Timo-Tapio Palmu

Järjestelmä langattoman anturidatan keräämiseen XBee-moduuleilla

Metropolia Ammattikorkeakoulu
Insinööri(AMK)
Elektroniikan koulutusohjelma
Insinöörityö
1.6.2011

Tiivistelmä

Tekijä Otsikko Sivumäärä Päivämäärä	Timo-Tapio Palmu Järjestelmä langattoman anturitiedon tallentamiseen 35 sivua+ 3 liitettä 19.4.2011
Tutkinto	Insinööri(AMK)
Koulutusohjelma	Elektroniikan koulutusohjelma
Suuntautumisvaihtoehto	Elektroniikkasuunnittelu
Ohjaaja	Lehtori Jukka-Pekka Pirinen

Tässäinsinöörityössä on suunniteltu ja toteutettu järjestelmä langattoman anturitiedon keräämiseen ja tallentamiseen XBee-moduuleita hyväksikäyttäen. Prosessi toteutettiin erillisen anturikeskitinlaitteen avulla. Anturikeskitin sijoittuu järjestelmässä niiden XBee-moduulien, jotka lähettävät mittausdataa, ja palvelimen väliin.

XBee-moduuli käyttää hyväksi ZigBee-määritelmää, joka rakentuu IEEE 802.15.4 -standardin siirtotien päälle. XBee sisältää ZigBee-määritelmän lisäksi käyttöliittymät UART-tiedonsiirtolinjalle, hallittavuuden AT-kommennoin ja itsenäisen analogisesta digitaaliseen -muunnoksen.

Suunnitelman lähtökohtana oli käyttää XBee-moduulin analogisesta digitaaliseen-muunnosta ja moduulin sisäänrakennettua kehystä muunnoksen välittämiseen moduulien välillä. Sen jälkeen kyseinen data oli purettava, käsiteltävä ja välitettävä järjestelmän seuraavalle osalle, eli tallennusrakenteeseen. Tehtävää varten suunniteltiin anturikeskitin, joka kerää usealta XBee-moduulilta mittauksia, säilöö ne ja lähettää pyydettäessä Ethernet-verkon yli palvelimelle. Ethernet-verkon protokollaksi valittiin UDP/IP sen yksinkertaisuuden vuoksi, ja sen päälle määriteltiin oma protokolla mittaustietojen siirtämistä varten.

Anturikeskitin toteutettiin Atmega168-mikrokontrollerilla, johon liitettiin ENC28J60-verkkopiiri ja XBee-moduuli. Toteutusta varten piti myös kirjoittaa mikrokontrolleriohjelmisto, joka purkaa XBee-moduulin ADC-kehyksen, säilöö puretun kehyksen muistiin ja lähettää sen pyydettäessä palvelimelle. Lisäksi ohjelmiston täytyy keskustella yhteys palvelimelle.

Palvelinohjelmisto toteutettiin POSIX-yhteensopivassa ympäristössä. Sen tehtäväksi jäi keskustella yhteys anturikeskittimelle ja tallentaa tieto tietokantaan, joka tässä tapauksessa oli MySQL-tietokanta.

Avainsanat	IEEE 802.15.4, ZigBee, Ethernet, UDP, Atmega, MySQL

Author Title Number of Pages	Timo-Tapio Palmu A system for saving wireless sensor data 35 pages + 3 appendices

Abstract

Date	19 April 2011
Degree	Bachelor of Engineering
Degree Programme	Electronics engineering
Specialisation option	Electronics design
Instructor	Jukka-Pekka Pirinen, Lecturer

The objective of this thesis was to design and implement a system for gathering and saving information measured with XBee modules.

The Xbee module uses ZigBee definition which extends the IEEE 802.15.4 standard. In addition, the XBee modules include an interface for UART data transfer, command-interface using AT commands and built-in analog-to-digital conversion for up to 6 inputs.

The design is based around the analog-to-digital conversion of the Xbee module and the ADC frame that it uses to communicate the data between the modules. The data needs to be extracted from the said frame and forwarded to the next part of the system, which is the data storage. A sensor hub was designed for this purpose. The sensor hub gathers data from several XBee modules and then sends the data to the server through an Ethernet interface.

The UDP/IP protocol was chosen to be used over the Ethernet protocol because of its simplicity. An additional protocol was designed over UDP for the purpose of relaying the measurement data.

The sensor hub was implemented with the Atmega168 microcontroller which was connected to an XBee module and the ENC28J60 Ethernet controller. The implementation also required software for the microcontroller which extracts the data from XBee ADC frame, stores it in RAM and sends the data to the server when it is requested. In addition, the software must initialize a connection to the server.

The server software was implemented in a POSIX compliant environment. The server's duties are accepting connections from the sensor hubs and saving the data to a MySQL database.

Keywords	IEEE 802.15.4, ZigBee,Ethernet, UDP, Atmega, MySQL

Sisällys

Liitteet

Liite 1. Anturikeskittimen ohjelma

Liite 2. Palvelinohjelma

Liite 3. Anturikeskittimen kytkentä

1 Johdanto

Tässä insinöörityössä suunnitellaan ja toteutetaan kokonainen järjestelmä, jolla kerätään tietoa langattomasti XBee-moduulissa kiinni olevilta antureilta. Keskeinen osa työtä on anturikeskitin, joka ottaa vastaan XBee-moduuleilta tulevia mittauksia ja lähettää ne palvelimelle, joka tallentaa tiedot tietokantaan.

Järjestelmä koostuu XBee-päätelaitteista, anturikeskittimestä ja palvelimesta joka tallentaa datan tietokantaan. Anturit yhdistetään Xbee-moduuliin, joka lähettää mittausdatan anturikeskittimen XBee-moduulille.

Anturikeskittimen lisäksi vaaditaan myös se tietovarasto, mihin tieto tallennetaan lopullisesti. Tämän osan hoitaa palvelinohjelmisto, joka yhdistyy tietokantaan. Tieto siirretään anturikeskittimeltä palvelimelle tavanomaisen Ethernet-verkon yli käyttäen UDP/IP-protokollapinoa. Kun mittausdata on palvelimella, se tallennetaan tietokantaan.

XBee-moduulit ovat valmiita komponentteja, joissa toiminnallisuus on sisäänrakennettu. Anturikeskitin suunnitellaan ja toteutetaan Atmega168-mikrokontrollerin ympärille, johon liitetään XBee-moduuli ja Ethernet-verkkoliitäntä. Palvelinohjelma toteutetaan POSIX-yhteensopivaan ympäristöön vastaanottamaan mittausdataa anturikeskittimeltä.

2 Lyhyesti ZigBeestä

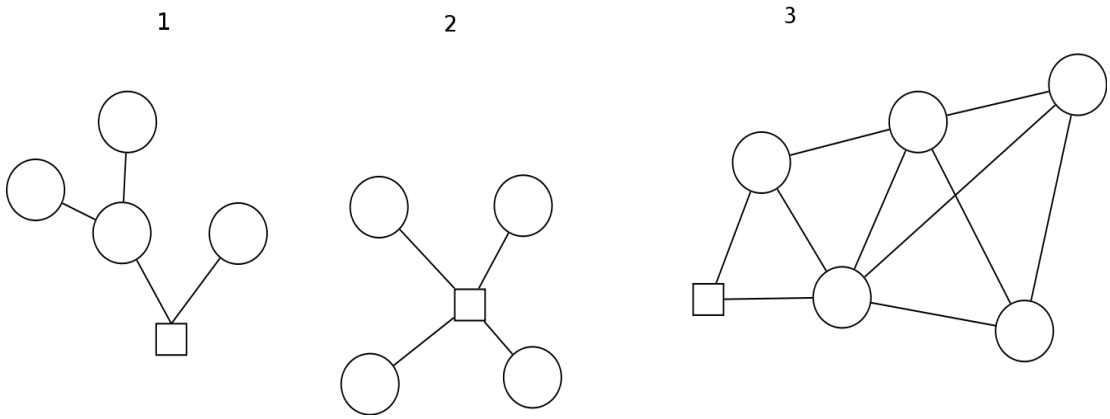
2.1 IEEE 802.15.4 -standardin esittely

IEEE 802.15.4 -standardi määrittelee protokollan lyhyen kantaman matalan nopeuden langattomalle verkolle. Toisin sanoen, IEEE 802.15.4 on ratkaisu matalan nopeuden henkilökohtaiselle langattomalle verkolle (LR-WPAN; Low Rate Wireless Personal Area Network). IEEE 802.15.4 toimii 2.45 Ghz:n taajuusalueen lisäksi 868 MHz:n (Eurooppa) ja 915 MHz:n (Yhdysvallat) taajuusalueilla. [1, s. 27.]

IEEE 802.15.4 määrittelee kahden tyyppisiä laitteita FFD-laitteita (Full-Function Device) ja RFD-laitteita (reduced function device). FFD-laite voi toimia verkossa koordinaattorina, reitittimenä tai päätelaitteena. FFD-laitteet voivat kommunikoida sekä FFD-laitteiden, että RFD-laitteiden kanssa. RFD-laitteet voivat keskustella vain ja ainoastaan yhden FFD-laitteen kanssa. [1, s. 14 – 15.]

2.1.1 IEEE 802.15.4 -standardin verkkotyypit

IEEE 802.15.4 -verkko voidaan rakentaa tähti-, puu-, tai solmuverkoksi. Yksinkertaisin ratkaisu on tähtirakenne (*Star Topology*), jossa päätelaitteet kommunikoivat vain koordinaattorin kanssa. Puuverkkorakenteessa (*Cluster Tree Topology*) päätelaitteet ovat yhteydessä koordinaattoriin tai reitittimeen. Reitittimet ovat yhteydessä seuraavaan reitittimeen tai koordinaattoriin välittäen informaatiota koordinaattorilta päätelaitteelle tai päätelaitteelta koordinaattorille. Puuverkkorakenne voidaan mieltää rakenteeksi, jossa on useita tähtiverkkorakenteita. Solmuverkkorakenne (*Mesh Topology*) koostuu samoista osista kuin puuverkkorakenne, mutta siinä päätelaitteet voivat olla yhteydessä useampaan reitittimeen. Reitittimet ja koordinaattori ovat FFD-laitteita ja päätelaitteet ovat joko FFD- tai RFD-laitteita. Kuvassa 1 esitetään edellämaintut verkkorakenteet. [1, s. 13-15; 2, s. 11-14.]



Kuva 1. 1) Puu-, 2) Tähti- ja 3) Solmurakenne

2.2 ZigBee-spesifikaatio

ZigBee on määritelmä, joka jatkaa IEEE 802.15.4-2003 (WPAN; *Wireless Personal Area Network*) -standardia. ZigBee määrittelee IEEE 802.15.4 -standardin päälle verkkokerroksen (*ZigBee Network Layer*) ja sovelluskerroksen (*ZigBee Application Layer*). [3; 4.]

ZigBee-standardi on suunnattu matalatehoisiin radiotaajuussovelluksiin, jotka vaativat pitkää akku-/patterikestoa ja tietoturvaa. ZigBee-standardi käyttää taajuuksia 2.4 GHz:n ympäristössä sekä 868 Mhz:n (Eurooppa) ja 915 Mhz:n (Yhdysvallat) ympäristöissä. Siinä, missä IEEE 802.15.4 määrittelee protokollasta fyysisen kerroksen ja siirtoyhteyskerroksen, määrittelee ZigBee verkko-, sovelluskerrokset. [4.]

Tietoturva on ollut yksi keskeisistä päämääristä ZigBee-standardin kehittämisessä. ZigBee on suunniteltu käyttämään SKKE-standardia (Symmetric-Key Key Exchange) joka perustuu AES-standardiin (Advanced Encryption Standard). [2, s.15.]

2.3 XBee-moduuli

XBee-moduuli on ZigBee-protokollan mukainen verkkolaite, joka toimii 2,4 GHz:n taajuudella. XBee-moduuli tarjoaa ZigBee-protokollan lisäksi 10-bittisen A/D-muunnoksen ilman erillisiä laitteita. Tämä A/D-muunnos voidaan lähettää automaattisesti eteenpäin omassa kehyksessä. A/D-muunnokseen voidaan käyttää yhteensä kuutta I/O-linjaa. [2, s. 17; 5, s. 13.]

XBee-moduuli tarjoaa myös UART-siirtotien, jotta sen voi yhdistää esimerkiksi mikrokontrolleriin. Lisäksi XBee-moduulia voi konfiguroida AT-komennoilla. [5, s. 10;5, s. 4.]

3 Järjestelmän suunnitelma

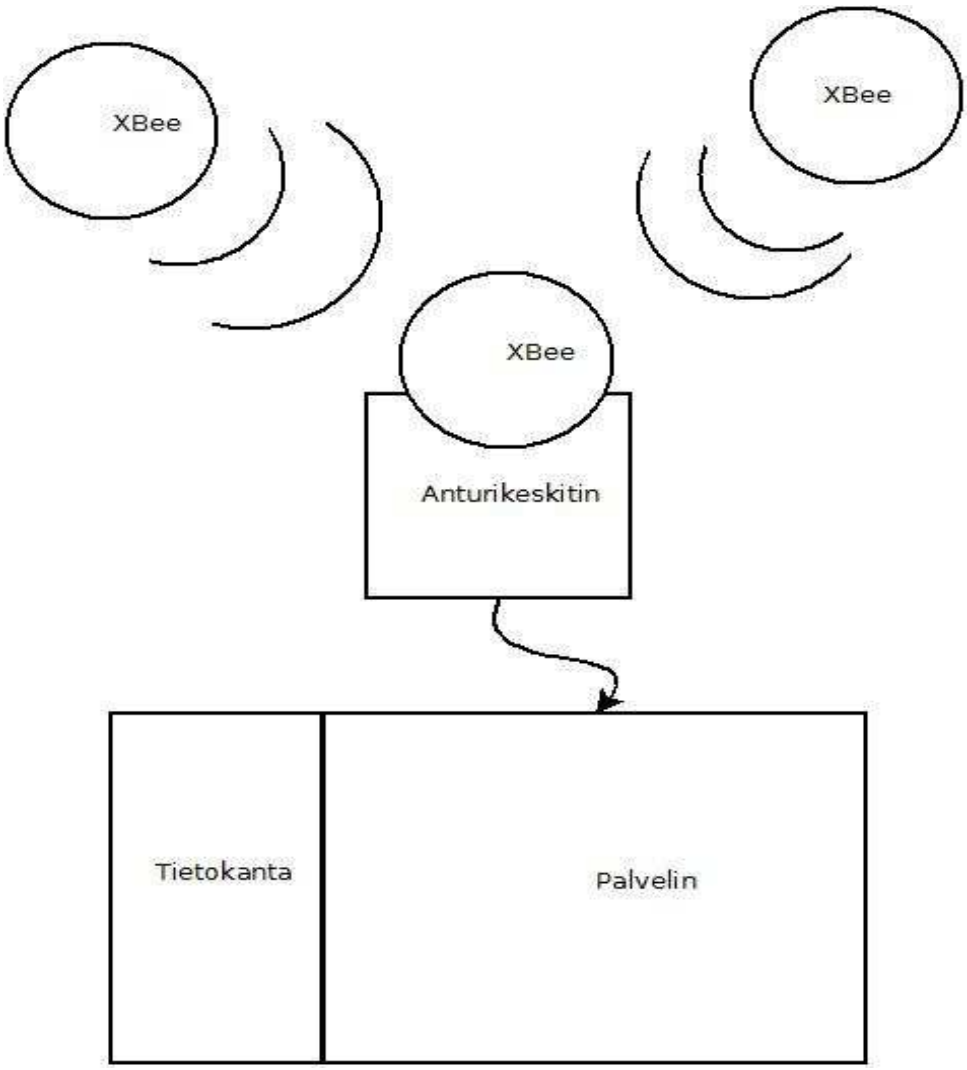
Tässä insinöörityössä tavoitteena oli suunnitella ja toteuttaa Xbee-moduuleille keskitin, jolla voidaan siirtää anturimittauksia tietovarastoon. Sitä varten täytyi käytännössä rakentaa kokonainen järjestelmä sen ympärille. Mahdollisia ratkaisuja oli monia, mutta parhaalta vaihtoehdolta tuntui rakentaa anturikeskitin normaaliin Ethernet-verkkoon sen yhteensopivuuden ja laajan tuen vuoksi. Järjestelmä on lähtökohtaisesti suunniteltu laajennettavaksi ja senkaltaiseksi, että kaikkia sen osia voidaan käyttää erikseen. Kuva 2 (seuraava sivu) esittää järjestelmän rakenteen.

Järjestelmän voi jakaa kolmeen osaan;

- antureihin ja niiden mittauksia eteenpäin lähettäviin XBee-moduuleihin
- anturikeskittimeen
- palvelimeen.

Tässä työssä ei oteta kantaa itse antureihin ja lähettäviin XBee-moduuleihin muuten kuin peruskonfiguraation osalta. Työ keskittyy mittausdatan välittämiseen sen jälkeen kun se on lähtenyt mittaavalta Xbee-moduulilta.

Järjestelmä voidaan jakaa myös kahteen osaan; fyysiseen ja ohjelmalliseen. Fyysinen osa sisältää anturikeskittimen fyysisen toteutuksen, ohjelmallinen osa anturikeskittimen ohjelman, palvelinohjelman sekä tietokannan.



Kuva 2. Järjestelmän rakenne

3.1 Eri osien välinen kommunikointi

Tiedonsiirto antureilta tietokantaan käy läpi eri välivaiheita, kuten tiedon paketointia ja sen siirtämistä yli fyysisen siirtotien. Kommunikointi tapahtuu langatonta siirotietä pitkin XBee-moduulien välillä ja langallisesti Ethernet-verkossa

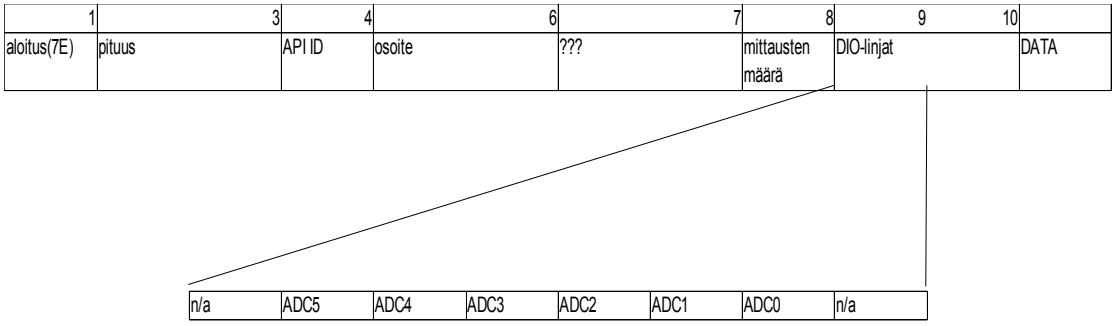
OSI-malli	
7	Sovelluskerros
6	Esitystapakerros
5	Istuntokerros
4	kuljetuskerros
3	verkkokerros
2	siirtokerros
1	fyysinen kerros

Kuva 3. OSI-mallin kerrokset

Kuvassa 3 on esitetty OSI-malli. XBee-moduuli käyttää ZigBee-määritelmän mukaisesti kaikkia tasoja kommunikointiin. Anturikeskittimen ja palvelimen välinen kommunikointi voidaan jakaa siten että Ethernet-piiri ja fyysinen verkko toimii kerroksilla 1 ja 2. Internet-protokolla sijoittuu tasolle 3 ja UDP sijoittuu tasolle 4. ASPv0 sijoittuu tasoille 5-7.

3.2 Kommunikointi XBee-moduulilta anturikeskittimelle

XBee-moduuli, jossa on analogisia antureita kiinni, lähettää analogisesta digitaaliseenmuunnoksen esimääriteltynä kehyksenä anturikeskittimelle. Anturikeskittimen XBee-moduuli vastaanottaa kehyksen, purkaa sen tiedot ja tallettaa tiedot muistiin.



Kuva 4. XBee-moduulin A/D-muunnoksen kehys [5, s. 13; 5, s. 57].

Kehyksen sisältämän API ID:n perusteella määräytyy osoitteen koko, joka tässä implementaatiossa on 16-bittinen. API ID on tässä tapauksessa 83h, joka kertoo että kyseessä on ADC-kehys 16-bittisellä osoitteella. A/D-muunnokseen käytettävien jalkojen nastat löytyvät DIO-linjan merkityksellisemmästä tavusta. Mittausdata rakentuu kuvan 5 mukaisesti. Kuvissa 4 ja 5 esitettyjen tavujen lisäksi kehykseen sisältyy *checksum*-tavu, mutta sen merkitys on vähäinen, sillä XBee-moduulit hallitsevat virheenkorjauksen.

Kuva 5. XBee-moduulin ADC-kehyksen mittausdatan rakenne.

3.3 Kommunikointi anturikeskittimeltä palvelimelle

Anturitiedon välittäminen palvelimelle tapahtuu tavanomaisen Ethernet-verkon yli käyttäen UDP/IP-protokollapinoa. Anturitieto kapseloidaan ASP:n anturidatakehykseen UDP/IP-protokollan alle. Alkuperäisenä suunnitelmana oli käyttää TCP/IP-protokollaa, mutta se hylättiin sen toteuttamisen monimutkaisuuden ja mikrokontrollerin resurssien vähyyden vuoksi. Anturikeskitin ei juuri olisi hyötynyt TCP/IP-protokollan käytöstä, sillä tarvittaessa tietoliikennepakettien varmistamisen voi hoitaa korkeamman tason protokollalla, mikäli sellaista ylipäänsä tarvitsee.

Kun protokollapinoa muodostetaan, lähdetään liikkeelle aina alimman tason protokollasta ja siirrytään seuraavan tason protokollaan, kunnes päästään paketin informaatiosisältöön. Protokollapinon muodostaminen toteutetaan niin, että lähtevään tietoliikennepakettiin kirjoitetaan kunkin käytettävän protokollan otsikko matalimman tason protokollasta ylimmän tason protokollaan. Käytännössä tämä tarkoittaa sitä, että viestin alkuun kirjoitetaan alimman tason otsikko, jota seuraa ylemmän tason otsikko, kunnes päästään käsiksi varsinaiseen hyötydataan.

Kuva 6. Informaatio (DATA) joka on kapseloitu ASP-kehykseen, joka on kapseloitu UDP-kehykseen, joka on kapseloitu IP-kehykseen, joka on kapseloitu ETH-kehykseen.

3.3.1 Ethernet-protokolla

Ethernet-protokollan otsikko määrittelee lähettävän ja vastaanottavan laitteen MAC (Medium Access Control) -osoitteen ja kertoo seuraavan tason protokollan. Tässä tapauksessa seuraavan tason protokolla on IP.

Pelkän Ethernet-protokollan avulla voidaan olla yhteydessä vain suoraan samassa verkossa oleviin laitteisiin. Jokaisella laitteella on uniikki MAC-osoite, jonka perusteella voidaan suodattaa pois ne paketit, joita ei ole tarkoitettu jollekin tietylle laitteelle.

6	12	14
vastaanottava osoite	lähdeosoite	protokolla

Kuva 7. Ethernet-otsikko [6].

3.3.2 Internet-protokolla

Internet-protokolla (IP) on Ethernetistä seuraavan tason protokolla. Se sisältää mahdollisuuden muun muassa reitityksiin ja tietoliikennepakettien pilkkomiseen. Tämän työn kannalta olennaisinta protokollassa on se, että konventiot sen käyttöön ovat vakiintuneita ja sitä on melko yksinkertaista käyttää. Lisäksi IP mahdollistaa laajempien verkkorakenteiden käytön kuin pelkkä Ethernet-protokolla. Kuvassa 8 esitetään Internet-protokollan otsikko.

1	2	4	6	8	9	10
versio/IHL	DS	pituus	id	liput&fragment offset	Time to live	protokolla
12		16		20		
header pituus		lähdeosoite		vastaanottava osoite		

Kuva 8. IP-otsikko [7].

3.3.3 User Datagram Protocol -protokolla

User Datagram Protocol -protokolla (UDP) määrittelee IP:stä seuraavan tason protokollan. UDP toimii yhteydettömänä. Yhteydettömyys tarkoittaa sitä, että jokainen vastaanotettava paketti on itsenäinen eikä liity aiempiin paketteihin muuten kuin mahdollisesti korkeamman tason määrittelyn perusteella. UDP määrittää porttiosoitteen sekä lähtevästä että vastaanottavasta päästä ja pituuden viestille. Lisäksi UDP tarjoaa 2 tavua varmistamaan, että tieto on eheää (*checksum*). *Checksum*-tavun käyttö on UDP-protokollassa vapaaehtoista ja esimerkiksi BSD-*socket* -kirjasto ei oletuksena siitä välitä. Kuvassa 9 esitetään UDP-protokollan otsikko.

2	4	6	8
Lähdeportti	vastaanottava portti	pituus	checksum

Kuva 9. UDP-otsikko [8].

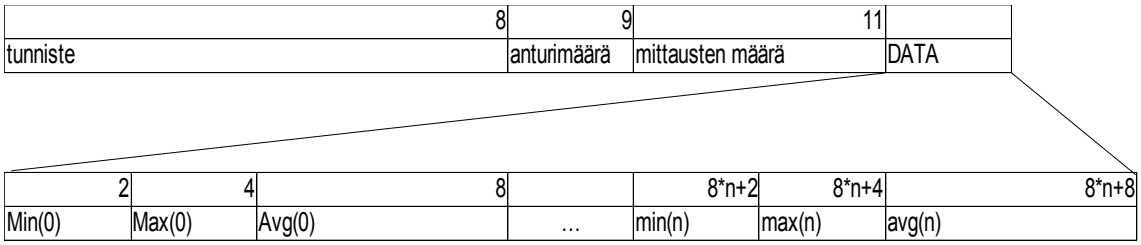
3.3.4 Arbitary sensor protocol versio 0 -protokolla

Arbitary Sensor protocol -protokollan versio 0(ASPv0) määrittelee sen, miten mittaukset välitetään palvelimelle. Otsikko määrittelee tunnisteen moduulille sekä antureiden ja mittausten määrän. Otsikkoa seuraa mittaustieto, kuten kuvassa 10 esitetään. ASPv0 määriteltiin tätä työtä varten mahdollisimman yksinkertaiseksi tavaksi siirtää anturidataa anturikeskittimeltä palvelimelle.

Valmiita protokollia olisi ollut tarjolla, mutta anturikeskittimen muistirajoitteisuuden vuoksi määriteltiin oma minimalistinen protokolla, jotta tietoliikennepuskuri saataisiin pidettyä mahdollisimman pienenä. Vaikka protokolla on hyvin rajoitettu, se antaa myöden esimerkiksi lippujen välittämiseen anturimäärän seassa, sillä se on kokonaisen tavun kokoinen, mutta käyttää siitä vain korkeintaan 3 bittiä (ohjelmassa tuota kutsutaan OPTION-tavuksi).

ASPv0 sisältää myös alkeellisen kättelyn palvelimelle. Kun asiakas haluaa rekisteröityä palvelimelle, palvelimelle lähetetään ASCII-koodattuna "REG". Kun palvelin on rekisteröinyt asiakaan, se vastaa asiakkaalle ASCII-koodattuna "ROK".

Palvelin pyytää asiakkaalta mittaustiedot lähettämälllä asiakkaalle pyynnön (ASCII-koodattu "REQ"). Asiakas vastaa pyyntöön lähettämällä kuvassa 10 esitettävän ASPv0-paketin.



Kuva 10. ASPv0-kehys

3.4 Informaation tallennus

Jotta kerätyistä tiedoista olisi mitään hyötyä, se pitää tallentaa jonkinlaiseen tietorakenteeseen. Tietorakenteella on vaatimuksena kyetä erittelemään data siten, että jokaisen anturin mittaukset näkyvät uniikkeina ja järjestyksessä. Järjestys saadaan helposti numeroimalla sisääntulevat mittaukset. Järjestyksen lisäksi voi olla hyvä tarkistaa myös tallennusaika, jotta mittaukset voidaan sijoittaa ajallisesti. Uniikeiksi anturit saataisiin esimerkiksi asettamalla pitkä merkkijono viittaamaan jokaiseen anturiin jokaisessa mittauksessa. Jokaiseen mittaukseen pitkän uniikin merkkijonon sisällyttäminen saattaa olla sekä kömpelöä, joten hierarkkinen järjestely helpottaa datan jäsentelyä. Yksi lähestymistapa on taltioida mittaukset anturikeskittimen perusteella siten, että jokaisella mittauksella on viite keskittimeen, jolta se on tullut, ja anturin järjestysnumero. Rakenteiden ollessa riittävän monimutkaisia, on hyvä ratkaisu käyttää tietovarastona tietokantaa.

4 Järjestelmän toteutus

4.1 Anturikeskittimen toteutus

Anturikeskitin ottaa vastaan XBee-moduulien lähettämän A/D-muunnoskehyksen, purkaa kyseisen kehyksen ja tallentaa sen sisältämän mittaustiedon muistiin. Lisäksi palvelimen pyytäessä anturikeskitin lähettää varastoidun mittaustiedon palvelimelle.

4.1.1 Anturikeskittimen komponentit

Anturikeskittimessä on kolme olennaista komponenttia; mikrokontrolleri, Ethernet-piiri ja XBee-moduuli. Keskittimeen on sisällytetty myös LM-317-regulaattori, jotta saadaan 3.3 V:n jännite piiriin ilman ulkoista virtalähdettä, joka tarjoaisi kyseisen jännitteen.

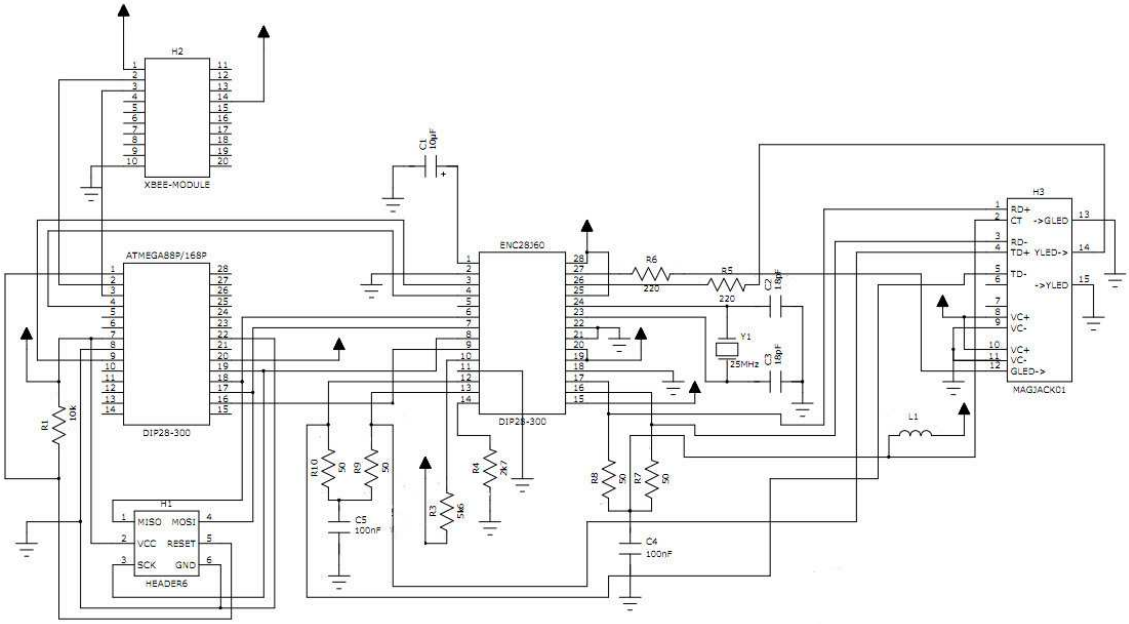
Anturikeskittimen ydin on Atmelin Atmega 168 -mikrokontrolleri. Atmega 168 on valittu tähän yksinkertaisuutensa vuoksi. Yksinkertaisuudestaan huolimatta se sisältää kaiken tarvittavan, jotta laitteen voi toteuttaa; SPI-väylän verkkopiiriä varten ja USART-käyttöliittymän XBee-moduulia varten. Muistia laitteessa on 1 kB, joka riittää tähän toteutukseen. Tarvittaessa piiri on myös pienillä muokkauksilla korvattavissa Atmega 32 -mikrokontrollerilla, jonka muistikapasiteetti on kaksinkertainen verrattuna Atmega 168 -mikrokontrolleriin.

ENC28J60 on käytettävyydeltään yksinkertainen Ethernet-piiri. Kyseisessä piirissä on vain 28 jalkaa, ja sen vuoksi sitä on hyvin yksinkertaista käyttää. Se kommunkoi mikrokontrollerin kanssa SPI-väylällä.

MagJack on RJ-45-pistoke, joka sisältää magnetoinnin ja valodiodit sisäänrakennettuina. Tämän ansiosta se toimii erinomaisesti tässä työssä. Edellä mainittujen lisäksi mikrokontrolleriin on liitettynä XBee-moduuli USART-liitännän kautta, jotta antureilta voitaisiin vastaanottaa langattomasti tietoa.

4.1.2 Anturikeskittimen kytkentä

Anturikeskittimen kytkentä (kuva 11) on melko yksinkertainen, kiitos sen että MagJack-verkkopistoke sisältää koteloituna Ethernet-muuntimen. ISP-liittimen (H1) voi jättää pois, jos mikrokontrolleri esiohjelmoidaan, eikä sen ohjelmaa oleteta päivitettävän.



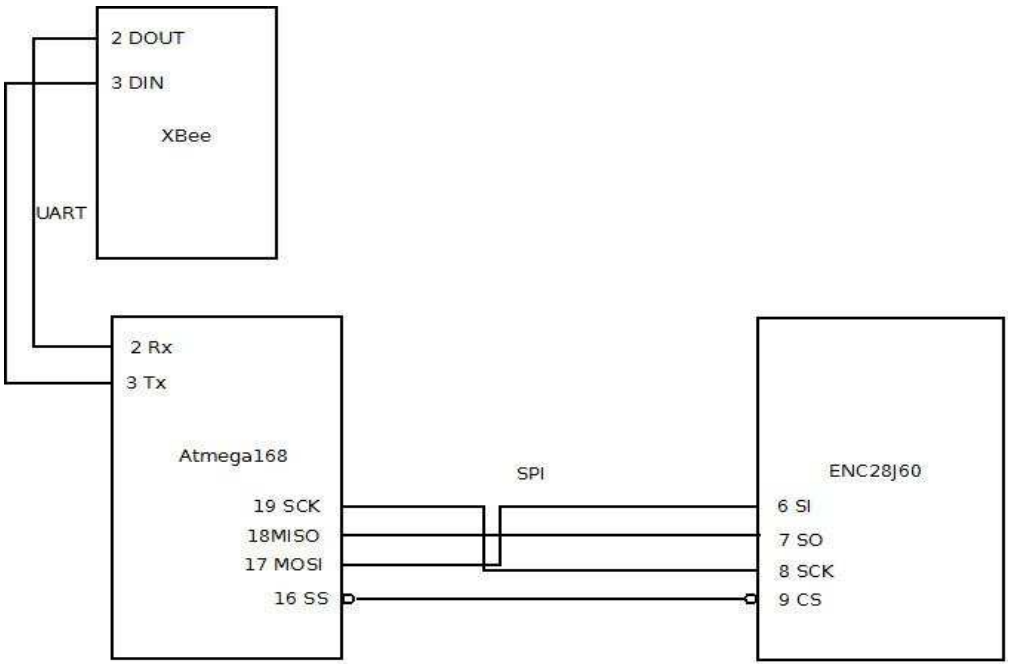
Kuva 11. Anturikeskittimen kytkentä

Vastukset R7-R9 ovat 50Ω:n terminointia varten. Tätä tarkoitusta varten ovat myös 100 nF:n keraamiset kondensaattorit C5 ja C4. Elektrolyyttikondensaattori C1 on ENC28J60-piirin sisäisen regulaattorin filtteröintikondensaattori. L1-induktanssista ei ole piirin datalehdessä sanottu sen enempää, kuin että se tulisi olla mitoitettu ainakin 80 milliampeerille. Tähän käy pieni ferriittirengas, jonka johdin kiertää 7 kertaa ympäri.

Komponentti	**Arvo**	**Yksikkö**	**tyyppi**
R1	10	KΩ	Vastus
R3	5,6	KΩ	Vastus
R4	2,7	KΩ	Vastus
R5-R6	220	Ω	Vastus
R7-R10	50	Ω	Vastus
C1	10	µF	Elektrolyyttikondensaattori
C2-C3	18	pF	Keraaminen kondensaattori
C4-C5	100	nF	Keraaminen kondensaattori
Y1	25	Mhz	Kideoskillaattori

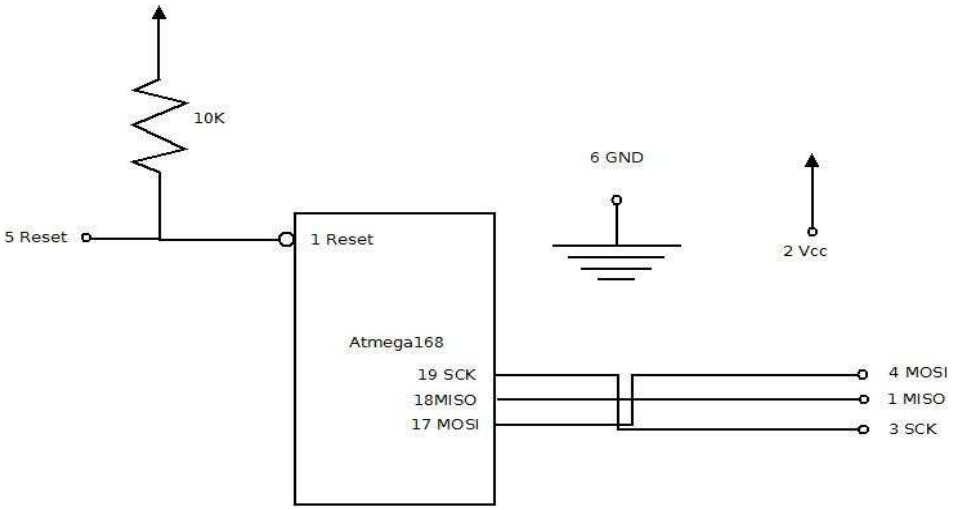
Kuva 12. Kytkennän peruskomponentit

Mikrokontrolleri kytketään Ethernet-piiriin SPI-väylällä ja XBee-moduuliin USART-kytkennällä kuvan 13 mukaisesti. Kuten kuvasta 13 nähdään, USARTin käyttöön tarvitaan vain kaksi johdinta



Kuva 13. Liitännät mikrokontrollerista verkkopiiriin ja XBee-moduuliin

Ohjelmointilaite voidaan kytkeä ISP-liitynnällä piiriin käyttäen osittain samoja nastoja kuin SPI-väylässä. ISP-liitännän esittää kuva 14. Tätä ohjelmointirajapintaa käyttää esimerkiksi AVRispmkII, jota on käytetty tässä työssä mikrokontrollerin ohjelmoimiseen.

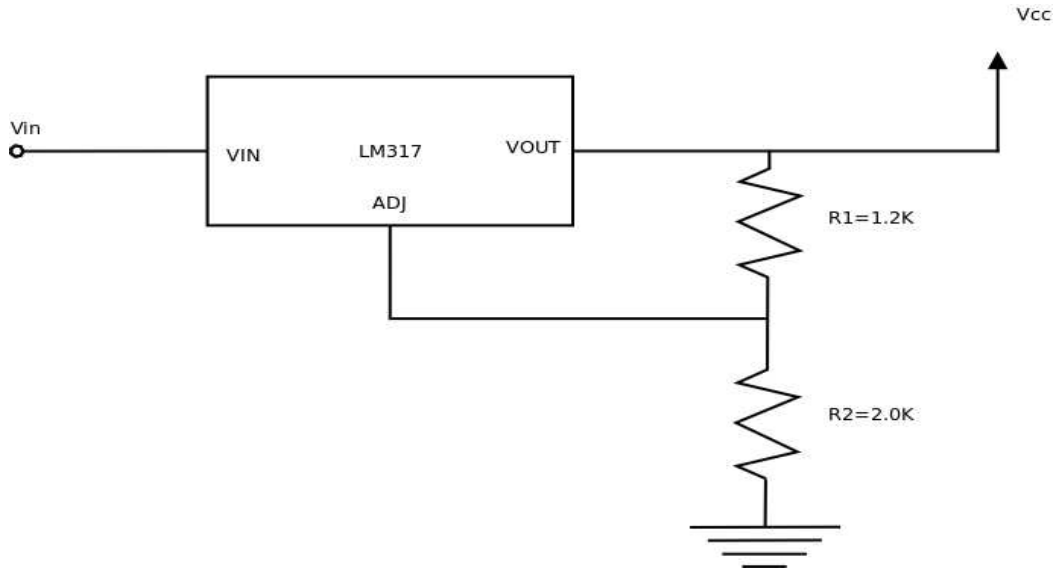


Kuva 14. Liitännät ISP-ohjelmointilaitteelle

Varsinaisen kytkennän lisäksi piirille tarjotaan käyttöjännite regulaattorikytkennällä. Regulaattorikytkennällä saadaan määritettyä käyttöjännite sopivaksi piirille. Kuvan 15 mukaisesti määritellään vastus R1 1.2 kilo-ohmiin ja vastus R2 2.0 kilo-ohmiin. Käyttöjännitteeksi saadaan siis noin 3.3 V seuraavan datalehdessä esitellyn kaavan mukaisesti:

$$Vout = Vref \times (1 + \frac{R2}{R1}) + Iadj \times R2$$

, jossa Vref= 1.25V ja Iadj on häviävän pieni.



Kuva 15. Regulaaattorikytkentä

4.1.3 XBee-moduulien konfiguraatio

Anturikeskittimen XBee-moduuli konfiguroidaan XBee-verkon koordinaattoriksi. Lähettävät XBee-moduulit konfiguroidaan samalle kanavalle ja PAN ID:lle, mutta ne konfiguroidaan päätelaitteiksi tai reitittämään. Lähettävät XBee-moduulit konfiguroidaan myös lähettämään data anturikeskittimen Xbee-moduulille.

Lähettävien moduulien konfiguraatiossa tulee myös ottaa huomioon se, miten monta mittausta lähetetään sen ajan sisään, jonka väleissä palvelin pyytää anturikeskittimeltä tietoja. Mikäli palvelimien pyyntöjen välissä tulee yli 65 535 mittausta keskittimelle, vuotaa keskittimen mittausmäärälle varattu muisti yli ja mittausten määrän laskeminen aloitetaan alusta.

4.2 Anturikeskittimen ohjelma

Anturikeskittimen ohjelma on kirjoitettu C-kielellä käyttäen GNU C Compiler (gcc) -kääntäjää. Kääntäjän lisäksi kääntämiseen tarvitaan avrlibc-kirjasto, jonka perusteella ohjelma voidaan kääntää mikrokontrollerin ymmärtämään binäärimuotoon. Lisäksi käännöksenhallintaan on käytetty GNU make -ohjelmaa ja laitteen ohjelmoimiseen avr-dude-ohjelmaa.

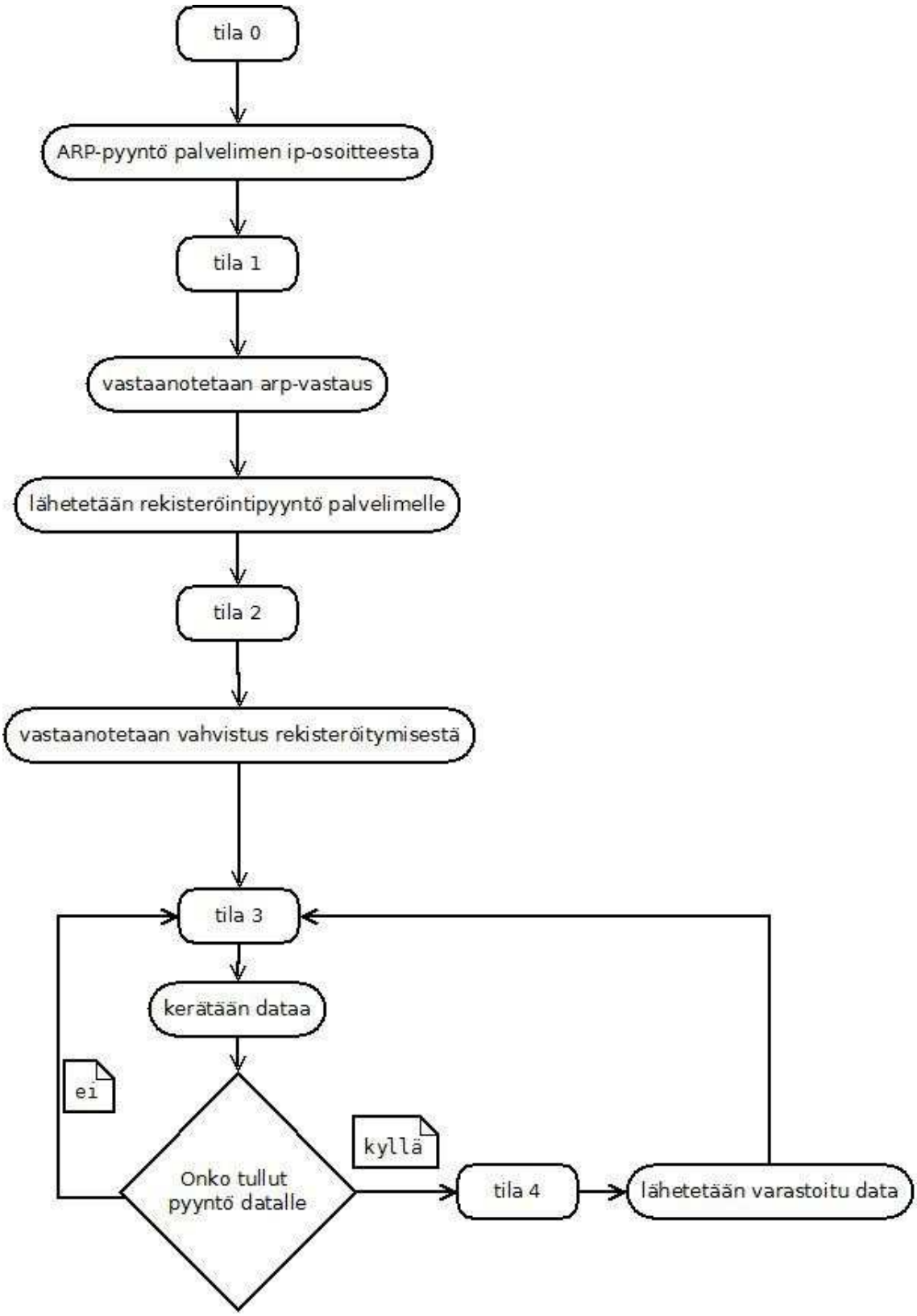
Ohjelma on jaettu eri osiin tehtävien mukaan; pääohjelmaan, muistitoimintoihin, XBee-moduulin toimintoihin ja Ethernet-verkon toimintoihin. Lisäksi ohjelmaan sisältyy AVR-libin U(S)ART- toiminnot, Guido Socherin Ethernet-ajuri ja osa ETH/IP/UDP-pinoa.

Ohjelman pääsilmukka on jaettu eri tiloihin, joista siirrytään eteenpäin kun jokin tilaan sopiva tapahtuma tapahtuu. Ennen siirtymistä pääsilmukkaan alustetaan muisti, UART-yhteys ja verkkorajapinta.

Pääsilmukassa on viisi tilaa, jotka määrittävät, mitä anturikeskitin tekee:

0. Tilassa 0 pyydetään ARP-kutsulla palvelimen MAC-osoitetta esimääritetyn IP-osoitteen perusteella ja siirrytään tilaan 1.
1. Tilassa 1 vastaanotetaan ARP-vastaus, lähetetään rekisteröintipyyntö ja siirrytään tilaan 2.
2. Tilassa 2 odotetaan vahvistusta rekisteröitymisestä ja kun vahvistus palvelimelle rekisteröitymisestä on saatu, siirrytään tilaan 3
3. Tilassa 3 kerätään antureilta mittaustuloksia, kunnes palvelin pyytää mittauksia ja siirrytään tilaan 4.
4. Tilassa 4 lähetetään varastoidut mittaukset ja siirrytään tilaan 3.

Kuvassa 16 (seuraava sivu) esitetään tilojen vuokaavio.



Kuva 16. Pääsilmukan tilojen vuokaavio

Ohjelman udp/ip/eth-verkkoa käyttävä osa perustuu Guido Socherin Atmega-piireille suunniteltuun www-asiakasohjelmaan. Kokonaisuudessaan tämä ohjelmisto sisältää www-palvelimen ja asiakasohjelman, joka käyttää hyväkseen ENC28J60-piiriä. Ohjelmakoodia on muokattu tarpeeseen sopivaksi.

Ohjelma sisältää ohjelmamuistiin kirjoitettavan templaatin Internet protokollan otsikosta sekä ARP-pyynnöstä. Templaatit täytetään muuttujien, kuten palvelimen IP-osoitteen, kanssa muodostamaan tietoliikennepaketti.

Ohjelmallisesti USART-yhteys XBee-moduuliin toteutetaan avrlib-kirjaston avulla. Tämä tapahtuu niin, että luetaan usart-porttia niin kauan että vastaanotetaan XBee-moduulin ADC-kehyksen ensimmäinen tavu. Kun kehyksen ensimmäinen tavu (7E**h**) on luettu, suoritettaan Xbee_store_datapacket-funktio, joka lukee ja tallettaa ADC-kehyksen muistiin ja luo tarvittaessa uuden moduulin datakehyksen muistiin.

4.2.1 XBee-kehyksen purkaminen

Kun tietoliikenne-puskurista on luettu XBee-kehyksen aloitustavu 7E**h** siirrytään purkamaan sen informaatiota. Seuraavat 2 tavua kertovat kehyksen pituuden ilman aloitustavua, kokotavuja ja *checksum*-tavua. Tämän jälkeen kehys luetaan tietoliikenne-puskuriin koon perusteella.

Koon jälkeen kehyksestä luetaan moduuliin kiinnitettyjen anturien määrä (*node count*) sen perusteella, kuinka monta bittiä DIO-linjan merkitsevämmässä tavussa on arvolla 1. Ohjelma käy läpi jokaisen A/D-muunnoksesta kertovan DIO-linjan bitin yksitellen läpi.

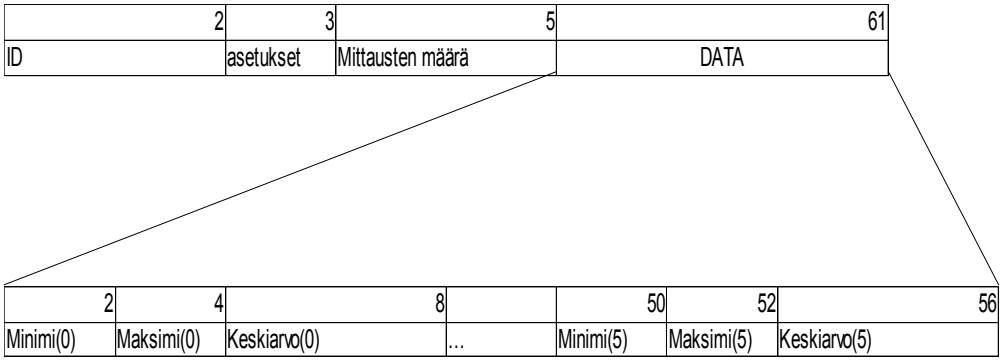
Kun kehys on luettu, tarkistetaan onko kyseessä olevaa moduulia jo muistissa. Mikäli moduuli on uusi, se alustetaan ja merkitään muistiin tallennettujen moduulien määrään lisäys. Tarkistus tehdään XBee-moduulin osoitteen perusteella.

Kun moduulin muistipaikka ja anturien määrä on tiedossa, voidaan anturi-informaation lukeminen aloittaa. Jokaisen anturin mittaustulokset luetaan erikseen ja talletetaan muistiin siten, että puskurista luetaan getValues()-funktiolla mittausten määrä, minimi-

ja maksimiarvot sekä mittaustulosten keskiarvo. Tämän jälkeen muistiin tallennetaan arvot siten, että minimi- ja maksimiarvoja verrataan edellisiin minimi- ja maksimiarvoihin ja tallennetaan, mikäli niissä on tapahtunut muutoksia. Keskiarvo määritellään aiemman ja uuden keskiarvon mittausmäärien perusteella kahden keskiarvon keskiarvona. Lopuksi päivitetään mittaustulosten määrä muistiin.

4.2.2 Mikrokontrolleriohjelman muisti

Anturitieto varastoidaan mikrokontrollerin muistiin lineaariselle alueelle, jossa jokaisen talletettavan moduulin datalla on määrätty koko. Tämä tarkoittaa sitä, että jos yksikin moduuli lähettää 6:n anturin tiedot, vaaditaan joka moduulin muistiavaruuteen 6:n anturin tiedot, vaikka niihin ei mitään tallennettaisikaan. Etuna tässä lähestymistavassa on sen yksinkertaisuus ja nopeus. Jokainen moduuli käyttää 56 tavua muistia dataan. Lisäksi varataan yksi tavu muistia pitämään kirjaa talletettujen moduulien määrästä. Kuva 17 esittää muistin rakenteen yhdelle moduulille. Lisäksi datamuisti sisältää yhden tavun, jolla pidetään kirjaa tallennettujen moduulien määrästä.



Kuva 17. Anturikeskittimen muistirakenne

Muisti varataan esimääritetylle määrälle moduuleita. Kun uusi moduuli luodaan, alustetaan uuden moduulin muistialue siten, että moduulin 2 tavuinen osoite(ID) varastoidaan ja lopulle muistialueelle kirjoitetaan esimääritetyt tiedot. Esimääritetyt tiedot ovat pääasiassa arvolla 0, mutta tähän tallennetaan anturien määrä ja minimiarvo asetetaan suurimpaan mahdolliseen arvoon.

4.3 Palvelinohjelman toteutus

Palvelin toimii koko järjestelmän selkärankana ja hoitaa anturikeskittimeltä tulevan yhteyden, ASPv0-kehyksen purkamisen ja mittausdatan tallentamisen tietokantaan. Tietokantana käytetään MySQL-tietokantaa MySQL++-rajapinnan kautta. CDatabase-luokan metodeja ja muuttujia muokkaamalla palvelin voi teoriassa käyttää mitä tahansa tietokantapalvelinta, kunhan sille löytyy ohjelmointirajapinta. Palvelinohjelmisto on myös BSD soket-kirjastoa lukuunottamatta käyttöjärjestelmäriippumaton (nyt se toimii vain ympäristöissä, joissa on tuki BSD-socket kirjastolle). Ohjelman toiminta voidaan jakaa seuraaviin osiin: Tietoliikenneyhteyksien kuuntelu, tietojen prosessointi ja tietojen tallennus. Palvelinohjelmisto on kirjoitettu C++-kielellä käyttäen Kdevelop-ohjelmointiympäristöä. Käännöksenhallintaan käytetään Cmake-ohjelmaa. Ohjelman voi myös kääntää suoraan komentroiviltä.

Palvelinohjelman ydin on Cserver-olio, joka on yleinen viitekehys palvelimen kaikelle toiminnalle. CServer-olio sisältää oliot yhteyksien ja tietokantojen käsittelyyn (CClientConnection, CClientDeviceConnection, CDatabase). Lisäksi ohjelmistoon kuuluu CdeviceDataSet-olio, jota käytetään datan kuljettamiseen CClientDeviceConnection ja CDatabase -olioiden välillä. CDeviceDataSet sisältää kaiken tarvittavan tiedon anturidatan tallentamiseksi tietokantaan. CClientConnection on yleiskäyttöinen olio hallitsemaan UDP/IP-yhteyksiä. CClientDeviceConnection laajentaa CClientConnection-oliota anturikeskittimen käsittelyä varten.

Kun palvelinohjelma vastaanottaa viestin verkosta, tarkistetaan ensimmäisenä onko se tullut osoitteesta, joka on jo määritelty jollekin yhteysoliolle. Mikäli vastaanotettava viesti on tullut olemassaolevalta yhteydeltä, tallennetaan se vastaavan CClientConnection- tai CClientDeviceConnection-olion viestipuskuriin. Mikäli kyseessä on uusi yhteys, tarkistetaan onko se ASPv0-rekisteröintipyyntö. Kun kyseessä on ASPv0-rekisteröinti luodaan uusi CClientDeviceConnection-olio. Mikäli uudesta osoitteesta tuleva viesti on jotain muuta kuin ASPv0-rekisteröintipyyntö, luodaan CClientConnection-tyyppinen olio.

4.3.1 Palvelinohjelman säikeet

Palvelinohjelma on säikeistetty, jotta ohjelman eri toiminnallisuudet eivät joutuisi odottamaan toistensa suoritusta. Ohjelman säikeet on jaettu ohjelman käynnistävän prosessin lisäksi tietoliikenneviestien vastaanotto-, tiedon prosessointi- ja tietokantasäikeisiin. Esimerkiksi, jos palvelinohjelma menettää yhteyden tietokantaan, ei anturidatan vastaanottamista joudu odottamaan.

Oikeassa toteutuksessa pitäisi ne tietovarastot, joihin kirjoitetaan, lukita, jotta samaan paikkaan ei kirjoitettaisi montaa säiettä yhtä aikaa. Tässä yksinkertaisessa ja demonstratiivisessa toteutuksessa lukot on jätetty pois, joten teoriassa palvelinohjelma voi kaatua tähän tai sekoittaa ohjelman toimintaa. Käytännössä se ei kuitenkaan ole kovin todennäköistä yksinkertaisessa ympäristössä.

4.3.2 Palvelinohjelman oliot

CServer-olio toimii palvelinohjelman ohjelmarunkona. Säikeet ovat tämän olion metodeja. CServer-olio sisältää myös tietorakenteet muiden olioiden säilömiseen.

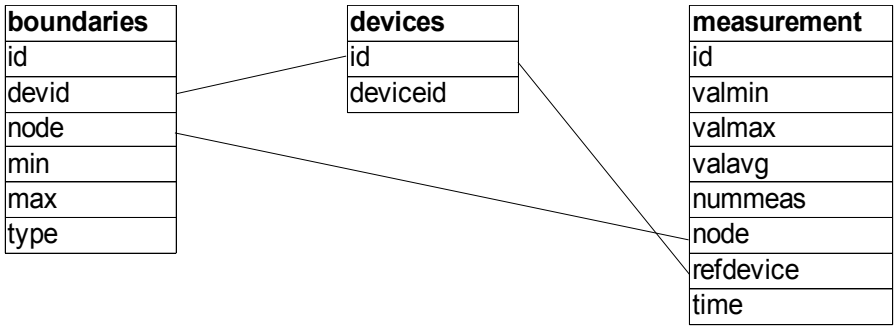
CClientConnection sisältää tarvittavat tietorakenteet ja metodit tietoliikennemäärityksiin. Lisäksi CClientConnection sisältää getRegister-metodin joka kertoo paketista, onko se rekisteröintipyyntö ja jos on, niin minkälainen (Config, Device). CclientDeviceConnection-olio laajentaa CClientConnection-oliota anturidatan tarpeiden mukaan sekä anturikeskittimen kanssa kommunikointiin.

CDatabase-olio hoitaa tietokantaan tallentamisen. Tämä on toteutettu siten, että ensin tarkistetaan onko tietylle moduulille olemassa tietuetta *devices*-taulussa. Mikäli ei ole, luodaan se sille. Kun tietue on varmistettu, tallennetaan mittausdata *measurement*-tauluun.

Tiedon liikuttamiseksi tietokantaoliolle järkevästi on ohjelmassa määritetty CdeviceDataSet-olio, joka sisältää anturidatan. CDeviceDataSet-olio liikuttaa anturidataa CClientDeviceConnection- ja Cdatabase-olioiden välillä.

4.4 Tietokannan rakenne

Tietokanta koostuu kolmesta taulusta; laitetaulusta (*device*), rajataulusta (*boundaries*) ja mittaustaulusta (*measurement*). Laitetaulu pitää sisällään lähettävien XBee-moduulien tunnisteet yhdistettynä laitetunnisteeseen. Rajataulu sisältää ylä- ja alarajan todellisen maailman arvoille. Rajataulu on siis tarpeellinen vain ja ainoastaan mittaustulosten lukemiseen. Lisäksi rajataulu sisältää viitteen siitä mille anturille se kuuluu. Mittaustaulu sisältää kaikki mittaukset jotka on vastaanotettu tietokantaan. Mittaustaulu sisältää maksimiarvon, minimiarvon ja keskiarvon sekä tiedon mittausten määrästä. Mittaustaulu sisältää myös viitteen XBee-moduuliin sekä XBee-moduulissa kiinni olevaan anturiin. Kuva 18 esittää tietokantojen väliset suhteet



Kuva 18. Tietokannan taulut ja niiden suhteet

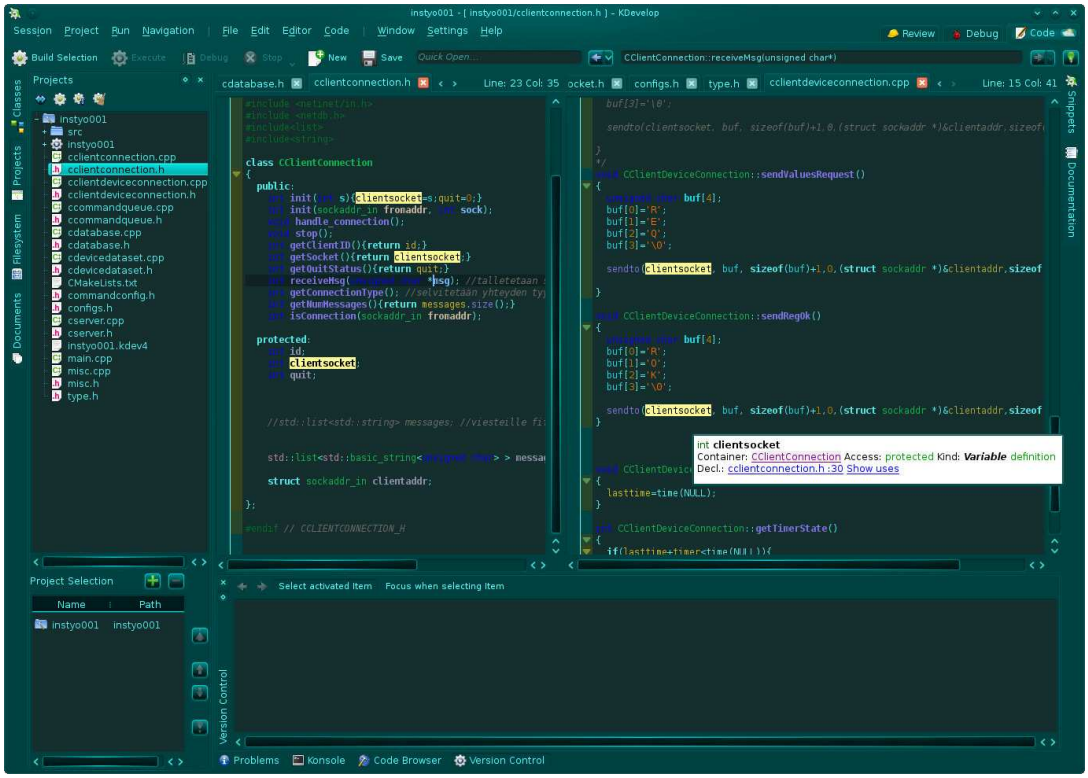
Palvelinohjelmistosta tallennettu data on raakaa, eli se tallennetaan arvoina 0-1023. Jotta tämä raaka data saadaan järkevään muotoon, tarvitaan määrittelijä, joka muuntaa arvot todellisen maailman arvoiksi. Tätä varten tietokannassa on *boundaries*-taulu, jonka rajat ovat todellisen maailman vastineita arvoille 0 ja 1023.

$$Todellinen\,arvo = Alempi\,raja + Mitattu\,arvo \times \left(\frac{Ylempi\,raja - Alempi\,raja}{1023}\right)$$

5 Järjestelmän testaus ja käytetyt työkalut

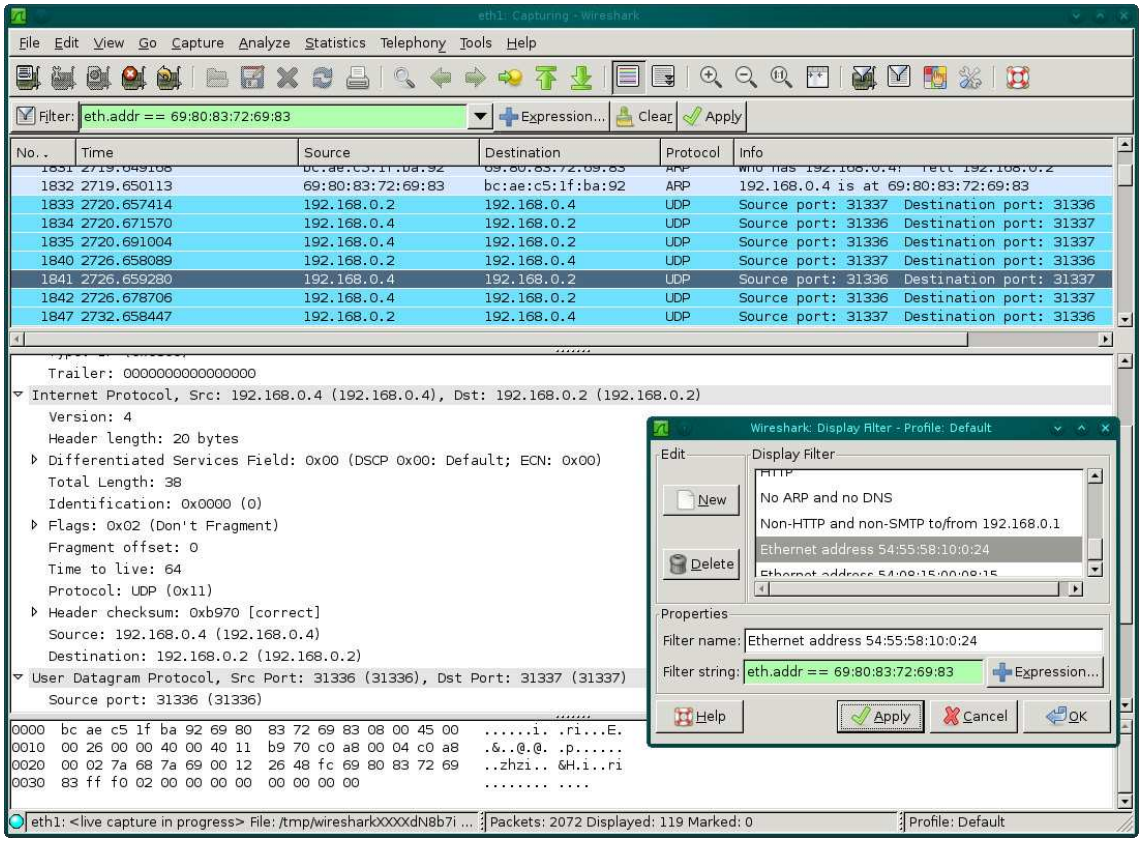
Testaus toteutettiin käytännössä samalla kun ohjelmia on rakennettiin. Palvelinohjelma on kirjoitettu Kdevelop4-kehitysympäristöllä ja mikrokontrolleriohjelma Kate-tekstieditorilla. Testaus on hoidettu Wireshark ja MySQL Query Browser-ohjelmalla. MySQL Query Browser on käynnistetty MySQL Administrator -ohjelmalla, jolla on luotu myös tietokanta ja sen taulut. XBee-moduulit on konfiguroitu Digin X-CTU-ohjelmalla. Kommunikointi tietokoneen ja XBee-moduulien välillä on hoidettu USB-adapterilla. Anturikeskittimen ohjelmointi on tehty avrdude-ohjelmalla.

Kdevelop4 on täysimittainen kehitysympäristö, joka soveltuu etenkin C++-kielen ohjelmointiin. Se sisältää projektinhallintaan liittyviä ominaisuuksia, kuten luokkageneraattorin, jaetut editointi-ikkunat, automaattisen täydennyksen ja se kertoo lennossa esimerkiksi mitä parametreja johonkin metodiin tarvitsee, missä metodi on määritelty (ja kaiken lisäksi linkin sen tiedoston siihen kohtaan, jossa määritys on) ja niin edelleen. Editorina voi käyttää useampaa vaihtoehtoa, mutta tässä tapauksessa käytössä oli *Kate*.



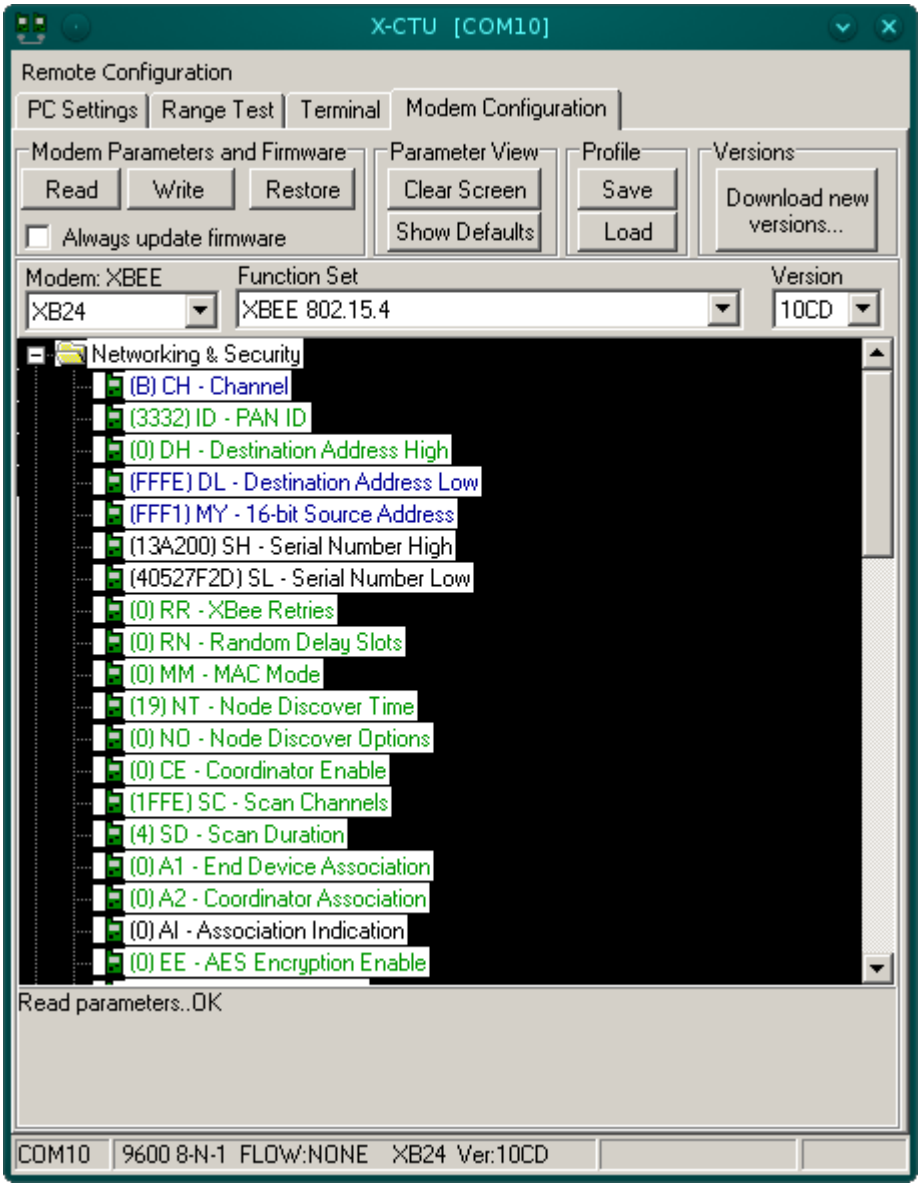
Kuva 19. KDevelop4

Wireshark on monipuolinen ohjelma tietoliikennepakettien tarkkailuun. Wireshark yhdistetään johonkin verkkolaitteeseen, jonka jälkeen Wireshark alkaa kerätä tietoliikennepaketteja kyseisestä laitteesta. Wiresharkissa on myös filtteröinti, jonka voi määrittää esimerkiksi näyttämään pelkästään paketit, jotka ovat tulleet tai ovat menossa tiettyyn IP- tai MAC-osoitteeseen. Tämän kaltaisessa työssä Wiresharkin kaltainen työkalu on lähes välttämätön, sillä kehityksessä on ollut tärkeää seurata tietoliikennepaketteja ja niiden sisältöä. Wireshark esittää valitun paketin sisällön sekä heksana, että eri osiin (kuten Ethernet- tai ip-kerrosten otsikoiden) jaoteltuna.



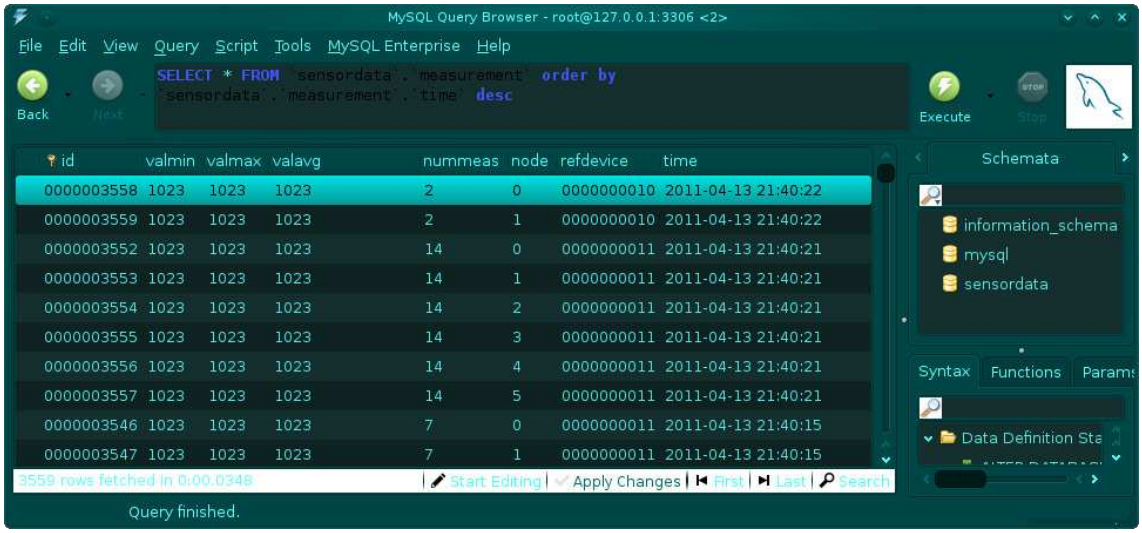
Kuva 20. Wireshark

X-CTU on Digi Internationalin oma hallintaohjelma XBee-moduuleille. X-CTU:n ominaisuuksiin kuuluu konfiguroinnin ja firmwaren-päivittämisen lisäksi terminaali, jolla voi lähettää ja vastaanottaa dataa XBee-moduulien välillä. X-CTU on Windows-ohjelma, mutta se toimii suhteellisen vaivattomasti myös GNU/Linux-ympäristössä WINE (Wine Is Not an Emulator) -ohjelmiston kanssa. Ainoa erityinen säätö, jota X-CTU:ta käytettäessä tuli tehdä, oli tehdä linkit kommunikaatioporteille (esimerkiksi ~/.wine/dosdevices/COM11 → /dev/ttyUSB0) ja lisätä ne manuaalisesti ohjelman sisällä. Lieviä piirrosvirheitä ohjelmassa toki on WINEn kautta ajettuna, kuten kuvasta 20 nähdään.



Kuva 21. X-CTU-ohjelma WINEn kautta ajettuna

MySQL Query Browser -ohjelmaa on käytetty lopullisen mittausdatan tarkkailuun. MySQL Query Browser -ohjelma on ajettu MySQL Administrator -ohjelman kautta, koska sillä voi näppärästi selata tietokantoja ja tauluja.



Kuva 22. MySQL Query Browser näyttämässä mittaustaulun sisältöä

Avrdude-ohjelma on komentoriviltä ajettava ohjelma AVR-mikrokontrollereiden ohjelmointiin. Se tukee montaa eri mikrokontrolleria ja ohjelmointilaitetta. AVRispmkII-ohjelmointi Atmega168-mikrokontrollerille tapahtuu seuraavasti: *avrdude -p m168 -P usb -c avrisp2 -U flash:w:anturikeskitin.hex*, jossa anturikeskitin.hex on käännetty ohjelma. Mikäli oikeuksia USB-porteille ei ole erikseen asetettu, on ohjelma ajettava root-oikeuksin.

6 Järjestelmän jatkokehitys ja oikea toteutus

Järjestelmä sellaisenaan toimii konseptina siitä sille, miten ZigBee rajapintaa hyödyntävää XBee-moduulia voidaan käyttää tiedon varastointiin. Sen voisi myös toteuttaa ilman anturikeskitintä, liittämällä koordinaattorina toimiva XBee-moduuli suoraan tietojärjestelmään esimerkiksi USB-kaapelilla. Etuna anturikeskittimessä on se, että se voidaan sijoittaa mihin tahansa, mistä se pääsee tavalliseen Ethernet-verkkoon. Tämän ansiosta se on käytännöllinen ratkaisu kun verkkoon pääsy on helppoa. Esimerkiksi taloautomaatioon tämä voisi olla erityisen hyvä ratkaisu.

Järjestelmä on kuitenkin konseptitasoinen ja todelliseen implementaatioon tarvitaan huomattavasti kehitystä. Anturikeskittimen ohjelman tulisi lukea asetukset, kuten MAC-osoite, oma IP-osoite, palvelimen IP-osoite sekä XBee-moduulin asetukset esimerkiksi EEPROM-muistista ja optimaalisesti sitä tulisi voida konfiguroida joko suoraan tai palvelimen kautta. Lisäksi tässä kehitysvaiheessa koko verkon pitää olla samassa fyysisessä verkossa. Tämä ongelma voidaan ohittaa määrittelemällä anturikeskittimelle yhdyskäytävän IP-osoite, josta tehdään ARP-pyyntö palvelimen IP-osoitteen sijaan.

Palvelimeen tulisi kirjoittaa CClientConnection-oliota laajentava olio asiakasohjelmaa varten, jotta tietoa voitaisiin muun muassa tarkkailla ja tehdä sääntöjä tietokannan boundaries-tauluun. Asiakasohjelmalaajennuksissa tulisi huomioida autentikointi. Palvelinohjelman pitäisi sisältää myös tapa lukea asetukset esimerkiksi tiedostosta, jotta asetuksia muuttaakseen ei tarvitsisi kääntää ohjelmaa uudestaan.

Anturikeskittimen ohjelmaa voisi jatkaa esimerkiksi tietyillä raja-arvoilla jonka ylitys tai alitus lähettää hälytyspaketin palvelimelle, joka vuorostaan jatkaisi sen asiakasohjelmiston yhteydelle. Anturikeskittimeen voisi myös lisätä I/O-linjoja joilla voisi esimerkiksi säätää venttiileitä. Toisaalta voisi myös ajatella asiakaslaitteita jotka voisivat ottaa vastaan käskyjä tai ohjeita anturikeskittimeltä. Lisäksi mahdolliset ylivuodot ja virheet olisi hyvä etsiä ja paikata.

Anturikeskittimen ja palvelimen väliseen kommunikointiin olisi hyvä lisätä varmistuksia. Esimerkiksi varmistus siitä, että data on tullut eheästi palvelimelle.

Todellisessa toteutuksessa, toteutustavasta riippuen, Atmega168-mikrokontrolleri kannattaa korvata sellaisella alustalla, jossa on enemmän muistia. Vaihtoehtona mikrokontrollerin vaihtamiselle voisi olla hankkia ulkoista muistia, jolloin muistintarve ratkeaa.

7 Yhteenveto

Tässä insinöörityössä oli tavoitteena suunnitella ja toteuttaa järjestelmä langattoman anturitiedon keräämiseen ja tallennukseen XBee-moduuleita hyväksikäyttäen. Prosessi toteutettiin erillisen anturikeskitinlaitteen avulla. Anturikeskitin sijoittuu järjestelmässä niiden XBee-moduulien, jotka lähettävät mittausdataa, ja palvelimen väliin.

XBee-moduuli käyttää hyväkseen ZigBee-määritelmää, joka rakentuu IEEE 802.15.4-standardin siirtotien päälle. XBee sisältää ZigBee-määritelmän lisäksi käyttöliittymät UART-tiedonsiirtolinjalle, hallittavuuden AT-kommennoin ja itsenäisen analogisesta digitaaliseen -muunnoksen.

Suunnitelman lähtökohtana oli käyttää XBee-moduulin analogisesta digitaaliseen -muunnosta ja moduulin sisäänrakennettua kehystä muunnoksen välittämiseen moduulien välillä. Tämän jälkeen kyseinen data oli purettava, käsiteltävä ja välitettävä järjestelmän seuraavalle osalle, eli tallennusrakenteeseen. Tätä tehtävää varten suunniteltiin anturikeskitin, joka kerää useammalta XBee-moduulilta mittauksia, säilöö ne ja lähettää pyydettäessä Ethernet-verkon yli palvelimelle. Ethernet-verkon protokollaksi valittiin UDP/IP sen yksinkertaisuuden vuoksi ja sen päälle määriteltiin oma protokolla mittaustietojen siirtämistä varten.

Anturikeskitin toteutettiin Atmega168-mikrokontrollerilla, johon liitettiin ENC28J60-verkkopiiri ja XBee-moduuli. Toteutusta varten piti myös kirjoittaa mikrokontrolleriohjelmisto joka purkaa XBee-moduulin ADC-kehyksen, säilöö sen muistiin ja lähettää palvelimelle. Lisäksi ohjelmiston täytyi keskustella yhteys palvelimelle.

Palvelinohjelmisto toteutettiin POSIX-yhteensopivassa ympäristössä. Palvelimen tehtäväksi jäi keskustella yhteys anturikeskittimelle ja tiedon tallentaminen tietokantaan, joka tässä tapauksessa oli MySQL-tietokanta.

Kaiken kaikkiaan työn toteutus onnistui hyvin, joskin joistain alun perin suunnitelluista ominaisuuksista, kuten tiedonsiirron varmistuksesta, täytyi luopua. Järjestelmän kunnollinen toteuttaminen olisi vaatinut huomattavasti enemmän aikaa ja resursseja, sekä merkittävän määrän testausta. Rajoittuneisuudestaan huolimatta järjestelmä osoittautui nykyiselläänkin tarkoituksenmukaiseksi, demonstratiiviseksi toteutukseksi. Järjestelmä kerää dataa usealta moduulilta, säilöö datan anturikeskittimeen ja tallentaa datan palvelimelle, kuten oli tarkoituskin.

Lähteet

1. IEEE 802.15 working group. 2006. Standardi. IEEE Standard 802.15.4TM-2006

2. Ruuska, Jukka. 2009. Insinöörityö. ZigBee-langaton tiedonsiirtoympäristö.

3.ZigBee Specification Overview. Verkkodokumentti. <http://www.zigbee.org/Specifications/ZigBee/Overview.aspx>. Luettu 12.4.2011

4. 802.15.4 vs ZigBee. Verkkodokumentti. <http://sensor-networks.org/index.php?page=0823123150>. Luettu 12.4.2011

5. Digi International. 2009. Ohjekirja. Xbee®/XBee-PRO® RF Modules

6. IEEE 802.3, Ethernet. 2011. Verkkodokumentti.

<http://www.networksorcery.com/enp/protocol/IEEE8023.htm>. Luettu 12.4.2011

7. IP, Internet Protocol. 2011. Verkkodokumentti.

<http://www.networksorcery.com/enp/protocol/ip.htm>. Luettu 12.4.2011

8. UDP, User Datagram Protocol. 2011. Verkkodokumentti.

<http://www.networksorcery.com/enp/protocol/udp.htm>. Luettu 12.4.2011

Liite 1. Anturikeskittimen ohjelma

Anturikeskittimen ohjelman Makefile

```
MCU=atmega168
CC=avr-gcc
OBJCOPY=avr-objcopy
# optimize for size:
CFLAGS=-g -mmcu=$(MCU) -Wall -Wstrict-prototypes -Os -mcall-prologues
#------------------
all: anturikeskitin.hex
#------------------
help:
                        @echo "Usage: make all|anturikeskitin"
                        @echo "or"
                        @echo "Usage: make clean"
#------------------
anturikeskitin.hex : anturikeskitin.out
                        $(OBJCOPY) -R .eeprom -O ihex anturikeskitin.out anturikeskitin.hex
                        avr-size anturikeskitin.out

anturikeskitin.out : main.o ip_arp_udp.o enc28j60.o timeout.o xbee.o uart.o rprintf.o buffer.o mem.o xbee.o
                        $(CC) $(CFLAGS) -o anturikeskitin.out -Wl,-Map,eth_rem_dev.map main.o ip_arp_udp.o enc28j60.o timeout.o xbee.o uart.o rprintf.o
buffer.o mem.o
enc28j60.o : enc28j60.c avr_compat.h timeout.h enc28j60.h
                        $(CC) $(CFLAGS) -Os -c enc28j60.c
ip_arp_udp.o : ip_arp_udp.c net.h avr_compat.h enc28j60.h mem.h mem.c
                        $(CC) $(CFLAGS) -Os -c ip_arp_udp.c
main.o : main.c ip_arp_udp.h avr_compat.h enc28j60.h timeout.h net.h xbee.h uart.h rprintf.h mem.h
                        $(CC) $(CFLAGS) -Os -c main.c
timeout.o : timeout.c timeout.h
                        $(CC) $(CFLAGS) -Os -c timeout.c
rprintf.o : rprintf.c rprintf.h global.h uart.h buffer.h
                        $(CC) $(CFLAGS) -Os -c rprintf.c
uart.o : uart.c uart.h global.h buffer.h
                        $(CC) $(CFLAGS) -Os -c uart.c
xbee.o : xbee.c xbee.h uart.h rprintf.h mem.h
                        $(CC) $(CFLAGS) -Os -c xbee.c
buffer.o : buffer.c buffer.h global.h
                        $(CC) $(CFLAGS) -Os -c buffer.c
mem.o  : mem.c mem.h
                        $(CC) $(CFLAGS) -Os -c mem.c

#------------------
clean:
                        rm -f *.o *.map *.out anturikeskitin.hex
#------------------
```

pääohjelma

main.c

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Timo-Tapio Palmu
 * Based on the work of Guido Socher
 * Copyright: GPL V2
 *
 * Ethernet remote device and and XBee-module
 *
 * Chip type           : ATMEGA168 with ENC28J60
 *********************************************/
#include <avr/io.h>
#include <stdlib.h>

#include "global.h"

#include <util/delay.h>
#include "ip_arp_udp.h"
#include "enc28j60.h"
#include "timeout.h"
#include "avr_compat.h"
#include "net.h"

#include "xbee.h"
#include "mem.h"

//AVRlib
#include "uart.h"
#include "rprintf.h"
```

```
//alustavat asetukset pitäisi oikeasti lukea vaikka eeprom:sta
static uint8_t mymac[6] = {0x69,0x80,0x83,0x72,0x69,0x83}; //oma MAC
static uint8_t myip[4] = {192,168,0,4}; //oma IP-osoite
static uint16_t myport =31336; // oma UDP-portti
static uint8_t serverip[4]={192,168,0,2}; //palvelimen osoite
static uint16_t serverport=31337; //palvelimen UDP-portti

//tietoliikennepuskurin koko
#define BUFFER_SIZE 150

int main(void){


	uint16_t plen; //tietoliikennepaketin koko
	uint8_t i=0; //apumuuttuja silmukoihin
	uint8_t opSTATE; //toimintatila
	uint16_t xbeelen;//xbee-paketin koko



	//puskurit
	uint8_t *mem;
	uint8_t *buf;

		mem=malloc(1+(MAX_MODULES+1)*MEM_MODULE_SIZE);//varataan muisti moduuleille + mem[0]:
tallennettujen moduulien määrä
	buf=malloc(BUFFER_SIZE+1);//varataan muisti tietoliikennepuskurille


	CLKPR=(1<<CLKPCE); //asetetaan prescaler kellolle
	CLKPR=0; // 8 MHZ
	delay_ms(10);


	DDRD&= ~(1<<DDD2);









	/*alustetaan enc28j60*/
	enc28j60Init(mymac);
	enc28j60clkout(2);
	delay_ms(20);

	enc28j60PhyWrite(PHLCON,0x476); //määritetään Verkkopistokkeen ledit: A: link status, B:lähetys/vastaanotto
	delay_ms(20);



	init_ip_arp_udp(mymac,myip);//alustetaan verkkotoiminnot



	set_server_ip(serverip); //asetetaan palvelimen ip-osoite


	DDRD|= (1<<DDD7); //d-portin 7 jalka ulos debuggausta varten


	uartInit();//alustetaan avrlib-uart -järjestelmä
	uartSetBaudRate(9600); //asetetaan nopeus uartille
	rprintfInit(uartSendByte);
	delay_ms(200);
```

```
clearmem(mem,1+(MAX_MODULES+1)*MEM_MODULE_SIZE); //tyhjennetään datamuisti

opSTATE=0; //toimintatila 0: alustus: tehdään arp-kysely

while(1){

 //ethernet-osuus
      plen = enc28j60PacketReceive(BUFFER_SIZE, buf); //luetaan onko pakettia tullut verkosta, jos on niin tallennetaan tietoliikennepuskuriin ja kerrotaan koko plen muuttujaan




 if(plen>0){ //onko mitään paketteja verkosta

  if(eth_type_is_arp_and_my_ip(buf,plen)){//onko ARP-paketti


   if (buf[ETH_ARP_OPCODE_L_P]==ETH_ARP_OPCODE_REQ_L_V){ //ARP pyyntö ->vastaus
    make_arp_answer_from_request(buf,plen);
   }else if(buf[ETH_ARP_OPCODE_L_P]==ETH_ARP_OPCODE_REPLY_L_V&&opSTATE==1){//ARP vastaus
    if(store_server_mac(buf)){ //tallennetaan palvelimen mac-osoite,mikäli tämä on oikeasta ip-osoitteesta
     sendNodeRegister(buf,myport,serverport); //rekisteröidytään palvelimelle
     opSTATE=2; //siirrytään toimintatilaan 2: odotetaan vahvistusta siitä että ollaan rekisteröidytty palvelimelle
    }
   }
  }

  if(eth_type_is_ip_and_my_ip(buf,plen)){ //sisääntuleva ip-paketti

   if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V && buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V){// ping-paketti lähetetään pong
    make_echo_reply_from_request(buf,plen);
   }

   if(udpPacketReceived(buf,myport)){ //onko paketti udp-paketti kuunneltavaan porttiin
    //PORTD &= ~(1<<PD7);
    if(isNodeRegistered(buf)&&opSTATE==2){ //saadaan vahvistus rekisteröitymisestä
     //PORTD &= ~(1<<PD7);
     opSTATE=3; //siirrytään toimintatilaan 3: kerätään XBee-moduuleilta adc-dataa
    }

    if(isDataRequest(buf)&&opSTATE==3){ //onko pyyntö datalle
     opSTATE=4;//siirrytään toimintatilaan 4: lähetetään data palvelimelle

    }


   }

  }
 }//if(plen>0)

 //xbee-osuus


 if(opSTATE==3){
  xbeelen=XBee_store_datapacket(mem,buf); //kerätään anturidataa ja tallennetaan muistiin
 }



 //tilat


 if(opSTATE==0){ //STATE init
  PORTD|=(1<<DDD7);
```

```
          client_arp_whohas(buf,serverip); //lähetetään ARP-pyyntö palvelimen ip-osoitteesta
          opSTATE=1;
        }



        if(mem[0]>MAX_MODULES){//there is a temporary node->send it
          //sendTmpNode(buf,serverport);  //katotaan myöhemmin
        }

        if(opSTATE==4){

          //mikäli moduuleilta on kerätty tietoa, lähetetään ne palvelimelle ja tyhjennetään muisti
          if(mem[0]>0){
            for(i=0;i<mem[0];i++){
              sendModuleData(buf,mem,i,myport,serverport);
              delay_ms(20);
            }

            for(i=0;i<mem[0];i++){
              clearModule(mem,i);//tyhjennetään moduulit
              delay_ms(20);
            }
            PORTD&=~(1<<DDD7);



          }
          mem[0]=0;//moduulien määrä=0
          opSTATE=3;//siirrytään takaisin datankeräystilaan
        }

        ;
      }
      return (0);
}
```

Muisti määrittely - mem.h:

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author:  Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * Memory functions.
 *
 * Title: Memory management
 *********************************************/

#ifndef MEM_H
#define MEM_H
//defines

//structure: <id:XBee-address 2Bytes > <amtnodesoptions 1 byte> <amtmeas 2 bytes>  <<  <minmax 4 bytes> <avg
4 bytes>   >>
#define MEM_ID_SIZE 2
#define MEM_OPTION_SIZE 1
#define MEM_MEAS_SIZE 2
#define MEM_MIN_MAX_SIZE 4
#define MEM_AVG_SIZE 4
#define MEM_DATA_SIZE 8
#define MAX_INST 6

#define MAX_MODULES 4
```

```
static uint8_t MEM_MODULE_SIZE =MEM_ID_SIZE+MEM_MEAS_SIZE+MEM_OPTION_SIZE+
(MAX_INST*(MEM_DATA_SIZE));

//oma datatyyppi helpottamaan liukulukujen käsittelyä
typedef union FloatUint8{
  float f;
  uint8_t b[4];

} floatuint;

//prototyypit
void storeNode(uint8_t *datamem,uint8_t module,uint8_t node,uint16_t measamt,uint16_t min,uint16_t max,uint8_t
*avg);
void clearModule(uint8_t *datamem, uint8_t module);
uint8_t addressIsStored(uint8_t *address, uint8_t *datamem);
uint8_t createModule(uint8_t *datamem, uint8_t *xbeeaddress,uint8_t amtnodes);

void clearmem(uint8_t *datamem,uint16_t size);
void updateMeas(uint8_t *datamem,uint8_t module,uint8_t measamt);

#endif
```

toiminta – mem.c

```
/*****************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author:  Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * Memory functions.
 *
 * Title: Memory management
 *****************************************/

//datamem eka paikka määrittää moduulien määrän maksimimäärä määritetään mem.h:ssa huomattava että pitää
jättää väliaikaiselle muistille paikka
#include <avr/io.h>
#include <stdlib.h>
#include"mem.h"


//storenode - tallentaa yhden jalan uudet mittaukset muistiin
//Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//module: uint8_t: moduulin indeksi
//node: uint8_t: "jalka" josta ADC-muunnoksen arvot on luettu
//measamt: uint16_t: uusien mittausten määrä
//min: uint16_t: minimiarvo
//max: uint16_t: maksimiarvo
//avg: uint8_t*:keskiarvo


void storeNode(uint8_t *datamem,uint8_t module,uint8_t node,uint16_t measamt,uint16_t min,uint16_t max,uint8_t
*avg){

  uint16_t tmpmin,tmpmax,meastmp;
  uint8_t OFFSET,pos,i;
  floatuint f,f2;


    OFFSET=(module*MEM_MODULE_SIZE)+1;
    pos=MEM_OPTION_SIZE+MEM_ID_SIZE+MEM_MEAS_SIZE+node*MEM_DATA_SIZE;
    tmpmin=(datamem[OFFSET+pos]<<8)|datamem[OFFSET+pos+1];
    tmpmax=(datamem[OFFSET+pos+2]<<8)|datamem[OFFSET+pos+3];

    if(min<tmpmin){
        tmpmin=min;
    }
```

```
    datamem[OFFSET+pos]=( 0xFF&(tmpmin>>8) );
    datamem[OFFSET+pos+1]= (0xFF&tmpmin);

    if(max>tmpmax){
      tmpmax=max;
    }

    datamem[OFFSET+pos+2]=(0xFF&(tmpmax>>8));
    datamem[OFFSET+pos+3]=(0xFF&tmpmax);

    for(i=0;i<4;i++){
      f.b[i]=avg[i]; //syötetty floatti
      f2.b[i]=datamem[OFFSET+pos+4+i]; //muistista floatti
    }
    meastmp=(datamem[OFFSET+3]<<8)|datamem[OFFSET+4];
    //f.f= f2.f+( (f.f-f2.f)/((float)meastmp+(float)measamt) ); //lasketaan keskiarvo //lets underp this
    f.f=(f2.f*(float)meastmp+f.f*(float)measamt)/((float)measamt+(float)meastmp);


    for(i=0;i<4;i++){
      datamem[OFFSET+pos+4+i]=f.b[i];
    }







}

//updateMeas - päivittää uuden mittausmäärän moduulille
//Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//module: uint8_t: moduulin indeksi
//measamt: uint16_t: uusien mittausten määrä
void updateMeas(uint8_t *datamem, uint8_t module,uint8_t measamt){
  uint16_t meastmp;
  uint8_t OFFSET;
  OFFSET=(module*MEM_MODULE_SIZE)+1;
  meastmp=(datamem[OFFSET+3]<<8)|datamem[OFFSET+4];
  meastmp+=measamt;
  datamem[OFFSET+3]=(uint8_t)((0xFF00&meastmp)>>8);
  datamem[OFFSET+4]=(0xFF&meastmp);

}

//clearmem – poistaa/tyhjntää moduulin
// Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//module: uint8_t: moduulin indeksi
void clearModule(uint8_t *datamem, uint8_t module){
  uint8_t i;
  uint16_t OFFSET;
  OFFSET=(module*MEM_MODULE_SIZE)+1;

  for(i=0;i<MEM_MODULE_SIZE;i++){
    datamem[OFFSET+i]=0;
  }

}

//addressIsStored – tarkastaa onko tietyllä osoitteella oleva moduuli tallennettu muistiin
//Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//address: uint8_t*: osoitin XBee-moduulin osoitteeseen
//Palauttaa: löydetyn moduulin indeksin tai 250 jos moduulia ei löydy


uint8_t addressIsStored(uint8_t *address, uint8_t *datamem){
```

```
  uint8_t i;
  for(i=0;i<MAX_MODULES;i++){
    if(datamem[1+(MEM_MODULE_SIZE*i)]==address[0] && datamem[1+(MEM_MODULE_SIZE*i)+1]==address[1]){
      return i;
    }
  }
  return 250;
}

//createModule - alustaa muistipaikan uudelle moduulille
//Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//xbeeaddress: uint8_t*: osoitin XBee-moduulin osoitteeseen
//amtnodes: uint8_t: jalkojen määrä josta adc-muunnos muodostetaan
//Palauttaa: luodun moduulin indexin

uint8_t createModule(uint8_t *datamem, uint8_t *xbeeaddress,uint8_t amtnodes){

  uint8_t module,i,pos;
  uint16_t OFFSET;
  module=datamem[0];

  OFFSET=(module*MEM_MODULE_SIZE)+1;

  datamem[OFFSET]=xbeeaddress[0];
  datamem[OFFSET+1]=xbeeaddress[1];

  datamem[OFFSET+2]=amtnodes;

  pos=MEM_ID_SIZE+MEM_OPTION_SIZE;

  datamem[OFFSET+pos]=0;
  datamem[OFFSET+pos+1]=0;

  for(i=0;i<MAX_INST;i++){
    pos=MEM_ID_SIZE+MEM_OPTION_SIZE+MEM_MEAS_SIZE+(MEM_DATA_SIZE*i);
    datamem[OFFSET+pos]=0xff;
    datamem[OFFSET+pos+1]=0xFF;
    datamem[OFFSET+pos+2]=0;
    datamem[OFFSET+pos+3]=0;
    datamem[OFFSET+pos+4]=0;
    datamem[OFFSET+pos+5]=0;
    datamem[OFFSET+pos+6]=0;
    datamem[OFFSET+pos+7]=0;

  }
  datamem[0]=module+1;

  return module;
}

//clearmem - tyhjentää muistialueen(ts. nollaa sen)
//Parametrit:
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//size: uint16_t: muistin määrä joka tyhjennetään
void clearmem(uint8_t *datamem,uint16_t size){
  uint16_t i;
  for(i=0;i<size;i++){
      datamem[i]=0;
  }
}
```

XBee-moduulin hallinta

määrittely - xbee.h

```
/*****************************************
 * vim:sw=8:ts=8:si:et
```

```
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author:  Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * functions to manage XBee module ADC-data.
 *
 * Title: XBee
 ********************************************/
#ifndef XBEE_H
#define XBEE_H

uint16_t XBee_store_datapacket(uint8_t *datamem,uint8_t *buf);
void getValues(uint8_t *buf,uint8_t node, uint8_t numnodes, uint16_t *min, uint16_t *max, uint8_t *avg);
uint8_t getNodeCount(uint8_t *buf);
uint16_t XBee_store_datapacket_dbg(uint8_t *datamem,uint8_t *buf,uint8_t *dbgbuf);

#endif
```

toiminta xbee.c

```
/********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author:  Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * functions to manage XBee module ADC-data.
 *
 * Title: XBee
 ********************************************/
#include <avr/io.h>
#include "uart.h"
#include "rprintf.h"
#include "xbee.h"
#include "mem.h"

//XBee_store_datapacket - lukee USART-porttia vastaanottaakseen XBee-moduulin ADC-kehyksen ja tallentaa sen tietoliikennepuskuriin ja
//purkaa ADC-kehyksen tietoliikennepuskurista datamuistiin
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//Palauttaa: luetut tavut
uint16_t XBee_store_datapacket(uint8_t *datamem,uint8_t *buf){
  uint8_t rb;
  uint8_t i;
  uint16_t len,pos;
  uint8_t nodecnt,module;
  uint16_t min,max;
  uint8_t avg[4];
  uint8_t xbeeaddr[2];

  if(!uartReceiveByte(&rb)){ //jos ei ole tullut mitään->palautetaan 0
    return 0;
  }

  if(rb==0x7e){ //onko XBee adc-framen eka tavu
    pos=0;//alussa 0;

    while(pos<2){ //luetaan pituus
      if(uartReceiveByte(&rb)){
        buf[pos]=rb;
        pos++;
      }
    }
    len=(buf[0]<<8)+buf[1];

    //luetaan ADC-paketti muistiin
    while(pos<(len+3)){//luetaan myös checksum +1 ja kokotavut +2
      if(uartReceiveByte(&rb)){
```

```
      buf[pos]=rb;
      pos++;
    }
   }


   nodecnt=getNodeCount(buf); //luetaan "jalkojen" määrä

   //XBee 16-bittinen osoite
   xbeeaddr[0]=buf[3];
   xbeeaddr[1]=buf[4];

   module=250; //liian iso luku muistiin asetettavaksi, indexi voi kuitenkin olla 0
   module=addressIsStored(xbeeaddr, datamem); //tarkistetaan onko kyseinen moduuli jo tallennettuna muistiin
   if(module==250){ //ei ole tallennettu muistiin
     module=createModule(datamem, xbeeaddr,nodecnt); //luodaan moduulille muistiin tila

   }
   for(i=0;i<nodecnt;i++){
     getValues(buf,i,nodecnt,&min,&max,avg); //luetaan "jalan" i datat min-, max- ja avg-muuttujiin
     storeNode(datamem, module,i,buf[7], min, max,avg); //tallennetaan "jalan" tiedot muistiin




   }
   updateMeas(datamem,module,buf[7]); //päivitetää mittausten määrä



   return pos;//+kokotavut + checksum

  }else{ // ei ollut XBee adc-framen eka tavu->palautetaan 0;
    return 0;
  }

}

const int MEAS_P =10;//mittausten kohta

//getValues - lukee tietoliikennepuskurista XBee-moduulin ADC-kehyksestä yhden jalan muunnokset ja
//kertoo minimi-, maksimi- ja keskiarvot
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//node: uint8_t: "jalka" josta ADC-muunnoksen arvot on luettu
//numnodes:uint8_t: kuinka monesta jalasta ADC-muunnos on tehty, tarvitaan siihen että voidaan liikkua puskurissa
oikein
//min:uint16_t*: osoitin minimiarvoon
//max:uint16_t*: osoitin maksimiarvoon
//avg:uint8_t*:osoitin keskiarvoon, single precision float-tyyppinen joten pituus on 4 tavua
void getValues(uint8_t *buf,uint8_t node, uint8_t numnodes,uint16_t *min, uint16_t *max, uint8_t *avg){

  //luetaan bufferista kuinka motna mittausta per node
  uint8_t nummeas;

  nummeas=buf[7];
  uint16_t tmp,sum;
  uint8_t i;

  *min=0xFFFF; //otoksen minimi aloitetaan ylisuuresta arvosta
  *max=0;      //otoksen maksimi aloitetaa arvosta 0

  floatuint f;

  sum=0;
  for(i=0;i<nummeas;i++){
          tmp=((buf[MEAS_P+node*2+(i*2*numnodes)]<<8)+(buf[MEAS_P+node*2+(i*2*numnodes)+1]));   //luetaan
puskurista arvo väliaikaismuuttujaan
    sum+=tmp; //juokseva summa
    if(*min>tmp){ //onko pienempi kuin nykyinenn minimi
```

```
      *min=tmp;

    }
    if(*max<tmp){ //onko isompi kuin nykyinen maksimi
      *max=tmp;
    }

  }

  f.f=(float)sum/(float)nummeas; //keskiarvo
  for(i=0;i<4;i++){
    avg[i]=f.b[i]; //kirjoitetaan keskiarvo puskuriin

  }


}

//getNodeCount - kertoo kuinka monesta jalasta AD-muunnos on otettu
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//Palauttaa: kuinka monesta jalasta AD-muunnos on otettu
uint8_t getNodeCount(uint8_t *buf){

  //buf[8] channel indikaattori MSB: bitit 1-6 (lsb-msb) kertovat onko tietty adc käytössä
  uint8_t i;
  uint8_t cnt;
  //käydään läpi jokainen  channel indicator
  cnt=0;
  for(i=0;i<6;i++){
    if( (buf[8]&(1<<(i+1))) ){
      cnt++;
    }


  }
  return cnt;
}
```

Tietoliikenne palvelimelle
määrittely ip_arp_udp.h

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher extended for use with UDP/ASPv0 by Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * IP, Arp and UDP functions.
 *
 * Title: Microchip ENC28J60 Ethernet Interface Driver
 * Chip type           : ATMEGA88 with ENC28J60
 *********************************************/
//@{
#ifndef IP_ARP_UDP_H
#define IP_ARP_UDP_H
// you must call this function once before you use any of the other functions:
extern void init_ip_arp_udp(uint8_t *mymac,uint8_t *myip);
//
extern uint8_t eth_type_is_arp_and_my_ip(uint8_t *buf,uint8_t len);
extern uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint8_t len);
extern void make_arp_answer_from_request(uint8_t *buf,uint8_t len);
extern void make_echo_reply_from_request(uint8_t *buf,uint8_t len);
extern void make_udp_reply_from_request(uint8_t *buf,char *data,uint8_t datalen,uint16_t port);
extern void setmac_server(uint8_t *macaddress);
extern uint8_t store_server_mac(uint8_t *buf);
extern void client_arp_whohas(uint8_t *buf,uint8_t *ip_we_search);
extern void send_udp(uint8_t *buf,uint8_t *data,uint8_t datalen,uint16_t srcport,uint16_t destport);
extern void set_server_ip(uint8_t *ipaddress);
```

```
//omat
void sendNodeRegister(uint8_t *buf,uint16_t srcport,uint16_t destport);
uint8_t isNodeRegistered(uint8_t *buf);
uint8_t isDataRequest(uint8_t *buf);
uint8_t udpPacketReceived(uint8_t *buf,uint16_t port);
void sendModuleData(uint8_t *buf,uint8_t *datamem,uint8_t module,uint16_t srcport,uint16_t destport);
void sendNodeAmt(uint8_t *buf,uint8_t *datamem,uint16_t srcport,uint16_t destport);
#endif /* IP_ARP_UDP_H */
//@}
```

toiminta – ip_arp_udp.c

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher extended for use with UDP/ASPv0 by Timo-Tapio Palmu
 * Copyright: GPL V2
 *
 * IP, Arp and UDP functions.
 *
 * Title: Microchip ENC28J60 Ethernet Interface Driver
 * Chip type           : ATMEGA88 with ENC28J60
 *********************************************/
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "avr_compat.h"
#include "net.h"
#include "enc28j60.h"

#include "mem.h"

static uint8_t macaddr[6];
static uint8_t ipaddr[4];
uint8_t servermac[6];
uint8_t serverip[4];
const char arpreqhdr[] PROGMEM ={0,1,8,0,6,4,0,1};
const char iphdr[] PROGMEM ={0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x40, 0x11, 0xb9, 0x77};

const uint8_t startbyte_data=0xFC;

// The Ip checksum is calculated over the ip header only starting
// with the header length field and a total length of 20 bytes
// unitl ip.dst
// You must set the IP checksum field to zero before you start
// the calculation.
// len for ip is 20.
//
// For UDP/TCP we do not make up the required pseudo header. Instead we
// use the ip.src and ip.dst fields of the real packet:
// The udp checksum calculation starts with the ip.src field
// Ip.src=4bytes,Ip.dst=4 bytes,Udp header=8bytes + data length=16+len
// You must set the upd checksum field to zero before you start
// the calculation.
// len for udp is: 8 + 8 + data length
//
// For more information on how this algorithm works see:
// http://www.netfor2.com/checksum.html
// http://www.msc.uky.edu/ken/cs471/notes/chap3.htm
// The RFC has also a C code example: http://www.faqs.org/rfcs/rfc1071.html
uint16_t checksum(uint8_t *buf, uint16_t len,uint8_t type){
        // type 0=ip
        //      1=udp
        //      2=tcp
```

```
        uint32_t sum = 0;


        if(type==1){
                sum+=IP_PROTO_UDP_V; // protocol udp
                // the length here is the length of udp (data+header len)
                // =length - IP addr length
                sum+=len-8; // = real udp len
        }
        if(type==2){
                sum+=IP_PROTO_TCP_V;
                // the length here is the length of tcp (data+header len)
                // =length - IP addr length
                sum+=len-8; // = real tcp len
        }
        // build the sum of 16bit words
        while(len >1){
                sum += 0xFFFF & (*buf<<8|*(buf+1));
                buf+=2;
                len-=2;
        }
        // if there is a byte left then add it (padded with zero)
        if (len){
                sum += (0xFF & *buf)<<8;
        }
        // now calculate the sum over the bytes in the sum
        // until the result is only 16bit long
        while (sum>>16){
                sum = (sum & 0xFFFF)+(sum >> 16);
        }
        // build 1's complement:
        return( (uint16_t) sum ^ 0xFFFF);
}

// you must call this function once before you use any of the other functions:
void init_ip_arp_udp(uint8_t *mymac,uint8_t *myip){
        uint8_t i=0;
        while(i<4){
                ipaddr[i]=myip[i];
                i++;
        }
        i=0;
        while(i<6){
                macaddr[i]=mymac[i];
                i++;
        }
}

uint8_t eth_type_is_arp_and_my_ip(uint8_t *buf,uint8_t len){
        uint8_t i=0;
        //
        if (len<41){
                return(0);
        }
        if(buf[ETH_TYPE_H_P] != ETHTYPE_ARP_H_V ||
           buf[ETH_TYPE_L_P] != ETHTYPE_ARP_L_V){
                return(0);
        }
        while(i<4){
                if(buf[ETH_ARP_DST_IP_P+i] != ipaddr[i]){
                        return(0);
                }
                i++;
        }
        return(1);
}

uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint8_t len){
        uint8_t i=0;
        //eth+ip+udp header is 42
        if (len<42){
```

```
                return(0);
        }
        if(buf[ETH_TYPE_H_P]!=ETHTYPE_IP_H_V ||
           buf[ETH_TYPE_L_P]!=ETHTYPE_IP_L_V){
                return(0);
        }
        while(i<4){
                if(buf[IP_DST_P+i]!=ipaddr[i]){
                        return(0);
                }
                i++;
        }
        return(1);
}
// make a return eth header from a received eth packet

void make_eth(uint8_t *buf)
{
        uint8_t i=0;
        //
        //copy the destination mac from the source and fill my mac into src
        while(i<6){
                buf[ETH_DST_MAC +i]=buf[ETH_SRC_MAC +i];
                buf[ETH_SRC_MAC +i]=macaddr[i];
                i++;
        }
}




// make a return ip header from a received ip packet

void make_ip(uint8_t *buf)
{
        uint8_t i=0;
        uint16_t ck;
        while(i<4){
                buf[IP_DST_P+i]=buf[IP_SRC_P+i];
                buf[IP_SRC_P+i]=ipaddr[i];
                i++;
        }
        // clear the 2 byte checksum
        buf[IP_CHECKSUM_P]=0;
        buf[IP_CHECKSUM_P+1]=0;
        buf[IP_FLAGS_P]=0x40; // don't fragment
        buf[IP_FLAGS_P+1]=0;  // fragement offset
        buf[IP_TTL_P]=64; // ttl
        // calculate the checksum:
        ck=checksum(&buf[IP_P], IP_HEADER_LEN,0);
        buf[IP_CHECKSUM_P]=ck>>8;
        buf[IP_CHECKSUM_P+1]=ck& 0xff;
}

void make_arp_answer_from_request(uint8_t *buf,uint8_t len)
{
        uint8_t i=0;
        //
        make_eth(buf);
        buf[ETH_ARP_OPCODE_H_P]=ETH_ARP_OPCODE_REPLY_H_V;
        buf[ETH_ARP_OPCODE_L_P]=ETH_ARP_OPCODE_REPLY_L_V;
        // fill the mac addresses:
        while(i<6){
                buf[ETH_ARP_DST_MAC_P+i]=buf[ETH_ARP_SRC_MAC_P+i];
                buf[ETH_ARP_SRC_MAC_P+i]=macaddr[i];
                i++;
        }
        i=0;
        while(i<4){
```

```
                buf[ETH_ARP_DST_IP_P+i]=buf[ETH_ARP_SRC_IP_P+i];
                buf[ETH_ARP_SRC_IP_P+i]=ipaddr[i];
                i++;
        }
        // eth+arp is 42 bytes:
        enc28j60PacketSend(42,buf);
}

void make_echo_reply_from_request(uint8_t *buf,uint8_t len)
{
        make_eth(buf);
        make_ip(buf);
        buf[ICMP_TYPE_P]=ICMP_TYPE_ECHOREPLY_V;
        // we changed only the icmp.type field from request(=8) to reply(=0).
        // we can therefore easily correct the checksum:
        if (buf[ICMP_CHECKSUM_P] > (0xff-0x08)){
                buf[ICMP_CHECKSUM_P+1]++;
        }
        buf[ICMP_CHECKSUM_P]+=0x08;
        //
        enc28j60PacketSend(len,buf);
}

//from 3.6

void fill_buf_p(uint8_t *buf,uint16_t len, const prog_char *progmem_s)
{
        while (len){
                *buf= pgm_read_byte(progmem_s);
                buf++;
                progmem_s++;
                len--;
        }
}

void client_arp_whohas(uint8_t *buf,uint8_t *ip_we_search)
{
        uint8_t i=0;
        //
        while(i<6){
                buf[ETH_DST_MAC +i]=0xff;
                buf[ETH_SRC_MAC +i]=macaddr[i];
                i++;
        }
        buf[ETH_TYPE_H_P] = ETHTYPE_ARP_H_V;
        buf[ETH_TYPE_L_P] = ETHTYPE_ARP_L_V;
        fill_buf_p(&buf[ETH_ARP_P],8,arpreqhdr);
        i=0;
        while(i<6){
                buf[ETH_ARP_SRC_MAC_P +i]=macaddr[i];
                buf[ETH_ARP_DST_MAC_P+i]=0;
                i++;
        }
        i=0;
        while(i<4){
                buf[ETH_ARP_DST_IP_P+i]=*(ip_we_search +i);
                buf[ETH_ARP_SRC_IP_P+i]=ipaddr[i];
                i++;
        }
        // 0x2a=42=len of packet
        enc28j60PacketSend(0x2a,buf);
}

void set_server_ip(uint8_t *ipaddressi){
   uint8_t i=0;

   while(i<4){
     serverip[i]=ipaddressi[i];
     i++;
```

```
   }

}

uint8_t store_server_mac(uint8_t *buf)
{
	uint8_t i=0;
	while(i<4){
		if(buf[ETH_ARP_SRC_IP_P+i]!=serverip[i]){

			return 0;
		}
		i++;
	}
	i=0;
	while(i<6){
		servermac[i]=buf[ETH_ARP_SRC_MAC_P +i];
		i++;
	}
	return(1);
}

//omat funktiot/laajennukset

//make_eth_to send - muunnos alkuperäisestä make_eth -funktiosta jossa vastaanottajan MAC-osoite otetaan ARP:lla haetusta
//Parametrit:
//buf: unit8_t* : osoitin tietoliikennepuskuriin
void make_eth_to_send(uint8_t *buf){
  uint8_t i=0;
  while(i<6){
    buf[ETH_DST_MAC +i]=servermac[i];
    buf[ETH_SRC_MAC +i]=macaddr[i];
    i++;
  }
  buf[ETH_TYPE_H_P] = ETHTYPE_IP_H_V;
  buf[ETH_TYPE_L_P] = ETHTYPE_IP_L_V;
}

//make_ip_to_send - muunnos alkuperäisestä make_ip -funktiosta jossa vastaanottava ip-osoite määrittyy tallennetun palvelimen ip:n perusteella
//Parametrit:
//buf: uint8_t* : osoitin tietoliikennepuskuriin
//datalen: uint8_t : kuinka monta tavua dataa lähetetään
void make_ip_to_send(uint8_t *buf,uint8_t datalen){
  //buf[IP_DST_P]=a;buf[IP_DST_P+1]=b;buf[IP_DST_+2P]=c;buf[IP_DST_P+3]=d;
  uint8_t i=0;
  uint16_t ck;
  fill_buf_p(&buf[IP_P],9,iphdr);
  buf[IP_PROTO_P]=IP_PROTO_UDP_V;
  while(i<4){
    buf[IP_DST_P+i]=serverip[i];
    buf[IP_SRC_P+i]=ipaddr[i];
    i++;
  }

  buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+UDP_HEADER_LEN+datalen;

  // clear the 2 byte checksum
  buf[IP_CHECKSUM_P]=0;
  buf[IP_CHECKSUM_P+1]=0;
  buf[IP_FLAGS_P]=0x40; // don't fragment
  buf[IP_FLAGS_P+1]=0;  // fragement offset
  buf[IP_TTL_P]=64; // ttl
  // calculate the checksum:
  ck=checksum(&buf[IP_P], IP_HEADER_LEN,0);
  buf[IP_CHECKSUM_P]=ck>>8;
  buf[IP_CHECKSUM_P+1]=ck& 0xff;
```

```
}

//udpPacketReceived - tarkistaa onko tietoliikennepuskurissa viesti tiettyyn porttiin
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//port: uint16_t : UDP-portti
uint8_t udpPacketReceived(uint8_t *buf,uint16_t port){

                        if(buf[IP_PROTO_P]==IP_PROTO_UDP_V&&buf[UDP_DST_PORT_H_P]==(0xFF&(port>>8)            )
&&buf[UDP_DST_PORT_L_P]==(0xFF&port)){
    return 1;
  }else{
    return 0;
  }


}

//sendModuleData - lähettää muistiin tallennetun XBee-moduulin palvelimelle
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//datamem: uint8_t*: osoitin muistialueeseen joka pitää sisällään moduulien datan
//module: uint8_t: lähetättävä moduuli (0-n)
//srcport: uint16_t: lähtevä portti
//destport: uint16_t: vastaanottava portti
void sendModuleData(uint8_t *buf,uint8_t *datamem,uint8_t module,uint16_t srcport,uint16_t destport){
  make_eth_to_send(buf);


  uint8_t i=0;
  uint8_t amtnodes;
  uint16_t ck,OFFSET;
  uint8_t datalen;
  OFFSET=MEM_MODULE_SIZE*module+1;
  amtnodes=datamem[OFFSET+2];//(0x0F)&(datamem[OFFSET+2]>>4); //DERP
  datalen=8+4+amtnodes*8;//;
  if(datalen>220)datalen=220; //restrict size//saas nähdä mitä tollekin...



  make_ip_to_send(buf,datalen);
  buf[IP_TOTLEN_H_P]=0;
  buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+UDP_HEADER_LEN+datalen;


  buf[UDP_DST_PORT_H_P]=destport>>8;
  buf[UDP_DST_PORT_L_P]= destport&0xff;
  buf[UDP_SRC_PORT_H_P]=srcport>>8;
  buf[UDP_SRC_PORT_L_P]=srcport & 0xff;

  buf[UDP_LEN_H_P]=0;
  buf[UDP_LEN_L_P]=UDP_HEADER_LEN+datalen;

  buf[UDP_CHECKSUM_H_P]=0;
  buf[UDP_CHECKSUM_L_P]=0;


  //amtnodes=(0x0F)&(datamem[OFFSET+2]>>4);

  //kirjoitetaan puskuri
  buf[UDP_DATA_P]=0xFC;
  for(i=0;i<6;i++){
    buf[1+UDP_DATA_P+i]=macaddr[i];
  }

  buf[1+UDP_DATA_P+6]=datamem[OFFSET];
  buf[1+UDP_DATA_P+7]=datamem[OFFSET+1];
  buf[1+UDP_DATA_P+8]=amtnodes;
  buf[1+UDP_DATA_P+9]=datamem[OFFSET+3];
  buf[1+UDP_DATA_P+10]=datamem[OFFSET+4];
```

```
  for(i=0;i<amtnodes*MEM_DATA_SIZE;i++){
    buf[1+UDP_DATA_P+11+i]=datamem[OFFSET+5+i];
  }




  ck=checksum(&buf[IP_SRC_P], 16 + datalen,1);
  buf[UDP_CHECKSUM_H_P]=ck>>8;
  buf[UDP_CHECKSUM_L_P]=ck& 0xff;
  enc28j60PacketSend(UDP_HEADER_LEN+IP_HEADER_LEN+ETH_HEADER_LEN+datalen,buf);

}

//sendNodeRegister - lähettää rekisteröintipyynnön("REG") palvelimelle
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
//srcport: uint16_t: lähtevä portti
//destport: uint16_t: vastaanottava portti
void sendNodeRegister(uint8_t *buf,uint16_t srcport,uint16_t destport){
  make_eth_to_send(buf);



  uint16_t ck;
  uint8_t datalen;
  datalen=3;
  make_ip_to_send(buf,datalen);
  buf[IP_TOTLEN_H_P]=0;
  buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+UDP_HEADER_LEN+datalen;


  buf[UDP_DST_PORT_H_P]=destport>>8;
  buf[UDP_DST_PORT_L_P]= destport&0xff;
  buf[UDP_SRC_PORT_H_P]=srcport>>8;
  buf[UDP_SRC_PORT_L_P]=srcport & 0xff;

  buf[UDP_LEN_H_P]=0;
  buf[UDP_LEN_L_P]=UDP_HEADER_LEN+datalen;

  buf[UDP_CHECKSUM_H_P]=0;
  buf[UDP_CHECKSUM_L_P]=0;

  buf[UDP_DATA_P]='R';
  buf[UDP_DATA_P+1]='E';
  buf[UDP_DATA_P+2]='G';


  ck=checksum(&buf[IP_SRC_P], 16 + datalen,1);
  buf[UDP_CHECKSUM_H_P]=ck>>8;
  buf[UDP_CHECKSUM_L_P]=ck& 0xff;
  enc28j60PacketSend(UDP_HEADER_LEN+IP_HEADER_LEN+ETH_HEADER_LEN+datalen,buf);


}

//isNodeRegistered - tarkistaa onko tietoliikennepuskurissa vahvistus rekisteröinnistä
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
uint8_t isNodeRegistered(uint8_t *buf){
  if(buf[UDP_DATA_P]=='R'&&buf[UDP_DATA_P+1]=='O'&&buf[UDP_DATA_P+2]=='K'){
    return 1;
  }else{
    return 0;
  }


}
```

```
//isDataRequest - tarkistaan onko tietoliikennepuskurissa pyyntö moduulien datalle
//Parametrit:
//buf: uint8_t*: osoitin tietoliikennepuskuriin
uint8_t isDataRequest(uint8_t *buf){
  if(buf[UDP_DATA_P]=='R'&&buf[UDP_DATA_P+1]=='E'&&buf[UDP_DATA_P+2]=='Q'){
    return 1;
  }else{
    return 0;
  }


}
```

tukitiedot

ethernet/ip/udp-määrittelyitä – net.h

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher
 * Copyright: GPL V2
 *
 * Based on the net.h file from the AVRlib library by Pascal Stang.
 * For AVRlib See http://www.procyonengineering.com/
 * Used with explicit permission of Pascal Stang.
 *
 * Chip type           : ATMEGA88 with ENC28J60
 *********************************************/
//@{

#ifndef NET_H
#define NET_H

// ******* ETH *******
#define ETH_HEADER_LEN  14
// values of certain bytes:
#define ETHTYPE_ARP_H_V 0x08
#define ETHTYPE_ARP_L_V 0x06
#define ETHTYPE_IP_H_V  0x08
#define ETHTYPE_IP_L_V  0x00
// byte positions in the ethernet frame:
//
// Ethernet type field (2bytes):
#define ETH_TYPE_H_P 12
#define ETH_TYPE_L_P 13
//
#define ETH_DST_MAC 0
#define ETH_SRC_MAC 6

//arp pos from newer version
#define ETH_ARP_P 0xe

// ******* ARP *******
#define ETH_ARP_OPCODE_REPLY_H_V 0x0
#define ETH_ARP_OPCODE_REPLY_L_V 0x02
//
#define ETHTYPE_ARP_L_V 0x06
// arp.dst.ip
#define ETH_ARP_DST_IP_P 0x26
// arp.opcode
#define ETH_ARP_OPCODE_H_P 0x14
#define ETH_ARP_OPCODE_L_P 0x15
// arp.src.mac
#define ETH_ARP_SRC_MAC_P 0x16
#define ETH_ARP_SRC_IP_P 0x1c
#define ETH_ARP_DST_MAC_P 0x20
#define ETH_ARP_DST_IP_P 0x26

#define ETH_ARP_OPCODE_REQ_L_V 0x01
```

```
// ******* IP *******
#define IP_HEADER_LEN   20
// ip.src
#define IP_SRC_P 0x1a
#define IP_DST_P 0x1e
#define IP_CHECKSUM_P 0x18
#define IP_TTL_P 0x16
#define IP_FLAGS_P 0x14
#define IP_P 0xe
#define IP_TOTLEN_H_P 0x10
#define IP_TOTLEN_L_P 0x11


#define IP_PROTO_P 0x17


#define IP_PROTO_ICMP_V 1
#define IP_PROTO_TCP_V 6
#define IP_PROTO_UDP_V 17
// ******* ICMP *******
#define ICMP_TYPE_ECHOREPLY_V 0
#define ICMP_TYPE_ECHOREQUEST_V 8
//
#define ICMP_TYPE_P 0x22
#define ICMP_CHECKSUM_P 0x24


// ******* UDP *******
#define UDP_HEADER_LEN  8
//
#define UDP_SRC_PORT_H_P 0x22
#define UDP_SRC_PORT_L_P 0x23
#define UDP_DST_PORT_H_P 0x24
#define UDP_DST_PORT_L_P 0x25
//
#define UDP_LEN_H_P 0x26
#define UDP_LEN_L_P 0x27
#define UDP_CHECKSUM_H_P 0x28
#define UDP_CHECKSUM_L_P 0x29
#define UDP_DATA_P 0x2a


#endif
//@}
```

AVRlib määrittelyt – globals.h

```
//*****************************************************************************
//
// File Name    : 'global.h'
// Title                : AVR project global include
// Author               : Pascal Stang
// Created              : 7/12/2001
// Revised              : 9/30/2002
// Version              : 1.1
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
//      Description : This include file is designed to contain items useful to all
//                                      code files and projects.
//
// This code is distributed under the GNU Public License
//              which can be found at http://www.gnu.org/licenses/gpl.txt
//
//*****************************************************************************

#ifndef GLOBAL_H
#define GLOBAL_H

// global AVRLIB defines
#include "avrlibdefs.h"
// global AVRLIB types definitions
```

```
#include "avrlibtypes.h"

// project/system dependent defines

// CPU clock speed
//#define F_CPU        16000000                 // 16MHz processor
//#define F_CPU        14745000                 // 14.745MHz processor
#define F_CPU        8000000                  // 8MHz processor
//#define F_CPU        7372800                  // 7.37MHz processor
//#define F_CPU        12500000                 // 12.5MHz processor
//#define F_CPU        3686400                  // 3.69MHz processor
#define CYCLES_PER_US ((F_CPU+500000)/1000000)  // cpu cycles per microsecond

#endif
```

Lisäksi tarvitaan Guido Socherin ENC28J60-ajuri ja AVRlib-kirjastosta uart,rprintf ja buffer osuudet. Näitä ei ole sisällytetty tähän sillä ne toimivat muokkaamattomina.

Liite 2. Palvelinohjelma

Palvelinohjelman kääntäminen

```
g++ -o palvelin main.cpp cserver.cpp cclientconnection.cpp cclientdeviceconnection.cpp \
cdevicedataset.cpp cdatabase.cpp misc.cpp -I/usr/include -I/usr/include/mysql -I/usr/include/mysql++\
-lnsl -lresolv -lboost_thread -lmysqlpp
```

pääohjelma
main.cpp

```
#include <iostream>
#include "cserver.h"

int main(int argc, char **argv) {
  CServer server;

  server.init(31337); //initialisoidaan kuunnellen porttia 31337
  server.start(); //käynnistys

  return 0;
}
```

palvelinolio
määrittely – cserver.h

```
#ifndef CSERVER_H
#define CSERVER_H

#include<iostream>
#include<vector>
#include "cclientconnection.h"
#include "cclientdeviceconnection.h"
#include "cdatabase.h"

#include <boost/thread.hpp>
#include <boost/date_time.hpp>
#include <boost/bind.hpp>

#include "configs.h"

class CServer
{
  public:

    int init(int portnum);
    int getMessage();
    void join_all();          //liitetään käynnistetyt prosessit alkuperäiseen prosessiin
    void stop();              //serveriohjelman pysäytysmetodi
    void start();             //serveriohjelman käynnistysmetodi
    void run();               //serveriohjelman ajometodi
    Int establish(int portnum);//metodi kuuntelusocketin määrittelemiseen
    int getRegister(std::string msg);

    //säikeiden metodit
    void processListen();
    void processClients();
    void processDatabase();



  private:
    std::vector<CClientConnection> clientconnections;
    std::vector<CClientDeviceConnection> clientdeviceconnections;
```

```
    CDatabase database;

    boost::thread listenthread;
    boost::thread clientthread;
    boost::thread dbthread;

    int serversocket;
    int portnumber;
    int running;
    int idlelooptimer;
};

#endif // CSERVER_H
```

toiminta cserver.cpp

```
#include "cserver.h"
#include <sys/types.h>
#include <sys/socket.h> //socket(), bind() jne
#include <netinet/in.h> // sockaddr_in, htons jne
#include <netdb.h> //gethostbyname()
#include <errno.h> //virheilmoitukset
#include <fcntl.h> //fcntl()

#include "configs.h"
#include "misc.h"
#include<ctime>

void CServer::join_all(){
  listenthread.join();
  clientthread.join();
  dbthread.join();

}

int CServer::establish(int portnum){
  char hostname[MAXHOSTNAME+1];//isännän nimi jonka kokoon lisätään 1 nollaterminointia varten
//  int s; //socket
  struct sockaddr_in sa; //socketin osoite
  struct hostent *he; // isännän entiteetti
  memset(&sa, 0,sizeof(struct sockaddr_in)); //tyhjennetään osoite(kirjoitetaan nollia täyteen)
  gethostname(hostname,MAXHOSTNAME);//luetaan isännän nimi
  he=gethostbyname(hostname);//luetaan isännän entiteetti nimen perusteella

  if(he==NULL){ //jos ei onnistunut...
    return -1; //palautetaan arvo -1 ja poistutaan metodista
  }

  sa.sin_family=he->h_addrtype;//otetaan osoitteen tyyppi isäntäentiteetiltä
      sa.sin_port=htons(portnumber);//määritellään tietoliikenneportti.. htons() muuttaa short-tyyppisen numeron isäntäkoneen tavujärjestyksestä verkkoliikenne tavuj?rjestykseen(htons=host to network short)

   if((serversocket=socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP))<0){ // luodaan socketti, domain Protocol Family: I(nter)NET(PF_INET), type: SOCK_DGRAM(UDP on DGRAM tyyppinen), protokolla: UDP
    return -1;
  }

  int yes=1;//setsockoptia varten


      if((setsockopt(serversocket,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)))==-1){  //määritellään että samaan osoitteeseen voi liittyä useampi socketi. Määritelty socket-tasolla(vrt. ipproto)
    std::cout << "Error @ setsockopt()"<<std::endl;
    return -1;
  }


  if(bind(serversocket,(struct sockaddr*)&sa,sizeof(struct sockaddr_in)) <0){ //sidotaan osoite sockettiin
    close(serversocket); //ei onnistunut: suljetaan socket
    return -1;//ja palautetaan virhe
```

```
  }


    fcntl(serversocket,F_SETFL,O_NONBLOCK); //asetetaan socketin statusflageista päälle tila jossa yhteys(luku) ei pysäytä ohjelman suoritus(pääsee jatkamaan jos ei ole yhtään viestiä/uutta yhteyttä jne)



}

int CServer::getRegister(std::string msg)
{
  if(msg.c_str()[0]=='R'&&msg.c_str()[1]=='E'&&msg.c_str()[2]=='G'){
    return DEVICE_CONNECTION;
  } else if(msg.c_str()[0]=='R'&&msg.c_str()[1]=='E'&&msg.c_str()[2]=='C'){
    return CONFIG_CONNECTION;
  }else{

   return 0;
  }
}

int CServer::getMessage(){

  boost::posix_time::seconds mytimer(idlelooptimer);

  struct sockaddr_in fromaddr;
  socklen_t addrlen;
  int i,connectionfound,msgsize;
  unsigned char buffer[MAXMSGLEN];

  std::string msg;


  msgsize=recvfrom(serversocket,buffer,MAXMSGLEN,0,(struct sockaddr*)&fromaddr,&addrlen);


  buffer[MAXMSGLEN]='\0'; //nolla-terminoidaan



  if(msgsize>0){ //onko viestiä



    msg=uctostr(buffer,MAXMSGLEN);
    connectionfound=0;
    for(i=0;i<clientdeviceconnections.size();i++){
      if(clientdeviceconnections[i].isConnection(fromaddr)){
        clientdeviceconnections[i].receiveMsg(buffer);
        connectionfound=1;
      }
    }
  }else{

    return 0;

  }

  if(!connectionfound){
    if(getRegister(msg)==CONFIG_CONNECTION){
      CClientConnection tmp;
      tmp.init(fromaddr,serversocket);//samaan sockettiin lähtee mistä tulikin..
      clientconnections.push_back(tmp);



    }else if(getRegister(msg)==DEVICE_CONNECTION){
      CClientDeviceConnection tmp;
```

```
      tmp.init(fromaddr,serversocket);//samaan sockettiin lähtee mistä tulikin..
      tmp.stampTime();
      tmp.setTimer(5);//kerran minuutissa
      tmp.sendRegOk();
      clientdeviceconnections.push_back(tmp);



    }else{
     return 0;

    }
   }else{
    //oikeastaan turha, sillä viesti tallennettu puskuriin jo
   }


   return 1;


}

int CServer::init(int portnum){
  portnumber=portnum;

  idlelooptimer=1;

  return 1;
}

void CServer::processListen(){
  //
  boost::posix_time::seconds mytimer(idlelooptimer);

  while(running){
    if(getMessage()){
      //uusi yhteys saatu

    }else{
      boost::this_thread::sleep(mytimer); //jos ei tule viestiä niin odotellaan hetki

    }


  }


}

void CServer::processClients(){
  //
  int i;                //perusslooppimuuttuja
  int isnotidle;        //muuttuja tarkastelemaan onko kierroksen aikana tapahtunut mitään
  boost::posix_time::seconds mytimer(idlelooptimer);

  //time_t curtime=time(NULL);


  while(running){
    isnotidle=0;

    for(i=0;i<clientdeviceconnections.size();i++){ //käydään kaikki Anturikeskitin-yhteyksien oliot läpi

      if(clientdeviceconnections[i].getTimerState()){ //onko aika pyytää uudet paketit
        clientdeviceconnections[i].sendValuesRequest();//lähetetään pyyntö anturidatasta
        clientdeviceconnections[i].stampTime();//leimataan aika
        isnotidle=1;

      }
```

```
      if(clientdeviceconnections[i].getNumMessages()>0){ //onko viestejä
        clientdeviceconnections[i].processMessageQueue(); //käydään läpi seuraava viesti jonossa
        isnotidle=1;
      }




      //CClientDeviceConnection-jako request- ja sensordataviestehin
      if(clientdeviceconnections[i].getNumDataMessages()>0){ //onko dataviestejä
          database.saveContainer(clientdeviceconnections[i].getDataSet()); //tallennetaan seuraava dataviesti tietokannan
datajonoon
        isnotidle=1;
      }

      if(!isnotidle) boost::this_thread::sleep(mytimer); //jos ei ole tekemistä->odotellaan hetki

    }


  }


}

void CServer::processDatabase(){
  //tietokannan toiminnot
  boost::posix_time::seconds mytimer(idlelooptimer);

  while(running){
    if(!database.isEmpty()){ //onko datajonossa tavaraa

      database.saveData(); //tallennetaan seuraava dataset tietokantaan

    }else{

      boost::this_thread::sleep(mytimer); //jos ei ole tekemistä, odotetaan hetki
    }
  }
}

void CServer::run(){


  while(running){
    //jotain

  }
  stop();

}

void CServer::start(){
  running=1;
  database.init();
  listenthread=boost::thread(boost::bind(&CServer::processListen,this));
  clientthread=boost::thread(boost::bind(&CServer::processClients,this));
  dbthread=boost::thread(boost::bind(&CServer::processDatabase,this));
  establish(portnumber);
  run();

}

void CServer::stop(){
  join_all();
  close(serversocket);

}
```

yleinen yhteysolio määrittely – cclientconnection.h

```
#ifndef CCLIENTCONNECTION_H
#define CCLIENTCONNECTION_H


#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netdb.h>
#include<list>
#include<string>


class CClientConnection
{
  public:
    int init(int s){clientsocket=s;quit=0;}
    int init(sockaddr_in fromaddr, int sock);
    void handle_connection();
    void stop();
    int getClientID(){return id;}
    int getSocket(){return clientsocket;}
    int getQuitStatus(){return quit;}
    int receiveMsg(unsigned char *msg); //talletetaan saatu viesti listaan

    int getNumMessages(){return messages.size();}
    int isConnection(sockaddr_in fromaddr);

  protected:
    int id;
    int clientsocket;
    int quit;




    std::list<std::basic_string<unsigned char> > messages;//viesteille fifo-puskuri

    struct sockaddr_in clientaddr; //tämä oikeastaan turha sillä kaikki tapahtumat UDP:ssa tapahtuu samasta socketista..
helpottaa tosin käyttöä

};

#endif // CCLIENTCONNECTION_H
```

toiminta – cclientconnection.cpp

```
#include "cclientconnection.h"
#include<string>
#include"configs.h"
#include"misc.h"
#include <string.h>


#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>


int CClientConnection::receiveMsg(unsigned char * msg)
{

  //jotta ei pysähdytä ensimmäiseen 0 tavuun;
  std::basic_string<unsigned char> tmpmsg;//
  for(int i=0;i<MAXMSGLEN;i++){
```

```
    tmpmsg.push_back(msg[i]);

  }

  messages.push_back(tmpmsg);//pistetään viesti viestipuskuriin



}

int CClientConnection::init(sockaddr_in fromaddr,int sock)
{
  memcpy((void *)&clientaddr,(void *)&fromaddr,sizeof(struct sockaddr_in));//kopioidaan osoitetieto talteen
  clientsocket=sock;
  quit=0;
}

int CClientConnection::isConnection(sockaddr_in fromaddr) //onko viesti kyseisen olion yhteydelle
{
  if(fromaddr.sin_addr.s_addr==clientaddr.sin_addr.s_addr){
    return 1;
  }

  return 0;
}
```

laiteyhteyden olio
määritelmä – cclientdeviceconnection.h

```
#ifndef CCLIENTDEVICECONNECTION_H
#define CCLIENTDEVICECONNECTION_H

#include "cclientconnection.h"
#include <string>
#include <vector>
#include <list>
#include "cdevicedataset.h"
#include <ctime>


#include "configs.h"

class CClientDeviceConnection :public CClientConnection
{
  public:

    void sendValuesRequest();//lähetetään datapyyntö
    int processMessageQueue(); //käsitellään viestijonoa
    int getNumDataMessages(){return sensordatamessages.size();}
    void stampTime(); //leimataan aika
   void setTimer(int t){timer=t;} //asetetaan ajastin sille kuinka usein Anturikeskittimeltä pyydetään dataa
    int getTimerState(); //onko aika pyytää uutta dataa
    CDeviceDataSet getDataSet(); //palautetaan sensordatamessages-puskurista CDeviceDataSet-olio
    void sendRegOk();

  private:
    std::list<std::basic_string<unsigned char> > sensordatamessages; //dataviestit
    std::list<std::basic_string<unsigned char> > requestmessage; //muut viestit, ei käytössä

    time_t lasttime;
    int timer;


};

#endif // CCLIENTDEVICECONNECTION_H
```

toiminta – cclientdeviceconnection.cpp

```
#include "cclientdeviceconnection.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include<sstream>
#include<iomanip>
#include "type.h"


void CClientDeviceConnection::requestDeviceInfo()
{

}

CDeviceDataSet CClientDeviceConnection::getDataSet(){
  CDeviceDataSet tmp;
  unsigned char farray[4];
  unsigned int numnodes,nummeas;//antureiden määrä,väliaikainen muuttuja mittausten määrälle

  float *ftmp;
  float favg;
  unsigned int fmax,fmin;

  floatbyte fb;

  std::basic_string<unsigned char> tempstring=sensordatamessages.front();

  //luetaan ASPv0-viestistä ID ja tallennetaan se tekstinä(Heksamuotoiltuna)
  std::ostringstream oss;
  oss.str("");
  for(int i=0;i<8;i++){
    oss<<std::hex<<(int)tempstring.c_str()[i+1];
  }
  tmp.setDeviceID(oss.str());



  //luetaan antureiden määrä
  numnodes=(int)tempstring.c_str()[9];
  //luetaan mittausten määrä
  nummeas=(unsigned int)((tempstring.c_str()[LEN_HEADER-2] <<8)|tempstring.c_str()[LEN_HEADER-1]);

  tmp.setNodes(numnodes);
  tmp.setMeas(nummeas);


  //luetaan jokaisen XBee-moduulin "jalan" anturidata ja talletetaan se CDeviceDataSet-tyyppiseen olioon
  for(int i=0;i<numnodes;i++){

     fmin=(unsigned int)((tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)]<<8)|tempstring.c_str()[LEN_HEADER+
(i*LEN_NODEDATA)+1]); //luetaan minimiarvo
                  fmax=(unsigned     int)((tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)+2]<<8)|tempstring.c_str()
[LEN_HEADER+(i*LEN_NODEDATA)+3]);//luetaan maksimiarvo

    //luetaan float-tyyppinen keskiarvo
    fb.b[0]=tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)+4];
    fb.b[1]=tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)+5];
    fb.b[2]=tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)+6];
    fb.b[3]=tempstring.c_str()[LEN_HEADER+(i*LEN_NODEDATA)+7];



    favg=fb.f;


    tmp.addValues(fmin,fmax,favg);


  }
```

```
  sensordatamessages.pop_front(); //poistetaan puskurista/jonosta ensimmäinen viesti
  return tmp;

}

int CClientDeviceConnection::processMessageQueue(){
  std::basic_string<unsigned char> tmp=messages.front();
  messages.pop_front();
  unsigned char temp;

  temp=(unsigned char)tmp.c_str()[0];

  if(temp==STARTBYTE){ //onko kyse hyväksytystä viestistä, & onko kyse paketista jossa on antureiden arvoja FCh
    temp=(unsigned char)tmp.c_str()[1];

      sensordatamessages.push_back(tmp);//pistetään viestioikeaan jonoon




  }



}

void CClientDeviceConnection::sendValuesRequest()
{
  unsigned char buf[4];
  buf[0]='R';
  buf[1]='E';
  buf[2]='Q';
  buf[3]='\0';

  sendto(clientsocket, buf, sizeof(buf)+1,0,(struct sockaddr *)&clientaddr,sizeof(clientaddr));

}

void CClientDeviceConnection::sendRegOk()
{
  unsigned char buf[4];
  buf[0]='R';
  buf[1]='O';
  buf[2]='K';
  buf[3]='\0';

  sendto(clientsocket, buf, sizeof(buf)+1,0,(struct sockaddr *)&clientaddr,sizeof(clientaddr));
}

void CClientDeviceConnection::stampTime()
{
  lasttime=time(NULL);
}

int CClientDeviceConnection::getTimerState()
{
  if(lasttime+timer<time(NULL)){
    return 1;

  }else{
    return 0;
  }
}
```

mittausdatan säilöntäolio

määrittely – CDeviceDataSet.h

```
#ifndef CDEVICEDATASET_H
#define CDEVICEDATASET_H

#include<string>
#include<vector>

class CDeviceDataSet
{
  public:
    std::string getDeviceID() {return deviceid;} //palautetaan  ID
    int getNumNodes(){return numnodes;} //palautetaan nodejen määrä
    std::vector<float> getValues(){return values;} //palutetaan float-tyyppinen vektori jossa datasisältö
    int getNumMeas(){return nummeasures;} //palautetaan mittausten määrä
    void setNodes(int i){numnodes=i;} //asetetaan nodejen määrä
    void setDeviceID(std::string devid){deviceid=devid;}//asetetaan ID
    void setMeas(int i){ nummeasures=i;} //asetetaan mittausten määrä
    void setNumNodes(int i){numnodes=i;}//asetetaan nodejen määrä
    void addValues(float,float,float); //lisätään vektoriin arvot minimi,maksimi,keskiarvo


  private:
    std::string deviceid; //ID
    int nummeasures;//mittausten määrä
    std::vector<float> values; //arvot
    int numnodes; //nodejen(=antureiden) määrä

};

#endif // CDEVICEDATASET_H
```

toiminta – CDeviceDataSet.cpp

```
#include "cdevicedataset.h"
void CDeviceDataSet::addValues(float a, float b, float c){
  values.push_back(a);
  values.push_back(b);
  values.push_back(c);



}
```

tietokantaolio

määrittely – cdatabase.h

```
#ifndef CDATABASE_H
#define CDATABASE_H

#include<string>
#include<list>
#include<vector>

#include<mysql++/mysql++.h>

#include"cdevicedataset.h"

class CDatabase
{
  public:
    void init();//initialisoidaan tietokantayhteys
    int connect(); //yhdistetään tietokantaan
    void disconnect();//katkaistaan yhteys tietokantaan
    bool isEmpty(){return databuffer.empty();} //onko datapuskuri tyhjä
    int saveContainer(CDeviceDataSet dataset); //lisätään puskuriin dataa

    void saveData();//tallennetaan dataa tietokantaan
```

```
  private:
    //muuttujia yhdistämiseen
    std::string host;
    std::string user;
    std::string pass;
    std::string db;


    std::list<CDeviceDataSet> databuffer; //datapuskuri

    mysqlpp::Connection conn; //tietokantayhteyden olio



};

#endif // CDATABASE_H
```

toiminta – cdatabase.cpp

```
#include "cdatabase.h"
void CDatabase::init(){
  //oikeasti nämä tiedot pitäisi ottaa esim konfiguraatiotiedostosta.
  host="127.0.0.1";
  user="root";
  pass="einytkerrotapassuaomallepalvelimelle";
  db="sensordata";
}

int CDatabase::connect(){
  conn.connect(db.c_str(),host.c_str(),user.c_str(),pass.c_str());
  if(conn.connected()){
      return 1;
  }else {
      return 0;

  }


}

void CDatabase::disconnect(){
  conn.disconnect();



}

int CDatabase::saveContainer(CDeviceDataSet dataset){
  databuffer.push_back(dataset);




}

void CDatabase::saveData(){
    std::string tmpquerystring;


    int ok=0,connectloop=0;
```

```
int dbid;
std::cout <<"db_DBG\n";
CDeviceDataSet tmp=databuffer.front(); //poimitaan jonosta ensimmäinen datapaketti

//tarkistetaan onko yhteys tietokantaan päällä. mikäli ei ole, koitetaan yhdistää 5 kertaa
while(ok==0 && connectloop<5){
  if(conn.connected()){
    ok=1;

  }else{

    connect();

    connectloop++;
  }

}

if(ok){ //onko yhteys



  mysqlpp::Query query=conn.query();//luodaan mysqlpp::Query-olio ja liitetään se yhteyteen conn
  mysqlpp::StoreQueryResult res;

  std::vector<float> values;
  int nummeas;

  values=tmp.getValues();
  nummeas=tmp.getNumMeas();

  query << "select id,deviceid from devices where deviceid=\'"<<tmp.getDeviceID()<<"\'";

  std::cout <<"db_DBG_bsaveres\n";
  res=query.store();



  if(res.num_rows()!=0){ //tarkistetaan löytyykö kyseinen tunniste tietokannasta
    //moduulin tunnus löytyy tietokannasta -> lisätään arvot viitteineen tietokantaan

    for(int i=0;i<tmp.getNumNodes();i++){
    query<< "insert into measurement (valmax,valmin,valavg,nummeas,node,refdevice,time)"
      <<"values(\'"<<values[i*3]<<"\',\'"<<values[i*3+1]<<"\',\'"<<values[i*3+2]<<"\',"<<nummeas<<","<<i<<"
,"<<res[0][0]<<",NOW())";
    query.exec();

    }


  }else{
      //moduulin tunnusta ei löydy tietokannasta->lisätään tietokantaan tunniste ja lisätään arvot viitaten uuteen
tunnisteeseen
    //luodaan tunniste
    std::cout <<"db_DBG_bcreatedevice\n";
    query<<"insert into devices (deviceid) values(\'"<<tmp.getDeviceID()<<"\')";
    query.exec();

    //haetaan moduulitunnisteen perusteella tietokannan tunniste johon viitataan

    query << "select id,deviceid from devices where deviceid=\'"<<tmp.getDeviceID()<<"\'";
    res=query.store();

    std::cout<<res[0][0]<<std::endl;



    for(int i=0;i<tmp.getNumNodes();i++){

    query<< "insert into measurement (valmin,valmax,valavg,nummeas,node,refdevice,time) "
```

```
        <<"values(\'"<<values[i*3]<<"\',\'"<<values[i*3+1]<<"\',\'"<<values[i*3+2]<<"\',"<<nummeas<<","<<i<<"
,"<<res[0][0]<<",NOW())";
       query.exec();

       }

     }



   }else{//ei yhteyttä
     //TODO: kirjoitetaan vaikka tiedostoon jos ei tietokantaan

   }
 databuffer.pop_front(); //poistetaan käsitelty CDeviceDataSet puskurista

}
```

ylijäämä joka ei sisälly mihinkään olioon

määrittely – misc.h

```
#ifndef H_MISC
#define H_MISC

#include<iostream>
#include<sstream>

//palautetaan std::string unsigned char arraysta on vain sitä varten olemassa ettei std::string ota vastaan unsigned
char-tyyppistä dataa
std::string uctostr(unsigned char *buffer,int len);
#endif
```

toiminta – misc.cpp

```
#include<iostream>
#include<sstream>
std::string uctostr(unsigned char *buffer,int len){
  std::ostringstream oss;
  for(int i=0;i<len;i++){
    oss << static_cast<unsigned char>(buffer[i]);

  }
  return oss.str();
}
```

lisämäärittelyt

config.h

```
#ifndef CONFIGS_H
#define CONFIGS_H

#define MAXHOSTNAME 200        // maksimipituus isännän nimelle
#define PORTNUM 31337          // kuunneltavan portin numero
#define MAXMSGLEN 256          // suurin lähetettävä/vastaanotettava tavumäärä/viesti
```

```
const int DEVICE_CONNECTION=2;  // tunnus laiteyheydelle
const int CONFIG_CONNECTION=1;  // tunnus konfigurointiyhteydelle

//ASP-kehyksen purkuun liittyvät vakiot

const int LEN_STARTBYTE=1; //olennainen vain datan liikutuksessa ja erikoistapauksissa
const int LEN_ID=8;
const int LEN_NUMNODES=1;
const int LEN_MEAS=2;
const int LEN_NODEDATA=8;// 2* int + float
const int LEN_HEADER=LEN_STARTBYTE+LEN_ID+LEN_NUMNODES+LEN_MEAS;

const unsigned char STARTBYTE=0xFC;

#endif // CONFIGS_H
```

tyypit – type.h

```
typedef union Floatbyte{
  float f;
  unsigned char b[4];

} floatbyte;
```

Liite 3. Anturikeskittimen kytkentä