Metropolia

TEKNIIKKA JA LIIKENNE

Kone- ja tuotantotekniikka

Koneautomaatio

INSINÖÖRITYÖ

MITTAUS JA TIEDONKERUU QT-OHJELMOINTIYMPÄRISTÖLLÄ

Työn tekijä: Pasi Yrjölä
Työn ohjaaja: Jari Savolainen

Työ hyväksytty: ___. ___. 2011

Jari Savolainen
lehtori

Metropolia

TIIVISTELMÄ

Työn tekijä: Pasi Yrjölä

Työn nimi: Mittaus ja tiedonkeruu Qt-ohjelmointiympäristöllä

Päivämäärä: 14.3.2011 **Sivumäärä:** 33 s. + 2 liitettä

Koulutusohjelma: Kone- ja tuotantotekniikka

Suuntautumisvaihtoehto: Koneautomaatio

Työn ohjaaja: Jari Savolainen

Tässä insinöörityössä otettiin käyttöön Metropolia Ammattikorkeakoulun tilaama Advantech USB-4716 -tiedonkeruumoduuli sekä laadittiin graafisella käyttöliittymällä varustettu ohjelma, jonka avulla moduulia on mahdollista käyttää erilaisissa mittaustilanteissa. Ohjelmointi tehtiin Qt:lla, joka on avoimeen lähdekoodiin perustuva, C++ -ohjelmointikieleen pohjautuva ohjelmointiympäristö. Ohjelman tuli olla kattava ja helppokäyttöinen. Lisäksi sen tuli olla helposti muokattavissa moduulilla tulevaisuudessa tehtävää tiedonkeruuta varten.

Ohjelman ja tiedonkeruumoduulin tehokkuutta testattiin käytännössä mittauksissa, joissa sahoilla käytettävää tukin kuorimateriää iskettiin suurella, katosta roikkuvalla heilurilla. Mittauksissa selvitettiin voima- ja venymäliuska-antureiden avulla muun muassa erilaisten iskujen energiaa ja niiden vaikutusta terän kiinnityksen jännitystilaan. Mittausten onnistuminen edellytti mittausjärjestelmältä nopeutta ja tarkkuutta.

Työssä pyrittiin mahdollisimman edullisiin ratkaisuihin, millä pyrittiin luomaan vertailukohta kalliille teollisille tiedonkeruujärjestelmille. Teoriapohjaksi tutustuttiin tiedonkeruun ja signaalinkäsittelyn perusteisiin.

Avainsanat: Tiedonkeruu, mittaus, Qt, C++

Metropolia

ABSTRACT

Name: Pasi Yrjölä

Title: Measurement and data acquisition with Qt programming framework

Date: 14.3.2011 **Number of pages:** 33 + 2 appendices

Department: Mechanical engineering

Study Programme: Machine automation

Supervisor: Jari Savolainen

The goal of this final year project was to introduce the Advantech USB-4716 data aqcuisition module by making a program, which is used to control the device and provide a graphical user interface to aid in data aqcuisition and measurement situations. The programming was done using Qt, an open source programming framework, which uses custom C++ -based programming language. The objective of the program was to create a comprehensive basis for any future data acquisition with the device. At the same time it would need to be simple to use and modify.

The theoretical basis of this document consists of basics of data acquisition and signal conditioning. For the practical part, the efficiency of the data acquisition module and the program was tested in measurements, in which a log-peeling blade, used by lumber mills, was hit with a large pendulum. The goal of these measurements was to resolve the impact energy compared to the tensions in the blade's fastening. The measurements required high speed and precision from the data acquisition system to be successful.

The project was carried out by using low-cost solutions in order to create a comparison to expensive industrial data acquisition systems.

Keywords: data acquisition, measurement, Qt, C++

SISÄLLYS

1 JOHDANTO

Vielä ennen 2000-lukua tarkka teollinen tiedonkeruu oli kallis prosessi, ja sen käyttöä oli harkittava ja suunniteltava tarkkaan. Viime vuosien aikana markkinoille on kuitenkin tullut edullisia, mutta silti nopeita ja tarkkoja laitteistoja ja järjestelmiä. Näitä käytettäessä ohjelmointityö, joka laitteiston käyttämiseen tarvitaan, on kuitenkin usein tehtävä itse, kun taas kalliissa ja viimeistellyissä järjestelmissä käyttäjältä vaadittu työ on pyritty minimoimaan.

Tässä insinöörityössä otetaan käyttöön Metropolia Ammattikorkeakoulun tilaama Advantech USB-4716 -tiedonkeruumoduuli ja tehdään ohjelma, jolla moduulia käytetään ja mittausprosessin tulokset nähdään reaaliajassa ja saadaan myös helposti taulukkolaskentaohjelmaan. Tietopohjaksi tutustutaan tiedonkeruun periaatteisiin ja sen eri osa-alueisiin.

Aihetta lähestytään menetelmin, jotka ovat mahdollisimman kustannustehokkaita. Tiedonkeruumoduuli on hinnaltaan edullinen. Kaikki käytettävä ohjelmisto on vapaaseen lähdekoodiin perustuvaa ja siten ilmaista aina käyttöjärjestelmää myöten.

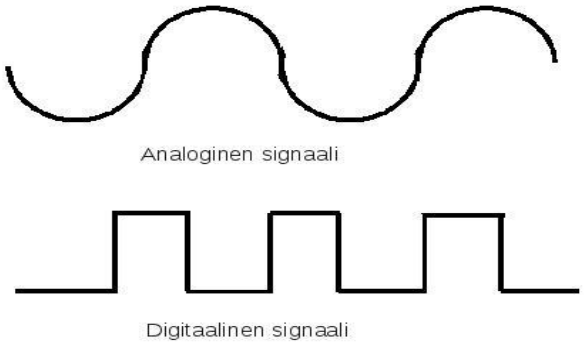
Ohjelma tehdään Qt-ohjelmointiympäristöllä, joka pohjautuu C++ -ohjelmointikieleen. Insinöörityön ymmärtämiseksi C++ -osaaminen ei ole kuitenkaan välttämätöntä, sillä itse lähdekoodia käsitellään hyvin vähän. Lähdekoodi on liitteenä työn lopussa. Aihetta tarkastellaan yleisemmin, koska tiedonkeruuohjelmointiin on muitakin tapoja kuin C++, ja kaikissa ei edes tarvita perinteistä tekstipohjaista ohjelmointitaitoa.

Ohjelmaa sovelletaan lopuksi mittauksiin, joiden toteutus on toisen opiskelijan insinöörityö. Mittauksissa selvitetään tukin kuorimaterän, jota käytetään sahalaitoksissa, optimaalinen iskuvoima tukkiin. Mittauksissa luetaan seitsemän venymäliuska-anturin ja yhden voima-anturin jännitettä.

Työn tavoitteena on lyhyesti sanottuna tutkia edullisen tiedonkeruujärjestelmän soveltuvuutta mittauksiin, joissa vaaditaan tarkkuutta ja nopeutta sekä saada aikaan hyvä ja kattava pohja tämän järjestelmän käyttämiselle tulevaisuudessa.

2 SIGNAALIT MITTAUSTEKNIIKASSA

Mittauksessa käytetyt anturit muuttavat mitattavan ilmiön signaaleiksi, joita voidaan lukea ja käsitellä. Signaalit voidaan jakaa kahteen ryhmään: analogiset ja digitaaliset. Analoginen signaali voi saada minkä tahansa arvon ajan funktiona, kun taas digitaalinen signaali saa joko arvon 1 tai 0. Digitaalista signaalia ei tule sekoittaa digitoituun signaaliin, joka tarkoittaa näytteistettyä, diskreettiaikaista analogiasignaalia (kappale 3.4). Kuvassa 1 on havainnollistettu analogisen ja digitaalisen signaalin eroa. [1.]



Kuva 1: Analoginen ja digitaalinen signaali

2.1 Analoginen signaali

Analogisissa signaaleissa tärkeitä piirteitä ovat signaalin oloarvo, muoto ja taajuus.

Oloarvo kertoo mitattavan ilmiön tilan, esimerkiksi paineen tai lämpötilan arvon kyseisellä hetkellä. Mittauksen tarkkuus on otettava oloarvoa tutkittaessa huomioon, sillä signaalin oloarvo mittalaitteessa voi erota merkittävästikin signaalin lähteen vastaavasta arvosta.

Signaaleja on usean muotoisia, ja signaalia tutkittaessa muodosta voidaan usein tehdä oleellisia päätelmiä. Esimerkiksi jyrkkäreunainen signaali on nopeasti muuttuvaa, ja jos signaalin häiriötaajuus on 50 Hz, syynä on usein sähköverkosta tuleva häiriö.

Signaalin taajuus määrittää sen, kuinka usein signaalin muoto toistuu. Taajuus pitää tietää useissa sovelluksissa, esimerkiksi äänisignaalissa tai seismisissä mittauksissa. Sen lisäksi taajuuden perusteella signaaleita voidaan

suodattaa käyttökelpoisemmiksi (kappale 2.3.1). Jos signaalin taajuuksia halutaan tarkemmin analysoida, on tehtävä Fourier-muunnos, joka on matemaattinen operaatio, joka kertoo sen, kuinka paljon signaalissa on eri taajuuksia. [1.]

2.2 Digitaalinen signaali

Digitaalisessa signaalissa kiinnostavat piirteet ovat tila ja muutosnopeus.Tila kertoo sen, onko tutkittava kohde päällä vai pois päältä tai auki vai kiinni. Esimerkiksi rajakytkimen tila nähdään tästä. Muutosnopeus kertoo sen, miten tila muuttuu ajan suhteen. Esimerkiksi moottorin pyörimisnopeus nähdään muutosnopeuden perusteella. [1.]

2.3 Signaalinkäsittely

Signaalinkäsittelyllä tarkoitetaan toimenpiteitä, joita tarvitaan signaalin muuntamisessa sovelluksen kannalta hyödyllisempään muotoon. Näitä toimenpiteitä ovat muun muassa signaalissa olevan kohinan poisto ja signaalissa olevien mielenkiintoisten piirteiden erottelu turhista. [2, s. 1.]

Signaalinkäsittely voidaan tehdä analogisesti tai digitaalisesti. Digitaalinen signaalinkäsittely on teollisuudessa jo kauan sitten ohittanut suosiossaan analogisen muun muassa paremman tarkkuutensa ja joustavuutensa ansiosta.

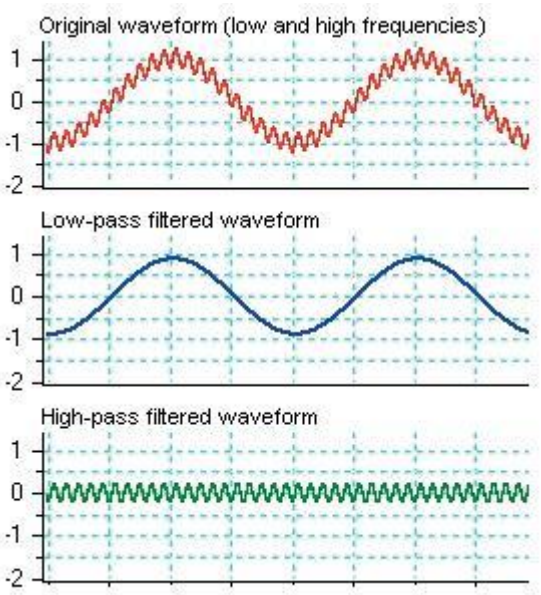
Digitaaliseen signaalinkäsittelyyn sisältyvät A/D-muunnos, eli analogisen signaalin muuttaminen digitaaliseksi, D/A-muunnos, joka on vastakkainen operaatio edellämainitulle, sekä itse suodatustoimenpiteet, jotka tehdään tavallisesti PC:llä tai digitaalisella signaaliprosessorilla. Jos suodatus tehdään analogisesti, suodatin koostuu elektroniikan komponenteista. [2, s. 5]

Signaalin suodatuksen tavoitteena on poistaa ei-toivottuja taajuuksia signaalista ja tehdä siitä siten helpommin luettavaa ja käsiteltävää. Suodatuksen voi tehdä ennen tiedonkeruutapahtumaa (laitesuodatus) tai jälkeenpäin, jolloin suodatus tehdään yleensä tietokoneella (ohjelmallinen suodatus). Jälkeenpäin tehtävän suodatuksen etuna on sen joustavuus, mutta haittana häiriöiden korostuminen laitesuodatukseen verrattuna. Yleisin suodatustapa on alipäästösuodin, muita yleisiä ovat ylipäästö- ja kaistanpäästösuodin.

Alipäästösuodin (low-pass filter) päästää läpi signaalissa tietyn rajan alapuolella olevat taajuudet ja tasoittaa sitä korkeammat taajuudet. Sitä käytetään yleensä poistamaan ympäristön häiriöitä, jolloin saadaan tasaisempi signaali. [3, s. 2] Keskiarvoistus, jota tässä työssä on käytetty, on omanlaisensa alipäästösuodin; se suodattaa nopeasti vaihtelevaa signaalia tasaisemmaksi.

Ylipäästösuodin (high-pass filter) on alipäästösuotimen vastakohta. Se pitää korkeataajuisen signaalin ennallaan ja tasoittaa matalat taajuudet. Sitä käytetään signaalin tasoittamiseksi siten, että voidaan tarkastella korkeita taajuuksia paremmin. Kuvassa 2 on havainnollistettu ali- ja ylipäästösuotimien toimintaa. [3, s. 4.]

Kaistanpäästösuodin (band-pass filter) on edellisten yhdistelmä. Se päästää läpi tietyllä alueella olevat taajuudet ja suodattaa sitä matalammat ja korkeammat taajuudet. Kaistanestosuodin (band-stop filter) on tälle vastakkainen toimenpide, eli se suodattaa pois tietyllä taajuusalueella olevat signaalit. [3, s. 5.]



Kuva 2: Ali- ja ylipäästösuotimien toiminta [3, s-4]

3 TIEDONKERUU

Tiedonkeruusta käytetään usein englanninkielistä lyhennettä DAQ (Data Acquisition). Sen tarkoitus on jonkin ilmiön, kuten jännitteen, lämpötilan tai paineen mittaaminen. Tiedonkeruujärjestelmän koostumuksen määrittelevät sen kohteen vaatimukset. [1.]

Tiedonkeruuta on käsitteenä vaikea määritellä yksiselitteisesti. Yhteistä asiantuntijoiden mielipiteille tiedonkeruun luonteesta ovat ainakin seuraavat seikat:

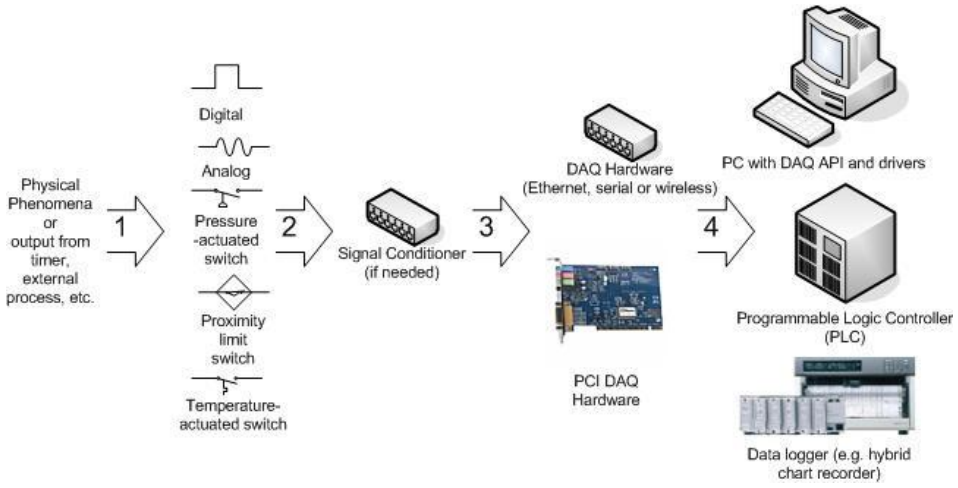
- Mittausvälineistön ohjelmointiin sekä tiedon tallennukseen ja käsittelyyn käytetään tietokonetta, joksi lasketaan mikä tahansa laite, joka tukee mittaukseen tarvittavaa käyttöjärjestelmää ja ohjelmistoa. [4, kpl 1, s. 2.]

- Mittausvälineistönä toimii tietokoneeseen liitettävä mittauskortti tai ulkoinen mittausväline. Tietokoneen ei tarvitse koko ajan hallita mittausta tai edes olla jatkuvassa yhteydessä mittausvälineistöön. [4, kpl 1, s. 2.]

- Mittausvälineistö voi suorittaa kerrallaan yhtä tai useampaa tiedonkeruu- ja hallintaprosessia, joita ovat muun muassa analogisten ja digitaalisten tulosignaalien lukeminen ja lähtösignaalien hallinta. [4, kpl 1, s. 2.]

Raja tiedonkeruuvälineen ja tietokoneen välillä on joskus vaikea määrittää, sillä jotkut tiedonkeruuvälineet toimivat nykyään lähes samalla tavoin kuin PC. Ne voivat esimerkiksi suorittaa mittauksia itsenäisesti ilman PC:tä ja tallentaa mittaustulokset muistitikulle tai -kortille, josta ne voidaan siirtää PC:lle analysoitavaksi. Joissain laitteissa voi olla myös sulautettu tietokone, joka toimii käytännössä samalla tavoin kuin tavallinen PC. [4, kpl 1, s. 2.]

Kuva 3 havainnollistaa teollista tiedonkeruuta ja sen eri osia.



Kuva 3: Teollinen tiedonkeruu. Nuolet kuvaavat signaalin kulkua. [5.]

3.1 Tiedonkeruuvälineistöt ja välineiden valinta

Tiedonkeruuvälineet jakautuvat kahteen ryhmään: tietokoneen sisäisiin tiedonkeruukortteihin ja ulkoisiin laitteisiin. Ulkoiset mittausvälineet ovat nykyään ominaisuuksiltaan suurin piirtein samanlaisia kuin sisäiset mittakortit. Ulkoisilla välineillä on kuitenkin se etu, että niihin eivät vaikuta tietokoneen sisäiset sähkö- ja magneettikentät niin vahvasti kuin sisäisiin tiedunkeruukortteihin.

Tavallinen tiedonkeruuväline koostuu ainakin A/D-muuntimesta, joka muuttaa tulevat analogiset signaalit digitaaliseksi tiedoksi, D/A-muuntimesta analogisen signaalin lähettämistä varten, I/O-ajurista digitaalisia tulo- ja lähtösignaaleja varten, sarjaliikennerajapinnasta, josta tieto lähetetään tietokoneelle sekä mikrokontrollerista, joka ohjaa kaikkea, mitä kortilla tapahtuu. Lisäksi välineessä on virransyöttöpiiri, joka on usein sarjaliikenneportin yhteydessä. [5.]

Tiedonkeruuvälineistöjä on valtavasti erilaisia ja ne tulisikin valita kiinnittäen huomiota käyttökohteen tarpeisiin. Pääsääntöisesti välineiden tarkkuus, nopeus ja toimintojen määrä lisääntyvät hinnan kasvaessa. [5.]

Erityisesti huomionarvoisia asioita ovat seuraavat seikat [5.]

- Analogiatulo-, analogialähtö- ja digitaalisten I/O -kanavien lukumäärä. Vähintään niin monta kuin tarvitaan eri tarkoituksiin. Analogialähtökanavia löytyy tiedonkeruulaitteesta yleensä enintään kolme, joten jos tarvitaan enemmän, on syytä hankkia erillinen analogialähtölaite.

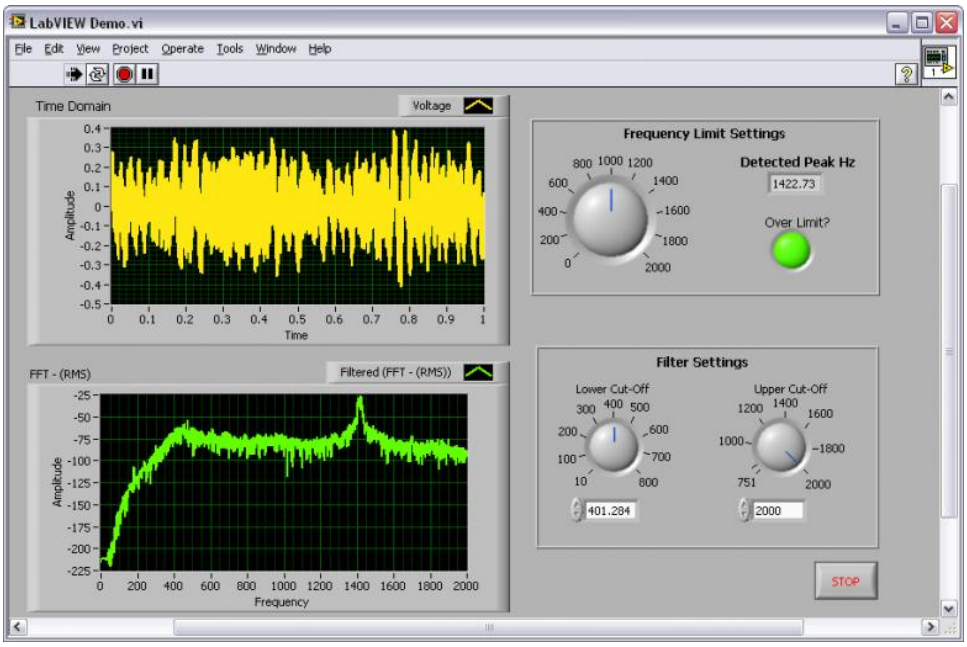
- Näytteenottotaajuus, joka määrittää sen, kuinka usein A/D-muunnin muuttaa analogiasignaalia digitaaliseksi. Tiedonkeruuvälinettä käyttävän PC:n käyttöjärjestelmä aiheuttaa hieman häiriötä mittaukseen, joten näytteenottotaajuuden tulisi olla jonkin verran suurempi, kuin mittauksessa tarvitaan. Myös nyquistin ehto, jota käsitellään kohdassa 3.5, on otettava huomioon, jos mitattava signaali on jaksottaista.

- Resoluutio eli erottelukyky, joka määrittää sen, kuinka suuri on pienin havaittava muutos jännitteen oloarvossa.

- PC-liitäntä, joka kannattaa valita tarpeen mukaan. USB on helppo liitäntätapa, mutta jos laitteita on useita, kannattaa muitakin vaihtoehtoja punnita. Myös langaton viestinlähetys on vaihtoehto.

3.2 Tiedonkeruuohjelmointi

Tiedonkeruu- ja mittausohjelma on yhtä tärkeä kuin välineistö, koska ilman sitä on mittauksia mahdotonta tehdä tavallisilla DAQ-korteilla ja -moduuleilla. Tiedonkeruu- ja mittausohjelman tekemiseen on monta tapaa. On olemassa useita kaupallisia ohjelmistoja, joilla voi tehdä halutunlaisen mittausjärjestelyn ilman varsinaista ohjelmointia. Joillain voi myös tehdä graafisen käyttöliittymän mittaustiedon visualisoimiseksi (kuva 4). Tunnetuin näistä lienee National Instrumentsin LabVIEW. Lisäksi useimmat DAQ-laitteistojen valmistajat toimittavat laitteen mukana ohjelmalliset komponentit, kuten kirjastot, jotka antavat mahdollisuuden tehdä tiedonkeruu- ja mittausohjelma itse käyttäen jotain yleisistä ohjelmointikielistä.



Kuva 4: LabVIEW-ohjelmistolla tehty graafinen käyttöliittymä

Tässä työssä keskitytään ohjelmointiin C++ -ohjelmointikielellä käyttäen ilmaista Qt-ohjelmointiympäristöä, joka esitellään osiossa 4.3.

Ilmaisten ja kaupallisten ohjelmointitapojen vertailu

Tässä työssä ei perehdytty kovin syvällisesti kaupallisiin tiedonkeruu- ja mittausohjelmointiympäristöihin, kuten LabVIEWiin, mutta saatiin kuitenkin luetun perusteella kohtuullinen näkemys niistä. Ne ovat tehokas ja yksinkertainen työkalu mittausohjelmien tekemiseen eivätkä vaadi paljoa kokemusta ohjelmoijalta. Ne ovat kuitenkin usein varsin hintavia eivätkä välttämättä sovellu jokaiseen tarkoitukseen.

Työssä käytetty C++ -ohjelmointi on joustavaa ja monipuolista, mutta oppimiskynnys on melko korkea, ja etenkin monimutkaisten rakenteiden hallitseminen vaatii kokemusta. Ohjelmointiympäristöt, kuten työssä käytetty Qt, kuitenkin helpottavat ohjelmointia ja antavat kokemattomammallekin ohjelmoijalle paremman mahdollisuuden käyttää kielen koko potentiaalia. C++ -ohjelmointia varten on ilmaisia ohjelmointiympäristöjä, ainakin avoimen lähdekoodin kirjoittamista varten.

C++, kuten muutkin perinteiset ohjelmointikielet, on melko hidasta ja usein työlästä kirjoittaa. Kaupallisten ohjelmointitapojen etu onkin, että ne säästä-

vät aikaa ja vaivaa. Paras ratkaisu omaan tarpeeseen tulisi valita sen perusteella, onko kaupallisella ohjelmistolla saavutettu ajan säästö suurempi kuin C++:lla tehdessä säästetyt kulut. Tähän vaikuttavat monet tekijät, kuten kokemustaso C++:sta, mittausten järjestelmällisyys sekä erilaisten mittausten määrä ja monipuolisuus.

On myös syytä ottaa huomioon, että monissa, etenkin joihinkin tiettyihin mittauksiin erikoistuneissa, tiedonkeruulaitteistoissa tulee mukana ohjelmisto ennalta määriteltyihin mittauksiin. Kolmannen osapuolen tekemiä valmiita ohjelmia tai osittain valmiita pohjia, maksullisia ja ilmaisia, löytyy joillekin laitteille ja järjestelmille myös internetistä. Näitä hyödyntämällä voi säästää ajassa ja usein myös kustannuksissa.

3.3 Analogiasignaalien mittausmetodit

Analogiaviestejä voidaan vastaanottaa DAQ-laitteilla kahdella tavalla: single-ended -metodilla tai differentiaalisesti. Ensin mainitussa metodissa mittauskohteen plus-puoli yhdistetään yhteen mittakortin AI-kanavaan ja miinuspuoli maakanavaan (GND). Differentiaalisessa mittaustavassa molemmat puolet yhdistetään AI-kanaviin, jolloin kanavia on käytössä luonnollisesti puolet vähemmän. Lopullinen jännite lasketaan tällöin kanavien jännite-erosta.

Single-ended -metodi on herkkä ulkoisille häiriöille ja kohinalle, joten se ei ole hyvä vaihtoehto matalan jännitetason tarkkaan mittaamiseen. Lisäksi single-ended -signaali ei sovi korkean mittaustaajuuden sovelluksiin, koska kapasitanssi ja induktanssi suodattaisivat tällöin signaalia haitallisesti. Korkeampien jännitteiden ja hitaamman signaalin mittaukseen se kuitenkin sopii. Single-ended -metodissa on se etu, että itse signaali tarvitsee vain yhden johdon. [6, s. 2.]

Differentiaalinen mittaustapa sietää kaksi kertaa enemmän ulkoisia häiriöitä kuin single-ended. Differentiaalisella mittaustavalla ei ole myöskään vaaraa, että kanavien miinusjännitteet vaikuttaisivat toisiinsa yhteisestä maasta johtuen, mikä on mahdollista single-ended -metodissa. Differentiaalinen signaali on siis käyttökelpoisempi, kun jännitetaso on alhainen, taajuus on korkea tai häiriötä on paljon. [6, s. 4.]

3.4 A/D-muunnos

A/D-muunnin muuttaa analogisen signaalin digitaaliseksi tiedoksi eli digitoi sen. Analogisen signaalin jännitteen oloarvo tarkastetaan ja koodataan digitaaliseksi arvoksi aina tietyin väliajoin. Tätä aikaa kutsutaan näyteväliksi. Usein puhutaan myös näytetaajuudesta, joka on näytevälin käänteisarvo. Jännitteen arvo poimitaan hetkeksi sample and hold -piiriin, jossa se pysyy vakiona niin kauan, että A/D-muunnin saa arvon koodattua digitaaliseksi. Näyteväli on riippuvainen sample and hold -piirin nopeudesta poimia uusi arvo. [7, s. 11 - 14.]

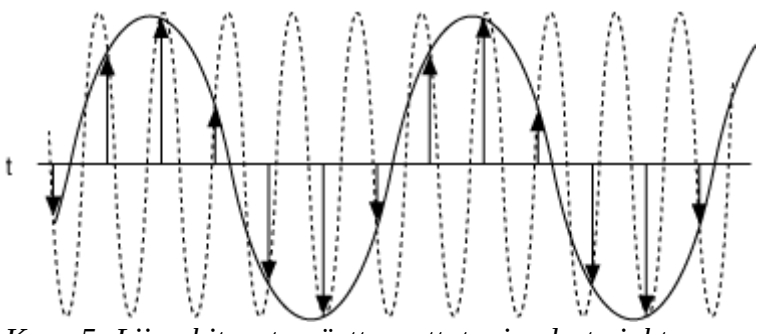
Muunnoksen tarkkuuden kertoo muuntimen erottelukyky eli resoluutio. Resoluutio ilmoitetaan bitteinä: erottelutasoja, eli eri arvoja, jotka signaalia vastaavalla digitaalisella luvulla voi olla, on 2^n kappaletta, jossa n on resoluutio bitteinä. [7, s. 11 - 14.]

D/A-muunnos on A/D-muunnoksen vastakkainen toiminto, eli digitoidun signaalin muuntaminen analogiseksi.

3.5 Nyquistin ehto

Nyquistin ehdon mukaan näytteenottotaajuuden tulee olla vähintään kaksinkertainen korkeimpaan mitattavassa signaalissa olevaan taajuuteen verrattuna. Jos näin ei ole, tulokset vääristyvät ns. laskostumisilmiön (aliasing) takia. [8, s. 1.]

Kuvassa 5 katkoviivalla merkittyä 5 MHz:n taajuista signaalia näytteistetään 6 MHz:n taajuudella. Nuolet osoittavat hetket, joilla näyte on otettu ja yhtenäinen viiva on niiden kautta piirretty kuvaaja. Kuten kuvasta käy ilmi, mittaustulokset ovat vääristyneet ja on saatu 1 MHz:n taajuista signaalia 5MHz:n sijaan. [9.]

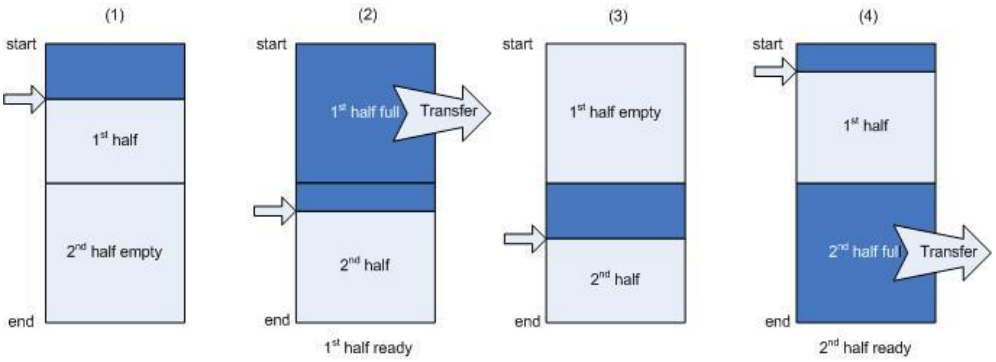


Kuva 5: Liian hitaasta näytteenottotaajuudesta johtuva laskostuminen

3.6 Tiedon puskurointi

DAQ-korttien ja -laitteiden avulla voi lukea analogiasignaalia hyvinkin suurella taajuudella, joillakin nykyään jopa miljoonia näytteitä sekunnissa. Luettaessa kanavia suoraan tallentamatta tietoa mihinkään ei voida kuitenkaan saavuttaa korkeaa näytteenottotaajuutta. Nopeaan lukemiseen on käytettävä laitteen sisäistä FIFO-puskuria (First in, first out), joka löytyy käytännössä jokaisesta DAQ-laitteesta.

FIFO-puskurin tehtävä on pitää tiedon lukeminen vakaana mittausohjelman pyöriessä ja tallentaessa tietoa kovalevylle. Puskuri toimii siten, että mitattava tieto tallentuu siihen näyte kerrallaan, ja ohjelma ottaa sitä vastaan paketteina. Normaalisti ohjelma saa luvan ottaa vastaan paketillisen tietoa silloin, kun puolikas FIFO-puskurista on tullut täyteen, kuten kuvassa 6 on havainnollistettu. Tällöin puskuri jatkaa tiedon lukemista toiseen puolikkaaseen sillä välin, kun täysi puolikas lähetetään ohjelmalle. Ohjelman on luonnollisesti vastaanotettava tietopaketteja nopeammin kuin puskurin puolikkaan täyttyminen kestää, muuten tietoa menee hukkaan. [4, kpl 3, s. 12 - 13; 10.]

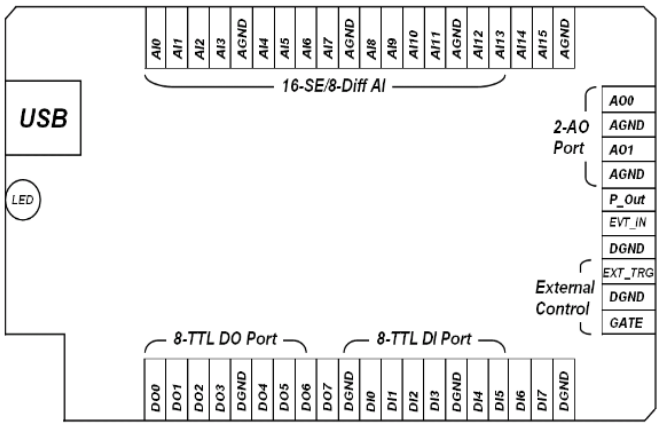


Kuva 6: Fifo-puskurin toiminta [10.]

4 TYÖN LÄHTÖKOHDAT

4.1 Laitteisto

Työssä käytettiin Advantech USB 4716 -tiedonkeruumoduulia, joka on tietokoneen ulkoinen laite. USB-moduuli liitetään tietokoneeseen nimensä mukaisesti USB-johdolla, jota kautta se saa myös käyttöjännitteensä ja analogialähtöjen jännitteen. Muuta yhteyttä tietokoneeseen tai virtalähteeseen ei tarvita. Kuvassa 7 näkyy moduulin liitäntäkaavio. Kuva itse moduulista mittaustilanteessa on osiossa 6.1.



Kuva 7: Advantech USB-4716 –tiedonkeruumoduulin kytkentäkaavio [11, s. 13.]

Moduulissa on 16 analogiatulokanavaa, kaksi analogialähtökanavaa, 16 digitaalista I/O-kanavaa sekä yksi 32-bittinen laskuri. Laite pystyy ottamaan vastaan 200 000 näytettä sekunnissa. [11, s. 2.]

Moduulilla on 16 bitin resoluutio, eli sen erottelutarkkuus on 65536 arvoa. Tulojen ja lähtöjen jännitteen vaihteluväli voi olla korkeintaan +- 10 V, ja sitä voidaan ohjelmallisesti säätää portaittain resoluution optimoimiseksi kuhunkin käyttötarkoitukseen. Esimerkiksi vaihteluvälillä +- 10 V yksi erotteluväli on 20 V / 65536 =~ 0,31 mV mutta vaihteluvälillä 0 – 2,5 V yksi väli on 2,5 V / 65536 =~ 0,038 mV. [11, s. 2.]

Moduulissa on FIFO-puskuri, jonka koon voi säätää ohjelmallisesti. Puskuriin tallentuu tietoa siten, että AI0-kanavasta lähtien jokaisesta käytössä olevasta kanavasta tallennetaan vuorotellen yksi arvo. Kun jokaisen kanavan arvo on tallennettu, aloitetaan kierros alusta. Esimerkiksi kolmea kanavaa käytettä-

essä puskurin ensimmäinen arvo on AI0:sta, toinen AI1:stä kolmas AI2:sta ja neljäs taas AI0:sta. Tämä on otettava huomioon, kun mittaustietoa käsitellään ohjelmallisesti.

Lisäksi työtä varten oli käytössä kannettava tietokone, johon asennettiin käyttöjärjestelmä ja käytettävän moduulin laiteajurit. Tietokoneeseen asennettiin myös QT-kehitysympäristö ja Qwt-lisäosa, joiden avulla itse ohjelmointi ja käyttöliittymän suunnittelu tehtiin.

4.2 Käyttöjärjestelmä

Työn osaksi otettiin edullisempien vaihtoehtojen etsiminen kaupallisille tiedonkeruu- ja mittauskäytännöille, joten pyrittiin alusta asti tukeutumaan ilmaisiin ja avointa lähdekoodia painottaviin ratkaisuihin. Tästä syystä käyttöjärjestelmäksi valittiin Debian 4.0, joka on avoimen lähdekoodin Linux-jakelupaketti. Advantechin DAQ-laitesarjalle on myös olemassa valmiit ajurit Debian 4.0:aan, mikä oli kriittinen tekijä käyttöjärjestelmän valinnassa.

Windows olisi ollut tutumpi käyttöjärjestelmä ja DAQ-moduulin laiteajureita olisi ollut helpompi käyttää, mutta Windowsissa on muun muassa se haittapuoli, että se suorittaa taustalla monia prosesseja, jotka voivat pahimmassa tapauksessa häiritä huomattavasti nopeasti toimivan mittausohjelman toimintaa [4, kpl 3, s. 14 - 15]. Linux on jokseenkin vakaampi käyttöjärjestelmä. Windows-lisenssi on lisäksi maksullinen, joskin se tulee mukana lähes kaikissa PC:issä.

4.3 Qt-ohjelmointiympäristö

Qt on norjalaisen Trolltechin kehittämä ja vuonna 2008 Nokian ostama ohjelmien ja graafisten käyttöliittymien kehitysympäristö. Se on alustariippumaton, eli se toimii missä tahansa yleisistä käyttöjärjestelmistä. Pohjimmiltaan Qt on C++ -luokkakirjasto, jonka käyttämät tiedostot ovat samoja kuin missä tahansa muussa C++ -kehitysympäristössä. Se on avoimen lähdekoodin kirjoittamista varten ilmainen ohjelmisto, mutta sitä myydään myös lisensseinä, joilla voi kirjoittaa suljettua, kaupallista lähdekoodia. [12, s. 3 - 4.]

Qt:n ehkä tärkein ominaispiirre ovat signaalit ja slotit. Ne muodostavat mekanismin, jolla oliot viestittävät toisilleen. Muun muassa käyttöliittymäelementit lähettävät signaalin silloin, kun jotain tapahtuu. Ohjelmoija voi tehdä

erityisen funktion, slotin, ja yhdistää signaalin tähän slotiin connect-funktion avulla. Tällöin slot-funktio suoritetaan silloin, kun signaali lähetetään. Esimerkiksi Sulje-painikkeen clicked-signaali voidaan yhdistää ohjelman quit-slotiin, jolloin painamalla Sulje-painiketta ohjelma sammuu. Esimerkkikoodissa 1 on tehty kyseinen yhdistäminen. [13.]

```
connect(ui>Sulje,SIGNAL(clicked(),this,SLOT(close())));
```

Esimerkkikoodi 1: Signaalin ja slotin yhdistäminen connect-funktiolla

Qwt eli *Qt Widgets for Technical Applications* on QT-kirjasto, joka lisää QT:hen käyttöliittymäkomponenttejä lähinnä teknisiin sovelluksiin. Qwt:n lisäämät komponentit ovat tehokkaita ja hyvin toteutettuja, mutta Qwt:n niukka dokumentointi tekee niistä hankalia käyttää kokemattomalle Qt-ohjelmoijalle. Tässä työssä käytettiin plot-komponenttia, jolla piirretään kuvaajia annetuista double-tyyppisistä arvoista.

Qwt:ta käytettäessä Qt:n projektitiedostoon (.pro) on lisättävä tieto siitä, että Qwt on mukana ohjelmassa. Tarkemmat ohjeet ovat liitteessä 1.

4.4 Laiteajurien käyttöönotto

Laiteajuri on ohjelmallinen rajapinta tietokoneen käyttöjärjestelmän ja minkä tahansa tietokoneeseen liitetyn laitteen välillä. Ajuri on abstraktiotaso, joka suojaa käyttäjää laitteen monimutkaisuudelta tehden laitteen käytöstä käyttäjälle yksinkertaista. Kaikki komennot tietokoneelta laitteelle ja viestit laitteelta tietokoneelle kulkevat ajurin kautta, joten se on kriittinen osa laitteen ja tietokoneen välisessä yhteydessä. [4, kpl 3, s. 7.]

Advantech USB -sarjan laiteajurit olivat helpot asentaa Debian 4.0:lle, mutta käyttöönotto ei ollut täysin ongelmatonta. Ensin piti suorittaa template-tiedosto, jossa on määritelty käytettävät laitteet, tässä tapauksessa vain USB-4716. Tämän jälkeen piti sitoa laite tiedostoon, jonka avulla laite avataan mittausohjelmassa funktiolla DRV_DeviceOpen. Lisäksi piti vielä antaa Linux-jakeluille yhteiselle laiteajurien kansion /dev sisältämille tiedostoille oikeudet lukea ja kirjoittaa. Nämä toimenpiteet täytyy myös tehdä aina kun tietokone käynnistetään uudestaan, jotta ohjelma saa yhteyden käytettävään laitteeseen. Ajurien käyttöönotto-ohjeet ovat liitteessä 1.

5 MITTAUSOHJELMA

Työn tärkein tavoite oli laatia ohjelma, jonka avulla voidaan tehdä monipuolisesti mittauksia käyttäen USB-4716 -moduulia. Ohjelmasta tuli tehdä joustava ja toimiva mahdollisimman laajassa skaalassa mittaustilanteita, mutta myös selkeä ja yksinkertainen kokemattomallekin käyttäjälle. Tässä osiossa käsitellään työn tuloksena syntynyttä ohjelmaa ja sen eri osa-alueita.

Ohjelma laadittiin valmiin pohjan päälle, jossa oli muun muassa valmis rakenne ja yksinkertainen käyttöliittymä. Lisäksi käytössä oli DAQ-moduulin dokumentaation mukana toimitettuja yksinkertaisia C-kielisiä esimerkkitiedostoja moduulin käyttöön tarvittavista funktioista.

Ohjelmointiprosessi aloitettiin heti työn alettua ja jatkui läpi koko työn. Alussa kehitettiin ohjelman runko valmiiksi ja keskityttiin DAQ-moduulin toimintaan ja käyttöön liittyviin funktioihin. Alkuvaiheessa tehtiin paljon yhteistyötä toisen samanmallista DAQ-moduulia käyttävän insinöörityön tekijän kanssa moduulin käyttöön tarvittavien kirjastojen toiminnan selvittämiseksi ja yhteisen pohjan tekemiseksi. Loppuvaiheessa keskityttiin ohjelman yksityiskohtiin ja ominaisuuksien lisäämiseen

5.1 Ohjelman toiminta

Ohjelma sisältää kolme erillistä osiota, jotka toimivat toisistaan riippumatta omissa säikeissään (thread). Yksi säie lukee USB-modulia, käyttää puskureita ja tekee tarvittavat tiedonkäsittelytoimenpiteet. Toinen käsittelee käyttöliittymää, eli näyttää arvot, piirtää kuvaajat ja ottaa vastaan käyttäjän painallukset ja parametrien muutokset. Kolmas säie käyttää analogialähtökanavaa, jota pohdittiin mahdolliseksi syöttöjännitteeksi mittaussovelluksiin, mutta jonka myöhemmin todettiin soveltuvan siihen huonosti.

Säikeistys on ohjelmassa tärkeää, koska se poistaa käyttöliittymästä viiveen, joka aiheutuu muiden prosessien ajamisesta samaan aikaan. Ilman säikeistystä prosessin pitäisi suoriutua loppuun ennen kuin käyttöliittymässä tehdyt toimenpiteet, esimerkiksi napin painallus, tulisivat voimaan. Lisäksi säikeistys nopeuttaa ohjelmaa salliessaan käyttöliittymän ja prosessien toimia samaan aikaan, jos vain tietokoneen laskentateho riittää siihen. Säikeet eivät

voi käsitellä toistensa sisältämiä muuttujia, mutta ne voivat lähettää toisilleen tietoja signaalien avulla (ks. kohta 4.3).

Qt Creatorilla säikeiden tekeminen on melko yksinkertaista, sillä Qt:ssa on valmis luokka säikeille: QThread. Uuden säikeen voi tehdä periyttämällä säikeen luokka QThreadista kuten esimerkkikoodissa 2, jossa mittausohjelman mittaussäikeen luokka tehdään.

```
class MeasurementThread : public QThread
```

Esimerkkikoodi 2: Säikeen periyttäminen luokasta QThread

5.1.1 Laitteen ja analogiatulojen alustus

Alustukset ovat toimintoja, jotka tehdään kertaalleen ohjelman alussa. Painettaessa käyttöliittymän Start-painiketta, ohjelma alustaa USB-4716-laitteen suorittamalla InitDevice-funktion, eli käynnistää sen ja tunnistaa, että juuri tätä laitetta käytetään. Tähän kohtaan voisi pienellä muutoksella koodiin asettaa käytettäväksi minkä tahansa laitteen, joka käyttää samoja funktioita kuin USB-4716.

Tämän jälkeen alustetaan analogiatulot, eli ajetaan InitAnalogInputs-funktio, jossa määritellään käytettävät mittausfunktiot ja -metodit. Ohjelma hakee myös tässä yhteydessä käyttöliittymästä asetetut parametrit, jotka liittyvät analogiatulojen mittaukseen. Nämä parametrit tulee siis säätää mittausta varten kohdalleen ennen lukemisen käynnistystä.

Painettaessa käyttöliittymän Stop-painiketta ohjelma katkaisee yhteyden laitteelle. Tämän jälkeen ohjelma tekee alustukset uudestaan painettaessa Start-painiketta.

5.1.2 Analogiatulojen lukeminen

Alustusten jälkeen ohjelma siirtyy silmukkaan, jossa analogiatulot luetaan puskurista, tehdään halutut tiedonkäsittelyoperaatiot ja siirretään lopuksi käyttöliittymälle tulostettavaksi. Lukeminen toimii ns. keskeytyksillä (interrupt), eli ohjelma odottaa signaalia tapahtumasta (event), jonka jälkeen se siirtyy kyseistä tapahtumaa vastaavaan ohjelmahaaraan. Tapahtumat ”High buffer ready” ja ”Low buffer ready” ovat toivottuja tapahtumia, eli niitä, joita

pitäisi tulla tasaiseen tahtiin ohjelman toimiessa oikein. Ne tarkoittavat sitä, että puskurin ylempi tai alempi puolikas on täynnä (ks. kohta 3.6).

Jos saadaan toivotun tapahtuman signaali, siirretään arvot puskurista lukujonoon, jossa ne ovat niin kauan kuin uusi puskurin puolikas tulee valmiiksi, minkä jälkeen uuden puskurin puolikkaan arvot korvaavat vanhat. Arvoille pitää tehdä halutut toimenpiteet, kuten tallennus, piirto kuvaajaan tai käsittely, ennen kuin uusi puskurin puolikas tulee valmiiksi. Muuten tapahtuu puskurin ylivuoto (overrun), ja tieto, jota ei ehditty siirtää, menee hukkaan.

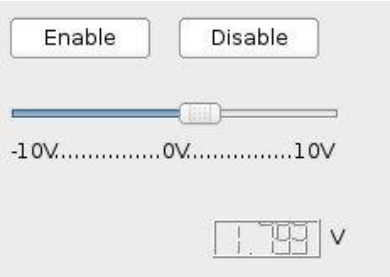
Ylivuodon lisäksi toinen ei-toivottu tapahtuma on ”timeout”, josta signaali tulee silloin, kun mitään muuta tapahtumaa ei tule tietyn ajan sisällä. Tämä aika voidaan säätää alustuksen yhteydessä. Tässä ja ylivuodon tapauksessa tulostetaan käyttöliittymään teksti, joka kertoo kyseisestä tapahtumasta. Tällöin käyttäjä tietää, että tiedonsiirrossa tai hänen säätämissään asetuksissa on jotain vikana.

Myös alustusvaiheessa tulevat virheet tulostetaan käyttöliittymään. Kuvassa 8 on tilanne, jossa käyttöliittymän tekstilaatikossa on analogiatulojen alustuksen yhteydessä tullut virheilmoitus koodilla 3. Virhekoodit ja niiden merkitykset on luetteloitu Advantechin DAQ-sarjan laiteajurin käyttöohjeissa.

Kuva 8: Käyttöliittymän Status-tekstilaatikko, jossa näkyy virheilmoitus

5.1.3 Analogialähdön käyttö

Ohjelman käyttöliittymään tehtiin välilehti analogialähtökanavan AO0 hallitsemiseksi. Kanavasta voi lähettää +-10 V jännitettä. Analogialähdön käyttö tehtiin omaan säikeeseensä, sillä se on riippumaton mittaussäikeestä ja käyttöliittymästä, jonka tapahtumat välitetään säikeelle signaaleina. Kuvassa 9 on analogialähdön hallintapaneeli käyttöliittymässä.



Kuva 9: Analogialähdön hallinta käyttöliittymässä

Analogialähdön käyttämistä yritettiin aluksi soveltaa osion 6 mittauksissa venymäliuska-antureiden ja voima-anturin vahvistimen käyttöjännitteen syöttämiseen, mutta kanavan lähettämä analogiasignaali todettiin huonolaatuiseksi ulkoisiin virtalähteisiin verrattuna, sillä signaalin havaittiin sisältävän paljon kohinaa. Lisäksi kanava ei pystynyt syöttämään paljoa virtaa, sillä sitä rajoittaa USB-väylän virransyöttökyky, josta myös itse DAQ-moduuli ottaa osansa käyttövirtana. Moduulin analogialähtö ei siis sovellu kohtuullistakaan tarkkuutta vaativiin mittausteknisiin sovellutuksiin, mutta esimerkiksi toimilaitteiden ohjausjännitteeksi analogialähtö lienee sopiva.

5.1.4 Tiedon tallennus

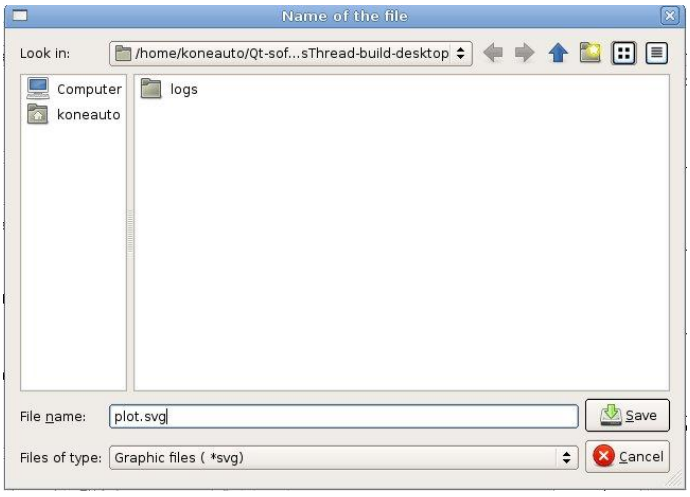
Useimmissa mittauksissa ei riitä, että saadaan piirrettyä näytölle kuvaaja. On myös ensiarvoisen tärkeää saada mittaustulokset otettua ylös numeerisina arvoina, jotta niitä voidaan tarvittaessa käsitellä ja analysoida tarkemmin jälkeenpäin.

Ohjelmaan tehtiin toiminto, joka tallentaa mitatut arvot tekstitiedostoon. Toiminto tekee kaikille mittauksessa mukana oleville kanaville oman sarakkeensa, ja ensimmäiseksi sarakkeeksi tulee kulunut aika mikrosekunteina. Sarakkeet erotellaan toisistaan pilkuilla, jolloin tiedoston saa helposti siirrettyä taulukkolaskentaohjelmaan, jossa tietoja voi käsitellä.

Kulunut aika näytteiden välillä lasketaan mittaustaajuuden perusteella, sillä Advantech USB-4716 -moduulissa ei ole sisäistä kelloa, joka ilmoittaisi tarkan ajan. Käyttöjärjestelmän prosessien mittaukselle aiheuttama häiriö sekoittaa kuitenkin hieman mittaustaajuutta, jolloin kulunut aika jää hieman jälkeen oikeasta ajasta. Jos tarkkaa aikaa tarvitaan, on ajan laskenta kalibroitava, eli esimerkiksi mitatattava, kuinka paljon laskettu kulunut aika jää jälkeen oikeasta ajasta ja kerrottava varsinaisissa mittauksissa kulunut aika tällä kertoimella. Ongelmaksi muodostuu kuitenkin se, että käyttöjärjestelmän aiheuttaman häiriön määrä ei ole aina täysin vakio.

Tallennustoimintoon tehtiin myös mahdollisuus näyttää yhdessä sarakkeessa tietokoneen kellonaika Qt:n QTime-luokan avulla. Tästä ei kuitenkaan ole paljoa hyötyä mittauksien kannalta, sillä tieto tulee tietokoneelle paketteina, minkä takia tarkkaa aikaa mittaustiedon eri alkioille ei saada. Nähdään siis vain se, milloin tietopaketti on vastaanotettu. Lisäksi QTime pystyy näyttämään kellonajan vain millisekuntien tarkkuudella, mikä on liian pitkä väli tarkimpiin mittauksiin. Tietokoneen kellon avulla voidaan kuitenkin esimerkiksi tehdä edellä mainittu ajan kalibrointi ilman ulkoista kelloa.

Painettaessa käyttöliittymän ”Enable datalogging” -checkbox-kytkintä ohjelma avaa dialogi-ikkunan, joka kysyy käyttäjältä nimen ja tallennuskansion tekstitiedostolle (kuva 10) ja aloittaa tämän jälkeen tietojen tallennuksen. Tietojen tallennus lopetetaan, kun kytkintä painetaan uudestaan. Tiedoston tallentamistiedoston nimen ja paikan kyselyä varten saa tehtyä dialogi-ikkunan helposti Qt:n QFileDialog-luokan avulla.



Kuva 10: Dialogi-ikkuna tiedon tallennusmistiedoston luomiseen

Mitattaessa suurilla näytteistystaajuuksilla tallennustiedostoon tulee jo lyhyessä ajassa valtava määrä rivejä. Taulukkolaskentaohjelmaan ei kuitenkaan mahdu kuin vähän päälle 65 000 riviä, mikä voi joskus aiheuttaa ongelmia tai vähintään ylimääräistä työtä, kun mittaajan pitää poistaa merkityksettömät arvot tiedostosta itse. Ohjelmaan tehtiin tämän takia toiminto, jolla voidaan käyttöliittymässä määrittää, että tietoja tallennetaan vain silloin, kun kanavan AI0, tai kaikkien käytössä olevien kanavien, oloarvo jossain kohdassa mittausarvojen lukujonoa ylittää raja-arvon, joka voidaan käyttöliittymästä säätää halutun suuruiseksi. Näin voidaan poistaa pitkässä tai hyvin taajassa mittauksessa tulevat merkityksettömät arvot mittauksen alusta ja lopusta. Toiminnossa on kuitenkin se vaara, että jos tulokset ovat mittauksen keskellä pitkään raja-arvon alapuolella, mittauksesta voivat jäädä nämä arvot pois.

5.1.5 Ohjelmallinen keskiarvotus

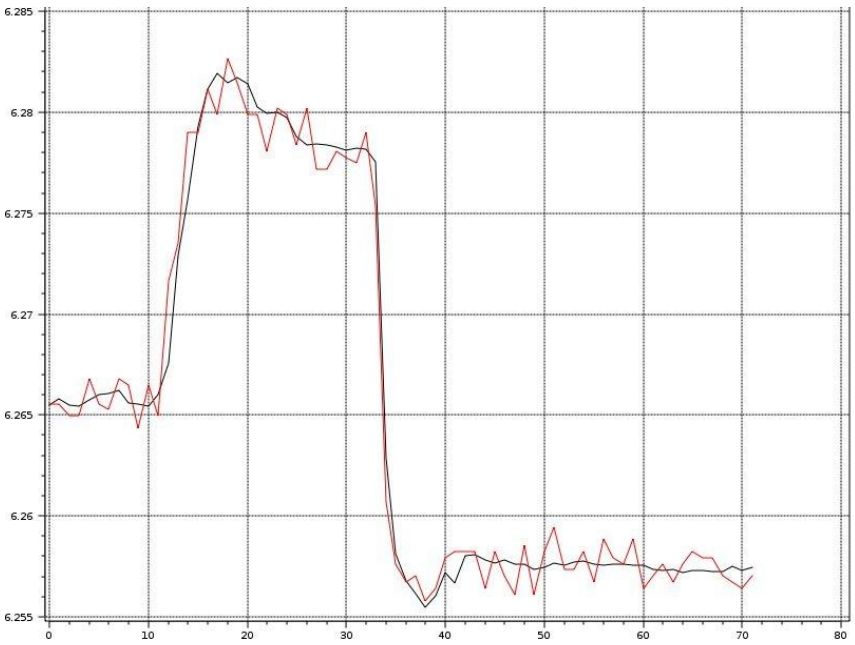
Käytännössä kaikissa mitattavissa signaaleissa on ulkoisten häiriöiden tai muiden mittaukseen liittyvien tekijöiden takia paljon kohinaa tai karkeutta. Sen takia ohjelmaan oli syytä tehdä jonkinlainen signaalin suodatustoiminto. Koska käsiteltiin pitkähköä lukujonoa kerralla hyvin nopeasti, suodatustoiminnon tuli olla toiminnaltaan kevyt ja nopea, jottei se kuluttaisi mittaustietokoneen resursseja paljon, eikä ohjelman muu toiminta häiriintyisi.

Ohjelmaan lisättiin aritmeettisen keskiarvon laskenta, joka on tehokas ja yksinkertainen keino poistaa kohinaa. Keskiarvon laskeminen vie lähes olemattoman määrän aikaa prosessorilta, koska kyseessä on yksinkertaisia kerto- ja jakolaskuja. Riskinä keskiarvotuksessa on kuitenkin nopeasti muuttuvan tiedon vääristyminen tai häviäminen. Lisäksi signaalia on näytteistettävä niin monta kertaa nopeammin kuin arvoja keskiarvoistetaan, jotta päästään samaan lukunopeuteen kuin ilman keskiarvotusta. Esimerkiksi otettaessa keskiarvo neljästä arvosta saadaan vain yksi lopullinen arvo, eli kolme arvoa neljästä jää pois. Liukuvassa keskiarvossa ei olisi tätä ongelmaa, mutta se vääristäisi tulosta enemmän ja olisi vaikeampi tehdä tähän ohjelmaan, koska tieto tulee paketeissa eikä arvo kerrallaan.

Keskiarvon laskenta tehdään erikseen ja eri tavalla kuvaajalle ja tiedon tallennukseen, ja niiden käytöstä ja vahvuudesta päätetään myös käyttöliittymässä erikseen.

Kuvaajaan piirretään oletusarvoisesti mittausarvojen lukujonon ensimmäinen arvo kustakin käytettävästä kanavasta, joten keskiarvo lasketaan muista lukujonossa olevista arvoista, jotka ovat muuten kuvaajan osalta käyttämättä. Tiedon tallennustoimintoon keskiarvo lasketaan mittaustiedot sisältävän lukujonon peräkkäisistä arvoista, ja sen vahvuuden on oltava jaollinen lukujonon pituudella, eli oletusarvoisesti 512, jotta keskiarvot menevät tasan lukujonon kesken. Siksi tiedon tallennuksen keskiarvoistuksen vahvuuden valintaelementissä on ennalta määrätyt arvot: 4, 8, 16 jne., toisin kuin kuvaajan vastaavassa.

Kuvassa 11 on havainnollistettu keskiarvotuksen vaikutusta signaalin laatuun. Punainen on alkuperäinen signaali, josta näytetään yksi arvo 512:sta ja musta 100 arvon keskiarvo alkuperäisestä. Y-akseli on jännite ja x-akseli on kuvaajan alkioiden määrä, joka on suoraan verrannollinen aikaan. Kuvaajasta näkyy muun muassa se, että esimerkiksi liukuvalle keskiarvolle tyypillistä viivettä nousuissa ja laskuissa ei ole. Alkuperäisenkään signaalin kohina ei ole tässä tapauksessa kovin suurta, y-akselin arvot vaihtelevat 20 - 30 millivoltin amplitudilla.



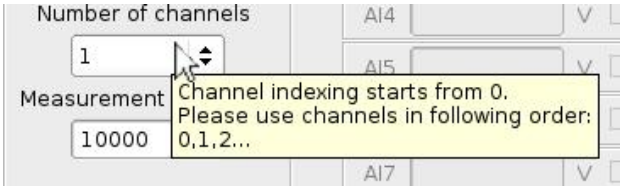
Kuva 11: Keskiarvotuksen vaikutus signaaliin

5.2 Ohjelman graafinen käyttöliittymä

Ohjelmaan tehtiin Qt Creator -ohjelmistolla graafinen käyttöliittymä, josta mittaustiedot näkee reaaliajassa ja josta pystyy säätämään mittauksen parametrit kohdalleen. Käyttöliittymä tuli tehdä mahdollisimman selkeäksi ja helpoksi käyttää unohtamatta toiminnallisuutta ja tarvittavia säätömahdollisuuksia. Käyttöliittymän tuli myös olla niin dynaaminen, että se soveltuisi mahdollisimman laajaan skaalaan erilaisia mittaustilanteita.

Käyttöliittymästä tuli nähdä käytössä olevien analogiatulokanavien jännitteet numeerisena ja niistä tuli piirtää graafiset kuvaajat, joista näkyy jännite ajan funktiona. Käyttöliittymään tuli myös lisätä säätömahdollisuudet kaikille niille muuttujille ja toiminnoille, joita pitäisi mahdollisesti muuttaa mittausten aikana tai niiden välissä, jotta tarve tehdä muutoksia lähdekoodiin poistuisi. Tämä mahdollistaa sen, että myös käyttäjä, joka ei osaa C++ -ohjelmointia tai joka ei tunne lähdekoodia, pystyy käyttämään ohjelmaa kaikissa tilanteissa. Lisäksi ohjelmasta tulee nopeampi ja mukavampi käyttää.

Käyttöliittymässä on melko paljon erilaisia elementtejä, joten sen käyttö ei välttämättä ole kokemattomalle henkilölle helppoa. Tämän takia useimpiin elementteihin lisättiin tooltip-avustustekstit (kuva 12), jotka tulevat esiin, kun hiiren osoitinta pidetään elementin yläpuolella. Tooltipissä neuvotaan, miten kyseinen elementti toimii ja mitä tulee ottaa huomioon sitä säädettäessä.



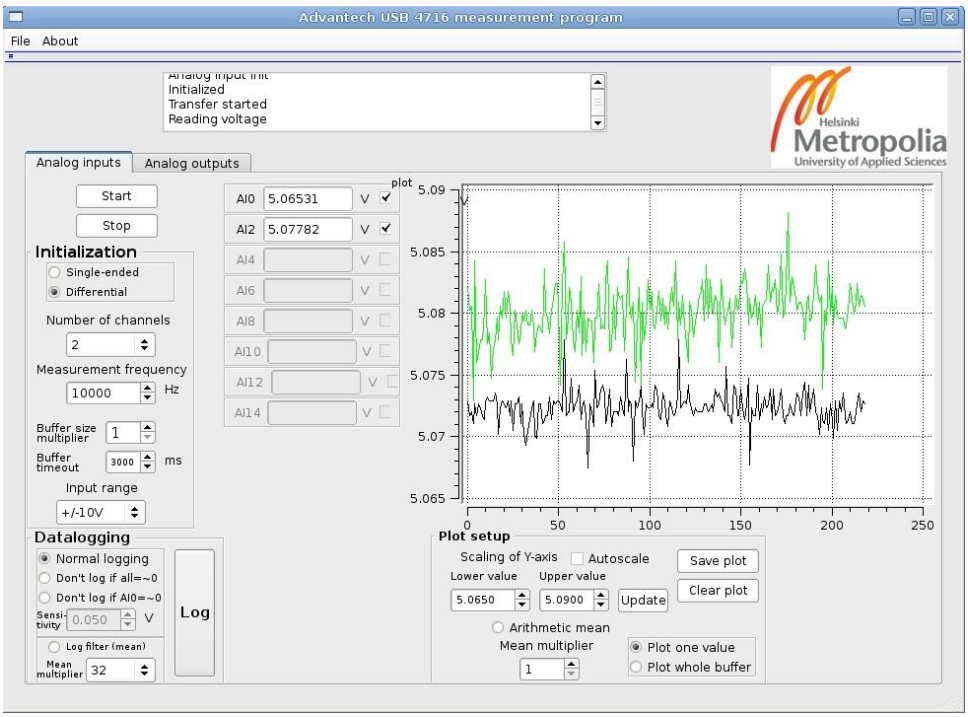
Kuva 12: Käyttöliittymän tooltip-avustusteksti

Kuvassa 13 näkyy mittausohjelman graafinen käyttöliittymä mittaustilanteessa, jossa selvitetään ympäristöstä tulevia häiriöitä. Kuvassa ylhäällä näkyvässä valkoiseen tekstilaatikkoon tulevat viestit ohjelmalta käyttäjälle, kuten häiriöilmoitukset tai tilannepäivitykset. Sen alapuolella olevasta välilehtivalikosta voi valita näkymän sen mukaan, mitä halutaan tehdä. Ohjelmaan tehtiin vain analogiatulot ja -lähdöt, mutta tähän voisi lisätä myös muun muassa digitaaliset tulot ja lähdöt.

Kuvan vasemmassa laidassa ovat initialisointiasetukset, jotka tulee säätää mittauksen mukaisiksi ennen mittauksen aloitusta. Kun tulojen lukeminen aloitetaan initialialisointiasetuseten yläpuolella olevasta Start-painikkeesta, valintaelementit muuttuvat harmaiksi eikä niitä voi enää muuttaa ennen kuin lukeminen on pysäytetty Stop-painikkeesta.

Initialisointitietojen alapuolella on mittaustietojen tallennuksen (datalogging) asetukset ja painike, josta tietojen tallennus aloitetaan. Kuten initialisoinnissa, asetukset on säädettävä ennen tallennuksen aloittamista.

Kuvaajan ja initialisointiasetusten välissä näkyy kunkin kanavan tarkka oloarvo voltteina. Kunkin arvon oikealla puolella olevasta checkbox-elementistä voi määrittää, piirretäänkö kyseisen kanavan tulotietoa myös kuvaajaan. Kanavia näytetään käyttöliittymässä kahdeksan tai 16 riippuen siitä, kumpi mittaustapa on valittu initialisointiasetuksissa. Lisäksi käyttämättömät kanavat näytetään harmaina. Kuvassa 13 käytössä on kaksi kanavaa differentiaalisella mittaustavalla, eli kaksi kahdeksasta elementtiryhmästä ovat aktiivisia, eli ei harmaita.



Kuva 13: Mittausohjelman graafinen käyttöliittymä

Kuvaajan alapuolella ovat asetukset kuvaajalle. Niistä voi säätää muun muassa y-akselin ylä- ja alarajat halutuille arvoille, mutta asetuksissa on myös automaattiskaalaus-vaihtoehto, joka rajaa kuvaajan niin, että kaikki mittausarvot näkyvät. Asetuksissa on myös mahdollisuus säätää aritmeettinen keskiarvoistus kuvaajan arvoille, joka on hyödyllinen toiminto esimerkiksi hyvin häiriöpitoisen signaalin suodattamiseen. Myös kuvaajan tallennus- ja tyhjennyspainike löytyvät asetusten yhteydestä.

Kuvaajan piirto

Mittausohjelman käyttöliittymän tärkeimpiä ominaisuuksia on graafisen kuvaajan piirto mittausarvoista. Tässä ohjelmassa piirto toimii Qwt-lisäosan (ks.4.3.1) plot-elementin avulla. Plot on melko monimutkainen mutta tehokas kokonaisuus. Kun se on otettu käyttöön ja alustettu, sillä voi piirtää kuvaajia reaaliajassa tai yhdellä kerralla päivittämällä kuvaajassa käytettävien lukujonojen arvot ja suorittamalla replot-komento. Tämä työ aloitettiin pisteestä, jossa oli käytettävissä sovellus, jossa plot-elementti ja joitain käyttöliittymän ominaisuuksia oli jo mukana, joten plotin opetteluun ja käyttöön ei tarvinnut käyttää paljoa aikaa.

Koska oletuksena oli, että mittaustaajuus olisi hyvin suuri, yli 10 000 arvoa sekunnissa, ohjelma asetettiin päivittämään kuvaajaan vain lukujonon ensimmäinen arvo. Tällöin puskurin oletuskoolla, joka on 512 arvoa, 511 jää näyttämättä, mutta jos mitattava ilmiö ei ole kovin nopeasti muuttuva, merkittävää määrää informaatiota ei kuitenkaan jää pois. Esimerkiksi taajuudella 10 000 arvoa sekunnissa kuvaajaa päivitetään lähes 20 kertaa sekunnissa. Jos näin nopealla mittaustaajuudella jokainen arvo tulostettaisiin, kuvaajasta tulisi melkein heti niin tiheä, että siitä ei saisi selvää. Lisäksi kuvaaja jouduttaisiin tyhjentämään jatkuvasti, sillä double-tyyppisiin mittaustiedon sisältäviin lukujonoihin ei mahdu kuin noin 65 000 arvoa.

Ohjelmaan tehtiin myös mahdollisuus näyttää kuvaajassa kaikki mittausarvot puskurista, mutta koska käyttökohteena olevassa mittauksessa ei tarvittu tätä ominaisuutta, sitä ei jatkettu kovin käyttökelpoiseksi. Huomattavasti hitaammissa mittauksissa siitä voisi kuitenkin olla hyötyä. Ohjelmaan voitaisiin myös tulevaisuudessa lisätä mahdollisuus säätää tulostettavien arvojen määrä. Tällöin tulisi kiinnittää huomiota siihen, että lukujono jaetaan tasan

tulostettavien arvojen kesken, jotta tulostettavan tiedon välit olisivat saman kokoisia ja kuvaaja pysyisi loogisesti jatkuvana.

Kuvaajan voi tallentaa 'Save plot' -painikkeesta. Tällöin avautuu dialogi-ikkuna, joka kysyy tallennustiedoston nimen ja tallennuspaikan kovalevyllä. Dialogi-ikkuna on samanlainen kuin kohdan 5.1.4 kuvassa 10.

6 KUORIMATERÄN MITTAUKSET

Tehtyä ohjelmaa sovellettiin Metropolia Ammattikorkeakoulun energiatekniikan laboratoriossa tehtyihin mittauksiin, joissa oli tarkoitus selvittää tukkien kuorimateriään kohdistuvan iskun voima ja terän venymä ajan funktiona. Mittauksissa simuloitiin kuorimapisteeseen tulevan tukin iskeytymistä terään.

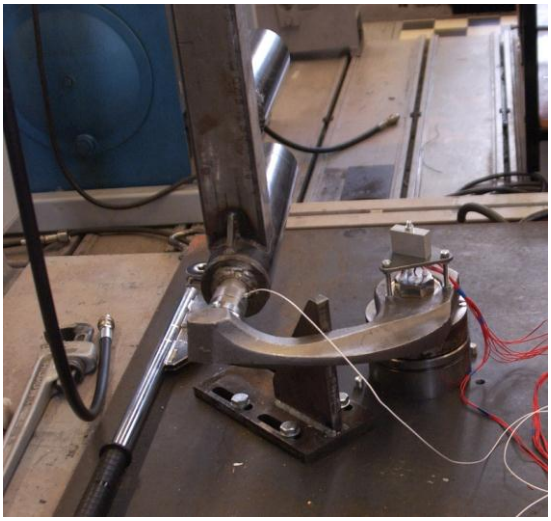
Tähän työhön kuului mittauksia varten sopivan ohjelman toimittaminen sekä avustaminen USB-moduulin ja ohjelman käytössä. Lisäksi tuli olla mukana osassa mittauksia ja niitä edeltäviä koe- ja kalibrointimittauksia, jotta pystyttäisiin kartoittamaan mittausten tarpeet. Itse mittaukset ja niiden järjestely olivat toisen opiskelijan insinöörityö.

Tässä osiossa selostetaan mittausten toteutus ja tavoitteet sekä pohditaan mittausohjelman ja -moduulin toimintaa mittauksissa.

6.1 Mittausten toteutus

Lopullisissa mittauksissa kuorimateriää oli tarkoitus iskeä katosta roikkuvalla heilurilla, jonka osumakohdassa oli voima-anturi ja itse kuorimaterässä kahdeksan eri kohtiin sijoitettua venymäliuska-anturia. Voima-anturi oli vahvistimen kautta yhdistetty USB-moduuliin ja venymäliuskat suoraan Wheatstonen siltojen kautta.

Kuvassa 14 näkyy heiluri ja kuorimaterä ennen mittauksia. Valkoinen johto menee voima-anturiin ja punaiset johdot aktiivisille venymäliuskoille. Kuorimaterä on kiinni lattiaan kiinnitetyssä akselirungossa. Rungon päällä on metallilevy, josta heijastetaan lasersäde valkoiselle taululle, minkä perusteella nähdään, kuinka paljon terä liikkui iskun vaikutuksesta.



Kuva 14: Heiluri ja kuorimaterä

Kuvassa 15 on pöytä, jonka päällä mittauksissa käytetty laitteisto ja mittaustietokone olivat. Oikealla on tietokone, jossa mittausohjelma toimii. Tietokoneen vasemmalla puolella on voima-anturin vahvistin. Keskellä on virtalähde, joka on säädetty syöttämään tasan 10 voltin jännitettä. Virtalähteen vasemmalla puolella on itse DAQ-moduuli, johon on kytketty seitsemän venymäliuskaa ja voima-anturi. Kuvan vasemmassa laidassa ovat terästankoon kiinnitetyt passiiviset venymäliuska-anturit, jotka tarvitaan Wheatstonen siltojen muodostamiseen.

Kuva 15: Mittauslaitteisto

Venymäliuskamittauksissa käytetään yleensä venymäliuskavahvistinta, joka vahvistaa tulojännitteen millivolteista volttien suuruusluokkaan ja suodattaa signaalia, tehden siitä huomattavasti selkeämmän ja helpommin luettavan. Tässä mittauksessa otettiin kuitenkin tavoitteeksi mitata ilman vahvistimia,

koska tulosten ei tarvinnut olla täysin tarkkoja ja koska vahvistimet ovat melko kalliita. Luotettiin myös siihen, että DAQ-modulin 16 bitin resoluutio olisi tarpeeksi hyvä, ja siihen, että tarvittaessa voitaisiin käyttää ohjelmallista suodatusta tai käsitellä mittaustuloksia jälkeenpäin taulukkolaskentaohjelmalla.

Alustavia koemittauksia tehtäessä huomattiin, että single-ended-mittaustapa (ks. kappale 3.3) ei sovellu mittaukseen, sillä eri kanavien mittaustulosten havaittiin vaikuttavan toisiinsa. Tämä johtuu yhteisestä nollajännitteestä, joka on single-ended-mittaustavassa yhdistetty maahan. Pääteltiin, että mittaukset on tehtävä differentiaalisella mittaustavalla, jossa nollajännitteet ovat omassa kanavassaan.

Ongelmaksi muodostui se, että differentiaalisesti mitattaessa USB-4716-modulissa on käytössä vain kahdeksan kanavaa, koska yksi tulo käyttää kahta kanavaa, mutta mittauksessa tarvittiin yhdeksän. Ongelma kierrettiin aluksi käyttämällä kahta samanlaista DAQ-moduulia, jolloin tarvittiin myös toista tietokonetta. Näin ollen pöytätietokoneelle asennettiin samat komponentit, jotka olivat käytössä kannettavassa tietokoneessa: Debian 4.0 -käyttöjärjestelmä, Qt, Qwt ja USB-4716:n laiteajuri. Kahden tietokoneen mittausten synkronointiin käytettiin liipaisutoimintoa, jonka avulla tiedon tallennuksen voi aloittaa, kun kanavan AI0, jossa voima-anturi tässä tapauksessa on, jännite ylittää tietyn raja-arvon. Molempiin DAQ-moduuleihin kytkettiin viisi tuloa: voima-anturi ja neljä venymäliuskaa.

Lopulta tultiin kuitenkin siihen tulokseen, että yhdestä venymäliuskasta voidaan luopua. Näin saatiin toinen DAQ-moduuli ja kannettava tietokone muuhun käyttöön, ja mittausjärjestelyt yksinkertaistuivat.

Kannettavan tietokoneen poistuttua mittauksista havaittiin, että mitattavien signaalien häiriö väheni hieman. Tämän pääteltiin johtuvan kannettavan tietokoneen aiheuttaman magneettikentän indusoitumisesta mittausjärjestelmän johdotuksiin. Päätelmää tuki se, että kannettavan tietokoneen kotelointi on muovia, kun taas pöytätietokoneen kotelointi on peltilevyä. Myös kannettavan tietokoneen irrallinen muuntaja saattoi olla ainakin osasyy.

6.2 Mittausten tavoitteet

Lopullisena tavoitteena oli saada selville impulssi, jolla terä liikkuu lepoasemastaan. Impulssi on voiman (F) ja sen vaikutusajan (Δt) tulo, joka saadaan mittaustuloksista graafisesti integroimalla, eli laskemalla pinta-ala, joka jää voima- ja aika-akselin väliin iskun ajalla.

Venymäliuskojen tulojännitteestä oli tarkoitus nähdä kuorimaterän kiristysmomentti eri kohdissa mittausjärjestelmää. Tärkein näistä oli kuorimaterän akseliinsa kiinnittävän ruuvin kiristys, sillä siitä havaitaan, avautuuko ruuvin kiinnitys iskettäessä. Kaksi liuskaa muodostivat toisilleen parin, joiden tulojännitteen erotuksesta nähdään jännitysten ero eri puolilla ruuvia. Venymäliuskat olivat tärkeitä mittausjärjestelmän kalibroinnissa ja kohdalleen kiristämisessä, mutta myös itse mittauksissa, joissa niiden tulojännitteen kuvaajista nähtäisiin jännityksen muutokset iskujen välillä.

6.3 Ohjelman ja laitteen soveltuvuus mittauksiin

Advantech USB-4716 sopi mittauksiin tarkkuudeltaan ja nopeudeltaan hyvin. Ongelmia toisaalta aiheutti se, että laitteessa ei ole sisäistä kelloa, joka ilmoittaisi tarkan ajankohdan, jolloin näyte on otettu. Ajan ilmoittaminen pystyttiin tekemään laskemalla aika lukutaajuuden perusteella, mutta laskettu aika jäi kuitenkin jonkin verran jälkeen oikeasta ajasta.

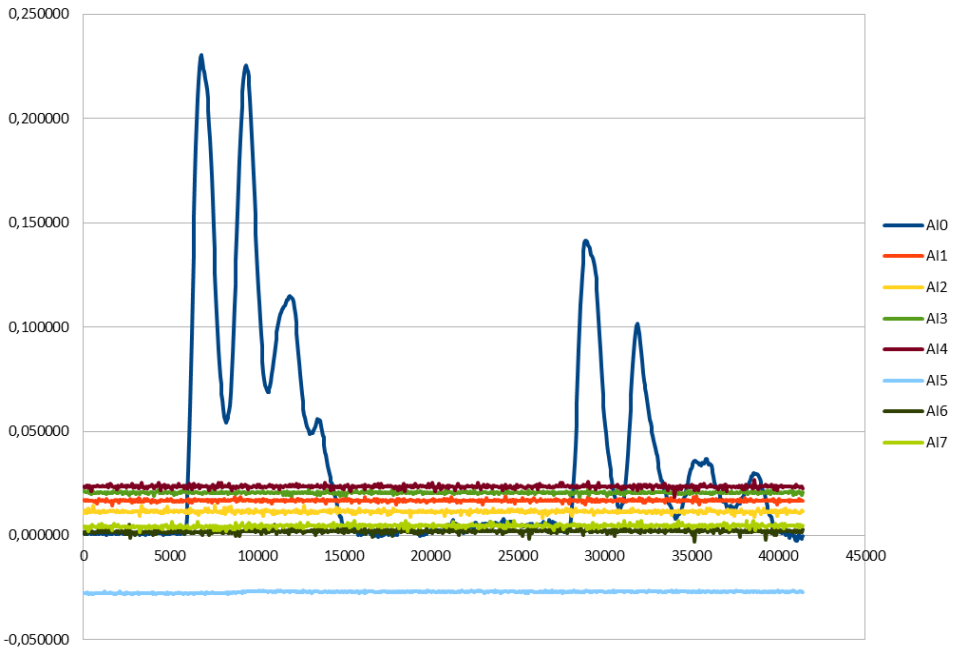
Ohjelma saatiin toimimaan kutakuinkin siten, kuin sen mittauksissa pitikin toimia. Mittaaja näki säätö- ja kalibrointivaiheessa tarvittavat tiedot ohjelman piirtämästä kuvaajasta. Hän myös sai varsinaisissa mittauksissa tarvittavat tarkat lukuarvot helposti ja nopeasti taulukkolaskentaohjelmaan. Säätömahdollisuudet saatiin tarpeeksi monipuolisiksi, mutta käyttöliittymä pysyi kuitenkin melko yksinkertaisena käyttää. Ohjelman käytössä piti aluksi opastaa hieman, mutta hyvin nopeasti mittaaja pystyi käyttämään ohjelmaa itsenäisesti ilman ulkoista apua.

Suurin ongelma ohjelman osalta oli se, että liipaisutoiminto, jolla saadaan tallennettua vain tarpeellinen tieto, ei aina toiminut oikein. Loppuvaiheessa toiminto ei aloittanut mittausta ajoissa tai ollenkaan noin kerran kahdeksasta mittauksesta.

Kuvassa 16 on kuvaaja, joka on tehty taulukkolaskentaohjelmalla mittausohjelman tallentamista arvoista yhdessä monista kuorimaterän iskukokeista. Y-akseli on jännite voltteina ja x-akseli aika mikrosekunteina (10^{-6} s). Tummansininen käyrä on voima-anturi ja muut ovat venymäliuska-antureita. Tässä mittauksessa tarkasteltiin lähinnä voima-anturin jännitettä ajan suhteen, mutta venymäliuska-anturit olivat mukana, jotta nähtäisiin, muuttuuko akselipultin jännitys, toisin sanoen aukeaako se yhtään iskun vaikutuksesta.

Ensimmäiset nousut voima-anturin ulostulojännitteessä ovat itse isku. Se, että kuvaajaan muodostuu useampi piikki, johtuu heilurin iskukohdan materiaalin, tässä tapauksessa puun, joustavuudesta. Toiset nousut johtuvat heijastusaallosta, joka palautuu kuorimaterästä heiluriin.

Kuvaajasta näkyy, että mittauksessa käytetty liipaisutoiminto on toiminut oikein. Vain tarpeellinen tieto on siis tullut mukaan kuvaajaan. Venymäliuskoissa näkyy pientä kohinaa, joka on kuitenkin tarpeeksi vähäistä, jotta pienetkin muutokset havaitaan.



Kuva 16: Iskumittaus tallennettuna ohjelmalla

7 YHTEENVETO

Insinöörityön tavoitteina olivat Advantech USB-4716 -tiedonkeruumoduulin käyttöönotto, tiedonkeruun periaatteisiin perehtyminen ja Qt-ohjelmointiympäristöä hyödyntäen monikäyttöisen mittausohjelman ohjelmointi moduulin käyttöympäristöksi. Pohjana ohjelmalle oli puolivalmis runko, jossa osa toiminnallisuudesta oli valmiina. Tavoitteena oli myös tehdä työ kustannuksiltaan mahdollisimman edullisella tavalla säilyttäen kuitenkin hyvä tarkkuus ja suorituskyky mittauksissa.

Mittausohjelmaa tuli soveltaa tukkien kuorimaterän iskumittauksiin, joissa oli tarkoituksena selvittää impulssi, jolla kuorimaterä liikkuu asemastaan, ja jännitykset, jotka terään tällöin kohdistuvat. Mittauksissa käytettiin yhtä voima-anturia ja kahdeksaa venymäliuska-anturia, joista karsittiin myöhemmin yksi pois. Mittaajaa tuli myös opastaa ohjelman käytössä. Ohjelmaa tuli kehittää sen mukaan, mitä mittauksissa tarvittiin.

Työn lopputuloksena syntynyt ohjelma on toimiva ja dynaaminen ja soveltuu moniin mittaustilanteisiin, joissa tarvitaan nopeaa analogiatulojen lukemista. Ohjelmassa on käytetty suurinta osaa moduulin ohjelmallisista säätömahdollisuuksista, ja ne ovat helposti asetettavissa mittauksen tarpeiden mukaisiksi graafisesta käyttöliittymästä. Ohjelma piirtää mitatuista signaalista kuvaa reaaliajassa. Lisäksi sillä voi tallentaa tulokset ja kuvaajan myöhempää käyttöä varten. Tulokset tallentuvat tekstitiedostona, josta ne on helppo avata taulukkolaskentaohjelmalla.

Ohjelma toimii hyvänä pohjana kaikkeen tulevaisuudessa tehtävään työskentelyyn koulun hankkimilla kahdella USB-4716-moduulilla. Perustason tiedonkeruuseen ja mittaukseen ohjelma kelpaa sellaisenaan. Esimerkiksi toimilaitteiden ohjaamiskäyttöön ohjelmaan olisi tehtävä pieniä muutoksia. Pienellä ohjelmallisella muutoksella moduuli on käytettävissä myös muilla Advantech USB -sarjan laitteilla sekä luultavasti melko helposti myös PCI- ja PCM-sarjan laitteilla. Nämä laitteet käyttävät pääsääntöisesti keskenään samoja funktioita.

Kuorimaterän mittausprojekti toimi hyvänä käytännön kokeena ohjelman toiminnalle. Ohjelmaan tehtiin useita lisäyksiä ja korjauksia mittausten tarpeiden mukaan. Suurin lisäys projektia varten oli liipaisutoiminto tulosten tallentamisen aloittamiseksi ja lopettamiseksi raja-arvon ylityttyä tai alituttua. Tämän kanssa oli myös ongelmia, koska aluksi kävi usein niin, että liipaisu toimi jäljessä tai tietoa ei tallentunut lainkaan lyhyissä mittauksissa. Ongelmia mittauksissa tuotti myös se, että tiedonkeruulaitteessa ei ollut sisäistä kelloa, jonka perusteella näytteiden lukuajankohta olisi nähty tarkkaan. Aika, joka laskettiin ohjelmallisesti näytteistystaajuuden perusteella, jäi jonkin verran jälkeen todellisesta ajasta.

Merkittävä havainto tiedonkeruun osalta oli, että mitattaessa venymäliuskaantureiden jännitteitä, joilta näissä mittauksissa edellytettiin kohtuullista tarkkuutta, ei tarvittu lainkaan venymäliuskavahvistimia, joita on suositeltu käytettävän tällaisissa mittauksissa. Tiedonkeruumoduulin hyvä tarkkuus, ympäristön aiheuttaman häiriön minimoiminen muun muassa kierrekaapeleita sekä hyvää virtalähdettä käyttämällä, ja jopa kannettavan tietokoneen korvaaminen pöytätietokoneella näkyivät melko tasaisena signaalina. Vahvistimien pois jättäminen oli suuri säästö mittalaitteiden hankintakustannuksissa.

Insinöörityön tulokset täyttivät asetetut tavoitteet hyvin. Mittausohjelmasta tehtiin joustava ja monipuolinen, tosin pieniä puutteita löytyi vielä työn loppuvaiheilla. Työn järjestelyt saatiin tehtyä hyvin pienin kustannuksin, kuten toivottiinkin. Tähän päästiin käyttämällä avoimeen lähdekoodiin perustuvia ohjelmistoja ja edullisia laitteita sekä minimoimalla mittauksissa käytettävät komponentit. Osa komponenteista korvattiin ohjelman toiminnoilla, kuten keskiarvon laskennalla. Lisäksi käytettiin tarkkaa mutta kuitenkin edullista mittausmoduulia.

Loppuun todettakoon, että Advantech USB-4716 on hinta-laatusuhteeltaan hyvä DAQ-moduuli yksinkertaisempiin tiedonkeruusovellutuksiin. Se on tarkka ja nopea. Moduulin ohjelmoinnissa halutunlaiselle mittaukselle on kuitenkin melko paljon työtä. Suurin heikkous on sisäisen kellon puuttuminen. Tämä puute rajaa laitteen pois mittauksista, joissa tarvitaan tarkkaa aikaakselia.

VIITELUETTELO

[1] Introduction to Data Acquisition. 2011. Verkkodokumentti National Instruments Corporation. <http://zone.ni.com/devzone/cda/tut/p/id/3536>. Luettu 10.3.2011.

[2] Huttunen, Heikki. 2005. Signaalinkäsittelyn menetelmät. Verkkodokumentti. Tampereen teknillinen yliopisto. <http://www.cs.tut.fi/kurssit/SNG-1200/Moniste.pdf> Luettu 11.3.2011.

[3] Signal Filtering. 2009. Verkkodokumentti. ADInstruments. <http://www.adinstruments.com/solutions/attachments/pr-signalfilters-05a.pdf>. Luettu 11.3.2011.

[4] Keithley. 2001. Data Acquisition and Control Handbook. 1st ed. Keithley Instruments Inc.

[5] Data Acquisition Systems (DAQ) and Equipment. 2011. Verkkodokumentti. <http://www.data-acquisition.us/>. Luettu 11.3.2011.

[6] Differential Signaling. 2001. Verkkodokumentti. Lattice Semiconductor Corporation. <http://www.latticesemi.com/lit/docs/appnotes/pac/an6019.pdf>. Luettu 11.3.2011.

[7] Omega Engineering Inc. 1998. Transactions volume 2: Data Acquisition. Verkkokirja. <http://www.omega.com/literature/transactions/Transactions_Vol_II.pdf>. Luettu 11.3.2011.

[8] Olshausen, Bruno. 2000. Aliasing. Verkkodokumentti. <redwood.berkeley.edu/bruno/npb261/aliasing.pdf>. Luettu 11.3.2011.

[9] Nyquist and Shannon's Sampling Theorems. 2009. Verkkodokumentti. National Instruments Inc. <http://zone.ni.com/reference/en-XX/help/370524M-01/siggenhelp/fund_nyquist_and_shannon_theorems/>. Luettu 11.3.2011.

[10] Advantech. 2006. Advantech Device Driver User Manual. Html-dokumentti.

[11] Advantech. 2006. USB-4716 User Manual. Saatavilla <support.elmark.com.pl/advantech/pdf/iag/USB-4716-manual.pdf>. Luettu 11.3.2011.

[12] Qt 4.6 Whitepaper. 2009. Verkkodokumentti. Nokia Oyj. <qt.nokia.com/files/pdf/qt-4.6-whitepaper>. Luettu 11.3.2011.

[13] Signals and Slots. 2010. Verkkodokumentti. Nokia Oyj. <http://doc.qt.nokia.com/4.6/signalsandslots.html>. Luettu 11.3.2011.

Laitteen käyttöönotto-ohjeet

Laadittu 7.3.2011. Kaikkiin linkkeihin viitattu kyseisenä päivämääränä.

Nämä ohjeet on laadittu Advantech USB-4716 -tiedonkeruumoduulin käyttöön ja ohjelmointiin Qt-ohjelmointityökalulla. Ohjeet on tarkoitettu englanninkieliselle, 32-bittiselle Debian 4.0 -käyttöjärjestelmälle. Ohjeissa ei käsitellä käyttöjärjestelmän asentamista. Käyttöjärjestelmän asennustiedostot ja -ohjeet löytyvät osoitteesta http://www.debian.org/. Laite toimii myös muissa käyttöjärjestelmissä. Lisätietoja osoitteessa http://www.advantech.com/products/USB-4716/mod_A3AB933C-C6D3-49EB-9D25-58CACECDEF7A.aspx

1. Lataa ja asenna Qt

- Uusin versio on saatavilla osoitteesta

  http://qt.nokia.com/downloads.

2. Lataa ja asenna Qwt-lisäosa

- Tämä onnistuu helpoiten Synaptic-pakettienhallintaohjelmalla, joka on Debianissa valmiina. Ohjelma asentaa automaattisesti Qwt:n Qt:n yhteyteen.

3. Lataa ja asenna ajuripaketti

- Paketin saa ladattua osoitteesta

  http://downloadt.advantech.com/download/downloadsr.aspx?File_Id=1-FHP60V

- Avaa komentopääte (terminal). Mene kansioon johon edellämainittu paketti on ladattu. Hakemistoja voi selata komennoilla “cd [kansion nimi]” ja “cd ..”. Komennolla “ls” näkee hakemiston sisällön.

- Suorita komento: su

  Tämä antaa järjestelmänvalvojan oikeudet.

- Suorita komento: dpkg -i advdaq_1.07.0001-1_i386.deb

  Tämä purkaa ja asentaa paketin.

- Mene kansioon /usr/src/Advatech/DAQ.../
- Avaa tiedosto “adv_load_template” tekstinkäsittelyohjelmalla, esim. Gedit:llä. Komentopäätteessä: gedit adv_load_template
- Poista kommenttimerkki laitteen USB-4716 kohdalta ja talleta tiedosto.

4. Alusta laitteen ajuri

- Tämä on tehtävä myös aina, kun tietokone on käynnistetty uudestaan.
- Avaa komentopääte ja suorita komento: su
- Mene kansioon /usr/src/Advatech/DAQ.../
- Suorita komentopäätteessä: ./adv_load_template
- Suorita komentopäätteessä: advdevice_bind usb4716 0 /dev/advdaq0

  Tällöin laitteen on oltava kiinni tietokoneen USB-portissa.
- Anna kansiolle /dev oikeudet lukea ja kirjoittaa. Tämä onnistuu esimerkiksi seuraavasti:
- Suorita komentopäätteessä: nautilus
- Valitse file → open parent
- Etsi kansio /dev, paina hiiren oikealla näppäimellä ja avaa “properties”.
- Välilehdestä “Permissions”, valitse “File Access” -kohtiin “Read and Write” ja paina “Apply permissions”.

Tämän jälkeen Qt:lla tehty ohjelma saa yhteyden laitteeseen. Tarkempia tietoja ja lista laitteen käyttöön liittyvistä funktioista löytyy laitteajurin ohjeista, jotka asentuvat ajureita purkaessa ja asentaessa hakemistoon /usr/src/Advantech/DAQ-1.07.0001/docs.

Huom. Jotta tekemäsi Qt-projektit toimisivat Qwt:n ja laiteajurin kanssa seuraavat rivit .pro-tiedostoon:

```
INCLUDEPATH += /usr/local/include/Advantech/ \
               /usr/include/qwt-qt4

LIBS += -L/usr/lib -ladvdaq \
        -L/usr/lib -lqwt-qt4
```

OHJELMAN LÄHDEKOODI

Lähdekoodi on tehty Qt-ohjelmointiympäristöllä Debian 4.0 käyttöjärjestelmässä. Koodin pitäisi toimia sellaisenaan Linux-käyttöjärjestelmissä. Käyttöliittymän form-tiedostoa, mainwindow.ui, ei ole mukana.

Projektitiedosto, AdvMittausThread.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2010-09-30T12:22:12
#
#-------------------------------------------------


QT       += core gui


TARGET = AdvMittausThread
TEMPLATE = app


INCLUDEPATH += /usr/local/include/Advantech/ \
               /usr/include/qwt-qt4


LIBS += -L/usr/lib -ladvdaq \
        -L/usr/lib -lqwt-qt4


SOURCES += main.cpp\
        mainwindow.cpp \
    measurementthread.cpp \
    controlthread.cpp


HEADERS  += mainwindow.h \
    measurementthread.h \
    controlthread.h


FORMS    += mainwindow.ui
```

Pääohjelma, main.cpp

```
#include <QtGui/QApplication>
#include "mainwindow.h"


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();


    return a.exec();
}
```

Käyttöliittymän header-tiedosto, mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H


#include <QMainWindow>
#include <qmessagebox.h>
#include <QFileDialog>
#include <QtSvg/QSvgGenerator>
#include <QPainter>
#include <QColor>
#include <Advantech/advdevice.h>
#include <Advantech/advdaq.h>
#include "measurementthread.h"
#include "controlthread.h"
#include <signal.h>
```

```
#include <qdatetime.h>


#include <qwt-qt4/qwt_legend.h>
#include <qwt-qt4/qwt_scale_draw.h>
#include <qwt-qt4/qwt_math.h>
#include <qwt-qt4/qwt_plot_item.h>
#include <qwt-qt4/qwt_plot_curve.h>
#include <qwt-qt4/qwt_text.h>
#include <qwt-qt4/qwt_plot_grid.h>
#include <qwt-qt4/qwt_plot_printfilter.h>


namespace Ui {
    class MainWindow;
}


class MainWindow : public QMainWindow
{
    Q_OBJECT


public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    int StartFlag;
    bool autoscale;
    double lowery;
    double uppery;
    QString filename;
    QFile file;
    QTextStream out;


private:
    Ui::MainWindow *ui;


    QString AnalogValue;
    QString Error;
    PTR_T fd;
    INT32S ret;


    MeasurementThread mthread;  //Threadin olio
    controlthread cthread;


    //Plotin muuttujat
    QwtPlot *AnalogInput;
    QwtPlotCurve *AnalogInput0;
    QwtPlotCurve *AnalogInput1;
    QwtPlotCurve *AnalogInput2;
    QwtPlotCurve *AnalogInput3;
    QwtPlotCurve *AnalogInput4;
    QwtPlotCurve *AnalogInput5;
    QwtPlotCurve *AnalogInput6;
    QwtPlotCurve *AnalogInput7;
    QwtPlotCurve *AnalogInput8;
    QwtPlotCurve *AnalogInput9;
    QwtPlotCurve *AnalogInput10;
    QwtPlotCurve *AnalogInput11;
    QwtPlotCurve *AnalogInput12;
    QwtPlotCurve *AnalogInput13;
    QwtPlotCurve *AnalogInput14;
    QwtPlotCurve *AnalogInput15;
    QwtPlotGrid *AnalogInput1Grid;


    int PlotAika;
    double x0[10000],y0[10000];
    double x1[10000],y1[10000];
    double x2[10000],y2[10000];
    double x3[10000],y3[10000];
    double x4[10000],y4[10000];
    double x5[10000],y5[10000];
    double x6[10000],y6[10000];
    double x7[10000],y7[10000];
    double x8[10000],y8[10000];
    double x9[10000],y9[10000];
    double x10[10000],y10[10000];
    double x11[10000],y11[10000];
    double x12[10000],y12[10000];
    double x13[10000],y13[10000];
    double x14[10000],y14[10000];
    double x15[10000],y15[10000];


    int ydemo;
    int p;
    int buffersize;
    int AmountChannels;
    int SAMPLE_RATE;
    bool checked;
    unsigned long MittausAika;
    int num_chan;
    bool PlotCheck;
    bool DiffOn;
```

```
    int BufValues;
    QStringList Kanavat;
    QStringList Ranges;
    QStringList Means;
    QTime raw_time;
    QString time;
    QString timeandtext;


private slots:
    void GUIStartMeasurement();
    void GUIStopMeasurement();
    void GUIExit();
    void GUIAbout();
    void InitPlot();
    void exportSVG();
    void Scaling(bool);
    void GUIUpdatePlotData(float *,int,int,int);
    void GUIStartControlling();
    void GUIStopControlling();
    void GUIPrintToMonitor(const QString&);
    void GUIPrintToMonitorPlain(const QString &);
    void ClearPlot();
    void GetFileName();
    void PlotAll(bool);
    void LcdValue(float);
    void ShowHideSensitivity(bool);
    void ShowHideUnusedChans(int);
    void ShowHideLogging(bool);
    void DifferentialMeasurement(bool);
    void BufferValuesDisplay(int);


signals:
    void Filename(QString&);
};


#endif // MAINWINDOW_H
```

Mittausluokan header-tiedosto, measurementthread.h

```
#ifndef MEASUREMENTTHREAD_H

#define MEASUREMENTTHREAD_H


#include <QObject>
#include <QThread>
#include <Advantech/advdevice.h>
#include <Advantech/advdaq.h>
#include <signal.h>
#include <QFile>
#include <QFileDialog>
#include <QTextStream>
#include <qdatetime.h>


#define FIFO_SIZE 1024 // Puskurin koko, laitteen ominainen vakio
#define DATA_TYPE 1    // 0-BinaryIn, 1-VoltageIn


class MeasurementThread : public QThread
{
    Q_OBJECT


public:
    MeasurementThread();
    void run();
    void cancel();
    int RequestChannels();
    QString getTime();


signals:
    void initanaloginput();
    void measure();
    void updateplot_and_data();
    void pause();
    QString FilenameRequest();


    void toMonitor(const QString&);
    void updatedata(float *voltage_array, int multiplier, int num_chan, int SAMPLE_RATE);


private:
    int State;
    int InitFlag;


    QString AnalogValue;
    QString Error;
```

```
    PTR_T fd;
    PTR_T fd2;
    INT32S ret;


    //AI muuttujia ja olioita
    PT_FAIIntScanStart fai_int_scan_start;
    PT_EnableEvent enable_event;
    PT_CheckEvent check_event;
    PT_FAITransfer fai_transfer;
    PT_FAICheck   fai_check;
    unsigned int buffer2;
    int length;
    unsigned short *reading_array;
    unsigned short overrun;
    float *voltage_array;
    float *mean_array;


    void *bufptr_inp;   // buffer pointer for input
    void *bufptr_out;   // buffer pointer for output
    int pacer_rate;
    char err_msg[100];
    int fai_running;
    int i;
    int stop_flag;
    bool bufrdy;
    unsigned int CONVERTION_NUMBER;
    bool checked;
    int SAMPLE_RATE;
    int multiplier;
    double uppery;
    double lowery;
    bool flag;
    unsigned long MittausAika;
    unsigned long lenght;
    int KeskiarvoLuku;
    bool KeskiarvoON;
    int Timeout;
    bool SLog;
    bool SLogAI0;
    bool SLogFlag;
    bool LogMean;
    int LogMeanMultiplier;
    double Sensitivity;
    bool DiffOn;
    bool ShowClock;
    float Memory[1024];
    bool memflag;
    int BufValues;
    int DividedValue;
    int i1, i2;
    QString time;
    QTime raw_time;
    QString ProcessorClock;
    float Apuri0,Apuri1,Apuri2,Apuri3,Apuri4,Apuri5,Apuri6,Apuri7,Apuri8,Apuri9,
          Apuri10,Apuri11,Apuri12,Apuri13,Apuri14,Apuri15;
    float KA0[512],KA1[512],KA2[512],KA3[512],KA4[512],KA5[512],KA6[512],KA7[512],
    KA8[512],KA9[512],KA10[512],KA11[512],KA12[512],KA13[512],KA14[512],KA15[512];


    double KA1Vektori[10000];
    int KA1Indeksi;
    double KA1Summa;


    QString filename;
    QFile file;
    QTextStream out;


    unsigned short start_chan; //Ensimmäinen luettava kanava
    unsigned short num_chan;   //Luettavien kanavien lkm
    unsigned short gain_code;
    unsigned short *gain_array;
    //------------------------------//


public slots:
    void stopmeasurement();
    void Measure();
    void InitAnalogInputs();
    void release_region_func
         (int fd, PT_EnableEvent *event, unsigned short *buffer0, unsigned short *buffer1, float *buffer2);
    void InitDevice(char *);
    void amountchannels(int);
    void LogCheck(bool);
    void Frequency(int);
    void Multiplier(int);
    void Datalog();
    void MeanOn(bool);
    void MeanMultiply(int);
    void GetFilename(QString&);
    void BufTimeout(int);
    void InputRange(QString);
    void SmartLog(bool);
    void DatalogSensitivity(double);
    void DatalogMean(bool);
    void DatalogMeanSelector(QString);
```

```
    void DifferentialMeasurement(bool);
    void SmartLogAI0(bool);
    void BufferValuesDisplayed(int);
    void ShowProcessorClock(bool);
};


#endif // MEASUREMENTTHREAD_H
```

Analogialähtöjen header-tiedosto, controlthread.h

```
#include <QThread>
#include <Advantech/advdevice.h>
#include <Advantech/advdaq.h>


class controlthread : public QThread
{
    Q_OBJECT


public:
    controlthread();
    void run();


private:
    PT_AOConfig AOConfig;
    PT_AOVoltageOut AOVoltageOut;
    PT_AOBinaryOut AOBinaryOut;


    FP32 VoltageArray[16];
    FP32 aoutOutputValue;


    QString AnalogValue;
    QString Error;
    PTR_T fd;
    INT32S ret;
    bool controlflag;


    float SliderValue;


signals:
    void toMonitor(const QString &);
    void toMonitorPlain(const QString &);
    void toValue(float);


public slots:
    void InitAnalogOutputs();
    void ControlValve();
    void UpdateSliderValue(int);
    void InitDevice(char *);
    void stop();
};


#endif // CONTROLTHREAD_H
```

Käyttöliittymän ohjelmatiedosto, mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->InitPlot();


    StartFlag = 0;
    autoscale = 0;
    lowery = -10.0;
    uppery = 10.0;
    MittausAika = 0;
    PlotCheck = 0;
    DiffOn = 0;
    BufValues = 1;


    ui->MetropoliaLabel->setPixmap(QPixmap("Metropolia_logo.jpg"));


    Kanavat <<"1"<<"2"<<"3"<<"4"<<"5"<<"6"<<"7"<<"8"<<"9"<<"10"<<"11"<<"12"<<"13"<<"14"<<"15"<<"16";
```

```
    ui->KanavienLkm->addItems(Kanavat);


    Ranges <<"+/-10V"<<"+/-5V"<<"+/-2.5 V"<<"+/-1.25 V"<<"+/-0.625V"<<"0 to 10V "<<"0 to 5V"<<"0 to 2.5V"<<"0 to 1.25V";
    ui->RangeSelector->addItems(Ranges);


    Means <<"4"<<"8"<<"16"<<"32"<<"64"<<"128"<<"256"<<"512";
    ui->FilterSelector->addItems(Means);
    ui->FilterSelector->setCurrentIndex(3);


    //GUI-puolen yhdistamiset
    connect(ui->Start,SIGNAL(clicked()),this,SLOT(GUIStartMeasurement()));
    connect(ui->Stop,SIGNAL(clicked()),this,SLOT(GUIStopMeasurement()));
    connect(ui->actionExit,SIGNAL(triggered()),this,SLOT(GUIExit()));
    connect(ui->actionAbout,SIGNAL(triggered()),this,SLOT(GUIAbout()));
    connect(ui->exportSVG,SIGNAL(clicked()),this,SLOT(exportSVG()));
    connect(ui->Stop,SIGNAL(clicked()),&mthread,SLOT(stopmeasurement()));
    connect(ui->EnableAO,SIGNAL(clicked()),this,SLOT(GUIStartControlling()));
    connect(ui->DisableAO,SIGNAL(clicked()),this,SLOT(GUIStopControlling()));
    connect(ui->DisableAO,SIGNAL(clicked()),&cthread,SLOT(stop()));
    connect(&cthread,SIGNAL(toValue(float)),this,SLOT(LcdValue(float)));
    connect(ui->Log_check,SIGNAL(clicked(bool)),this,SLOT(OpenLog(bool)));
    connect(ui->ClearPlot,SIGNAL(clicked()),this,SLOT(ClearPlot()));
    connect(ui->PlotAll,SIGNAL(toggled(bool)),this,SLOT(PlotAll(bool)));
    connect(ui->BufValues,SIGNAL(valueChanged(int)),this,SLOT(BufferValuesDisplay(int)));


    connect(ui->Autoscale,SIGNAL(toggled(bool)),this,SLOT(Scaling(bool)));
    connect(ui->UpdateButton,SIGNAL(clicked(bool)),this,SLOT(Scaling(bool)));
    connect(ui->NormalLog,SIGNAL(toggled(bool)),this,SLOT(ShowHideSensitivity(bool)));
    connect(ui->KanavienLkm,SIGNAL(currentIndexChanged(int)),this,SLOT(ShowHideUnusedChans(int)));
    connect(ui->Log_check,SIGNAL(toggled(bool)),this,SLOT(ShowHideLogging(bool)));
    connect(ui->RButton_Differential,SIGNAL(toggled(bool)),this,SLOT(DifferentialMeasurement(bool)));


    //Threadilta tulevat signaalit
    connect(&mthread,SIGNAL(initdevice(char *)),this,SLOT(GUIInitDevice(char *)));
    connect(&mthread,SIGNAL(measure()),this,SLOT(GUIMeasure()));
    connect(&mthread,SIGNAL(updatedata(float *,int,int,int)),this,SLOT(GUIUpdatePlotData(float*,int,int,int)));
    connect(&mthread,SIGNAL(initanaloginput()),this,SLOT(GUIInitAnalogInput()));
    connect(&mthread,SIGNAL(FilenameRequest()),this,SLOT(GetFileName()));


    //ui:lta threadille
    connect(ui->KanavienLkm,SIGNAL(currentIndexChanged(int)),&mthread,SLOT(amountchannels(int)));
    connect(ui->RangeSelector,SIGNAL(currentIndexChanged(QString)),&mthread,SLOT(InputRange(QString)));
    connect(ui->Log_check,SIGNAL(toggled(bool)),&mthread,SLOT(LogCheck(bool)));
    connect(ui->MeaFreq,SIGNAL(valueChanged(int)),&mthread,SLOT(Frequency(int)));
    connect(ui->Multiplier,SIGNAL(valueChanged(int)),&mthread,SLOT(Multiplier(int)));
    connect(this,SIGNAL(Filename(QString&)),&mthread,SLOT(GetFilename(QString&)));
    connect(ui->MeanOn,SIGNAL(toggled(bool)),&mthread,SLOT(MeanOn(bool)));
    connect(ui->MeanMultiplier,SIGNAL(valueChanged(int)),&mthread,SLOT(MeanMultiply(int)));
    connect(ui->TimeoutBox,SIGNAL(valueChanged(int)),&mthread,SLOT(BufTimeout(int)));
    connect(ui->SmartLog,SIGNAL(toggled(bool)),&mthread,SLOT(SmartLog(bool)));
    connect(ui->LogSensitivity,SIGNAL(valueChanged(double)),&mthread,SLOT(DatalogSensitivity(double)));
    connect(ui->FilterOn,SIGNAL(toggled(bool)),&mthread,SLOT(DatalogMean(bool)));
    connect(ui->FilterSelector,SIGNAL(currentIndexChanged(QString)),&mthread,SLOT(DatalogMeanSelector(QString)));
    connect(ui->RButton_Differential,SIGNAL(toggled(bool)),&mthread,SLOT(DifferentialMeasurement(bool)));
    connect(ui->SmartLog_2,SIGNAL(toggled(bool)),&mthread,SLOT(SmartLogAI0(bool)));
    connect(ui->BufValues,SIGNAL(valueChanged(int)),&mthread,SLOT(BufferValuesDisplayed(int)));
    connect(ui->ShowClock,SIGNAL(toggled(bool)),&mthread,SLOT(ShowProcessorClock(bool)));


    connect(&mthread,SIGNAL(toMonitor(const QString&)),this,SLOT(GUIPrintToMonitor(const QString&)));
    connect(&cthread,SIGNAL(ControlValve()),this,SLOT(GUIControlValve()));
    connect(&cthread,SIGNAL(toMonitor(const QString&)),this,SLOT(GUIPrintToMonitor(const QString&)));
    connect(&cthread,SIGNAL(toMonitorPlain(QString)),this,SLOT(GUIPrintToMonitorPlain(QString)));
    connect(ui->AOSlider,SIGNAL(valueChanged(int)),&cthread,SLOT(UpdateSliderValue(int)));
}


MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::GUIExit()
{
    QApplication::quit();
}


void MainWindow::GUIAbout()
{
    QMessageBox about;


    about.setText("Version 0.6");
    about.exec();
}


void MainWindow::GUIStartMeasurement()
{
    mthread.start();
    ui->groupBox->setDisabled(1);
```

```
}
void MainWindow::GUIStopMeasurement()
{
    mthread.quit();
    ui->groupBox->setEnabled(1);
}


void MainWindow::GUIUpdatePlotData(float *voltage_array,int multiplier, int channels, int Freq)
{
        if(ui->AnalogCheck_0->isChecked()) //jos rasti ruudussa, tulostetaan kuvaajaan
        {
            if(PlotCheck==1) //TESTI: luetaan koko puskurin puolikas
            {
                for(int i=0; i<(512*multiplier); i++)
                {
                    x0[PlotAika]=PlotAika;
                    y0[PlotAika]=voltage_array[i];
                    AnalogInput0->setData(x0, y0, PlotAika);//laitetaan data kuvaajan käyrään
                    PlotAika++;
                    ui->statusBar->showMessage("Running");
                }
            }
            else
            {
                x0[PlotAika]=PlotAika;
                y0[PlotAika]=voltage_array[0];
                AnalogInput0->setData(x0, y0, PlotAika); //laitetaan data kuvaajan käyrään
            }
        }


    if(ui->AnalogCheck_1->isChecked())
    {
            x1[PlotAika]=PlotAika;
            y1[PlotAika]=voltage_array[1];
            AnalogInput1->setData(x1, y1, PlotAika);
    }
    if(ui->AnalogCheck_2->isChecked())
    {
        x2[PlotAika]=PlotAika;
        if(DiffOn==1) y2[PlotAika]=voltage_array[1];
        else y2[PlotAika]=voltage_array[2];
        AnalogInput2->setData(x2, y2, PlotAika);
    }
    if(ui->AnalogCheck_3->isChecked())
    {
        x3[PlotAika]=PlotAika;
        y3[PlotAika]=voltage_array[3];
        AnalogInput3->setData(x3, y3, PlotAika);
    }
    if(ui->AnalogCheck_4->isChecked())
    {
        x4[PlotAika]=PlotAika;
        if(DiffOn==1) y4[PlotAika]=voltage_array[2];
        else y4[PlotAika]=voltage_array[4];
        AnalogInput4->setData(x4, y4, PlotAika);
    }
    if(ui->AnalogCheck_5->isChecked())
    {
        x5[PlotAika]=PlotAika;
        y5[PlotAika]=voltage_array[5];
        AnalogInput5->setData(x5, y5, PlotAika);
    }
    if(ui->AnalogCheck_6->isChecked())
    {
        x6[PlotAika]=PlotAika;
        if(DiffOn==1) y6[PlotAika]=voltage_array[3];
        else y6[PlotAika]=voltage_array[6];
        AnalogInput6->setData(x6, y6, PlotAika);
    }
    if(ui->AnalogCheck_7->isChecked())
    {
        x7[PlotAika]=PlotAika;
        y7[PlotAika]=voltage_array[7];
        AnalogInput7->setData(x7, y7, PlotAika);
    }
    if(ui->AnalogCheck_8->isChecked())
    {
        x8[PlotAika]=PlotAika;
        if(DiffOn==1) y8[PlotAika]=voltage_array[4];
        else y8[PlotAika]=voltage_array[8];
        AnalogInput8->setData(x8, y8, PlotAika);
    }
    if(ui->AnalogCheck_9->isChecked())
    {
        x9[PlotAika]=PlotAika;
        y9[PlotAika]=voltage_array[9];
        AnalogInput9->setData(x9, y9, PlotAika);
    }
    if(ui->AnalogCheck_10->isChecked())
    {
        x10[PlotAika]=PlotAika;
        if(DiffOn==1) y10[PlotAika]=voltage_array[5];
        else          y10[PlotAika]=voltage_array[10];
        AnalogInput10->setData(x10, y10, PlotAika);
    }
    if(ui->AnalogCheck_11->isChecked())
    {
```

```
        x11[PlotAika]=PlotAika;
        y11[PlotAika]=voltage_array[11];
        AnalogInput11->setData(x11, y11, PlotAika);
    }
    if(ui->AnalogCheck_12->isChecked())
    {
        x12[PlotAika]=PlotAika;
        if(DiffOn==1) y12[PlotAika]=voltage_array[6];
        else          y12[PlotAika]=voltage_array[12];
        AnalogInput12->setData(x12, y12, PlotAika);
    }
    if(ui->AnalogCheck_13->isChecked())
    {
        x13[PlotAika]=PlotAika;
        y13[PlotAika]=voltage_array[13];
        AnalogInput13->setData(x13, y13, PlotAika);
    }
    if(ui->AnalogCheck_14->isChecked())
    {
        x14[PlotAika]=PlotAika;
        if(DiffOn==1) y14[PlotAika]=voltage_array[7];
        else          y14[PlotAika]=voltage_array[14];
        AnalogInput14->setData(x14, y14, PlotAika);
    }
    if(ui->AnalogCheck_15->isChecked())
    {
        x15[PlotAika]=PlotAika;
        y15[PlotAika]=voltage_array[15];
        AnalogInput15->setData(x15, y15, PlotAika);
    }


    ui->AnalogInputPlot->replot();
    if(PlotCheck==0) PlotAika++;


    if(PlotAika>9999)PlotAika=0;


if(DiffOn==0)
    {
    AnalogValue.setNum(voltage_array[0]);
    ui->AnalogValue->setText(AnalogValue);


    if(channels>=2)
        {
        AnalogValue.setNum(voltage_array[1]);
        ui->AnalogValue_2->setText(AnalogValue);
        }
    else ui->AnalogValue_2->clear();
    if(channels>=3)
        {
        AnalogValue.setNum(voltage_array[2]);
        ui->AnalogValue_3->setText(AnalogValue);
        }
    else ui->AnalogValue_3->clear();
    if(channels>=4)
        {
        AnalogValue.setNum(voltage_array[3]);
        ui->AnalogValue_4->setText(AnalogValue);
        }
    else ui->AnalogValue_4->clear();
    if(channels>=5)
        {
        AnalogValue.setNum(voltage_array[4]);
        ui->AnalogValue_5->setText(AnalogValue);
        }
    else ui->AnalogValue_5->clear();
    if(channels>=6)
        {
        AnalogValue.setNum(voltage_array[5]);
        ui->AnalogValue_6->setText(AnalogValue);
        }
    else ui->AnalogValue_6->clear();
    if(channels>=7)
        {
        AnalogValue.setNum(voltage_array[6]);
        ui->AnalogValue_7->setText(AnalogValue);
        }
    else ui->AnalogValue_7->clear();
    if(channels>=8)
        {
        AnalogValue.setNum(voltage_array[7]);
        ui->AnalogValue_8->setText(AnalogValue);
        }
    else ui->AnalogValue_8->clear();
    if(channels>=9)
        {
        AnalogValue.setNum(voltage_array[8]);
        ui->AnalogValue_9->setText(AnalogValue);
        }
    else ui->AnalogValue_9->clear();
    if(channels>=10)
        {
        AnalogValue.setNum(voltage_array[9]);
        ui->AnalogValue_10->setText(AnalogValue);
        }
    else ui->AnalogValue_10->clear();
```

```
    if(channels>=11)
        {
        AnalogValue.setNum(voltage_array[10]);
        ui->AnalogValue_11->setText(AnalogValue);
        }
    else ui->AnalogValue_11->clear();
    if(channels>=12)
        {
        AnalogValue.setNum(voltage_array[11]);
        ui->AnalogValue_12->setText(AnalogValue);
        }
    else ui->AnalogValue_12->clear();
    if(channels>=13)
        {
        AnalogValue.setNum(voltage_array[12]);
        ui->AnalogValue_13->setText(AnalogValue);
        }
    else ui->AnalogValue_13->clear();
    if(channels>=14)
        {
        AnalogValue.setNum(voltage_array[13]);
        ui->AnalogValue_14->setText(AnalogValue);
        }
    else ui->AnalogValue_14->clear();
    if(channels>=15)
        {
        AnalogValue.setNum(voltage_array[14]);
        ui->AnalogValue_15->setText(AnalogValue);
        }
    else ui->AnalogValue_15->clear();
    if(channels>=16)
        {
        AnalogValue.setNum(voltage_array[15]);
        ui->AnalogValue_16->setText(AnalogValue);
        }
    else ui->AnalogValue_16->clear();
  }
  else
 {
      AnalogValue.setNum(voltage_array[0]);
      ui->AnalogValue->setText(AnalogValue);
      if(channels>=2)
      {
      AnalogValue.setNum(voltage_array[1]);
      ui->AnalogValue_3->setText(AnalogValue);
      }
      else ui->AnalogValue_3->clear();
      if(channels>=3)
      {
      AnalogValue.setNum(voltage_array[2]);
      ui->AnalogValue_5->setText(AnalogValue);
      }
      else ui->AnalogValue_5->clear();
      if(channels>=4)
      {
      AnalogValue.setNum(voltage_array[3]);
      ui->AnalogValue_7->setText(AnalogValue);
      }
      else ui->AnalogValue_7->clear();
      if(channels>=5)
      {
      AnalogValue.setNum(voltage_array[4]);
      ui->AnalogValue_9->setText(AnalogValue);
      }
      else ui->AnalogValue_9->clear();
      if(channels>=6)
      {
      AnalogValue.setNum(voltage_array[5]);
      ui->AnalogValue_11->setText(AnalogValue);
      }
      else ui->AnalogValue_11->clear();
      if(channels>=7)
      {
      AnalogValue.setNum(voltage_array[6]);
      ui->AnalogValue_13->setText(AnalogValue);
      }
      else ui->AnalogValue_13->clear();
      if(channels>=8)
      {
      AnalogValue.setNum(voltage_array[7]);
      ui->AnalogValue_15->setText(AnalogValue);
      }
      else ui->AnalogValue_15->clear();
  }
}


void MainWindow::InitPlot()
{
    //Plotterin gridi
    AnalogInput1Grid = new QwtPlotGrid();
    AnalogInput1Grid->setPen(QPen(Qt::DotLine));
    AnalogInput1Grid->enableY(true);
    AnalogInput1Grid->enableX(true);
    AnalogInput1Grid->attach(ui->AnalogInputPlot);
    ui->AnalogInputPlot->setAxisScale(QwtPlot::yLeft,-10,10,0); //koordinaatiston akselien skaala


    //Plotterin curvet
```

```
    AnalogInput0  = new QwtPlotCurve();
    AnalogInput0->setPen(QPen(Qt::black));
    AnalogInput0->attach(ui->AnalogInputPlot);


    AnalogInput1  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput1->setPen(QPen(Qt::red));
    AnalogInput1->attach(ui->AnalogInputPlot);


    AnalogInput2  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput2->setPen(QPen(Qt::green));
    AnalogInput2->attach(ui->AnalogInputPlot);


    AnalogInput3  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput3->setPen(QPen(Qt::yellow));
    AnalogInput3->attach(ui->AnalogInputPlot);


    AnalogInput4  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput4->setPen(QPen(Qt::lightGray));
    AnalogInput4->attach(ui->AnalogInputPlot);


    AnalogInput5  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput5->setPen(QPen(Qt::gray));
    AnalogInput5->attach(ui->AnalogInputPlot);


    AnalogInput6  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput6->setPen(QPen(Qt::blue));
    AnalogInput6->attach(ui->AnalogInputPlot);


    AnalogInput7  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput7->setPen(QPen(Qt::cyan));
    AnalogInput7->attach(ui->AnalogInputPlot);


    AnalogInput8  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput8->setPen(QPen(Qt::darkBlue));
    AnalogInput8->attach(ui->AnalogInputPlot);


    AnalogInput9  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput9->setPen(QPen(Qt::darkCyan));
    AnalogInput9->attach(ui->AnalogInputPlot);


    AnalogInput10  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput10->setPen(QPen(Qt::darkGreen));
    AnalogInput10->attach(ui->AnalogInputPlot);


    AnalogInput11  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput11->setPen(QPen(Qt::magenta));
    AnalogInput11->attach(ui->AnalogInputPlot);


    AnalogInput12  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput12->setPen(QPen(Qt::darkMagenta));
    AnalogInput12->attach(ui->AnalogInputPlot);


    AnalogInput13  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput13->setPen(QPen(Qt::darkYellow));
    AnalogInput13->attach(ui->AnalogInputPlot);


    AnalogInput14  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput14->setPen(QPen(Qt::darkRed));
    AnalogInput14->attach(ui->AnalogInputPlot);


    AnalogInput15  = new QwtPlotCurve();//new luo kekoon (heap) uuden muuttujan tilavarauksen
    AnalogInput15->setPen(QPen(Qt::darkGray));
    AnalogInput15->attach(ui->AnalogInputPlot);


    PlotAika=0;
    ydemo=0;
}


void MainWindow::exportSVG()
{


QString fileName;
fileName = QFileDialog::getSaveFileName(this, tr("Name of the file"), "pictures/plot.svg","Graphic files ( *svg)");


if ( !fileName.isEmpty() )
{
    QSvgGenerator generator;
    generator.setFileName(fileName);
    generator.setSize(QSize(800, 600));
    QwtPlotPrintFilter filter;
```

```
    int options = QwtPlotPrintFilter::PrintAll;
    options |= QwtPlotPrintFilter::PrintFrameWithScales| QwtPlotPrintFilter::PrintMargin;
    filter.setOptions(options);
    QPainter painter(&generator);
    ui->AnalogInputPlot->print(&painter,QRect(0,0,800,600),filter);
}
}


void MainWindow::GetFileName()
{
    filename = QFileDialog::getSaveFileName(this, tr("Save File"),"./logs/log.csv",tr(""));
    emit Filename(filename);
}


void MainWindow::ClearPlot()
{
    PlotAika=0;
    ui->AnalogInputPlot->replot();
}


void MainWindow::GUIStartControlling()
{
    cthread.start();
}


void MainWindow::GUIStopControlling()
{
    cthread.quit();
}


void MainWindow::GUIPrintToMonitor(const QString& text)
{
    //raw_time = QTime::currentTime();
    //time =raw_time.toString("hh:mm:ss");
    //timeandtext = ;
    ui->Monitori->append(text);
}


void MainWindow::GUIPrintToMonitorPlain(const QString& text)
{
    ui->Monitori->setPlainText(text);
}


void MainWindow::PlotAll(bool m_PlotCheck)
{
    PlotCheck = m_PlotCheck;
}


void MainWindow::BufferValuesDisplay(int m_Values)
{
    BufValues = m_Values;
}


void MainWindow::LcdValue(float slider)
{
    ui->lcdNumber->display(slider);
}


void MainWindow::Scaling(bool autoscale)
{
    //Säädetään automaattinen skaalaus päälle tai pois ja piilotetaan
    //tai näytetään manuaalisen skaalauksen elementit
   if(autoscale == 1)
    {
       ui->ManualScaling->setDisabled(1);
       ui->AnalogInputPlot->setAxisAutoScale(QwtPlot::yLeft); //koordinaatiston autoscale
    }
   else
   {
       ui->ManualScaling->setEnabled(1);
       lowery = ui->LowerValue->value();
       uppery = ui->UpperValue->value();
       ui->AnalogInputPlot->setAxisScale(QwtPlot::yLeft,lowery,uppery,0); //koordinaatiston akselien skaala
   }
}


void MainWindow::ShowHideSensitivity(bool SOnOff)
{
    if(SOnOff==1) ui->LogSensitivity->setDisabled(1);
    else ui->LogSensitivity->setDisabled(0);


}


void MainWindow::ShowHideLogging(bool LogOn)
{
    if(LogOn==1){ui->groupBox_3->setDisabled(1); ui->groupBox_4->setDisabled(1);}
```

```
    else{ui->groupBox_3->setEnabled(1); ui->groupBox_4->setEnabled(1);}
}


void MainWindow::ShowHideUnusedChans(int Chans)
{
    //Näytetään vain ne kanavat joita luetaan
    if(DiffOn==0)
    {
    Chans = Chans+1;
    if(Chans<=15) ui->Group16->setDisabled(1);
    else ui->Group16->setDisabled(0);
    if(Chans<=14) ui->Group15->setDisabled(1);
    else ui->Group15->setDisabled(0);
    if(Chans<=13) ui->Group14->setDisabled(1);
    else ui->Group14->setDisabled(0);
    if(Chans<=12) ui->Group13->setDisabled(1);
    else ui->Group13->setDisabled(0);
    if(Chans<=11) ui->Group12->setDisabled(1);
    else ui->Group12->setDisabled(0);
    if(Chans<=10) ui->Group11->setDisabled(1);
    else ui->Group11->setDisabled(0);
    if(Chans<=9) ui->Group10->setDisabled(1);
    else ui->Group10->setDisabled(0);
    if(Chans<=8) ui->Group9->setDisabled(1);
    else ui->Group9->setDisabled(0);
    if(Chans<=7) ui->Group8->setDisabled(1);
    else ui->Group8->setDisabled(0);
    if(Chans<=6) ui->Group7->setDisabled(1);
    else ui->Group7->setDisabled(0);
    if(Chans<=5) ui->Group6->setDisabled(1);
    else ui->Group6->setDisabled(0);
    if(Chans<=4) ui->Group5->setDisabled(1);
    else ui->Group5->setDisabled(0);
    if(Chans<=3) ui->Group4->setDisabled(1);
    else ui->Group4->setDisabled(0);
    if(Chans<=2) ui->Group3->setDisabled(1);
    else ui->Group3->setDisabled(0);
    if(Chans<=1) ui->Group2->setDisabled(1);
    else ui->Group2->setDisabled(0);
    }
    else
    {
        if(Chans<=6) ui->Group15->setDisabled(1);
        else ui->Group15->setDisabled(0);
        if(Chans<=5) ui->Group13->setDisabled(1);
        else ui->Group13->setDisabled(0);
        if(Chans<=4) ui->Group11->setDisabled(1);
        else ui->Group11->setDisabled(0);
        if(Chans<=3) ui->Group9->setDisabled(1);
        else ui->Group9->setDisabled(0);
        if(Chans<=2) ui->Group7->setDisabled(1);
        else ui->Group7->setDisabled(0);
        if(Chans<=1) ui->Group5->setDisabled(1);
        else ui->Group5->setDisabled(0);
        if(Chans<=0) ui->Group3->setDisabled(1);
        else ui->Group3->setDisabled(0);
    }
}


void MainWindow::DifferentialMeasurement(bool m_DiffOn)
{
    DiffOn = m_DiffOn;
    if (DiffOn==1)
    {
        ui->Group2->setHidden(1);
        ui->Group4->setHidden(1);
        ui->Group6->setHidden(1);
        ui->Group8->setHidden(1);
        ui->Group10->setHidden(1);
        ui->Group12->setHidden(1);
        ui->Group14->setHidden(1);
        ui->Group16->setHidden(1);


        ui->KanavienLkm->removeItem(15);
        ui->KanavienLkm->removeItem(14);
        ui->KanavienLkm->removeItem(13);
        ui->KanavienLkm->removeItem(12);
        ui->KanavienLkm->removeItem(11);
        ui->KanavienLkm->removeItem(10);
        ui->KanavienLkm->removeItem(9);
        ui->KanavienLkm->removeItem(8);
    }
    else
    {
        ui->Group2->setHidden(0);
        ui->Group4->setHidden(0);
        ui->Group6->setHidden(0);
        ui->Group8->setHidden(0);
        ui->Group10->setHidden(0);
        ui->Group12->setHidden(0);
        ui->Group14->setHidden(0);
        ui->Group16->setHidden(0);


        ui->KanavienLkm->removeItem(7);
        ui->KanavienLkm->removeItem(6);
```

```
        ui->KanavienLkm->removeItem(5);
        ui->KanavienLkm->removeItem(4);
        ui->KanavienLkm->removeItem(3);
        ui->KanavienLkm->removeItem(2);
        ui->KanavienLkm->removeItem(1);
        ui->KanavienLkm->removeItem(0);


        ui->KanavienLkm->addItems(Kanavat);
    }
}
```

Mittausluokan ohjelmatiedosto, measurementthread.cpp

```
#include "measurementthread.h"


MeasurementThread::MeasurementThread()
{
    State = 0;
    num_chan = 1;
    SAMPLE_RATE = 10000;
    multiplier = 1;
    flag = 0;
    Apuri1 = 0;
    KeskiarvoLuku = 1;
    Timeout = 3000;
    gain_code = 4; //+-10V
    SLog = 0;
    SLogAI0 = 0;
    SLogFlag = 0;
    LogMean = 0;
    Sensitivity = 0.05;
    memflag = 0;
    LogMeanMultiplier = 32;
    buffer2 = 0x0000;
    BufValues = 1;
}


void MeasurementThread::run()
{
    State=1;
    if(flag==0)
    {
    this->InitDevice("/dev/advdaq0");
    flag=1;
    }
    this->InitAnalogInputs();


        while(State==1)
        {
            this->Measure();
        }
}


void MeasurementThread::stopmeasurement()
{
    State=0;
}


void MeasurementThread::amountchannels(int channels)
{
    num_chan = channels + 1; //channels on indeksiluku, alkaa 0:sta
}


void MeasurementThread::GetFilename(QString &m_filename)
{
    filename=m_filename;
}


void MeasurementThread::Frequency(int freq)
{
    SAMPLE_RATE = freq;
}


void MeasurementThread::Multiplier(int m_multiplier)
{
    multiplier = m_multiplier;
}


void MeasurementThread::MeanOn(bool MOn)
{
    KeskiarvoON = MOn;
}
```

```
void MeasurementThread::MeanMultiply(int mean)
{
    KeskiarvoLuku = mean;
}


void MeasurementThread::BufTimeout(int m_Timeout)
{
    Timeout = m_Timeout;
}


void MeasurementThread::BufferValuesDisplayed(int m_BufValues)
{
    BufValues = m_BufValues;
    DividedValue = CONVERTION_NUMBER/2/BufValues; //häh
}


void MeasurementThread::InputRange(QString Range)
{
    if(Range=="+/-10V")gain_code = 4;
    else if(Range=="+/-5V")gain_code = 0;
    else if(Range=="+/-2.5 V")gain_code = 1;
    else if(Range=="+/-1.25 V")gain_code = 2;
    else if(Range=="+/-0.625V")gain_code = 3;
    else if(Range=="0 to 10V ")gain_code = 16;
    else if(Range=="0 to 5V")gain_code = 17;
    else if(Range=="0 to 2.5V")gain_code = 18;
    else if(Range=="0 to 1.25V")gain_code = 19;
}


void MeasurementThread::SmartLog(bool m_SLog)
{
    SLog = m_SLog;
}


void MeasurementThread::SmartLogAI0(bool m_SLogAI0)
{
    SLogAI0 = m_SLogAI0;
}


void MeasurementThread::DatalogSensitivity(double m_Sensitivity)
{
    Sensitivity = m_Sensitivity;
}


void MeasurementThread::DatalogMean(bool m_LogMean)
{
    LogMean = m_LogMean;
}


void MeasurementThread::DatalogMeanSelector(QString m_LogMeanMultiplier)
{
    if(m_LogMeanMultiplier=="4")LogMeanMultiplier = 4;
    else if(m_LogMeanMultiplier=="8")LogMeanMultiplier = 8;
    else if(m_LogMeanMultiplier=="16")LogMeanMultiplier = 16;
    else if(m_LogMeanMultiplier=="32")LogMeanMultiplier = 32;
    else if(m_LogMeanMultiplier=="64")LogMeanMultiplier = 64;
    else if(m_LogMeanMultiplier=="128")LogMeanMultiplier = 128;
    else if(m_LogMeanMultiplier=="256")LogMeanMultiplier = 256;
    else if(m_LogMeanMultiplier=="512")LogMeanMultiplier = 512;
}


void MeasurementThread::DifferentialMeasurement(bool m_DiffOn)
{
    DiffOn = m_DiffOn;
    if (m_DiffOn==1)
    {
        buffer2=0xFFFF;
    }
    else buffer2=0x0000;
}


void MeasurementThread::ShowProcessorClock(bool m_ShowClock)
{
    ShowClock = m_ShowClock;
}


QString MeasurementThread::getTime()
{
    raw_time = QTime::currentTime();
    time =raw_time.toString("hh:mm:ss:zzz");
    return time;
}


void MeasurementThread::Measure()
{
    if(fai_running==1)
    {
```

```
// Step 6: Check event
    check_event.EventType = 0; // check all event
    check_event.Milliseconds = Timeout; // kuinka kauan odotetaan eventtiä ennen timeoutia (ms)
    ret = DRV_CheckEvent(fd, &check_event);
    if(ret != 0)
    {
        emit toMonitor("ERROR7 code:");
        emit toMonitor(Error);
    }


    switch (check_event.EventType)
    {
    case ADS_EVT_AI_OVERRUN:
        i++;
        emit toMonitor("    Buffer overrun!");
        bufrdy = 0;
        break;
    case ADS_EVT_AI_LOBUFREADY:
        i++;
        //emit toMonitor("Low buffer ready!");
        fai_transfer.DataBuffer = bufptr_inp;
        fai_transfer.DataType = DATA_TYPE;
        fai_transfer.start = 0;
        fai_transfer.count = CONVERTION_NUMBER / 2;
        fai_transfer.overrun = &overrun;


        ret = DRV_FAITransfer(fd, &fai_transfer);
        if(ret != 0)
        {
            emit toMonitor("ERROR8 code:");
            emit toMonitor(Error);
            release_region_func(fd, &enable_event, gain_array, reading_array, voltage_array);
        }


        bufrdy = 1;
        break;


    case ADS_EVT_AI_HIBUFREADY:
        i++;
        //emit toMonitor("High buffer ready");
        fai_transfer.DataBuffer = bufptr_inp;
        fai_transfer.DataType = DATA_TYPE;
        fai_transfer.start = CONVERTION_NUMBER / 2;
        fai_transfer.count = CONVERTION_NUMBER / 2;
        fai_transfer.overrun = &overrun;


        ret = DRV_FAITransfer(fd, &fai_transfer);
        if(ret != 0)
        {
            emit toMonitor("ERROR9 code:");
            emit toMonitor(Error);
            release_region_func(fd, &enable_event, gain_array, NULL, voltage_array);
        }


        bufrdy = 1;
        break;


    case ADS_EVT_AI_TERMINATED:
        i++;
        emit toMonitor("Interrupt AI terminated!");
        bufrdy = 0;
        break;
    case ADS_EVT_DEVREMOVED:
        emit toMonitor("Device removed!\n");
        release_region_func(fd, &enable_event, gain_array, reading_array, voltage_array);
        bufrdy = 0;
        break;
    case ADS_EVT_TIME_OUT:
        emit toMonitor("Event time out!\n");
        bufrdy = 0;
        break;
    default:
        emit toMonitor("No events!\n");
        bufrdy = 0;
        break;
    }


if (bufrdy==1) //TÄMÄN SISÄÄN TULEE KAIKKI TIEDON KÄSITTELY!
{
    //Jos tämä päällä, lasketaan keskiarvo logille
    if (LogMean==1)
        {
        int p=0;


        for(int j=0;j<(512*multiplier/LogMeanMultiplier);j++)
            {
            Apuri0=0,Apuri1=0,Apuri2=0,Apuri3=0,Apuri4=0,Apuri5=0,Apuri6=0,Apuri7=0,Apuri8=0,
            Apuri9=0,Apuri10=0,Apuri11=0,Apuri12=0,Apuri13=0,Apuri14=0,Apuri15=0;
```

```
        while(p<(LogMeanMultiplier*num_chan*(j+1)))
        {
            Apuri0+=voltage_array[p];
            if(num_chan>=2)Apuri1+=voltage_array[(p+1)];
            if(num_chan>=3)Apuri2+=voltage_array[(p+2)];
            if(num_chan>=4)Apuri3+=voltage_array[(p+3)];
            if(num_chan>=5)Apuri4+=voltage_array[(p+4)];
            if(num_chan>=6)Apuri5+=voltage_array[(p+5)];
            if(num_chan>=7)Apuri6+=voltage_array[(p+6)];
            if(num_chan>=8)Apuri7+=voltage_array[(p+7)];
            if(num_chan>=9)Apuri8+=voltage_array[(p+8)];
            if(num_chan>=10)Apuri9+=voltage_array[(p+9)];
            if(num_chan>=11)Apuri10+=voltage_array[(p+10)];
            if(num_chan>=12)Apuri11+=voltage_array[(p+11)];
            if(num_chan>=13)Apuri12+=voltage_array[(p+12)];
            if(num_chan>=14)Apuri13+=voltage_array[(p+13)];
            if(num_chan>=15)Apuri14+=voltage_array[(p+14)];
            if(num_chan>=16)Apuri15+=voltage_array[(p+15)];
            p+=num_chan;
        }


        KA0[j] = Apuri0/LogMeanMultiplier;
        if(num_chan>=2)KA1[j] = Apuri1/LogMeanMultiplier;
        if(num_chan>=3)KA2[j] = Apuri2/LogMeanMultiplier;
        if(num_chan>=4)KA3[j] = Apuri3/LogMeanMultiplier;
        if(num_chan>=5)KA4[j] = Apuri4/LogMeanMultiplier;
        if(num_chan>=6)KA5[j] = Apuri5/LogMeanMultiplier;
        if(num_chan>=7)KA6[j] = Apuri6/LogMeanMultiplier;
        if(num_chan>=8)KA7[j] = Apuri7/LogMeanMultiplier;
        if(num_chan>=9)KA8[j] = Apuri8/LogMeanMultiplier;
        if(num_chan>=10)KA9[j] = Apuri9/LogMeanMultiplier;
        if(num_chan>=11)KA10[j] = Apuri10/LogMeanMultiplier;
        if(num_chan>=12)KA11[j] = Apuri11/LogMeanMultiplier;
        if(num_chan>=13)KA12[j] = Apuri12/LogMeanMultiplier;
        if(num_chan>=14)KA13[j] = Apuri13/LogMeanMultiplier;
        if(num_chan>=15)KA14[j] = Apuri14/LogMeanMultiplier;
        if(num_chan>=16)KA15[j] = Apuri15/LogMeanMultiplier;
        }
    }
//Datalogataan, jos rasti ruudussa
if (checked==1)
{
    Datalog();
}


if (KeskiarvoON==1)
{
    //Katsotaan, että keskiarvokerroin ei ylitä puskurin kokoa
    if (KeskiarvoLuku>(CONVERTION_NUMBER/2))
    {
        KeskiarvoLuku=CONVERTION_NUMBER/2;
        emit toMonitor("Cannot set mean bigger than buffer size/2");
    }


    for(int j=0;j<BufValues;j++)
    {
    //Tehdään niin monta kertaa kuin puskurista halutaan arvoja


    //nollataan apurit
    Apuri0=0,Apuri1=0,Apuri2=0,Apuri3=0,Apuri4=0,Apuri5=0,Apuri6=0,Apuri7=0,Apuri8=0,
    Apuri9=0,Apuri10=0,Apuri11=0,Apuri12=0,Apuri13=0,Apuri14=0,Apuri15=0;


    //Lasketaan keskiarvo kanaville plotia varten, yksi arvo per puskurin puolikas
    for(int i=0;i<(KeskiarvoLuku*num_chan);i+=num_chan)
    //Kasvatetaan apumuuttujaa i kanavien lukumäärän mukaan
    //yhtä monta kertaa kuin keskiarvokerroin =
    {
        Apuri0+=voltage_array[i+(j*DividedValue)];
        if(num_chan>=2)Apuri1+=voltage_array[(i+1)+(j*DividedValue)];
        if(num_chan>=3)Apuri2+=voltage_array[(i+2)+(j*DividedValue)];
        if(num_chan>=4)Apuri3+=voltage_array[(i+3)+(j*DividedValue)];
        if(num_chan>=5)Apuri4+=voltage_array[(i+4)+(j*DividedValue)];
        if(num_chan>=6)Apuri5+=voltage_array[(i+5)+(j*DividedValue)];
        if(num_chan>=7)Apuri6+=voltage_array[(i+6)+(j*DividedValue)];
        if(num_chan>=8)Apuri7+=voltage_array[(i+7)+(j*DividedValue)];
        if(num_chan>=9)Apuri8+=voltage_array[(i+8)+(j*DividedValue)];
        if(num_chan>=10)Apuri9+=voltage_array[(i+9)+(j*DividedValue)];
        if(num_chan>=11)Apuri10+=voltage_array[(i+10)+(j*DividedValue)];
        if(num_chan>=12)Apuri11+=voltage_array[(i+11)+(j*DividedValue)];
        if(num_chan>=13)Apuri12+=voltage_array[(i+12)+(j*DividedValue)];
        if(num_chan>=14)Apuri13+=voltage_array[(i+13)+(j*DividedValue)];
        if(num_chan>=15)Apuri14+=voltage_array[(i+14)+(j*DividedValue)];
        if(num_chan>=16)Apuri15+=voltage_array[(i+15)+(j*DividedValue)];
    }
    //Jaetaan keskiarvokertoimella - saadaan keskiarvo
    voltage_array[0+(j*num_chan)] = Apuri0/KeskiarvoLuku;
    if(num_chan>=2)voltage_array[1+(j*num_chan)] = Apuri1/KeskiarvoLuku;
    if(num_chan>=3)voltage_array[2+(j*num_chan)] = Apuri2/KeskiarvoLuku;
    if(num_chan>=4)voltage_array[3+(j*num_chan)] = Apuri3/KeskiarvoLuku;
    if(num_chan>=5)voltage_array[4+(j*num_chan)] = Apuri4/KeskiarvoLuku;
    if(num_chan>=6)voltage_array[5+(j*num_chan)] = Apuri5/KeskiarvoLuku;
    if(num_chan>=7)voltage_array[6+(j*num_chan)] = Apuri6/KeskiarvoLuku;
    if(num_chan>=8)voltage_array[7+(j*num_chan)] = Apuri7/KeskiarvoLuku;
    if(num_chan>=9)voltage_array[8+(j*num_chan)] = Apuri8/KeskiarvoLuku;
```

```
            if(num_chan>=10)voltage_array[9+(j*num_chan)] = Apuri9/KeskiarvoLuku;
            if(num_chan>=11)voltage_array[10+(j*num_chan)] = Apuri10/KeskiarvoLuku;
            if(num_chan>=12)voltage_array[11+(j*num_chan)] = Apuri11/KeskiarvoLuku;
            if(num_chan>=13)voltage_array[12+(j*num_chan)] = Apuri12/KeskiarvoLuku;
            if(num_chan>=14)voltage_array[13+(j*num_chan)] = Apuri13/KeskiarvoLuku;
            if(num_chan>=15)voltage_array[14+(j*num_chan)] = Apuri14/KeskiarvoLuku;
            if(num_chan>=16)voltage_array[15+(j*num_chan)] = Apuri15/KeskiarvoLuku;
        }
        }
        //TÄSSÄ LÄHETETÄÄN DATA Measurementthreadiin
        emit updatedata(voltage_array,multiplier,num_chan,SAMPLE_RATE);
    }
    //Jos painettu stoppia, nollataan puskurit ja pysäytetään lukeminen
    if(State==0)
    {
        release_region_func(fd, &enable_event, gain_array, NULL, voltage_array);
    }
  }
}


void MeasurementThread::InitAnalogInputs()
{
    //23.9
    /* Step 2: Initialize */


    CONVERTION_NUMBER = (FIFO_SIZE * num_chan * multiplier); //KUINKA ISO BUFFERI TEHDÄÄN!!


    emit toMonitor("Analog input init");


    gain_array = (unsigned short *) malloc(num_chan * sizeof(unsigned short));
    reading_array = (unsigned short *) malloc(CONVERTION_NUMBER * sizeof(unsigned short));
    voltage_array = (float *) malloc(CONVERTION_NUMBER * sizeof(float));


    memset(reading_array, 0, CONVERTION_NUMBER * sizeof(unsigned short));
    memset(voltage_array, 0, CONVERTION_NUMBER * sizeof(float));


    memset(&fai_int_scan_start, 0, sizeof(PT_FAIIntStart));
    memset(&enable_event, 0, sizeof(PT_EnableEvent));
    memset(&check_event, 0, sizeof(PT_CheckEvent));
    memset(&fai_transfer, 0, sizeof(PT_FAITransfer));


    // u have to fill gain_array by yourself
        for (i = 0; i < num_chan; i++)
        {
            gain_array[i] = gain_code;
        }


    /* Step 3: Set Single-end or Differential */
    //buffer2 = 0xffff; /* 0x0000: all single-ended*/
                        /* 0xffff: all differential */
    ret = DRV_DeviceSetProperty(fd, CFG_AiChanConfig, &buffer2, sizeof(unsigned int));
    Error.setNum(ret);
    if(ret == 0) emit toMonitor("Initialized");
    else
    {
        emit toMonitor("ERROR1 code:");
        emit toMonitor(Error);
    }


   //23.9
   /* Step 4: Enable events*/


   //Bufferin ylivuoto
   enable_event.EventType = ADS_EVT_AI_OVERRUN;
   enable_event.Enabled   = 1;
   enable_event.Count     = 1;
   ret = DRV_EnableEvent(fd, &enable_event);
   if(ret == 0);
   else
   {
       emit toMonitor("ERROR2 code:");
       emit toMonitor(Error);
   }


   //"alempi" bufferin puolikas valmis
   enable_event.EventType = ADS_EVT_AI_LOBUFREADY;
   enable_event.Enabled    = 1;
   enable_event.Count     = 1;
   ret = DRV_EnableEvent(fd, &enable_event);
   if(ret == 0);
   else
   {
       emit toMonitor("ERROR3 code:");
       emit toMonitor(Error);
   }


   //"ylempi" bufferin puolikas valmis
```

```
    enable_event.EventType =  ADS_EVT_AI_HIBUFREADY;
    enable_event.Enabled    = 1;
    enable_event.Count     = 1;
    ret = DRV_EnableEvent(fd, &enable_event);
    if(ret == 0);
    else
    {
        emit toMonitor("ERROR4 code:");
        emit toMonitor(Error);
    }


    //Ohjelma lopetetaan
    enable_event.EventType = ADS_EVT_AI_TERMINATED;
    enable_event.Enabled    = 1;
    enable_event.Count     = 1;
    ret = DRV_EnableEvent(fd, &enable_event);
    if(ret == 0);
    else
    {
        emit toMonitor("ERROR5 code:");
        emit toMonitor(Error);
    }


    //Step 5: Start interrupt Transfer
    fai_int_scan_start.TrigSrc    = 0;
    fai_int_scan_start.SampleRate = SAMPLE_RATE;
    fai_int_scan_start.NumChans   = num_chan;
    fai_int_scan_start.StartChan  = 0; //start_chan;
    fai_int_scan_start.GainList   = gain_array;
    fai_int_scan_start.buffer     = reading_array;
    fai_int_scan_start.count      = CONVERTION_NUMBER;
    fai_int_scan_start.cyclic     = 1;
    fai_int_scan_start.IntrCount  = FIFO_SIZE / 2; /* 1: without fifo */


    ret = DRV_FAIIntScanStart(fd, &fai_int_scan_start);
    Error.setNum(ret);
    if(ret == 0) {emit toMonitor("Transfer started");fai_running = 1;}
    else
    {
        emit toMonitor("ERROR during AI startup, code:");
        emit toMonitor(Error);
        release_region_func(fd, &enable_event, gain_array, reading_array, voltage_array);
    }


    //Datatyypin switch (meillä CASE 1, jätetty myöhempiä muutoksia varten tähän)
    switch (DATA_TYPE)
    {
            case 0:                     // binary in
                    bufptr_inp = (void *) reading_array;
                    bufptr_out = (void *) reading_array;
                    break;
            case 1:                     // voltage in
                    bufptr_inp = (void *) voltage_array;
                    bufptr_out = NULL;
                    if(ret==0) emit toMonitor("Reading voltage");
                    break;
            default:
                    emit toMonitor("Wrong data type");
     }
}


void MeasurementThread::release_region_func
(int fd, PT_EnableEvent *event, unsigned short *buffer0, unsigned short *buffer1, float *buffer2)
{
    //Täällä tyhjennetään puskurit ja pysäytetään analogiatulojen luku.


    // disable event
    if (event != NULL)
    {
            event->EventType = ADS_EVT_AI_LOBUFREADY;
            event->Enabled    = 0;
            event->Count     = 1;
            DRV_EnableEvent(fd, event);


            event->EventType = ADS_EVT_AI_HIBUFREADY;
            event->Enabled    = 0;
            event->Count     = 1;
            DRV_EnableEvent(fd, event);


            event->EventType = ADS_EVT_AI_OVERRUN;
            event->Enabled    = 0;
            event->Count     = 1;
            DRV_EnableEvent(fd, event);


            event->EventType = ADS_EVT_AI_TERMINATED;
            event->Enabled    = 0;
            event->Count     = 1;
            DRV_EnableEvent(fd, event);
```

```
    }


    // terminate fast ai
    if (fai_running)
    {
        DRV_FAITerminate(fd);
    }


    // free buffers
    if (!buffer0)
    {
        free(buffer0);
    }


    if (!buffer1)
    {
        free(buffer1);
    }


    if (!buffer2)
    {
        free(buffer2);
    }
    //DRV_DeviceClose(&fd); //laitteen sulkeminen, (ei tarpeen)
}


void MeasurementThread::LogCheck(bool m_checked)
{
    checked = m_checked;
    //Tehdään logifile
        if(checked == true)
        {
            long MittausTiheys = 1000000/SAMPLE_RATE;


            //Haetaan tiedostonimi MainWindow:lta
            emit FilenameRequest();


            if(filename!=NULL)
            {
            emit toMonitor("Logging to file..");
            }
            else toMonitor("filename not received");


            QFile file(filename);
            file.open(QIODevice::Append | QIODevice::Text);
            QTextStream out(&file);


            //tulostetaan näyteväli ja mahdollinen keskiarvokerroin
            out << "Measurement period was: " << MittausTiheys << " microseconds * number of channels";
            if(LogMean==1) out << " * mean of " << LogMeanMultiplier << " values";
            out << endl;


            //Tehdään niin monta sarakeotsikkoa kuin on kanavia + aika
            out << "t(us),";
            out << "AI0,";
                    if(num_chan>=2){out << "AI1,";}
                    if(num_chan>=3){out << "AI2,";}
                    if(num_chan>=4){out << "AI3,";}
                    if(num_chan>=5){out << "AI4,";}
                    if(num_chan>=6){out << "AI5,";}
                    if(num_chan>=7){out << "AI6,";}
                    if(num_chan>=8){out << "AI7,";}
                    if(num_chan>=9){out << "AI8,";}
                    if(num_chan>=10){out << "AI9,";}
                    if(num_chan>=11){out << "AI10,";}
                    if(num_chan>=12){out << "AI11,";}
                    if(num_chan>=13){out << "AI12,";}
                    if(num_chan>=14){out << "AI13,";}
                    if(num_chan>=15){out << "AI14,";}
                    if(num_chan>=16){out << "AI15,";}
                    if(ShowClock==1){out << "t_Processor clock";}
            out << endl;
        }
        else
        {
            emit toMonitor("Logging ended");
            //timer2->stop();
            MittausAika=0;
        }
}


void MeasurementThread::Datalog()
{
    if(checked==1)
    {
        //Tulostetaan arvot tiedostoon
       QFile file(filename);
       file.open(QIODevice::Append | QIODevice::Text);
```

```
QTextStream out(&file);


    //Jos tämä päällä, ei logata, jos kaikki arvot ovat alle raja-arvon
    if(SLog==1)
    {
        SLogFlag=1; //oletusarvoisesti ei logata
        for(i1=0;i1>0;i1++)
        {
            if(voltage_array[i1]!=NULL)
            {
             if(voltage_array[i1]>=Sensitivity) //katsotaan, onko joku arvoista yli raja-arvon
             {SLogFlag=0;}                      //jos on, logataan
            }
            else break; //poistutaan for-lauseesta, kun lukujonon arvo on NULL, eli lukujono on loppu
        }
        i1=0;
    }
    //Logataan, jos AI0:n arvot ovat alle raja-arvon
    else if(SLogAI0==1)
    {
        SLogFlag=1;  //oletusarvoisesti ei logata
        for(i2=0;i2>=0;i2+=num_chan)
          {
            if(voltage_array[i2]!=NULL)
            {
              if(voltage_array[i2]>=Sensitivity) //katsotaan, onko joku arvoista yli raja-arvon
              {SLogFlag=0;}                      //jos on, logataan.
            }
            else break; //poistutaan for-lauseesta, kun lukujonon arvo on NULL, eli lukujono on loppu
          }
        i2=0;
    }
    else SLogFlag=0;


    if(LogMean==1 && SLogFlag==0)
    {
        //keskiarvottujen arvojen tulostaminen logiin
        for(int k=0;k<(512*multiplier/LogMeanMultiplier);k++)
        {
            out << MittausAika << ",";
            out << KA0[k] << ",";
            if(num_chan>=2) out << KA1[k] << ",";
            if(num_chan>=3) out << KA2[k] << ",";
            if(num_chan>=4) out << KA3[k] << ",";
            if(num_chan>=5) out << KA4[k] << ",";
            if(num_chan>=6) out << KA5[k] << ",";
            if(num_chan>=7) out << KA6[k] << ",";
            if(num_chan>=8) out << KA7[k] << ",";
            if(num_chan>=9) out << KA8[k] << ",";
            if(num_chan>=10) out << KA9[k] << ",";
            if(num_chan>=11) out << KA10[k] << ",";
            if(num_chan>=12) out << KA11[k] << ",";
            if(num_chan>=13) out << KA12[k] << ",";
            if(num_chan>=14) out << KA13[k] << ",";
            if(num_chan>=15) out << KA14[k] << ",";
            if(num_chan>=16) out << KA15[k] << ",";
            if(ShowClock==1) out << ProcessorClock;
            out << endl;
            MittausAika = MittausAika + (1000000/SAMPLE_RATE*num_chan*LogMeanMultiplier);
        }
    }


   if(SLogFlag==0 && LogMean==0)
    {
     //tavallisten tietojen tulostaminen logiin
     for(int i=0;i>=0;i+=num_chan) //Kasvatetaan muuttujaa i kanavien lukumäärän mukaan
     {
         if(voltage_array[i]!=NULL) //tehdään niin kauan kunnes lukujono "päättyy" (arvo on NULL lukujonon "ulkopuolella")
         {
         //Logataan tiedostoon sen mukaan, kuinka monta kanavaa on käytössä
         out << MittausAika << ",";
         if(num_chan==1) out << voltage_array[i];
         if(num_chan>=2) out << "," << voltage_array[i+1];
         if(num_chan>=3) out << "," << voltage_array[i+2];
         if(num_chan>=4) out << "," << voltage_array[i+3];
         if(num_chan>=5) out << "," << voltage_array[i+4];
         if(num_chan>=6) out << "," << voltage_array[i+5];
         if(num_chan>=7) out << "," << voltage_array[i+6];
         if(num_chan>=8) out << "," << voltage_array[i+7];
         if(num_chan>=9) out << "," << voltage_array[i+8];
         if(num_chan>=10) out << "," << voltage_array[i+9];
         if(num_chan>=11) out << "," << voltage_array[i+10];
         if(num_chan>=12) out << "," << voltage_array[i+11];
         if(num_chan>=13) out << "," << voltage_array[i+12];
         if(num_chan>=14) out << "," << voltage_array[i+13];
         if(num_chan>=15) out << "," << voltage_array[i+14];
         if(num_chan>=16) out << "," << voltage_array[i+15];
         else break;
         if(ShowClock==1)
         {
             ProcessorClock = getTime();
             out << "," << ProcessorClock;
         }
         out << endl;
```

```
                //Kasvatetaan mittausajan muuttujaa (mikrosekunteja)
                MittausAika = MittausAika + (1000000/SAMPLE_RATE*num_chan);
                }
                else break; //for-lauseen hajotus
            }
          }
        }
}


void MeasurementThread::InitDevice(char *device)
{
    emit toMonitor("Initializing device: ");
    emit toMonitor(device);
    emit toMonitor("...");


    ret = DRV_DeviceOpen(device, &fd);
    Error.setNum(ret);
    if(ret == 0) toMonitor("Device opened");


    else
    {
        emit toMonitor("Error code: ");
        emit toMonitor(Error);
    }
}
```

Analogialähtöjen ohjelmatiedosto, controlthread.cpp

```
#include "controlthread.h"


controlthread::controlthread()
{


}


void controlthread::run()
{
    controlflag=1;
    this->InitDevice("/dev/advdaq0");
    this->InitAnalogOutputs();//emit InitAnalogOutputs();
    while(controlflag==1)
    {
        this->ControlValve();
        msleep(100);
    }
}


void controlthread::stop()
{
    controlflag=0;
    emit toMonitor("Analog output closed");
}


void controlthread::InitAnalogOutputs()
{
    FP32 aoutMinValue = -10.0;
    FP32 aoutMaxValue = 10.0;
    aoutOutputValue = 0.0;


    AOConfig.chan = 0;
    AOConfig.RefSrc = 0;
    AOConfig.MinValue = aoutMinValue;
    AOConfig.MaxValue = aoutMaxValue;


    AOVoltageOut.chan = 0;
    AOVoltageOut.OutputValue = aoutOutputValue;


    ret = DRV_AOConfig(fd, &AOConfig);
    Error.setNum(ret);
    if(ret != 0)


        {
            emit toMonitor("Error during AOUT cconfig, code:");
            emit toMonitor(Error);
        }


    ret = DRV_AOVoltageOut(fd, &AOVoltageOut);
    Error.setNum(ret);
    if(ret != 0)
```

```
        {
            emit toMonitor("Error during AOUT voltageout, code:");
            emit toMonitor(Error);
        }
    emit toMonitor("Analog output initialized");
}


void controlthread::ControlValve()
{
    AOVoltageOut.OutputValue = SliderValue;


    ret = DRV_AOVoltageOut(fd, &AOVoltageOut);
    Error.setNum(ret);
    if(ret != 0)


        {
            emit toMonitor("Error during AOUT voltageout, code:");
            emit toMonitor(Error);
        }
}


void controlthread::UpdateSliderValue(int value)
{
    SliderValue = (float)value/3200;
    emit toValue(SliderValue);
}


void controlthread::InitDevice(char *device)
{
    emit toMonitor("Initializing device: ");
    emit toMonitor(device);
    emit toMonitor("...");


    ret = DRV_DeviceOpen(device, &fd);
    Error.setNum(ret);
    if(ret == 0) toMonitor("Device opened");


    else
    {
       emit toMonitor("Error code: ");
       emit toMonitor(Error);
    }
}
```