon avulla. Statistiikkapalvelun resoluutio on 1 % [1]. Taskien tarvitsemat kellojaksot mitattiin protolaitteistolla suorituskykymittausten tapaan. Mittaus aloitettiin periodisen suorituksen alussa ja lopetettiin, kun taski palasi takaisin odottamaan periodista suoritusta. Taskien tarvitsemat kellojaksot ja suoritusajat 100 MHz:llä BRAM-muistista suoritettaessa on esitetty taulukossa 24.

Taulukko 24. Protolaitteistolla BRAM-muistista suoritettujen t100us-, t100ms- ja t1ms-taskien tarvitsemat kellojaksot sekä suoritusajat 100 MHz:llä suoritettaessa.

Taski	Kellojaksot	Aika (100MHz:llä)	Suoritusväli
t100ms	233	2.33 µs	100 ms
t1ms	734	7.34 µs	1 ms
t100us	429	4.29 µs	100 µs

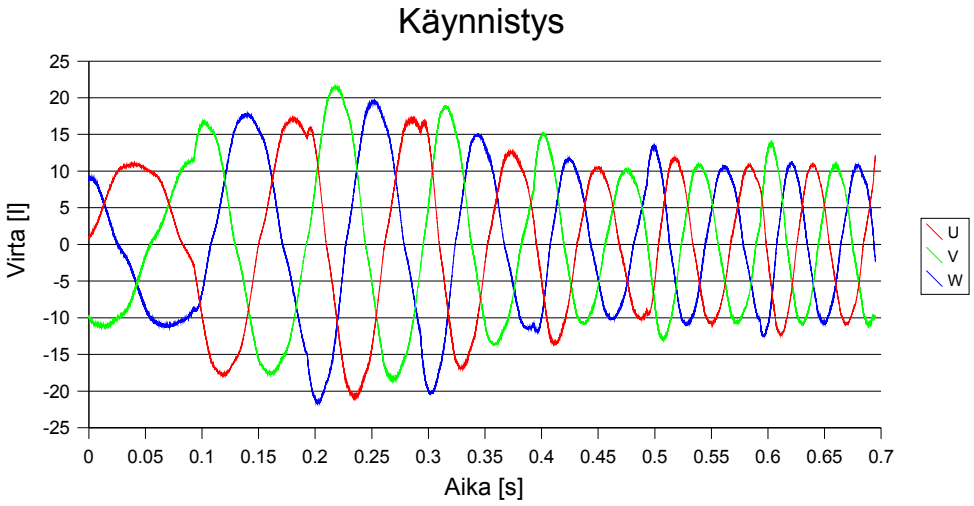
Säätötaskeja ohjataan omana taskinaan toimivan tilakoneen avulla. Tilakone saa vastaavasti tilasiirtymäohjeita omana taskinaan suoritettavalta käyttöliittymä-taskilta. Käyttöliittymä-taskilta on sarjaliikenneyhteys tietokoneelle FPGA-kortille jätettyjen

laajennus- ja debuggauspinnien kautta. Tämän sarjaliikenneyhteyden avulla terminaaliohjelman kautta voidaan antaa säätöjärjestelmälle käskyjä.

Ohjattavana moottorina oli Strömbergin valmistama oikosulkumoottori, jonka nimellisarvot ovat:

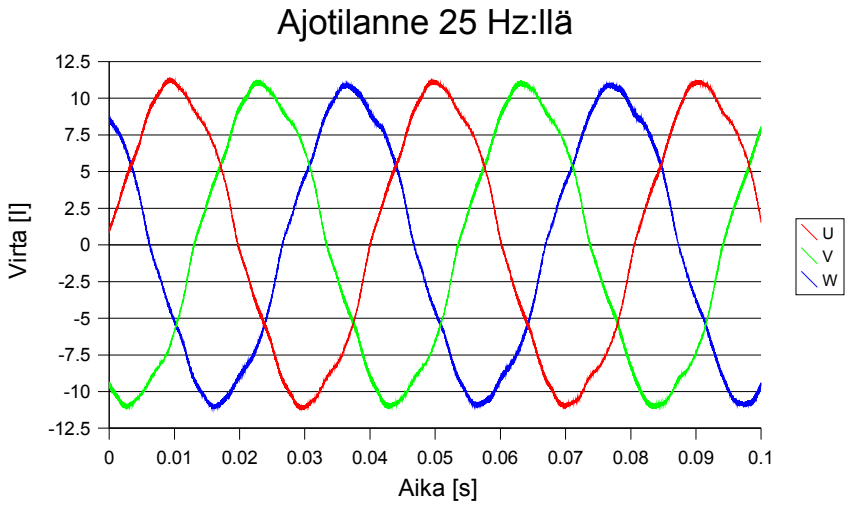
Nimellisteho: $P_N = 4$ kW
Nimellisjännite: $U_N = 380$ V
Nimellisvirta: $I_N = 8.7$ A
Nimellinen pyörimisnopeus: 1435 rpm
Nimellinen tehokerroin: 0.83 ind.

Järjestelmää testattiin ilman säätöosaa eli oikosulkumoottoria ajettiin pelkästään skalaariohjauksella. Moottorin vaihevirrat on mitattu kuvassa 21, kun moottori käynnistetään kuormaa vastaan 25 Hz:n pyörimisnopeudelle. Kuvasta nähdään käynnistyksessä tarvittava virtojen suureneminen ja käynnistyksen jälkeen virtojen tasoittuminen.



Kuva 21. Protolaitteistolla ajetun 4 kW:n sähkömoottorin vaihevirrat käynnistysvaiheessa, kun käynnistetään pyörimään 25 Hz:n pyörimisnopeudelle ja kuormaa vastaan.

Kuvassa 22 moottoria ajetaan kuorman kanssa 25 Hz:n pyörimisnopeudella.



Kuva 22. Protolaitteistolla ajetun 4 kW:n sähkömoottorin vaihevirrat, kun ajetaan 25 Hz:n pyörimisnopeudella.

Skalaariohjauksella moottori pyöri odotetusti ja prototyyppi todettiin vaatimusten mukaisesti toimivaksi sekä käyttökelpoiseksi alustaksi monimutkaisemmallekin säätöjärjestelmälle.

4 YHTEENVETO JA JOHTOPÄÄTÖKSET

Tehoelektroniikka ja sähkökäytöt ovat vaatineet aina hyvin nopeita ohjausjärjestelmiä. Riittävä nopeus on saavutettu vain käyttäen mikroprosessoreita, DSP ja ASIC-piirejä yhdessä. Näiden yhteenliittäminen on vaatinut piirikortilta paljon tilaa sekä piirien väliset tiedonsiirtoväylät ovat vaatineet paljon vetoja piirikortille. Nämä piirilevyvedot toimivat antennin tavoin eli ne keräävät herkästi ympäristöstä sähkömagneettisia häiriöitä, jotka häiritsevät tai pahimmillaan jopa sekoittavat koko järjestelmän toiminnan. Piirikortin valmistuksessa useiden eri piirien käyttäminen on myös hankinnaltaan ja logistiikaltaan hankalampaa.

Ohjelmoitavien logiikkapiirien (FPGA) hinta, koko ja teho ovat saavuttaneet tason, jolla niitä voidaan käyttää monissa kustannuksiltaan kriittisissä, mutta samalla nopeaa laskentaa vaativissa tuotteissa. FPGA-piirien etuna on niiden laitteistopohjainen toteutus eli toiminnot voidaan toteuttaa rinnakkain hyvin nopeasti. Kaikki toiminnot voidaan toteuttaa käyttäen omia toiminnallisia lohkoja, joita voidaan myös helposti jälkikäteen lisätä ja muutella täysin vapaasti. FPGA-piirit sopivat uudelleenohjelmoitavan luonteensa vuoksi myös käytettäväksi järjestelmien suunnittelussa, joissa lopputavoitteena on kuitenkin ASIC-toteutus.

Ohjelmoitavien logiikkapiirien toteutustavasta johtuen rinnakkaiset toteutukset voivat kuluttaa piiriltä hyvin paljon resursseja ja samalla suurin osa resursseista on kuitenkin toiminnassa vain hyvin vähän aikaa. Tällaisten toimintojen toteuttaminen onkin kannattavampaa toteuttaa käyttäen sarjamuotoista rakennetta eli käytännössä mikroprosessorirakenteella. FPGA-piirille voidaankin liittää prosessori joko kiinteästi tai käyttäen piirin logiikkaa. Jälkimmäisellä tavalla on etuna, että prosessorijärjestelmää voidaan muokata ja muunnella mielin määrin. Työssä tutustuttiin lähemmin Xilinx:n MicroBlaze-nimiseen ohjelmistoprosessoriin ja selvitettiin kuinka sen käyttämien lisälaitteiden resurssien tarvetta voidaan arvioida ennen toteutusta.

Sähkökäyttöjen ohjausjärjestelmät suorittavat suuren määrän laskenta-algoritmeja, joten niiden suoritusnopeudet ovat avainasemassa. Laskenta-algoritmit tarvitsevat yleensä desimaalilukuja. Suurin osa nopeista prosessoreista on kuitenkin kokonaislukutyyppisiä,

joten täytyy tietää millaisia lukuja käyttäen algoritmit on kannattavin toteuttaa. Lähemmin tarkasteltiin MicroBlaze-ohjelmistoprosessoria ja sillä liukuluvuilla laskeminen oli liian hidasta, joten parhaaksi tavaksi osoittautui perinteisten kiinteänpilkunlukujen käyttö.

Sulautetuilta prosessorijärjestelmiltä vaaditaan tarkkaa ennustettavuutta ja samalla useiden eri toimintojen yhtäaikaista suoritusta. Näiden vaatimusten täyttämiseksi on kehitetty lukuisia reaaliaikakäyttöjärjestelmiä, jotka helpottavat kyseisten vaatimusten mukaisten järjestelmien toteuttamista. Käyttöjärjestelmän palveluiden käyttäminen vaatii kuitenkin aina oman suoritusaikansa, joten suoritettiin suorituskykymittauksia MicroBlaze-prosessorilla suoritettavan µC/OS-II-tosiaikakäyttöjärjestelmän kellokeskeytysten, keskeytysten kieltämisen ja sallimisen sekä käyttöjärjestelmän palveluiden osalta. Näiden tulosten avulla voitiin arvioida, mitä palveluita kannattaa protolaitteistolla käyttää, kuinka pieni kannattaa kellokeskeytysten suoritusväli olla ja kuinka paljon suoritusaikaa ohjausalgoritmeille voidaan enintään käyttää.

Lopuksi prototyypin tarkoituksena oli testata, kuinka käytännössä voidaan toteuttaa koko taajuudenmuuttajan ohjausjärjestelmä yhdellä piirillä. Prototyypin alustana käytettiin jo aiemmassa hankkeessa kehitettyä laitteistoa, joten tämän työn tarkoituksena oli keskittyminen FPGA-piirin sisäisen rakenteen kehittelyyn ja suunnitteluun. FPGA-piirille sulautetulla MicroBlaze-prosessorilla suoritettiin moottorin ohjauksen säätöalgoritmeja 100 µs, 1 ms ja 100 ms väliajoin suoritettavissa taskeissa. Käyttöliittymänä käytettiin sarjaliikenneyhteyden kautta tietokoneelle toimivaa yksinkertaista komentotulkkirakennetta. Tällä järjestelmällä ajettiin 4 kW:n oikosulkumoottoria onnistuneesti ja mitattiin moottorin vaihevirroista kuvat eri ajotilanteista.

Työn aikana heränneitä jatkotutkimusideoita, joita ei tähän työhön voinut liittää, ovat esimerkiksi prosessorin välimuistin käytön analysointi ulkoista muistia käytettäessä, ohjelman jako sisäisen ja ulkoisen muistin välillä, jo valmiin muulle laitealustalle tehdyn C-kielisen ohjelmiston siirtäminen FPGA/MicroBlaze-alustalle, laitteistotason ohjelmistolaajennusten lisääminen ja säädön osien suoritusjako prosessorin sekä piiriosan kanssa. Monipuolisemman säätöalgoritmin testaaminen olisi ollut myös

mahdollista pienellä jatkopanoksella. Lisäksi muiden säätömenetelmien jakamisen tutkiminen prosessorilla ja piiriosalla suoritettaviin osiin sekä niiden testaaminen vaatisi kehittelemistä.

Mielenkiintoinen tutustumiskohde olisi myös tässä työssä vastaavanlaisen järjestelmän toteuttaminen käyttäen Flash-pohjaisia FPGA-piirejä, jolloin vältyttäisiin ainakin erillisen konfigurointimuistin käytöltä.

LÄHDELUETTELO

[1] MicroC/OS-II The Real-Time Kernel, Labrosse, Jean J., 2002, 606s. ISBN: 1-57820-103-9

[2] Jouko Niiranen. 1999. Sähkömoottorikäytön digitaalinen ohjaus. Otatieto, Helsinki. ISBN 951-672-270-9

[3] Pyrhönen, Juha. Sähkökäytöt 2003, luentomateriaali. Lappeenrannan teknillinen yliopisto.

[4] ABB: Teknisiä tietoja ja taulukoita -käsikirja, Luku 18. Sähkömoottorikäytöt. Versiopäiväys 07-2000. http://www.abb.com/global/fiabb/fiabb255.nsf/viewunid/C46D5509D325D21AC225695B002FB07B/$file/180_0007.pdf (viitattu 5.1.2004)

[5] Rauma K., Laakkonen O., Härkönen T., New bus structure for programmable logic devices controlling power electronics. Hyväksytty konferenssiin 6th IEEE Power Electronics Specialists Conference, PESC 2005, 12.-16.6.2005.

[6] Laakkonen O., Penttinen A., Luukko J., Pyrhönen O., Single chip implementation of the frequency converter controller using FPGA circuit and soft processor, hyväksytty konferenssiin 11th European Conference on Power Electronics and Applications, EPE 2005, 13.-15.9.2005.

[7] Laakkonen O., Rauma K., Sarén H., Luukko J., Pyrhönen O., Electric drive emulator using dSPACE real time platform for VHDL verification, 47th Midwest Symposium on Circuits and Systems (MWSCAS), volume 3, sivut 279-282, Hiroshima, Japani, 2004.

[8] Penttinen, Aki. FPGA:lla olevien IP-lohkojen debuggaustyökalu toteutettuna FPGA:lle sulautetulla mikroprosessorilla. Erikoistyö. LTY/Sähkötekniikan osasto, 2004.

[9] Virtex-II Platform FPGAs: Complete Data Sheet.
http://direct.xilinx.com/bvdocs/publications/ds031.pdf (27.9.2004)

[10] XC18V00 Series In-System Programmable Configuration PROMs Data Sheet, DS026 (v5.0.1) heinäkuu 20, 2004.
http://direct.xilinx.com/bvdocs/publications/ds026.pdf (viitattu 4.11.2004)

[11] Xilinx's Embedded System Tools Guide, Embedded Development Kit, EDK 6.3i elokuu 20, 2004.
ps_ug.pdf

[12] Platform Flash In-System Programmable Configuration PROMs, DS123 (v2.5) lokakuu 18,2004.
http://direct.xilinx.com/bvdocs/publications/ds123.pdf (viitattu 16.12.2004)

[13] Virtex-II™ V2MB1000 Development Board User's Guide. Version 3.0, joulukuu 2002.
V2MB_User_Guide_3_0.pdf

[14] White Paper: ”Systen Ace: Configuration Solution for Xilinx FPGAs”. v1.0, syyskuu 25, 2001.
http://www.xilinx.com/bvdocs/whitepapers/wp151.pdf (viitattu 17.1.2005)

[15] System ACE™ Configuration Solutions FAQ, helmikuu 6, 2003.
http://www.xilinx.com/isp/systemace/faq001_system-ace_1_1.pdf
(viitattu 17.1.2005)

[16] Free open source IP cores and chip design
http://www.opencores.org (viitattu 22.12.2004)

[17] Xilinx Processor Central: MicroBlaze™ Soft Processor Core.
http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sSecondaryNavPick=Design+Tools&key=micro_blaze (viitattu 21.9.2004)

[18] PicoBlaze™ 8-bit Microcontroller Reference Design for FPGAs and CPLDs. http://www.xilinx.com/bvdocs/ipcenter/data_sheet/picoblaze_productbrief.pdf (viitattu 20.12.2004)

[19] PicoBlaze Emulated 8051 Microcontroller (PB8051-MX/TF), syyskuu 10, 2003. http://www.xilinx.com/products/logicore/alliance/roman_jones/ romanjones_pb8051_microcontroller.pdf (viitattu 20.12.2004)

[20] IP Core Overview: OpenRISC 1200 RISC/DSP Core. http://www.opencores.org/projects.cgi/web/or1k/or1200/or1200_overview.pdf (viitattu 10.1.2005)

[21] OpenRISC 1000: OpenRISC 1200, Preminilary Results. http://www.opencores.org/projects.cgi/web/or1k/openrisc_1200 (viitattu 10.1.2005)

[22] Nios II Processor Cores. http://www.altera.com/products/ip/processors/nios2/cores/ni2- processor_cores.html (viitattu 20.12.2004)

[23] Patterson D.A, Hennessy J.L. 1990, 1996. Computer Architecture A Quantative Approach. Morgan Kaufmann, San Francisco. Second Edition. ISBN 1-55860-329-8

[24] MicroBlaze Processor Reference Guide, Embedded Development Kit, EDK 6.3i elokuu 24, 2004. mb_ref_guide.pdf

[25] OPB IPIF Architecture, Product Specification v1.21, EDK 6.3i, heinäkuu 30, 2002. opb_ipif.pdf

[26] Xilinx MicroBlaze Soft Processor FAQ.
http://www.xilinx.com/xlnx/xebiz/designResources/contentContainer.jsp?sGlobalNavPick=&sSecondaryNavPick=Design+Tools&iLanguageID=1&key=mb_faq (viitattu 27.9.2004)

[27] Xilinx's Embedded System Tools Reference Manual, Embedded Development Kit, EDK 6.3i, elokuu 20, 2004.
est_rm.pdf

[28] BFM Simulation in Platform Studio, Embedded Development Kit, EDK 6.3i syyskuu 10, 2004.
bfm_simulation.pdf

[29] AN-1013: µC/OS-II for the Xilinx MicroBlaze Soft-Core processor.
www.ucos-ii.com/microblaze/MicroBlazeContent.html (viitattu 18.10.2004)

[30] Luukko, Julius. Periodic task execution in uC/OS-II (2003, julkaisematon).
periodic.pdf

[31] IEEE Standard 754. IEEE Standard for Binary Floating-Point Arithmetic. 1985.
Saatavissa: http://ieeexplore.org (viitattu 16.2.2004)

[32] Kraeling, M.B. Fixed-Point math in time-critical C applications. Wescon/96, 22-24 lokakuuta 1996. ISBN 0-7803-3274-1
Saatavissa: http:ieeexplore.org (viitattu 16.2.2004)

[33] Xilinx Processor IP Sizing Calculator.
http://www.xilinx.com/ipcenter/processor_central/ppcip/calc.htm
(viitattu 17.1.2005)

[34] Petters Stefan, Färber G. Making worst-case execution time analysis for hard real-time tasks on state of the art processors feasible. 1999. Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99). IEEE Computing Society Press.

[35] Sebek, Filip. Cache memories and real-time systems. MRTC Technical Report 01/37. Department of Computer Engineering, Mälardalen University, Västerös, Sweden. Revision 2nd October 2001. http://www.idt.mdh.se/fsk/docs/sota.pdf (viitattu 27.4.2004)

[36] Agilent HFBR-5911L/AL Small Form Factor Optical Transceiver for Gigabit Ethernet (1.25 GBd) and iSCSI, Data Sheet. http://cp.literature.agilent.com/litweb/pdf/5988-7817EN.pdf (viitattu 14.2.2005)

[37] 10 Megabaud Versatile Link Fiber Optic Transmitter and Receiver for 1 mm POF and 200 µm HCS, Technical Data. http://cp.literature.agilent.com/litweb/pdf/5988-5674EN.pdf (viitattu 14.2.2005)

[38] Väliviita, S., Tiitinen P., Ovaska S.J., Improving the reusability of frequency converter software by using the structured analysis method., ISIE'97, Guimarães, Portugal, 1997.

[39] IEEE Standard 1149.1. IEEE Standard test access port and boundary – scan architecture. 1990. Saatavissa: http://ieeexplore.org (viitattu 20.4.2005)

[40] IEEE Standard 1532. IEEE Standard for in-system configuration of programmable devices. 2001. Saatavissa: http://ieeexplore.org (viitattu 20.4.2005)

[41] GNU C Library http://www.gnu.org/software/libc/libc.html (viitattu 21.4.2005)

[42] RTCA Standard DO-178B. Software Considerations in Airborne System and Equipment Certification. 1992.

[43] J.G. Proakis and D.G. Manolakis, Digital Signal Processing – Principles, Algorithms, and Applications. Prentice-Hall, Inc., 3 ed., 1996.
ISBN: 0-13-373762-4

[44] Déziel, J. Philippe. Applied digital signal processing. Prentice-Hall, Inc., 1st ed., 2001. ISBN: 0-13-775768-9

[45] Carbone, John. Optimize MicroBlaze processors for consumer electronics products. Xcell Journal, Issue 48, 2004.
http://www.xilinx.com/publications/xcellonline/xcell_48/xc_pdf/xc_expresslogic48.pdf (viitattu 28.4.2005)

LIITE 1, sivu 1. MicroBlaze-prosessorin käskykanta

Symbol	Description
Ra	R0 - R31, General Purpose Register, source operand a
Rb	R0 - R31, General Purpose Register, source operand b
Rd	R0 - R31, General Purpose Register, destination operand
Imm	16 bit immediate value
Imm*x*	*x* bit immediate value
FSL*x*	3 bit Fast Simplex Link (FSL) port designator where *x* is the port number
C	Carry flag, MSR[29]
Sa	Special Purpose Register, source operand
Sd	Special Purpose Register, destination operand
s(*x*)	Sign extend argument *x* to 32-bit value
*Addr	Memory contents at location Addr (data-size aligned)
&	Concatenate. E.g. “0000100 & Imm7” is the concatenation of the fixed field “0000100” and a 7 bit immediate value.

TYPE A	0-5	6-10	11-15	16-20	21-31	
TYPE B	0-5	6-10	11-15		16-31	Semantics
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + 1
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + 1
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000001	Rd := Rb + Ra + 1(signed)
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000011	Rd := Rb + Ra + 1(unsigned)
ADDI Rd,Ra,Imm	001000	Rd	Ra		Imm	Rd := s(Imm) + Ra
RSUBI Rd,Ra,Imm	001001	Rd	Ra		Imm	Rd := s(Imm) + Ra + 1
ADDIC Rd,Ra,Imm	001010	Rd	Ra		Imm	Rd := s(Imm) + Ra + C
RSUBIC Rd,Ra,Imm	001011	Rd	Ra		Imm	Rd := s(Imm) + Ra + C
ADDIK Rd,Ra,Imm	001100	Rd	Ra		Imm	Rd := s(Imm) + Ra
RSUBIK Rd,Ra,Imm	001101	Rd	Ra		Imm	Rd := s(Imm) + Ra + 1
ADDIKC Rd,Ra,Imm	001110	Rd	Ra		Imm	Rd := s(Imm) + Ra + C
RSUBIKC Rd,Ra,Imm	001111	Rd	Ra		Imm	Rd := s(Imm) + Ra + C
MUL Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000000	Rd := Ra * Rb
BSRL Rd,Ra,Rb	010001	Rd	Ra	Rb	00000000000	Rd : = Ra >> Rb
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	01000000000	Rd := Ra[0], (Ra >> Rb)
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	10000000000	Rd := Ra << Rb
MULI Rd,Ra,Imm	011000	Rd	Ra		Imm	Rd := Ra * s(Imm)
BSRLI Rd,Ra,Imm	011001	Rd	Ra	00000000000 & Imm5		Rd : = Ra >> Imm5
BSRAI Rd,Ra,Imm	011001	Rd	Ra	00000010000 & Imm5		Rd := Ra[0], (Ra >> Imm5)
BSLLI Rd,Ra,Imm	011001	Rd	Ra	00000100000 & Imm5		Rd := Ra << Imm5
IDIV Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000000	Rd := Rb/Ra, signed
IDIVU Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000001	Rd := Rb/Ra, unsigned
GET Rd,FSLx	011011	Rd	00000	0000000000000 & FSLx		Rd := FSLx (blocking data read) MSR[FSL] := FSLx_S_Control
PUT Ra,FSLx	011011	00000	Ra	1000000000000 & FSLx		FSLx := Ra (blocking data write)
NGET Rd,FSLx	011011	Rd	00000	0100000000000 & FSLx		Rd := FSLx (non-blocking data read), MSR[FSL] := FSL*x*_S_Control, MSR[C] := not FSL*x*_S_Exists
NPUT Ra,FSLx	011011	00000	Ra	1100000000000 & FSLx		FSLx := Ra (non-blocking data write), MSR[C] := FSLx_M_Full

LIITE 1, sivu 2. MicroBlaze-prosessorin käskykanta

TYPE A	0-5	6-10	11-15	16-20	21-31	
TYPE B	0-5	6-10	11-15	16-31		Semantics
CGET Rd,FSLx	011011	Rd	00000	0010000000000 & FSLx		Rd := FSLx (blocking control read), MSR[FSL] := not FSLx_S_Control
CPUT Ra,FSLx	011011	00000	Ra	1010000000000 & FSLx		FSLx := Ra (blocking control write)
NCGET Rd,FSLx	011011	Rd	00000	0110000000000 & FSLx		Rd := FSLx (non-blocking control read), MSR[FSL] := not FSLx_S_Control, MSR[C] := not FSLx_S_Exists
NCPUT Ra,FSLx	011011	00000	Ra	1110000000000 & FSLx		FSLx := Ra (non-blocking control write), MSR[C] := FSLx_M_Full
OR Rd,Ra,Rb	100000	Rd	Ra	Rb	00000000000	Rd := Ra or Rb
AND Rd,Ra,Rb	100001	Rd	Ra	Rb	00000000000	Rd := Ra and Rb
XOR Rd,Ra,Rb	100010	Rd	Ra	Rb	00000000000	Rd := Ra xor Rb
ANDN Rd,Ra,Rb	100011	Rd	Ra	Rb	00000000000	Rd := Ra and Rb
SRA Rd,Ra	100100	Rd	Ra	0000000000000001		Rd := Ra[0], (Ra >> 1); C := Ra[31]
SRC Rd,Ra	100100	Rd	Ra	0000000000100001		Rd := C, (Ra >> 1); C := Ra[31]
SRL Rd,Ra	100100	Rd	Ra	0000000001000001		Rd := 0, (Ra >> 1); C := Ra[31]
SEXT8 Rd,Ra	100100	Rd	Ra	0000000001100000		Rd[0:23] := Ra[24]; Rd[24:31] := Ra[24:31]
SEXT16 Rd,Ra	100100	Rd	Ra	0000000001100001		Rd[0:15] := Ra[16]; Rd[16:31] := Ra[16:31]
WIC Ra,Rb	100100	Ra	Ra	Rb	01101000	ICache_Tag := Ra, ICache_Data := Rb
WDC Ra,Rb	100100	Ra	Ra	Rb	01100100	DCache_Tag := Ra, DCache_Data := Rb
MTS Sd,Ra	100101	00000	Ra	1100000000000 & Sd		Sd := Ra , where Sd=001 is MSR
MFS Rd,Sa	100101	Rd	00000	1000000000000 & Sa		Rd := Sa , where Sa=000 is PC, 001 is MSR, 011 is EAR, and 101 is ESR
MSRCLR Rd,Imm	100101	Rd	00001	00 & Imm14		Rd := MSR; MSR := MSR ^ Imm14
MSRSET Rd,Imm	100101	Rd	00000	00 & Imm15		Rd := MSR; MSR := MSR ^ Imm14
BR Rb	100110	00000	00000	Rb	00000000000	PC := PC + Rb
BRD Rb	100110	00000	10000	Rb	00000000000	PC := PC + Rb
BRLD Rd,Rb	100110	Rd	10100	Rb	00000000000	PC := PC + Rb; Rd := PC
BRA Rb	100110	00000	01000	Rb	00000000000	PC := Rb
BRAD Rb	100110	00000	11000	Rb	00000000000	PC := Rb
BRALD Rd,Rb	100110	Rd	11100	Rb	00000000000	PC := Rb; Rd := PC
BRK Rd,Rb	100110	Rd	01100	Rb	00000000000	PC := Rb; Rd := PC; MSR[BIP] := 1
BEQ Ra,Rb	100111	00000	Ra	Rb	00000000000	if Ra = 0: PC := PC + Rb
BNE Ra,Rb	100111	00001	Ra	Rb	00000000000	if Ra /= 0: PC := PC + Rb
BLT Ra,Rb	100111	00010	Ra	Rb	00000000000	if Ra < 0: PC := PC + Rb
BLE Ra,Rb	100111	00011	Ra	Rb	00000000000	if Ra <= 0: PC := PC + Rb
BGT Ra,Rb	100111	00100	Ra	Rb	00000000000	if Ra > 0: PC := PC + Rb
BGE Ra,Rb	100111	00101	Ra	Rb	00000000000	if Ra >= 0: PC := PC + Rb
BEQD Ra,Rb	100111	10000	Ra	Rb	00000000000	if Ra = 0: PC := PC + Rb
BNED Ra,Rb	100111	10001	Ra	Rb	00000000000	if Ra /= 0: PC := PC + Rb
BLTD Ra,Rb	100111	10010	Ra	Rb	00000000000	if Ra < 0: PC := PC + Rb
BLED Ra,Rb	100111	10011	Ra	Rb	00000000000	if Ra <= 0: PC := PC + Rb
BGTD Ra,Rb	100111	10100	Ra	Rb	00000000000	if Ra > 0: PC := PC + Rb
BGED Ra,Rb	100111	10101	Ra	Rb	00000000000	if Ra >= 0: PC := PC + Rb
ORI Rd,Ra,Imm	101000	Rd	Ra		Imm	Rd := Ra or s(Imm)
ANDI Rd,Ra,Imm	101001	Rd	Ra		Imm	Rd := Ra and s(Imm)
XORI Rd,Ra,Imm	101010	Rd	Ra		Imm	Rd := Ra xor s(Imm)
ANDNI Rd,Ra,Imm	101011	Rd	Ra		Imm	Rd := Ra and s(Imm)
IMM Imm	101100	00000	00000		Imm	Imm[0:15] := Imm
RTSD Ra,Imm	101101	10000	Ra		Imm	PC := Ra + s(Imm)

LIITE 1, sivu 3. MicroBlaze-prosessorin käskykanta

TYPE A	0-5	6-10	11-15	16-20	21-31	
TYPE B	0-5	6-10	11-15	16-31		Semantics
RTID Ra,Imm	101101	10001	Ra	Imm		PC := Ra + s(Imm); MSR[IE] := 1
RTED Ra,Imm	101101	10010	Ra	Imm		PC := Ra + s(Imm); MSR[EE] := 1, MSR[EIP]:=0
RTBD Ra,Imm	101101	10010	Ra	Imm		PC := Ra + s(Imm); MSR[BIP] := 0
BRI Imm	101110	00000	00000	Imm		PC := PC + s(Imm)
BRID Imm	101110	00000	10000	Imm		PC := PC + s(Imm)
BRLID Rd,Imm	101110	Rd	10100	Imm		PC := PC + s(Imm); Rd := PC
BRAI Imm	101110	00000	01000	Imm		PC := s(Imm)
BRAID Imm	101110	00000	11000	Imm		PC := s(Imm)
BRALID Rd,Imm	101110	Rd	11100	Imm		PC := s(Imm); Rd := PC
BRKI Rd,Imm	101110	Rd	01100	Imm		PC := s(Imm); Rd := PC; MSR[BIP] := 1
BEQI Ra,Imm	101111	00000	Ra	Imm		if Ra = 0: PC := PC + s(Imm)
BNEI Ra,Imm	101111	00001	Ra	Imm		if Ra /= 0: PC := PC + s(Imm)
BLTI Ra,Imm	101111	00010	Ra	Imm		if Ra < 0: PC := PC + s(Imm)
BLEI Ra,Imm	101111	00011	Ra	Imm		if Ra <= 0: PC := PC + s(Imm)
BGTI Ra,Imm	101111	00100	Ra	Imm		if Ra > 0: PC := PC + s(Imm)
BGEI Ra,Imm	101111	00101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
BEQID Ra,Imm	101111	10000	Ra	Imm		if Ra = 0: PC := PC + s(Imm)
BNEID Ra,Imm	101111	10001	Ra	Imm		if Ra /= 0: PC := PC + s(Imm)
BLTID Ra,Imm	101111	10010	Ra	Imm		if Ra < 0: PC := PC + s(Imm)
BLEID Ra,Imm	101111	10011	Ra	Imm		if Ra <= 0: PC := PC + s(Imm)
BGTID Ra,Imm	101111	10100	Ra	Imm		if Ra > 0: PC := PC + s(Imm)
BGEID Ra,Imm	101111	10101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
LBU Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd[0:23] := 0, Rd[24:31] := *Addr
LHU Rd,Ra,Rb	110001	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd[0:15] := 0, Rd[16:31] := *Addr
LW Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd := *Addr
SB Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; *Addr := Rd[24:31]
SH Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; *Addr := Rd[16:31]
SW Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000	Addr := Ra + Rb;*Addr := Rd
LBUI Rd,Ra,Imm	111000	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd[0:23] := 0, Rd[24:31] :=*Addr
LHUI Rd,Ra,Imm	111001	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd[0:15] := 0, Rd[16:31] :=*Addr
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd := *Addr
SBI Rd,Ra,Imm	111100	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd[24:31]
SHI Rd,Ra,Imm	111101	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd[16:31]
SWI Rd,Ra,Imm	111110	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd[16:31]

LIITE 2 Suorituskykymittauksiin käytetyt ohjelmakoodit

```
#include <xtmrctr.h>


volatile INT32U start_time, end_time, sub_time;

/* Suorituskykymittauksen kalibrointi */
void SpeedMeasCalibration( void )
{
sub_time = 0;
/* Alustetaan laskurirekisteri */
XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_2_BASEADDR, 0, 0);
SpeedMeasStart();
sub_time = SpeedMeasStop();
}

/* Mittauksen aloitus */
void SpeedMeasStart( void )
{
XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_2_BASEADDR, 0,
XTC_CSR_LOAD_MASK);
XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_2_BASEADDR, 0,
XTC_CSR_ENABLE_TMR_MASK );
start_time = XTmrCtr_mGetTimerCounterReg
(XPAR_OPB_TIMER_2_BASEADDR, 0);
}

/* Mittauksen lopetus */
INT32U SpeedMeasStop( void )
{
end_time = XTmrCtr_mGetTimerCounterReg
(XPAR_OPB_TIMER_2_BASEADDR, 0);
XTmrCtr_mDisable(XPAR_OPB_TIMER_2_BASEADDR, 0);
return( end_time-start_time-sub_time );
}
```