Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Tietoverkkolaboratorio

Oskari Simola

Yleiskäyttöisen tietokoneen kellosynkronisointi ja käyttö verkkoliikenteen mittauksiin

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 15.6.2007

Työn valvoja: Professori Raimo Kantola

Työn ohjaaja: TkL Markus Peuhkuri

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Oskari Simola
Työn nimi:	Yleiskäyttöisen tietokoneen kellosynkronisointi ja käyttö verkkoliikenteen mittauksiin
Päivämäärä:	15.6.2007 **Sivuja:** 77
Osasto:	Sähkö- ja tietoliikennetekniikan osasto
Professuuri:	S-38 Tietoverkkotekniikka
Työn valvoja:	Professori Raimo Kantola
Työn ohjaaja:	TkL Markus Peuhkuri

Liikennemittauksia tehtäessä monesta paikasta yhtä aikaa on mittalaitteiden oltava kellosynkronisoituja. Kaupallisissa mittalaitteissa ja erikoisvalmisteisissa kaappauskorteissa tämä onnistuu GPS-järjestelmän avulla.

Diplomityön tavoitteena on kehittää halvempaa ratkaisua, jossa mittauksiin käytetään normaalia tietokonetta ja normaaleja verkkokortteja. Kellosynkronisointi suoritetaan SynPCI-X –kortin avulla, joka saa signaalinsa GPS-laitteelta. Lisäksi tavoitteena on kehittää mittausmenetelmiä, joilla saadaan mitattua pakettien aikaleimojen virhettä.

Mittauksissa määritettiin passiivisesti kaapattujen pakettien aikaleimojen virheen jakauma ja maksimivirhe. Kehitetty mittausmenetelmä mahdollistaa yhden tietokoneen pakettien aikaleimojen virheen mittaamisen muista tietokoneista riippumatta.

SynPCI-X –kortin avulla sykronoidussa monisuoritinkoneessa maksimivirhe on alle $30\,\mu$s. Yhden suorittimen tietokoneissakin reaaliaikakäyttöjärjestelmän avulla maksimivirhe on alle $60\,\mu$s. Maksimivirhe on 99 prosentilla paketeista monisuoritinjärjestelmissä alle $12\,\mu$s ja yhden suorittimen järjestelmissäkin alle $25\,\mu$s.

Kehitettyä synkronointijärjestelmää voi myös hyödyntää muissakin kuin passiviisissa liikennemittauksissa. Järjestelmää on mahdollista käyttää aktiivisissa monipistemittauksissa tai vaikka WLAN-verkkojen mittauksissa.

Avainsanat: SynPCI-X, kellosynkronointi, pakettikaappaus, tietoliikennemittaukset

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Oskari Simola
Name of the thesis:	Clock synchronization of personal computer and use for network traffic measurements
Date:	June 15, 2007 **Number of pages:** 77
Department:	Department of Electrical and Communications Engineering
Professorship:	S-38 Networking Technology
Supervisor:	Professor Raimo Kantola
Instructor:	Tech. Lic. Markus Peuhkuri

When network measurements are performed simultaneously in many places, measurement devices have to be clock synchronized. In commercial measurement devices and special capture cards this can be done, e.g., with a GPS system.

A goal of this Master's Thesis is to develop a less costly solution where network measurements are done on everyday personal computers and normal network cards. Clock synchronization is carried out with the recently developed SynPCI-X card which receives synchronization signals from a GPS device. A second goal is to design methods for measuring time stamp errors of packets.

The distribution and the maximum of the time stamp error of packets were measured. The methods developed made it possible to characterize the error of the time stamps independently of other computers.

A multiprocessor computer synchronized with the SynPCI-X card has a maximum error smaller than 30μs. Even in single processor systems the maximum error is less than 60μs on an operation system patched to function in real time. If only 99 per cent of the packets are counted in a multiprocessor system, the maximum error remains below 12μs and in a single processor system, it is less than 25μs.

This novel SynPCI-X synchronization system has uses in addition to the passive packet capture presented in this Thesis. Possible applications include active multi-point or even WLAN network measurements.

Keywords: SynPCI-X, clock synchronization, packet capture

Alkulause

Tämä diplomityö on tehty Tietoverkkolaboratoriossa osana LATE-projektia.

Haluan kiittää työn valvojaa Raimo Kantolaa ja työn ohjaajaa Markus Peuhkuria lukuisista hedelmällisistä keskusteluista ja työn ohjauksesta. Kiitokset kuuluvat myös Mika Ilvesmäelle mittauksiin liittyvistä ideoista ja työn oikolukemisesta.

Haluan myös kiittää Avaruustekniikan laboratoriota PADS-lisenssin ja Signaalinkäsittelytekniikan laboratoriota logiikka-analysaattorin lainaamisesta.

Haluan kiittää myös työtovereitani ja laboratorion sählyporukkaa. Kiitokset kuuluvat myös vanhemmilleni ja opiskelutovereilleni.

Otaniemessä 15. kesäkuuta 2007

Oskari Simola

Sisältö

Alkulause **iii**

Lyhenneluettelo **x**

1 Johdanto **1**

1.1 Työn tausta 1

1.2 Työn tavoite 2

1.3 Työn rakenne 2

2 Tietokoneen arkkitehtuuri **4**

2.1 Väylät 4

2.1.1 PCI 4

2.1.2 PCI-X 5

2.1.3 PCI Express 5

2.1.4 Hyper Transport 6

2.2 Keskeytykset 7

2.3 Laskurit 8

2.4 Verkkokorttien arkkitehtuuri 9

2.4.1 Liitäntäväylät 9

2.4.2 Fyysinen kerros 9

2.4.3 Siirtoyhteyskerros 10

2.4.4 Intel PRO/1000 10

2.4.5 Broadcom NetXtreme 13

2.5 Pakettikaappauksen pullonkaulat 13

3 Reaaliaikainen tiedonkäsittely 15

3.1 Reaaliaikajärjestelmät 15

3.1.1 Määritelmä 15

3.1.2 Pehmeät ja kovat aikarajat 15

3.1.3 Deterministisyys 16

3.2 Reaaliaikakäyttöjärjestelmät 16

3.3 Keskeytyksien hallinta 16

3.4 Verkkoliikenteen käsittely 17

3.4.1 Linux 2.6 17

3.4.2 FreeBSD 6.1 18

3.5 Ajan ylläpito 18

3.5.1 Linux 2.6.18 19

3.5.2 FreeBSD 6.1 19

4 SynPCI-X –kortti 21

4.1 Tietokoneen ajan ylläpito 21

4.2 Toimintaperiaate 23

4.3 SynPCI-prototyyppi 23

4.4 Kortin uusi versio 24

4.4.1 Parannustavoitteet 24

4.4.2 BGA-kanta 25

4.4.3 Viiden voltin PCI-väylä 25

4.4.4 VHDL 26

4.5 Havaitut ongelmat 28

4.5.1 133 MHz PCI-X –väylä 28

4.5.2 Vanhat PCI-väylät 28

4.5.3 Kahden kellosignaalin käyttö 28

4.6 Linux-laiteajuri 29

5 Mittaukset **31**

5.1 Mittauslaitteistot 31

5.1.1 Supermicro Dual Opteron 31

5.1.2 Supermicro Celeron P8SC8 32

5.1.3 Mittausjärjestelyt 33

5.2 Laskureiden lukuviive 35

5.3 Kulkuaikaviive 36

5.3.1 Opteron SynPCI-X 37

5.3.2 Celeron SynPCI-X 39

5.3.3 Celeron NTP 41

5.3.4 Linux ja FreeBSD 42

5.3.5 Intelin verkkokortti 44

5.4 Lämpötilan vaikutus NTP-synkronointiin 46

5.5 Virhelähteet 47

6 Yhteenveto **49**

6.1 Johtopäätökset 49

6.2 Jatkokehityskohteet 50

Lähdeluettelo **51**

A SynPCI-X –kortin Linux-ajuri **54**

B SynPCI-X –kortin VHDL-lähdekoodi **63**

B.1 synpcix.vhd 63

B.2 clkdiv2.vhd 70

B.3 nsec_counter.vhd 70

B.4 led_control.vhd 71

C SynPCI-X –kortin valokuva **74**

D SynPCI-X –kortin piirilevykuvat **75**

Kuvat

2.1 Intelin verkkokortin lohkokaavio 11

4.1 Synkronointikortin periaatepiirros 23
4.2 Väyläkytkimen periaate 25
4.3 Jännitetasomuunnin 26
4.4 PCI/PCI-X –tilakone 27

5.1 Opteron-emolevyn periaatepiirros 32
5.2 Celeron emolevyn periaatepiirros 33
5.3 Mittausjärjestelyt 34
5.4 Opteron-kone SynPCI-X synkronoituna 37
5.5 Sarjaportin ja verkkokortin viivejakauman vertailu 39
5.6 Kulkuaikaviivejakauma Celeron-koneella 41
5.7 Kulkuaikaviivejakauma normaaleilla ytimillä 44
5.8 Kulkuaikaviivejakauma Intelin verkkokortilla 45
5.9 Lämpötilan vaikutus aikapoikkeamaan 47

C.1 Valokuva SynPCI-X –kortista 74

D.1 Pintakerroksen silkkipainokuva 75
D.2 Pintakerroksen juotosmaski 75
D.3 Pintakerros 76
D.4 Invertoitu maakerros 76
D.5 Käyttöjännitekerros 76

D.6 Pohjakerros 77
D.7 Pohjakerroksen juotosmaski 77

Taulukot

5.1 Laskureiden lukuviive 35
5.2 SynPCI-X –kortin lukuviive 36
5.3 Opteron SynPCI-X kulkuaikaviive (μs) 38
5.4 Celeron SynPCI-X kulkuaikaviive (μs) 40
5.5 Celeron-koneen NTP-kulkuaikaviive (μs) 42
5.6 Celeron FreeBSD 6.1 ja Linux 2.6.18 kulkuaikaviive (μs) 43
5.7 Opteron SynPCI-X kulkuaikaviive Intelin verkkokortilla (μs) 46

Lyhenneluettelo

ACPI	Advanced Configuration and Power Interface
ADC	Analog to Digital Converter, Analogidigitaalimuunnin
AMD	Advanced Micro Devices
APIC	Advanced Programmable Interrupt Controller
BGA	Ball Grid Array
CPLD	Complex Programmable Logic Device, Kompleksinen ohjelmoitava logiikkapiiri
DAC	Digital to Analog Converter, Digitaalianalogimuunnin
DDR	Double Data Rate
DMA	Direct Memory Access
DMI	Direct Media Interface
ECC	Error Correction Code, Virheenkorjauskoodi
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSB	Front Side Bus
Gbit/s	Gigabits per second, Gigabittiä sekunnissa
GMII	Gigabit Media Independent Interface
GPS	Global Positioning System, Maailmanlaajuinen satelliittinavigointijärjestelmä
HPET	High Precision Event Timers
JTAG	Joint Test Action Group
LAPIC	Local Advanced Programmable Interrupt Controller
LVDS	Low Voltage Differential Signaling, Matalajännite differentiaali signalointi
MAC	Media Access Control
MB/s	Megabytes per second, Megatavua sekunnissa
Mbit/s	Megabits per second, Megabittiä sekunnissa
MHz	Megahertz, Megahertsi
MII	Media Independent Interface
MSI	Message Signaled Interrupts
MT/s	Megatransfers per second, Megasiirtoa sekunnissa

NTP	Network Time Protocol
OCXO	Oven-Controlled Crystal Oscillator, uunikideoskillaattori
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCI-X	Peripheral Component Interconnect Extended
PCIe	Peripheral Component Interconnect Express
PIC	Programmable Interrupt Controller, Ohjelmoitava keskeytyskäsittelijä
PIT	Programmable Interval Timer, Ohjelmoitava aikajaksoajastin
PLL	Phase Locked Loop, Vaihelukittu silmukka
PPM	Parts Per Million, Taajuussuhteen miljoonasosa
PPS	Pulse Per Second
TSC	Time Stamp Counter
TTL	Transistor-Transistor Logic
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
XGMII	10 Gigabit Media Independent Interface

Luku 1

Johdanto

1.1 Työn tausta

Operaattorien täytyy nykypäivänä mitata tietoverkkojaan, jotta vikatilanteista saadaan tietoa ja palvelunlaadusta saadaan jonkinlainen kuva. Mittausten perusteella voidaan myös arvioida verkkokapasiteetin laajennustarvetta.

Aktiivisia ja passiivisia tietoverkkojen mittauksia on ollut helppo tehdä, kun mittaus on suoritettu yhdellä mittalaitteella. Tällöin kaikkien tapahtuminen ajankohta pystytään määrittämään käyttäen samaa aikalähdettä. Tästä syystä aikaisemmin on suurimmaksi osaksi käytetty yksipistemittauksia tietoverkkojen mittauksiin ja analysointiin.

Kun mittauksia tehdään monesta paikasta usealla mittalaitteella, jokaisen laitteen kellon täytyy käydä samaa aikaa. Ilman synkronisointia ei ole mahdollista määrittää tapahtuman eriaikaisuutta mittapisteiden välillä. Kulkuaikaviive on yksi mitattavista asioista, joka vaatii tarkan kellojen synkronisoinnin. Tietoverkkojen siirtonopeuksien jatkuva kasvaminen aiheuttaa myös tarpeen synkronisoinnin parantamiselle. Kotikoneissakin yleisessä gigabittissä Ethernet-verkossa minimipituisen paketin lähetysviive on runsaasti alle mikrosekunnin.

Kaupallisten mittalaitteiden kellojen synkronisointi on mahdollista suorittaa tarkasti maailmanlaajuisen satelliittinavigointijärjestelmän (GPS) avulla. Mittapisteiden määrää rajoittaa ainoastaan mittalaitteiden korkea hinta. Jotta kustannukset pysyisivät kohtuullisina, mahdollisuutena on käyttää kalliiden mittalaitteiden sijaan normaaleja tietokoneita mittauksiin. Tällöin ongelmaksi kuitenkin muodostuu koneiden välinen kellosynkronisointi.

Passiivinen pakettien kaappaus on mahdollista tehdä erillisillä kaappaukseen suunnitelluilla korteilla, jotka osaavat pakettien aikaleimauksen. Jokainen erikoiskortti on mahdollista synkronoida GPS-vastaanottimen avulla ja näin saada tarkka aikainformaatio paketteihin. Ongelmana tässäkin vaihtoehdossa on kaappauskorttien korkea hinta, joka on vähintään kymmenenkertainen verrattuna normaaleihin verkkokortteihin.

1.2 Työn tavoite

Työn tavoitteet voidaan jakaa kolmeen eri osakokonaisuuteen. Ensimmäisenä tavoitteena on käyttää aikaisemmin hyväksi havaittua Antti Gröhnin kehittämää SynPCI-korttia [Grö04] kellosynkronisointiin. Kortista on tavoite kehittää uusi versio, joka lisää ominaisuuksia aikaisempaan prototyyppiin ja mahdollisesti parantaa sen suorituskykyä. Tavoitteena on myös saada laskettua kortin valmistuskustannuksia. Kehitystyö perustuu osittain Antti Gröhnin ideoihin, joita hänelle tuli työskennellessään Tietoverkkolaboratoriossa tämän aiheen parissa.

Toisena tavoitteena on kehittää kellosynkronoinnin hyvyyden ja pakettien aikaleimojen tarkkuuden mittausmenetelmiä. Menetelmän pitäisi minimoida mittavirheet tietokoneen ulkopuolisissa lähteissä. Kehitetyn menetelmän avulla on tarkoitus vertailla SynPCI-kortin avulla synkronoidun tietokoneen kellon tarkkuutta NTP:n avulla synkronoituun tietokoneeseen.

Viimeisenä tavoitteena on saada selvitettyä passiivisessa pakettikaappauksessa aikaleimojen virheen jakauman muoto, hajonta sekä minimi ja maksimi. Lisäksi työssä yritetään selvittää mistä komponenteista tämä virhe koostuu. Työssä on tarkoitus pohtia, kuinka löydettyjen virhelähteiden määrä ja vaikutuksia voitaisiin minimoida.

1.3 Työn rakenne

Aluksi työssä käydään läpi teoriaa mitkä osat tietokoneen arkkitehtuurissa liittyvät kellosynkronisointiin ja pakettikaappaukseen. Luvussa käydään läpi emolevyjen eri väylätekniikat ja ajan määrittämiseen käytettävät laskurit. Keskeytykset liittyvät olennaisena osana pakettikaappaukseen, koska verkkokortti ilmoittaa uuden paketin saapumisesta keskeytyksien avulla.

Reaaliaikainen tiedonkäsittely-luvussa käsitellään käyttöjärjestelmien toimintaa kellosynkronisoinnin ja pakettikaappauksen kannalta. Tämän lisäksi selvitetään, kuinka

aikaa ylläpidetään käyttöjärjestelmissä ja kuinka verkkoliikenne käsitellään.

SynPCI-X –kortista on oma lukunsa, jossa käydään ensiksi läpi miksi kortti päädyttiin alunperin kehittämään. Luvussa käydään läpi kortin uuden version suunnittelu ja sen ongelmat. Lisäksi kerrotaan, kuinka uusi kortti toimii ja mitä ongelmia siinä havaittiin.

Tämän jälkeen luvussa 5 käydään läpi mitä mitattiin ja minkälaisia tuloksia saatiin. Mittausten perusteella on tehty johtopäätökset mittauslaitteistojen synkronointi tarkkuudesta.

Luku 2

Tietokoneen arkkitehtuuri

Yleiskäyttöisen tietokoneen arkkitehtuuri koostuu suorittimen ympärillä olevista oheislaitteista. Luvussa käydään läpi kuinka eri komponentit ovat liitettyinä toisiinsa ja mitkä komponentit liittyvät ajan ylläpitoon ja verkkoliikenteen käsittelyyn. Yleisimpinä komponentteina tietokoneessa ovat etelä- ja pohjoissilta, jotka tarjoavat liitäntöjä oheislaitteille ja sisältävät tietokoneen toimintaan tarvittavaa logiikkaa. Suoritinta, etelä- ja pohjoissiltoja ja oheislaitteita yhdistävät eri väylät.

2.1 Väylät

Tietokoneessa väyliä käytetään oheislaitteiden liittämiseksi tietokoneeseen. Nykypäivän kotitietokoneissa yleisimpiä väyliä ovat Peripheral Component Interconnect (PCI), PCI Express (PCIe) ja Universal Serial Bus (USB). PCI- ja PCIe-väylä ovat tietokoneen sisäisiä väyliä ja USB-väylä on tarkoitettu ulkoisten oheislaitteiden liittämiseksi tietokoneeseen. USB-väylään onkin nykypäivänä mahdollista liittää melkein mikä tahansa oheislaite.

2.1.1 PCI

PCI-väylän spesifikaation ensimmäinen versio julkaistiin kesäkuussa 1992. Versiot 1.0 ja 2.0 sisälsivät tuen sekä 5 voltin että 3,3 voltin signaloinnille. Version 3.0 [ref02] myötä tuki 5 voltin signaloinnille poistettiin kokonaan. PCI-väylässä data siirretään rinnakkain joko 32 bitin tai 64 bitin väylässä. Väylänopeuksia on määritelty kaksi 33 MHz ja 66 MHz. Väylän siirtonopeus on maksimissaan $\frac{66\,\mathrm{MT/s}\times 64\,\mathrm{bit/T}}{8\,\mathrm{bit/B}} = 533\,\mathrm{MB/s}$. Yleisimmän 32-bittisen ja 33 MHz PCI-väylän nopeus on teoriassakin vain

133 MB/s. Koska dataväylä on jaettu kaikkien väylässä olevien laitteiden kesken, niin käytännössä nopeudet jäävät huomattavasti pienemmiksi kuin teoreettinen maksimi.

PCI-väylässä jokaisen laitteen, joka toimii siirron aloittajana, tulee pyytää vuoro liikennöintiin väylässä. Tämä tapahtuu käyttäen kysely (request) ja lupa (grant) –signaaleja. Koska jokaisen siirron alussa pitää kertoa muistiosoite, mihin dataa siirretään, väylässä siirrettävän hyötykuorman osuus pienenee mitä useammin vuoroja vaihdetaan. PCI-väylässä jokaiselle isäntälaitteelle annetaan maksimiaika, jonka yksi siirto voi kestää. Tällöin tiedetään kuinka kauan huonoimmassa tapauksessa jokin oheislaite joutuu odottamaan vuoroaan. Oheislaitteen sanotaan toimivan isäntätilassa, kun se toimii datasiirron aloittajana. Suorittimen toimiessa siirron aloittajana kortin sanotaan toimivan orjatilassa.

Normaalisti PCI-väylässä keskeytykset ilmaistaan käyttäen erillisiä keskeytysnastoja. Yhdessä väylässä erillisten keskeytysten määrä rajoittuu tällöin näiden nastojen takia neljään keskeytykseen. Myöhemmin versiossa 2.2 esiteltiin Message Signaled Interrupts (MSI) -viestit. Näiden avulla oheislaite voi käyttää useampaa kuin yhtä keskeytystä kirjoittamalla tavuja sille varattuun MSI-muistiavaruuteen.

2.1.2 PCI-X

PCI Extended (PCI-X) –määrittelyssä [ref00] tehtiin parannuksia väylän merkinantoprotokollaan väylänopeuden kasvattamisen lisäksi. Viiveen kannalta PCI-X –väylässä jokainen tapahtuma kestää yhden kellojakson enemmän, koska merkinantoprotokollaan on lisätty attribuuttivaihe. Määrittelyssä tehdyt parannukset keskittyvät lähinnä siirtonopeuden parantamiseen ja välimuistin parempaan yhteistoimintaan.

Versio 1.0 [ref00] määritteli mahdollisiksi väylänopeuksiksi 66 MT/s, 100 MT/s ja 133 MT/s. Määrittelyn versiossa 2.0 signalointinopeuksia lisättiin kaksi 266 MT/s ja 533 MT/s. PCI-X –määrittelyssä virheenkorjausta parannettiin pelkästä pariteettibitistä Error Correction Code (ECC) virheenkorjauskoodiin. Väylän laskennallinen maksiminopeus on $\frac{533\,\mathrm{MT/s} \times 64\,\mathrm{bit/T}}{8\,\mathrm{bit/B}} = 4264\,\mathrm{MB/s}$.

2.1.3 PCI Express

PCIe-väylä on Intelin kehittämä tekniikka. Suurimpana erona PCI- ja PCI-X –väyliin on sarjamuotoinen datasiirto ja jaetun väylän korvaaminen kaksipistelinkeillä ja kytkimellä. Jokaisella oheislaitteella on oma linkkinsä PCIe-kytkimeen, joka hoitaa pakettien välittämisen laitteiden välillä.

PCI Express –linkki koostuu kahdesta 2,5 Gbit/s Low Voltage Differential Signaling (LVDS) –parista. Laittamalla pareja rinnakkain 4, 8 tai 16 saadaan vastaavasti 4x, 8x tai 16x väylät. Jokaisen parin siirtonopeus on 250 MB/s. PCIe-väylässä on oma parinsa kumpaankin suuntaan ja näin siinä on mahdollista siirtää dataa molempiin suuntiin samanaikaisesti. PCIe-väylän maksiminopeus yhteen suuntaan on $16 \times 250\,\mathrm{MB/s} = 4000\,\mathrm{MB/s}$.

Nopean sarjamuotoisen datasiirron etuna rinnakkaismuotoiseen verrattuna on, että tarvitaan vähemmän nastoja signalointiin. Rinnakkaisessa datasiirrossa ongelmaksi muodostuu bittien välinen synkronointi ja ylikuuluminen, kun nopeutta kasvatetaan.

PCIe-väylässä nastojen määrää vähentää myös se, ettei jaetulle väylälle tarpeellisia vuoroja tarvitse signaloida erillisiä nastoja pitkin. Kaikki kontrolliliikenne mukaan lukien keskeytykset kulkevat PCIe-väylässä paketeissa muun dataliikenteen seassa. Tähän tarkoitukseen on laajennettu jo PCI-versiossa 2.2 esiteltyjä MSI-viestejä. PCIe-silta muuttaa kaikki luku- ja kirjoituskäskyt MSI-viesteiksi, jotka kulkevat paketteina oikealle laitteelle.

2.1.4 Hyper Transport

Hyper Transport –väylä on Hyper Transport –yhteenliittymän kehittämä väyläarkkitehtuuri. Yhteenliittymän perustajajäseniin kuuluvat Advanced Micro Devices, Alliance Semiconductor, Apple, Broadcom, Cisco Systems, NVIDIA, PMC-Sierra, Sun Microsystems ja Transmeta [ref06d].

Hyper Transportin alkuperäisessä 1.03 [ref01] määrittelyssä on tuki 2, 4, 8, 16 tai 32 bittiä leveille väylille. Mahdollisia kellotaajuuksia ovat 200, 300, 400, 500, 600, 800 tai 1000 MHz. Hyper Transport on pakettipohjainen väylä aivan kuten PCIe. Jokainen datasiirto aloitetaan kahdella 32-bitin sanalla, jotka sisältävät komennon ja kohdeosoitteen. Signaloinnissa käytetään 2,5 V LVDS-pareja PCIe-väylän tapaan ja jokaisella kellojaksolla siirretään dataa sekä nousevalla että laskevalla kellojaksolla. Kyseessä on siis Double Data Rate (DDR) väylä. Myöhemmin 1.03 versioon lisättiin tuki 64-bittisille osoitteille ja parannettiin yhteensopivuutta PCI-X 2.0 -siltojen kanssa.

Hyper Transport 2.0 -määrittely [ref04c] lisäsi kolme mahdollista väylätaajuutta 1,2 , 1,4 ja 1,6 GHz. PCI- ja PCI-X –siltojen tuen lisäksi uutena ominaisuutena on tuki PCIe-silloille. Versio 2.0 on kuitenkin täysin alaspäin yhteensopiva vanhan 1.03 version kanssa.

Määrittelyn 3.0 version [ref06b] myötä mahdollisia väylänopeuksia tuli lisää. Uusi-

na kellotaajuuksina esiteltiin 1,8 , 2,0 , 2,2 , 2,4 ja 2,6 GHz. Uutena ominaisuutena tuli väylän toiminta vaihtovirralla, joka mahdollistaa pidemmän fyysisen väylän kuin toiminta tasavirralla. Tämän ansiosta Hyper Transport –väylä on mahdollista kuljettaa kaapelissa esimerkiksi tietokoneiden välillä. Muita uusia ominaisuuksia olivat korttien kytkeminen ajonaikana, linkkien jakaminen ja virransäästöominaisuudet.

Hyper Transport 3.0 –väylän teoreettiseksi maksiminopeudeksi yhteen suuntaan saadaan $\frac{5200\,\mathrm{MT/s}\times 32\,\mathrm{bit/T}}{8\,\mathrm{bit/B}} = 20800\,\mathrm{MB/s}$. Vaikka teoreettista väylänopeutta ei voida saavuttaa kuin hetkellisesti, väylä on selvästi nopein kaikista edellä mainituista väylätekniikoista.

2.2 Keskeytykset

Tietokoneissa keskeytyksiä käytetään nimensä mukaisesti keskeyttämään suorittimen toiminta. Keskeytyksiä käytetään sen takia, että suorittimen ei tarvitse käydä säännöllisen välein tarkistamassa onko joku tapahtuma tapahtunut. Eli toisin sanoen, kun keskeytys tapahtuu, suoritin siirtyy tarkastamaan mikä tapahtuma aiheutti keskeytyksen ja mitä toimenpiteitä silloin on tehtävä. Tietokoneissa keskeytykset ovat jaettu laitteisto- ja ohjelmistopohjaisiin keskeytyksiin.

Ohjelmistopohjaiset keskeytykset tuotetaan tietyllä suorittimen konekielen käskyllä. Näin ollen niitä sanotaan synkronisiksi, koska keskeytys tapahtuu aina tietyn konekielisen käskyn jälkeen.

Laitteistopohjaiset keskeytykset tulevat normaalisti suorittimen ulkopuolelta kuten oheislaitteilta, jotka tarvitsevat suorittimen huomiota. Näiden keskeytysten ajankohtaa ei ole mahdollista tietää ennakolta, ja siksi niitä kutsutaan asynkronisiksi keskeytyksiksi. Nykyisten suorittimien pitkä liukuhihna joudutaan myös tyhjentämään tällaisen keskeytyksen saapuessa. Laitteistopohjaiset keskeytykset on kuitenkin yleensä mahdollista peittää. Tällöin suoritin jättää keskeytyksen huomioimatta kunnes keskeytyksen peittäminen otetaan pois päältä.

Laitteistopohjaisten keskeytysten hallintaan tarvitaan oma piirinsä, jolla useamman laitteen keskeytykset saadaan kanavoitua suorittimen keskeytysnastaan. Tähän tarkoitukseen käytettiin alun perin x86-arkkitehtuurissa yhtä tai kahta 8259A-piiriä. Tätä piiriä kutsutaan nimellä Programmable Interrupt Controller (PIC). Kyseinen 8259A-piiri tukee 8 keskeytystä ja mahdollistaa 8 orjapiirin liittämisen. Kahdella piirillä on mahdollista kanavoida 16 keskeytystä, joka on hyvin yleinen konfiguraatio. Piirin toteutus löytyy vielä tänäkin päivänä uusista emolevyistä integroituna

eteläsiltaan. Huonona puolena piirissä on, että keskeytyksiä joudutaan jakamaan, jos oheislaitteita on paljon. Lisäksi 8289A-piirissä on rajoittuneet keskeytysten priorisointimahdollisuudet.

Koska PIC-piiri ei tue kuin yhtä suoritinta, Intel kehitti monisuoritinjärjestelmiä varten Advanced Programmable Interrupt Controller (APIC) –piirin. Tällöin jokaisessa suorittimessa on paikallinen APIC (LAPIC) –piiri ja I/O-APIC-piiri hoitaa kaikki laitteistopohjaiset keskeytykset.

AMD:n uusien suorittimien liitäntäväylänä on Hyper Transport, joten emolevyllä on oltava piiri, joka emuloi PIC-piiriä ja muuntaa keskeytykset Hyper Transport –viesteiksi. Oletusarvoisesti BIOS konfiguroi emolevyn niin, että käynnistyksen jälkeen keskeytykset kulkevat PIC-piirin kautta. Käyttöjärjestelmän on mahdollista konfiguroida emolevyn piirisarjat uudestaan niin, että keskeytykset kulkevat joko I/O-APIC-piirin kautta tai suoraan Hyper Transport –keskeytyksinä.

2.3 Laskurit

Laskureita tietokoneissa käytetään mittamaan kahden tapahtuman välistä aikaa. Laskureita on myös mahdollista käyttää generoimaan jaksollisia keskeytyksiä. Näiden keskeytysten avulla on mahdollista suorittaa tehtäviä tasaisin väliajoin.

Alun perin emolevyt sisälsivät vain joko 8253 tai 8254 Program Interval Timer (PIT) –piirin. Piirit sisältävät kolme 16-bittistä ohjelmoitavaa laskuria, joita voidaan lukea tai kirjoittaa tavu kerrallaan. Laskureiden lukemiseen ja kirjoittamiseen tarvitaan kaksi operaatiota, joten tiheä kirjoittaminen ja lukeminen ei ole järkevää. Sen takia yhtä näistä laskureista voidaan käyttää generoimaan laitteistopohjaisen keskeytyksen numero 0 aina, kun laskuri saavuttaa nollan. Aikasemmin esitellyssä 8259A-keskeytyskäsittelijässä keskeytys numero 0 on varattu jaksollista keskeytystä varten.

Advanced Configuration and Power Interface (ACPI) –määritelmän [ref06a] mukaan jokaisen sitä tukevan emolevyn täytyy toteuttaa virranhallintalaskuri. Tämän laskurin taajuudeksi on määritelty 3,579545 MHz ja sen koko on joko 24 tai 32 bittiä.

Intelin 586 suorittimista lähtien suoritin on sisältänyt Time Stamp Counter (TSC) –laskurin. Tämä on jokaisella suorittimen kellojaksolla kasvava 64-bittinen laskuri. Koska laskuri on suorittimen sisällä, sen lukuviive on aina pienempi kuin muilla laskureilla.

Intel on kehittänyt PIT-piirin korvaajaksi High Precision Event Timers (HPET) –laskurin [ref04d]. HPET-laskuri on määritelty 64-bittiseksi laskuriksi, jonka taa-

juus on vähintään 10 MHz ja taajuusvirhe on pienempi kuin 500 ppm. HPET sisältää kolme vertailupiiriä, joilla on mahdollista luoda keskeytyksiä. Vähintään yhden vertailijoista on myös toimittava jaksollisessa tilassa.

2.4 Verkkokorttien arkkitehtuuri

ISO OSI-mallin [ISO94] mukaan verkkokortti toteuttaa mallin kaksi ensimmäistä kerrosta: ensimmäisen kerroksen eli fyysisen kerroksen ja toisen eli siirtoyhteyskerroksen. Näiden kahden osan lisäksi verkkokortti täytyy olla liitettynä tietokoneeseen jonkin väylän kautta.

Verkkokorteissa kustannussyistä nämä kaikki kolme osaa on toteutettu yhdellä ASIC-piirillä. Piirin lisäksi kortille tarvitaan suojaerotusmuuntaja, koska 802.3 standardissa [IEE05] on määritelty, että kortin tulee läpäistä IEC 60950 [IEC91] standardin mukainen suojaeristys. Piirin kellopulssia varten kortille tarvitaan yksi kide. Kiteen yleisin taajuus on 25 MHz, joka on 10/100 Mbit/s Ethernet-linkin signalointinopeus. Kyseisestä 25 MHz taajuudesta on myös helppo vaihelukitun silmukan (PLL) avulla generoida 125 MHz taajuus, joka on gigabittisen Ethernet-linkin signalointinopeus.

2.4.1 Liitäntäväylät

Fyysisen osan ja siirtoyhteysosan välinen liitäntäväylä on määritelty IEEE:n standardissa 802.3[IEE05]. Liitäntäväylä on 10/100 Mbit/s Ethernet-verkon tapauksessa nimeltään Media Independent Interface (MII), gigabittisen Ethernet-verkon tapauksessa Gigabit Media Independent Interface (GMII) ja 10 gigabitin Ethernet-verkon tapauksessa 10 Gigabit Media Independent Interface (XGMII).

Siirtoyhteyskerroksen liittämiseksi tietokoneeseen voidaan käyttää luvussa 2.1 esiteltyjä PCI-, PCI-X- ja PCIe-väyliä. Emolevyille integroidut verkkokortit ovat myös kytkettyinä jollain väylätekniikalla etelä- tai pohjoissiltaan. Poikkeuksena on muutama piirisarja, joissa Ethernet –toiminnallisuus on integroitu etelä- tai pohjoissiltaan. Suurimmaksi osaksi verkkokortti kuitenkin näkyy käyttöjärjestelmälle PCI-laitteena.

2.4.2 Fyysinen kerros

Fyysisen kerroksen tehtävänä on suorittaa muunnos digitaalisista biteistä analogiseksi signaaliksi ja päinvastoin. Lähetyspuolella tähän tarvitaan koodausosa, digitaalianalogimuunnin (DAC) ja pulssin muokkaussuodattimet. Vastaanottopuolella

tarvitaan analogidigitaalimuunnin (ADC), dekoodausosa ja kaiunpoistoon tarkoitetut suodattimet. Fyysisenkerroksen osia on tarkasteltu tarkemmin kappaleessa 2.4.4, jossa tutustutaan Intel PRO/1000 verkkokorttiin.

Gigabitin verkkokortin koodaus- ja dekoodausosien tulee toteuttaa Manchester-, 4B/5B- ja 4B/PAM5-enkoodaukset, jotta se voi tukea kaikkia kolmea nopeutta 10, 100 ja 1000 Mbit/s. Gigabittistä kuparista Ethernet-verkkoa varten ADC-muuntimen on toimittava vähintään 125 MHz näytteenottotaajuudella.

2.4.3 Siirtoyhteyskerros

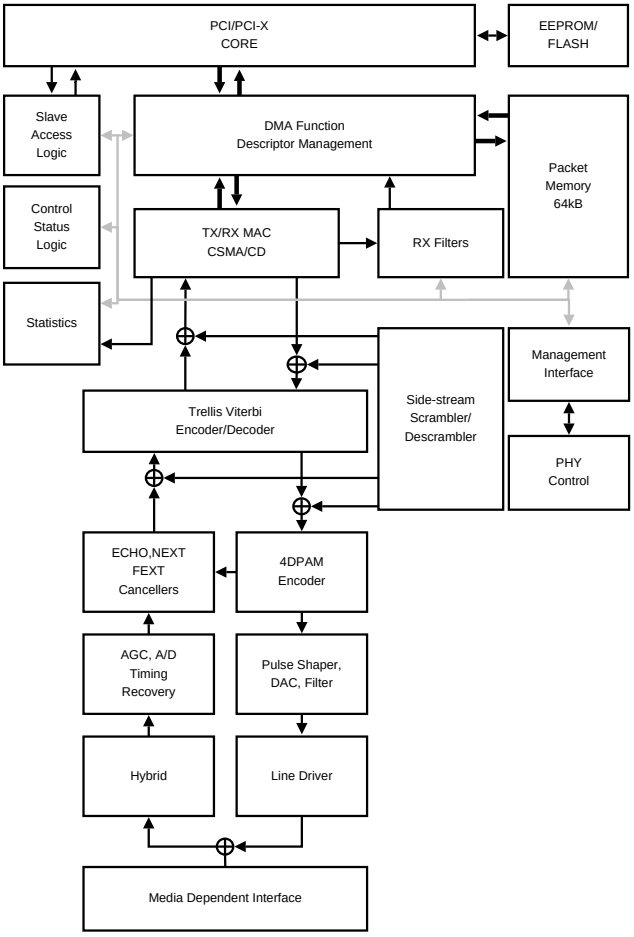
Siirtoyhteyskerroksen tehtävänä on toteuttaa Ethernet-verkon tarvitsemat lähetys- ja vastaanottoprotokollat, jotka on määritelty IEEE:n standardissa 802.3[IEE05]. Kerroksen tehtävänä on huolehtia pakettien siirrosta verkkokorttien välillä. Tätä varten jokaisella verkkokortilla on MAC-osoite, jonka avulla verkkokortit tunnistavat omat pakettinsa. Kerros huolehtii myös lähetysten ajoittamisesta, jos kyseessä on jaettu media. Nykyisessä kytketyssä Ethernet-verkossa kaikki linkit kuitenkin ovat kaksisuuntaisia kaksipistelinkkejä.

2.4.4 Intel PRO/1000

Intelin integroiduista piireistä 82545GM sisältää edellisessä kappaleessa kaikki kolme mainittua osaa. Piiristä on myös kaksiporttinen versio 82546GB [ref05a]. Siinä on samalle piirille kahdennettu kaikki muut osat paitsi PCI-logiikka. Kortti kuitenkin näkyy PCI-väylässä kahtena erillisenä laitteena. Lisäksi piiristä on olemassa PCIe-versio. Sisäisesti kortit ovat melkein samanlaisia lukuunottamatta pieniä eroja, liitäntätekniikkaa ja samalle piirille integroitujen verkkokorttien määrää.

Intelin gigabittisten verkkokorttien toiminta on selitetty tarkasti ohjelmistokehittäjän manuaalissa [ref06c]. Keskitymme tässä kappaleessa tarkemmin Intelin kaksiporttiseen versioon, mutta suurin osa esille tulevista asioista pätee myös muihin Intelin gigabittisiin verkkokortteihin.

Kuvassa 2.1 on Intelin verkkokorteissa käytetyn piirin lohkokaavio. Liitäntäväylänä tietokoneeseen toimii 32/64-bittinen PCI-X 1.0 tai PCI 2.3 -väylä, joka osaa toimia kaikilla mahdollisilla väylänopeuksilla. PCI/PCI-X –logiikka on yhteydessä kortin molempiin Direct Memory Access (DMA) -logiikoihin. Logiikka toimii väylässä orjatilassa PCI-laitteiden konfiguroinnin aikana ja silloin, kun kortin asetuksia muutetaan muistikartoituksen kautta. Kortti toimii isäntätilassa silloin, kun se siirtää



Kuva 2.1: Intelin verkkokortin lohkokaavio

paketteja kortin sisäisestä muistista keskusmuistiin tai päinvastaiseen suuntaan.

Kortin sisäistä 64 kB muistia käytetään vastaanotettaville ja lähetettäville paketeille FIFO-jonoina. Lähetys- ja vastaanottojonojen pituudet on mahdollista ohjelmallisesti muuttaa eri pituisiksi, kunhan ne vain mahtuvat kortin muistiin.

MAC-lohko toteuttaa sekä MII- että GMII-liitäntäväylät ja IEEE:n standardissa 802.3[IEE05] määritellyt Ethernet-verkon pakettien lähetys- ja vastaanottoprotokollat. Lisäksi kortilla on mahdollista suodattaa paketteja MAC-osoitteen ja VLAN-numeron perusteella. Kortille on pystytään asettamaan 16 yksittäis- tai ryhmä-MAC-osoitetta, jotka pääsevät vastaanottosuodattimesta läpi. Suodattimet voidaan myös ottaa kokonaan pois käytöstä erikseen yksittäis- ja ryhmälähetyksille. VLAN-suodatusta käytettäessä kortti poistaa paketista VLAN-kehyksen ja ottaa VLAN-numeron talteen. Kortti tukee 4096 VLAN-suodatinta eli kaikkia mahdollisia VLAN-

numeroita.

Fyysinen osa sisältää tarvittavat rinnansarjamuuntimet ja kooderit. Koodaustekniikkoina käytetään Manchester-, 4B/5B- ja 4D/PAM5-koodauksia linjanopeuden mukaan. Lisäksi piiri osaa tehdä symboleille polynomisen sekoituksen elektromagneettisen säteilyn vähentämiseksi. Koska gigabittisillä Ethernet-linkillä liikennöidään samalla parilla kumpaankin suuntaan yhtä aikaa, piirillä on tehtävä kaiunpoisto ennen dekoodausta. Gigabittisellä Ethernet-linkillä orjamoodissa toimivan kortin on myös generoitava sisäinen kellopulssi linjasignaalista. Tätä varten piiri toteuttaa kellosignaalin hienosäätämisen sisääntulevien symbolien mukaisesti. Jotta kellosignaali ei hukkuisi, gigabittinen Ethernet-verkkokortti lähettää idle-paketteja, kun muita paketteja ei ole lähetettävänä.

Piirin DMA-koneisto hoitaa pakettien siirtämisen piirin sisäisen muistin ja tietokoneen keskusmuistin välillä. Tähän käytetään omaa rengaspuskuria sekä lähetettäville että vastaanotettaville paketeille. 82546GB-piirillä rengaspuskurin maksimikoko on 4096 pakettia. Kortilla on osoittimet ensimmäiseen vapaaseen paikkaan ja jonon ensimmäiseen varattuun paikkaan. Käyttöjärjestelmän ajuri lukee jonon ensimmäisen paketin ja siirtää osoitinta yhdellä eteenpäin. Kortti siirtää paketteja sitä mukaa sisäisestä muististaan keskusmuistiin, jos vain keskusmuistissa olevassa rengaspuskurissa on tilaa.

Piirissä on monipuoliset ominaisuudet vähentää kortin generoimia keskeytyksiä. Keskeytyksien määrä sekunnin aikana on mahdollista rajata tiettyyn maksimimäärään. Käytännössä tämä tarkoittaa sitä, että keskeytyksiä ei generoida tiheämmin kuin mitä saadaan, jos maksimi keskeytysmäärä jaetaan tasavälein sekunnin ajalle. Keskeytyksien maksimimäärän ollessa 1000 tämä tarkoittaa esimerkiksi sitä, että keskeytyksiä ei generoida tiheämmin kuin yhden millisekunnin välein.

Pakettien vastaanotossa kortille on määritelty neljä erilaista keskeytystä. Piirille voidaan ohjelmoida laskuri, joka kertoo, kuinka kauan viimeisen paketin jälkeen odotetaan ennen kuin keskeytys signaloidaan. Tämä laskuri nollataan aina, kun paketti saapuu tai jokin muu vastaanottokeskeytys generoidaan. Tällä laskurilla keskeytykset vähenevät, jos useampi paketti saapuu tämän aikaikkunan sisällä edellisestä paketista.

Toinen ohjelmoitava laskuri pitää kirjaa kuinka paljon aikaa on kulunut edellisestä vastaanottokeskeytyksestä. Laskuri käynnistetään edellisen vastaanottokeskeytyksen jälkeen, kun seuraava paketti on siirretty keskusmuistiin. Tällä varmistetaan, etteivät paketit joudu odottamaan käsittelyään liian pitkään, jos paketteja saapuu riittävän tiheään.

Kortille on myös mahdollista asettaa pienen pakettikoon raja. Jos paketti on tätä rajaa pienempi, niin keskeytys generoidaan välittömästi, kun paketti on siirretty keskusmuistiin. Ajatuksena on pienentää pienten pakettien viivettä. Näitä paketteja ovat esimerkiksi TCP-ack –viestit ja DNS-kyselyt. Neljäs vastaanottokeskeytys liittyy rengaspuskurin ylivuotoon eli keskeytys generoidaan, jos vastaanottorengaspuskuri on täynnä.

2.4.5 Broadcom NetXtreme

NetXtreme-sarja sisältää Broadcomin integroidut gigabittiset Ethernet-piirisarjat. Piirisarjaperhe sisältää piirejä eri käyttötarkoituksiin kuten esimerkiksi työpöytä-, mobiili- ja palvelinkäyttöön. Lisäksi piirejä on saatavilla PCI-, PCI-X- ja PCIe-väyläliitäntätekniikoilla. Mittauksissa käytetyistä koneista toisessa on emolevylle integroitu BCM5704C-piiri ja toisessa kaksi BCM5721-piiriä. Suurin ero piirien välillä on, että BCM5704C-piiri on suunniteltu käyttämään 64-bittistä PCI-X v1.0 tai PCI v2.3 –väylää ja BCM5721-piiri käyttää PCIe v1.0a x1 –väylää. BCM5704C-piirissä on samalle piirille integroitu kaksi verkkokorttia, kuten aiemmin esitellyssä Intelin kortissakin.

Erona Intelin gigabittisiin piireihin Broadcomin piireissä on integroitu RISC-suoritin pakettien luokittelua varten. Piirit tukevat neljää prioriteettijonoa ja pakettien luokittelu voidaan tehdä jo kortilla. Koska Broadcomin piirien toiminasta ei ole saatavilla tarkkaa tietoa kuten Intelin piireistä, toiminnan tarkastelu perustuu Linuxin ajureiden tarkasteluun ja tuotteiden esitelehtiin. Piirin toiminta on kuitenkin ajurin perusteella hyvin samanlaista kuin Intelillä. Rengaspuskurin koko on maksimissaan 512 pakettia vastaanotto- ja lähetyspuskurin tapauksessa. Keskeytyksien määrää voidaan pienentää kuten Intelin verkkokortissakin. Tähän tarkoitukseen on mahdollista määrittää arvo kuinka kauan odotetaan viimeisen paketin jälkeen ennenkuin signaloidaan keskeytys. Lisäksi on mahdollista arvo, joka määrittelee edellisen vastaanotetun paketin ja keskeytyksen signaloinnin väliajan.

2.5 Pakettikaappauksen pullonkaulat

Tietokoneella tehtävässä pakettikaappauksessa pullonkaulaksi muodostuu suorittimen nopeus käsitellä paketteja, jos verkkokortit on kytketty koneeseen riittävän nopealla väylällä. 32-bittiseen PCI-väylään voidaan kytkeä muutama 100 Mbit/s verkkokortti, jos levyohjaimet ovat samassa väylässä. Jo yksi gigabittinen verkko-

kortti tukkii normaalin 32-bittisen PCI-väylän. Gigabittiset verkkokortit pitäisikin kytkeä mieluiten PCIe-väylään, joka ei ole jaettu muiden laitteiden kanssa.

Vaikka suorittimen nopeus riittäisi käsittelemään paketteja, niin nykyisissä tietokoneissa muistien nopeudet ovat huomattavasti hitaampia kuin suorittimen nopeus. Tämän takia suorittimiin on integroitu erinopeuksisia ja -kokoisia välimuisteja.

Advanced Micro Devices:n (AMD) valmistamissa nykyaikasissä suorittimissa muistiohjain on integroitu suorittimeen. Intelin arkkitehtuurissa muistiohjain on entiseen tapaan emolevyn pohjoissillassa. Intelin tapauksessa pullonkaulaksi voi muodostua dataväylä suorittimen ja muistiohjaimen välillä. Intel on kuitenkin kehittänyt monimutkaisia algoritmeja, joiden perusteella dataa siirretään keskusmuistista suorittimen välimuisteihin. AMD:n ratkaisussa muistiviiveet jäävät kuitenkin pienemmiksi kuin Intelin nykyisessä ratkaisussa.

Pakettikaappauksessa siirretään suuria määriä dataa, jolloin muistin nopeus näyttelee merkittävää osaa. Kuten kappaleessa 2.4.4 esiteltiin, verkkokortin rengaspuskuri sijaitsee tietokoneen keskusmuistissa, johon verkkokortti siirtää paketit DMA:n avulla. Rengaspuskurissa paketteja ei kuitenkaan voida säilyttää kovin pitkään vaan suorittimen on käsiteltävä ne. Käsittelyn aikana suoritin voi joutua kopioimaan paketin toiseen muistipaikkaan useamman kerran. Muistin luku- ja kirjoitusoperaatiot kestävät useita suorittimen kellojaksoja, ellei muistialue ole suorittimen ykköstason välimuistissa. Pakettikaappauksen tapauksessa, kun pakettia käsitellään ensimmäistä kertaa, paketti ei yleensä ole missään välimuistissa. Tälläinen tapaus esintyy ainostaan, jos rengaspuskurissa on useampi paketti odottamassa käsittelyä.

Intel on kehittänyt I/O-kiihdytysteknologian [ref05d] tämän ongelman ratkaisuksi. Ajatuksena on lisätä pohjoissiltaan DMA-logiikka, jolla voidaan tehdä muistisiirtoja ilman, että suorittimen täytyy odottaa kopioinnin valmistumista.

Myöhemmin huomataan että, yleiskäyttöisiä käyttöjärjestelmiä, kuten Linuxia ja FreeBSD:tä, ei ole optimoitu pitäen silmällä pelkkää pakettikaappausta.

Luku 3

Reaaliaikainen tiedonkäsittely

Reaaliaika vaatimus tiedonkäsittelyssä asettaa tiettyjä vaatimuksia laitteistolle ja käyttöjärjestelmälle. Luvussa käydään läpi mitä vaatimuksia järjestelmän tulee toteuttaa ja kuinka käyttöjärjestelmien verkkoliikenteen käsittely ja ajan ylläpito on toteutettu.

3.1 Reaaliaikajärjestelmät

3.1.1 Määritelmä

Reaaliaikajärjestelmä on "järjestelmä, jonka täytyy suoriutua tehtävästään ennalta määrätyssä rajoitetussa vasteajassa tai muuten systeemi kärsii vakavista seurauksista"[Lap04]. Reaaliaikajärjestelmässä oikean vastauksen laskemiseen on ennalta määrätty rajoitettu aika. Reaaliaikajärjestelmä on myös mahdollista määritellä "systeeminä, jonka looginen virheettömyys perustuu sekä ulostulojen virheettömyyteen, että niiden oikea-aikaisuuteen"[Lap04]. Reaaliaikajärjestelmän kannalta oikeakin vastaus on väärä, jos sitä ei saada ennen ennalta määrättyä aikarajaa.

3.1.2 Pehmeät ja kovat aikarajat

Reaaliaikajärjestelmät voidaan jakaa kahteen luokkaan sen perusteella, mitä systeemille tapahtuu, kun se ei saavuta ennalta annettuja aikarajoja. Pehmeän reaaliaikajärjestelmän suorituskyky laskee, mutta ei kuitenkaan johda kaatumiseen, jos ei pysytä annetuissa aikarajoissa. Kovissa reaaliaikajärjestelmissä yksikin myöhästyminen annetuista aikarajoista johtaa järjestelmähäiriöön.

3.1.3 Deterministisyys

Determistininen systeemi määritellään "systeeminä, jonka jokaisella tilalla ja kaikilla mahdollisilla syötteillä on yksiselitteinen seuraava tila ja joukko ulostuloja"[Lap04]. Ajallisesti systeemi on deterministinen, jos jokaiselle syöte/tilaparille voidaan määrittää täsmällisesti suoritusajan yläraja.

3.2 Reaaliaikakäyttöjärjestelmät

Reaaliaikakäyttöjärjestelmät ovat käyttöjärjestelmiä, joissa on otettu huomioon edellä esitetyt reaaliaikajärjestelmien vaatimukset. Normaaleja käyttöjärjestelmiä ei ole suunniteltu reaaliaikajärjestelmän vaatimuksia ottaen huomioon. Esimerkiksi ei-keskeyttävässä (non-premptive) Linuxissa järjestelmäkutsujen suoritusaikaa ei voida ennalta tietää, jos kutsu joutuu odottamaan oheislaitetta. Linuxin versiosta 2.6 lähtien ydin on ollut käännettävissä tukemaan systeemikutsujen keskeytystä.

Linuxin 2.6 versiosta lähtien aikatauluttaja on ollut determistinen O(1). Tämä on hyvä asia reaaliaikakäyttöjärjestelmän kannalta, koska aikatauluttaja ei ole keskeytettävissä.

Normaalissa 2.6-sarjan ytimessä systeemikutsujen kriittiset osat on suojattu silmukkalukoilla. Silmukkalukon sisällä suoritinta (CPU) ei ole mahdollista keskeyttää. Huonoimmassa tapauksessa silmukkalukoista aiheutuvat viiveet ovat muutamien millisekuntien luokkaa. Alun perin Monta Vista Softwaren kehittämästä reaaliaikalaajennuksesta on olemassa vapaan lähdekoodin versio [ref05c]. Tämä laajennus tekee Linuxin ytimestä melkein täysin keskeytettävän. Laajennuksessa silmukkalukot on korvattu keskeytettävillä keskinäisillä poissulkemisilla (mutual exclusions). Näiden ulkopuolelle jää muutama ei-keskeytettävä systeemikutsu, joiden viiveet ovat kuitenkin determinisiä ja järjestelmän mukaan ne kestävät joitakin kymmeniä mikrosekunteja. Tällöin huonoimmassakin tapauksessa viiveet jäävät alle sataan mikrosekuntiin.

3.3 Keskeytyksien hallinta

Keskeytyksien hallinta täytyy suorittaa jollain tavalla käyttöjärjestelmissä, jotta keskeytyksen saapuessa voidaan aloittaa siihen liittyvän ohjelman suoritus.

Linuxissa keskeytyksien käsittely sisältää kolme eri abstraktiotasoa. Ensimmäisenä on rajapinta, jota ajurit käyttävät. Tämän kautta on mahdollista varata, vapauttaa,

estää, sallia sekä synkronoida keskeytyksiä. Seuraava abstraktiotaso määrittelee erilaisille keskeytystyypeille omat funktionsa, koska keskeytyksien käsittely on erilainen esimerkiksi sen mukaan, onko se reuna- tai tasoliipaistava.

Alin taso on rajapinta emolevyllä sijaitseville keskeytyskäsittelijöille, jotka esiteltiin kappaleessa 2.2. Tämä rajapinta määrittelee, kuinka keskeytykset saadaan kuitattua sekä kuinka keskeytyksen peittäminen saadaan asetettua päälle ja pois. Keskeytyskäsittelijän tiedot on tallennettu *irq_ chip* –nimiseen tietueeseen. Tietue sisältää esimerkiksi tiedon siitä, mitä funktiota kutsutaan, kun keskeytys tapahtuu.

Normaalissa Linuxissa ainoastaan ohjelmistokeskeytyksille on oma säikeensä. Monta Vistan reaaliaikalaajennos lisää myös laitteistopohjaiset keskeytykset omiin säikeisiinsä. FreeBSD:n puolella normaalissa ytimessäkin jokainen keskeytys on oma säikeensä. FreeBSD:ssä kaikki keskeytykset säilötään *intr_ event* –tietueeseen, joka on linkitetty lista. FreeBSD:ssä on, kuten Linuxissakin, määritelty rajapinta keskeytyskäsittelijöille.

3.4 Verkkoliikenteen käsittely

Jotta paketit saataisiin eri merkkisien verkkokorttien kautta ytimen verkkopinoon, täytyy ytimessä olla jokaiselle verkkokortille oma ajurinsa, joka huolehtii pakettien siirtämisestä ytimelle. Jokaisessa käyttöjärjestelmässä on paketeille oma tietueensa, johon verkkokorttien ajurit siirtävät tiedot paketeista ja niiden sisällöstä.

3.4.1 Linux 2.6

Kuten kappaleessa 2.4.4 mainittiin verkkokortti toteuttaa paketeille rengaspuskurin, johon verkkokortti siirtää paketit DMA-logiikan avulla. Linuxissa jokainen paketti laitetaan *sk_ buff* –nimiseen tietueeseen. Tietue sisältää kaiken paketin käsittelyyn tarvittavan tiedon. Verkkokortti asettaa tietueeseen ne tiedot, mitkä sillä on tiedossaan. Pakettien aikaleimaus tehdään kuitenkin tarvittaessa vasta, kun paketti saapuu verkkokortin ajurilta ytimelle.

Verkkokortin ajuri siirtää *sk_ buff* –tietueen ytimelle ja varaa rengaspuskuriin uuden *sk_ buff* –tietueen. Ennen kuin Linuxissa otettiin käyttöön uusi verkkorajapinta (NAPI), jokainen paketti käsiteltiin yksitellen aina keskeytyksen tullessa. Ongelmaksi tässä toimintatavassa kuitenkin muodostuu keskeytysten suuri määrä [SOK01]. Jos verkkokortilta tulee uusi keskeytys ennen kuin edellinen on ehditty käsitellä, kaikki suorittimen aika kuluu pelkkien keskeytysten käsittelyyn. NAPI-rajapinnassa

verkkokortin keskeytyksen saapuessa ainoastaan lisätään verkkokortti kiertokyselylistan viimeiseksi. Tämän jälkeen ajastetaan ohjelmistopohjainen keskeytys, joka on tyypiltään NET_RX_SOFTIRQ, ja estetään verkkokortin uudet keskeytykset. Ohjelmistokeskeytyskäsittelijöitä on yksi suoritinta kohti. Käsittelijän tehtävänä on jakaa vuoroja eri verkkokorttien kesken ja muiden ohjelmistokeskeytyksien välillä. Verkkokortin keskeytys sallitaan sen jälkeen, kun kaikki paketit on käsitelty rengaspuskurista. Näin ollen, jos suoritin ei ehdi käsittelemään kaikkia paketteja, verkkokortti jättää paketin käsittelemättä rengaspuskurin ollessa täynnä.

3.4.2 FreeBSD 6.1

FreeBSD:n toteutus on lähestulkoon samanlainen kuin Linuxin. FreeBSD:ssä paketin tiedot tallennetaan *mbuf* –tietueeseen. Tämä tietue siirrettään ytimelle, kun paketti saapuu, ja rengaspuskuriin varataan uusi tietue. Myös FreeBSD tukee verkkokorttien kiertokyselyä. Kiertokysely on kuitenkin toteutettu hitusen eri tavalla kuin Linuxissa. Keskeytyksiä ei käytetä, vaan verkkokorttien kiertokysely-funktiota kutsutaan joka kerta, kun aikatauluttaja käynnistetään. Käyttäjä voi myös itse määritellä kuinka monta prosenttia suorittimen ajasta käytetään maksimissaan kiertokyselyihin ja kuinka monta pakettia voidaan käsitellä yhdellä kiertokyselyllä. Koska keskeytyksiä ei käytetä, niin paketeille aiheutuu tarpeetonta viivettä. Esimerkiksi aikatauluttajan taajuudella 1000 Hz paketti joutuu huonoimmassa tapauksessa odottamaan yhden millisekunnin. Ongelman kiertämiseksi FreeBSD:ssä on mahdollista suorittaa kiertokyselyä myös sillon, kun suorittimella ei ole mitään tekemistä. Tämä avulla pakettien ei tarvitse odottaa käsittelyä turhaan, jos suorittimella ei ole tärkeämpää tekemistä.

Huonona puolena FreeBSD:n kiertokyselyssä on, että kiertokyselyprosessi käyttää vain yhtä suoritinta. Käyttäjä ei voi myöskään pakottaa tiettyä suoritinta kiertokyselemään tiettyä verkkokorttia. Keskeytyspohjaisessa käsittelyssäkään ei voida määrätä, mikä suoritin käsittelee tietyn verkkokortin keskeytykset. Tämän takia ainakaan FreeBSD 6.1 ei juuri hyödy verkkoliikenteen osalta useammasta suorittimesta.

3.5 Ajan ylläpito

Käyttöjärjestelmissä ajan ylläpitoa täytyy suorittaa, jotta voidaan määrittää tapahtumahetkien kellonaikoja tai esimerkiksi ajastaa ohjelma käynnistymään tietyllä hetkellä.

3.5.1 Linux 2.6.18

Linuxin-ytimen versiosta 2.6.18 lähtien ajan ylläpitoon määriteltiin *clocksource* –rajapinta. Tämä on abstraktio kasvavasta laskurista. Tietokoneista löytyviä laskureita on käsitelty kappaleessa 2.3. Tämän rajanpinnan kautta pystytään lukemaan laskuria ja se sisältää tarvittavat kertoimet arvojen muuttamiseksi nanosekunneiksi.

Ytimessä on ajan säilyttämiseen yksi globaali *xtime* –muuttuja, joka sisältää nykyisen ajan sekunti- ja nanosekuntiosan. Normaalissa Linuxin ytimessä näiden muuttujien päivitys hoidetaan ajastinkeskeytyksen yhteydessä. Päivityksen ajaksi kello lukitaan silmukkalukolla, koska ajan kysymiseen tarkoitetut funktiot *getnstimeofday* ja *do_gettimeofday* käyttävät *xtime* –muuttujaa kellonajan muodostamiseen. Nämä funktiot määrittävät laskurirajapinnan, avulla paljonko aikaa on kulunut edellisestä *xtime* –muuttujan päivityksestä ja lisäävät tämän ajan *xtime* –muuttujan kopioon. Linuxissa joudutaan kuitenkin tekemään yksi jakolasku, jos aikaa halutaan kysyä mikrosekunnin tarkkuudella, koska aika on tallennettuna *xtime* –muuttujassa nanosekunnin tarkkuudella.

Käytettävää laskuria voidaan Linuxissa vaihtaa toiseen ilman ytimen uudelleen käynnistystä *sysfs*:n kautta. Laskuritoteutus on mahdollista tehdä moduulina, koska lista käytettävistä laskureista on dynaaminen.

3.5.2 FreeBSD 6.1

FreeBSD:n ytimessä laskurirajapinta on nimeltään *timecounter*. Rajapinta tarjoaa metodit laskurin lukemiseen ja laskurin taajuuden selvittämiseen samalla tavalla kuin Linuxin rajapinta. Kellonaika säilötään *bintime*-nimiseen tietueeseen, jossa sekunnit ovat 64-bittisessä muuttujassa ja sekunnin murto-osat toisessa 64-bittisessä muuttujassa.

Kellonpäivitys poikkeaa Linuxista siten, että FreeBSD:n ytimessä on rengaspuskuri *timehands*-tietueita, jotka sisältävät tarvittavat tiedot kellonajan määrittämiseen tietyllä hetkellä. Kellonpäivityksen aikana päivitetään rengaspuskurin seuraavaa alkiota. Tällöin toinen suoritin voi edelleen käyttää vanhaa rengaspuskurin arvoa ja määrittää kuluneen ajan sen avulla. Vasta kun kellonpäivitys on saatu suoritettua, lukitaan kello hetkeksi ja vaihdetaan ajan määrittämiseen käytettävän osoittimen arvo uuteen *timehands*-tietueeseen.

Kun aikaa kysytään ytimeltä, muodostetaan *timehands*-tietueen avulla laskurirajapintaa käyttäen nykyinen *bintime*. Tämä muunnetaan joko *timespec*- tai *timeval*-

tietueeseen sopivaksi. Muunnokseen kumpaankin muotoon ei tarvita kuin yksi kertolasku ja bittiensiirto.

Samalla tavalla kuin Linuxissa käytettävän laskurin voi vaihtaa *sysctl*:n kautta lennossa toiseen. Ytimen käynnistyksen aikana valitaan laskureiden hyvyysarvojen perusteella paras laskuri.

Luku 4

SynPCI-X –kortti

Luvussa käydään läpi mihin ajan ylläpito perustuu tietokoneissa. Luvussa esitellään myös ajan ylläpidon tarkkuuden parantamiseen suunnitellun SynPCI-X –kortin suunnitteluvaiheet ja sen toimintaperiaate.

4.1 Tietokoneen ajan ylläpito

Tietokoneissa yleisin tapa on johtaa kaikki tarvittavat kellotaajuudet kellopiirin avulla 14,31818 MHz kiteestä. Kellopiiri koostuu useammasta vaihelukitusta silmukasta joita käytetään generoimaan tarvittavia kellotaajuuksia. Lämpötila vaikuttaa kiteen resonanssitaajuuden suhteelliseen muutokseen neliöllisesti ja materiaalista riippuen kiteellä on tietty lämpötilakerroin. Kvartsikiteiden normaali lämpötilakerroin on $-0,035\,\frac{ppm}{^\circ C^2}$ [Vig92]. Lämpötilakertoimesta voidaan päätellä, että kiteen taajuus muuttuu suhteellisesti enemmän mitä kauempana se on kalibrointilämpötilasta. Kaavasta 4.1 saadaan laskettua kiteen suhteellinen taajuuspoikkeama tietyssä lämpötilassa. Kaavassa T_1 on lämpötila, jossa kide on kalibroitu. T_2 ja T_3 ovat lämpötilat joiden välillä lämpötila muuttuu. Kiteiden yleisin kalibrointilämpötila on 25 °C. Tietokoneiden koteloiden sisäinen lämpötila on normaalisti kuitenkin tätä korkeampi. Kotelon sisällä valitsevaan lämpötilaan vaikuttaa suorittimen ja oheislaitteiden kuormitus sekä kuinka hyvin kotelon jäähdytys on hoidettu. Esimerkiksi lämpötilanmuutos 32 asteesta 35 asteeseen aiheuttaa kiteen taajuuteen 1,785 ppm suhteellisen muutoksen. Kiteen kalibrointitaajuuden olleessa 10 MHz:iä tämä tarkoittaa 17,85 hertsiä.

$$p = -0,035\,\frac{ppm}{^\circ C^2}((T_1 - T_2)^2 - (T_1 - T_3)^2) \tag{4.1}$$

Kappaleessa 3.5 esiteltyjen käyttöjärjestelmien kellonpäivitysproseduurit käyttävät kappaleessa 2.3 esiteltyjä laskuripiirejä kuluneen ajan määrittämiseen. Kaikkien näiden laskurien taajuus kuitenkin on johdettu emolevyn lämpötila riippuvasta kiteestä.

Network Time Protocol (NTP) avulla on mahdollista yrittää määrittää kiteen taajuusvirhettä. Verkon yli tapahtuvassa synkronoinnissa oletetaan, että pakettien kulkuaikaviive on symmetrinen molempiin suuntiin. Tätä se ei käytännössä koskaan kuitenkaan ole reitillä olevien laitteiden jonotusviiveiden sekä synkronoitavien laitteiden käyttöjärjestelmien kuormituksen takia. Koska edellä mainitut asiat luovat runsaasti mittauskohinaa, taajuusvirheen estimoitiin joudutaan käyttämään useita näytteitä, jotta kohina saadaan minimoitua. Myös siinä tapauksessa, että PPS-signaali tuodaan sarjaportin kautta tietokoneelle, joudutaan käyttämään useampaa näytettä määrittämään taajuusvirhettä. Tämä johtuu siitä, että sarjaportin keskeytyksien käsittelyviive ei ole vakio.

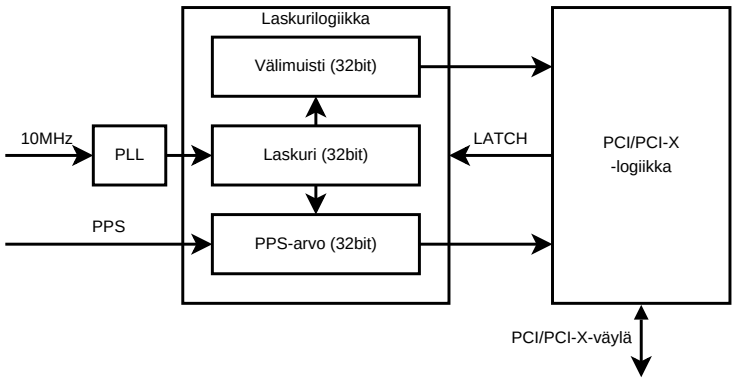
NTP:n avulla pystytään kohtuullisen hyvin määrittämään kiteen taajuusvirhe, jos tietokoneen ja verkon kuormitus on pieni. Suorittimen kuormituksen mukaan lämpötila kotelon sisällä voi kuitenkin muuttua jopa asteita minuuttien sisällä. Näin nopeisiin muutoksiin NTP-algoritmit eivät ehdi reagoimaan kohinanpoistosta johtuvan viiveen takia. Jotta ongelmasta päästäisiin kokonaan eroon, emolevyn kide pitäisi korvata lämpötilakompensoidulla kiteellä tai tuoda tarkka taajuus tietokoneen ulkopuolelta.

Mittalaitteiden synkronointiin voidaan käyttää GPS-vastaanotinta. Yleisin liitäntä mittalaitteiden ja GPS-vastaanottimen välillä on kaksi koaksiaalikaapelia, joissa toisessa kulkee yksi TTL-pulssi sekunnissa (PPS) ja toisessa 10 MHz:n ± 1 voltin siniaalto. Lisäksi RS-232 sarjakaapelissa tuodaan GPS-vastaanottimelta aikainformaatio sekunnin tarkkuudella. GPS-vastaanottimen tapauksessa tarkka 10 MHz ulostulo voidaan toteuttaa esimerkiksi uunikideoskillaattorilla (OCXO). Tällöin GPS-vastaanottimelta saatavalla ajastusinformaatiolla korjataan kiteen vanhenemisesta johtuva taajuusvirhe.

Ulkoisen uunikideoskillaattorin käyttämisellä tietokoneen kellon lähteenä saadaan poistettua lämpötilanvaihtelun vaikutukset. Lisäksi kalibroimalla kide GPS-vastaanottimen avulla saadaan tuotettua tarkka 10MHz:n taajuus.

4.2 Toimintaperiaate

SynPCI-kortin ideana on käyttää ulkoisia PPS- ja 10 MHz-signaaleja pitämään tietokoneen kello ajassa. Tätä varten tarvitaan laskurilogiikka, joka toimii 10 MHz:stä johdetulla tarkalla taajuudella. Tähän tarkoitukseen yksi vaihtoehto on käyttää kompleksista ohjelmoitavaa logiikkapiiriä (CPLD) laskurin toteuttamiseen. Kuva 4.1 on periaatepiirros SynPCI-kortin toimintaperiaatteesta.



Kuva 4.1: Synkronointikortin periaatepiirros

PLL:n avulla johdetaan 10 MHz:stä taajuus, jota käytetään ohjaamaan laskuria. Laskuri on yksinkertainen 32-bittinen ylöspäin juokseva laskuri, jonka arvo tarvittaessa siirretään väliaikaiseen rekisteriin, kun laskurin arvon lukeminen kortilta aloitetaan. Näin laskurin arvo ei muutu kesken lukemisen. Arvo siirretään myös talteen aina, kun PPS-pulssi saapuu. Tällöin voidaan tarkasti, määrittää millä laskurin arvolla sekunti on vaihtunut.

4.3 SynPCI-prototyyppi

Antti Gröhnin 2005 rakentamassa prototyypissä [Grö04] käytettiin kahta Latticen ispLSI5256VE CPLD-piiriä. Toisella piirillä toteutettiin laskurilogiikka ja toisella PCI-tilakone. Lisäksi prototyypissä on ulkoinen PLL-piiri, jolla sisääntuleva 10 MHz-signaali saadaan nostettua kymmenenkertaiseksi eli 100 MHz:iin. PPS- ja 10 MHz-signaalit tuodaan kortille optisten kuitujen kautta. Lähetys- ja vastaanottopäässä käytetään analogisia lähettimiä ja vastaanottimia, joten kortilla on kaksi nopeata komparaattoria, joilla signaalit rekonstruoidaan takaisin TTL-tasolle. Optisiin kuituihin päädyttiin sen takia, että koneiden välillä on joka tapauksessa tehtävä sähköinen erotus. Lisäksi paikoissa, joissa tehdään tietoverkkomittauksia, on mo-

nimuotokuituyhteyksiä yleensä helposti saatavilla. Signaalit saadaan vietyä pidempiäkin matkoja valmiiksi asennetuissa monimuotokuiduissa. Nykypäivänä koaksiaalikaapeleita löytyy hyvin harvoin samoista tiloista, joissa on tietoliikennelaitteita.

4.4 Kortin uusi versio

Prototyypissä käytettyjen CPLD-piirien ikääntyessä niiden hinta uudempiin piireihin verrattuna oli noussut moninkertaiseksi. Uudemmat piirit olivat myös suorituskyvyltään huomattavasti tehokkaampia. Kustannussyiden takia päädyttiin suunnittelemaan kortista uusi versio.

Uudessa kortissa päätettiin käyttää Latticen XO –sarjan LCMXO1200C3FT256C-piiriä. Piiri on välimalli CPLD- ja Field Programmable Gate Array (FPGA) –piireistä. Tämä piiri mahdollisti sekä laskurin että PCI/PCI-X –tilakoneen toteutuksen samalla piirillä. Piirin ongelmana kuitenkin on, että sen sisäinen PLL toimii ainoastaan, kun sisääntulevan signaalin taajuus on vähintään 25 MHz. Ongelma kierrettiin lisäämällä kortille erillinen Latticen PLL-piiri ispClock5610V, joka toimii myös 10 MHz sisääntulotaajuudella. Kyseinen piiri sisältää flash-muistin, joka on ohjelmoitavissa standardin Joint Test Action Group (JTAG) –liitännän kautta. Tämä mahdollistaa PLL:än kertoimien muuttamisen jälkikäteen ja näin ulostulotaajuus on vaihdettavissa.

4.4.1 Parannustavoitteet

Tavoitteena oli tehdä uudesta kortista PCI-X –yhteensopiva. PCI-X –tuen tarkoituksena on nostaa väylätaajuus yli 66 MHz:n ja näin pienentää kortin lukuviivettä. Lisäksi helpompaa vikojen määritystä varten kortille suunniteltiin kaksi kaksiväristä ohjelmoitavaa lediä. Näiden tarkoituksena on ilmaista, ovatko sisään tulevat signaalit kunnossa. Kortin ylälaitaan sijoitettiin paikka liittimelle, josta on voidaan tuoda signaaleja sisään tai ulos. Yhden CLPD-piirin toteutus myös mahdollisti helpomman piirin toiminnan simuloinnin.

Suunnittelun tavoitteena oli myös saada yksittäisen kortin kustannukset pienemmiksi kuin edellisessä prototyypissä. Jotta kortteja voisi valmistaa teollisesti useampia, korttiin piti lisätä kohdistusmerkinnät ja komponenttien sijoittelussa oli otettava huomioon riittävät toleranssit komponenttien ladontaa varten.

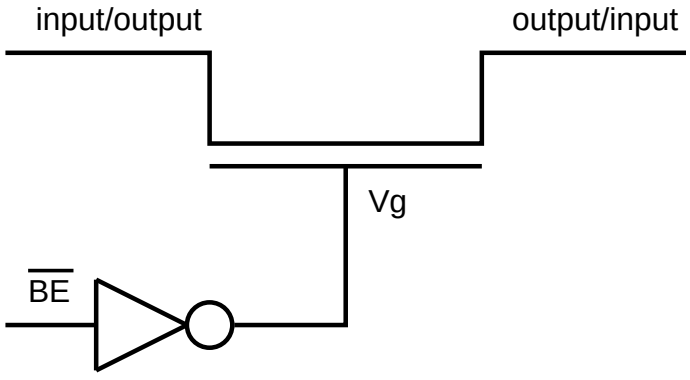
4.4.2 BGA-kanta

Kortissa käytettävän Lattice XO –piirin kanta on Ball Grid Array (BGA) –tyyppinen. Tämän takia kaksikerroksella piirilevyllä ei ole mahdollista käyttää läheskään kaikkia piirin jalkoja. Piirilevyjen kerroksia piti lisätä kahdesta vähintään neljään. Neljällä kerroksella saatiin käyttöjänniteille ja maalle omat piirilevykerrokset. SynPCI-X –kortin piirilevykuvat ovat liitteessä D.

BGA-piirien kiinnittäminen piirilevylle ilman reflow-uunia on lähes mahdoton toimenpide. Tästä syystä korttien valmistuksessa päädyttiin käyttämään komponenttien ladontapalvelua, jolloin kaikki komponentit kiinnitetään reflow-uunissa. Komponenttien ladontaa varten täytyy erikseen valmistaa metallista pastamaski, jonka hinta on komponenttien määrän muukaan satoja euroja. Koska tavoitteena oli saada yhden kortin hinta kohtuullisen halvaksi, niin kortteja piti tällä tavalla valmistaa useampia kappale kerrallaan. Valokuva lopullisesta SynPCI-X –kortista on liitteessä C.

4.4.3 Viiden voltin PCI-väylä

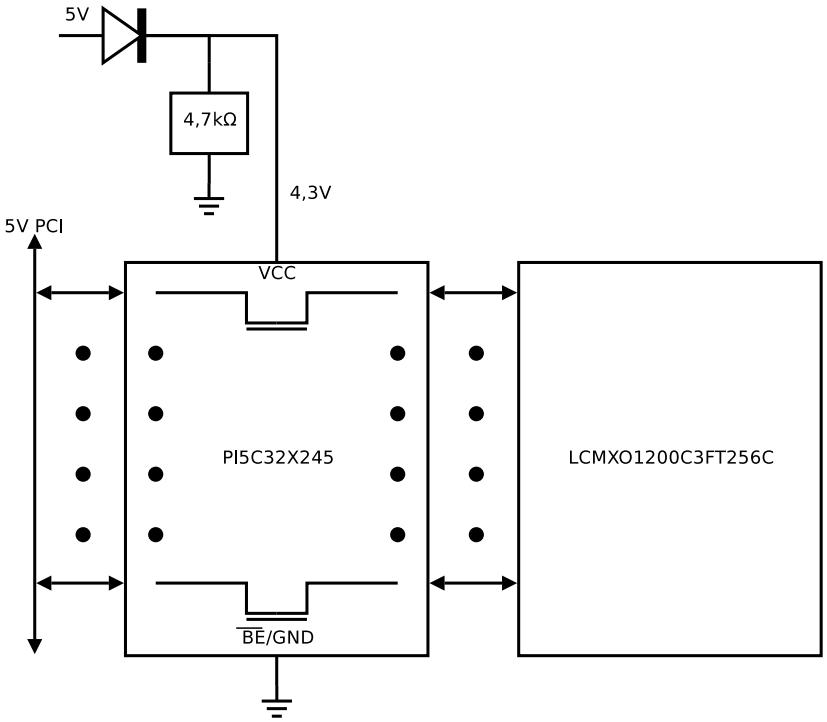
Lattice XO –piirissä ei ollut mahdollista käyttää enää 5 voltin TTL I/O -signalointia, joka on käytössä vanhassa PCI-väylässä. PCI-väylän ja Latticen piirin välillä oli tehtävä jännitteen muunnos 5 voltista 3,3 volttiin. Tähän tarkoitukseen käytettiin Pericomin PI5C32X245-piiriä. Piiri on 16-bittinen väyläkytkin, joka sisältää 16 MOS-transistoria, joiden kantaa ajaa looginen invertoiva CMOS-portti. Kuvassa 4.2 on periaatepiirros kytkimen toiminnasta.



Kuva 4.2: Väyläkytkimen periaate

Kytkin aktivoituu kun hila - lähde jännite V_g ylittää 1 voltin. Tämä jälkeen ulostulon jännite nousee sisääntulojännitteen funktiona. Ulostulon jännite kuitenkin rajoittuu

kynnysjännitteen verran alhaisemmaksi kuin käyttöjännite. Muunnos 5 voltista 3,3 volttiin onnistuu, kun väyläkytkimen käyttöjännitteeksi asetetaan kynnysjännitteen verran suurempi jännite eli 4,3 volttia.



Kuva 4.3: Jännitetasomuunnin

4,3 voltin jännite on helpointa muodostaa 5 voltista yhden diodin avulla. Diodissa tapahtuu kynnysjännitteen suuruinen jännitehäviö, joka on normaaleilla diodeilla 0,7 volttia. Käytetty kytkentä on esitetty kuvassa 4.3.

Kytkentä rajoittaa jännitteen 3,3 volttiin myös toiseen suuntaan. Tästä ei kuitenkaan synny ongelmaa, koska emolevyjen 5 voltin TTL-logiikan ykkösen tunnistamisen raja on 2 ja 3 voltin välissä sen mukaan minkälaisia komponentteja on käytetty emolevyllä.

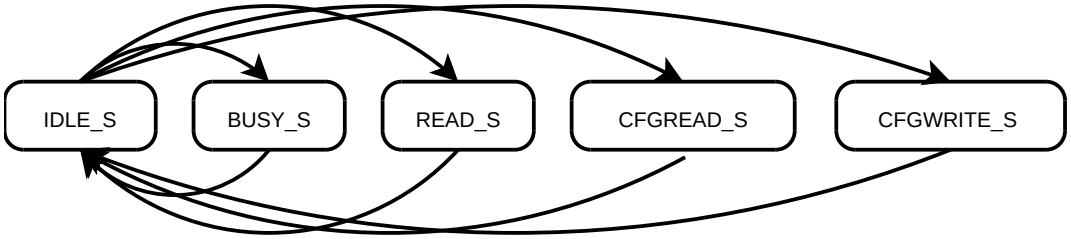
Viivettä PI5C32X245-piiri aiheuttaa ohjelehden [ref04g] mukaan maksimissaan 250 ps. Väylänopeuden ollessa 133 MHz kellojakso kestää 7500 ps. Näin ollen kytkimen aiheuttaman viiveen ei pitäisi tuottaa ongelmia.

4.4.4 VHDL

SynPCI-X –kortin CPLD-piirin PCI-tilakone ja laskuritoiminnallisuus on ohjelmoitu käyttäen VHDL-kieltä. VHDL-ohjelma koostuu neljästä eri osasta. Toteutuksessa on

kuvattu erikseen laskuri, ledien ohjaus ja kellotaajuuden puolittaminen. Pääohjelma toteuttaa PCI-tilakoneen ja käyttää hyväkseen muita ohjelmalohkoja. VHDL-ohjelman lähdekoodi sijaitsee kokonaisuudessaan liitteessä B.

PCI-tilakoneen eri tilat ovat esitettynä kuvassa 4.4. PCI-spesifikaation mukaan tilakone on IDLE-tilassa silloin, kun väylässä ei ole liikennettä. IDLE-tilassa on myös toteutettu muistiosoitteiden dekoodaus. Kun väylässä jokin laite aloittaa liikennöinnin, niin jokainen oheislaite tarkistaa, onko kohdeosoite kyseisellä kortilla. Jos kohdeosoite kuuluu kortin muistiavaruuteen ja CBE-arvo osoittaa lukuoperaatiota, siirrytään lukutilaan. PCI-tilakoneen tulee myös toteuttaa konfiguraation luku- ja kirjoitustilat, joiden avulla kortille määritellään sen muistialue ja muita parametreja. Jos mikään edellä olevista ehdoista ei toteudu, siirrytään BUSY-tilaan ja odotetaan, että liikennöinti väylässä loppuu.



Kuva 4.4: PCI/PCI-X –tilakone

Samalla kun siirrytään READ-tilaan, niin laskuriprosessille signaloidaan LATCH-signaalilla, että sen tulee siirtää tämän hetkinen laskurin arvo välimuistiin ja PCI-tilakoneen saataville kuten periaatepiirroksessa 4.1 on esitetty. LATCH-signaali nostetaan vain siinä tapauksessa, että korttia yritetään lukea ensimmäisestä muistiosoitteesta. Laskuriprosessilla on PCI-tilassa yksi kellojakso aikaa siirtää laskurin arvo PCI-tilakoneprosessin saataville ja PCI-X –tilassa kaksi kellojaksoa. Kortti palauttaa laskurin arvon, milloin PPS-pulssi on viimeksi saapunut, kun sitä luetaan toisesta muistiosoitteesta. Kolmas muistiosoite palauttaa virhelaskurit. Virhelaskurin 8 alinta bittiä kertovat, kuinka monta kertaa PPS-pulssi on jäänyt saapumatta. Laskuri kasvaa, jos PPS-pulssi ei saavu 10,016 μs sisällä, kun edellisestä pulssista on kulunut sekunti. Bitit 8-15 kertovat kuinka monta kertaa PPS-pulssi on saapunut välillä $(PPS + 50\,\mu\text{s}) - (PPS + 1\,\text{s} - 50\,\mu\text{s})$. Laskureiden avulla on mahdollista selvittää, aiheuttaako jokin häiriöitä PPS- tai 10 MHz-signaaleihin niin, että PPS-pulssi ei saavu tasan yhden sekunnin välein.

Laskuriprosessi toimii yksinkertaisena 29-bittisenä laskurina, jonka arvoa kasvatetaan joka kellojaksolla yhdellä. Pitämällä 32-bittisen luvun kolme alinta bittiä nolli-

na ja kasvattamalla vain ylempiä bittejä laskurin arvo kasvaa aina kahdeksalla. Kun PLL:ltä tulevan kellosignaalin taajuus on 125 MHz, kellojakson pituus on tarkalleen 8 ns. Tällöin laskuri kertoo suoraan kuluneet nanosekunnit. Laskuriprosessi tallentaa laskurin arvon aina PPS-pulssin saapuessa ja päivittää PCI-tilakoneen saatavissa olevaa arvoa.

LED-valojen ohjaus on toteutettu omana prosessinaan. Prosessi päivittää samalla aikaisemmin mainittuja virhelaskureita. Kortilla olevat kaksi lediä palavat vihreinä, jos PPS- ja 10 MHz-signaalit ovat kunnossa. Jos PLL-piiri saa lukittua itsensä 10 MHz-signaaliin, sen todetaan olevan kunnossa. PPS-signaali on kunnossa, jos pulssi saapuu 10 MHz:n avulla mitattuna sekunnin välein.

4.5 Havaitut ongelmat

4.5.1 133 MHz PCI-X –väylä

Latticen piirin VHDL-syntesisointiohjelmalla todettiin, että PCI-tilakoneen maksimisuoritusnopeus on vain hieman yli 100 MHz:iä. Edes koodia muuttamalla ei tilakoneen kaikkia osia saatu toimimaan täydellä 133 MHz:n nopeudella. Tämän takia kortti ei toimi kunnolla kaikissa 133 MHz-väylissä. Kortti saattaa kaataa joidenkin koneiden 133 MHz:n PCI-X –väylän täysin. Ongelma on kierrettävissä rajoittamalla väylänopeus 100 MHz:iin. Jos tämä ei onnistu ohjelmallisesti tai emolevyn hyppyjohtimilla, väylään kytkettäessä toinen 133 MHz:n PCI-X –kortti väylännopeus tippuu automaattisesti PCI-X –spesifikaation mukaan 100 MHz:iin.

4.5.2 Vanhat PCI-väylät

Valmiissa korteissa huomattiin, että ne eivät toiminetkaan vanhoissa PCI-väylissä. Viaksi todettiin, että vanhimmat PCI-väylät eivät tarjoa ollenkaan 3,3 voltin käyttöjännitettä. Tällöin Latticen XO -piiri jää kokonaan ilman käyttöjännitettä. Tämän ongelman voisi kiertää generoimalla tarvittavan 3,3 voltin jännitteen regulaattorilla 5 voltin käyttöjännitteestä. Uudemmat emolevyt kuitenkin syöttävät myös 3,3 voltin käyttöjännitteen, vaikka signalointi tapahtuukin 5 voltilla.

4.5.3 Kahden kellosignaalin käyttö

Aiemmassa prototyypissä huomattiin, että jos laskuri nollattiin jokaisen arvon lukemisen jälkeen, sekunnin aikana ei saatu tarkalleen miljardia nanosekuntia sum-

mattaessa luetut arvot yhteen. Kun laskuri jätettiin nollaamatta lukemisten välissä ja käyttettiin edellistä luettua arvoa kuluneen ajan määrittämiseen, saatiin kahden PPS-pulssin väliseksi ajaksi tarkalleen miljardi nanosekuntia. Tässä vaiheessa oletettiin kortin toimivan täysin oikein.

Ongelmaksi kuitenkin muodostuu se, ettei kortilta luettava arvo ole aina oikea. Tämä huomattiin, kun korttia luettiin silmukassa monta kertaa peräkkäin. Tällöin välillä kahden peräkkäisen arvon erotus oli negatiivinen. Negatiiviset arvot eivät kuitenkaan johtuneet laskurin ylivuodosta vaan siitä, että jokin bitti oli luetussa arvossa väärin.

Ongelmana on mitä luultavammin kahden eri kellosignaalin käyttö CLPD-piirin sisällä. PCI-tilakone käy PCI-väylästä saatavan kellosignaalin mukaan ja laskuriosuus toimii ulkoisen tarkan kellosignaalin avulla. Näiden kahden prosessin välillä tiedonsiirto ei ole aivan triviaalia, koska kellosignaalit eivät ole synkronoituja keskenään. ASIC-piireihin on mahdollista suunnitella erillinen puskurointiosuus, millä tiedonsiirto kahden kellon välillä on mahdollista tehdä luotettavasti. Kirjassa [WJD98] on tähän esitetty lukuisia toteutustapoja, joiden avulla tämä on mahdollista.

Vähiten virheitä näyttää tapahtuvan, jos laskurin arvo siirretään kahden prosessin välillä asynkronisesti eli arvo siirretään prosessien välillä silloin, kun laskuripiirin kellosignaalin arvo on nolla ja kun PCI-tilakoneen puolelta on nostettu signaali. Tällöinkin on kuitenkin mahdollista, että PCI-tilakoneen puolelta nostetaan signaalin arvo niin lähellä laskuripiirin nousevaa kellojakson reunaa, ettei arvoa saada talteen ennen kuin sen päivitys alkaa.

4.6 Linux-laiteajuri

Kortin Linux-laiteajuri toteuttaa kappaleessa 3.5 esitellyn *clocksource* –rajapinnan. Ajurin alustuksessa kortin fyysiselle muistialueelle tehdään muistikartoitus ytimen virtuaaliseen muistiavaruuteen. Tämän jälkeen on mahdollista lukea kortin muistialuetta aivan kuten normaalia muistia. Alustusvaiheessa myös määritetään kortin tarkan laskurin avulla suorittimen sen hetkinen kellotaajuus.

Kun *clocksource* –rajapinnan callback-funktiota kutsutaan, käytetään kortin laskuria ja PPS-arvoa määrittämään tämän hetkinen sekunnin nanosekuntiosa. Tämä arvo laitetaan *xtime* –muuttujan nanosekuntiosaan. Sekuntiosaa kasvatetaan, jos PPS-laskurin arvo on muuttunut edellisen päivityskerran jälkeen. Nanosekuntiosa määritetään myös TSC-laskurin avulla. TSC-laskurin taajuutta korjataan niin, että seuraavalla päivityskerralla TSC-laskurin avulla määritetty nanosekuntiosa täsmäisi

SynPCI-X –kortin avulla määritettyyn nanosekuntiosaan.

Ajuri sisältää myös tarkistuksia sille, luetaanko korttia väärin, ja tarvittaessa korttia luetaan uudestaan. Korttia ei kuitenkaan kannata lukea liian montaa kertaa uudestaan, koska tällöin kellonpäivitykseen kuluva aika ei ole enää determistinen. Ajurissa on päädytty hyväksymään alle mikrosekunnin virheet laskurin arvoissa. Jos virhe on kolmannen lukukerran jälkeen edelleen suurempi, ei enää yritetä uudelleen. Kellonaika kuitenkin korjautuu oikeaksi seuraavalla päivityskerralla.

Ajuri palauttaa kuluneen ajan käyttäen TSC-laskuria ja viimeisellä päivityskerralla määritettyä nanosekuntiosaa. Tähän päädyttiin sen takia, että TSC-laskuria on huomattavasti nopeampi lukea ja sen arvo on joka kerralla oikea. SynPCI-X –kortilta arvoa luettaessa joudutaan aina tekemään tarkistukset luetulle arvolle. Mittaukset laskureiden lukuviiveistä löytyvät kappaleessa 5.2. Koska TSC-laskurin taajuus muuttuu ajan funktiona, taajuutta on kalibroitava jatkuvasti. Ajurin lähdekoodi löytyy liitteestä A.

Luku 5

Mittaukset

Suoritetuissa mittauksissa oli tarkoitus selvitää kuinka tarkasti kaapattujen pakettien aikaleimat pitävät paikkansa ja kuinka paljon aikaleimoissa on maksimissaan virhettä. Pakettien aikaleimojen virheiden arvioimista varten kehittiin mittauksia varten erillinen mittausmenetelmä.

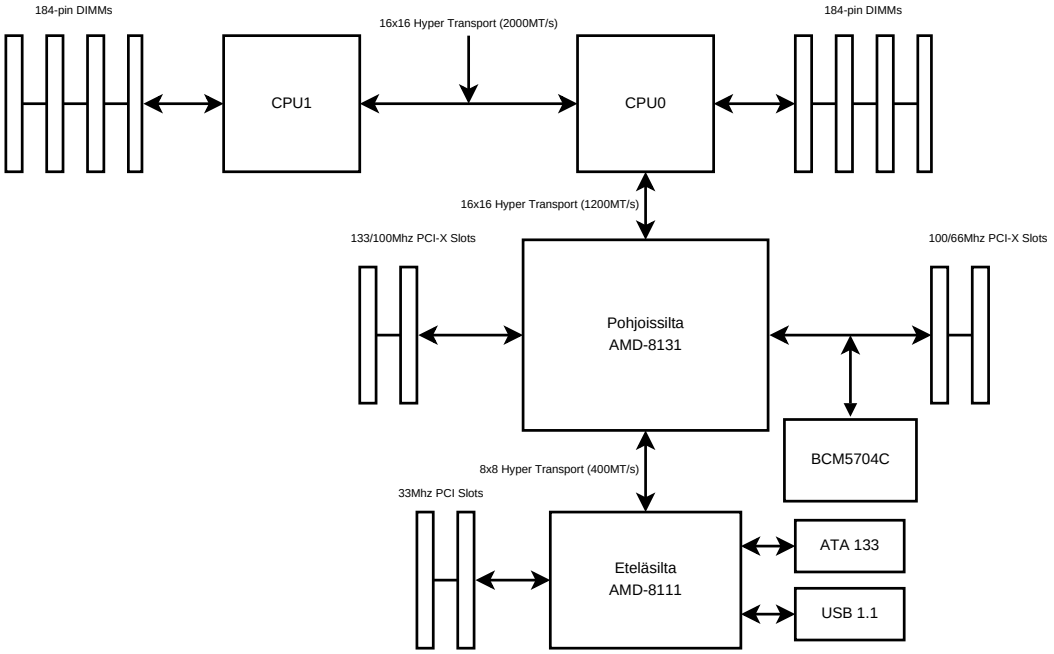
5.1 Mittauslaitteistot

5.1.1 Supermicro Dual Opteron

Opteron-koneen kaksi suoritinta käyvät 2,2 GHz taajuudella. Kuvassa 5.1 on emolevyn piirisarjat ja niiden liitokset toisiinsa. Emolevyllä käytetty pääväylä on Hyper Transport, joka yhdistää suorittimet sekä pohjois- ja eteläsillan. Muistit on koneessa sijoitettu ensimmäisen suorittimen muistiväylään. Muisteina koneessa on kaksi 512 MB 400 MHz DDR ECC muistikampaa.

Pohjoissiltana toimii AMD 8131 [ref04b] piirisarja, joka on silta Hyper Transport ja PCI-X –väylän välillä. Piirisarja tarjoaa kaksi erillistä PCI-X –väylää. Toiseen väylistä on liitettynä emolevylle integroidut gigabittiset verkkokortit. Verkkokorttien piirisarjana toimii kappaleessa 2.4.5 esitelty BCM5704C. Hyper Transport –väylä pohjoissillan ja ensimmäisen suorittimen välillä toimii 600 MHz:llä ja signalointinopeus on 1200 MT/s. Koska väylässä siirretään 16-bittiä jokaisella tapahtumalla, väylänopeus on kumpaankin suuntaan 2400 MB/s.

Eteläsilta AMD 8111 [ref04a] tarjoaa loput emolevyllä tarvittavista liitännöistä kuten levy- ja USB-liitännät. Piirisarja on yhdistetty 8-bittisellä 200 MHz:n Hyper Transport –väylällä pohjoissiltaan. Väylän siirtonopeus molempiin suuntiin on 400 MB/s.



Kuva 5.1: Opteron-emolevyn periaatepiirros

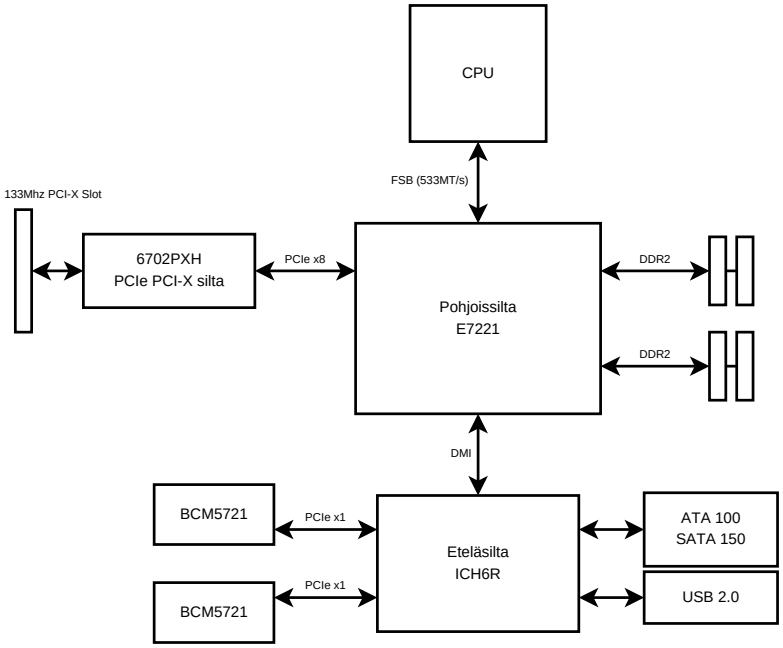
5.1.2 Supermicro Celeron P8SC8

Mittauksissa käytettyjen kahden Intelin Celeron-pohjaisten Supermicro-koneiden suorittimien kellotaajuus on 2,66 GHz. Kuvaan 5.2 on piirretty emolevyjen piirisarjat, tärkeimmät liitännät ja niiden väliset väylät.

Intelin arkkitehtuurissa muistiohjain sijaitsee pohjoissillassa. Emolevyllä käytetyssä E7221 [ref04f] -piirisarjassa on kaksikanavainen DDR2-muistiohjain. Koneiden molemmat muistiväylät on varustettu yhdellä 512 MB:n 533 MHz:n DDR2-muistikammalla. Intel käyttää omaa Front Side Bus (FSB) -väylää suorittimen yhdistämiseen pohjoissiltaan. Väylä koostuu omasta väylästä osoitteille, kontrollille ja datalle. Dataväylän leveys on 64 bittiä ja signalointinopeuden ollessa Celeronin tapauksessa 533 MT/s väylän siirtonopeus on yhteen suuntaan 4264 MB/s.

6702PXH [ref04e] -piiri toimii siltana PCIe ja PCI-X –väylän välillä. Emolevyllä on yksi 64-bittinen 133 MHz:n PCI-X –liitäntä, joka on tämän piirin välityksellä kiinni pohjoissillan 8x PCIe-väylässä.

Eteläsiltana emolevyllä on käytetty ICH6R [ref05b] -piirisarjaa, joka on liitetty pohjoissiltaan Direct Media Interface:n (DMI) avulla. DMI-väylä on kaksisuuntainen ja sen nopeus on 1 GB/s yhteen suuntaan. Piirisarja sisältää neljä 1x PCIe-väylää, joista kahteen on kytketty emolevylle integroidut BCM5721-verkkokortit. Piiristä löytyy



Kuva 5.2: Celeron emolevyn periaatepiirros

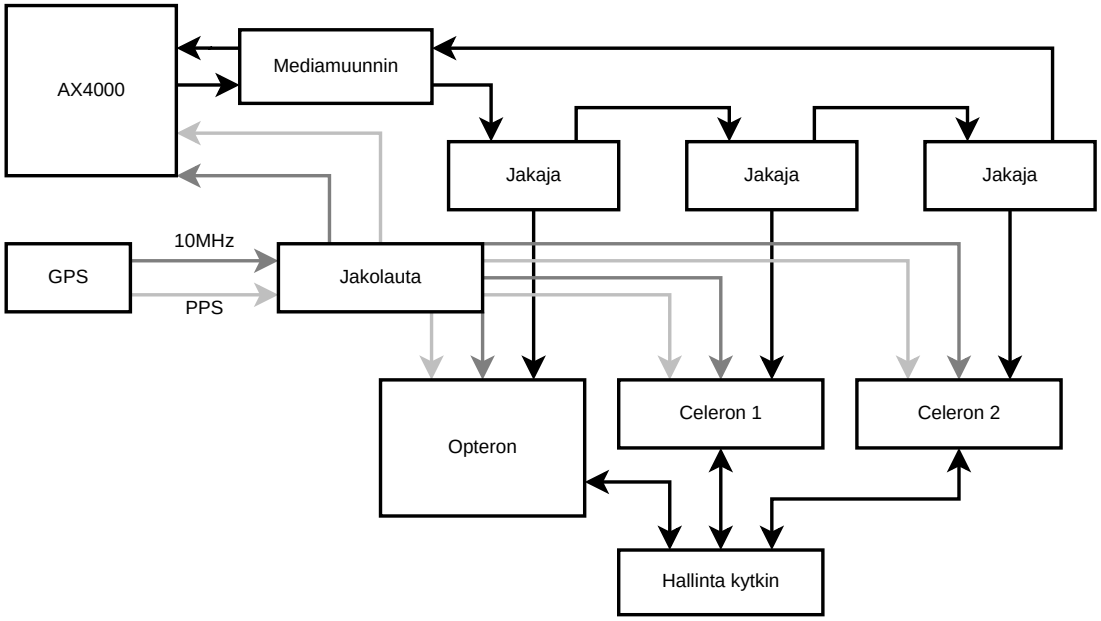
myös normaali PCI-väylä, levyliitännät ja muu normaalisti tietokoneen toimintaan tarvittava logiikka.

5.1.3 Mittausjärjestelyt

Mittauslaitteistoon kuuluu edellä esiteltyjen kolmen koneen lisäksi Spirentin AX/4000-mittalaite, jolla voidaan generoida paketteja ja analysoida niitä. Mittauksissa käytetty GPS-laite on Trimblen valmistama Thunderbolt GPS Disciplined Clock. Valmistajan mukaan 10 MHz ulostulon tarkkuus on $1,16 \cdot 10^{-12}$ yhden päivän yli keskiarvoistettuna. Tämä tarkoittaa sitä, että taajuus poikkeaa keskimäärin ilmoitetusta 10 MHz:stä $1,16 \cdot 10^{-5}$ hertsiä. PPS-signaali poikkeaa valmistajan mukaan UTC-ajasta maksimissaan 20 ns.

10 MHz- ja PPS-signaalit jaetaan koneille ja AX/4000-mittalaitteelle jakolaudan avulla samasta GPS-vastaanottimesta. Jokaisessa koneessa on SynPCI-X –kortti, johon signaalit tuodaan. Kuva 5.3 esittää mittausjärjestelyä ja kuinka kaikki laitteet on kytketty toisiinsa. Koska AX/4000-mittalaitteessa ei ole gigabittisiä kupari Ethernet –liitäntäkortteja, täytyy paketeille tehdä mediamuunnos optisesta sähköiseen muotoon. Paketit kulkevat ensin mediamuuntimeen läpi ja sen jälkeen kiertävät kolmen 10/100/1000-kuparijakajan kautta ja saapuvat mediamuuntimen kautta takaisin

analysointipuolelle. Jakaja nimensä mukaisesti jakaa saman paketin useammalle linkille. Tässä tapauksessa sama paketti tulee ulos saman aikaisesti jakajan kahdesta eri portista. Jokaisen jakajan monitorointiportti on kytkettynä koneiden toiseen verkkokorttiin. Koneiden toisissa verkkokorteissa on kiinni 10/100-hallintakytkin, jonka kautta suoritetaan NTP-synkronointi.



Kuva 5.3: Mittausjärjestelyt

Koska gigabittisellä Ethernet-linkillä ei ole mahdollista tehdä passiivista pakettien kuuntelemista, jokainen paketti joudutaan vastaanottamaan ja lähettämään uudestaan. Tästä syystä jokainen jakaja aiheuttaa vakioviiveen paketeille. Lisäämällä jakajia ketjuun saatiin mitattua yhden jakajan viiveeksi n. 300-320 ns. AX/4000-mittalaitteen aikaleimauksen resoluution ollessa 10 ns viivettä ei saada mitattua tarkemmin. Paketit tulivat mittausten perusteella yhden aikaisesti ulos jakajien sekä monitori- että linkkiportista.

Generoimalla yli 10 miljoonaa pakettia koko jakajaketjun läpi saatiin keskimääräiseksi viiveeksi 2,026 μs. Läpimenoviiveen oli minimissään 1,93 μs ja maksimissaan 2,10 μs. Tämä 170 ns hajonta kiertoaikaviiveessä johtuu siitä, että AX/4000-mittalaite noudattaa Ethernet-protokollaa ja lähettää paketit sitä noudattaen. Kun lähetyspuoli kytkettiin suoraan lyhyellä kuidulla vastaanottopuoleen, mittalaitteen pakettien generoimisen ja vastaanottamisen aiheuttamaksi viiveeksi saatiin keskimäärin 593 ns. Kun tähän lisätään kahden mediamuuntimen aiheuttama viive, kokonaisviiveeksi saadaan 1109 ns. Lisättäessä edelliseen viiveeseen kolmen jakajan ai-

heuttama viive päästään lähelle mitattua kokonaisviivettä.

Jos oletetaan AX/4000-mittalaiteen aiheuttavan yhtä paljon viivettä sekä lähetys- että vastaanottopäässä, tehdään maksimissaan 300 ns virhearviointi. Tällä oletuksella ensimmäisen jakajan kohdalla viivettä on n. 860 ns, toisen n. 1170 ns ja kolmannen 1480 ns. Nämä viiveet voidaan ottaa huomioon tuloksia analysoitaessa.

5.2 Laskureiden lukuviive

Kuten kappaleessa 3.5 esiteltiin, kellonajan määrittämiseen käytetään jotakin tietokoneesta löytyvää laskuripiiriä. Laskuripiiri on TSC-laskuria lukuun ottamatta integroituna johonkin oheislaitteeseen tai piirisarjaan. Tämän takia sen lukuviive on huomattavasti suurempi kuin esimerkiksi suorittimen yhdessä kellojaksossa suorittama laskutoimitus. Tästä syystä päätettiin mitata, kuinka paljon eri laskureiden lukemisessa on viivettä.

	Lukuviive (ns)			
Laskuri	Opteron CPU0	Opteron CPU1	CPU1-CPU0	Celeron
PIT	4115-4120	4230-4235	110-120	3993-3996
HPET	1460-1461	1522-1527	61-67	537-545
PM	1027-1028	1087-1088	59-61	1126-1127
TSC	6-7	6-7	0	37-38

Taulukko 5.1: Laskureiden lukuviive

Mittaukset suoritettiin lukemalla laskureita 100000 kertaa peräkkäin ja laskemalla kulutetusta ajasta keskimääräinen lukuviive. Taulukossa 5.1 on mittaustulokset kaikista kappaleessa 2.3 esitellyistä laskurien lukuviiveistä. Opteron-koneessa lukuviiveessä on eroja sen mukaan, kummalla suorittimella laskuria luetaan. Tämä johtuu siitä, ettei toinen suoritin ole suoraan yhteydessä pohjoissiltaan ja sitä kautta eteläsiltaan vaan lukupyynnöt kiertävät toisen suorittimen kautta kuten kuvasta 5.1 käy ilmi. Lisäviivettä syntyy toisen suorittimen edestakaisesta läpäisystä noin 60 ns. PIT-laskurin tapauksessa lisäviive on kaksinkertainen, mutta se johtuu siitä, että 16-bittistä laskuria voidaan lukea vain kahdeksan bittiä kerrallaan. Muut laskurit voidaan lukea kokonaan yhdellä kerralla. TSC-laskurin lukuviive on pienin aivan oletetusti, koska se sijaitsee suorittimessa.

Taulukossa 5.2 on mittaustulokset SynPCI-X –kortin lukuviiveestä. Mittaukset suoritettiin jokaiselle väylänopeudelle erikseen. Lukuviive pienenee oletetusti, kun väy-

Väylä	Lukuviive (ns)			
	Opteron CPU0	Opteron CPU1	CPU1-CPU0	Celeron
PCI 33 MHz	605-606	664-665	58-60	715-721
PCI 66 MHz	438-439	490-492	51-54	566-573
PCI-X 66 MHz	391-393	443-444	50-53	604-610
PCI-X 100 MHz	347-349	404-405	55-58	545-551
PCI-X 133 MHz	309-312	362-364	50-55	514-521

Taulukko 5.2: SynPCI-X –kortin lukuviive

länopeus kasvaa. Väylänopeuden kaksinkertaistaminen ei kuitenkaan puolita lukuviivettä, koska lukuviive ei koostu pelkästään käytetyn väylän aiheuttamasta viiveestä. Viiveeseen vaikuttavat myös emolevyllä käytetyt piirisarjat ja niiden väliset väylät. Tässäkin mittauksessa näkyy Opteron-koneen eri suorittimien väliset erot, jotka johtuvat samasta syystä kuin aikaisemmassa mittauksessa.

Opteron-koneessa SynPCI-X –kortin lukuviive on selvästi seuraavaksi lyhin TSC-laskurin jälkeen. Celeron-koneessa SynPCI-X –korttia on nopeimmalla väylällä vain hiukan nopeampaa lukea kuin HPET-laskuria. Joka tapauksessa nopein vaihtoehto on käyttää TSC-laskuria ajan määrittämiseen ja kalibroida TSC-laskuria jonkin muun tarkemman laskurin avulla. TSC- ja HPET-laskurin taajuus generoidaan emolevyllä sijaitsevasta kiteestä, ja näin kumpikin laskuri on altis lämpötilanmuutoksille, HPET-laskurin taajuus ei kuitenkaan muutu koneen ollessa virransäästötilassa.

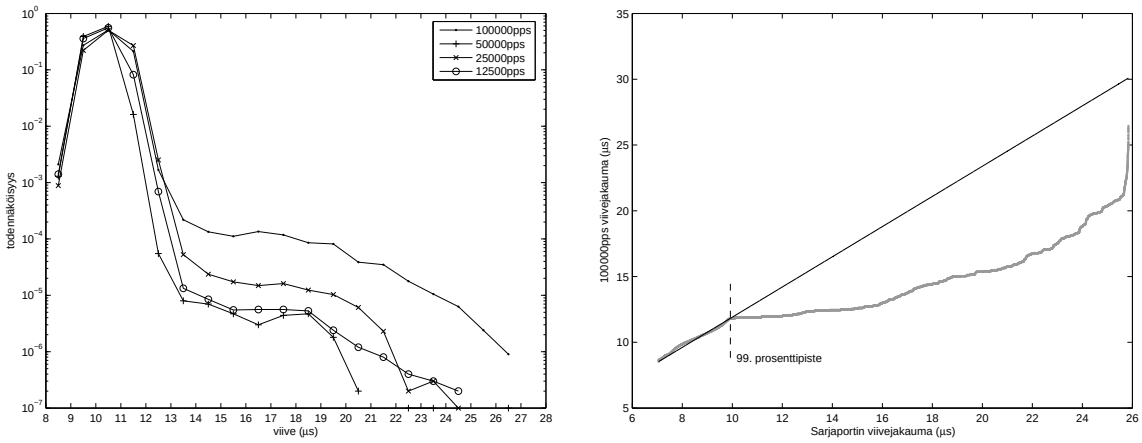
5.3 Kulkuaikaviive

Koska AX/4000-mittalaite on mahdollista synkronoida GPS-vastaanottimen avulla, niin sen generoimiin testipaketteihin tulee tarkka yksiselitteinen aikaleima. Jakajien avulla saadaan sama paketti monistettua jokaisen koneen verkkokortille lähes samanaikasesti. Kun vähennetään kaappaustiedoston aikaleimasta AX/4000-mittalaitteen generoima aikaleima, saadaan laskettua, kuinka kauan paketilla kesti saapua AX/4000-mittalaitteelta ytimen verkkopinoon, jossa se aikaleimataan. Kuinka paketit käsitellään käyttöjärjestelmissä on selitetty tarkemmin kappaleessa 3.4.

Mitattu kulkuaikaviive koostuu paketin kulkuajasta mittalaitteelta verkkokortille. Kuten aikaisemmin mitattiin tämän vaikutus tuloksiin on yhden mikrosekunnin luokkaa ja se on lähes vakio. Lisäksi kulkuaikaviiveeseen vaikuttaa verkkokortin käsittelyviive, joka koostuu paketin vastaanottamisesta ja siirtämisesta tietokoneen keskus-

muistiin. Tämän lisäksi viiveeseen vaikuttaa tietokoneen keskeytysviive ja ytimen yhden paketin käsittelyviive ennenkuin se aikaleimataan.

5.3.1 Opteron SynPCI-X



(a) Histogrammi

(b) P-P kuvaaja

Kuva 5.4: Opteron-kone SynPCI-X synkronoituna

Kuvassa 5.4 (a) on kulkuaikaviivejakauma Dual Opteron-koneen tapauksessa eri pakettinopeuksilla. Koneessa käytetty ydin on muuten normaali paitsi, että siihen on lisätty PF_RING ja SynPCI-X –tuki. Lisäksi verkkokortin keskeytykset oli mittauksen aikana siirretty omalle suorittimelleen, jossa ei ollut muita prosesseja pakettien kaappaukseen käytetyn *tcpdump*:n lisäksi suorituksessa. Broadcomin verkkokortin keskeytysviive oli säädetty *Ethtool*:a käyttäen minimiin eli yhteen mikrosekuntiin.

Taulukossa 5.3 on esitetty mittausdata lukuina. Histogrammin jakovälin koko on yksi mikrosekunti ja y-akseli on piirretty logaritmisena, jotta pienet erot saadaan näkyviin. Kymmenestä miljoonasta paketista 99 prosenttia muodostaa lähes identtisen histogrammin. Erot tulevat esiin vasta yhden prosentin muodostamassa hännässä. Tässä näkyvät erot voidaan ainakin osittain selittää sillä, että pakettinopeuden kasvaessa suorittimen kuormitus kasvaa ja suurempi osa paketeista joutuu odottamaan käsittelyä hieman pidempään. Tämä johtuu suoraan siitä, että keskeytyksiä hoitavalla suorittimella on myös ajossa pakettien kaappausta hoitava sovellus. Pakettien siirtäminen kiintolevylle on yksi suurimmista pullonkauloista pakettinopeuden kasvaessa. Mittauksissa käytetyillä pakettinopeuksilla verkkokortin käyttämän väylän kaistan käyttöaste jää kuitenkin niin pieneksi, ettei sen pitäisi aiheuttaa jakauman

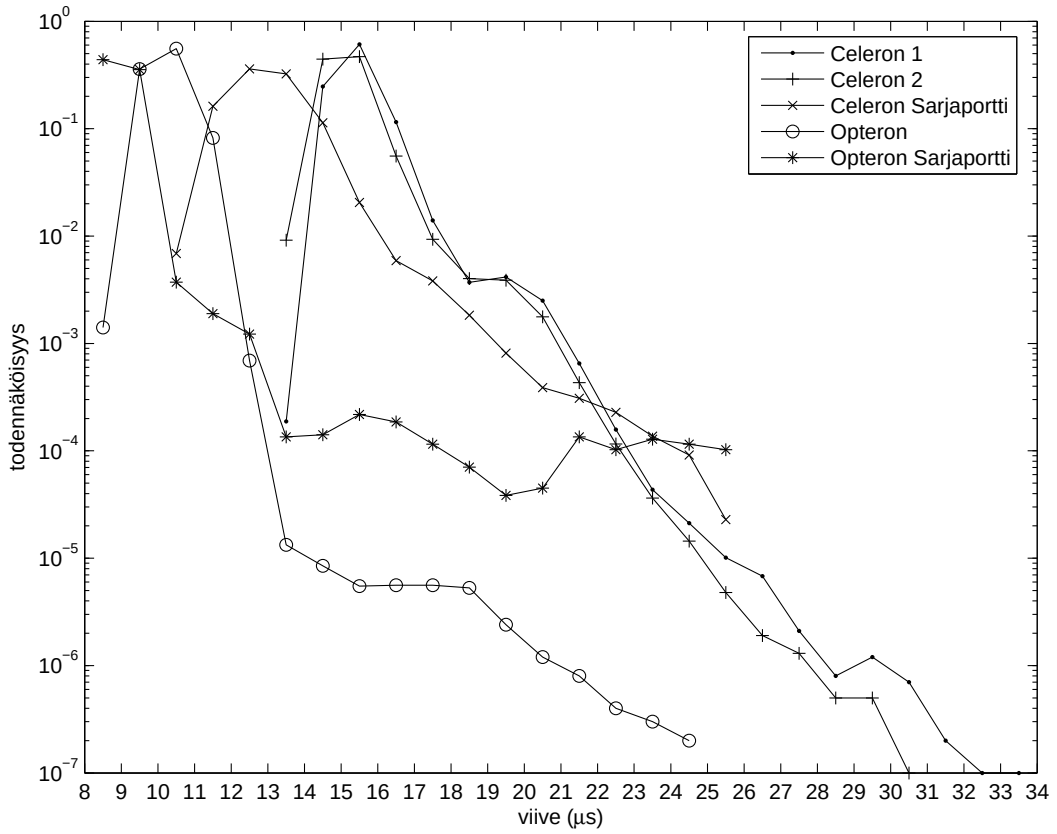
muuttumista.

	SynPCI					Sarjaportti
	100000 pps	50000 pps	25000 pps	12500 pps	6250 pps	
Näytteitä	10^7	10^7	10^7	10^7	10^7	156009
Keskiarvo	10,46	10,13	10,56	10,23	10,30	8,72
Varianssi	0,46	0,19	0,42	0,29	0,27	0,66
Vinous	1,26	0,19	0,14	0,54	0,52	4,61
Huipukkuus	18,78	4,85	3,89	4,59	4,40	78,17
Minimi	8,53	8,57	8,61	8,61	8,70	7,06
25. PP	9,99	9,82	10,07	9,86	9,95	8,12
50. PP	10,45	10,11	10,57	10,20	10,28	8,69
75. PP	10,95	10,45	11,07	10,57	10,61	9,26
99. PP	11,82	11,07	11,86	11,65	11,74	9,92
Maksimi	26,45	26,24	24,20	24,86	23,99	25,84

Taulukko 5.3: Opteron SynPCI-X kulkuaikaviive (μs)

Eräs syy pakettien viivejakauman hännän muodostumiselle on keskeytysviive. Tämän takia keskeytysviivettä mitattiin myös sarjaportin kautta. GPS-laitteen PPS-signaali tuotiin sarjaportin kautta tietokoneelle ja ydin tulosti ajan, jolloin sarjanportin keskeytys käsiteltiin. Myös tässä mittauksessa oli sarjaportin keskeytys siirretty omalle suorittimelle, jossa ei ollut muita prosesseja ajossa.

Taulukon 5.3 viimeisessä sarakkeessa on esitetty sarjaportin keskeytysviiveen mittaustulokset. Kuvassa 5.4 (b) on piirretty P-P -kuvaaja Opteron-koneen sarjaportin keskeytysviivejakaumasta ja 100000 pps mittauksen kulkuaikaviivejakaumasta. Taulukon arvoista nähdään, että 99 prosenttia paketeista muodostaa lähes samanlaisen jakauman. Sarjaportin viive näyttää kuitenkin olevan 1,7 μs pienempi. P-P -kuvaajasta nähdään, että asia on todellakin näin. Paketeista 99 prosenttia osuu viivalle, joka on piirretty minimin ja 99:nen prosenttipisteen kautta. Kuvassa 5.5 on piirretty samaan kuvaan sarjaportin ja kulkuaikaviiveen histogrammit. Niissäkin muoto on samanlainen lukuun ottamatta histogrammin häntää. Kuvassakin näkyvä 1,7 μs ero voidaan selittää sillä, että pakettien kulkuaikaviive AX/4000-mittalaitteelta verkkokortille on mikrosekunnin luokkaa kuten kappaleessa 5.1.3 oli mitattu. Lisäksi verkkokortti aiheuttaa viivettä paketin vastaanottamisessa ja siirtämisessä keskusmuistiin.



Kuva 5.5: Sarjaportin ja verkkokortin viivejakauman vertailu

5.3.2 Celeron SynPCI-X

Aikaisemmin esitellyt Celeron-koneet ovat komponenteiltaan identtiset. Kummassakin koneessa on reaaliaika Linux-ydin ja kaappausta hoitavan verkkokortin keskeytys prioriteetti on asetettu korkeimmaksi. Muuten asetukset ovat samat kuin Opteron-koneessa. Keskeytyksiä Celeron-koneissa ei tietenkään voitu siirtää omalle suorittimelleen, koska koneissa on vain yksi suoritin.

Kuten Opteron-koneen tapauksessa huomattiin, viivejakauma muuttuu vain hieman, kun pakettinopeus laskee tarpeeksi. Tämän takia Celeron-koneissa ei tarkastella hitaampia pakettinopeuksia kuin 12500 pps. Taulukkoon 5.4 on laskettu tunnusluvut neljästä eri mittauksesta kummaltakin Celeron-koneelta. Lisäksi keskeytysviivettä mitattiin sarjaportin kautta samalla tavalla kuin Opteron-koneen tapauksessa. Mittausdatasta piirretyt histogrammit löytyvät kuvasta 5.6 (a).

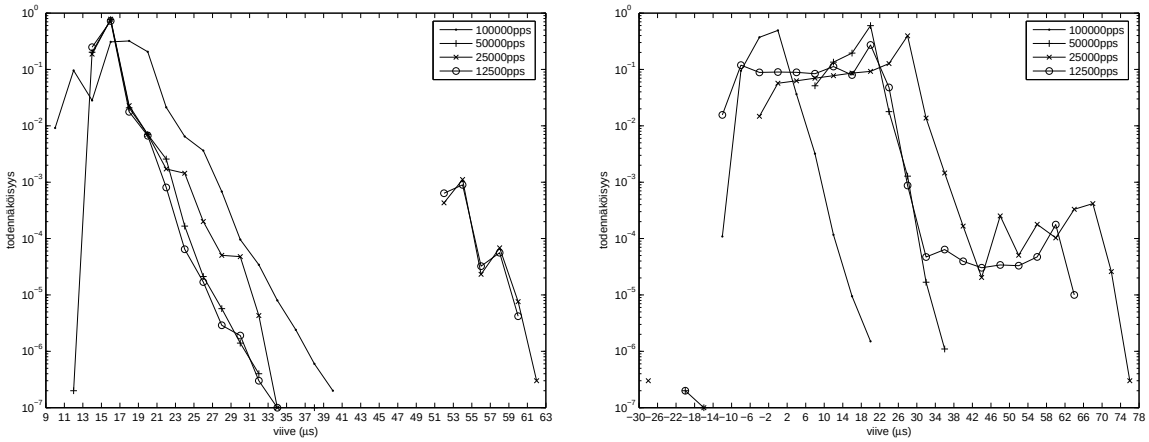
Celeron-koneiden välillä jokaisessa mittauksessa näkyy yhden jakajan aiheuttama

	SynPCI-X (Celeron 1/Celeron 2)								Sarjaportti
	100000 pps		50000 pps		25000 pps		12500 pps		
Näytteitä	10^7	10^7	10^7	10^7	10^7	10^7	10^7	10^7	87523
Keskiarvo	17,20	16,87	15,55	15,19	15,64	15,26	15,51	15,24	13,00
Varianssi	6,64	6,70	0,62	0,64	3,09	1,47	2,87	3,45	1,21
Vinous	-0,36	-0,33	3,14	3,08	17,08	19,55	18,42	17,33	1,51
Huipukkuus	3,78	3,58	22,33	21,56	362,56	593,31	409,21	350,97	10,72
Minimi	9,70	9,24	11,95	13,36	13,86	13,65	13,82	13,53	10,61
25. PP	15,90	15,53	15,07	14,74	15,11	14,78	15,03	14,74	12,26
50. PP	17,36	17,03	15,45	15,07	15,49	15,15	15,36	15,07	12,92
75. PP	18,95	18,65	15,86	15,49	15,86	15,53	15,75	15,45	13,60
99. PP	23,24	22,74	18,90	18,57	19,57	18,53	18,78	18,61	16,53
Maksimi	40,90	45,53	37,82	31,65	61,82	60,99	60,82	60,53	25,44

Taulukko 5.4: Celeron SynPCI-X kulkuaikaviive (μs)

noin 300 ns ero. Muuten arvot ovat lähes identtiset lukuun ottamatta maksimiarvoja. Histogrammista voi huomata, että 100000 pps mittauksen jakauma on hiukan erilainen kuin kolmessa muussa mittauksessa. Histogrammin korkein piikki on siirtynyt parilla mikrosekunnilla ja lisäksi 12 mikrosekunnin kohdalle on ilmestynyt uusi heikompi piikki. Tämä näkyy myös siinä, että vinousarvo on muuttunut negatiiviseksi. Tämä tarkoittaa sitä, että jakauman painopiste on siirtynyt keskiarvon vasemmalle puolelle. Ainoa selitys tälle on, että NAPI:n vaikutus alkaa näkymään tällä pakettinopeudella. Yhden paketin käsittelyyn kuluu kokonaan pakettien välinen 10 μs aika ja ydin käsittelee heti perään rengaspuskurista seuraavaan paketin ilman uutta keskeytystä. Koska suorittimen ei tarvitse suorittaa kontekstin vaihtoa, joka joudutaan tekemään aina keskeytyksen alussa, niin kulkuaikaviive jää jopa pienemmäksi kuin sarjaportin kautta mitattu minimi keskeytysviive.

Histogrammista ja taulukosta on nähtävissä, että jakauman maksimiarvo kasvaa, kun paketteja vastaanotetaan hitaammin. Pitää kuitenkin muistaa, että hitaammalla pakettinopeudella 10 miljoonan paketin generoimiseen kuluu pidempi aika. Tällöin myös kasvaa todennäköisyys, että paketti saapuu juuri silloin, kun ydin on tilassa, jota ei voida keskeyttää.



(a) SynPCI-X

(b) NTP

Kuva 5.6: Kulkuaikaviivejakauma Celeron-koneella

5.3.3 Celeron NTP

NTP-mittauksissa kumpikin Celeron-kone synkronoitiin NTP:llä 10/100 Ethernet –hallintakytkimen yli. Hallintakytkimeen liitetyn verkkokortin lähetys- ja vastaanottokeskeytyksien viiveet olivat säädettynä minimiin eli yhteen mikrosekuntiin. Tällä tavalla saatiin pakettien kulkuaikaviive lähes symmetriseksi hallintakytkimen yli. NTP-palvelimena käytettiin Opteron-konetta, joka oli synkronisoitu käyttäen SynPCI-X –korttia. Kuten kappaleessa 4.1 mainittiin tietokoneen kiteen taajuus on hyvin lämpötilariippuva. Celeron-koneiden emolevyn kiteiden lämpötilaa mitattiin käyttäen Dallasin DS1820-lämpötila-antureita. Mittauksen aikana otettiin myös talteen NTP:n ilmoittamat kellopoikkeamat. Taulukossa 5.5 on mittausdatan lisäksi ilmoitettu mittauksen aikana mitatut minimi- ja maksimiarvot kiteen lämpötilalle ja kellon aikapoikkeamalle. Mittausdatasta piirretyt histogrammit löytyvät kuvasta 5.6 (b).

Kun paketteja generoidaan 100000 paketin sekuntinopeudella, 10 miljoonan paketin generoimiseen kuluu aikaa vain 100 sekuntia. Kiteen lämpötila ehtii nousta kuormituksen vuoksi tänä aikana toisessa koneessa 0,4 °C ja toisessa 0,6 °C. Lämpötilan muutos kasvattaa NTP:n ilmoittaman aikapoikkeaman vaihtelua. Näin lyhyessä mittauksessa kello ei kuitenkaan ehdi kummassakaan koneessa vaeltamaan kuin muutaman mikrosekunnin. Korjaamalla kulkuaikaviiveitä NTP:n ilmoittamalla aikapoikkeamalla saadaan hyvin lähelle samanlaisia tuloksia kuin SynPCI-X –mittauksissa. Varianssi on kasvanut vain hieman kellon vaeltamisen takia.

Kun paketteja generoidaan 50000 pps nopeudella, mittaus kestää kaksi kertaa pidem-

	NTP (Celeron 1/Celeron 2)							
	100000 pps		50000 pps		25000 pps		12500 pps	
Näytteitä	10^7	10^7	10^7	10^7	10^7	10^7	10^7	10^7
Keskiarvo	-1,97	-3,62	17,70	20,31	19,79	20,74	8,89	-9,20
Varianssi	6,95	6,92	13,24	28,80	97,72	352,46	119,68	642,97
Vinous	-0,31	-0,24	-0,99	-1,08	-0,65	-0,79	-0,27	-0,28
Huipukkuus	3,58	3,52	3,03	3,09	2,65	2,36	1,79	1,59
Minimi	-10,84	-12,63	-19,78	-23,78	-29,07	-39,45	-18,53	-54,84
25. PP	-3,34	-5,00	15,53	17,28	12,40	7,07	-1,04	-33,13
50. PP	-1,84	-3,50	19,32	22,78	23,45	26,95	10,53	-6,21
75. PP	-0,17	-1,84	20,32	24,28	28,32	37,65	20,11	16,86
99. PP	3,90	2,40	22,74	26,90	30,61	40,65	22,86	21,61
Maksimi	21,99	22,74	36,07	39,53	75,65	84,07	65,70	64,90
Kide minimi	33,1°C	34,1°C	32,2°C	33,2°C	33,8°C	35,0°C	33,7°C	34,9°C
Kide maksimi	33,5°C	34,7°C	33,1°C	34,3°C	34,8°C	36,7°C	34,4°C	35,9°C
Poikkeama minimi	17,95	18.00	-7,00	-10,00	-13,00	-25,04	-5,00	-4,00
Poikkeama maksimi	18,89	22.00	-4,00	-2,14	7,63	15,00	26,00	69,70

Taulukko 5.5: Celeron-koneen NTP-kulkuaikaviive (μs)

pään eli 200 s. Tässäkin mittauksessa toisen Celeron-koneen kide on hieman viileämpi ja sen lämpötilan muutos on pienempi kuin toisen Celeron-koneen. Lämpötilan muutos on kuitenkin kummassakin koneessa kaksinkertainen, kun mittausaika on kaksi kertaa pidempi. Pidempi mittausaika näkyy myös siinä, että NTP:n ilmoittama kellon aikapoikkeama muuttuu huomattavasti enemmän. Kellon vaeltaminen näkyy myös selvästi varianssissa. Histogrammeja tarkasteltaessa jokaisessa näkyy edelleen yksi korkeampi piikki, mutta jakauma on levinnyt tämän piikin vasemmalle puolelle.

Pakettinopeuden laskiessa suorittimen kuorma laskee ja lämpötila nousee hitaammin. Mittausaika kuitenkin pitenee ja näin lämpötila ehtii muuttumaan lähes yhtä paljon. Mittausajan kasvaessa aikapoikkeama ehtii myös muuttumaan enemmän ja näin kulkuaikaviiveen varianssi kasvaa.

5.3.4 Linux ja FreeBSD

Vertailun vuoksi kulkuaikaviivettä mitattiin myös normaalilla FreeBSD 6.1 käyttöjärjestelmällä ja Linuxillla, joka oli varustettu 2.6.18 ytimellä PF_RING-tuen kanssa. FreeBSD:n tapauksessa kaappaava verkkokortti oli kiertokyselytilassa ja yti-

men *idle_poll* –optio oli laitettu päällä. Tällöin verkkokortin kiertokyselyä suoritetaan myös silloin, kun suorittimella ei ole muuta ohjelmaa suoritettavana. Linuxissa asetukset olivat samat kuin aikaisemmissa mittauksissa, mutta taustalla oli ajossa huonoimmalla prioriteetilla *stress*-ohjelma, joka pitää suorittimen kuorman koko ajan maksimissa.

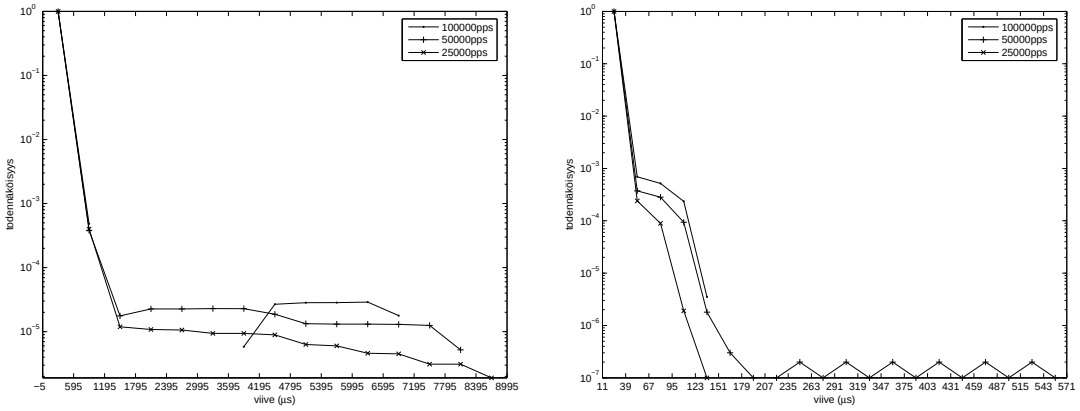
	NTP (Celeron 1/Celeron 2)							
	FreeBSD 6.1				Linux 2.6.18			
	100000 pps		50000 pps		100000 pps		50000 pps	
Näytteitä	10^7	10^7	10^7	10^7	10^7	10^7	10^7	10^7
Keskiarvo	17,08	17,35	6,91	4,11	15,53	12,87	19,12	20,50
Varianssi	7536,1	6486,8	5886,9	3553,2	5,72	7,05	4,52	6,36
Vinous	38,66	35,50	54,45	44,23	26,29	26,29	27,11	15,36
Huipukkuus	2544,1	2458,1	4337,7	3567,3	826,3	800,7	3073,6	296,5
Minimi	-2,42	-2,79	-4,38	-6,00	12,70	10,32	15,32	15,49
25. PP	-1,42	-1,42	-2,71	-4,46	14,99	12,40	17,90	19,03
50. PP	-1,13	-0,92	-1,84	-4,13	15,36	12,70	19,32	21,03
75. PP	-0,75	-0,38	-1,13	-3,75	15,78	13,07	20,20	21,70
99. PP	266,03	269,49	229,74	221,45	17,70	14,86	21,82	23,40
Maksimi	7133,3	7040,0	8237,8	6889,4	136,74	159,49	558,61	157,78
Kide minimi	40,0°C	43,3°C	40,2°C	43,4°C	40,1°C	43,8°C	40,0°C	43,5°C
Kide maksimi	40,1°C	43,3°C	40,3°C	43,4°C	40,2°C	43,8°C	40,2°C	43,7°C
Poikkeama minimi	-5,96	-5,92	-6,98	-8,92	4,15	2,03	6,11	6,96
Poikkeama maksimi	-3,97	-3,97	-3,00	-6,04	9,86	2,96	10,94	12,87

Taulukko 5.6: Celeron FreeBSD 6.1 ja Linux 2.6.18 kulkuaikaviive (μs)

Taulukossa 5.6 on esitetty tulokset kummankin käyttöjärjestelmän osalta 100000 pps ja 50000 pps pakettinopeuksilla. Koska mittauksien aikana suorittimen kuormitus on koko ajan maksimissaan, lämpötilan vaihtelu on pienempi kuin aikaisemmissa mittauksissa. Mittausajankohdat oli myös valittu niin, että mittaushuoneen lämpötilanmuutos mittauksen aikana oli hyvin pieni. Pakettikaappaus aiheuttaa joka tapauksessa kiintolevylle kirjoitusoperaatioita, jotka mahdollisesti vaikuttavat koneen sisäiseen lämpötilaan. Lisääntynyt verkkoliikenne vaikuttaa myös NTP:n toimintaan, koska taajuusvirhettä mitataan verkon kautta ja paketit kulkevat saman verkkopinon läpi kuin kaapattavat paketit.

FreeBSD:n mittaustuloksia tarkkailtaessa nähdään, että kellovirhe huomioon ottaen minimi kulkuaikaviive on hiukan alle 3 μs. Koska aikapoikkeaman minimiarvo on

kummassakin koneessa lähes sama, yhden jakajan aiheuttama 300 ns viive näkyy minimiarvossa. FreeBSD:n tapauksessa kulkuaikaviivejakauma on kuitenkin huomattavasti leveämpi kuin Linuxin tapauksessa. Kuvassa 5.7 (a) on esitetty histogrammi kulkuaikaviiveestä eri pakettinopeuksilla. Kuvasta nähdään, että pieni osa paketeista odottaa buffereissa ennen aikaleimausta jopa millisekunteja. Tällöin 99 prosentin rajakin on noussut yli 200μs:n.



(a) FreeBSD 6.1

(b) Linux 2.6.18

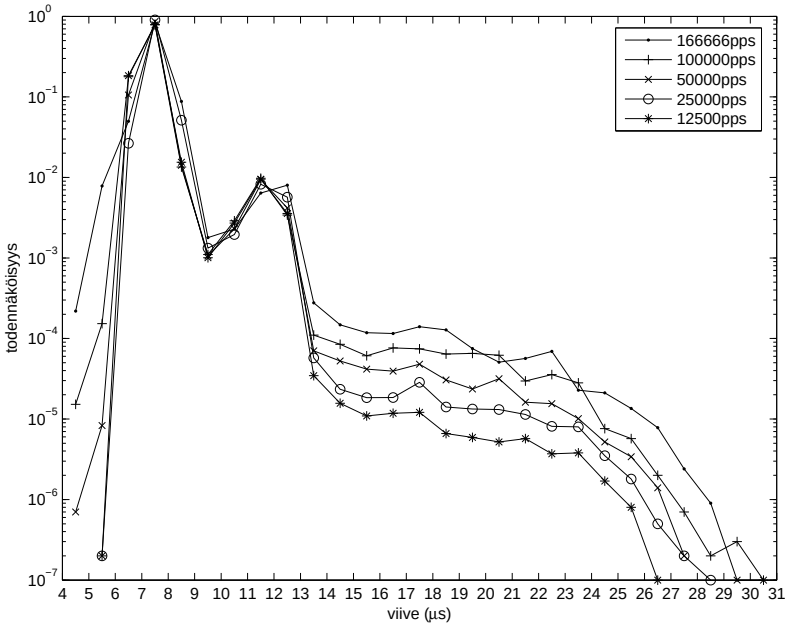
Kuva 5.7: Kulkuaikaviivejakauma normaaleilla ytimillä

Linuxin kohdalla tulokset poikkeavat aikaisemmista NTP-mittauksen tuloksista, koska kiteen lämpötilanmuutos on hyvin pieni. Tulokset eroavat SynPCI-X –mittauksista vain kellopoikkeaman verran. Koska käytössä ei ole reaaliaikaydintä, maksimiviive kasvaa kuitenkin satoihin mikrosekunteihin. Tämä johtuu suoraan siitä, ettei normaalin Linuxin ytimen kaikkia systeemikutsuja voida keskeyttää. Kuvassa 5.7 (b) on esitetty jakauma pakettien kulkuaikaviiveestä. Jakaumasta voidaan sanoa, että suurempi pakettinopeus vaikuttaa hiukan viimeisen prosentin aiheuttamaan häntään. Pakettinopeus ei kuitenkaan vaikuta siihen minkälaisen ei-keskeytettävän systeemikutsun aikana paketti sattuu saapumaan. Ei-keskeytettävien systeemikutsujen jäljellä oleva suoritusaika voi vaihdella varsin paljon.

5.3.5 Intelin verkkokortti

Jotta verkkokortin osuus kulkuaikaviivejakaumasta saataisiin selville, päätettiin mitata kulkuaikaviivettä Opteron-koneella ja Intelin verkkokortilla. Verkkokorttina käytettiin kappaleessa 2.4.4 esiteltyä kaksiporttista gigabitin korttia. Kortti oli kytkettynä samaan PCI-X –väylään kuin emolevylle integroitu Broadcomin verkkokortti.

Mittauksien ajaksi Intelin verkkokortista on otettu pois päältä kaikki kappaleessa 2.4.4 esitellyt keskeytyksiin liittyvät laskurit, jotka viivästyttävät keskeytyksiä tai rajoittavat keskeytyksien määrää. Lisäksi aivan kuten aikaisemmissa mittauksissa verkkokortin keskeytykset ovat siirretty omalle suorittimelleen, jossa ei ollut muita prosesseja ajossa.



Kuva 5.8: Kulkuaikaviivejakauma Intelin verkkokortilla

Kuvassa 5.8 on piirrettynä kulkuaikaviivejakauma, kun pakettien kaappaukseen on käytetty Intelin verkkokorttia. Aivan kuten aikaisemmissa mittauksissa jakauma ei muutu kuin hännän osalta pakettinopeuden muuttuessa. Sama ilmiö on myös nähtävissä taulukossa 5.7 olevista arvoista. Jakauma ei kuitenkaan ole samanlainen mitä Broadcomin verkkokortilla ja sarjaportin kautta mitattuna. Kun vertaa jakaumaa kuvassa 5.4 (a) olevaan jakaumaan, huomataan Intelin verkkokorttin aiheuttaneen jakaumaan toisen piikin 4μs:n päähän ensimmäisestä. Aikasarjoja tarkasteltaessa tämä näkyy ilmiönä, jossa tasaisin välein yksi paketti on viivästynyt noin 4μs. Tämä ilmeisemmin johtuu verkkokortin arkkitehtuurista, koska vastaavaa ilmiötä ei ole nähtävissä Broadcomin verkkokortilla.

Muuten kulkuaikaviiveestä voidaan sanoa, että kulkuaikaviive on systemaattisesti $3\,\mu$s pienempi kuin Broadcomin verkkokortilla. Toisaalta osa paketeista, jotka viivästyvät $4\,\mu$s muita paketteja enemmän siirtävät 99 prosenttipisteen samalle tasolle

	166666pps	100000pps	50000pps	25000pps	12500pps
Näytteitä	10^7	10^7	10^7	10^7	10^7
Keskiarvo	7,62	7,42	7,49	7,61	7,41
Varianssi	0,60	0,47	0,43	0,41	0,40
Vinous	5,92	6,60	5,90	5,49	5,15
Huipukkuus	71,05	79,68	61,33	50,93	42,42
Minimi	4,36	4,50	4,45	5,95	5,32
25. PP	7,28	7,07	7,15	7,28	7,07
50. PP	7,57	7,36	7,40	7,53	7,36
75. PP	7,82	7,61	7,7	7,82	7,61
99. PP	11,95	11,49	11,57	11,65	11,45
Maksimi	29,49	30,57	30,78	28,61	26,57

Taulukko 5.7: Opteron SynPCI-X kulkuaikaviive Intelin verkkokortilla (μs)

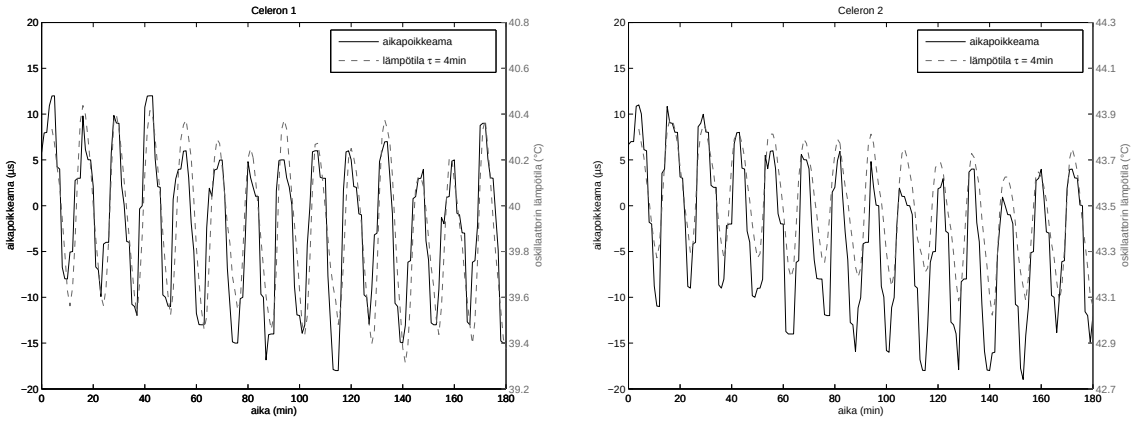
kuin Broadcomin verkkokortilla. Ilmiö voi myös johtua Intelin verkkokortin ajurista. Mittauksissa käytettiin Intelin ajurin versiota 7.5.5.1.

5.4 Lämpötilan vaikutus NTP-synkronointiin

Koska kiteen lämpötilan vaihtelulla on suuri merkitys NTP-synkronoinnin tarkkuuteen, päätettiin mitata kiteen lämpötilaa ja NTP-palvelimen ilmoittamaa aikapoikkeamaa. Kumpaankin suuretta mitattiin minuutin välein kummaltakin Celeron-koneelta. NTP-synkronointi suoritettiin SynPCI-X –synkronoidulta NTP-palvelimelta yhden hyvin vähän kuormitetun hallintakytkimen yli. Celeron-koneiden NTP-synkronointiväli oli säädettynä minimiin eli 16 sekuntiin.

Kuvassa 5.9 on esitetty kummankin koneen NTP:n ilmoittama aikapoikkeama ja lämpötila viivästettynä 4 minuutilla kolmen tunnin ajalta. Kuvasta voidaan päätellä, että NTP reagoi lämpötilanmuutokseen 4 minuutin viiveellä. Kuvassa näkyvä 12 minuutin lämpötilan jaksollisuus johtuu mittaushuoneen ilmastointilaitteesta, mutta asteen lämpötilanmuutos voi myös syntyä koneen kuormitusasteen muutoksen johdosta. Tilannetta voidaan hiukan parantaa tuomalla PPS-signaali sarjaportin kautta suoraan ytimelle. Tällöin laskurin taajuusvirhe pystytään määrittämään joka sekunti 16 sekunnin sijaan ja lämpötilanmuutos pystytään kompensoimaan nopeammin.

Tällöinkin NTP-prosessi kuitenkin reagoi huomattavasti hitaammin lämpötilan muutoksiin kuin SynPCI-X –kortin avulla synkronisoitu TSC-laskuri. SynPCI-X –synk-



(a) (b)

Kuva 5.9: Lämpötilan vaikutus aikapoikkeamaan

ronoinnin tapauksessa kun TSC-laskuria kalibroidaan vähintään 100 kertaa sekunnissa.

5.5 Virhelähteet

Mittauksien perusteella voidaan sanoa, että pakettikaappauksessa aikaleimojen virheet koostuvat keskeytysviiveestä, kellon virheestä sekä käytettävästä verkkokortista. Kellon virheeseen NTP:n tapauksessa suurimmaksi osaksi vaikuttaa tietokoneen emolevyn kiteen lämpötilan muutokset.

SynPCI-X –korttia käytettäessä saadaan kellovirhe minimoitua alle mikrosekuntiin. Tämän jälkeen suurimman virheen aiheuttaa keskeytysviive. Koneissa, joissa on useampi suoritin keskeytysviive saadaan minimoitua, kun yksi suorittimista varataan pelkästään pakettikaappaukseen. Yhden suorittimen koneissa keskeytysviiveen maksimia saadaan huomattavasti pienennettyä, kun käytetään Linuxin reaaliaikalaajennosta. Useamman suorittimen tilanteessa absoluuttisen virheen maksimiksi saadaan noin 25μs. Tämä tietenkin riippuu käytetystä laitteistosta. Yhden suorittimen tilanteessa reaalilaajennuksen avulla maksimi keskeytysviive saadaan pidettyä 60μs luokassa. Tämä tietenkin on riippuvainen käytetystä laitteistosta ja nopeimmilla koneilla arvo luultavasti pienentyy.

NTP-synkronoinnin tapauksessa kellopoikkeama aiheuttaa optimitilanteessakin lisää mittavirhettä kymmeniä mikrosekunteja ja kuormitetun verkon tapauksessa helposti

satoja mikrosekunteja. Jos mittauksissa käytetyt koneet synkronoidaan lähiverkon yli tarkalta stratum 1 NTP-palvelimelta, niin samalla helposti voidaan määrittää kellopoikkeaman minimi ja maksimi. Aikaleimojen maksimivirhe saadaan tällöin määritettyä lisäämällä SynPCI-X –mittauksista saatuihin rajoihin kellon aikapoikkeama.

Luku 6

Yhteenveto

6.1 Johtopäätökset

Työn ensimmäisenä tavoitteena oli kehittää SynPCI-korttia. Työn aikana kortista suunniteltiin uusi versio, joka on huomattavasti halvempi valmistaa kuin edeltäjänsä. Lisäksi uudessa versiossa kortille on mahdollista tuoda ulkoisia signaaleja ja ledit helpottavat virheiden havainnointia sisääntulevissa signaaleissa. Yhden SynPCI-X –kortin hinnaksi tuli hieman yli sata euroa. Kun tähän lisätään kahden Intelin kaksiporttisen gigabittisen verkkokortin hinta, neljäporttisen kaappaussysteemin kustannuksiksi tulee noin 500 euroa. Erikoisvalmisteisia kaappauskortteja käytettäessä neljän portin hinta nousee halvimmillaankin kymmenenkertaiseksi. GPS-vastaanottimen hintaa ei ole otettu tässä huomioon, mutta yksi vastaanotin on joka tapauksessa hankkittava jokaiseen mittauspaikkaan.

Toisena tavoitteena oli kehittää mittausmenelmiä pakettien aikaleimojen virheiden määrittämiseksi. Työn aikana kehitettiin menetelmä, jossa paketit aikaleimataan tarkasti lähetyspäässä ja sen jälkeen sama paketti jaetaan useammalle koneelle. Tälläisella menetelmällä saadaan minimoitua tietokoneen ulkopuolisten virhelähteiden vaikutus. Käyttämällä AX/4000-mittalaitteessa kalliimpaa pakettigeneraattorikorttia saataisiin pakettien lähettämisen ajoituksesta aiheutuva pieni virhe aikaleimoissa eliminoitua.

Viimesenä tavoitteena oli mitata passiivisissa pakettikaappauksessa aikaleimojen tarkkuutta. Mittauksissa todettiin SynPCI-X –kortin avulla päästään pakettikaappauksissa muutaman kymmenen mikrosekunnin tarkkuuteen. Opteron-koneella 99 prosentilla paketeista aikavirhe on vain hieman yli kymmenen mikrosekuntia. Celeron-koneella 99 prosentin raja on hiukan alle 30 mikrosekuntia. Nämä rajoitukset huomioon

ottaen yleiskäyttöinen tietokone soveltuu käytettäväksi monipistemittauksissa kohtuullisen hyvin.

SynPCI-X –kortin avulla synkronisoidussa koneessa kellon pitkäaikainen stabiilisuus on myös hyvä, jos GPS-laite pysyy synkronoituna satelliiteihin. Lisäksi kuormituksen vaihtelu ei vaikuta oleellisesti kellon tarkkuuteen. NTP-synkronointiin verrattuna saavutetaan huonoimmassakin tapauksessa yli 10 mikrosekunnin parannus ja normaalisti kymmenien tai satojen mikrosekuntien parannus.

Kaappausprosessin sopiva optimointi saattaisi pitää maksimivirheen alle 10 mikrosekunnissa, jos kaappaukselle on varattu oma suorittimensa. Tällä hetkellä maksimivirhettä luultavasti kasvattaa kellon lukitseminen Linuxissa päivityksen ajaksi.

Yhden suorittimen koneissa maksimivirhe on aina suurempi, koska suoritin hoitaa välillä ei-keskeytettäviä prosesseja. Reaaliaikaytimellä saadaan muista prosesseista johtuva viive kuitenkin pienennettyä alle 50 mikrosekuntiin. Tulevaisuudessa nopeimmilla koneilla tämäkin viive pienenee.

Alle mikrosekunnin tarkkuuteen pääseminen ei ole mahdollista nykyisillä koneilla ja keskeytyspohjaisilla systeemeillä. Tähän tarkoitukseen on pakko käyttää erikoisvalmisteisia kaappausverkkokortteja, jotka tekevät aikaleimauksen kortilla. Kun tällainen kortti synkronoidaan GPS-vastaanottimen avulla, päästään alle mikrosekunnin leimaustarkkuuteen.

Kortin käyttömahdollisuudet eivät rajoitu pelkkään passiiviseen pakettikaappaukseen Ethernet-verkoissa, toisin kuin erikoisvalmisteisilla kaappauskorteilla. SynPCI-X –korttia on mahdollista käyttää synkronointiin aktiivisissa liikennemittauksissa. Lisäksi kortti ei aseta rajoitteita käytettävälle siirtoyhteystekniikalle. Kortti soveltuu myös muihin pitkäaikaista kellosynkronisoinnin stabiilisuutta vaativiin mittauksiin. Koska kortille voidaan tuoda ulkoisia signaaleja, niin mittauskohteet eivät myöskään rajoitu pelkästään tietokoneen sisäisten tapahtumien mittaamiseen.

6.2 Jatkokehityskohteet

Työssä keskityttiin ainoastaan vastaanotettavien pakettien aikaleimaukseen. Aktiivimittauksissa tarvitaan kuitenkin myös lähetettävien pakettien aikaleimausta. Tätä varten olisi tutkittava, päästäänkö lähetyspuolella samaan tarkkuuteen kuin vastaanottopuolella.

Saatujen tulosten perusteella voidaan sanoa, että aikaleimojen tarkkuuden parantamiseksi kannattaisi keskittyä keskeytysviiveen optimoitiin. Hyper Transport, MSI

ja MSI-X –keskeytyksien käytön vaikutusta tulisi mitata ja todeta kuinka ne vaikuttavat viiveeseen. Tällä tavoin saataisiin mahdollisesti pienennettyä keskeytysviivettä.

Koska suorittimen keskeytysviivettä ei kuitenkaan saada kokonaan poistettua, niin yhtenä ratkaisuna olisi ohjata verkkokortin keskeytykset SynPCI-X –kortille. MSI-keskeytykset mahdollistavat muistialueen määrittämisen, johon oheislaite kirjoittaa, kun se haluaa aiheuttaa keskeytyksen. Kyseinen muistialue voidaan konfiguroida SynPCI-X –kortin muistialueesta. Tämä mahdollistaisi sen, että SynPCI-X –kortti kirjoittaisi suoraan verkkokortin rengaspuskuriin paketin aikaleiman. Toteutuksessa tarvitsisi kuitenkin verkkokortin ajuria ja SynPCI-X –kortin VHDL-lähdekoodia muokata niin, että ne toimivat yhteistyössä.

Tällaisella ratkaisulla voitasiin päästä hyvin lähelle samaa tarkkuutta kuin erikoisvalmisteisilla kaappauskorteilla. Ainoa lisäviive muodostuisi paketin kopioimisesta kortin muistilta keskusmuistiin. Tämä viive ei ole vakio, koska paketin pituus vaikuttaa kopiointiaikaan. Lisäksi MSI-keskeytyksien signaloinnista muodostuu oma viiveensä, joka ei välttämättä ole vakio.

Samalla kun verkkokortin ajuria jouduttaisiin muokkaamaan runsaasti, kannattaisi ajuri optimoida pelkästään pakettikaappausta varten. Pakettikaappauksen optimoimiseksi koko kaappausprosessi kannattaisi tehdä ydintilassa, koska datan siirtäminen välillä käyttäjätilaan on turha aikaavievä toimenpide. Tämä on järkevää, jos analyysi tehdään reaaliajassa. Jotta aktiivimittausohjelmien kirjoittaminen helpottuisi, täytyisi ytimen tarjota rajapinta pakettien luomiseen ja analysoimiseen.

Ytimen tarjoamia levy-I/O –funktioita voitaisiin kuitenkin hyödyntää. Hyvänä puolena tässä toteutusvaihtoehdossa on, ettei paketteja tarvitse kopioida keskusmuistissa paikasta toiseen. Kun pakettien tiedot kirjoitetaan sopiviin tietueisiin, niin ne voidaan kirjoittaa sellaisenaan kiintolevylle.

Toinen vaihtoehto tarkkuuden parantamiseksi on käyttää SynPCI-X –kortille tuotavia ulkoisia signaaleja, joiden avulla liipaistaan laskurin arvo talteen. Tällä tavalla ulkoisten tapahtumien aikaleimaustarkkuudessa päästäisiin laskurin resoluution tarkkuuteen. Tässäkin tapauksessa VHDL-lähdekoodia pitäisi muokata niin, että kortti siirtäisi laskurista liipaistut arvot tietokoneen keskusmuistissa olevaan rengaspuskuriin.

Kirjallisuutta

[Grö04] Antti Gröhn. Testiverkon kellosynkronointi. Master's thesis, Teknillinen korkeakoulu, Lokakuu 2004.

[IEC91] IEC. *IEC Standard 60950-1*, 1991.

[IEE05] IEEE. *802.3-2005: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, December 2005.

[ISO94] ISO/IEC. *ISO Standard 7498-1 Second edition*, 1994.

[Lap04] Phullip A. Laplante. *Real-Time Systems Design and Analysis, 3rd. Edition.* Wiley - IEEE Press, 2004.

[ref00] *PCI-X Addendum to the PCI Local Bus Specification Revision 1.0a*, July 2000.

[ref01] *HyperTransport I/O Link Specification Revision 1.03*, October 2001.

[ref02] *PCI Local Bus Specitication Revision 3.0*, August 2002.

[ref04a] *AMD-8111 HyperTransport I/O Hub Data Sheet Revison 3.03*, July 2004.

[ref04b] *AMD-8131 HyperTransport PCI-X Tunnel Data Sheet Revision 3.02*, August 2004.

[ref04c] *HyperTransport I/O Link Specification Revision 2.00*, February 2004.

[ref04d] *IA-PC HPET (High Precision Event Timers) Specification Revision 1.0a*, October 2004.

[ref04e] *Intel 6702PXH 64-bit PCI Hub Data Sheet*, September 2004.

[ref04f] *Intel E7221 Chipset Data Sheet*, September 2004.

[ref04g] *Pericom PI5C32X245 Data Sheet*, August 2004.

[ref05a] *82546GB Dual Port Gigabit Ethernet Controller Datasheet Revision 1.7*, October 2005.

[ref05b] *Intel Controller Hub 6 (ICH6) Family Data Sheet*, January 2005.

[ref05c] *http://source.mvista.com/linux_2_6_RT.html* , April 2005.

[ref05d] *http://www.intel.com/technology/ioacceleration/*, April 2005.

[ref06a] *Advanced Configuration and Power Interface Specification Revision 3.0b*, October 2006.

[ref06b] *HyperTransport I/O Link Specification Revision 3.00*, April 2006.

[ref06c] *PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer's Manual Revision 3.3*, December 2006.

[ref06d] *http://www.hypertransport.org/consortium/index.cfm*, April 2006.

[SOK01] Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov. Beyond sofnet. In *Proceedings of the 5th Annual Linux Showcase & Conference*, 2001.

[Vig92] John R. Vig. *Introduction to Quartz Frequency Standards*. Army Research Laboratory Electronics and Power Sources Directorate, 1992.

[WJD98] John W. Poulton William J. Dally. *Digital systems engineering*, chapter 10. Cambridge University press, 1998.

Liite A

SynPCI-X –kortin Linux-ajuri

```
diff -urN linux-2.6.18/arch/i386/Kconfig linux-2.6.18-synpcix/arch/i386/Kconfig
--- linux-2.6.18/arch/i386/Kconfig  2006-09-20 06:42:06.000000000 +0300
+++ linux-2.6.18-synpcix/arch/i386/Kconfig  2007-04-05 10:25:56.000000000 +0300
@@ -211,6 +211,16 @@

 source "arch/i386/Kconfig.cpu"

+config SYNPCIX_TIMER
+        bool "SynPCI-X Timer Support"
+        depends on PCI
+        default y
+        help
+          Support for SynPCI-X Synchronization PCI Card.
+
+          Say Yes to use SynPCI-X card for
+          system clock update process.
+
 config HPET_TIMER
    bool "HPET Timer Support"
    help
diff -urN linux-2.6.18/arch/i386/kernel/Makefile linux-2.6.18-synpcix/arch/i386/kernel/Makefile
--- linux-2.6.18/arch/i386/kernel/Makefile  2006-09-20 06:42:06.000000000 +0300
+++ linux-2.6.18-synpcix/arch/i386/kernel/Makefile  2007-04-05 10:25:56.000000000 +0300
@@ -38,6 +38,7 @@
 obj-$(CONFIG_VM86)     += vm86.o
 obj-$(CONFIG_EARLY_PRINTK) += early_printk.o
 obj-$(CONFIG_HPET_TIMER)   += hpet.o
+obj-$(CONFIG_SYNPCIX_TIMER)    += synpcix.o
 obj-$(CONFIG_K8_NB)        += k8.o

 EXTRA_AFLAGS   := -traditional
diff -urN linux-2.6.18/arch/i386/kernel/synpcix.c linux-2.6.18-synpcix/arch/i386/kernel/synpcix.c
--- linux-2.6.18/arch/i386/kernel/synpcix.c 1970-01-01 02:00:00.000000000 +0200
+++ linux-2.6.18-synpcix/arch/i386/kernel/synpcix.c 2007-04-05 10:26:23.000000000 +0300
@@ -0,0 +1,306 @@
```

```
+/*
+ * SynPCI-X Timer
+ * 2006 Oskari Simola TKK Networking Laboratory
+ *
+ * based on arch/i386/kernel/tsc.c.
+ * See comments there for proper credits.
+ */
+
+#include <linux/clocksource.h>
+#include <linux/workqueue.h>
+#include <linux/cpufreq.h>
+#include <linux/jiffies.h>
+#include <linux/init.h>
+#include <linux/dmi.h>
+#include <linux/proc_fs.h>
+
+#include <asm/delay.h>
+#include <asm/tsc.h>
+#include <asm/io.h>
+
+/* For SynPCI-X device */
+#include <linux/pci.h>
+#define SYNPCI_VENDOR_ID    0x1204
+#define SYNPCI_DEVICE_ID    0x9500
+#define MODULE_VERS "1.5"
+#define MODULE_NAME "synpcix"
+static unsigned long *synpcix_memaddr;
+static struct pci_dev *synpcix_dev;
+
+static struct proc_dir_entry *tsc_diff, *synpci_dir;
+long *tsc_diff_hist;
+static int error1=0,error2=0,errormin=0,errormax=0;
+
+static int proc_read_tsc_diff(char *page, char **start,
+                off_t off, int count,
+                int *eof, void *data)
+{
+  int i,len = 0;
+  for(i=0;i < 41;i++){
+    len += sprintf((char *)(&page[len+off]), "%d:%ld\n",(i-20)*50,tsc_diff_hist[i]);
+    if((len+36) > count){
+      eof[0] = 1;
+      return len;
+    }
+  }
+  len += sprintf((char *)(&page[len+off]), "error 1: %d error2: %d error min: %d
+          error max: %d\n",error1,error2,errormin,errormax);
+  eof[0] = 1;
+  return len;
+}
+
+
+static inline u32 read_synpcix(void)
```

```
+{
+  u32 c1,c2,c3;
+  int d;
+
+  /* read synpcix card three times because sometimes card retuns
+     false counter value */
+  c1 = readl(synpcix_memaddr);
+  c2 = readl(synpcix_memaddr);
+  c3 = readl(synpcix_memaddr);
+  if(c2 < c1 || c3 < c2){
+    d = c2 - c1;
+    if(d > 0)
+       return c1;
+    else
+      return c3;
+  }else
+    return c2;
+}
+
+static unsigned long calculate_cpu_khz_synpcix(void)
+{
+   unsigned long long start, end;
+   u64 delta64;
+   u32 synpcix_start, synpcix_end, synpcix_interval;
+   int i;
+   unsigned long flags;
+
+   local_irq_save(flags);
+
+   /* run 3 times to ensure the cache is warm */
+   for (i = 0; i < 3; i++) {
+     synpcix_start = read_synpcix();
+     rdtscll(start);
+     do {
+       synpcix_end = read_synpcix();
+       synpcix_interval = synpcix_end - synpcix_start;
+       /* run loop for 50ms */
+     } while ( synpcix_interval < 50000000);
+     rdtscll(end);
+   }
+   /*
+    * Error: ECTCNEVERSET
+    * The CTC wasn't reliable: we got a hit on the very first read,
+    * or the CPU was so fast/slow that the quotient wouldn't fit in
+    * 32 bits..
+    */
+
+   delta64 = end - start;
+   /* divide by 1000 to get usec */
+   synpcix_interval /= 1000000;
+
+   /* cpu freq too fast: */
+   if (delta64 > (1ULL<<32))
```

```
+       goto err;
+
+   /* cpu freq too slow: */
+   if (delta64 <= synpcix_interval)
+       goto err;
+
+   delta64 += synpcix_interval/2; /* round for do_div */
+   do_div(delta64,synpcix_interval);
+
+   local_irq_restore(flags);
+   return (unsigned long)delta64;
+err:
+   local_irq_restore(flags);
+   return 0;
+}
+
+/* clock source code */
+
+static int synpcix_update_callback(void);
+
+static cycle_t read_tsc(void)
+{
+   cycle_t ret;
+
+   rdtscll(ret);
+
+   return ret;
+}
+
+static struct clocksource clocksource_synpcix = {
+   .name           = "synpcix",
+   .rating         = 400,
+   .read           = read_tsc,
+   .mask           = CLOCKSOURCE_MASK(64),
+   .mult           = 0, /* to be set */
+   .shift          = 26,
+   .update_callback    = synpcix_update_callback,
+   .is_continuous      = 1,
+};
+
+
+
+static int synpcix_update_callback(void)
+{
+  static unsigned int last_sec;
+  static unsigned int last_pps_val;
+
+  int loops = 2, pps_pulse = 0;
+  unsigned int count_val, last_nsec;
+  unsigned int tsc_inc, synpci_nsec;
+  unsigned int pps_val;
+  long int diff;
+  cycle_t cycle_delta, cycle;
```

```
+
+  last_nsec = xtime.tv_nsec;
+
+  while(loops > 0){
+    count_val = readl(synpcix_memaddr);
+    cycle = clocksource_synpcix.read();
+    pps_val = readl(synpcix_memaddr + 1);
+    cycle_delta = (cycle - clocksource_synpcix.cycle_last) & clocksource_synpcix.mask;
+    synpci_nsec = (count_val&0xfffffffe) - pps_val;
+    tsc_inc = cyc2ns(&clocksource_synpcix, cycle_delta);
+    if(last_pps_val != pps_val){
+      if((pps_val - last_pps_val) != NSEC_PER_SEC){
+   pps_val = readl(synpcix_memaddr + 1);
+   if((pps_val - last_pps_val) != NSEC_PER_SEC){
+     printk("synpcix DEBUG: pps_val: %d last_pps_val: %d\n", pps_val, last_pps_val);
+     last_pps_val = pps_val;
+     diff = last_nsec + tsc_inc - synpci_nsec - NSEC_PER_SEC;
+     break;
+   }
+      }
+      xtime.tv_sec++;
+      pps_pulse = 1;
+
+      if(last_nsec > NSEC_PER_SEC)
+   printk("Missing PPS pulse %ld nsec: %d tsc: %d mult: %d\n", xtime.tv_sec,
+               xtime.tv_nsec, last_nsec + tsc_inc, clocksource_synpcix.mult);
+    }
+    last_pps_val = pps_val;
+
+    if(pps_pulse){
+      diff = last_nsec + tsc_inc - synpci_nsec - NSEC_PER_SEC;
+    }else{
+      diff = last_nsec + tsc_inc - synpci_nsec;
+    }
+
+    if(diff < 1000 && diff > -1000){
+      tsc_diff_hist[(int)((diff/50)+20)]++;
+      break;
+    }else{
+      if(loops == 2)
+   error1++;
+      if(loops == 1){
+   error2++;
+   if(diff < errormin)
+     errormin = diff;
+   if(diff > errormax)
+     errormax = diff;
+      }
+    }
+    loops--;
+  }
+
+  if(diff > 10000){
```

```
+    xtime.tv_sec++;
+    printk("synpcix DEBUG: over 10us positive offset. 1 sec added to xtime\n");
+  }
+
+  clocksource_synpcix.cycle_last = cycle;
+  xtime.tv_nsec = synpci_nsec;
+
+  if(diff < 0){
+    clocksource_synpcix.mult++;
+  }else{
+    clocksource_synpcix.mult--;
+  }
+
+  return 0;
+}
+
+static int __init init_synpcix_clocksource(void)
+{
+  unsigned long mmio_start, mmio_end, mmio_len, mmio_flags;
+
+  synpcix_dev = NULL;
+  /* Look for SynPCI-X device */
+  synpcix_dev = pci_get_device(SYNPCI_VENDOR_ID, SYNPCI_DEVICE_ID, synpcix_dev);
+  if(synpcix_dev) {
+    /* device found, enable it */
+    if(pci_enable_device(synpcix_dev)) {
+      printk(KERN_NOTICE "Could not enable SynPCI-X device\n");
+      return 0;
+    }
+  }else{
+    printk(KERN_NOTICE "SynPCI-X device not found\n");
+    return 0;
+  }
+
+  /* get PCI memory mapped I/O from space base address BAR0*/
+  mmio_start = pci_resource_start(synpcix_dev, 0);
+  mmio_end = pci_resource_end(synpcix_dev, 0);
+  mmio_len = pci_resource_len(synpcix_dev, 0);
+  mmio_flags = pci_resource_flags(synpcix_dev, 0);
+
+  /* make sure BAR0 region is MMIO */
+  if(!(mmio_flags & IORESOURCE_MEM)) {
+    printk(KERN_NOTICE "SynPCI-X: region #0 not MMIO region\n");
+    return 0;
+  }
+
+  if(pci_request_regions(synpcix_dev, MODULE_NAME)) {
+    printk(KERN_NOTICE "SynPCI-X: could get PCI regions\n");
+    return 0;
+  }
+
+  /* ioremap MMIO region */
+  synpcix_memaddr = ioremap(mmio_start, mmio_len);
```

```
+  if(synpcix_memaddr == NULL) {
+    printk(KERN_NOTICE "SYNPCI: Could not do ioremap\n");
+    return 0;
+  }
+
+  /* create directory */
+  synpci_dir = proc_mkdir(MODULE_NAME, NULL);
+  if(synpci_dir == NULL) {
+    return -ENOMEM;
+  }
+
+  synpci_dir->owner = THIS_MODULE;
+
+  tsc_diff = create_proc_read_entry("tsc_diff",
+                  0444, synpci_dir,
+                  proc_read_tsc_diff,
+                  NULL);
+
+  tsc_diff->owner = THIS_MODULE;
+
+  tsc_diff_hist = (unsigned long *) kmalloc(sizeof(unsigned long)*(40+1),GFP_KERNEL);
+
+  if(tsc_diff_hist == NULL) {
+    remove_proc_entry("tsc_diff", synpci_dir);
+    return -ENOMEM;
+  }
+
+  cpu_khz = calculate_cpu_khz_synpcix();
+
+  printk("Detected %lu.%03lu MHz processor.\n",
+    (unsigned long)cpu_khz / 1000,
+    (unsigned long)cpu_khz % 1000);
+
+  if (cpu_has_tsc && !tsc_disable) {
+    clocksource_synpcix.mult = clocksource_khz2mult(cpu_khz,
+                            clocksource_synpcix.shift);
+    return clocksource_register(&clocksource_synpcix);
+  }
+
+  return 0;
+}
+
+module_init(init_synpcix_clocksource);
diff -urN linux-2.6.18/arch/i386/kernel/time.c linux-2.6.18-synpcix/arch/i386/kernel/time.c
--- linux-2.6.18/arch/i386/kernel/time.c    2006-09-20 06:42:06.000000000 +0300
+++ linux-2.6.18-synpcix/arch/i386/kernel/time.c    2007-04-05 10:25:56.000000000 +0300
@@ -157,7 +157,6 @@
     * the irq version of write_lock because as just said we have irq
     * locally disabled. -arca
     */
-   write_seqlock(&xtime_lock);

 #ifdef CONFIG_X86_IO_APIC
```

```
    if (timer_ack) {
@@ -175,6 +174,7 @@
    }
 #endif

+   write_seqlock(&xtime_lock);
    do_timer_interrupt_hook(regs);


diff -urN linux-2.6.18/kernel/timer.c linux-2.6.18-synpcix/kernel/timer.c
--- linux-2.6.18/kernel/timer.c 2006-09-20 06:42:06.000000000 +0300
+++ linux-2.6.18-synpcix/kernel/timer.c 2007-04-05 10:25:56.000000000 +0300
@@ -884,6 +884,9 @@

    write_seqlock_irqsave(&xtime_lock, flags);

+#ifdef CONFIG_SYNPCIX_TIMER
+        if (memcmp(clock->name,"synpcix",7) != 0){
+#endif
    nsec -= __get_nsec_offset();

    wtm_sec  = wall_to_monotonic.tv_sec + (xtime.tv_sec - sec);
@@ -892,6 +895,11 @@
    set_normalized_timespec(&xtime, sec, nsec);
    set_normalized_timespec(&wall_to_monotonic, wtm_sec, wtm_nsec);

+#ifdef CONFIG_SYNPCIX_TIMER
+        }else{
+     xtime.tv_sec = sec;
+   }
+#endif
    clock->error = 0;
    ntp_clear();

@@ -1118,7 +1126,9 @@
    /* Make sure we're fully resumed: */
    if (unlikely(timekeeping_suspended))
        return;
-
+#ifdef CONFIG_SYNPCIX_TIMER
+   if (memcmp(clock->name,"synpcix",7) != 0){
+#endif
 #ifdef CONFIG_GENERIC_TIME
    offset = (clocksource_read(clock) - clock->cycle_last) & clock->mask;
 #else
@@ -1158,13 +1168,21 @@
    /* store full nanoseconds into xtime */
    xtime.tv_nsec = (s64)clock->xtime_nsec >> clock->shift;
    clock->xtime_nsec -= (s64)xtime.tv_nsec << clock->shift;
-
+#ifdef CONFIG_SYNPCIX_TIMER
+        }else{
+          time_status = STA_PLL | STA_PPSFREQ | STA_PPSTIME | STA_PPSSIGNAL;
```

```
+          time_freq = 0;
+          time_maxerror = 1;
+          time_esterror = 0;
+        }
+#endif
    /* check to see if there is a new clocksource to use */
    if (change_clocksource()) {
        clock->error = 0;
        clock->xtime_nsec = 0;
        clocksource_calculate_interval(clock, tick_nsec);
    }
+
 }

 /*
```

Liite B

SynPCI-X –kortin VHDL-lähdekoodi

B.1 synpcix.vhd

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

entity synpcix is

    port (
    GPS_CLK     : in  std_logic;
    PPS_PULSE   : in  std_logic;
    PLL_LOCK    : in  std_logic;
    LED1_GREEN  : out std_logic;
    LED1_RED    : out std_logic;
    LED2_GREEN  : out std_logic;
    LED2_RED    : out std_logic;
    PCI_CLK     : in  std_logic;
    PCI_RST     : in  std_logic;
    PCI_IRDY    : in  std_logic;
    PCI_FRAME   : in  std_logic;
    PCI_IDSEL   : in  std_logic;
    PCI_CBE     : in  std_logic_vector(3 downto 0);
    PCI_REQ     : out std_logic;
    PCI_GNT     : out std_logic;
    PCI_PAR     : out std_logic;
    PCI_PERR    : out std_logic;
    PCI_SERR    : out std_logic;
    PCI_TRDY    : inout std_logic;
    PCI_STOP    : inout std_logic;
    PCI_DEVSEL  : inout std_logic;
```

```
    PCI_AD      : inout std_logic_vector(31 downto 0));

end;

architecture behavioral of synpcix is

    component clkdiv2
        port (
        CLK,RESETN  : in  std_logic;
        CLK_OUT     : out std_logic);
    end component;

    component nsec_counter
        port (
        CLK     : in  std_logic;
        PCI_CLK     : in  std_logic;
        PPS_PULSE   : in  std_logic;
        LATCH       : in  std_logic;
        NSEC        : out std_logic_vector(31 downto 0);
        NSEC_PPS    : out std_logic_vector(31 downto 0);
        PPS_OUT     : out std_logic);
    end component;

    component led_control
        port (
        CLK,RESETN  : in  std_logic;
        PPS_PULSE   : in  std_logic;
        PLL_LOCK    : in  std_logic;
        LED1_GREEN  : out std_logic;
        LED1_RED    : out std_logic;
        LED2_GREEN  : out std_logic;
        LED2_RED    : out std_logic;
        PPS_ERROR   : out std_logic_vector(31 downto 0));
    end component;

    signal GPS_CLK_2: std_logic;
    signal GPS_CLK_4: std_logic;
    signal ATTR_CLK : std_logic;
    signal ATTR_CLK2: std_logic;
    signal LATCH_I  : std_logic;
    signal PPS_OUT_I : std_logic;
    signal PPS_OUT_I2 : std_logic;
    signal NSEC_PPS_I : std_logic_vector(31 downto 0);
    signal NSEC_I   : std_logic_vector(31 downto 0);
    signal PPS_ERROR : std_logic_vector(31 downto 0);

    type t_pci_fsm_state is (IDLE_S, BUSY_S, READ_S, CFGREAD_S, CFGWRITE_S);
    signal pci_fsm_state : t_pci_fsm_state;

begin -- behavioral

synpcix     : process(PCI_CLK,PCI_RST)
```

```
        constant DEVICE_ID           :std_logic_vector(15 downto 0) := X"9500";
        constant VENDOR_ID           :std_logic_vector(15 downto 0) := X"1204"; -- Lattice
        constant CLASS_CODE          :std_logic_vector(23 downto 0) := X"111000";
        constant REVISION_ID         :std_logic_vector(7 downto 0)  := X"01";
        constant SUBSYSTEM_ID        :std_logic_vector(15 downto 0) := X"0001"; -- Card identifier
        constant SUBSYSTEM_VENDOR_ID :std_logic_vector(15 downto 0) := X"BEBE"; -- Card identifier
        constant DEVSEL_TIMING       :std_logic_vector(1 downto 0)  := "01";    -- Fast

        constant MEMREAD     :std_logic_vector(3 downto 0) := "0110";
        constant MEMWRITE    :std_logic_vector(3 downto 0) := "0111";
        constant PCIXMEMREAD :std_logic_vector(3 downto 0) := "1000";
        constant PCIXMEMWRITE:std_logic_vector(3 downto 0) := "1001";
        constant CFGREAD     :std_logic_vector(3 downto 0) := "1010";
        constant CFGWRITE    :std_logic_vector(3 downto 0) := "1011";
        constant MEMREADMULT :std_logic_vector(3 downto 0) := "1100";
        constant DUALADDRCYC :std_logic_vector(3 downto 0) := "1101";
        constant MEMREADLINE :std_logic_vector(3 downto 0) := "1110";
        constant MEMWRITEAIN :std_logic_vector(3 downto 0) := "1111";

        variable addrhit: std_logic;
        variable address: std_logic_vector(5 downto 0);
        variable cbe_r  : std_logic_vector(3 downto 0);

        variable memen  : std_logic;
        variable fb2b   : std_logic;
        variable bar0   : std_logic_vector(19 downto 0); -- BAR0 4kB memory window
    -- PCI-X variables
        variable pcix_mode : std_logic;
        variable dev_num: std_logic_vector(4 downto 0);
        variable bus_num: std_logic_vector(7 downto 0);
        variable dpere  : std_logic;
        variable mmrbc  : std_logic_vector(1 downto 0);
        variable most   : std_logic_vector(2 downto 0);
begin

    PCI_REQ <= 'Z';
    PCI_GNT <= 'Z';
    PCI_PAR <= 'Z';
    PCI_PERR <= 'Z';
    PCI_SERR <= 'Z';

    if PCI_RST = '0' then
        bar0            := (others => '0');
        memen           := '0';
        fb2b            := '0';
        dev_num         := (others => '1');
        bus_num         := (others => '1');
        dpere           := '0';
        mmrbc           := (others => '0');
        most            := (others => '0');
        address         := (others => '0');
        pci_fsm_state   <= IDLE_S;
        addrhit         := '0';
```

```
        cbe_r           := (others => '0');
        ATTR_CLK        <= '0';
        ATTR_CLK2       <= '0';
        PCI_STOP        <= 'Z';
        PCI_DEVSEL      <= 'Z';
        PCI_TRDY        <= 'Z';
        PCI_AD          <= (others => 'Z');
        LATCH_I         <= '0';

        if (PCI_STOP = '0' or PCI_DEVSEL = '0' or PCI_TRDY = '0') then
            pcix_mode := '1';
        else
            pcix_mode := '0';
        end if;

    elsif PCI_CLK'event and PCI_CLK = '1' then
        case pci_fsm_state is
            when IDLE_S =>
                PCI_AD <= (others => 'Z');
                if PCI_FRAME = '0' then
                    if (PCI_CBE = CFGWRITE and PCI_IDSEL = '1' and PCI_AD(1 downto 0) = "00") then
                        if (pcix_mode = '1' and PCI_AD(10 downto 8) = "000") then
                            dev_num := PCI_AD(15 downto 11);
                            ATTR_CLK <= '1';
                        else
                            ATTR_CLK <= '0';
                        end if;
                        pci_fsm_state <= CFGWRITE_S;
                        PCI_DEVSEL <= '0';
                        PCI_TRDY <= '1';
                        PCI_STOP <= '1';
                    elsif (PCI_CBE = CFGREAD  and PCI_IDSEL = '1' and PCI_AD(1 downto 0) = "00") then
                        pci_fsm_state <= CFGREAD_S;
                        if pcix_mode = '1' then
                            ATTR_CLK <= '1';
                        else
                            ATTR_CLK <= '0';
                        end if;
                        PCI_DEVSEL <= '0';
                        PCI_TRDY <= '1';
                        PCI_STOP <= '1';
                    elsif (memen = '1' and PCI_AD(31 downto 12) = bar0) then
                        if(PCI_CBE = MEMREAD or PCI_CBE = PCIXMEMREAD or PCI_CBE = MEMREADMULT
                           or PCI_CBE = MEMREADLINE) then
                            PCI_DEVSEL <= '0';
                            PCI_TRDY <= '1';
                            PCI_STOP <= '1';
                            pci_fsm_state <= READ_S;
                            if address(1 downto 0) = "00" then
                                LATCH_I <= '1';
                            end if;
                            if pcix_mode = '1' then
                                ATTR_CLK <= '1';
```

```
                else
                    ATTR_CLK <= '0';
                end if;
                ATTR_CLK2 <= '1';
            else
                pci_fsm_state <= BUSY_S;
            end if;
        else
            pci_fsm_state <= BUSY_S;
        end if;
    else
        pci_fsm_state <= IDLE_S;
        PCI_DEVSEL <= 'Z';
        PCI_TRDY <= 'Z';
        PCI_STOP <= 'Z';
        LATCH_I <= '0';
    end if;
when BUSY_S =>
    PCI_AD <= (others => 'Z');
    PCI_DEVSEL <= 'Z';
    PCI_TRDY <= 'Z';
    PCI_STOP <= 'Z';
    if PCI_FRAME = '1' then
        pci_fsm_state <= IDLE_S;
    end if;

when READ_S =>
    LATCH_I <= '0';
    if ATTR_CLK = '1' then
        ATTR_CLK <= '0';
    elsif ATTR_CLK2 = '1' then
        ATTR_CLK2 <= '0';
        if address(1 downto 0) = "00" then
            PCI_AD <= NSEC_I;
        elsif address(1 downto 0) = "01" then
            PCI_AD <= NSEC_PPS_I;
        else
            PCI_AD <= PPS_ERROR;
        end if;
    elsif PCI_TRDY = '1' then
        PCI_TRDY <= '0';
    elsif PCI_FRAME = '1' then
        pci_fsm_state <= IDLE_S;
        PCI_AD <= (others => 'Z');
        PCI_DEVSEL <= '1';
        PCI_TRDY <= '1';
    end if;

when CFGREAD_S =>
    case address is
        when "000000" =>
            PCI_AD(31 downto 16) <= DEVICE_ID;
            PCI_AD(15 downto 0) <= VENDOR_ID;
```

```
            when "000001" =>
                PCI_AD(31 downto 27) <= (others => '0');
                PCI_AD(26 downto 25) <= DEVSEL_TIMING;
                PCI_AD(24 downto 16) <= "000110000";
                PCI_AD(15 downto 10) <= (others => '0');
                PCI_AD(9) <= fb2b;
                PCI_AD(8 downto 2) <= (others => '0');
                PCI_AD(1) <= memen;
                PCI_AD(0) <= '0';
            when "000010" =>
                PCI_AD(31 downto 8) <= CLASS_CODE;
                PCI_AD(7 downto 0) <= REVISION_ID;
            when "000100" =>
                PCI_AD(31 downto 12) <= bar0;
                PCI_AD(11 downto 0) <= (others => '0');
            when "001011" =>
                PCI_AD(31 downto 16) <= SUBSYSTEM_ID;
                PCI_AD(15 downto 0) <= SUBSYSTEM_VENDOR_ID;
            when "001101" =>
                PCI_AD(31 downto 8) <= (others => '0');
                PCI_AD(7 downto  0) <= X"40";
            when "010000" =>
                PCI_AD(31 downto 23) <= (others => '0');
                PCI_AD(22 downto 20) <= most;
                PCI_AD(19 downto 18) <= mmrbc;
                PCI_AD(17) <= '0';
                PCI_AD(16) <= dpere;
                PCI_AD(15 downto 0) <= X"0007";
            when "010001" =>
                PCI_AD(31 downto 16) <= X"0002";
                PCI_AD(15 downto 8) <= bus_num;
                PCI_AD(7 downto 3) <= dev_num;
                PCI_AD(2 downto 0) <= (others => '0');
            when others =>
                PCI_AD <= (others => '0');
            end case;
        if ATTR_CLK = '1' then
            ATTR_CLK <= '0';
        elsif PCI_TRDY = '1' then
            PCI_TRDY <= '0';
        elsif PCI_FRAME = '1' then
            pci_fsm_state <= IDLE_S;
            PCI_AD <= (others => 'Z');
            PCI_DEVSEL <= '1';
            PCI_TRDY <= '1';
        end if;
    when CFGWRITE_S =>
        PCI_AD <= (others => 'Z');
        if ATTR_CLK = '1' then
            bus_num := PCI_AD(7 downto 0);
            ATTR_CLK <= '0';
        elsif PCI_IRDY = '0' then
            case address is
```

```
                        when "000001" =>
                            memen := PCI_AD(1);
                            fb2b := PCI_AD(9);
                        when "000100" =>
                            bar0 := PCI_AD(31 downto 12);
                        when "001000" =>
                            dpere := PCI_AD(16);
                            mmrbc := PCI_AD(19 downto 18);
                            most := PCI_AD(22 downto 20);
                        when others =>
                    end case;
                    PCI_TRDY <= '0';
                elsif PCI_FRAME = '1' then
                    pci_fsm_state <= IDLE_S;
                    PCI_DEVSEL <= '1';
                    PCI_TRDY <= '1';
                end if;
        end case;
    end if;
end process synpcix;

    clkdivinst1 : clkdiv2
    port map (
        CLK => GPS_CLK,
        RESETN  => PCI_RST,
        CLK_OUT => GPS_CLK_2);

    clkdivinst2 : clkdiv2
    port map (
        CLK => GPS_CLK_2,
        RESETN  => PCI_RST,
        CLK_OUT => GPS_CLK_4);

    nsecinst1 : nsec_counter
    port map (
        CLK     => GPS_CLK,
        PCI_CLK     => PCI_CLK,
        PPS_PULSE   => PPS_PULSE,
        LATCH       => LATCH_I,
        NSEC        => NSEC_I,
        NSEC_PPS    => NSEC_PPS_I,
        PPS_OUT     => PPS_OUT_I);

    ledinst1 : led_control
    port map (
        CLK     => GPS_CLK_4,
        RESETN      => PCI_RST,
        PPS_PULSE   => PPS_PULSE,
        PLL_LOCK    => PLL_LOCK,
        LED1_GREEN  => LED1_GREEN,
        LED1_RED    => LED1_RED,
        LED2_GREEN  => LED2_GREEN,
        LED2_RED    => LED2_RED,
```

```
        PPS_ERROR   => PPS_ERROR);

end behavioral;
```

B.2 clkdiv2.vhd

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

entity clkdiv2 is

port (
    CLK,RESETN  : in  std_logic;
    CLK_OUT     : out std_logic);

end clkdiv2;

architecture behavioral of clkdiv2 is

    signal CLK_DIV_2    : std_logic;

begin -- behavioral
clkdiv_proc: process (CLK, RESETN) begin if
RESETN = '0' then -- async reset, active low
        CLK_DIV_2 <= '0';
    elsif (CLK'event and CLK = '1') then -- rising clock edge
        CLK_DIV_2 <= not CLK_DIV_2;
        CLK_OUT <= not CLK_DIV_2;
    end if;
end process clkdiv_proc;
end behavioral;
```

B.3 nsec_counter.vhd

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

entity nsec_counter is
port (
    CLK     : in  std_logic;
    PCI_CLK     : in  std_logic;
    PPS_PULSE   : in  std_logic;
    LATCH       : in  std_logic;
    NSEC        : out std_logic_vector(31 downto 0);
    NSEC_PPS    : out std_logic_vector(31 downto 0);
```

```
    PPS_OUT     : out std_logic);

end nsec_counter;

architecture behavioral of nsec_counter is

    signal PPS_PULSE_I  : std_logic;

begin -- behavioral

nsec_counter    : process(CLK,LATCH)
    variable nsec_i     : std_logic_vector(28 downto 0);
    variable pps_out_i  : std_logic;
begin
    if CLK'event and CLK = '1' then
        if PPS_PULSE = '1' then
            if PPS_PULSE_I = '0' then
                NSEC_PPS(31 downto 3) <= nsec_i;
                NSEC_PPS(2 downto 0) <= (others => '0');
                PPS_PULSE_I <= '1';
                pps_out_i :=  not pps_out_i;
            end if;
        elsif PPS_PULSE = '0' then
            PPS_PULSE_I <= '0';
        end if;

        nsec_i := nsec_i + 1;
        PPS_OUT <= pps_out_i;

    elsif CLK = '0' and LATCH = '1' then
        NSEC(31 downto 8) <= nsec_i(28 downto 5);
        NSEC(7 downto 0) <= (others => '0');
    end if;
end process nsec_counter;
end behavioral;
```

B.4 led_control.vhd

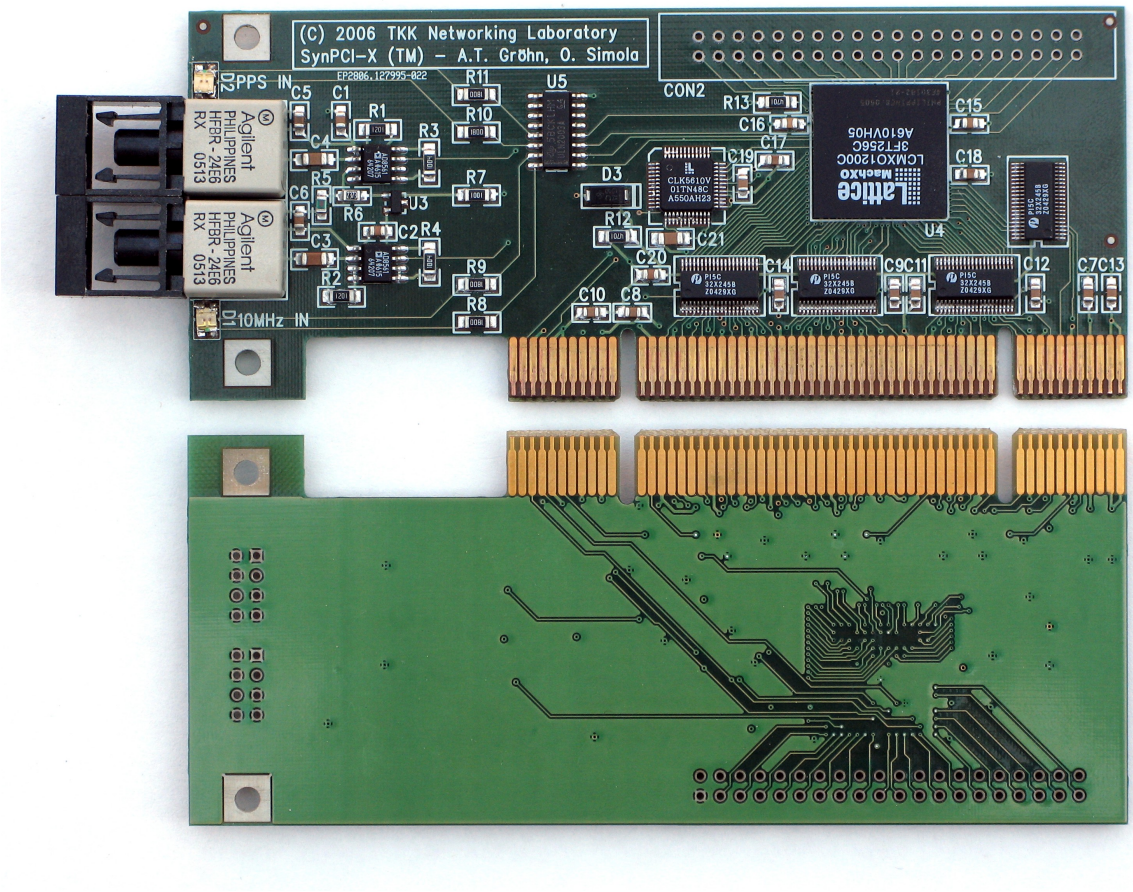
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity led_control is

port (
    CLK,RESETN  : in  std_logic;
    PPS_PULSE   : in  std_logic;
    PLL_LOCK    : in  std_logic;
    LED1_GREEN  : out std_logic;
    LED1_RED    : out std_logic;
```

```
    LED2_GREEN  : out std_logic;
    LED2_RED    : out std_logic;
    PPS_ERROR   : out std_logic_vector(31 downto 0));
end;

architecture behavioral of led_control is

begin -- behavioral

led_control : process(CLK,RESETN)
    variable counter        : std_logic_vector(25 downto 0);
    variable pps_error_b    : std_logic_vector(7 downto 0);
    variable pps_error_m    : std_logic_vector(7 downto 0);
    variable pps_missing    : std_logic;

begin
    if RESETN = '0' then
        LED1_RED    <= '1';
        LED2_RED    <= '1';
        LED1_GREEN  <= '0';
        LED2_GREEN  <= '0';
        PPS_ERROR   <= (others => '0');
        counter     := (others => '1');
        pps_error_b := (others => '0');
        pps_error_m := (others => '0');
        pps_missing := '1';
    elsif PLL_LOCK = '1' then
        LED1_RED    <= '1';
        LED1_GREEN  <= '0';
        pps_error_b := (others => '0');
        pps_error_m := (others => '0');
    elsif CLK'event and CLK = '1' then
        LED1_RED    <= '0';
        LED1_GREEN  <= '1';

        PPS_ERROR(15 downto 8) <= pps_error_b;
        PPS_ERROR(7 downto 0) <= pps_error_m;

        if PPS_PULSE = '1' then
            -- if PPS_PULSE is up between 50us - (1s-50us)
            if counter > 1563 and counter < 31248438 then
                pps_error_b := pps_error_b + 1;
            end if;
            pps_missing := '0';
            counter := (others => '0');
        end if;

        if counter(25) = '0' then
            counter := counter + 1;
        end if;

        -- we are running local oscillator something between 18Mhz and 26Mhz
        -- 125Mhz/4 = 31.25Mhz  31250313 = 1.000010016s = 1s + 10.016us
```

```
        if counter < 31250313 then
            LED2_RED <= '0';
        else
            LED2_RED <= '1';
            if pps_missing = '0' then
                pps_error_m := pps_error_m + 1;
                pps_missing := '1';
            end if;
        end if;

        if counter < 200000 then
            LED2_GREEN <= '1';
        else
            LED2_GREEN <= '0';
        end if;
    end if;
end process led_control;
end behavioral;
```

Liite C

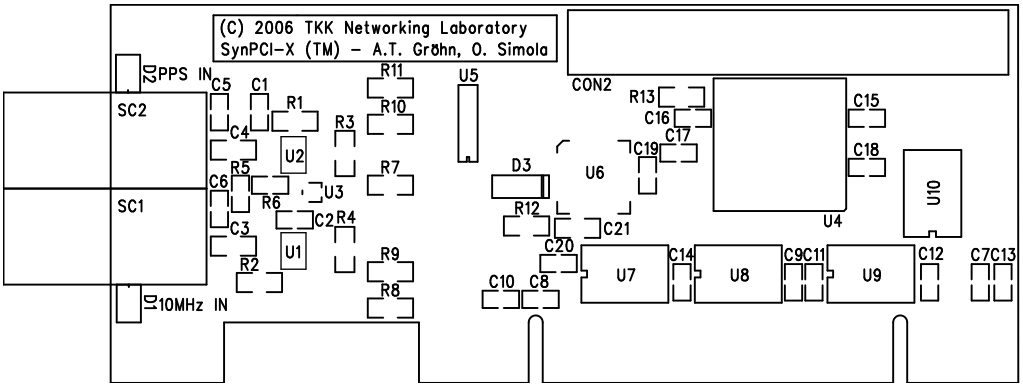
SynPCI-X –kortin valokuva



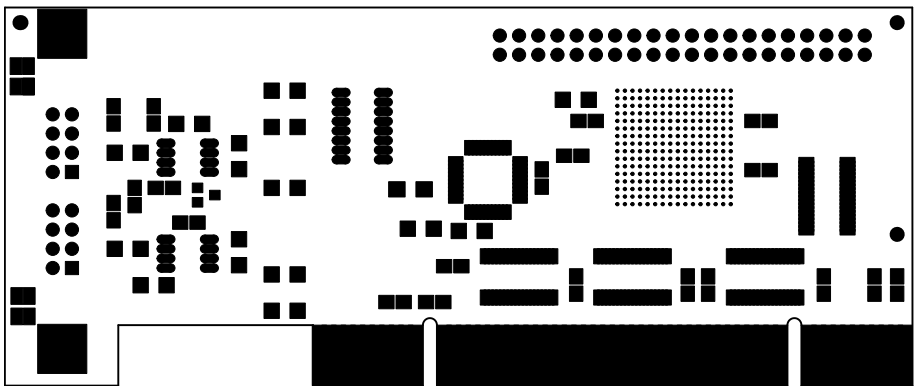
Kuva C.1: Valokuva SynPCI-X –kortista

Liite D

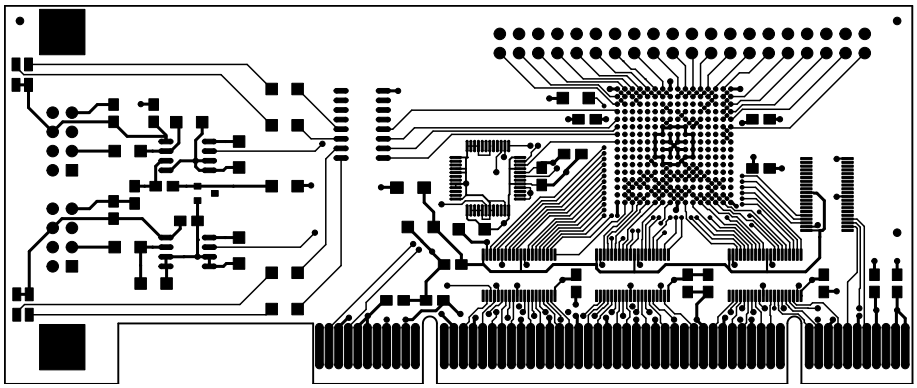
SynPCI-X –kortin piirilevykuvat



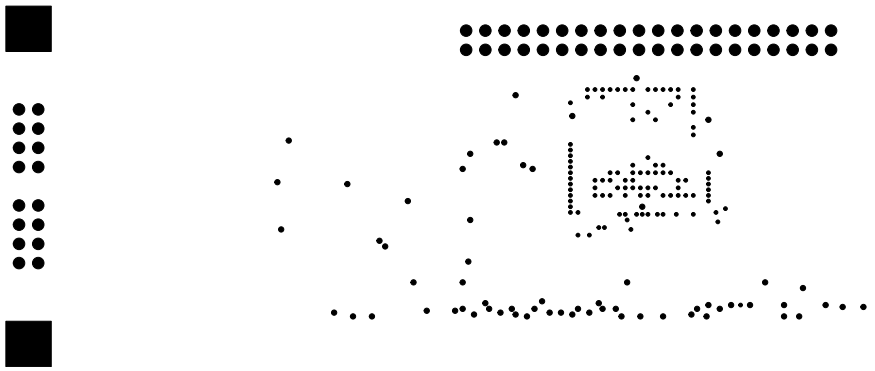
Kuva D.1: Pintakerroksen silkkipainokuva



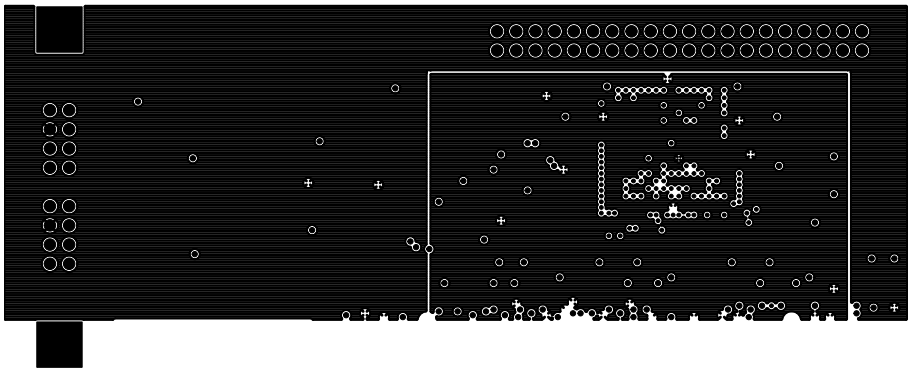
Kuva D.2: Pintakerroksen juotosmaski



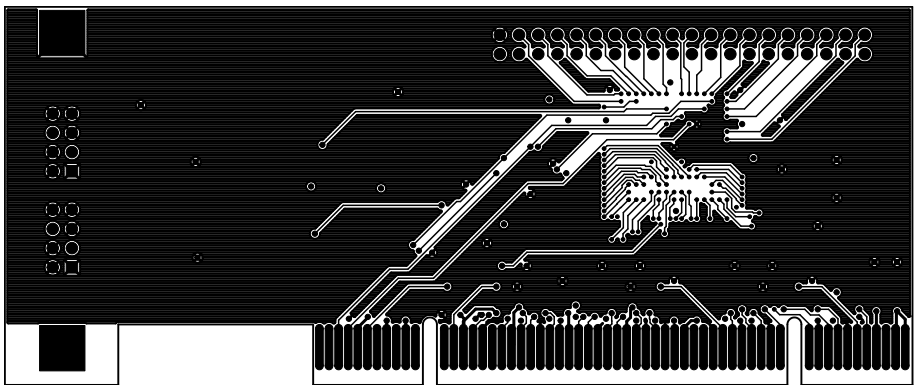
Kuva D.3: Pintakerros



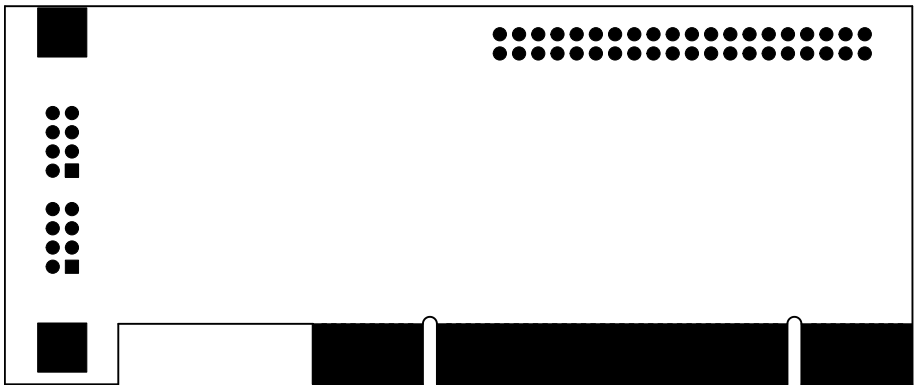
Kuva D.4: Invertoitu maakerros



Kuva D.5: Käyttöjännitekerros



Kuva D.6: Pohjakerros



Kuva D.7: Pohjakerroksen juotosmaski