TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JOUKO HAAPALUOMA

NOPEIDEN ANALOGIAMITTAUSTEN SUORITTAMINEN SULAUTETUSSA LINUX-JÄRJESTELMÄSSÄ

Diplomityö

Tarkastaja: professori Seppo Kuikka
Tarkastaja ja aihe hyväksytty
Teknisten tieteiden tiedekuntaneuvoston
kokouksessa 5. maaliskuuta 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
HAAPALUOMA, JOUKO: Nopeiden analogiamittausten suorittaminen sulautetussa Linux-järjestelmässä
Diplomityö, 56 sivua, 9 liitesivua
Huhtikuu 2014
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Seppo Kuikka
Avainsanat: analogia-digitaalimuunnos, sulautettu järjestelmä, Linux, reaaliaika-Linux, näytteistystaajuus, oikosiirto, suorituskyky

Linux-käyttöjärjestelmän käyttö kasvaa etenkin sulautetuissa järjestelmissä merkittävästi. Yleiskäyttöisyyden lisäksi Linuxille on kehitetty reaaliaikaominaisuuksia, joiden avulla Linuxia on alettu hyödyntää myös teollisuuden mittaus- ja ohjausjärjestelmissä. Wapice Remote Management (WRM) on Wapice Oy:n kehittämä kokonaisvaltainen teollisuuden etähallintajärjestelmä, joka sisältää palvelimen ja mittauksia suorittavia terminaalilaitteita. Terminaalilaitteilla käytetään sulautettua reaaliaika-Linuxia.

Tässä työssä tutkittiin nopeiden analogiamittausten suorittamista WRM-terminaalilaitteella. Analogiamittauksia voidaan WRM:ssä hyödyntää esimerkiksi moniin teollisuuden mittauksiin. Joissain tapauksissa analogiamittauksia halutaan suorittaa korkealla näytteistystaajuudella, jotta mittausten korkeataajuuksisetkin komponentit saadaan tallennettua. Työn tavoitteena oli tutkia, mitä vaihtoehtoja nopeiden analogiamittausten toteuttamiselle on WRM-terminaalilaitteessa käytetyssä sulautetussa Linuxissa. Lisäksi tutkittiin, miten toteutuksesta saadaan mahdollisimman luotettava ja tehokas, jotta WRM-terminaalilaite ei kuormitu liikaa suurillakaan näytteistystaajuuksilla.

Työn alkuosassa käsitellään yleisesti analogia-digitaalimuunnoksia, tutustutaan Linuxin laiteajurien yksityiskohtiin sekä käsitellään reaaliaika-Linuxien ominaisuuksia. Teoreettisen tarkastelun jälkeen pohditaan eri tapoja toteuttaa nopeat analogiamittaukset. Käytännössä toimivat ratkaisut rajattiin kahteen tapaan: keskeytyksiin perustuva ratkaisu sekä oikosiirtoon perustuva ratkaisu. Pääteltiin, että alkuperäisten vaatimusten mukaista 3 kilohertsin näytteistystaajuutta pystytään käyttämään molemmilla tavoilla. Keskeytyksiin perustuvan ratkaisun katsottiin olevan vähätöisempi, joten sillä toteutettiin yksinkertainen demototeutus. Keskeytyksiin perustuva ratkaisu on kuitenkin epäluotettava ja skaalautuu huonosti, joten yleiskäyttöinen toteutus päätettiin tehdä oikosiirolla.

Lopuksi verrattiin kummankin ratkaisun suorituskykyä. Tuloksien perusteella oikosiirtoon perustuvalla ratkaisulla pystytään analogiamuunninta käyttämään kaikilla WRM:n analogiatuloilla suurimmalla mahdollisella näytteistystaajuudella (lähes 200 kilohertsiä) luotettavasti ja tehokkaasti. Keskeytyksiin perustuva ratkaisu osoittautui erittäin heikoksi, kun näytteistystaajuutta aletaan kasvattaa. Suuri keskeytysten määrä aiheuttaakin ison prosessorikuorman, mutta vielä suurempi heikkous on näytteiden katoaminen, jota puolestaan oikosiirtoon perustuvalla ratkaisulla ei voi tapahtua. Johtopäätöksenä oikosiirtoon perustuvalla ratkaisulla saatiin kehitettyä luotettava ja tehokas nopeiden analogiamittausten toteutus osaksi WRM-järjestelmää.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
HAAPALUOMA, JOUKO: Fast analog measurements in embedded Linux environment
Master of Science Thesis, 56 pages, 9 Appendix pages
April 2014
Major: Automation Software Engineering
Examiner: Professor Seppo Kuikka
Keywords: analog-to-digital conversion, embedded system, Linux, real-time Linux, sampling frequency, DMA transfer, performance

The usage of the Linux operating system is increasing especially with embedded devices. Various Real-Time Linux solutions have also enabled Linux usage on industrial measurement and control systems. Wapice Remote Management (WRM) is a remote control and measurement system developed by Wapice Ltd. The WRM system consists of a server and measurement terminals. The WRM terminals use Real-Time Linux.

The purpose of this thesis was to research fast analog measurement implementation aspects for the WRM terminal. Analog measurements with the WRM system can be used in various industrial measurement applications. Analog measurements have to be performed with a high sampling rate if high frequency components of the measured signal need to be captured. The objective was to research which different approaches can be found to implement fast analog measurements for the Real-Time Linux system used by the WRM terminal. Another objective was to figure out how maximal performance and reliability can be achieved.

The first part of the thesis discusses analog-to-digital conversions, Linux device driver implementation details and Real-Time Linux aspects. The latter part presents different options to implement fast analog measurements. Generally two plausible options were found: interrupt based implementation and DMA (Direct Memory Access) based implementation. It was concluded that both implementations would meet the initial requirements for a 3 kilohertz sampling rate. The interrupt based implementation would be simpler and therefore a simple demo implementation was created using the interrupt based approach. However, the interrupt based implementation has some weaknesses. It is somewhat unreliable and the scalability is weak. Therefore the generic implementation was created based on the DMA approach.

Finally the performance of both implementations was compared. The results implicate that the DMA based implementation can achieve highest possible sampling rate (almost 200 kilohertz) with high reliability and performance. The interrupt based implementation suffers from significantly bad scalability. Even bigger weakness with the interrupt based implementation is the occasional loss of samples when the system is under load. As a conclusion, a highly reliable and powerful fast analog measurement implementation was created for the WRM system using the DMA based approach.

ALKUSANAT

Tämä diplomityö on kirjoitettu Wapice Oy:lle vuosien 2013 ja 2014 aikana. Haluan kiittää Wapice Oy:tä mielenkiintoisesta ja erittäin haastavasta diplomityöaiheesta sekä diplomityön tekemistä tukevasta työympäristöstä. Erityiset kiitokset haluan esittää professori Seppo Kuikalle työn tarkastamisesta sekä erinomaisista vinkeistä työn sisältöön liittyen.

Lisäksi esitän kiitokset Wapice Oy:n sulautetun segmentin päällikölle, Risto Pajulalle, erinomaisista teknisistä ohjeista, sekä Joonas Ihoselle oikolukemisesta sekä parannusehdotuksista.

Tampereella 24.4.2014

_____

Jouko Haapaluoma

SISÄLLYS

TERMIT JA LYHENTEET

Adeos	Virtualisointikerros Xenomaissa
Alijärjestelmä	Subsystem, sysfs-tiedostojärjestelmään kuuluvia rajapintakokoelmia
ARM	Advanced RISC Machines, prosessoriarkkitehtuuri
Bottom-half	Keskeytyskäsittelijän alaosa, jossa voidaan tehdä raskaita operaatioita
DMA Engine	Linuxin laitteistoriippumaton rajapinta oikosiirtoja varten
Dma_cyclic	Linux DMA Engine -rajapinnan jaksollinen oikosiirto-operaatio
Dma_pool	Mekanismi, jolla voidaan luoda paljon samankokoisia oikosiirtopuskureita
FIFO	First In First Out, menetelmä, jossa tietoalkioita käsitellään siinä järjestyksessä kuin ne ovat saapuneet
GPS	Global Positioning System, paikannusjärjestelmä
IEEE	Institute of Electrical and Electronics Engineers, kansainvälinen tekniikan alan järjestö
IIO	Industrial IO, sysfs-alijärjestelmä, joka on tarkoitettu teollisuuden mittaustarpeisiin
Interleaved_dma	Linux DMA Engine -rajapinnan oikosiirto-operaatio, jossa voidaan yhdistellä erityyppisiä puskureita ja siirtotapoja
Ioctl	Input/Output Control, mekanismi, jolla laiteajureille voidaan tehdä ohjauskomentoja Linuxissa
Järjestelmäkutsu	System call, käyttöjärjestelmän mekanismi, jolla käyttäjätilan prosessi voi siirtyä ydintilaan
Keskeytyskonteksti	Interrupt context, ydintila, jossa keskeytyskäsittelijät suoritetaan
Kfifo	FIFO-tyyppinen puskuri Linuxissa
Kmalloc	Linux-ytimen muistinvarausmekanismi, jolla voidaan varata fyysisesti jatkuvaa muistia
Käyttäjäprosessi	Prosessi, jota suoritetaan käyttäjätilassa
Käyttäjätila	User space, konteksti, jossa käyttäjäprosessit ajetaan

Käänteiskeskeytysongelma	Keskeytyskäsittelijä keskeyttää korkeamman prioriteetin prosessin suorituksen
Merkkilaiteajuri	Character device, yleisin käyttäjärajapinta Linuxin laiteajureilla
Muistinhallintayksikkö	MMU, Memory Management Unit
Mutex	Poissulkemismekanismi
Muutostiedosto	Patch, tiedosto, joka ilmaisee kahden tiedoston välisen eron
NOTHREAD	Lippu, jolla saadaan tavallinen keskeytyskäsittelijä käyttöön RT_PREEMPT-lisäosan kanssa
Nucleus	Xenomain reaaliaikaydin
Nyquistin teoreema	Signaalista tulee ottaa näytteitä taajuudella, joka on suurempi kuin kaksinkertainen alkuperäisessä signaalissa esiintyvään suurimpaan taajuuteen nähden
Oikosiirto	DMA, Direct Memory Access, tiedon kopiointi laitteistolla ilman suorittimen toimenpiteitä
POSIX	Portable Operating System Interface, yhtenäistetty käyttäjärajapinta Unix-järjestelmille
Pirstaloituminen	Muistin käyttö hajaantuu koko muistin alueelle niin, että vapaa muisti koostuu useista pienistä alueista
Prosessi/ydin-malli	Process/Kernel Model, käyttöjärjestelmäarkkitehtuuri, jossa käyttöjärjestelmä on ytimessä ja suoritettavat käyttäjäohjelmistot ovat prosesseja
Prosessikonteksti	Process context, ydintila, jossa prosessien palvelut suoritetaan ytimen sisällä
Reaaliaikalaajennos	Laajennos, jolla Linuxin rinnalle tuodaan reaaliaikaominaisuuksia
Reaaliaikakäyttäjärjestelmä	RTOS, Real-Time Operating System, käyttöjärjestelmä, joka soveltuu reaaliaikasovelluksiin
RT_PREEMPT	Lisäosa, joka pyrkii tekemään Linuxista täysin keskeytettävän
RTAI	Linuxin reaaliaikalaajennos
Scatter/Gather-puskuri	Puskuri, joka koostuu useista pienemmistä puskureista

SDRAM	Synchronous Dynamic Random-Access Memory, synkroninen dynaaminen luku- ja kirjoitusmuisti
Semafori	Muuttuja, jolla hallitaan pääsyä kriittisille alueille moniajoympäristössä
Sisäinen sirpaloituminen	Internal fragmentation, muistin hukkaaminen, kun käytetään väärän kokoisia muistialkioita
Slab-varaaja	Muistinhallintamenetelmä pirstaloitumisen ehkäisemiseksi Linuxissa
Slab-välimuisti	Slab-varaajan käyttämä vapaalista
Slave_sg	Linux DMA Engine -rajapinnan oikosiirto-operaatio, jossa hyödynnetään Scatter/Gather-puskureita
SMP	Symmetric Multiprocessing, moniprosessorijärjestelmä, jossa kaikkia prosessoriytimiä käytetään symmetrisesti yhdellä käyttöjärjestelmäinstanssilla
Softirq	Mekanismi, jolla keskeytyskäsittelijöiden alaosat toteutetaan
Spinlock	Keskeyttämätön poissulkemismekanismi Linux-ytimessä
Sysfs	Virtuaalinen tiedostojärjestelmä Linuxissa
Tasklet	Keskeytyskäsittelijän alaosa, joka suoritetaan mahdollisimman pian
Top-half	Keskeytyskäsittelijän yläosa, joka suoritetaan mahdollisimman nopeasti
USB	Universal Serial Bus, yleisin oheislaitteiden liitäntäväylä tietokoneissa
Unix	Laitteistoriippumaton käyttöjärjestelmä, josta on nykyään monia variantteja
Vapaalista	Free list, lista vapaista muistialueista
Vmalloc	Linux-ytimen muistinvarausmekanismi, jolla voidaan varata virtuaalisesti jatkuvaa muistia
Workqueue	Keskeytyskäsittelijän alaosa, joka ajetaan ydintilan prosessikontekstissa
Window Compare	Laitteiston ominaisuus, jolla voidaan tunnistaa, jos mitattu signaali ylittää tai alittaa asetetun raja-arvon

WRM	Wapice Remote Management, Wapice Oy:n kehittämä etähallintajärjestelmä
Vuoronnus	Scheduling, ajettavan prosessin vaihtaminen
Ydintila	Kernel space, konteksti, jossa ytimen koodi suoritetaan
Yleiskäyttöinen käyttöjärjestelmä	GPOS, General-Purpose Operating System, käyttöjärjestelmä, joka sopii yleisesti monenlaisiin käyttötarkoituksiin
Xenomai	RTAI-projektista haarautunut reaaliaikalaajennos, joka on tällä hetkellä suosituin Linuxin reaaliaikalaajennos

1 JOHDANTO

Wapice Remote Management (WRM) on Wapice Oy -yrityksen kehittämä kokonaisvaltainen etähallintajärjestelmä, joka sisältää palvelimen sekä etähallintalaitteet. Koko järjestelmä on Wapicen suunnittelema ja toteuttama, ja se sisältää elektroniikan sekä ohjelmiston. Tässä työssä keskitytään WRM-etähallintalaitteelle toteutettavaan menetelmään suorittaa jatkuvia nopeita analogiamittauksia.

Linux-käyttöjärjestelmän käyttö vaativissakin teollisuussovelluksissa on yleistymässä. Vaikka Linux on alun perin tarkoitettu ainoastaan yleiskäyttöiseksi käyttöjärjestelmäksi, nykyään sitä voidaan hyödyntää myös reaaliaikaisuutta vaativissa tehtävissä. WRM-etähallintalaitteessa on käytetty ARM-prosessoria (Advanced RISC Machines) sekä Linux-käyttöjärjestelmää. Jotta WRM-laite pystyisi vastaamaan teollisuussovelluksien asettamiin reaaliaikavaatimuksiin, siinä on käytetty Linuxin RT_PREEMPT-lisäosaa, joka parantaa Linuxin reaaliaikaominaisuuksia.

Tässä työssä on tarkoitus tutkia, miten pystytään toteuttamaan jatkuvia nopeita analogiamittauksia WRM-etähallintalaitteella siten, että voidaan saavuttaa korkea näytteistaajuus ilman kohtuutonta järjestelmän kuormitusta. Koska WRM-laite on suunniteltu yleiskäyttöiseksi, voidaan samanaikaisesti suorittaa useita erilaisia mittauksia esimerkiksi nopeista kenttäväylistä, kiihtyvyysanturista ja GPS-paikantimesta (Global Positioning System). Tämän vuoksi on tärkeää, että erilaiset mittaukset pystytään toteuttamaan prosessoriaikaa sekä järjestelmän resursseja säästäen. Tutkimuskysymyksiä tässä työssä ovat:

- *Millä tavoilla nopeat analogiamittaukset voidaan toteuttaa sulautetussa Linux-ympäristössä?*
- *Mitä laitteiston ja Linuxin palveluita voidaan hyödyntää, jotta toteutuksesta tulee mahdollisimman luotettava ja tehokas?*

Analogiamittauksen alkuperäiset vaatimukset saatiin eräältä asiakkaalta. Tarkoituksena on näytteistää analogiasignaaleja useiden kilohertsien taajuudella, ja tarkkailla näytteistetyn signaalin tilaa. Signaalille voidaan asettaa kriittinen raja, jonka ylitys aiheuttaa mittaustapahtuman. Tällöin signaalin näytteitä tulisi olla tallennettuna vakiomäärä ennen tapahtumahetkeä ja tapahtumahetken jälkeen. Työn tuloksena tulisi toteuttaa asiakkaan vaatimukset täyttävä ratkaisu, jota voidaan hyödyntää yleiskäyttöisenä ja skaalautuvana moduulina WRM-järjestelmässä.

Tässä työssä tutustutaan ensin Wapice Oy -yritykseen ja WRM-järjestelmään luvussa 2, käsitellään yleisesti analogiamittauksia sulautetuissa järjestelmissä luvussa 3 ja tutkitaan Linuxin tarjoamia rajapintoja sekä mekanismeja keskeytysten, oikosiirtojen sekä käyttäjärajapintojen toteuttamiseksi luvussa 4. Lisäksi luvussa 5 tutkitaan yleisesti

reaaliaika-Linuxia ja sen erityispiirteitä laiteajureissa. Seuraavaksi pohditaan eri vaihtoehtoja nopeiden analogiamittausten toteuttamiseksi luvussa 6 ja toteutetaan kaksi eri ratkaisua: yksinkertainen toteutus asiakkaalle demonstroitavaksi luvussa 7 sekä yleiskäyttöinen ja tehokkaampi ratkaisu luvussa 8. Lisäksi näiden kahden ratkaisun suorituskykyä vertaillaan luvussa 9. Yhteenveto esitetään luvussa 10.

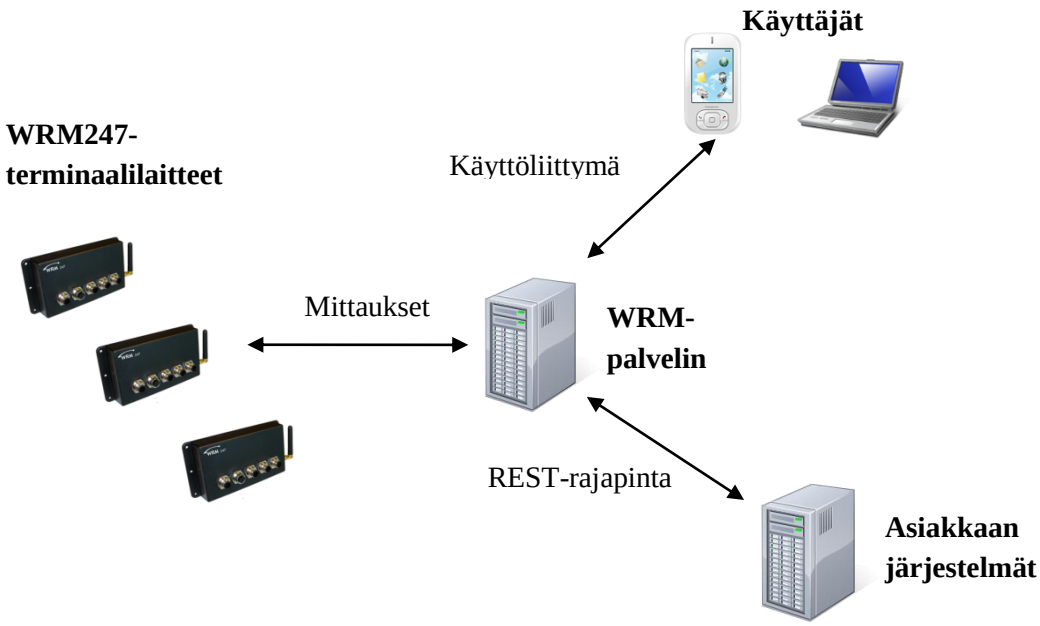
2 WAPICE OY JA WRM-ETÄHALLINTAJÄRJESTELMÄ

Wapice Oy on itsenäinen informaatioteknologian palveluyritys, jonka tavoitteena on vastata teollisuusyritysten vaativiin ohjelmistotarpeisiin. Wapice tarjoaa kokonaisvaltaista teknologiakumppanuutta, jolloin palvelu tarjotaan asiakkaan tarpeiden mukaan. Wapice voi hoitaa kokonaisen tuotteen elinkaaren suunnittelusta ja valmistuksesta ylläpitoon asti, tai osallistua asiakkaan tuotekehitysprojektiin tai liiketoimintaan tiiviissä yhteistyössä.

Wapicen organisaatio on jaettu kolmeen segmenttiin: liiketoimintaratkaisut, teollisuusjärjestelmät ja sulautetut järjestelmät. Liiketoimintaratkaisut-segmentti on erikoistunut teollisuuden liiketoimintaratkaisujen toteuttamiseen ja kehittämiseen. Ratkaisuissa hyödynnetään Wapicen omaa kehitystyötä kaupallisiin tuotteisiin sekä avoimen lähdekoodin projekteihin. Teollisuusjärjestelmät-segmentti keskittyy teollisuuden järjestelmiin ja alustoihin. Toteutuksissa käytetään esimerkiksi useita eri käyttöliittymäteknologioita ja kohteina ovat koneiden ja laitteiden ohjaukseen sekä tuotannon ja varaston hallintaan liittyvät ohjelmistot. Sulautetut järjestelmät -segmentti tarjoaa palveluita elektroniikkasuunnittelusta ja ohjelmoinnista valmiiseen tuotteeseen asti. Teknologioina käytetään uusimpia prosessoriarkkitehtuureja, kenttäväyliä, reaaliaikakäyttöjärjestelmiä sekä sulautettua Linuxia. [1.]

2.1 WRM-järjestelmä

WRM on Wapicen kehittämä etähallintaratkaisu, jonka tavoitteena on tarjota kokonaisvaltainen etähallintapalvelu teollisuuden mittaus- ja valvontatarpeisiin. WRM on kehitetty modulaariseksi ja yleiskäyttöiseksi, jotta sen käyttöönotto olisi helppoa ja suoraviivaista mahdollisimman monissa teollisuussovelluksissa. WRM koostuu palvelimesta sekä terminaalilaitteista. WRM-järjestelmä on esitetty kuvassa 2.1.



Kuva 2.1: *WRM-järjestelmä*

WRM247-terminalilaitteen tehtävä on suorittaa mittauksia, jotka lähetetään palvelimelle. Palvelimen käyttöliittymällä voidaan konfiguroida terminaalit ja tarkastella mitattuja tietoja. Käyttöliittymä on optimoitu kosketusnäyttölaitteita varten. WRM-palvelimella on lisäksi REST-rajapinta (Representational State Transfer), jonka avulla WRM-järjestelmä voidaan liittää asiakkaan ulkoisiin järjestelmiin. [2.]

2.2 WRM247-etähallintalaite

WRM247-etähallintalaite on ARM-prosessoriin perustuva edullinen sulautettu laite, joka pyrkii tarjoamaan yleisimmät teollisuudessa tarvittavat liitynnät ja mittaukset. Laite on kokonaan Wapicen suunnittelema ja tuottama. Laitteen päivitetty WRM247Plus-versio esitetään kuvassa 2.2.

Kuva 2.2: *WRM247Plus-laite*

WRM247 käyttää Atmel AT91SAM9260-mikrokontrolleria, joka sisältää ARM9-prosessoriytimen. Laitteessa on NAND-flashmuistia 128 megatavua ja SDRAM-muistia (Synchronous Dynamic Random-Access Memory) 64 megatavua. WRM247Plus on laitteen päivitetty versio, joka sisältää Atmel ATSAMA5D35-mikrokontrollerin, jossa on ARM Cortex A5 -prosessoriydin, 256 megatavua NAND-flashmuistia, 256 megatavua SDRAM-muistia sekä enemmän liityntöjä. Molempien laiteversioiden ominaisuudet esitetään taulukossa 2.1.

Taulukko 2.1: *WRM247-laitteiden ominaisuudet [3; 4]*

Ominaisuus	**WRM247**	**WRM247Plus**
Prosessori	ARM9 200 MHz	ARM Cortex A5 536 MHz
NAND-muisti	128 MB	256 MB
SDRAM-muisti	64 MB	256 MB
GPRS/3G	GPRS	GPRS/3G
GPS	1x	1x
Kiihtyvyysanturi	1x	1x
WiFi/Bluetooth	-	1x
Ethernet	1x	2x
CAN	2x	2x
RS-232	1x	1x
RS-485	1x	1x
1-wire	1x	1x
USB	1x	1x
Digital OUT	4x	4x
Digital IN	4x	4x
Analog IN	2x	2x
Virtaviestimittaus	2x	2x

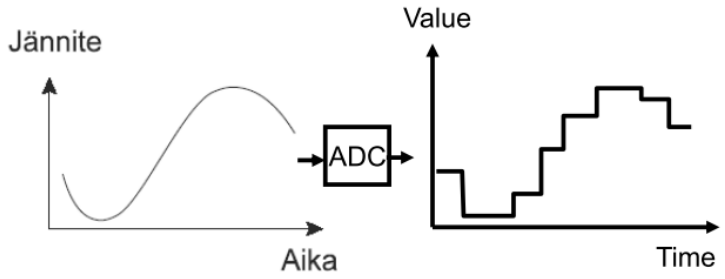
WRM247-laitteet sisältävät kaksi analogiasisääntuloa, joiden jännitealue on 0-32 volttia, ja kaksi virtaviestimittausta. Näitä neljää analogiasisääntuloa mitataan mikrokontrollerin sisäisellä analogia-digitaalimuuntimella. Tässä diplomityössä käytetään testilaitteena ainoastaan uutta WRM247Plus-laitetta.

3 ANALOGIA-DIGITAALIMUUNNOKSET SULAUTETUISSA JÄRJESTELMISSÄ

Sulautetuissa järjestelmissä tyypillisesti tehdään erilaisia mittauksia, joiden perusteella suoritetaan algoritmeja ja tuotetaan ulostulo tai ohjaus. Monet sähköiset anturit ilmaisevat mitattavan suureen arvon joko jännitteenä tai virtana. Sulautetuissa järjestelmissä kuitenkin logiikka toimii täysin digitaalitekniikalla, jolloin tarvitaan analogia-digitaalimuunnosta antureiden arvon määrittämiseksi. Analogia-digitaalimuunnokset ovatkin sulautetuissa järjestelmissä erittäin yleinen tapa suorittaa mittauksia. [5, s. 1-10.]

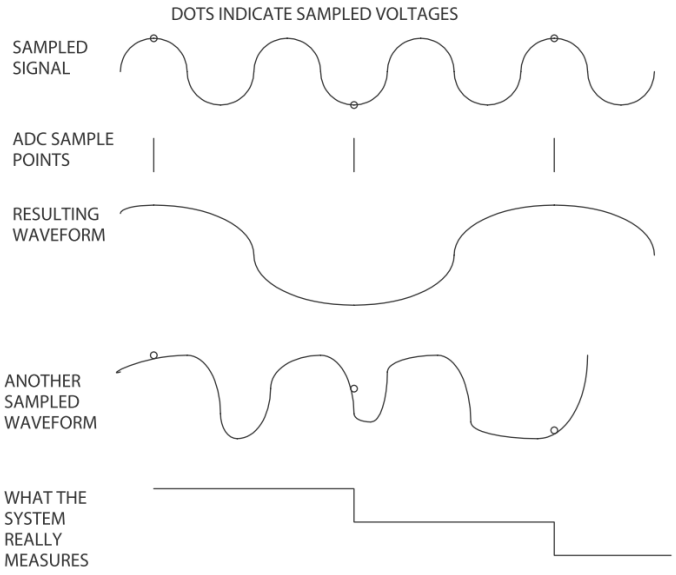
3.1 Analogia-digitaalimuunnos

Analogia-digitaalimuunnoksen tavoite on saada muutettua jatkuva analoginen signaali diskreeteiksi digitaalisiksi arvoiksi. Tällöin analogista signaalia näytteistetään tasaisin väliajoin ja muodostetaan analogista signaalia muistuttava epäjatkuva joukko diskreettejä arvoja. [6, s. 410-421.] Tätä havainnollistetaan kuvassa 3.1.



***Kuva 3.1**: Analoginen signaali digitaaliseksi [7, s. 5]*

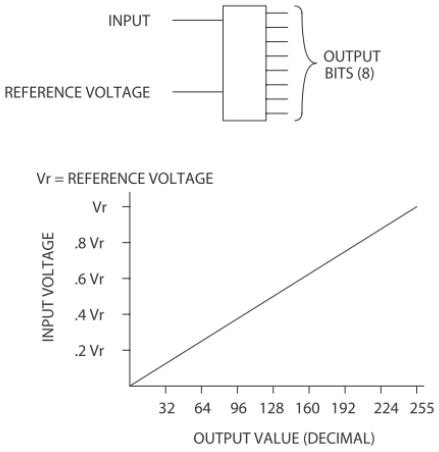
Näytteistystaajuuden valinnassa täytyy huomioida myös mitattavan signaalin taajuusominaisuudet. Jos näytteistystaajuus ei ole riittävän suuri, mitattava signaali laskostuu. Tällöin mittaustulos ei sisällä mitattavan signaalin oikeita taajuuskomponentteja. Nyquistin teoreeman mukaan näytteistystaajuuden tulisi olla vähintään kaksinkertainen mitattavan signaalin taajuuteen verrattuna, jotta alkuperäinen signaali voitaisiin rekonstruoida näytteiden perusteella [8]. Näytteistystaajuus ja laskostuminen on esitetty kuvassa 3.2.



***Kuva 3.2**: Näytteistystaajuus ja laskostuminen [5, s. 12]*

Yleensä analogiamittausten tavoitteena on saada suuri tarkkuus, korkea näytteistystaajuus sekä alhainen hinta. Kaikkia näitä ei kuitenkaan voida saavuttaa, joten käytännössä näistä ominaisuuksista täytyy valita sovelluskohteeseen parhaiten sopiva kompromissi. Tarkkuuden kasvattaminen hidastaa toimintaa tai aiheuttaa lisää kustannuksia. [5, s. 17-28.]

Yleisesti analogia-digitaalimuunninta voidaan ajatella mustana laatikkona, jonka sisääntulona on mitattava jännite sekä referenssijännite, ja ulostulona on signaalin digitaalinen arvo. Mittaus suoritetaan aina vertaamalla mitattavaa jännitettä referenssijännitteeseen ja tuloksena saatu digitaalinen arvo ilmoittaa, miten suuri mitattu arvo on referenssijännitteeseen verrattuna. Tästä syystä referenssijännite rajoittaa, miten suurta jännitettä enintään voidaan mitata. [5, s. 13-16.] Kuvassa 3.3 esitetään yksinkertainen periaatekuva analogia-digitaalimuuntimesta.



***Kuva 3.3**: Analogia-digitaalimuuntimen periaate [5, s. 15]*

Analogia-digitaalimuuntimia on myös useita erityyppisiä hieman eri ominaisuuksilla: Flash, Successive approximation, Dual slope, Pipeline ja Sigma-delta [9]. Eri muunnintyypit sekä niiden keskeiset ominaisuudet on esitetty taulukossa 3.1.

Taulukko 3.1: *Analogia-digitaalimuuntimien eri tyypit [9]*

Tyyppi	**Keskeiset ominaisuudet**
Flash	Todella suuri näytteistystaajuus, suurehko tehonkulutus
Successive approximation	Keskinkertainen resoluutio 8-16 bittiä, näytteistystaajuus alle 5 MHz, alhainen tehonkulutus ja koko
Dual slope	DC-signaalin valvontaan, korkea resoluutio, alhainen tehonkulutus, alhainen kohina
Pipeline	Korkea näytteistystaajuus 1-100 MHz, resoluutio 8-16 bittiä, keskinkertainen tehonkulutus
Sigma-delta	Korkea resoluutio, alhainen tai keskinkertainen nopeus, digitaalinen suodatus parantaa kvantisointivirhettä

Eri analogia-digitaalimuunnintyyppien toteutus eroaa toisistaan merkittävästi, mutta tämän diplomityön kannalta niiden tarkempi vertailu ei ole tarpeellista, sillä toteutuksessa käytetään WRM247-laitteen mikrokontrollerin sisäistä analogia-digitaalimuunninta, joka on pipeline-tyyppinen, 12-bittinen, 12-kanavainen sekä tukee enintään 1 MHz näytteistystaajuutta [10, s. 1618].

3.2 Tarkkuus

Analogia-digitaalimuuntimen bittien määrä sekä referenssijännite määräävät muunnoksen resoluution, jolla tarkoitetaan pienintä mitattavaa jännitteen muutosta. Kun bittejä lisätään, saadaan enemmän mitattavia arvoja ja resoluutio suurenee. Toisaalta myös referenssijännitettä laskemalla voidaan saada resoluutiota suurennettua, mutta tällöin mitattava jännitealue pienenee. [5, s. 16-17.]

Korkea resoluutio ei kuitenkaan tarkoita, että mittaus olisi tarkka. Kytkennöissä on komponentteja, joilla on tietyt toleranssit. Lisäksi ympäristötekijät, kuten lämpötila, voivat vaikuttaa kytkentöihin aiheuttaen epätarkkuutta tuloksiin. Tämän ongelman ratkaisemiseksi täytyy mittaukselle suorittaa kalibrointi, jos äärimmäistä tarkkuutta vaaditaan. Tällöin yleensä syötetään mittaukseen tunnettuja arvoja ja verrataan näitä mittaustuloksiin. Sulautetuissa järjestelmissä kalibrointi voidaan toteuttaa helposti ohjelmallisesti, jolloin muistiin tallennetaan korjauskertoimia, joilla mittaustulos saadaan eri toimintapisteissä korjattua todellisuutta vastaavaksi. [5, s. 233-234.]

3.3 Käyttökohteita

Analogia-digitaalimuunnosta sovelletaan nykyaikaisessa digitaalitekniikassa yksinkertaisiinkin asioihin, kuten äänen tallennukseen. Sulautetuissa järjestelmissä kuitenkin analogia-digitaalimuunnosta tarvitaan pääsääntöisesti erilaisten anturien arvojen määrittämistä varten. Sovelluskohteesta riippuen antureita voidaan käyttää hyvin moniin eri tarkoituksiin. Esimerkiksi teollisuussovelluksissa anturi on yleensä osana säätöpiiriä, jolloin mitataan säädetyn suureen hetkellistä arvoa. Anturit mittaavat esimerkiksi lämpötilaa, painetta, nestetasoa tai virtausta [11]. Toisaalta taas käyttöliittymäsovelluksessa voidaan kosketusnäytön resistiivisellä anturilla mitata kosketuksen paikkaa ja painetta [10, s. 1630-1637].

Tyypillisesti anturit ilmaisevat mittaamansa suureen jännitteenä tai virtana. Teollisuuskäytössä käytetään tyypillisesti virtaviestillä toimivia antureita. Tällöin anturin läpi kulkeva virta (yleensä välillä 4 - 20 milliampeeria) ilmaisee mitattavan suureen arvon. Virtaviestin käyttö mahdollistaa pitkät johtimet, sillä ylimääräisten jännitehäviöiden muodostuminen ei vääristä tulosta. Virran mittaus toteutetaan käytännössä virranmittausvastuksella, jolloin vastuksen jännitehäviön perusteella voidaan laskea sen läpi kulkeva virta. Virtaviestikytkentä sisältää anturin, virtalähteen sekä mittauksen. [12.] Tyypillinen virtaviestikytkentä on esitetty kuvassa 3.4.

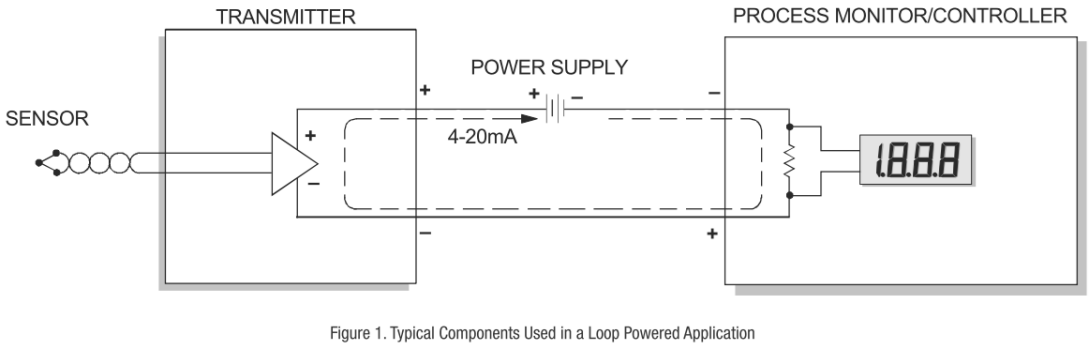


Figure 1. Typical Components Used in a Loop Powered Application

***Kuva 3.4**: Tyypillinen virtaviestikytkentä [12]*

Koska teollisuusantureissa jännitetasot ovat tyypillisesti välillä 0 - 24 volttia [12], tarvitaan sulautetuissa järjestelmissä erityisiä sovituskytkentöjä, joilla jännitetaso saadaan sovitettua analogia-digitaalimuuntimelle sopivaksi. Yksinkertaisimmillaan tämä voi tarkoittaa jännitteenjakoa, mutta suurta nopeutta tai tarkkuutta vaativissa mittauksissa tarvitaan vahvistinpiiriä ja huolellista suunnittelua. [5, s. 47-56, 225-241.]

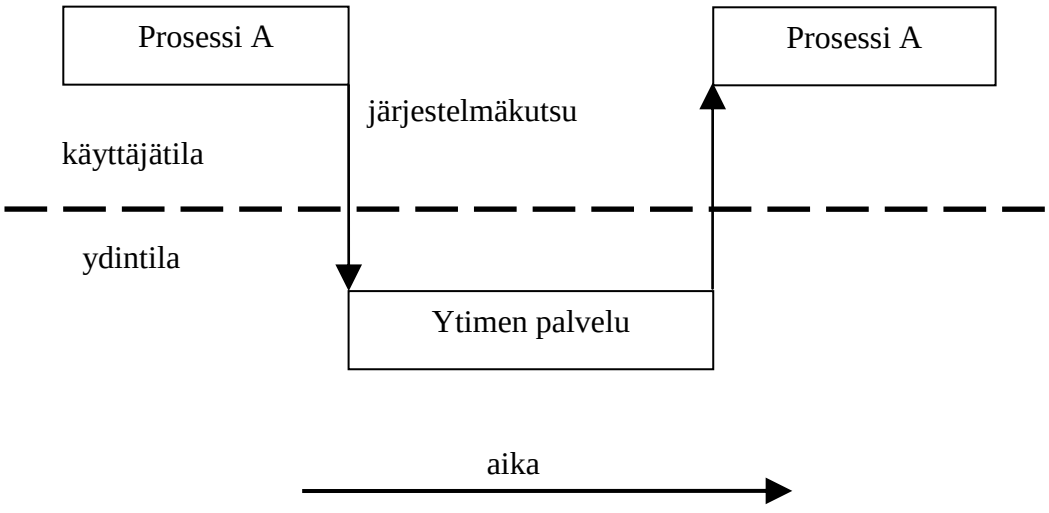
4 LAITEAJURIT LINUXISSA

Linux on Unix-tyyppinen käyttöjärjestelmäydin, jonka kehittämisen aloitti Linus Torvalds vuonna 1991. Nykyään Linuxia kehitetään aktiivisesti satojen kehittäjien voimin ympäri maailman. Linuxin erikoisuus on, että se hyödyntää avoimen lähdekoodin lisenssiä, jolloin sen lähdekoodi on kaikille vapaasti saatavilla. [13, s. 1.]

Linux-ydin sisältää suurehkon määrän valmiita laiteajureita, joita ytimen kehittäjät ylläpitävät. Laitteelle sopivaa ajuria ei kuitenkaan välttämättä ole saatavilla tai se ei tarjoa sovelluskohteen kannalta sopivia palveluita. Tällöin joudutaan tekemään oma laiteajuri. Laiteajureita varten on toteutettu useita rajapintoja, joiden avulla voidaan toteuttaa yleisiä laiteajurien tarvitsemia ominaisuuksia, kuten keskeytyskäsittelijät, puskurit, oikosiirrot (DMA, Direct Memory Access) ja käyttäjärajapinnat [14, s. 1-2].

4.1 Laiteajurin käyttäminen

Linux perustuu prosessi/ydin -malliin (Process/Kernel Model), jossa suorittimella on kaksi eri suoristustilaa: käyttäjätila (User space) ja ydintila (Kernel space). Normaalit ohjelmat suoritetaan yleensä käyttäjätilassa erillisinä prosesseina ja ydintilaan siirrytään silloin, kun prosessi tarvitsee ytimen palveluita tai ydin suorittaa omia toimintojaan. Lisäksi käyttäjätilassa ei pystytä käsittelemään suoraan ytimen muistialueita, jolloin ytimen palvelut ja toiminnot on eristetty käyttäjätilasta. Kun käyttäjätilan prosessi tarvitsee ytimen palveluita, se käyttää järjestelmäkutsua (System call), jolloin suoritus siirtyy ydintilaan. [13, s. 19-21.] Kuva 4.1 esittää järjestelmäkutsun toimintaa.



Kuva 4.1: *Järjestelmäkutsun toiminta*

Laiteajurit yleisesti hallitsevat ohjaamaansa laitteistoa alimmalla tasolla. Tämä sisältää esimerkiksi laitteen rekisterien lukua ja kirjoitusta. Tästä syystä Linuxissa laiteajurit suoritetaan ydintilassa. Koska laiteajurit suoritetaan ydintilassa, prosessit joutuvat käyttämään laiteajureita järjestelmäkutsuilla. Laiteajureita varten onkin kehitetty useita käyttäjärajapintoja, jotka voivat hyödyntää useita eri järjestelmäkutsuja. [14, s. 2-8.]

4.2 POSIX-rajapinta

POSIX (Portable Operating System Interface) on IEEE:n (Institute of Electrical and Electronics Engineers) standardikokoelma, joka määrittelee Unix-käyttöjärjestelmien ohjelmointirajapintoja. Tavoitteena on ollut luoda yhtenäisiä rajapintoja, jotta ohjelmat olisivat lähdekooditasolla siirrettävissä eri käyttöjärjestelmille. Käytännössä tällöin käyttöjärjestelmäkohtaiset järjestelmäkutsut piilotetaan yhtenäisen POSIX-rajapinnan taakse. Ensimmäinen POSIX-standardi IEEE Std 1003.1-1988 julkaistiin vuonna 1988 ja uusin voimassa oleva versio julkaistiin vuonna 2008. [15, s. 7-8.]

POSIX koostuu kolmesta osasta: POSIX.1, POSIX.1b ja POSIX.1c. POSIX.1 sisältää useimpien käyttöjärjestelmien tarjoamat ydinpalvelut, kuten prosessienhallinta, signaalit ja tiedostonkäsittely. POSIX.1b on standardin laajennos, joka lisää reaaliaikaominaisuuksia käyttöjärjestelmän rajapintaan. Tähän kuuluu esimerkiksi prioriteetteihin perustuva vuoronnus (Scheduling), kellot ja ajastimet, semaforit sekä jaettu muisti. Standardin toinen laajennos (POSIX.1c) lisää rajapintaan säikeiden käsittelyn. [16.]

4.3 Rajapinnat käyttäjätilaan

Jotta sovellukset voisivat käyttää laiteajuria, täytyy laiteajurin toteuttaa jokin käyttäjärajapinta. Yleisin käyttäjärajapinta on merkkilaiteajuri (Character device), joka tarjoaa POSIX-rajapinnan mukaiset tiedostonkäsittelymetodit: open, close, read ja write. Nämä metodit käyttävät taustalla Linux-ytimen vastaavia järjestelmäkutsuja, jotka päätyvät lopulta laiteajuriin. [14, s. 42-70.]

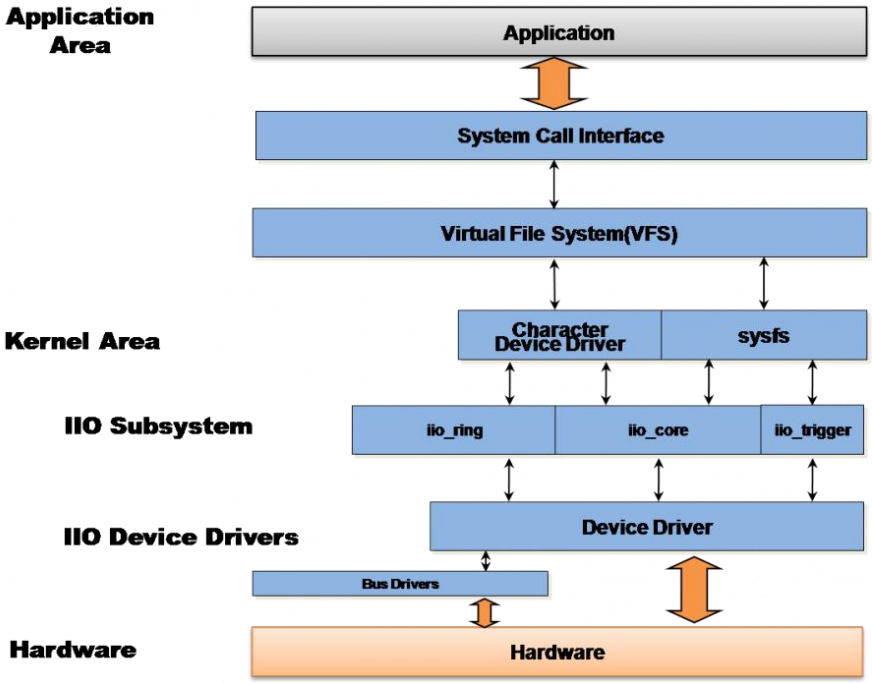
Tässä rajapinnassa ideana on, että laiteajuri on kuten tavallinen tiedostojärjestelmän tiedosto, jota voidaan lukea ja kirjoittaa merkki kerrallaan. Useimmat laiteajurit välittävät enimmäkseen tietoa joko laitteelta käyttäjälle tai käyttäjältä laitteelle, jolloin tiedoston luku- ja kirjoitusoperaatiot sopivat tähän hyvin. [14, s. 6.] Laiteajurit voivat kuitenkin sisältää paljon muutakin toiminnallisuutta, joka vaatii monimutkaisempaa rajapintaa. Tämän vuoksi merkkilaiteajurit sisältävät myös ioctl-metodin (Input/Output control), jolla voidaan toteuttaa mielivaltaisia komentoja tai pyyntöjä laiteajurille. [14, s. 135-137.]

Merkkilaiteajurit on kuitenkin joissain tapauksissa koettu hieman hankaliksi, joten on kehitetty myös vapaamuotoisempia rajapintoja Linuxin virtuaaliseen sysfs-tiedostojärjestelmään. Sysfs-tiedostojärjestelmän tarkoitus on esittää Linux-järjestelmän laitteet, ajurit ja kommunikaatiomenetelmät hierarkkisena rakenteena. Laitteet on jaoteltu ominaisuuksiensa mukaan erilaisiin alijärjestelmiin (Subsystem), jotka ovat käytän-

nössä hakemistoja sysfs-tiedostojärjestelmässä. Alijärjestelmät voivat sisältää erilaisiin käyttötarkoituksiin räätälöityjä rajapintoja. [17.]

4.4 Industrial IO -alijärjestelmä

Industrial IO -alijärjestelmä (IIO) on sysfs-tiedostojärjestelmään kehitetty rajapinta teollisuussovelluksissa tarvittavia tiedonkeräystarpeita varten. Käyttökohteita ovat esimerkiksi: analogia-digitaalimuuntimet, kiihtyvyysanturit, paikannuslaitteet ja lämpötila-anturit. [18.] IIO-alijärjestelmä on esitetty kuvassa 4.2.



Kuva 4.2: *Linux IIO-alijärjestelmän rakenne [18]*

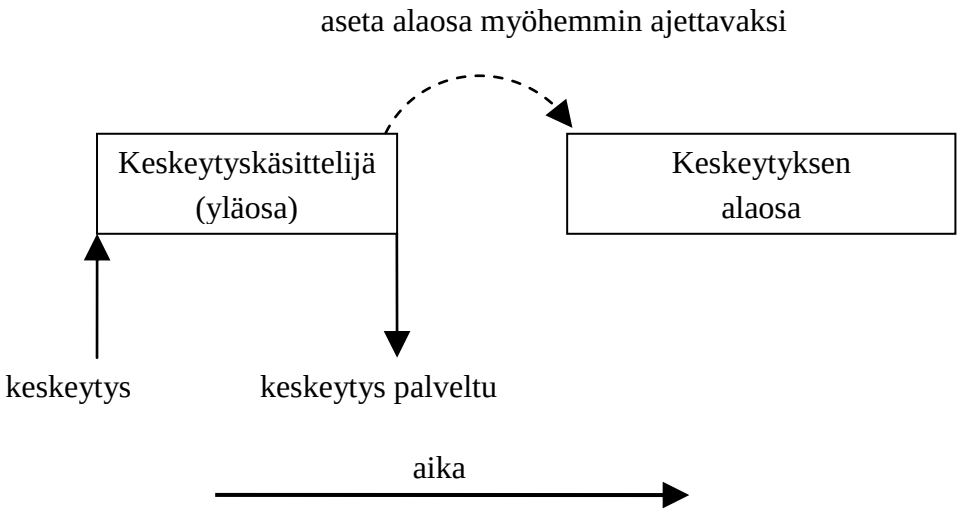
Kuvassa 4.2 esitetty *IIO Subsystem* on eräänlainen ohjelmistokehys, joka tarjoaa käyttäjille IIO-alijärjestelmän rajapinnan. Rajapinta on jaettu sekä merkkilaiteajuriin että sysfs-rajapintaan. Jokaista laitetta kohden voidaan toteuttaa *IIO Device Driver*, eli laiteajuri, joka hyödyntää IIO-alijärjestelmän ohjelmistokehystä. Tällä tavoin voidaan helposti toteuttaa uusia laiteajureita siten, että ne kaikki tarjoavat samanlaisen, standardin rajapinnan käyttäjille.

IIO-alijärjestelmä tarjoaa valmiin rengaspuskuritoteutuksen (*iio_ring),* jonne voidaan tallentaa mittauksia sekä niiden aikaleimoja. IIO-alijärjestelmä tarjoaa myös erilaisia liipaisimia (*iio_trigger),* jotka määrittävät, milloin mittaus tallennetaan rengaspuskuriin. Liipaisimia voidaan määrittää laitekohtaisesti. Näitä voivat olla esimerkiksi ohjelmistolla aiheutettu liipaisu tai ajastinliipaisu. [19.]

4.5 Keskeytysten käsittely

Useimmat oheislaitteet hyödyntävät prosessorin keskeytyksiä. Sen vuoksi laiteajureissa pitää yleensä toteuttaa laitteelle keskeytyskäsittelijä. [14, s. 258.] Linux hallitsee alustakohtaiset erot keskeytyksissä ja tarjoaa yhtenäisen sekä laitteistoriippumattoman rajapinnan keskeytysten hallitsemiseksi [20]. Laiteajuri voi rekisteröidä itselleen keskeytyskäsittelijän kutsumalla funktiota *request_irq()*, jolle annetaan laitteistokohtainen keskeytysnumero, osoitin keskeytyskäsittelijään, nimi sekä keskeytyslippuja [14, s. 259-260].

Keskeytyskäsittelijät suoritetaan erityisessä keskeytyskontekstissa (Interrupt context), jolloin ei voida käyttää kaikkia ytimen palveluita (kuten nukkumista). Lisäksi niiden suoritusta ei voida keskeyttää. Koska keskeytyskäsittelijä nimensä mukaisesti aina keskeyttää jonkin muun toiminnon suorituksen, tulisi keskeytyskäsittelijöistä aina saada mahdollisimman lyhyita ja nopeita. Useimmat keskeytyskäsittelijät vaativat kuitenkin melko monimutkaisia operaatioita, joiden suorittaminen olisi liian hidasta keskeytyskäsittelijässä. Tämän vuoksi käytännöksi on muodostunut tapa, jossa keskeytyskäsittelijä jaetaan kahteen osaan: yläosaan (Top-half) ja alaosaan (Bottom-half). [21, s. 115-123.] Keskeytyskäsittelijän jakaminen kahteen osaan esitetään kuvassa 4.3.



***Kuva 4.3**: Keskeytyskäsittelijän jakaminen ylä- ja alaosaan*

Yläosan tarkoitus on suorittaa ainoastaan laitteen toiminnan kannalta välttämättömät toiminnot nopeassa keskeytyskäsittelijässä. Näitä ovat esimerkiksi rekisterien luku ja kirjoitus sekä lippujen nollaus. Alaosan tarkoitus on suorittaa laajemmat algoritmit, kuten tiedon kopiointi ja muokkaus sekä valmistelu käyttäjärajapintaa varten. Yläosa asettaa alaosan ajettavaksi jonain myöhempänä ajanhetkenä, jolloin yläosan suoritusaika jää mahdollisimman lyhyeksi. [21, s. 115-116.]

4.6 Keskeytyskäsittelijän alaosat

Yleisimmät tavat keskeytyskäsittelijän alaosan toteuttamiseen ovat tasklet, softirq ja workqueue. Softirq on alimman tason mekanismi, jonka varaan esimerkiksi myös tasklet on toteutettu. Sama softirq voidaan suorittaa usealla prosessorilla samanaikaisesti, minkä vuoksi softirq-mekanismia käytettäessä tarvitaan huolellista rinnakkaisuuden suunnittelua. Tasklet sen sijaan voidaan suorittaa vain yhdellä prosessorilla kerrallaan, jolloin sen käyttö on helpompaa kuin softirq-mekanismin käyttö. Tämän vuoksi taskletin käyttöä suositellaan ja softirq-mekanismin käyttö on perusteltua vain, kun tarvitaan äärimmäistä suorituskykyä tai skaalautuvuuta. [22, s. 132.]

Tasklet ja softirq kuitenkin suoritetaan samassa kontekstissa kuin keskeytyskäsittelijät, joten ne sisältävät samoja ominaisuuksia kuin keskeytyskäsittelijät: ne eivät voi nukkua eikä niitä voi keskeyttää. Workqueue-mekanismi puolestaan on toteutettu siten, että se suorittaa keskeytyskäsittelijän alaosan prosessikontekstissa (Process context). Tällöin alaosa voi nukkua ja voidaan käyttää nukkuvia funktioita, kuten semaforia ja mutex-poissulkemismekanismia. Suorituskyky on kuitenkin hieman taskletia heikompi, sillä keskeytyskontekstin tehtävät vuoronnetaan aina suuremmalla prioriteetilla kuin prosessikontekstin tehtävät. [22, s. 135.]

Taskletia voidaan käyttää, kun tarvitaan nopea ja yksinkertainen alaosa, jonka ei tarvitse nukkua. Workqueueta tarvitaan, jos alaosan täytyy pystyä nukkumaan ja voidaan sallia hieman suuremmat viiveet. Seuraavassa ohjelmassa 4.1 esitetään yksinkertaistettu esimerkki taskletin käyttämisestä.

```
struct tasklet_struct tasklet;

void tasklet_work()
{
  /* Prosessoi keskeytyksen vaatimat toimenpiteet */
}

static irqreturn_t interrupt_handler(int irq, void *dev_id)
{
  /* Aseta tasklet ajettavaksi */
  tasklet_schedule(&tasklet);
  return IRQ_HANDLED;
}

void __init driver_init()
{
  /* Alusta tasklet */
  tasklet_init(&tasklet, tasklet_work, dev);
}
```

***Ohjelma 4.1**: Esimerkki taskletin käytöstä*

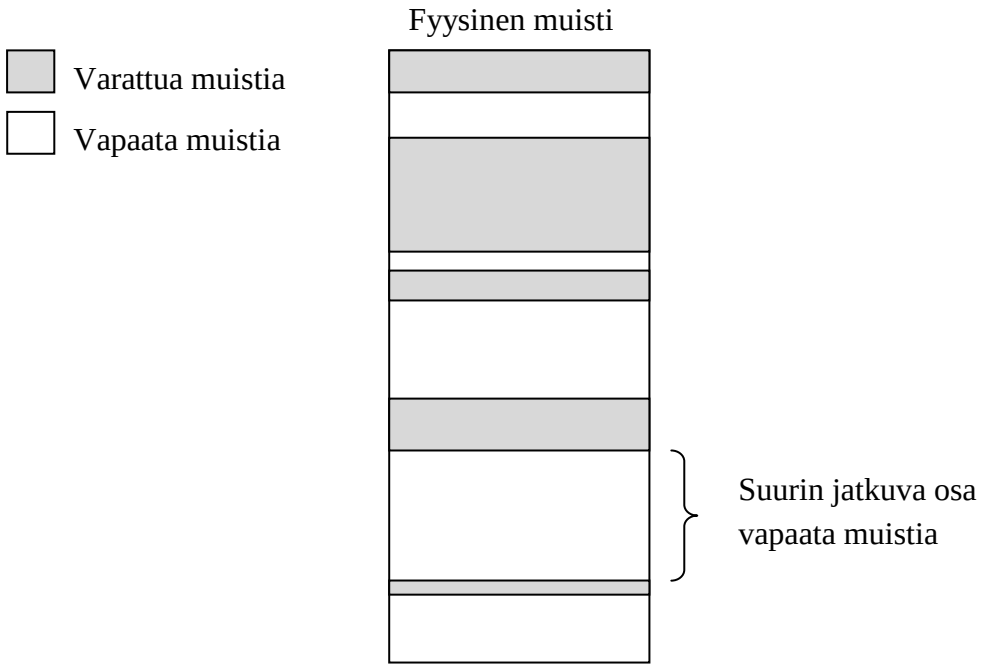
Tasklet täytyy alustaa ajurin alustusfunktiossa funktiolla *tasklet_init()*. Funktiolle annetaan parametrina osoitin *tasklet_struct*-tietueeseen sekä osoitin taskletissa suoritettavaan metodiin. Tämän jälkeen keskeytyskäsittelijä voi asettaa taskletin ajettavaksi metodilla *tasklet_schedule()*. Tällöin Linux suorittaa taskletin metodin joskus keskeytyskäsittelijän suorittamisen jälkeen. [21, s. 142-146.]

4.7 Muistin varaaminen

Linux-ytimessä voidaan varata muistia useilla eri tavoilla. Ydintilassa muistin varaaminen kuitenkin vaatii hieman tarkempia määrityksiä kuin käyttäjätilassa. Esimerkiksi ydintilassa muistia varaava prosessi ei välttämättä voi nukkua ja varaamisessa tapahtuvista virheistä ei voida palautua helposti. Linux käyttää sivuttavaa virtuaalimuistia, jolloin koko muisti on jaettu vakiokokoisiin muistisivuihin, joita muistinhallintayksikkö (MMU, Memory Management Unit) ylläpitää sivutaulun avulla. Tämän vuoksi yksi muistisivu on pohjimmiltaan pienin muistinhallintayksikkö Linuxissa. [21, s. 231-233.]

Pääsääntöiset muistinvarausmetodit ydintilassa ovat kmalloc- ja vmalloc-metodit. Kmalloc-metodi takaa, että varattu muistialue on täysin jatkuva fyysisessä muistissa, kun taas vmalloc-metodilla voidaan varata virtuaalisesti jatkuva muistialue, joka ei välttämättä ole fyysisessä muistissa jatkuva. [21, s. 238-245.] Kmalloc-metodilla pystytään nykyään varaamaan enintään neljän megatavun kokoisia jatkuvia puskureita, kun taas vmalloc-metodia käytettäessä ei puskurin koolle ole käytännössä ylärajaa, kunhan muistia on riittävästi vapaana [23]. Kmalloc on kuitenkin yleisempi muistinvaraustapa, sillä virtuaalimuistin hallinta vmalloc-metodia käytettäessä kuluttaa hieman enemmän järjestelmän resursseja [21, s. 244].

Erikokoisten puskurien varaaminen ja vapauttaminen on yksi yleisimpiä toimenpiteitä modernissa käyttöjärjestelmässä. Tämä kuitenkin johtaa helposti muistin pirstaloitumiseen, jolloin fyysisesti jatkuvien isojen puskurien varaaminen vaikeutuu. [21, s. 245-246.] Ongelma on esitetty kuvassa 4.4.



***Kuva 4.4**: Fyysisen muistin pirstaloituminen*

Pirstaloitumisongelman vuoksi yleensä käytetään vapaalistaa (Free list), johon tallennetaan tietoa varatuista ja vapaista muistialueista. Tällöin uutta puskuria varattaessa tarkistetaan ensin, sisältääkö vapaalista ennalta varatun, mutta jo vapautetun ja sopivan kokoisen muistialueen. Pirstaloitumisen ehkäisemisen lisäksi tämä myös tehostaa muistinvarausta, sillä vapaalista on myös eräänlainen välimuisti varattaville muistialueille. [21, s. 245-246.]

Linuxissa vapaalista-mekanismi on toteutettu erityisellä slab-varaajalla (Slab Allocator). Slab-vaarajassa ideana on, että jokaista usein esiintyvää muistiobjektia varten luodaan slab-välimuisti (Cache), joka on käytännössä vapaalista. Slab-varaajaa käytettäessä jokainen muistinvaraus ja vapautus sijoitetaan sopivaan slab-välimuistiin. Yhden slab-välimuistin sisältämät muistiobjektit ovat kaikki samankokoisia. [21, s. 246-249.]

Kmalloc-metodi käyttää myös taustalla slab-varaajaa [21, s. 246]. Kmalloc-metodia varten on toteutettu useita yleiskäyttöisiä slab-välimuisteja, joihin kaikki kmalloc-metodilla varatut puskurit sijoitetaan. Erään järjestelmän kaikki yleiskäyttöiset slab-välimuistit on esitetty kuvassa 4.5.

```
size-4194304            size-4096
size-2097152            size-2048
size-1048576            size-1024
size-524288             size-512
size-262144             size-256
size-131072             size-192
size-65536              size-128
size-32768              size-64
size-16384              size-32
size-8192
```

Kuva 4.5*: Erään järjestelmän kaikki yleiskäyttöiset slab-välimuistit*

Koska yleiskäyttöiset slab-välimuistit on ennalta määritetty, jokainen kmalloc-metodilla varattu puskuri täytyy sijoittaa lähimpään sopivaan slab-välimuistiin, jonka muistiobjektin koko on suurempi kuin varattavan puskurin koko. Jos varattavan puskurin koko ei ole täysin sama kuin jossakin yleiskäyttöisessä slab-välimuistissa, muistinkäyttö ei ole aivan optimaalista. Tällaista muistin haaskaamista kutsutaan sisäiseksi sirpaloitumiseksi (Internal fragmentation) [24, s. 131].

Jos käyttäjän pitääkin varata useita samankokoisia puskureita, jotka eivät ole jonkin yleiskäyttöisen slab-välimuistin muistiobjektien kokoisia, kmalloc-metodin käyttö ei ole suositeltavaa. Käyttäjän kannattaa tällöin luoda itse uusi slab-välimuisti, joka sisältää halutun kokoisia muistiobjekteja. Slab-varaaja osaa tällöin hyödyntää muistia tehokkaammin, kun näitä puskureita varataan ja vapautetaan. Slab-välimuisti voidaan luoda *kmem_cache_create()*-funktiolla ja tietystä välimuistista voidaan varata puskureita *kmem_cache_alloc()*-funktiolla [21, s. 249-257].

4.8 Oikosiirtojen käyttäminen

Oikosiirrot ovat sellaisia tiedonsiirto-operaatioita, joissa prosessori ei suorita kopiointia itse, vaan sen tekee tähän tarkoitukseen erikseen kehitetty laitteisto, eli oikosiirto-ohjain. Oikosiirtoa voidaan hyödyntää, kun tietoa täytyy kopioida paikasta toiseen. Käyttötapauksia ovat esimerkiksi tiedon kopioiminen muistissa toiseen muistipaikkaan, oheislaitteelta muistiin, tai muistista oheislaitteelle. Jos siirrettävää tietoa on paljon, oikosiirron hyödyntäminen usein vapauttaa prosessorin kuormitusta huomattavasti. [14, s. 440-441.]

Linux tarjoaa oikosiirtoja varten DMA Engine -mekanismin, joka tarjoaa laitteistoriippumattoman rajapinnan oikosiirtoja varten [14, s. 444]. DMA Engine tarjoaa erityyppisiä oikosiirtoja: slave_sg, dma_cyclic ja interleaved_dma. Slave_sg tarkoittaa kertaalleen suoritettavaa oikosiirtoa, jossa siirretään yksi Scatter/Gather-tyyppinen puskuri. Scatter/Gather-puskurilla tarkoitetaan puskuria, joka kokonaisuudessaan koostuu useista pienemmistä puskureista. Dma_cyclic tarkoittaa jaksollista oikosiirtoa. Tällöin määritellään yksi kiinteän kokoinen rengaspuskuri, johon oikosiirto-ohjain voi kirjoittaa jatkuvasti saapuvaa tietoa. [25.] Interleaved_dma tarkoittaa monipuolista konfiguroita-

vaa oikosiirtoa, jossa voidaan yhdistää erityyppisiä siirtoja esimerkiksi Scatter/Gather -puskurista yhtenäiseen puskuriin [26].

Puskurien käyttäminen oikosiirrossa vaatii myös erityistä huomiota välimuistin koherenssin takia. Kun prosessori on kirjoittanut tietoa muistiin, täytyy varmistaa, että tieto on päätynyt välimuistista päämuistiin ennen kuin oikosiirto aloitetaan muistista oheislaitteelle. Vastaavasti tieto täytyy hakea päämuistista välimuistiin ennen kuin sitä voidaan käsitellä sen jälkeen, kun oikosiirto on suoritettu oheislaitteelta muistiin. Linux tarjoaa rajapintoja, joilla välimuistin koherenssi voidaan varmistaa. Voidaan luoda valmiiksi koherentteja (Coherent DMA) oikosiirtopuskureita, jolloin käyttäjän ei tarvitse huolehtia välimuistin koherenssista. Tämä on kuitenkin joissain tapauksissa suorituskyvyn kannalta arveluttavaa, joten toinen vaihtoehto on muuttaa valmiiksi varattu muistialue dynaamisesti oikosiirtopuskuriksi (Streaming DMA), jolle täytyy erikseen suorittaa koherenssin varmistavat funktiot. [14, s. 445-446.] Valmiiksi varattu puskuri voidaan muuttaa oikosiirtopuskuriksi funktiolla *dma_map_single()*. Kun prosessori haluaa käsitellä tätä puskuria, kutsutaan funktiota *dma_sync_single_for_cpu()*. Puskurin käsittelyn jälkeen taas tulee kutsua funktiota *dma_sync_single_for_device()*. [27.]

Oikosiirtoja varten täytyy myös huomioida muita yksityiskohtia puskurien varaamisessa. Koska oikosiirron suorittaa erillinen oikosiirto-ohjain, joka on Linuxin virtuaalimuistin ulkopuolella, siirrettävän puskurin pitää olla jatkuva fyysisessä muistissa [14, s. 442]. Tämän vuoksi kmalloc on yksi mahdollinen tapa varata muistia oikosiirtoja varten. Vaikka muistia olisi kuitenkin yhteensä paljon vapaana, muisti voi olla silti pirstaloitunut, jolloin isoa, fyysisessä muistissa jatkuvaa puskuria, ei voida varata.

Tämän ongelman ratkaisemiseksi on kehitetty Scatter/Gather-puskurit, jolloin yksi iso oikosiirto voidaan suorittaa useille pienille puskureille. Nykyaikaiset oikosiirto-ohjaimet tukevat tällaista siirtotapaa, jossa oikosiirto-ohjain hakee muistista jokaisen pienen puskurin osoitteen automaattisesti. [14, s. 450-451.] Jos halutaankin yhden ison Scatter/Gather-puskurin sijaan monta pientä oikosiirtoon soveltuvaa puskuria, kannattaa käyttää dma_pool-rajapintaa. Dma_pool-rajapinta käyttää taustalla slab-varaajaa ja luo uuden slab-välimuistin, jolloin muistinkäyttö on tehokasta. Lisäksi dma_pool tekee puskureista koherentteja sekä takaa, että puskurit on sijoitettu oikein muistisivuihin. [27.]

Oikosiirtopuskurit tulisi muutenkin aina sijoittaa muistiin siten, että ne ovat tasattu välimuistin lohkon koon mukaan. Jos prosessori ja oikosiirto-ohjain pääsevät kirjoittamaan saman välimuistilohkon sisälle sopivaan muistialueeseen, tietoa saattaa hukkua välimuistin operaatioiden seurauksena. Kmalloc-metodi takaa, että varatut puskurit on tasattu välimuistin lohkon koon mukaan. [27.] Slab-välimuistia luodessa voidaan myös erikseen antaa lippu SLAB_HWCACHE_ALIGN, jolloin slab-varaaja myös takaa varattavien puskurien tasauksen välimuistin lohkon koon mukaan [21, s. 249].

5 REAALIAIKA-LINUX

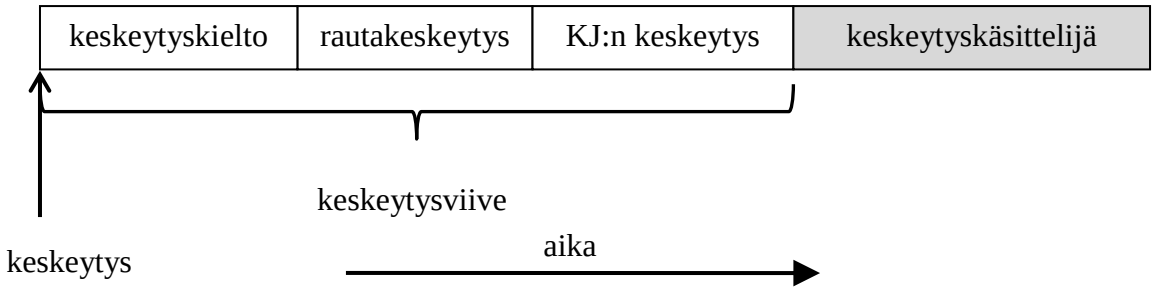
Reaaliaikajärjestelmän (RTOS, Real-Time Operating System) tavoite on täyttää sille asetetut aikavaatimukset. Yleensä tämä tarkoittaa, että järjestelmän täytyy pystyä tuottamaan oikea vaste sille tulleeseen syötteeseen tiettyjen aikarajojen sisällä. Riippuen järjestelmän tarkoituksesta, aikarajat voivat olla kovia, puolikovia tai pehmeitä. Kovan aikarajan ylittäminen tarkoittaa, että järjestelmässä syntyy virhetoiminto. Reaaliaikajärjestelmä, joka täyttää aikavaatimuksensa, on oikea-aikainen. [28.]

Linux on yleiskäyttöinen käyttöjärjestelmä (GPOS, General-Purpose Operating System), joka ei sellaisenaan sovellu reaaliaikasovelluksiin. Yleiskäyttöiset käyttöjärjestelmät eivät tavoittele pientä vasteaikaa, vaan ne keskittyvät korkeaan suoritustehoon monikäyttäjäympäristössä. Tyypillisesti reaaliaikasovelluksissa on käytetty kevyempiä reaaliaikakäyttöjärjestelmiä, jotka käyttäytyvät deterministisesti ja mahdollistavat paremman vasteen. [29, s. 360.] Linuxille on kuitenkin kehitetty erilaisia ratkaisuja reaaliaikaominaisuuksien parantamiseksi, jolloin reaaliaikasovelluksiin saadaan mukaan yleiskäyttöisen käyttöjärjestelmän monipuolisia ominaisuuksia.

Reaaliaika-Linuxin kaksi yleisintä toteutustapaa ovat: reaaliaikalaajennos ja keskeytettävä Linux-ydin. Reaaliaikalaajennokset lisäävät Linuxin rinnalle erillisen reaaliaikaytimen, jolla voidaan suorittaa reaaliaikasovelluksia Linuxista riippumatta. Linuxin RT_PREEMPT-lisäosa puolestaan pyrkii parantamaan Linux-ytimen rakennetta siten, että reaaliaikaominaisuudet paranevat ja ydin olisi lähes täysin keskeytettävä. [29, s. 361-364.]

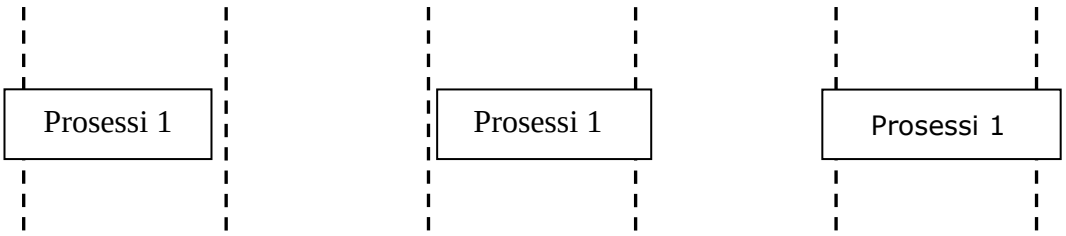
5.1 Haasteet

Keskeytysviiveet ja vuoronnuksen huojunta (Jitter) ovat reaaliaikakäyttöjärjestelmien suurimmat haasteet. Keskeytysviiveellä tarkoitetaan sitä aikaa, joka kuluu keskeytyksen tapahtumishetkestä keskeytyskäsittelijän aloitushetkeen. Keskeytysviive muodostuu kriittisestä alueesta, jossa keskeytykset on kielletty, keskeytyksen laitteistotoiminnoista sekä käyttöjärjestelmän keskeytysmekanismista. [30, s. 22-35.] Keskeytysviiveen muodostuminen on esitetty kuvassa 5.1.



***Kuva 5.1**: Keskeytysviiveen koostuminen*

Ilman käyttöjärjestelmää keskeytysviive on pienin mahdollinen. Käyttöjärjestelmästä aiheutuvat lisäviiveet johtuvat kontekstinvaihdosta, prosessitaulun ylläpidosta sekä keskeytyskäsittelijöiden käynnistämisestä. Lisäksi käyttöjärjestelmät sisältävät yleensä suurehkon määrän kriittistä koodia, jota ei voida välittömästi keskeyttää. [30, s. 22-35.]



***Kuva 5.2:** Vuoronnuksen huojunta*

Vuoronnuksen huojunta on esitetty kuvassa 5.2. Vuoronnuksen huojunnalla tarkoitetaan sitä, että prosessin saama ajoaika vaihtelee. Tämä voi tarkoittaa, että prosessin suoritus kestää ajallisesti kauemmin, koska jokin muu prosessi on keskeyttänyt sen välillä, tai prosessin suorituksen alkamisajankohta viivästyy. Vaikka prosessien prioriteetilla ja vuoronnusalgoritmilla voidaan vaikuttaa huojuntaan, ei sitä voida moniajojärjestelmässä kuitenkaan kokonaan välttää. [31.]

Keskeytysviiveiden ja vuoronnuksen huojunnan lisäksi ongelmia reaaliaikaominaisuuksiin aiheuttavat dynaaminen muistinvaraus sekä sivuttava virtuaalimuisti. Dynaamisen muistinvarauksen tuomia ongelmia voidaan kuitenkin joissain tapauksissa kiertää siten, että muisti varataan ennakkoon ennen reaaliaikasovelluksen suoritusta. [30, s. 16-17.]

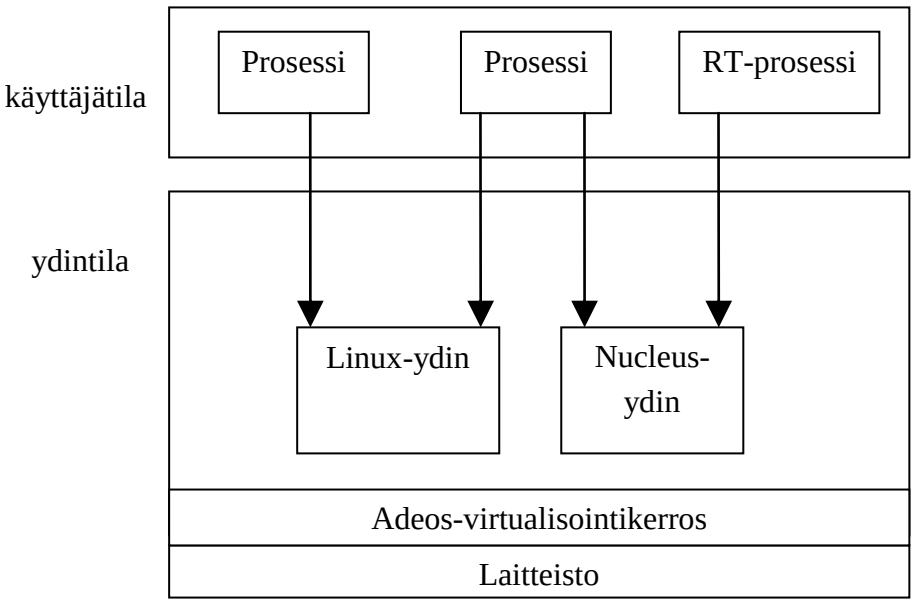
Lisäksi lähes kaikki modernit prosessorit käyttävät sekä data- että käskyvälimuistia. Koska kaikki muisti ei tietenkään voi mahtua välimuisteihin, tulee muistinkäsittelystä epädeterminististä, sillä päämuistista lukeminen on huomattavasti hitaampaa kuin välimuistista lukeminen. Tästä syystä välimuisti voidaan jopa poistaa käytöstä tietyltä muistialueelta, jotta deterministinen muistiviittaus voidaan taata. [32.] Vaihtoehtoisesti voidaan myös tiettyjä muistialueita lukita pysyvästi välimuistiin [33].

5.2 Reaaliaikalaajennokset

Reaaliaikalaajennoksen tarkoitus on lisätä Linux-ytimen rinnalle erillinen reaaliaikaydin. Reaaliaikaytimen tehtävä on hallita reaaliaikasovellusten suorittamista Linux-ytimen rinnalla. Laajennos joutuu tällöin myös vaikuttamaan Linux-ytimen toimintaan, jotta keskeytykset ohjataankin reaaliaikaytimelle ja Linuxia suoritetaan toissijaisesti reaaliaikasovellusten jälkeen. [29, s. 361.]

Yleisin reaaliaikalaajennos on tällä hetkellä Xenomai, joka on alun perin haarautunut RTAI-projektista. RTAI:n ja Xenomain rakenteet ovat siten hyvin samanlaiset. Xenomai on kuitenkin keskittynyt enemmän myös ylläpidettävyyteen sekä alustariippumattomuuteen, mikä on houkutellut projektiin enemmän kehittäjiä. [34.]

Xenomai perustuu Nucleus-reaaliaikaytimeen sekä Adeos-virtualisointikerrokseen. Adeos-virtualisointi on koko järjestelmän alin kerros ja se mahdollistaa samojen laitteistoresurssien yhteiskäytön kummallekin ytimelle. Xenomai noudattaa Linuxin prosessi/ydin -mallia, jolloin prosessit ja ydin ovat erotettu toisistaan. [29, s. 368-374.] Xenomai sisältää kuitenkin mahdollisuuden toteuttaa prosesseja myös ydintilaan, mutta tätä ollaan poistamassa tulevaisuudessa. Ydintilassa sjaitsevat prosessit voivat saavuttaa parhaimmat reaaliaikaominaisuudet, mutta tällainen järjestelmäarkkitehtuuri ei sovi yhteen Linuxin kanssa. [35.] Kuvassa 5.3 havainnollistetaan Xenomai-järjestelmän rakenne.



Kuva 5.3: *Xenomai-järjestelmän rakenne*

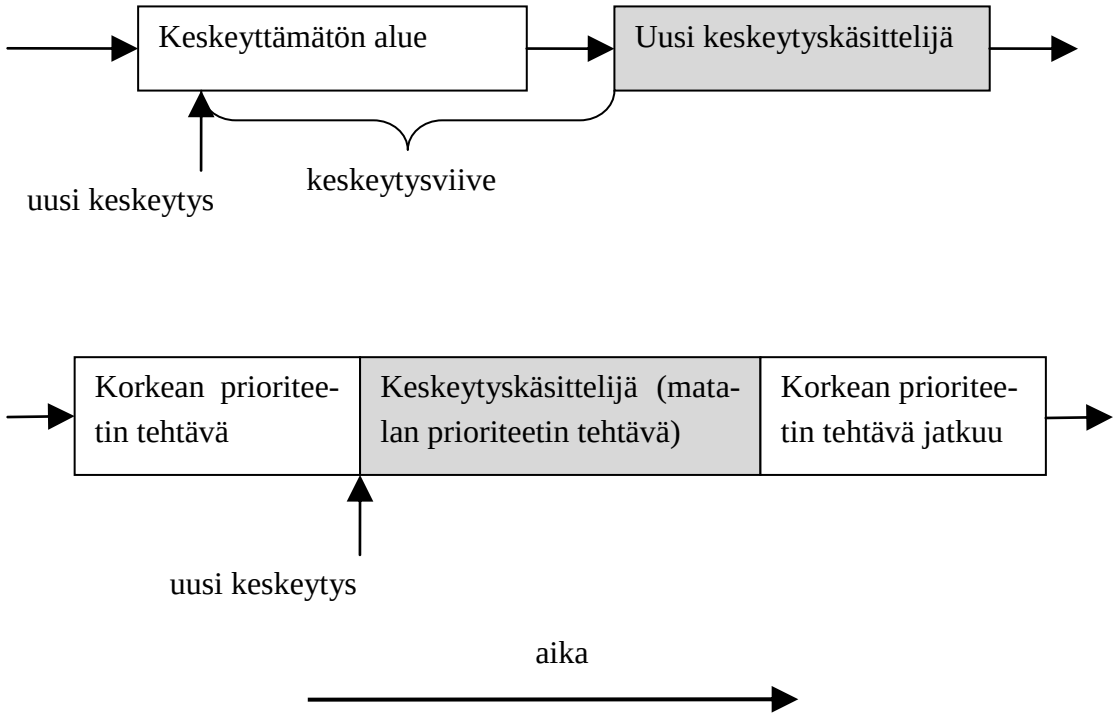
Xenomai voi sisältää prosesseja, jotka käyttävät ainoastaan Linux-ydintä, tai prosesseja, jotka käyttävät Nucleus-ydintä. Xenomai sisältää myös mekanismin, jolla Nucleus-ydintä käyttävä prosessi voi ilman erityisiä toimenpiteitä käyttää myös Linux-ydintä.

[36.] Tämä mahdollistaa joustavan toteutuksen, mutta tällöin kuitenkin prosessin kovat reaaliaikaominaisuudet katoavat, sillä suoritus siirtyy tavallisen Linuxin puolelle.

Edellä kuvatusta ominaisuudesta käytetään termejä: primäärinen tila ja sekundäärinen tila. Primäärisessä tilassa reaaliaikaprosessi käyttää ainostaan Xenomain tarjoamia palveluja ja kova reaaliaikaisuus säilyy. Jos prosessi kuitenkin käyttää jotain tavallisia Linuxin palveluja, suoritus siirtyy automaattisesti sekundääriseen tilaan ja kova reaaliaikaisuus katoaa. [36.]

5.3 RT_PREEMPT

RT_PREEMPT-lisäosan tavoitteena on muokata Linux-ydintä siten, että se sisältäisi mahdollisimman vähän osia, joita ei voi keskeyttää. Tavallisen Linux-ytimen keskeytettävyyttä on kehitetty jatkuvasti, mutta se sisältää silti edelleen osia, joita ei voida keskeyttää. Näitä osia ovat kriittiset alueet, jotka on suojattu spinlock-tyyppisillä lukituksilla, sekä keskeytyskäsittelijät. Keskeyttämättömät alueet ytimessä kasvattavat keskeytysviivettä. [37, s. 443-452.] Lisäksi keskeytyskäsittelijät normaalissa Linuxissa suoritetaan käytännössä kaikista korkeimmalla prioriteetilla. Tällöin kohdataan käänteiskeskeytysongelma (Interrupt inversion), jolloin keskeytyskäsittelijä voi suorittaa alemman prioriteetin tehtäviä kesken korkean prioriteetin prosessin suorituksen. Nämä ongelmat on havainnollistettu kuvassa 5.4.



Kuva 5.4: *Keskeytysviive ja käänteiskeskeytysongelma tavallisessa Linuxissa*

RT_PREEMPT on pyrkinyt ratkaisemaan keskeytyskäsittelijöihin liittyvät ongelmat siten, että keskeytyskäsittelijät suoritetaan ytimen säikeissä. Kun keskeytyskäsittelijät ovat säikeitä, niitä voidaan vuorontaa normaalisti. Tämä mahdollistaa myös keskeytyskäsittelijöiden keskeyttämisen ja pienentää keskeytysviivettä. Se myös ratkaisee käänteiskeskeytysongelman, sillä alemman prioriteetin keskeytyskäsittelijä voidaan vuorontaa vasta sitten, kun korkeamman prioriteetin prosessit on suoritettu loppuun. [29, s. 388-390.] Tämä kuitenkin tekee keskeytysten käsittelemisestä hieman raskaampaa, koska vuorontaja joutuu hallitsemaan kaikkia keskeytyksiä.

Kriittisten alueiden ongelma RT_PREEMPT-lisäosassa selvitetään siten, että kaikki tavalliset spinlock-lukitukset muutetaan mutex-tyypisiksi. Mutex-tyyppiset lukitukset eivät spinlock-lukitusten tapaan estä järjestelmän keskeytyksiä, ja lisäksi ne voivat nukkua. Koska keskeytyskäsittelijät on muutettu säikeiksi, ne voivat turvallisesti nukkua ja mutex-lukitukset toimivat. [37, s. 474.] Perinteisissä Linuxin keskeytyskäsittelijöissä nukkuvat lukitukset eivät voi toimia, sillä järjestelmä lukkiutuisi.

Lisäksi RT_PREEMPT pyrkii parantamaan vuorontajan tarkkuuttaa suurentamalla järjestelmäajastimen tarkkuutta. Tyypillisesti Linuxin järjestelmäajastin toimii noin 1-10 millisekunnin tarkkuudella, mutta RT_PREEMPT lisää Linuxiin korkeammalla tarkkuudella toimivan järjestelmäajastimen (High Resolution Timer). Kun järjestelmäajastin toimii pienemmällä aikavälillä, vuoronnus tarkentuu ja huojunta vähenee. Toisaalta tällöin prosessoriaikaa myös käytetään enemmän järjestelmäajastimen keskeytyspalveluun sekä toimintoihin, mikä hieman alentaa järjestelmän kokonaissuorituskykyä. [29, s. 407-410.]

RT_PREEMPT-lisäosa pyrkii parantamaan Linuxin reaaliaikaisuutta kaikilla osa-alueilla. Huomattavaa on se, että toimenpiteet pyrkivät pienentämään suurimpia mahdollisia viiveitä, jotka ovat kovan reaaliaikaisuuden kannalta tärkeimpiä. Nämä toimenpiteet kuitenkin saattavat hieman heikentää järjestelmän kokonaissuorituskykyä sekä kasvattaa pienimpiä mahdollisia viiveitä.

RT_PREEMPT ei yllä yhtä hyviin reaaliaikatuloksiin kuin esimerkiksi Xenomai. Linux-ydin on hyvin massiivinen ja RT_PREEMPT ei silti pysty vielä muuttamaan kaikkea koodia keskeytettäväksi. Jos järjestelmä vaatii kovaa reaaliaikaisuutta, kuten esimerkiksi säätöä, ei RT_PREEMPT-lisäosan käyttöä voida välttämättä suositella. Jos taas voidaan sallia hieman epävarmuutta sekä epädeterministisyyttä, esimerkiksi tiedonkeräyssovelluksessa, voi RT_PREEMPT tarjota juuri sopivasti reaaliaikaisuutta sekä yksiytimisen, tavallisen Linuxin järjestelmäarkkitehtuurin. [38.]

Yksi RT_PREEMPT-lisäosan tavoitteista on kuitenkin saada kaikki muutokset lopulta sisällytettyä tavalliseen Linuxiin mukaan, jolloin erillistä RT_PREEMPT-lisäosaa ei tarvittaisi. Tavoitteen saavuttaminen lähestyy, sillä monia osia RT_PREEMPT-lisäosasta on viime vuosien aikana siirretty mukaan Linuxiin. [29, s. 387.]

5.4 RT_PREEMPT laiteajureissa

RT_PREEMPT-lisäosan käyttö vaatii laiteajurien suunnittelussa huolellisuutta. Vaikka RT_PREEMPT-lisäosan tavoitteena on välttää koodimuutosten tekemistä, etenkin keskeytyskäsittelijät sekä lukitukset vaativat erityistä huomiota. RT_PREEMPT esimerkiksi mahdollistaa myös tavallisten keskeytyskäsittelijöiden sekä spinlock-lukitusten käytön. Näiden käyttöä on kuitenkin vältettävä, sillä koko järjestelmän reaaliaikaominaisuudet kärsivät, kun keskeyttämätöntä koodia lisätään. [39.] Joitain asioita Linux-ytimessä ei ole silti pystytty järkevästi toteuttamaan ilman näitä. Niitä ovat esimerkiksi ajastinkeskeytykset, joille vuoronnuksen aiheuttama lisäviive aiheuttaisi ongelmia [29, s. 390]. Joidenkin laiteajureiden rakenne on myös ennestään tehty sellaiseksi, ettei se toimi suoraan RT_PREEMPT-lisäosan kanssa.

Tavallinen Linuxin keskeytyskäsittelijä saadaan RT_PREEMPT-lisäosassa käyttöön IRQF_NODELAY-lipulla, joka uusimmissa ytimen versioissa on uudelleennimetty IRQ_NOTHREAD-nimiseksi [40]. Tavalliset spinlock-lukitukset taas saadaan käyttöön raw_spinlock-määreellä [39]. Käytettäessä tavallisia Linuxin keskeytyskäsittelijöitä täytyy olla tarkkana, ettei vahingossa jonkin rajapinnan kautta käytetä nukkuvia operaatioita. Tämä on yleinen virhe, kun toteutetaan ajureita RT_PREEMPT-lisäosalle, sillä RT_PREEMPT muuttaa yleisimmän lukitusmekanismin nukkuvaksi, jolloin monet Linux-ytimen palvelut saattavat yllättäen nukkua.

5.5 WRM247Plus reaaliaikamittauksia

Wapice on tehnyt muutamia reaaliaikamittauksia, jotta on pystytty määrittämään, millaisiin reaaliaikavasteisiin järjestelmä pystyy. Testit suoritettiin WRM247Plus-laitteella ja käytössä oli Linux-ytimen versio 3.6.9-rt20 RT_PREEMPT-lisäosan kanssa.

Ensimmäisessä testissä mitattiin ulkoisen keskeytyksen keskeytysviivettä ydintilaan. Testejä varten tehtiin yksinkertainen laiteajuri, joka käynnistää mikrokontrollerilta ajastinlohkon ja vastaanottaa ajastimen keskeytyksen. Ajastinlohkon laitteisto aiheuttaa ajastimen ylivuotaessa keskeytyksen. Tällöin keskeytys tapahtuu täsmälleen silloin, kun ajastin alkaa arvosta nolla. Kun keskeytyskäsittelijä suoritetaan laiteajurissa, voidaan ensimmäisenä lukea ajastimen arvo ja määrittää tästä keskeytysviive. Keskeytyskäsittelijänä käytettiin NOTHREAD-keskeytyskäsittelijää, jotta saadaan pienin mahdollinen viive mitattua. Testit suoritettiin kuormittamalla järjestelmää samaan aikaan usealla raskaalla käyttäjätilan säikeellä. Lisäksi tehtiin keskeytysshokkeja kytkemällä USB-muistitikkua (Universal Serial Bus) järjestelmään testien aikana. Testitulokset esitetään taulukossa 5.1.

Taulukko 5.1: *Keskeytysviive ydintilaan*

Kuormitus	Min (µs)	Avg (µs)	Max (µs)
Tyhjäkäynti	1,58	2,30	9,09
5 säiettä	1,45	2,42	9,70
10 säiettä	1,45	2,42	12,97
Keskeytysshokki			26,55

Pienin mahdollinen viive on vain hieman yli yhden mikrosekunnin. Parhaimmassa tapauksessa mikään muu prosessi ei estä keskeytyskäsittelijän suorittamista ja keskeytyskäsittelijän ohjelmakoodi on prosessorin välimuistissa, jolloin hidasta päämuistia ei tarvitse käyttää ennen keskeytyskäsittelijän suorittamista. On huomattavaa, että tyhjäkäynnilläkin järjestelmässä kuitenkin suurin keskeytysviive on lähes kymmenkertainen verrattuna pienimpään viiveeseen. Tämä osoittaa, että käyttöjärjestelmän normaalit ylläpitotoimenpiteetkin riittävät aiheuttamaan suhteellisen isoja viiveitä. Kuormitettuna suurin viive kasvaa vain vähän, mutta keskeytysshokki aiheuttaa kaikista suurimman viiveen. Tämä osoittaa, että järjestelmä joutuu muiden keskeytysten sattuessa ajamaan kuitenkin enemmän sellaista koodia, jota ei pystytä heti keskeyttämään. Lisäksi muut keskeytykset voivat keskeyttää testiajurin toiminnan hetkeksi, mikä aiheuttaa myös lisää viivettä. Teoreettista pahinta mahdollista keskeytysviivettä on kuitenkin vaikea saada mitattua tietämättä ennalta, mikä yhdistelmä Linux-koodin suoritusta aiheuttaa eniten keskeytyskäsittelijöiden suoritusta estäviä lukituksia.

Seuraavaksi testattiin keskeytysviivettä käyttäjätilan prosessiin. Testijärjestely oli muuten samanlainen, mutta lisäksi tehtiin käyttäjätilan prosessi, joka jää odottamaan signaalia laiteajurilta. Kun laiteajuri saa ajastimen ylivuotokeskeytyksen, se lähettää signaalin nukkuvalle käyttäjätilan prosessille. Kun prosessi herää, se lukee ajastimen arvon, josta saadaan viive keskeytyksestä käyttäjätilan prosessin heräämiseen asti. Käyttäjätilan prosessia käytettiin normaalilla prioriteetilla, jotta muiden prosessien vaikutus viiveeseen nähdään. Tulokset esitetään taulukossa 5.2.

Taulukko 5.2: *Keskeytysviive käyttäjätilan prosessiin*

Kuormitus	Min (µs)	Avg (µs)	Max (µs)
Tyhjäkäynti	11,15	14,18	21,09
5 säiettä	7603,9	7846,9	8088,7
10 säiettä	7604,0	7846,8	8093,5
Keskeytysshokki			17860,8

Tuloksista käy ilmi, että parhaimmillaan tyhjäkäynnillä viive on hyvin pieni, lähes samaa luokkaa kuin ytimen sisällä. Kuormitus kuitenkin osoittaa, että normaalilla prioriteetilla toimiva prosessi on hyvin herkkä muiden prosessien vuoronnukselle. Viive on

tällöin noin kahdeksan millisekunnin luokkaa. Keskeytysshokin tapauksessa viive on kaikista suurin, lähes 18 millisekuntia.

Seuraavassa testissä mitattiin järjestelmän jaksollisten toimintojen tarkkuutta cyclictest-ohjelmalla. Ohjelma suorittaa jaksollisia prosesseja sleep- ja wake-operaatioilla sekä mittaa näiden huojuntaa haluttuun jaksonaikaan verrattuna. Testiohjelman prioriteettina käytettiin korkeinta reaaliaikaprioriteettia 99. Testitulokset esitetään seuraavassa taulukossa 5.3.

Taulukko 5.3: *Jaksollisen prosessin huojunta*

Kuormitus	Min (µs)	Avg (µs)	Max (µs)
Tyhjäkäynti	17	27	50
5 säiettä	17	28	51
10 säiettä	16	28	55
Keskeytysshokki			86

Tulokset osoittavat, että vaihteluväli suurimman ja pienimmän arvon välillä on melko suuri. Lisäksi alemmalla prioriteetilla ajettavat kuormitussäikeet eivät odotetusti vaikuttaneet tuloksiin. Keskeytysshokki kuitenkin suurensi pahinta tapausta jopa yli 50 prosenttia.

6 NOPEIDEN ANALOGIAMITTAUSTEN TOTEUTUSMENETELMIÄ

Nopeat analogiamittaukset voidaan toteuttaa sulautetussa Linux-ympäristössä useilla eri tavoilla. Oikean toteutustavan valintaan vaikuttavat mittaukselle asetettavat vaatimukset sekä laitteiston ja ohjelmiston asettamat rajoitukset. Eri toteutustavat tässä diplomityössä tarkoittavat käytännössä erilaisia ohjelmistoarkkitehtuurillisia ratkaisuja.

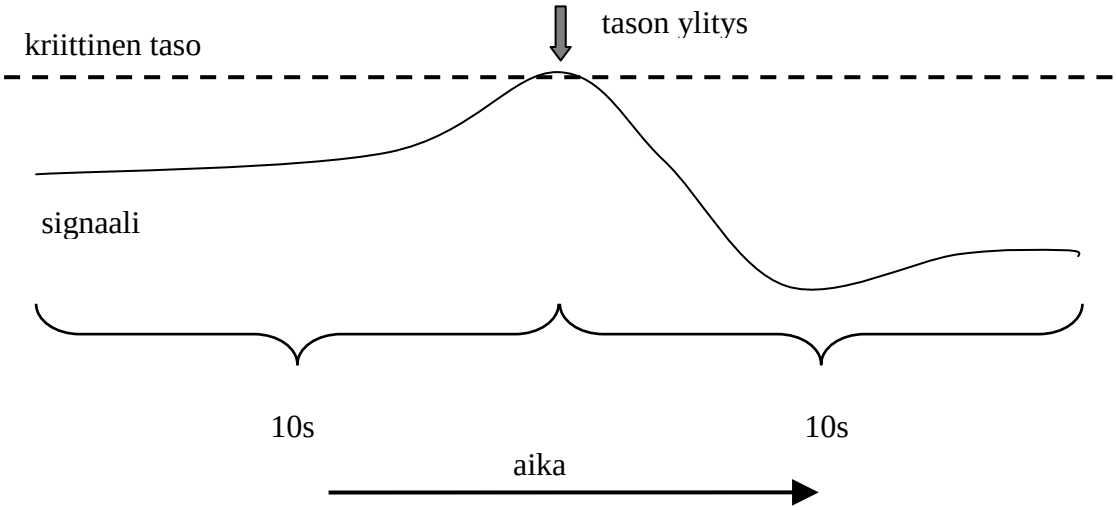
WRM on etähallintajärjestelmä, jota käytetään useissa teollisuuden mittaus- ja valvontasovelluksissa. Tästä syystä nopeiden analogiamittausten toteuttaminen tarjoaisi asiakkaille tehokkaan työkalun valvonnan tehostamiseksi. WRM on kuitenkin suunniteltu erittäin yleiskäyttöiseksi, joten sovelluskohteen ei tarvitse rajoittua vain teollisuuden mittaustarpeisiin, joissa käytetään perinteisiä teollisuusantureita. Käytännössä WRM-terminaalilaitteen analogiamittauksia voidaan hyödyntää monissa sulautettujen järjestelmien mittauskohteissa.

Valvonta- ja mittaussovelluksissa edellytetään usein mittausten jatkuvaa valvontaa, jotta poikkeamat mittauksissa voidaan havaita. Jos kaikkea mitattua tietoa ei voida tallentaa, tallennus täytyy kohdistaa poikkeaman tapahtumahetkeen. Tallennus voidaan aloittaa, kun tapahtuma on havaittu, mutta tällöin ei voida tietää, miten signaali käyttäytyi ennen tapahtumahetkeä. Siksi tallennuksesta saadaankin huomattavasti enemmän lisäarvoa, jos pystytään tallentamaan signaali myös ennen poikkeaman tapahtumahetkeä.

6.1 Vaatimukset

Nopeiden analogiamittausten vaatimukset saatiin erään asiakkaan sovellustarpeesta, jossa halutaan tallentaa mitattava signaali, jos se nousee liian korkealle tasolle. Tällöin halutaan tietää, mitä tapahtui ennen ja jälkeen, kun signaali oli ylittänyt kriittisen tason.

Vaatimusten mukaan tavoitteena on näytteistää yhtä analogiakanavaa vähintään kolmen kilohertsin taajuudella. Koska näin suurella näytteistystaajuudella kaikkea näytteistettyä tietoa ei kannata tallentaa, halutaan näytteet tallentaa vain silloin, kun näytteistettävä signaali ylittää tietyn tason. Näytteistämistä ei kuitenkaan voida aloittaa vasta siitä hetkestä, kun kriittinen taso ylitetään, sillä halutaan tarkastella myös, mitä tapahtui ennen kuin signaali ylitti kriittisen tason. Alkuperäisten vaatimusten mukaan näytteitä pitää kerätä kymmenen sekuntia ennen kriittisen tason ylitystä ja kymmenen sekuntia ylityksen jälkeen. Tämä havainnollistetaan kuvassa 6.1.

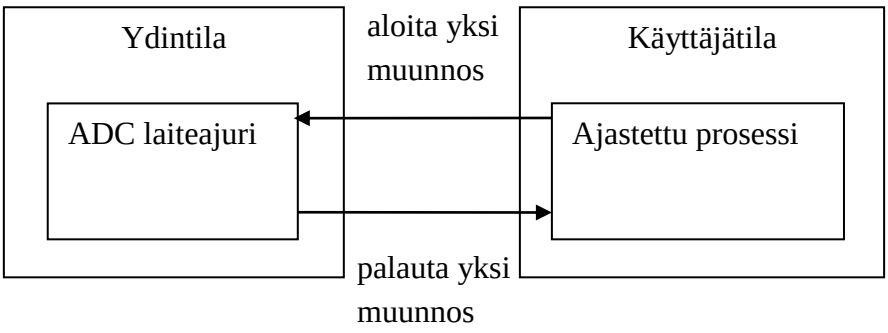


Kuva 6.1: *Kriittisen tason ylitys*

Jotta näytteitä olisi valmiiksi tallessa kymmenen sekunnin ajalta, kun signaali ylittää kriittisen tason, täytyy näytteitä käytännössä tallentaa jatkuvasti, esimerkiksi rengaspuskuriin. Lisäksi tallennettuja näytteitä täytyy myös analysoida jatkuvasti, jotta voidaan havaita kriittisen tason ylitys.

6.2 Käyttäjätilan ajastettu prosessi

Nopeiden analogiamittausten toteuttaminen vaatii jaksollista analogiamuunnoksen suorittamista. Yksinkertaisimmillaan muunnos voidaan suorittaa täysin ohjelmallisesti käyttöjärjestelmän ajastimen laukaisemana. Tällöin voidaan käyttää valmiita laiteajureita käynnistämään muunnos ja tallentamaan tulos. Esimerkiksi käyttäjätilan prosessi voidaan ajastaa kysymään uutta muunnosta laiteajurilta jaksollisesti. Käyttäjätilan prosessi voi sitten käsitellä vastaanotettuja muunnoksia tarkemmin, jotta muut vaaditut ominaisuudet täyttyvät. Menetelmä on esitetty kuvassa 6.2.



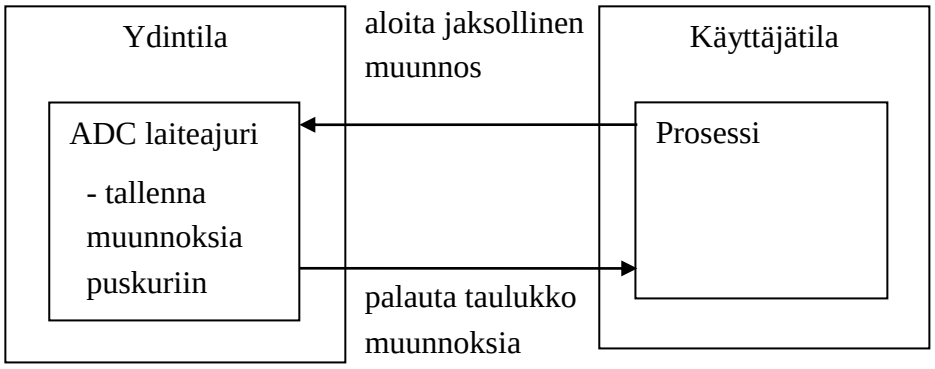
Kuva 6.2: *Käyttäjätilan ajastettu prosessi*

Tällainen toteutus olisi järkevä, jos vaaditut näytteistystaajuudet olisivat 1-2 dekadia tarvittavia matalammat. Jos näytteitä haluttaisiin kerätä esimerkiksi 1-100 kappaletta sekunnissa, voitaisiin saavuttaa tarpeeksi tarkka ja tehokas ratkaisu hyvin helposti pelkällä käyttäjätilan prosessilla. Kun näytteistystaajuutta nostetaan useihin kilohertseihin, jaksonaika on mikrosekuntien luokkaa. Tällöin vaste ja huojunta alkavat kuitenkin olla liian suuria kappaleessa 5.5 esitettyjen mittaustulosten perusteella, vaikka prosessia käytettäisiin suurimmalla reaaliaikaprioriteetilla. Kun jaksollisen prosessin huojunta on kymmeniä prosentteja mittauksen jaksonajasta, alkaa mittausten tarkkuus kärsiä jo huomattavasti. Lisäksi vaadituilla näytteistystaajuuksilla tällainen toteutustapa aiheuttaisi käyttöjärjestelmälle paljon kuormaa, kun jokaista erillistä näytettä varten pitäisi tehdä kontekstinvaihtoja ydintilaan ja suorittaa samoja ohjelmakoodin osia ajurissa.

Etuna tällä ratkaisulla olisi kuitenkin erittäin yksinkertainen ja suoraviivainen toteutus. Käyttäjätilan prosessien toteuttaminen ja käyttöjärjestelmän palvelujen hyödyntäminen on huomattavasti helpompaa, yleisempää ja tunnetumpaa kuin esimerkiksi laiteajurin toteuttaminen.

6.3 Uuden laiteajurin toteuttaminen

Toteuttamalla oma analogiamuuntimen laiteajuri voidaan laitteiston ominaisuuksia hyödyntää huomattavasti paremmin. Mitä paremmin voidaan hyödyntää mikrokontrollerin oheislaitteita, sitä vähemmän käyttöjärjestelmä ja prosessori kuormittuvat analogiamuunnosten takia. Esimerkiksi jaksollinen muunnos voidaan tällöin toteuttaa laitteistolla, jolloin käyttöjärjestelmän ominaisuudet eivät voi vaikuttaa näytteistystaajuuden tarkkuuteen eikä jokaisen muunnoksen käynnistämistä varten tarvitse suorittaa ohjelmakoodia. Käyttäjätilan prosessi voi tällöin vain käynnistää jaksollisen muunnoksen ja vastaanottaa vaikka koko mittauksen kerralla. Tämä on esitetty kuvassa 6.3.



Kuva 6.3: *Kustomoitu laiteajuri ja jaksollinen muunnos laitteistolla*

Tällöin analogiamuuntimen laiteajurin täytyy myös tallentaa jokainen muunnos puskuriin, jotta ne voidaan myöhemmin kopioida käyttäjätilan prosessille. Muunnosten

analysointi voidaan tällöin suorittaa joko käyttäjätilan prosessissa vastaanotetulle puskurille tai suoraan jo laiteajurissa.

6.4 Analogiamuuntimen käyttäminen

Tarkempia toteutusmahdollisuuksia varten täytyy myös tarkastella, miten analogia-digitaalimuunnin toimii ATSAMA5D35-mikrokontrollerissa. Analogia-digitaalimuuntimen tärkeimmät rekisterit ovat ADC_CR, ADC_MR ja ADC_TRGR, joiden avulla alustetaan ja konfiguroidaan analogiamuunnin. ADC_TRGR-rekisterirn TRGMOD-bittien avulla voidaan valita, minkä signaalin perusteella analogiamuunnokset käynnistetään. [10, s. 1643-1674.] Vaihtoehdot esitetään taulukossa 6.1.

***Taulukko 6.1**: Signaalit muunnoksen käynnistämiseksi*

Vaihtoehto	**Muunnoksen käynnistävä tekijä**
NO_TRIGGER	Ohjelmallisesti ADC_CR-rekisteristä
EXT_TRIG_RISE	Ulkoisen signaalin nousevalla reunalla
EXT_TRIG_FALL	Ulkoisen signaalin laskevalla reunalla
EXT_TRIG_ANY	Ulkoisen signaalin kummallakin reunalla
PEN_TRIG	Kosketusnäytöllä havaitaan kosketus (vain kosketusnäyttöjä käytettäessä)
PERIOD_TRIG	Jaksollinen signaali analogiamuuntimen sisäisestä ajastimesta (jaksonaika määritetään biteissä TRGPER)
CONTINUOUS	Edellinen muunnos on valmistunut (välittömästi jatkuva muunnos)

Kun analogiamuunnin on tilassa NO_TRIGGER, voidaan muunnos käynnistää ainoastaan ohjelmallisesti kirjoittamalla rekisteriin ADC_CR. Analogiamuunnoksen voi käynnistää EXTR_TRIG-tiloissa ulkoisista signaaleista, joita voivat olla esimerkiksi mikrokontrollerin ajastinlohkot tai jopa ulkoinen signaali IO-nastalta. Kosketusnäyttölaitteita varten analogiamuunnin sisältää myös ominaisuuden, jolloin muunnos voidaan aloittaa, kun havaitaan kosketus. Jaksollisia muunnoksia varten muunnin sisältää erityisen PERIOD_TRIG-tilan, jolloin muunnoksen käynnistävä jaksollinen signaali voidaan generoida sisäisesti analogiamuuntimessa. Jaksonaika voidaan asettaa ADC_TRGR-rekisterin TRGPER-bitteihin. CONTINUOUS-tilassa uusi analogiamuunnos käynnistetään välittömästi, kun edellinen on valmistunut.

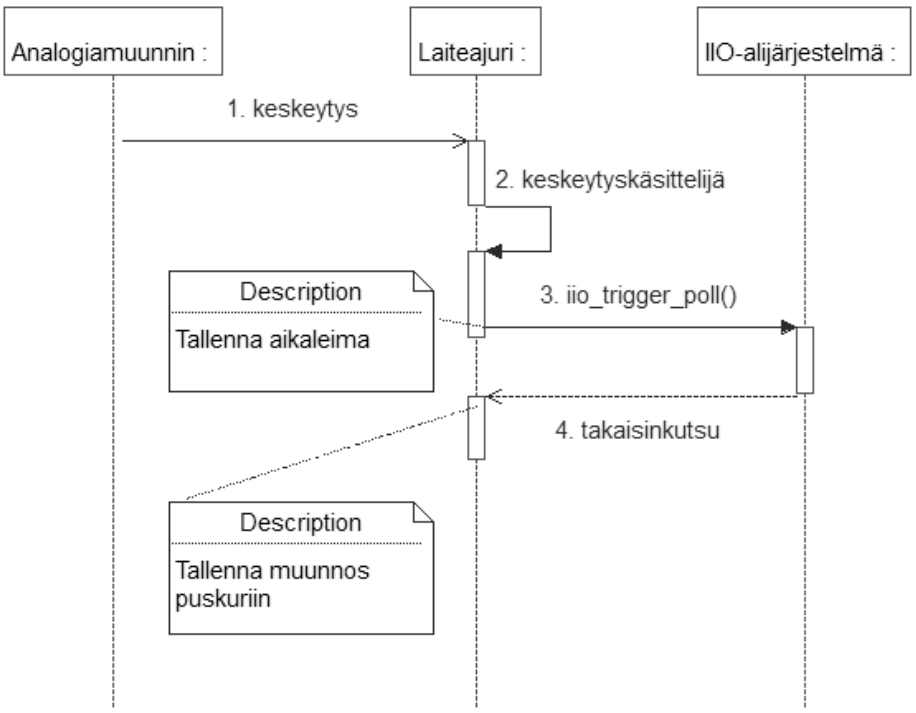
Kun analogiamuunnos on valmistunut, ADC_ISR-tilarekisteri ilmaisee, mikä rekisteri sisältää valmistuneen muunnoksen. Viimeisenä valmistunut muunnos löytyy aina rekisteristä ADC_LCDR ja kanavakohtaiset muunnokset löytyvät lisäksi rekistereistä ADC_CDR. Kun valmis muunnos kirjoitetaan johonkin näistä rekistereistä, analogiamuunnin voi generoida keskeytyksen. Keskeytykset kytketään tarvittaessa päälle ADC_IER-rekisteristä.

6.5 Näytteiden tallentaminen laiteajurissa

Jaksollinen muunnos laitteistolla on helpointa toteuttaa analogiamuuntimen tilassa PERIOD_TRIG, jolloin analogiamuuntimen sisäinen ajastin aloittaa analogiamuunnoksen automaattisesti. Vaihtoehtoisesti jaksollinen muunnos voidaan toteuttaa myös ulkoisella signaalilla, joka voidaan saada ajastinlohkoilta tai jopa mikrokontrollerin ulkoisesta IO-nastasta.

Näytteet voidaan vastaanottaa yksinkertaisesti ja nopeasti käyttämällä keskeytyksiä. Jokaisen keskeytyksen saapuessa pitää tällöin lukea valmistunut muunnos ja tallentaa se puskuriin. Jotta kaikki muunnokset saadaan tallennettua, keskeytysviive ei saa koskaan ylittää jaksollisen muunnoksen jaksonaikaa. Edellinen muunnos hukataan, jos uusi muunnos kirjoitetaan rekisteriin edellisen tilalle. Kappaleessa 5.5 esitettyjen mittaustulosten mukaan suurin keskeytysviive ydintilassa on noin 27 mikrosekuntia. Tällä perusteella teoreettinen yläraja näytteistystaajuudelle on noin 37 kilohertsiä, kun uudet muunnokset vastaanotetaan keskeytyksillä. Vaadittu kolmen kilohertsin näytteistystaajuus onnistuukin tällä menetelmällä helposti.

IIO-alijärjestelmä tarjoaa valmiin rengaspuskuritoteutuksen, joka vastaanottaa valmistuneet muunnokset keskeytyksillä ja tallentaa ne aikaleiman kanssa rengaspuskuriin. Rengaspuskuriin tallentaminen on toteutettu siten, että laiteajurin täytyy kutsua funktiota iio_trigger_poll(), jolle annetaan parametrina aikaleima. IIO-alijärjestelmä kutsuu sitten laiteajurin asettamaa takaisinkutsufunktiota, jossa varsinainen tiedon tallennus suoritetaan rengaspuskuriin. Tallennus voidaan aloittaa esimerkiksi keskeytyskäsittelijässä heti, kun muunnos on valmistunut. Aikaleima tallennetaan jo ennen takaisinkutsufunktion suorittamista, jotta se vastaisi todellisuutta mahdollisimman hyvin. On huomattava, että aikaleima tallennetaan puskuriin erikseen jokaiselle muunnokselle. IIO-rengaspuskuri on toteutettu kfifo-jonolla ja sen käyttäjärajapinta on toteutettu IIO-alijärjestelmän sysfs-rajapintaan, jossa voidaan asettaa puskurin koko sekä lukea näytteitä puskurista FIFO-periaatteen (First In, First Out) mukaan. IIO-rengaspuskuriin tallentaminen esitetään sekvenssikaaviona kuvassa 6.4.



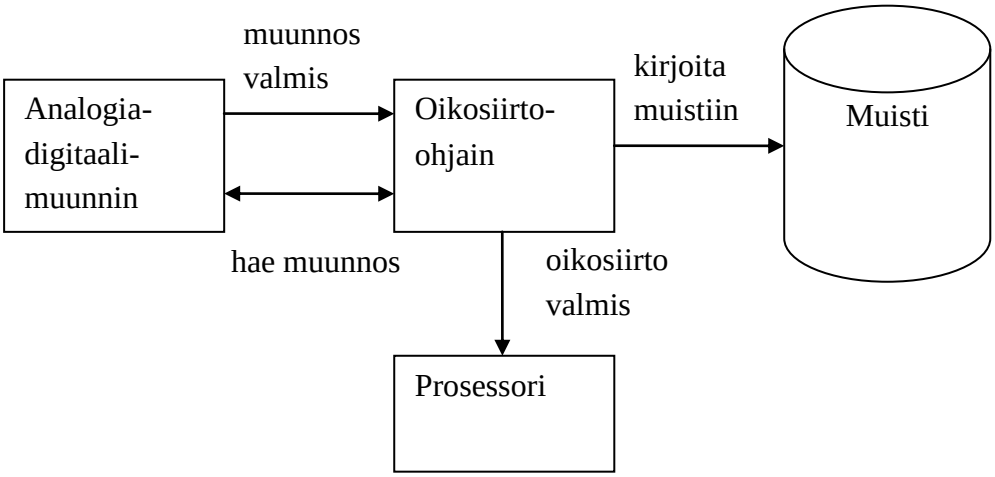
Kuva 6.4: *Tallentaminen IIO-rengaspuskuriin*

IIO-rengaspuskurin käyttö helpottaisi laiteajurin toteuttamista, mutta sovelluksen vaatimusten kannalta se sisältää myös turhia ominaisuuksia. Esimerkiksi aikaleimaa ei tarvitse tallentaa erikseen jokaiselle muunnokselle, sillä jaksollisen muunnoksen ansiosta aikaleima voidaan laskea jokaiselle näytteelle, jos tiedetään jonkin yksittäisen näytteen aikaleima. Lisäksi IIO-rengaspuskurista lukeminen on tarkoitettu käyttäjäprosessien tehtäväksi, joten laiteajurin voisi olla hankalaa suorittaa analysointi itse. IIO-rengaspuskurin vaihtoehto olisi tehdä itse toteutettu rengaspuskuri. Tällöin rengaspuskurista voidaan tehdä sovelluskohteeseen mahdollisimman sopiva.

Keskeytyksiin pohjautuvan ratkaisun heikkous on, että jokainen keskeytyskäsittelijä pitää pystyä suorittamaan mahdollisimman nopeasti ennen seuraavan muunnoksen valmistumista. Jos järjestelmä on hyvin kuormitettu, saattaa keskeytyskäsittelijän suorittaminen viivästyä tai keskeytyä hetkeksi, jolloin pahassa tapauksessa uusi muunnos ylikirjoittaa vanhan ennen kuin se on luettu analogiamuuntimen rekisteristä. Lisäksi näytteistystaajuuden kasvattaminen kasvattaa järjestelmän keskeytysmäärää. Keskeytysten suuri määrä kuormittaa käyttöjärjestelmää huomattavasti, sillä järjestelmä joutuu tekemään paljon kontekstinvaihtoja sekä prosessien ylläpitoa yrittäessään palvella jokaista keskeytystä mahdollisimman nopeasti.

Keskeytysmäärää voidaan pienentää hyödyntämällä muunnosten tallentamisessa oikosiirtoa. Oikosiirtoa käyttäessä laitteisto pystyy tallentamaan muunnokset analogiamuuntimelta suoraan muistiin ilman ohjelmakoodin suorittamista. Näytteitä ei tällöin voida hukata ohjelmiston hitauden takia. Tällöin muunnoksen valmistumisen ilmoittava signaali ohjataankin analogiamuuntimelta prosessorin keskeytysohjaimen sijaan oikosiirto-ohjaimelle. Oikosiirto-ohjain voi keskeytyksen avulla ilmoittaa prosessorille, kun

yksi tai useampi muunnos on kirjoitettu muistiin. Periaatekuva oikosiirto-ohjaimen käytöstä analogiamuuntimen kanssa esitetään kuvassa 6.5.



Kuva 6.5: *Oikosiirron käyttäminen analogia-digitaalimuuntimen kanssa*

Oikosiirtojen hyödyntäminen edellyttää kuitenkin itse toteutettua rengaspuskuria, sillä oikosiirto-ohjain tallentaa muunnokset yleensä suoraan peräkkäisiin muistiosoitteisiin. IIO-rengaspuskuria ei pystytä hyödyntämään oikosiirtojen kanssa, sillä tallennukset IIO-rengaspuskuriin tehdään rajapinnan kautta.

6.6 Näytteiden analysointi

Vaatimuksena on, että näytteistettävää signaalia tarkkaillaan jatkuvasti, jotta 20 sekuntia pitkä jakso näytteitä voidaan tallentaa, kun signaali ylittää kriittisen tason. Käytännössä tämä tarkoittaa, että jokaista näytettä täytyy yksitellen verrata kriittiseen tasoon. Tämä voidaan toteuttaa joko ohjelmistolla tai laitteistolla.

Analogia-digitaalimuunnin ATSAMA5D35-mikrokontrollerissa sisältää Window Compare -ominaisuuden, jolla voidaan asettaa näytteistettävälle signaalille ylä- sekä alaraja, joiden sisällä signaalin tulee pysyä [10, s. 1626]. Jos signaali menee näiden rajojen ulkopuolelle, analogiamuunnin lähettää prosessorille keskeytyksen. Tällöin analogiamuunnin tarkistaa jokaisen näytteen laitteistolla ja ilmoittaa rajan ylityksestä välittömästi keskeytyksellä. Laitteiston hyödyntäminen olisi järkevää, sillä ilmoitus rajan ylityksestä saataisiin välittömästi eikä se riippuisi ohjelmiston toteutuksesta. Prosessoriaikaa ei tarvitse tällöin kuluttaa ollenkaan näytteiden analysointiin. Tämä toimisi erittäin tehokkaasti oikosiirtoon perustuvan tallennuksen kanssa, jolloin yksittäisiä näytteitä ei tarvitsisi käsitellä ohjelmistolla lainkaan.

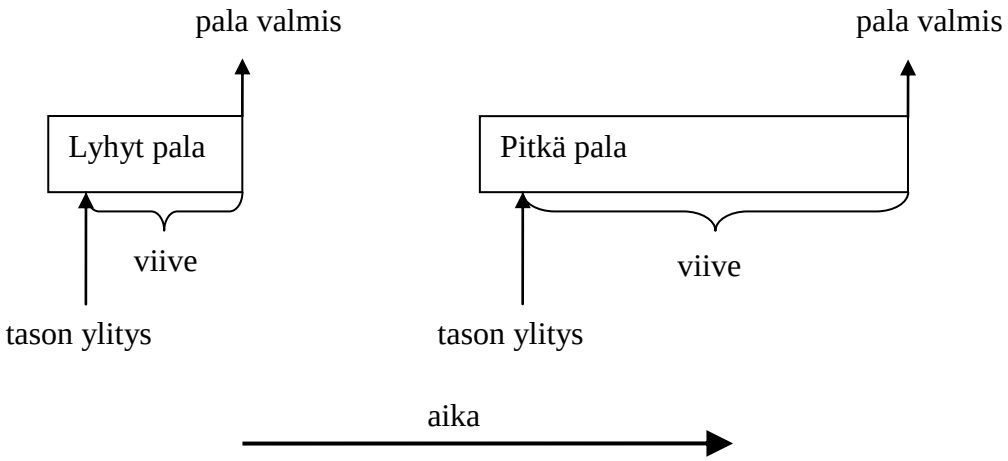
Vaihtoehtoisesti jokainen näyte voitaisiin analysoida ohjelmallisesti. Jaksollisen muunnoksen toteutustavasta riippuen näytteitä voidaan analysoida eri tavoilla. Ensimmäinen askel on päättää, analysoidaanko näytteet käyttäjätilan prosessissa vai laiteajurissa ydintilassa. Jaksolliseen käyttäjätilan prosessiin perustuva toteutus sekä IIO-rengaspuskuria käyttävä laiteajuritoteutus vaativat käytännössä, että näytteitä analysoidaan käyttäjätilan prosessissa. Etu tässä tapauksessa on laiteajurin helpompi toteutus tai jopa kokonaan valmiin ajurin käyttö. Jaksollisen käyttäjätilan prosessin kanssa tämä ratkaisu todennäköisesti toimii, jos näytteistystaajuuden vaatimus muutenkin sallii sen käytön.

Laiteajuritoteutuksienkin kanssa jokainen näyte voidaan viedä puskuriin tallennuksen jälkeen tai oikosiirron loputtua käyttäjätilan prosessille. Tällainen ratkaisu aiheuttaisi kuitenkin huomattavan paljon kuormitusta käyttöjärjestelmälle, sillä jokaisen keskeytyksen tai oikosiirron jälkeen pitäisi tehdä kontekstinvaihto ja suorittaa käyttäjätilan prosessia. Viive ydintilasta käyttäjätilaan voi olla erittäin suuri kappaleen 5.5 mittaustulosten perusteella, joten kriittisen tason ylitys havaittaisiin vasta huomattavan viiveen jälkeen.

Tehokkaampi keino on toteuttaa näytteiden analysointi laiteajuriin ydintilaan mahdollisimman lähelle näytteiden tallennusta. Keskeytyksiin perustuvassa tallennuksessa voidaan jokainen näyte analysoida suoraan keskeytyskäsittelijässä. On kuitenkin huomioitava, että keskeytyskäsittelijässä ei saa tehdä raskaita, kauan kestäviä operaatioita. Jos analysointi on raskasta, se pitää suorittaa erillisessä keskeytyskäsittelijän alaosassa. Vaadittu näytteen vertailu kriittiseen tasoon on kuitenkin erittäin kevyt operaatio, joten se voidaan tehdä suoraan keskeytyskäsittelijässä.

Oikosiirto antaa mahdollisuuden vähentää keskeytysten lukumäärää, mutta tällöin ei myöskään voida jokaista näytettä tarkastaa heti niiden valmistumisen jälkeen. Jotta näytteiden analysointi voidaan suorittaa, oikosiirto-ohjaimen pitää aiheuttaa keskeytys,

kun tietty määrä näytteitä on siirretty muistiin. Jos oikosiirto-ohjaimen keskeytys tehdään jokaisen siirretyn näytteen jälkeen, keskeytysmäärä on yhtä suuri kuin keskeytyksiin perustuvassa ratkaisussa. Viive kriittisen tason ylityksestä sen havaitsemiseen on kuitenkin suurempi, jos jokaista näytettä ei tarkasteta välittömästi sen valmistumisen jälkeen. Oikosiirtopalan koko, eli näytteiden määrä oikosiirto-ohjaimen keskeytysten välissä ja sen vaikutus viiveeseen esitetään kuvassa 6.6.



***Kuva 6.6**: Oikosiirtopalan koon vaikutus näytteiden analysoinnin viiveeseen*

Oikosiirtopalan koosta voidaan kuitenkin päätellä, kuinka suuri viive tason ylityksestä oikosiirto-ohjaimen keskeytykseen voi enintään olla. Viive voi kuitenkin vaihdella riippuen siitä, missä kohdassa oikosiirtopalaa tason ylitys tapahtuu.

6.7 Yhteenveto menetelmistä

Edellä esitetyistä menetelmistä ainakin kaksi laiteajuritoteutusta täyttää vaatimukset hyvin:

- keskeytyksiin perustuva tallennus ja analysointi ydintilassa tai Window Compare-ominaisuudella,
- oikosiirtoon perustuva tallennus ja analysointi ydintilassa oikosiirtopaloilla tai Window Compare -ominaisuudella.

Jaksollisesti suoritettava käyttäjätilan prosessi ei sovellu useiden kilohertsien näytteistystaajuudelle, joten täytyy toteuttaa laiteajuri, joka käynnistää jaksollisen muunnoksen laitteistoajastimella. Mikrokontrollerin laitteistoajastimilla saadaan ohjelmistosta riippumaton, erittäin tarkka jaksonaika näytteistykselle, joten käyttäjärjestelmän ajastimia ei kannata hyödyntää jaksollisen muunnoksen toteuttamiseksi.

Laiteajureiden toteutuksessa näytteiden tallentaminen onnistuisi IIO-rengaspuskurilla helposti, mutta se käyttäisi resursseja hieman turhaan, sillä aikaleima tallennetaan erikseen jokaiselle näytteelle. Lisäksi IIO-rengaspuskurin käyttö vaatii, että näytteiden jatkuva analyysi suoritetaan käyttäjätilassa. Jatkuva siirtyminen käyttäjätilaan jokaisen valmistuneen näytteen jälkeen kuormittaisi järjestelmää hyvin paljon. Te-

hokasta ratkaisua varten pitää käyttää itse toteutettua rengaspuskuria ja toteuttaa näytteiden analysointi ydintilassa laiteajurissa tai suoraan laitteistolla Window Compare -ominaisuutta käyttämällä.

Tallentamalla näytteet suoraan keskeytyskäsittelijässä voidaan saavuttaa tarpeeksi hyvä suorituskyky vaatimusten täyttämiseksi. Tällöin jokainen näyte voidaan myös analysoida erikseen keskeytyskäsittelijässä. Heikkoutena tällä toteutustavalla on kuitenkin, että keskeytysten määrä ja siten järjestelmän kuormitus kasvaa näytteistystaajuuden kasvaessa. Lisäksi kuormituksen alaisena keskeytysviive voi ajoittain nousta. Jos keskeytysviive on joskus suurempi kuin näytteistystaajuuden jaksonaika, näyte hukataan.

Oikosiirrolla voidaan vähentää keskeytysten määrää ja siten järjestelmän kuormaa. Oikosiirrolla ja Window Compare -ominaisuutta käyttämällä voidaankin näytteiden tallennus ja analysointi suorittaa kokonaan laitteistolla, mikä vapauttaa järjestelmän resursseja huomattavasti. Ilman Window Compare -ominaisuutta oikosiirto täytyy suorittaa oikosiirtopaloissa, jotka sisältävät useita näytteitä. Tämä aiheuttaa kuitenkin hieman enemmän viivettä näytteiden analysointiin.

7 KESKEYTYKSIIN PERUSTUVA RATKAISU

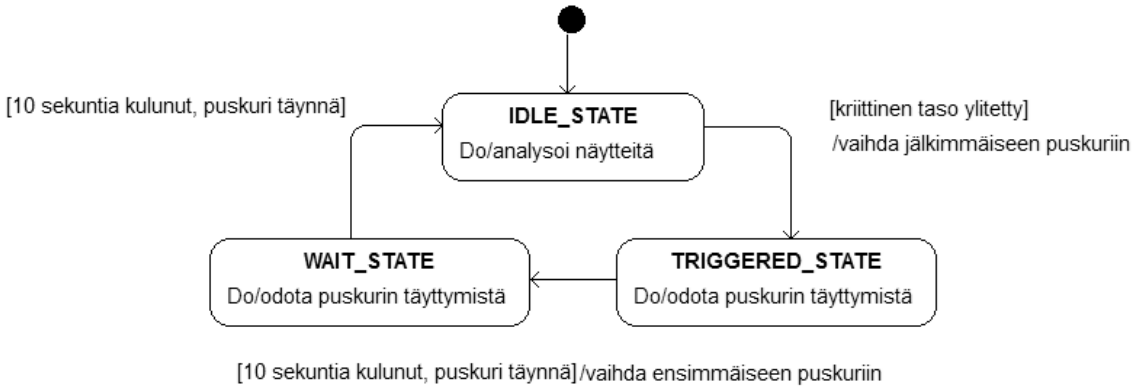
Ensimmäisenä tarvittiin demototeutus, jotta vaatimukset laatinut asiakas pääsisi mahdollisimman nopeasti kokeilemaan nopeita analogiamittauksia. Jotta toimiva ratkaisu saatiin mahdollisimman nopeasti, päätettiin käyttää keskeytyksiin perustuvaa tallennusta.

Toiminnalliset vaatimukset ajurille on kuvattu edellisessä kappaleessa 6. Helpoimman mahdollisen toteutuksen takaamiseksi ajuriin ei tehdä yleiskäyttöisyyttä tukevia ominaisuuksia. Näytteistystaajuus on asetettu kiinteästi kolmeen kilohertsiin ja koko mittaustulos on 20 sekuntia pitkä. Asiakasdemon lisäksi toteutuksen tarkoituksena on arvioida sen toimivuutta sekä suorituskykyä, jotta oikea ratkaisu valitaan, kun lopullinen yleiskäyttöinen ratkaisu toteutetaan osaksi WRM-järjestelmää.

7.1 Ajurin toimintaperiaate

Analogiamuuntimen laiteajuri asettaa laitteistoajastimen suorittamaan jaksollista muunnosta ja ajurin keskeytyskäsittelijä vastaanottaa jokaisen valmistuneen muunnoksen keskeytyskäsittelijässä. Ajuri toteutettiin myös vanhalle WRM247-laitteellle, jonka analogiamuunnin ei tue Window Compare -ominaisuutta, joten näytteiden analysointi tehdään ohjelmallisesti keskeytyskäsittelijässä jokaiselle näytteelle.

Näytteet tallennetaan itse toteutettuun rengaspuskuriin, joka muodostuu kahdesta osasta. Ensimmäinen osa sisältää näytteet kriittisen tason ylitystä edeltävältä 10 sekunnilta. Kun odotetaan kriittisen tason ylitystä, ensimmäistä puskurin osaa täytetään rengaspuskurin tavoin. Jos kriittinen taso ylitetään, vaihdetaan tallennettava puskuri toiseen, johon tallennetaan kriittisen tason ylitystä seuraavat 10 sekuntia. Rengaspuskuriin tallentamisen eri vaiheita on mallinnettu tilakoneen avulla. Tilakone esitetään kuvassa 7.1.



***Kuva 7.1**: Rengaspuskurin tilakone*

IDLE-tilassa odotetaan ja analysoidaan näytteitä. Kun jokin näyte ylittää kriittisen tason, siirrytään TRIGGERED-tilaan ja vaihdetaan jälkimmäiseen puskurin osaan. TRIGGERED-tilassa odotetaan, että loput näytteistä saadaan kerättyä, jolloin siirrytään WAIT-tilaan ja vaihdetaan takaisin puskurin ensimmäiseen osaan. WAIT-tilan tarkoituksena on estää seuraavan liipaisuehdon täyttyminen liian aikaisin, jotta päällekkäisiä näytteitä ei tallennettaisi.

7.2 Käyttäjärajapinta

Ajurin käyttäjärajapinnaksi toteutettiin yksinkertainen merkkilaiteajuri. Ajurin käyttäjä voi käynnistää jaksollisen muunnoksen sekä asettaa analysoinnin kriittisen tason ioctl-komennolla. Tallennetut näytteet voidaan kopioida käyttäjätilan prosessille read()-komennolla. Jos yhtään näytteitä ei ole saatavilla, read()-funktio tunnistaa sen ajurin tilalipuista ja palautuu välittömästi. Käyttäjätilan prosessin täytyy käydä tasaisin väliajoin kysymässä ajurilta, onko näytteitä valmiina.

Kun on siirrytty TRIGGERED-tilaan, read() palauttaa ensimmäisen 10 sekunnin osan puskurista. Kun taas on siirrytty WAIT-tilaan, read() palauttaa jälkimmäisen 10 sekunnin osan puskurista. Kun molemmat osat on vastaanotettu, käyttäjätilan prosessi voi yhdistää puskurit, jolloin muodostuu yhtenäinen 20 sekunnin puskuri. Read()-funktion toteutuksessa on huomioitu, että näytteet kopioidaan rengaspuskurista oikeassa järjestyksessä vanhin näyte ensin. Kummassakin tilasiirtymässä käyttäjätilan prosessilla on 10 sekuntia aikaa hakea puskurit ajurilta ennen kuin ne ylikirjoitetaan.

7.3 Rajoitteet

Demototeutus täyttää asiakaan vaatimukset, mutta yleiskäyttöisyyden kannalta merkittävin rajoite tällä toteutustavalla on, että ainoastaan yhden kanavan jaksollinen muunnos onnistuu kerrallaan. Lisäksi huomattavan epäkäytännöllistä on, että minkään muun kanavan yksittäistä hetkellistä arvoa ei voida lukea, jos yhdellä kanavalla on jaksollinen muunnos käynnissä.

Käyttämällä kanavakohtaisia keskeytyksiä voitaisiin eriyttää eri kanavien valmistuneet muunnokset, mutta tilakoneen ja rengaspuskurin täytyisi olla kanavakohtaisia, jotta jaksollinen muunnos voisi olla useammalla kanavalla käytössä samanaikaisesti. Hetkellisen arvon lukeminen muilta kanavilta jaksollisen muunnoksen ollessa päällä yhdellä kanavalla voitaisiin toteuttaa kohtuullisen helposti tekemällä tätä varten uusi rajapintatoiminto esimerkiksi ioctl-funktiolla. Kanava voidaan kytkeä päälle käyttäjän pyynnöstä, jolloin käyttäjän prosessi jää odottamaan valmistuvaa muunnosta, joka tallennetaan kanavakohtaisesta keskeytyksestä.

Edellä mainitut ongelmat on ratkaistavissa valitulla keskeytyksiin perustuvalla arkkitehtuurilla, mutta suurimmaksi ongelmaksi osoittautui keskeytyskäsittelijän aikakriittisyys tässä ratkaisussa. Käytettäessä säikeellistä keskeytyskäsittelijää toteutus ei pysty-

nyt suoriutumaan edes vaaditusta kolmen kilohertsin taajuudesta luotettavasti. Jos järjestelmää kuormitettiin raskaasti, näytteitä alkoi kadota jokaisessa mittauksessa. Ongelma saatiin poistettua asettamalla keskeytyskäsittelijä säikeettömäksi, kuten kappaleen 5.5 testeissäkin.

Ongelmat keskeytyskäsittelijän aikakriittisyyden kanssa osoittivat, että keskeytyksiin perustuva tallennus on hyvin riskialtis ja hyvin riippuvainen järjestelmän kuormituksesta. Kappaleen 5.5 mittauksissa keskeytysshokki myös osoittaa, että pahinta mahdollista keskeytysviivettä Linux-järjestelmässä on hankala arvioida. Lisäksi toteutuksen skaalautuvuus on huono, sillä näytteistystaajuuden kasvattaminen kasvattaa myös keskeytysmäärää. Jos vielä haluttaisiin esimerkiksi näytteistää jokaista neljää analogiakanavaa samanaikaisesti, olisi keskeytysmäärä tällä ratkaisulla nelinkertainen. Vaikka säikeetön keskeytyskäsittelijä parantaa ratkaisun suorituskykyä, se huonontaa koko järjestelmän reaaliaikavastetta, sillä se suoritetaan kaikki muut keskeytykset kiellettynä.

8 OIKOSIIRTOON PERUSTUVA YLEISKÄYTTÖINEN RATKAISU

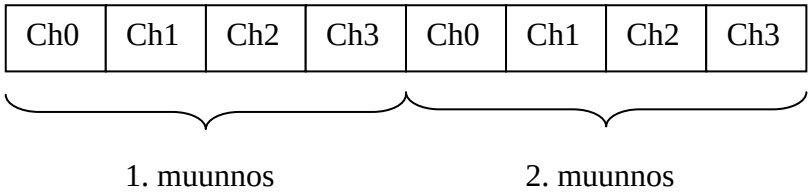
Keskeytyksiin perustuvan demoratkaisun toteuttamisen jälkeen päätettiin, että toteutuksen tulisi hyödyntää oikosiirtoa, jotta keskeytysmäärää pystyttäisiin vähentämään ja keskeytyskäsittelijän aikakriittisyydestä päästäisiin eroon. Tällöin toteutuksen pitäisi pystyä suurempaan näytteistystaajuuteen pienemmällä prosessorikuormalla. Samalla toteutuksesta tehtäisiin yleiskäyttöinen ja konfiguroitava, jotta sitä voitaisiin hyödyntää tehokkaasti WRM-järjestelmän osana.

Näytteistystaajuuden ja puskurin koon tulisi olla konfiguroitavia. Lisäksi halutaan uusia liipaisimia, jotta näytteistyspuskuri voidaan tallentaa, kun signaali alittaa jonkin määritellyn tason, tai tapahtuu jokin ulkoinen tapahtuma. Ulkoinen tapahtuma mahdollistaa monenlaiset geneeriset liipaisut. Esimerkiksi WRM-järjestelmän käyttöliittymän pyyntö voi tällöin aiheuttaa liipaisun. Kun edellisessä toteutuksessa pystyttiin näytteistämään ainostaan yhtä kanavaa, tulisi nyt neljän analogiakanavan toimia samanaikaisesti. Lisäksi jokaisen analogiakanavan hetkellinen arvo tulisi pystyä selvittämään milloin tahansa riippumatta näytteistyksen tilasta.

8.1 Oikosiirron hyödyntäminen

Koska keskeytysten suuri lukumäärä ja keskeytyskäsittelijän aikakriittisyys ovat edellisen ratkaisun suurimpia rajoitteita, pyritään suorituskykyä tehostamaan käyttämällä oikosiirtoa. ATSAMA5D35-mikrokontrolleri sisältää kaksi DMAC-ohjainta, jotka kumpikin tukevat kahdeksaa oikosiirtokanavaa. Suurin osa mikrokontrollerin oheislaitteista on kytketty suoraan oikosiirto-ohjaimille, jolloin tiedon siirto näiden välillä onnistuu täysin laitteiston ohjaamana. Analogia-digitaalimuunnin on kytketty DMAC1-ohjaimeen [10, s. 511].

DMAC käynnistää muunnostuloksen siirtämisen, kun tietty tilalippu analogiamuuntimen ADC_SR-tilarekisterissä nousee ylös. DMAC siirtää tällöin kaikkien käytössä olevien analogiakanavien muunnostuloksen muistipuskuriin. Oletuksena muunnostulokset siirretään puskuriin analogiakanavien kasvavassa numerojärjestyksessä, jolloin puskurin sisältö on kuvan 8.1 kaltainen.



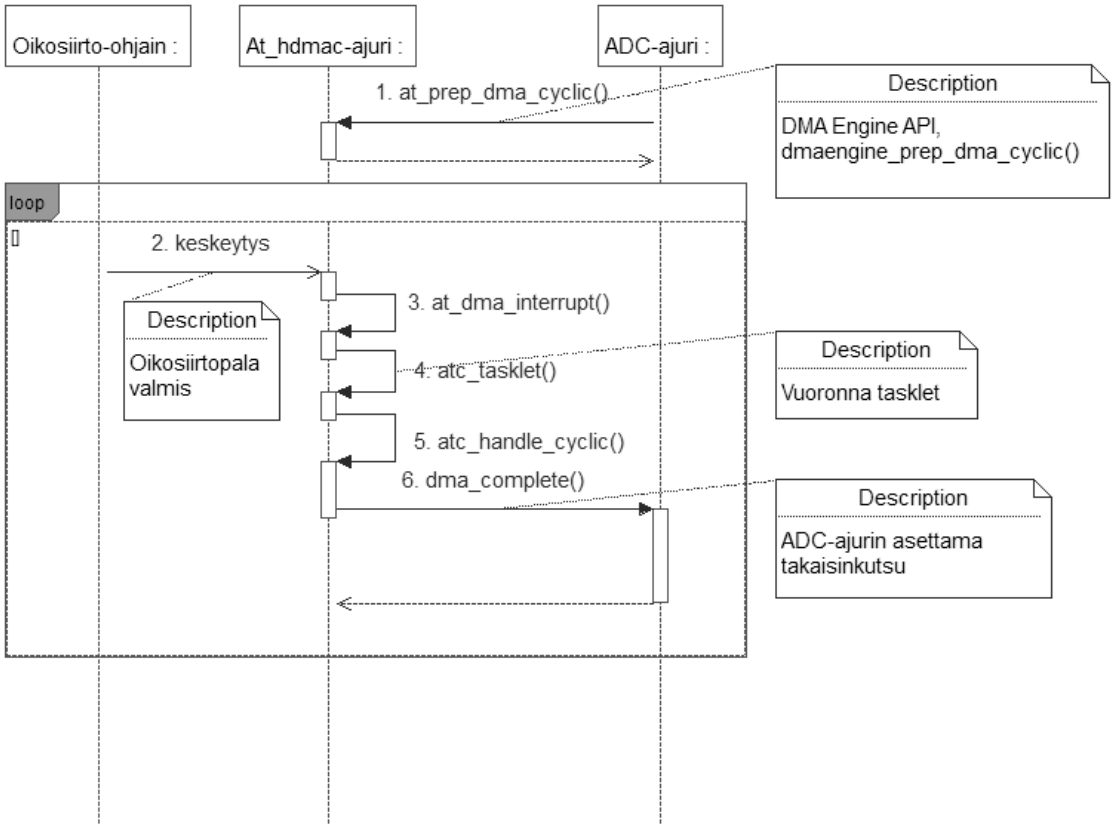
***Kuva 8.1**: Puskurin sisältö kahden analogiamuunnoksen jälkeen*

On huomattava, että keskeytystä yksittäisistä analogiamuuntimen tilasignaaleista ei saada, jos analogia-digitaalimuunnin on asetettu DMAC-ohjaimen käyttöön. DMAC siirtääkin jokaisen valmistuneen muunnoksen puskuriin, joten ohjelmallisesti ei voida saada tietoa analogiamuuntimelta, milloin jokainen yksittäinen muunnos on valmistunut. Jotta jokaisen neljän kanavan hetkellinen arvo on koko ajan saatavilla, pitää käytännössä jokaisen kanavan muunnostulos vastaanottaa oikosiirrolla. Koska näytteistystaajuus on kaikille kanaville sama, joudutaan joidenkin kanavien muunnos tällöin tallentamaan turhaan, jos kanavalta tarvitaan vain ajoittain hetkellistä arvoa eikä muita näytteitä tarvita.

8.2 Laitekohtaisen oikosiirtoajurin soveltuvuus

Ennen kuin toteutusta lähdettiin tekemään, arvioitiin *at_hdmac*-oikosiirtoajurin soveltuvuus käytetyssä Linux-ytimen versiossa 3.6.9-rt21 [41]. Linux DMA Engine -rajapinnan tarjoaman syklisen oikosiirron pitäisi soveltua tarkoitukseen, sillä tavoitteena on kerätä analogianäytteitä jatkuvasti rengaspuskuriin. Lisäksi syklinen oikosiirto voidaan jakaa tietyn kokoisiin paloihin, jolloin yksittäinen oikosiirto voi olla pienempi kuin koko puskurin koko.

Oikosiirtoajuri konfiguroi oikosiirto-ohjaimen siten, että ohjain hakee automaattisesti muistista osoitteet jokaista oikosiirtopalaa varten. Palat on ketjutettu toisiinsa ja viimeinen pala johtaa ensimmäiseen, jolloin muodostuu rengaspuskuri. Oikosiirto-ohjain pystyy tällöin tallentamaan näytteitä täysin itsenäisesti ilman ohjelmakoodin suoritusta. Analogianäytteitä ei koskaan pitäisi kadota, koska ohjelmiston ei tarvitse tehdä mitään aikakriittistä oikosiirtopalojen välillä. Oikosiirtoajurin sekvenssikaavio esitetään kuvassa 8.2.



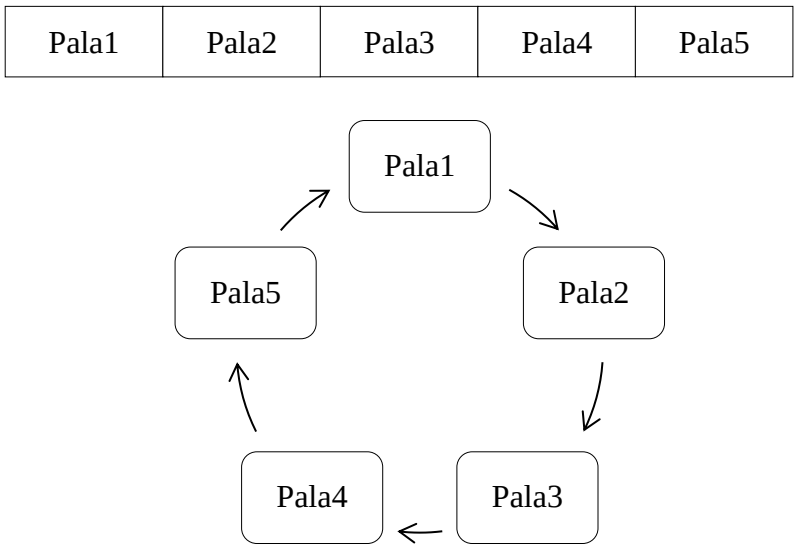
Kuva 8.2: *Oikosiirtoajurin sekvenssikaavio*

Funktio *atc_prep_dma_cyclic()* suorittaa syklisen oikosiirron konfiguroinnin. Funktio jakaa annetun puskurin paloihin sekä asettaa oikosiirron sykliseen tilaan. Oikosiirto-ohjain alkaa tällöin suorittaa palojen siirtoja syklisesti. Aina kun yksi oikosiirtopala on valmis, oikosiirto-ohjain aiheuttaa keskeytyksen, joka käsitellään *at_dma_interrupt()*-keskeytyskäsittelijässä. Keskeytyskäsittelijä vuorontaa *atc_tasklet()*-taskletin, joka kutsuu funktiota *atc_handle_cyclic().* Tämä funktio ainoastaan kutsuu käyttäjän asettamaa takaisinkutsufunktiota. On huomattava, että oikosiirto ja siten näytteiden tallentaminen jatkuu laitteistolla automaattisesti riippumatta siitä, mitä toimenpiteitä takaisinkutsufunktiossa suoritetaan.

Oikosiirtoajuri (*at_hdmac*) olettaa, että varataan yksi iso fyysisessä muistissa jatkuva puskuri, joka jaetaan paloihin. Tämä ei ole ideaalista, jos halutaan tehdä suuri puskuri, sillä fyysinen muisti sirpaloituu, kun järjestelmää käytetään. Tarpeeksi isoa puskuria varatessa saattaakin käydä niin, että yhtään tarpeeksi isoa fyysisesti jatkuvaa muistilohkoa ei ole saatavilla, kuten kappaleessa 4.7 käsiteltiin. Olisi järkevämpää, jos palat voitaisiin muodostaa varaamalla useita pieniä puskureita, jolloin isoa fyysisesti jatkuvaa muistilohkoa ei tarvita. Kuitenkin tarkastelun perusteella syklinen oikosiirto on toteutettu *at_hdmac*-ajuriin tehokkaasti, sillä oikosiirto toimii käynnistyksen jälkeen kokonaan laitteistolla. Tämä mahdollistaa tehokkaan toteutuksen analogianäytteiden keräämiseksi.

8.3 Näytteistysajurin toimintaperiaate

Oikosiirtoajurin arvioinnin perusteella näytteistysajurin toiminnan tulisi perustua rengastyyppiseen oikosiirtopuskuriin, joka on jaettu mielivaltaisen kokoisiin oikosiirtopaloihin. Tällöin jokaisen palan valmistumisen jälkeen suoritetaan syklisen oikosiirron takaisinkutsufunktio. Takaisinkutsufunktiossa voidaan tällöin käydä läpi juuri valmistunut pala ja suorittaa tarvittavat toimenpiteet liipaisimien sekä näytteistysajurin tilan ylläpitämistä varten. Esimerkki oikosiirtopuskurin jakamisesta paloihin on esitetty kuvassa 8.3.



Kuva 8.3: *Syklinen oikosiirtopuskuri jaettu paloihin*

Oikosiirtopalojen määrä sekä analogiamuuntimen näytteistystaajuus vaikuttavat siihen, miten usein takaisinkutsufunktio suoritetaan. Pienempi oikosiirtopalojen määrä sekä pienempi näytteistystaajuus vähentävät takaisinkutsufunktion suorituskertoja ja siten myös prosessorikuormaa. Pienempi oikosiirtopalojen määrä kuitenkin suurentaa palojen kokoa, mikä tarkoittaa sitä, että takaisinkutsufunktio suoritetaan harvemmin. Tällöin myös liipaisuehdot tarkistetaan harvemmin ja viive liipaisun tapahtumasta sen havaitsemiseen kasvaa. Tämä on myös selvä heikkous verrattuna edelliseen, keskeytyksiin perustuvaan ratkaisuun, jossa liipaisuehdot tarkistetaan jokaisen näytteenoton jälkeen keskeytyskäsittelijässä.

Esimerkkivaatimusten mukaan 20 sekunnin puskuri kolmen kilohertsin taajuudella neljältä kanavalta tallennettuna vaatii puskurin kooksi: $20s \cdot 3000\frac{1}{s} \cdot 2B \cdot 4 = 480KB$, mikä on vielä kohtuullisen vähän, sillä WRM247Plus-laitteessa RAM-muistia on 256 megatavua. Jos näytteistystaajuutta ja puskurin pituutta kasvatetaan, tarvittava puskurin koko nousee kuitenkin nopeasti niin suureksi, että sen kokoista jatkuvaa fyysistä muistilohkoa ei välttämättä löydetä. Tämän vuoksi oikosiirtoajuriin päätettiin toteuttaa uusi metodi, jolla voidaan toteuttaa syklinen oikosiirto valmiiksi varatuille pienille puskureil-

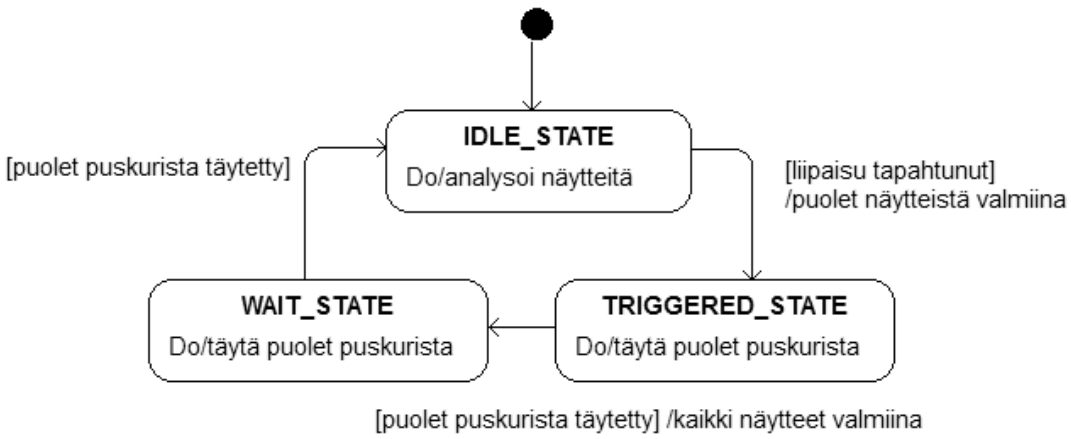
le. Näytteistysajurissa varataan tällöin oikosiirtopalojen kokoisia erillisiä puskureita, joiden osoitteet välitetään oikosiirtoajurille. Toteutettu metodi on *atc_prep_dma_cyclic_pool()*, jolle annetaan parametrina jokaisen puskurin virtuaalinen ja fyysinen osoite taulukkona, puskureiden lukumäärä sekä niiden pituus. Metodin toteutuksessa otettiin mallia *atc_pre_dma_cyclic()*-metodista. Toteutettu metodi esitetään liitteessä 1.

Jotta muistinkäyttö olisi tällöin tehokasta, käytettiin puskurien toteuttamiseen kappaleessa 4.7 käsiteltyä Slab-välimuistia. Oikosiirtoja varten oli huomioitava vielä puskurien tasaus välimuistin lohkon koon mukaan, mikä onnistuu helposti SLAB_HWCACHE_ALIGN-lipulla. Puskureiksi päätettiin toteuttaa dynaamisia oikosiirtopuskureita, sillä valmiiksi koherenteille oikosiirtopuskureille on varattu kiinteä määrä muistia.

8.4 Liipaisujen käsittely

Window Compare -ominaisuus analogiamuuntimessa vaikuttaa houkuttelevalta, mutta se ei ole kanavakohtainen. Näin ollen samoja raja-arvoja jouduttaisiin käyttämään kaikille näytteistettäville kanaville. Koska vaatimuksena on saada jokaiselle kanavalle eri raja-arvot liipaisimille, ei Window Compare -ominaisuutta voida tässä tapauksessa hyödyntää. Liipaisuehtojen tarkastelu voidaan tehdä takaisinkutsufunktiossa jokaisen oikosiirtopalan valmistumisen jälkeen. Tällöin juuri valmistunut oikosiirtopala käydään läpi ja verrataan jokaisen aktiivisen kanavan näytteitä asetettuihin liipaisuehtoihin.

Jotta näytteistämisprosessia voitaisiin hallita, toteutettiin jokaiselle kanavalle erillinen tilakone. Tilakone vastaa käytännössä kappaleessa 7.1 esitettyä tilakonetta, mutta tilasiirtymien ehdot on muutettu yleiskäyttöistä toteutusta tukeviksi. Kanavakohtainen tilakone esitetään kuvassa 8.4.

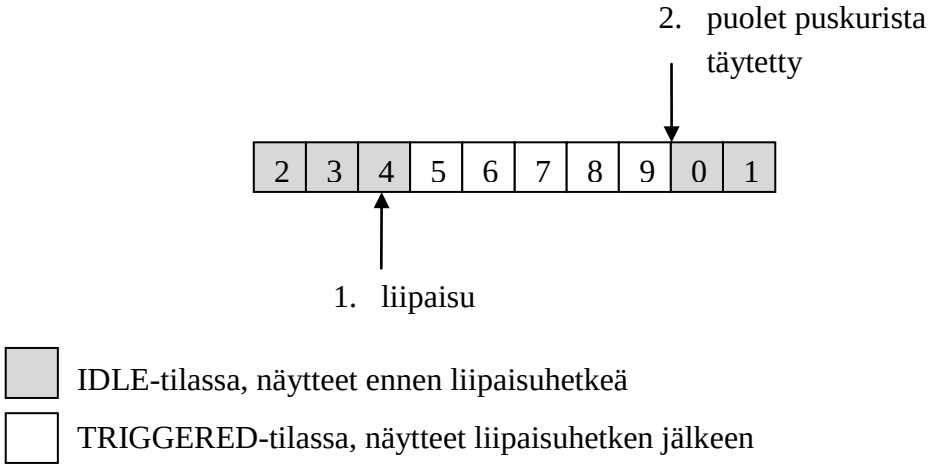


Kuva 8.4: *Kanavakohtainen tilakone*

Merkittävä ero kappaleen 7.1 toteutukseen on, että nyt tilakone ei ole koko laiteajurin laajuinen vaan täysin kanavakohtainen. Kanavat pystyvätkin olemaan toisistaan riippumatta täysin eri tiloissa. Lisäksi tallennettavaa puskuria ei tässä toteutuksessa vaihdeta tilan mukaan, sillä käytössä on vain yksi rengaspuskuri.

8.5 Tiedon siirtäminen

Palojen muodostaman rengaspuskurin tulisi kokonaisuudessaan kattaa koko näytteistettävä aika, joka sisältää näytteet ennen ja jälkeen liipaisuhetken. Tällöin liipaisuhetkellä voidaan ottaa talteen puolet puskurista, jolloin saadaan näytteet ennen liipaisuhetkeä. Tämän jälkeen TRIGGERED-tilassa odotetaan, että on jälleen kerätty puolet puskurista ja tästä saadaan näytteet liipaisuhetken jälkeen. Rengaspuskurin hyödyntäminen havainnollistetaan kuvassa 8.5. Kuvassa palojen indeksi osoittaa, missä järjestyksessä palat ovat lopullisessa mittauksessa.

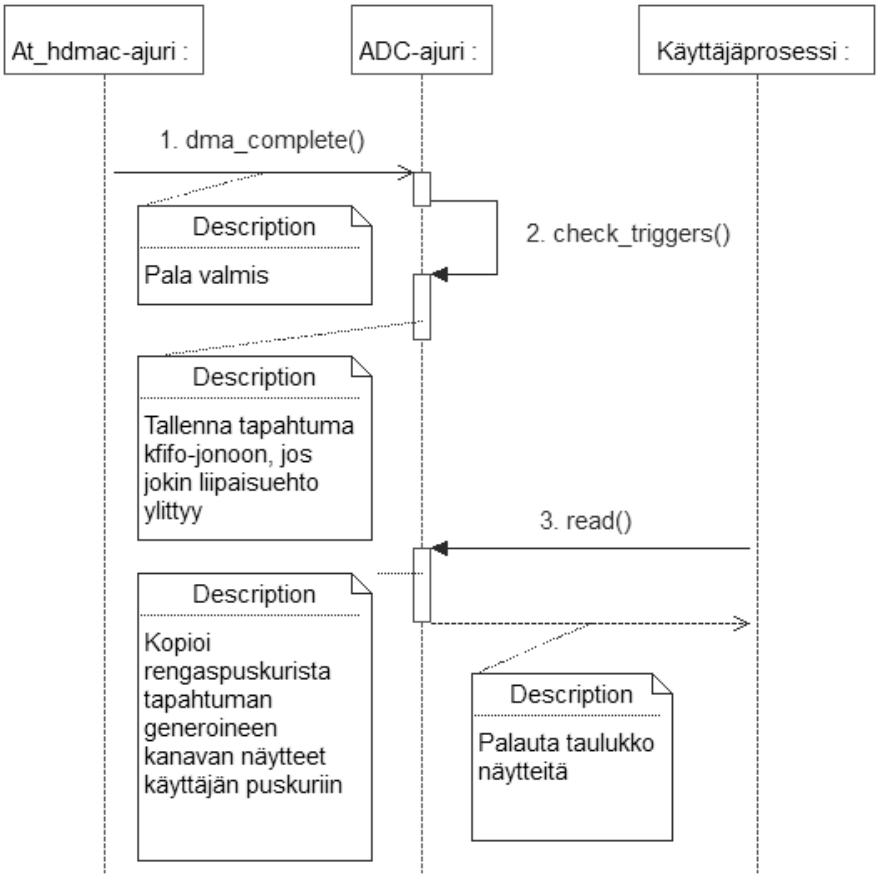


Kuva 8.5: *Rengaspuskurin käyttö ja liipaisuhetki*

Näytteet täytyy kopioida rengaspuskurista talteen toiseen puskuriin kummassakin tilasiirtymässä, jotta näytteitä ei ylikirjoiteta uudella datalla. Kummassakin tilasiirtymässä generoidaan tapahtuma, joka asetetaan kfifo-jonoon. Tapahtumaan tallennetaan aikaleima, kanava sekä palan indeksi, jossa liipaisu tapahtui. Näytteistysajurin merkkilaiteajurin *read()*-metodissa ajurin käyttäjä tarkistaa aina kfifo-jonosta, löytyykö sieltä tapahtumia. Jos jokin tapahtuma löytyy, kopioidaan annetusta indeksistä lähtien edellinen puskurin puolikas ajurin käyttäjälle.

Käyttäjälle kopioidaan aina vain yhden kanavan näytteet kerrallaan. Kfifo-jonon käyttäminen tapahtumien tallentamista varten takaa, että tapahtumat luetaan aina aikajärjestyksessä. Lisäksi kfifo-jono ratkaisee rinnakkaisuusongelman kfifo-jonon kirjoittajan ja lukijan välillä automaattisesti, kun lukijoita ja kirjoittajia on tässä tapauksessa vain yksi [42]. Näytteiden kerääminen on valmis, kun käyttäjä on vastaanottanut samalta kanavalta molemmat puskurin puolikkaat. Käyttäjälle välitetään myös tieto siitä,

minkä kanavan tapahtuma on kyseessä, ja ovatko näytteet puskurin ensimmäinen vai jälkimmäinen osa. Edellä kuvatun toteutuksen vuoksi tapahtumia voidaan generoida miltä tahansa kanavalta mielivaltaisessa järjestyksessä. Käyttäjän vastuulle jää vain yhdistää jokaisen kanavan puskurin puolikkaat toisiinsa.



***Kuva 8.6**: Näytteiden kopiointi rengaspuskurista ajurin käyttäjälle*

Kuvassa 8.6 esitetään tiedon kopiointi sekvenssikaaviona, kun siirrytään IDLE-tilasta TRIGGERED-tilaan. Tapahtuma on vastaava myös, kun siirrytään TRIGGERED-tilasta WAIT-tilaan, mutta tällöin ei enää valvota liipaisuehtoja, vaan ainoastaan kerätään toinen puolikas rengaspuskurista täyteen näytteitä. Toteutus olettaa, että käyttäjä käy säännöllisesti kysymässä read()-metodilla, onko tapahtumia tullut kfifo-jonoon. Read()-metodin paluuarvo kertoo, montako tavua vastaanotettiin. Jos kfifo-jono on tyhjä, read()-metodi palauttaa nollan. Käyttäjällä on puolet rengaspuskurin koosta aikaa käydä kopioimassa näytteet ennen kuin niitä aletaan ylikirjoittaa uusilla näytteillä. Tämä aika on varsin pitkä, sillä rengaspuskurin kokona käytetään vähintään muutamia sekunteja.

8.6 Käyttäjärajapinta

Näytteistysajurin käyttäjärajapintana hyödynnettiin IIO-alijärjestelmää. IIO-alijärjestelmän valmista rengaspuskuritoteutusta ei hyödynnetty, koska käytettiin oikosiirtoa. IIO-alijärjestelmän rajapinnasta hyödynnettiin kuitenkin rengaspuskurin hallintaominaisuuksia, kuten puskurin pituuden asettaminen sekä käyttöönotto. Puskurin lukemista ja ajurin konfigurointia varten luotiin merkkilaiteajuri, jolloin näytteet luetaan *read()*-metodilla ja asetuksia sekä liipaisuja voidaan muuttaa *ioctl()*-metodilla.

Kanavan hetkellinen arvo kuitenkin luetaan IIO-alijärjestelmän rajapinnan avulla sysfs-tiedostojärjestelmästä. Kun käyttäjän prosessi haluaa lukea jonkin kanavan hetkellisen arvon, kanavan numero tallennetaan ja jäädään odottamaan signaalia. Kun seuraavan kerran suoritetaan oikosiirron takaisinkutsufunktio, luetaan kyseisen oikosiirtopalan viimeinen näyte ja lähetetään signaali, joka herättää nukkuvan käyttäjän prosessin.

Kun jaksollinen näytteistys ja oikosiirto eivät ole käynnissä, kanavan hetkellinen arvo luetaan eri tavalla. Silloin haluttu kanava kytketään päälle, käynnistetään yksittäinen muunnos ohjelmallisesti ja jäädään odottamaan tulosta. Kun muunnos on valmis, analogiamuuntimelta vastaanotetaan keskeytys ja suoritetaan keskeytyskäsittelijä, luetaan näytteen arvo ja lähetetään signaali nukkuvalle käyttäjän prosessille. Jokaisen kanavan hetkellinen arvo pystytään lukemaan jokaisessa tilanteessa riippumatta kunkin kanavan tilasta sekä jaksollisen näytteistyksen tilasta.

9 SUORITUSKYKYMITTAUKSIA

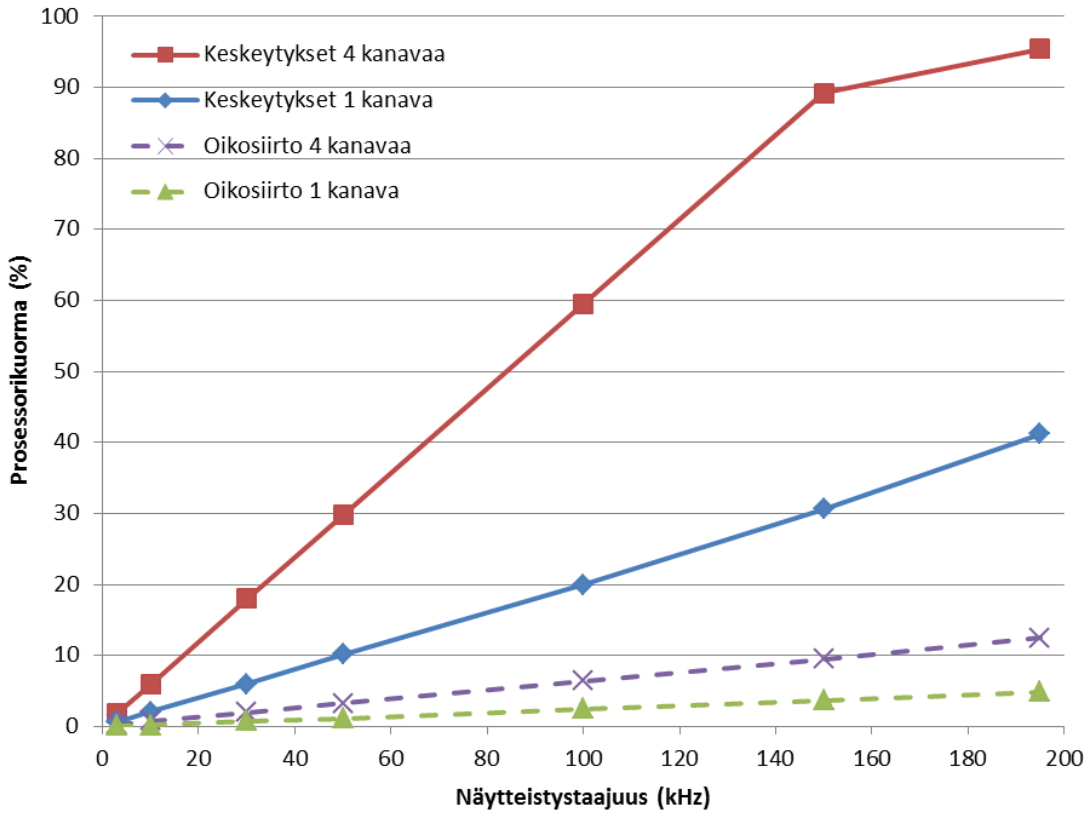
Kahden eri toteutuksen suorituskykyä vertailtiin suorittamalla yksinkertaisia testejä. Testituloksien avulla voidaan varmentaa, onko oikosiirtoon perustuva laiteajuri oikea ratkaisu yleiskäyttöiseksi ja skaalautuvaksi WRM-järjestelmän moduuliksi. Kappaleessa 7.3 todettiin, että keskeytyksiin perustuva ratkaisu pystyy täyttämään alkuperäiset asiakkaan vaatimukset. Arveltiin kuitenkin, että keskeytyksiin perustuva ratkaisu skaalautuu huonosti suuremmille näytteistystaajuuksille ja useammille analogiakanaville.

9.1 Prosessorikuorma

Skaalautuvuuden määrittämiseksi suoritettiin testejä, joilla määritettiin analogiamittauksen aiheuttama prosessorikuorma näytteistystaajuuden funktiona. Testaamista varten tehtiin mahdollisimman yksinkertainen käyttäjätilan testiohjelma, joka käynnistää nopean analogiamittauksen asetetulla taajuudella. Kun analogiamittaus on käynnissä, voidaan mitata prosessorin kuormitus. Mittaus tehtiin usealla eri näytteistystaajuudella.

Prosessorikuorman mittaus suoritettiin laskemalla Linux-ytimen tyhjäkäyntisilmukan kierrokset aikayksikössä. Vertailuarvo kuormitukselle määritettiin, kun käyttöjärjestelmä oli tyhjäkäynnillä ilman yhtäkään kuormittavaa prosessia tai toimintoa. Prosessien ja käyttöjärjestelmäytimen säikeiden prosessorikuormitus voidaan määrittää myös ohjelmalla *top*, joka tulostaa tietoja prosessien käyttämistä resursseista. Säikeetön keskeytyskäsittelijä ei kuitenkaan näy *top*-ohjelmassa, jolloin tarvitaan edellä mainittua tyhjäkäyntisilmukan laskuria. Tyhjäkäyntisilmukan laskuria varten tehdyt Linux-ytimen muutokset esitetään liitteessä 2.

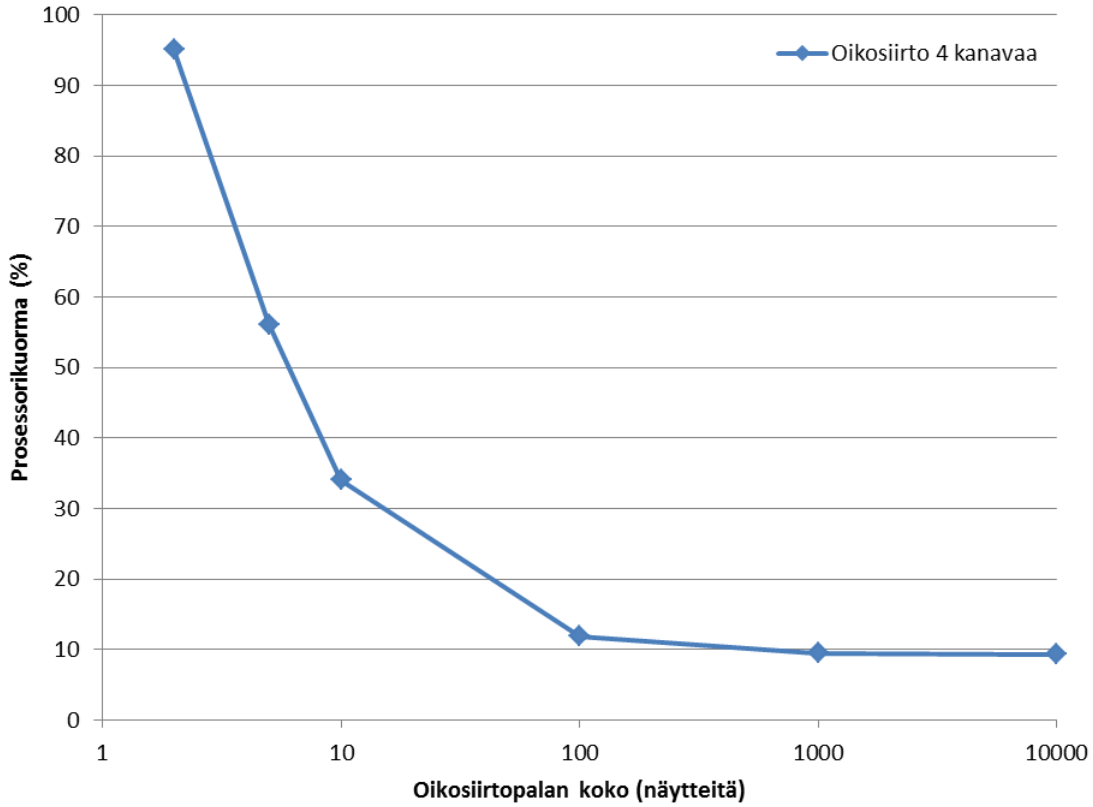
Prosessorikuorma määritettiin usealla eri näytteistystaajuudella 200 kilohertsiin asti. Mittaukset suoritettiin kummallakin ratkaisulla siten, että näytteistetään sekä yhtä että kaikkia neljää kanavaa. Keskeytyksiin perustuvassa toteutuksessa lisättiin kanavien määrää tekemättä logiikkaan kuitenkaan muutoksia. Tällöin saadaan prosessorin kuormitus kasvatettua oikealle tasolle, mutta mittausten lukeminen ajurilta antaisi kuitenkin vääriä tuloksia, koska ajurin logiikka on toteutettu tukemaan vain yhtä kanavaa. Oikosiirtoon perustuvan toteutuksen mittauksissa käytettiin 1000 näytteen kokoista oikosiirtopalaa. Mittaustulokset esitetään kuvassa 9.1.



Kuva 9.1: *Prosessorikuorma näytteistystaajuuden funktiona*

Tulokset osoittavat, että kummatkin toteutukset aiheuttavat kolmen kilohertsin näytteistystaajuudella erittäin pienen kuormituksen. Kuitenkin keskeytyksiin perustuva toteutus aiheuttaa huomattavan suuren prosessorikuorman, kun näytteistystaajuutta kasvatetaan. Lisäksi kanavien lisääminen moninkertaistaa prosessorikuorman, sillä tällöin keskeytysten määrä myös moninkertaistuu. Oikosiirtoon perustuva toteutus pystyy lähes 200 kilohertsin näytteistystaajuuteen ja prosessorikuormitus jää tällöinkin alle 15 prosenttiin. Vaikka oikosiirto-ohjain vastaanottaa aina kaikki neljä kanavaa, käytössä olevien kanavien lisääminen yhdestä neljään suurentaa kuormitusta hieman, sillä tallennettuja näytteitä joudutaan analysoimaan suurempi määrä.

On myös huomattava, että oikosiirtoon perustuvassa toteutuksessa oikosiirtopalan koko vaikuttaa prosessorin kuormitukseen. Kun oikosiirtopalan kokoa kasvatetaan, oikosiirto-ohjaimen keskeytykset vähenevät ja prosessorin kuormitus vähenee. Seuraavassa kuvassa 9.2 esitetään prosessorikuorma oikosiirtopalan koon funktiona, kun näytteistystaajuutena käytetään 150 kilohertsiä ja kaikki neljä kanavaa ovat käytössä. Kuvaajan vaaka-akseli on logaritminen.



Kuva 9.2: *Oikosiirtopalan koon vaikutus prosessorikuormaan, näytteistystaajuus 150 kHz*

Kuvaaja osoittaa, että oikosiirtopalan suurentaminen yli 1000 näytteen ei enää pienennä prosessorin kuormitusta merkittävästi. Tästä voidaan päätellä, että tällöin kuormitus ei synny oikosiirto-ohjaimen keskeytysten määrästä vaan pelkästään suuresta näytteiden määrästä, jotka täytyy yksitellen käydä läpi analysointia varten. Toisaalta, kun oikosiirtopalan kokoa pienennetään alle 1000 näytteeseen, suuri keskeytysten määrä alkaa tuoda huomattavasti ylimääräistä prosessorikuormitusta.

Oikosiirtopalan kokoa valittaessa on kuitenkin huomioitava, että näytteiden analysointi voidaan aloittaa vasta sitten, kun koko pala on vastaanotettu. Tämän vuoksi liian suuri palan koko aiheuttaa tarpeetonta viivettä analysoinnin aloittamiseen. Pienillä näytteistystaajuuksilla pala valmistuu myös hitaammin, joten näytteistystaajuutta pienennettäessä kannattaa samalla pienentää myös oikosiirtopalan kokoa.

9.2 Kadotetut näytteet

Prosessorikuorman lisäksi toteutuksen suorituskykyyn vaikuttaa merkittävästi näytteistämisen luotettavuus. Mikäli näytteitä katoaa mittauksesta, ei koko mittausta voida pitää onnistuneena. Keskeytyksiin perustuvassa ratkaisussa jokainen näyte täytyy ehtiä vastaanottamaan keskeytyskäsittelijässä ennen seuraavan näytteen valmistumista. Jos järjestelmän kuormitus suurentaa keskeytysviivettä, saatetaan näytteitä hukata. Näytteistystaajuuden suurentuessa myös suurin sallittu keskeytysviive pienenee merkittävästi.

Suurinta luotettavaa näytteistystaajuutta etsittiin kokeellisesti keskeytyksiin perustuvalla ratkaisulla. Järjestelmää kuormitettiin *doload.sh*-komentosarjalla, joka aiheuttaa monipuolista kuormitusta verkkoyhteyksille, vuorontajalle ja tiedostojärjestelmälle [43]. Muokattu komentosarja on esitetty liitteessä 3. Kaikkien prosessien prioriteetti pidettiin normaalina. Hukattu näyte voidaan havaita, kun keskeytyskäsittelijä lukee analogiamuuntimen tilarekisteristä ylivuotolipun tilan. Testeissä käytettiin säikeetöntä keskeytyskäsittelijää, sillä aiemmin kappaleessa 7.3 todettiin, että säikeellinen keskeytyskäsittelijä ei suoriudu luotettavasti edes kolmen kilohertsin näytteistystaajuudesta.

Ensimmäiset hukatut näytteet saadaan 15,5 kilohertsin näytteistystaajuudella, kun vain yksi analogiakanava on käytössä. Tämä tarkoittaa, että nopeita analogiamittauksia ei voida suorittaa tällä tai suuremmilla näytteistystaajuuksilla, kun käytetään keskeytyksiin perustuvaa toteutusta. Käytännössä olisi kuitenkin järkevää jättää myös hieman varmuusvaraa, jolloin suurin sallittu näitteistystaajuus olisi vieläkin pienempi. Lisäksi analogiakanavien lisääminen vielä alentaa suurinta sallittua näytteistystaajuutta. Oikosiirtoon perustuva ratkaisu on erityisesti tässä suhteessa parempi, koska toiminnallisen rakenteensa johdosta sillä ei ole mahdollista hukata yhtään näytettä edes suurimmilla näytteistystaajuuksilla.

10 YHTEENVETO

Työn tavoitteena oli tutkia, miten nopeat analogiamittaukset saadaan toteutettua WRM-etähallintalaitteella luotettavasti ja tehokkaasti. Valitun ratkaisun piti täyttää asiakkaan alustavat vaatimukset sekä tarjota yleiskäyttöinen ja skaalautuva toteutus osaksi WRM-järjestelmää. Mahdollisten ratkaisujen tunnistaminen ja toteuttaminen vaati tarkkaa perehtymistä Linux-laiteajurien toteuttamiseen ja Linux-ytimen tarjoamiin palveluihin sekä mekanismeihin. WRM-järjestelmässä käytössä oleva Linux-ytimen reaaliaikalisäosa RT_PREEMPT toi toteutukseen myös reaaliaikanäkökulmia.

Toteutusvaihtoehtoja tarkasteltaessa todettiin, että toimivaa ratkaisua ei pystytä toteuttamaan ilman uuden laiteajurin tekemistä. Laiteajurin toteuttamiseksi löydettiin kaksi toisistaan merkittävästi poikkeavaa toteutustapaa. Ensimmäiseksi toteutustavaksi valittiin keskeytyksiin perustuva toteutus, jossa jokainen valmistunut analogiamuunnos täytyy tallentaa keskeytyskäsittelijässä. Suoraviivaisen rakenteensa vuoksi keskeytyksiin perustuva demototeutus saatiin toteutettua ja toimitettua asiakkaalle kokeiluun hyvin nopeasti. Ratkaisu täytti asiakkaan vaatimukset, mutta se skaalautuisi huonosti suuremmille näytteistystaajuuksille ja useammille analogiakanaville, sillä tällöin keskeytysten määrä moninkertaistuisi. Lisäksi todettiin, että on suuri riski hukata näyte, jos keskeytyskäsittelijän suorittaminen viivästyy liikaa järjestelmän kuormituksen takia.

Näiden ongelmien ratkaisemiseksi valittiin lopulliseksi tuotteistettavaksi toteutukseksi oikosiirtoon perustuva ratkaisu, jossa keskeytysten määrä vähenee huomattavasti, kun oikosiirto-ohjain siirtää valmistuneet näytteet laitteistolla suoraan muistiin. Toteutuksessa piti huomioida paljon yksityiskohtia Linux-laiteajureista. Lisäksi piti arvioida Linux-ytimessä mukana olevaa oikosiirto-ohjaimen laiteajurin toteutusta, pystytäänkö sillä toteuttamaan sovellukseen sopiva tehokas oikosiirto.

Oikosiirto ratkaisee ongelman aikakriittisestä keskeytyskäsittelijästä sekä keskeytysmäärän moninkertaistumisesta, mutta aiheuttaa kuitenkin ongelman näytteiden analysoimiselle. Koska jokainen näyte pitäisi pystyä analysoimaan mahdollisimman pian analogiamuunnoksen valmistumisen jälkeen, täytyy oikosiirto jakaa paloihin. Jokainen pala täytyy analysoida vasta sitten, kun oikosiirto-ohjain ilmoittaa koko palan valmistumisesta. Oikosiirto-ohjaimen ominaisuudet mahdollistavat kokonaan laitteistolla toimivan rengaspuskurin muodostamisen pienemmistä oikosiirtopaloista, jolloin riskiä näytteiden hukkaamisesta ei synny edes palaa vaihtaessa.

Heikkous verrattuna keskeytyksiin perustuvaan ratkaisuun on kuitenkin se, että viive signaalissa tapahtuvasta muutoksesta sen havainnointiin on suurempi ja riippuu oikosiirtopalan koosta. Voidaan kuitenkin todeta, että nykyisellä toiminnallisuudella tällä viiveellä ei ole merkitystä, koska tietoa signaalissa tapahtuvasta muutoksesta käytetään

ainoastaan rengaspuskurin hallintaan, joka ei ole mitenkään aikakriittistä. Viive olisi merkityksellinen, jos signaalissa tapahtuvasta muutoksesta täytyisi jonkin aikarajan sisällä tuottaa esimerkiksi ohjaus tai hälytys.

Toteutusten suorituskykyä vertailtiin kokeellisesti. Tulokset osoittavat, että asiakkaan vaatimukset pystytään täyttämään molemmilla ratkaisuilla, mutta oikosiirtoon perustuva ratkaisu on merkittävästi parempi skaalautuvuudessa, tehokkuudessa ja luotettavuudessa. Pelkästään prosessorikuorman huomattava kasvu näytteistystaajuuden mukaan tekee keskeytyksiin perustuvasta ratkaisusta heikon, mutta vielä huomattavampaa on heikko luotettavuus. Koska näytteitä voi kadota jo 15,5 kilohertsin näytteistytaajuudella, ei sitä suurempia näytteistystaajuuksia voida edes harkita käytettäväksi. Oikosiirrolla taas voidaan käyttää suurinta kokoonpanon sallimaa näytteistystaajuutta kohtuullisen pienellä prosessorikuormalla ja ilman riskiä näytteiden katoamisesta.

Nopeita analogiamittauksia voidaan tulevaisuudessa soveltaa erilaisiin sovelluksiin. Käyttöä ei tarvitse rajoittaa pelkästään teollisuuden antureiden mittaukseen, vaan käytännössä WRM-laitteen analogiasisääntulojen ominaisuudet ovat muokattavissa, jolloin voidaan mitata monenlaisia analogiajännitteitä ja -virtoja sovelluskohteen vaatimalla tavalla. Tallennettuja mittauksia voidaan myös hyödyntää nykyistä laajemmin. Korkea näytteistystaajuus antaa mahdollisuuksia suorittaa mittaukselle esimerkiksi taajuustason analyysiä.

Työn tekeminen on edistynyt hyvin suunnitelman mukaan. Erityisesti yksinkertaisen demototeutuksen toteuttaminen keskeytyksiin perustuvalla ratkaisulla osoittautui hyväksi päätökseksi, sillä ensimmäinen toimiva toteutus saatiin näin nopeasti valmiiksi. Lisäksi demototeutuksen toimintaa pystyttiin analysoimaan ja arvioimaan ennen kuin päätettiin lopullisen toteutuksen rakenne. Aikataulu lopullisen toteutuksen valmistumiseen on hieman venynyt alkuperäisestä suunnitelmasta johtuen projektien välisistä prioriteeteista ja uuden WRM-laitteen kehittämisestä, mutta tärkeimmät aikataululliset tavoitteet silti saavutettiin.

Tutkimuksellisena osuutena työssä pohdittiin eri tapoja toteuttaa nopeat analogiamittaukset sulautetussa Linux-järjestelmässä sekä eri tapoja hyödyntää laitteistoa sekä Linux-ytimen palveluita niin, että toteutuksesta saadaan mahdollisimman luotettava ja tehokas. Vastauksina tutkimuskysymyksiin todetaan, että nopeat analogiamittaukset voidaan toteuttaa keskeytyksiin perustuvalla ratkaisulla sekä oikosiirtoon perustuvalla ratkaisulla. Oikosiirtoon perustuva ratkaisu osoittautui huomattavasti luotettavammaksi ja tehokkaammaksi, mutta toteutus vaati tarkkaa perehtymistä erityisesti Linuxin muistinhallintaan, Linuxin oikosiirtorajapintaan ja laitteiston oikosiirto-ohjaimeen sekä analogia-digitaalimuuntimeen.

Tulosten saavuttamiseksi suoritettiin iteratiivinen ja konstruktiivinen tutkimusta sekä toteuttamista sisältänyt prosessi. Tulosten perusteella saatiin toteutettua tehokas ja luotettava ratkaisu, joka täyttää alkuperäiset vaatimukset ja soveltuu erinomaisesti yleiskäyttöiseksi WRM-järjestelmän moduuliksi.

LÄHTEET

[1] Wapice Oy kotisivu [WWW]. [viitattu 20.9.2013]. Saatavissa: http://w3.wapice.com/fi/.

[2] Wapice Remote Management kotisivu [WWW]. [viitattu 14.1.2014]. Saatavissa: http://www.wrm.fi/en/.

[3] WRM247 datasheet. [viitattu 15.1.2014]. Saatavissa: http://www.wrm.fi/images/WRM247_DataSheet_2014.pdf.

[4] WRM247Plus datasheet. [viitattu 15.1.2014]. Saatavissa: http://www.wrm.fi/images/WRM247_DataSheet_2013.pdf.

[5] Ball, S.R. Analog Interfacing to Embedded Microprocessor Systems. 2nd ed. 2004, Newnes. 322 p.

[6] Chen, W.K. Analog and VLSI Circuits. 3rd ed. Boca Raton 2009, Taylor & Francis. 702 p.

[7] Laukkarinen, T. TKT-3500 Microcontroller systems Lec 7 - Analog-to-digital conversion. [viitattu 28.10.2013]. Saatavissa: http://www.tkt.cs.tut.fi/kurssit/3500/S11/Lec/pdfs/3500_Mikrokontr_07_ADC_tl11_v02.pdf.

[8] Bandwidth, Sample Rate, and Nyquist Theorem [WWW]. [viitattu 18.11.2013]. Saatavissa: http://www.ni.com/white-paper/2709/en/.

[9] LeClare, J. A Simple ADC Comparison Matrix [WWW]. (2003) [viitattu 25.10.2013]. Saatavissa: http://www.maximintegrated.com/app-notes/index.mvp/id/2094.

[10] Atmel SAMA5D3 Series Datasheet Rev 11121C. (2013) [viitattu 24.2.2014]. Saatavissa: http://www.atmel.com/Images/Atmel_11121_32-bit-Cortex-A5-Microcontroller_SAMA5D3_Datasheet.pdf.

[11] Sensata - Industrial sensing technologies [WWW]. [viitattu 3.12.2013]. Saatavissa: http://www.sensata.com/download/sensata-sensor-brochure.pdf.

[12] Murata 4-20mA Current Loop Primer. [viitattu 10.11.2013]. Saatavissa: http://www.murata-ps.com/data/meters/dms-an20.pdf.

[13] Bovet, D.P. & Cesati, M. Understanding the Linux Kernel. 3rd ed. Sebastopol 2005, O'Reilly Media. 923 p.

[14] Corbet, J., Rubini, A. & Kroah-Hartman, G. Linux Device Drivers. 3rd ed. Sebastopol 2005, O'Reilly Media. 640 p.

[15] Love, R. Linux System Programming: Talking Directly to the Kernel and C Library. 1st ed. 2007, O'Reilly Media. 392 p.

[16] POSIX services and extensions for embedded and real-time developers [WWW]. [viitattu 15.11.2013]. Saatavissa: http://www.lynuxworks.com/products/posix/posix2.php3.

[17] Mochel, P. & Murphy, M. Linux Kernel Documentation - sysfs. (2011) [viitattu 15.11.2013]. Saatavissa: https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt.

[18] Linux Industrial I/O Subsystem [WWW]. [viitattu 20.11.2013]. Saatavissa: http://wiki.analog.com/software/linux/docs/iio/iio.

[19] Ripard, M. IIO, a new kernel subsystem. (2012) [viitattu 20.11.2013]. Saatavissa: https://archive.fosdem.org/2012/schedule/event/693/127_iio-a-new-subsystem.pdf.

[20] Linux 3.12 source code - interrupt.h. [viitattu 2.3.2014]. Saatavissa: http://lxr.free-electrons.com/source/include/linux/interrupt.h?v=3.12.

[21] Love, R. Linux Kernel Development. 3rd ed. 2010, Pearson Education. 480 p.

[22] Venkateswaran, S. Essential Linux Device Drivers. Boston 2008, Pearson Education. 744 p.

[23] Billimoria, K. Kmalloc and vmalloc: Linux kernel memory allocation API limits [WWW]. [viitattu 10.1.2014]. Saatavissa: http://kaiwantech.wordpress.com/2011/08/17/kmalloc-and-vmalloc-linux-kernel-memory-allocation-api-limits/.

[24] Laplante, P.A. & Ovaska, S.J. Real-Time Systems Design and Analysis: Tools for the Practitioner. 4th ed. Hoboken 2011, Wiley. 584 p.

[25] Koul, V. Linux Kernel Documentation - DMA Engine API Guide. [viitattu 10.2.2014]. Saatavissa: https://www.kernel.org/doc/Documentation/dmaengine.txt.

[26] Brar, J. DMAEngine: Define interleaved transfer request api [WWW]. [viitattu 20.1.2014]. Saatavissa: http://lwn.net/Articles/460767/.

[27] Miller, D.S., Henderson, R. & Jelinek, J. Linux Kernel Documentation - Dynamic DMA mapping Guide. [viitattu 10.2.2014]. Saatavissa: https://www.kernel.org/doc/Documentation/DMA-API-HOWTO.txt.

[28] Kuikka, S. ACI-32020 Automaation reaaliaikajärjestelmät, luentokalvot 2012 - UML ja reaaliaikajärjestelmien kehitysprosessi. [viitattu 10.11.2013]. Saatavissa: http://www.ac.tut.fi/aci/courses/ACI-32020/RAJOKalvot2S2012.pdf.

[29] Yaghmour, K., Masters, J., Ben-Yossef, G. & Gerum, P. Building Embedded Linux Systems. Sebastopol 2008, O'Reilly Media. 464 p.

[30] What makes a good RTOS. [viitattu 10.1.2014]. Saatavissa: http://www.ece.cmu.edu/~ece749/docs/whatMakesAGoodRtos.pdf.

[31] Bril, R.J., Fohler, R. & Verhaegh, W.F.J. Execution times and execution jitter analysis of real-time tasks under fixed-priority pre-emptive scheduling. [viitattu 10.1.2014]. Saatavissa: http://www.win.tue.nl/~rbril/publications/CSR-08-27.pdf.

[32] Jacob, B. Cache Design for Embedded Real-Time Systems. [viitattu 6.2.2014]. Saatavissa: http://www.ee.umd.edu/~blj/papers/esc99.pdf.

[33] Altera - Real-Time Challenges and Opportunities in SoCs. [viitattu 5.2.2014]. Saatavissa: http://www.altera.com/literature/wp/wp-01190-real-time-socs.pdf.

[34] What are the differences between Xenomai and RTAI? [WWW]. [viitattu 28.1.2014]. Saatavissa: http://www.xenomai.org/index.php/FAQs#What_are_the_differences_between_Xenomai_and_RTAI.3F.

[35] Xenomai Roadmap. [viitattu 28.1.2014]. Saatavissa: http://www.xenomai.org/index.php/Xenomai:Roadmap.

[36] A Tour of the Native API. [viitattu 20.2.2014]. Saatavissa: http://www.xenomai.org/documentation/branches/v2.4.x/pdf/Native-API-Tour-rev-C.pdf.

[37] Embedded Linux system development. [viitattu 10.10.2013]. Saatavissa: http://free-electrons.com/doc/training/embedded-linux/slides.pdf.

[38] Brown, J.H. & Martin, B. How fast is fast enough? Choosing between Xenomai and Linux for real-time applications. [viitattu 15.12.2013]. Saatavissa: https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf.

[39] McKenney, P. A realtime preemption overview. [viitattu 3.3.2014]. Saatavissa: http://lwn.net/Articles/146861/.

[40] Linux 3.12 source code - irq.h. [viitattu 2.3.2014]. Saatavissa: http://lxr.free-electrons.com/source/include/linux/irq.h?v=3.12.

[41] Linux 3.6 source code - at_hdmac.c. [viitattu 10.10.2013]. Saatavissa: http://lxr.free-electrons.com/source/drivers/dma/at_hdmac.c?v=3.6.

[42] Seibold, S. new kfifo API v0.3. [viitattu 21.2.2014]. Saatavissa: http://lwn.net/Articles/347168/.

[43] Real Time solutions on AT91SAM SoC [WWW]. [viitattu 6.3.2014]. Saatavissa: http://www.at91.com/linux4sam/bin/view/Linux4SAM/RealTime.

LIITE 1: MUOKATTU SYKLINEN OIKOSIIRTO

Nimi:	Muokattu syklinen oikosiirto at_hdmac-ajuriin
Tyyppi:	Muutostiedosto (patch) Linux-ytimen versioon 3.6.9-rt20. Lisätyt ja poistetut rivit alkuperäiseen tiedostoon verrattuna esitetään rivin alussa esiintyvillä plus- ja miinus-merkeillä. Liitteeseen lisätyt kommentit ilmaistaan #-merkillä ja lihavoidulla tekstillä.
Kuvaus:	Liitteessä esitetään muutostiedostona muokattu syklisen oikosiirron funktio at_hdmac-ajuriin. Uusi funktio toteutettiin, jotta syklisessä oikosiirrossa pystyttäisiin hyödyntämään useita pieniä puskureita (oikosiirtopaloja) yhden ison puskurin sijaan.

```
# Muutostiedoston otsikkokenttä, ilmaisee ensimmäisen muutetun tiedoston polun
diff --git a/drivers/dma/at_hdmac.c b/drivers/dma/at_hdmac.c
index 5eb7d11..fdb1488 100644
--- a/drivers/dma/at_hdmac.c
+++ b/drivers/dma/at_hdmac.c
@@ -1040,6 +1039,111 @@
err_out:
   return NULL;
 }

#Lisätty funktio alkaa tästä
#Parametreja ovat:
# chan: käytettävä oikosiirtokanava
# buffers: taulukko kaikista puskureista
# buf_count: puskurien lukumäärä
# period_len: yksittäisen puskurin pituus tavuina
# direction: oikosiirron suunta
+
+struct dma_async_tx_descriptor *
+atc_prep_dma_cyclic_pool(struct dma_chan *chan,
+         struct dma_pool_buffer *buffers,
+         int buf_count, size_t period_len,
+         enum dma_transfer_direction direction)
+{
#Paikallisten muuttujien alustaminen
+  struct at_dma_chan   *atchan = to_at_dma_chan(chan);
+  struct at_dma_slave  *atslave = chan->private;
+  struct dma_slave_config    *sconfig = &atchan->dma_sconfig;
+  struct at_desc       *first = NULL;
+  struct at_desc       *prev = NULL;
```

```
+  unsigned long        was_cyclic;
+  unsigned int         reg_width;
+  unsigned int         i;
+  u32                  ctrla;
+  size_t               total_len = 0;
+
#Tarkistetaan, että oikosiirtokanava on syklisessä tilassa
+  was_cyclic = test_and_set_bit(ATC_IS_CYCLIC, &atchan->status);
+  if (was_cyclic) {
+     dev_dbg(chan2dev(chan), "prep_dma_cyclic: channel in use!\n");
+     return NULL;
+  }
+
#Asetetaan oikosiirron leveys
+  if (sconfig->direction == DMA_MEM_TO_DEV)
+     reg_width = convert_buswidth(sconfig->dst_addr_width);
+  else
+     reg_width = convert_buswidth(sconfig->src_addr_width);
+
#Muodostetaan puskureista (oikosiirtopaloista) linkitetty lista oiko-
siirto-ohjainta varten. Oikosiirto-ohjain käy laitteistolla läpi lin-
kitettyä listaa, kun oikosiirto etenee.
+  /* build cyclic linked list */
+  for (i = 0; i < buf_count; i++) {
+     struct at_desc    *desc;
+
#Varaa oikosiirtopalaa varten deskriptori
+     desc = atc_desc_get(atchan);
+     if (!desc)
+           goto error_desc_get;
+
#Aseta rekisteri- ja muistiosoiteasetukset oikosiirtopalalle
+     /* prepare common CRTLA value */
+     ctrla =   ATC_SCSIZE(sconfig->src_maxburst)
+           | ATC_DCSIZE(sconfig->dst_maxburst)
+           | ATC_DST_WIDTH(reg_width)
+           | ATC_SRC_WIDTH(reg_width)
+           | period_len >> reg_width;
+
+     switch (direction) {
+     case DMA_MEM_TO_DEV:
+           desc->lli.saddr = buffers[i].phys_addr;
+           desc->lli.daddr = sconfig->dst_addr;
+           desc->lli.ctrla = ctrla;
+           desc->lli.ctrlb = ATC_DST_ADDR_MODE_FIXED
+                       | ATC_SRC_ADDR_MODE_INCR
+                       | ATC_FC_MEM2PER
+                       | ATC_SIF(AT_DMA_MEM_IF)
+                       | ATC_DIF(AT_DMA_PER_IF);
+           break;
+
```

```
+     case DMA_DEV_TO_MEM:
+           desc->lli.saddr = sconfig->src_addr;
+           desc->lli.daddr = buffers[i].phys_addr;
+           desc->lli.ctrla = ctrla;
+           desc->lli.ctrlb = ATC_DST_ADDR_MODE_INCR
+                       | ATC_SRC_ADDR_MODE_FIXED
+                       | ATC_FC_PER2MEM
+                       | ATC_SIF(AT_DMA_PER_IF)
+                       | ATC_DIF(AT_DMA_MEM_IF);
+           break;
+     }
+
```

#Kasvatetaan kokonaispituutta ja linkitetään oikosiirtopala edelliseen palaan

```
+     total_len += period_len;
+     atc_desc_chain(&first, &prev, desc);
+  }
+
```

#Viimeinen pala linkitetään ensimmäiseen, jolloin saadaan rengaspuskuri

```
+  /* lets make a cyclic list */
+  prev->lli.dscr = first->txd.phys;
+
+  /* First descriptor of the chain embeds additional information */
+  first->txd.cookie = -EBUSY;
+  first->len = total_len;
+  first->tx_buswidth = reg_width;
+
+  return &first->txd;
+
```

#Virhekäsittely

```
+error_desc_get:
+  dev_err(chan2dev(chan), "not enough descriptors available\n");
+  atc_desc_put(atchan, first);
+error_out:
+  clear_bit(ATC_IS_CYCLIC, &atchan->status);
+  return NULL;
+}
```

#Tuodaan funktion symboli julkiseksi, jotta sitä voidaan käyttää ulkoisissa moduuleissa

```
+EXPORT_SYMBOL(atc_prep_dma_cyclic_pool);
+
 static int set_runtime_config(struct dma_chan *chan,
                 struct dma_slave_config *sconfig)
 {
```

```
#Muokattu otsikkotiedosto, jossa esitellään oikosiirtopalan tietotyyp-
pi
diff --git a/include/linux/dmaengine.h b/include/linux/dmaengine.h
index 9c02a45..6ac09dd 100644
--- a/include/linux/dmaengine.h
+++ b/include/linux/dmaengine.h
@@ -1023,4 +1023,10 @@
dma_cookie_t dma_memcpy_pg_to_iovec(struct dma_chan *chan,
   struct iovec *iov,
   struct dma_pinned_list *pinned_list, struct page *page,
   unsigned int offset, size_t len);

+struct dma_pool_buffer
+{
#Puskurin fyysinen osoite
+  dma_addr_t phys_addr;
#Puskurin virtuaalinen osoite
+  u8* buf;
+};
+
 #endif /* DMAENGINE_H */
--
```

LIITE 2: TYHJÄKÄYNTISILMUKAN LASKURI

Nimi:	Tyhjäkäyntisilmukan laskuri Linux-ytimeen
Tyyppi:	Muutostiedosto (patch) Linux-ytimen versioon 3.6.9-rt20. Lisätyt ja poistetut rivit alkuperäiseen tiedostoon verrattuna esitetään rivin alussa esiintyvillä plus- ja miinus-merkeillä. Liitteeseen lisätyt kommentit ilmaistaan #-merkillä ja lihavoidulla tekstillä.
Kuvaus:	Liitteessä esitetään muutostiedostona Linux-ytimen tyhjäkäyntisilmukkaan toteutettu laskuri. Laskurin kasvamisnopeudesta voidaan päätellä prosessorin kuormitus yksiprosessorijärjestelmässä. Mitä hitaammin laskuri kasvaa, sitä suurempi prosessorikuormitus järjestelmässä on. Laskuria voidaan käyttää proc-rajapinnan kautta tiedostosta ”/proc/cpu_idle”. Kun tiedosto luetaan, laskurin arvo tulostetaan ja laskuri nollataan seuraavaa mittausta varten. Laskurin alkuperäinen toteuttaja: Teemu Siuruainen, Wapice Oy

```
#Ensimmäinen muutettu tiedosto
diff --git a/arch/arm/kernel/process.c b/arch/arm/kernel/process.c
index 36fa1c4..0b4d1f2 100644
--- a/arch/arm/kernel/process.c
+++ b/arch/arm/kernel/process.c
@@ -58,18 +58,21 @@ static const char *isa_modes[] = {

 extern void setup_mm_for_reboot(void);

#hlt_counter-muuttuja pakotetaan tilaan 1, jolloin tyhjäkäyntisilmukkaa käytetään aina
-static volatile int hlt_counter;
+static volatile int hlt_counter=1;
+
+volatile unsigned long long cpu_idle_time=0;
+EXPORT_SYMBOL(cpu_idle_time);

 void disable_hlt(void)
 {
-  hlt_counter++;
+  hlt_counter=1;
 }
```

```
 EXPORT_SYMBOL(disable_hlt);

 void enable_hlt(void)
 {
-  hlt_counter--;
+    hlt_counter=1;
 }

 EXPORT_SYMBOL(enable_hlt);
@@ -82,7 +85,7 @@ static int __init nohlt_setup(char *__unused)

 static int __init hlt_setup(char *__unused)
 {
-  hlt_counter = 0;
+  hlt_counter = 1;
   return 1;
 }
```

#Varsinainen tyhjäkäyntisilmukka

```
@@ -189,8 +192,9 @@ void cpu_idle(void)
   while (1) {
      tick_nohz_idle_enter();
      rcu_idle_enter();
-     leds_event(led_idle_start);
      while (!need_resched()) {
```

#Kasvata laskuria jokaisella silmukan kierroksella

```
+            cpu_idle_time++;
 #ifdef CONFIG_HOTPLUG_CPU
            if (cpu_is_offline(smp_processor_id()))
                 cpu_die();
@@ -220,7 +224,7 @@ void cpu_idle(void)
            } else
                 local_irq_enable();
      }
-     leds_event(led_idle_end);
      rcu_idle_exit();
      tick_nohz_idle_exit();
      schedule_preempt_disabled();
```

#Toinen muutettu tiedosto, käännöstiedosto laskurin proc-rajapintaa varten

```
diff --git a/fs/proc/Makefile b/fs/proc/Makefile
index c1c7293..ea28a3f 100644
--- a/fs/proc/Makefile
+++ b/fs/proc/Makefile
@@ -21,6 +21,7 @@ proc-y      += uptime.o
 proc-y     += version.o
 proc-y     += softirqs.o
 proc-y     += namespaces.o
+proc-y     += cpu_idle.o
 proc-$(CONFIG_PROC_SYSCTL)   += proc_sysctl.o
```

```
 proc-$(CONFIG_NET)           += proc_net.o
 proc-$(CONFIG_PROC_KCORE)    += kcore.o
```

#Laskurin proc-rajapintaa varten lisätty tiedosto

```
diff --git a/fs/proc/cpu_idle.c b/fs/proc/cpu_idle.c
new file mode 100644
index 0000000..b4b582a
--- /dev/null
+++ b/fs/proc/cpu_idle.c
@@ -0,0 +1,68 @@
+#include <linux/cpumask.h>
+#include <linux/fs.h>
+#include <linux/gfp.h>
+#include <linux/init.h>
+#include <linux/interrupt.h>
+#include <linux/kernel_stat.h>
+#include <linux/proc_fs.h>
+#include <linux/time.h>
+#include <linux/sched.h>
+#include <linux/seq_file.h>
+#include <linux/slab.h>
+#include <linux/irqnr.h>
+#include <asm/cputime.h>
+
+static unsigned long g_lastRead=0;
```

#Laskuri-muuttuja

```
+extern volatile unsigned long long cpu_idle_time;
+
+static int show_idle(struct seq_file *p, void *v)
+{
```

#Laskuri pätee vain yksiprosessorijärjestelmissä (ei SMP, Symmetric Multiprocessing)

```
+#ifndef SMP
```

#Tallenna ja tulosta laskurin arvo. Nollaa laskuri seuraavaa mittausta varten.

```
+  unsigned long counter = cpu_idle_time;
+  cpu_idle_time = 0;
+  unsigned long old_time = g_lastRead;
+  g_lastRead = jiffies;
+
+    seq_printf(p,
+          "T:%lu C:%lu\n",
+     (g_lastRead - old_time),
+     counter);
+#endif
+
+  return 0;
+}
+
```

```
#Avattaessa proc-rajapinnan tiedostoa varataan tiedostolle resursseja
+static int idle_open(struct inode *inode, struct file *file)
+{
+  unsigned size = 20;
+  char *buf;
+  struct seq_file *m;
+  int res;
+
+  buf = kmalloc(size, GFP_KERNEL);
+  if (!buf)
+     return -ENOMEM;
+
+  res = single_open(file, show_idle, NULL);
+  if (!res) {
+     m = file->private_data;
+     m->buf = buf;
+     m->size = size;
+  } else
+     kfree(buf);
+  return res;
+}
+
#Proc-rajapinnan tiedosto-operaatiot (POSIX)
+static const struct file_operations cpu_idle_operations = {
+  .open          = idle_open,
+  .read          = seq_read,
+  .llseek        = seq_lseek,
+  .release = single_release,
+};
+
#Luo proc-rajapinta
+static int __init proc_idle_init(void)
+{
+  proc_create("cpu_idle", 0, NULL, &cpu_idle_operations);
+  return 0;
+}
+module_init(proc_idle_init);
```

LIITE 3: DOLOAD.SH-KOMENTOSARJA

Nimi:	Doload.sh-komentosarja järjestelmän kuormittamiseksi
Tyyppi:	Komentosarjatiedosto, joka käynnistää järjestelmää kuormittavia prosesseja. Liitteeseen lisätyt kommentit ilmaistaan #-merkillä ja lihavoidulla tekstillä.
Kuvaus:	Komentosarjan tarkoitus on käynnistää prosesseja, jotka kuormittavat järjestelmää mahdollisimman monipuolisesti. Kuormituksen kohteina ovat: • Verkkoyhteydet • Järjestelmäkutsut • Prosessori • Vuorontaja • Tiedostojärjestelmä Komentosarjan alkuperäinen toteutus: AT91 Linux4SAM [43]

```
#!/bin/sh
#Lataa isoa tiedostoa ftp-palvelimelta (verkkoyhteyksien kuormitus)
while [ 1 ] ; do wget -O - ftp://10.0.0.70/file > /dev/null 2>&1; done
&
a=$!

#Kopioi merkityksetöntä tietoa (prosessorin ja järjestelmäkutsujen
kuormitus)
dd if=/dev/zero of=/dev/null &
b=$!

#Vuorontojan kuormitus hackbench-ohjelmalla, joka vuorotellen käynnis-
tää ja tappaa useita rinnakkaisia prosesseja
while true; do killall hackbench  > /dev/null 2>&1; sleep 5; done &
d=$!
while true; do ./hackbench 1 > /dev/null 2>&1; done &
e=$!

#Tiedostojärjestelmän kuormitus rekursiivisella tiedostolistauksella
while true; do ls -lR / > /dev/null 2>&1; done &
f=$!

#Kiinteä aika (sekunteina), jonka jälkeen kuormitus lopetetaan
sleep 3600
echo "Stopping torture..."
kill $a $b $c $d $e $f
```