FLASH-MUISTIN KÄYTTÖ ANTURIVERKON DATAN TALLENTAMISESSA

Markku Meskanen

Opinnäytetyö
Kesäkuu 2012
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

MARKKU MESKANEN:
Flash-muistin käyttö anturiverkon datan tallentamisessa.

Opinnäytetyö 29 sivua
Kesäkuu 2012

Langattoman sensoriverkon on pystyttävä siirtämään verkon mittaustulokset jollekin palvelimelle, josta mittaustuloksia voidaan tarkastella ja missä niitä voidaan säilöä. Sensoriverkko voidaan liittää palvelimeen usealla eri tavalla ja opinnäytetyöni tavoitteena oli kehittää uusi tapa toteuttaa tämä tiedonsiirto.

Toteutuksessa langaton sensoriverkko luo mittalaitteita väliasemina käyttäen yhteyden mittalaitteeseen, joka on yhteydessä palvelimeen Modbus-väylän kautta. Modbus-väylään yhdistettävän mittalaitteen tulee pystyä säilyttää koko mittalaiteverkon viimeisimpiä mittaustuloksia flash-muistissaan ja pyydettäessä lähettää mittaustulokset edelleen palvelimelle.

Työn olennaisin osa oli suunnitella tapa, miten flash-muistia voidaan käyttää tehokkaasti mittaustulosten väliaikaiseen säilömiseen päätelaitteessa. Suunnittelussa tuli ottaa huomioon asioita kuten flash-muistin kuluminen toistuvissa kirjoituksissa ja luku- ja kirjoitusoperaatioiden vaatima aika.

Laiteohjelmiston kehittämiskielenä käytetään C++ -ohjelmointikieltä.

Asiasanat: langaton sensoriverkko, flash-muisti, modbus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information technology
Software development

MARKKU MESKANEN:
The usage of flash memory for storing data on a sensor network

Bachelor's thesis 29 pages
June 2012

A wireless sensor network uses each of the sensors attached to the network as access points in order to create a connection to the server which stores the acquired measurements. There are many ways to connect a wireless sensor network to a server and the object of this thesis was to create a new one using the Modbus serial communications protocol.

One of the sensors in the network is converted to be connected directly to the Modbus network and act as a data storage that collects and stores the latest measurements from each of the sensors in the network and transmits the measurements to the server when requested to.

The most important part of the task was to find a way to use the flash memory efficiently. There are a few things that has to be regarded. For example the wear of the flash memory on repeated write cycles and the time it takes to write and read data from the memory.

The program is written in C++ programming language.

Key words: wireless sensor network, flash memory, modbus

SISÄLLYS

LYHENTEET JA TERMIT

IDE	Integrated development enviroment. Ohjelmointiympäristö
Mcc18	MPLAB C Compiler for PIC18 MCUs
ID	Identification. Tunnus, jolla laitteet ja anturit voidaan yksilöidä.
Flash-muisti	Puolijohdemuisti, joka voidaan sähköisesti tyhjentää ja uudelleenohjelmoida.
Lähdekoodi	Ohjelman luettava koodi

1. JOHDANTO

Langattoman sensoriverkon toiminta perustuu usean mittalaitteen muodostamaan verkkoon, jossa mittalaitteet muodostavat yhteyden tukiasemaan toistensa kautta. Mittaustulokset toimitetaan tukiaseman kautta palvelimelle, joka kerää mittalaitteilta saatuja mittaustuloksia.

Tavoitteena opinnäytetyössäni oli suunnitella ja toteuttaa sensoriverkkoon päätelaite, joka toimisi tukiaseman tavoin. Päätelaitteen tulisi olla ohjelmallisesti muunnettavissa tavallisesta verkon mittalaitteesta ja liitettävissä Modbus-väylään, mistä palvelin pyytää päätelaitteelta verkon mittaustuloksia.

Suunnitteluvaiheessa tuli huomioida kohdelaitteen ominaisuuksien ja Modbus-väylän asettamat rajoitukset toteutukselle.

Opinnäytetyö on jaettu kolmeen osaan. Ensimmäisenä tarkastellaan käytössä olleita laitteita ja kehityksessä käytettyjä sovelluksia. Laitteiston ja ohjelmistojen esittelyn jälkeen käsitellään suunnitteluvaiheen tärkeintä vaihetta. Miten mittalaitteiden flash-muistia voidaan hyödyntää mahdollisimman tehokkaasti sensoriverkon mittaustulosten väliaikaiseen tallentamiseen. Viimeisenä käydään läpi itse ohjelmatoteutus. Osiossa pyritään selventämään ja perustelemaan, miksi, ja miten, kukin ohjelman toiminnallisuus on toteutettu.

2. KÄYTETYT OHJELMISTOT JA LAITTEISTO

Tässä luvussa käsitellään työn tekemiseen käytettyjä sovelluksia, kehitystyökaluja ja laitteistoa. Lähdekoodin kirjoittamiseen käytettiin Notepad++ ohjelmistoa ja toteutettu sovellus käännettiin MPLAB PIC18 -perheen mikropiireille tarkoitetulla kääntäjällä. Kääntäminen suoritettiin Cygwin sovelluksessa. Käännetty sovellus ohjelmoitiin laitteelle käyttäen MPLAB IDE v8 -sovellusta yhdessä Microchipin PICkit 3 ohjelmointi- ja testauslaitteen kanssa.

1.1 Ohjelmistot

Työn tekemiseen käytettiin Microchipin MPLAB IDE v8 ohjelmointiympäristöä, Microchipin MPLAB C-kääntäjää PIC18 mikrokontrollereille, Cygwin ajoympäristöä ja Notepad++ tekstieditoria.

1.1.1 Microchip MPLAB IDE v8

Microchipin MPLAB ohjelmointiympäristö on ilmainen integroitu työkalupaketti sulautettujen ohjelmistojen kehittämiseen Microchipin PIC perheen mikrokontrollereille. Se tarjoaa yhtenäisen graafisen käyttöliittymän Microchipin ja kolmansien osapuolien laitteistojenkehitystyökaluille. Ohjelmointiympäristöstä löytyy työkalut muun muassa ohjelmistojen ajamiseen piirille ja ajonaikaiseen virheidenetsintään.

1.1.2 MPLAB C compiler for PIC18 MCUs (MPLAB C18)

MPLAB C18 on monipuolinen ANSI yhteensopiva C-kääntäjä PIC18 perheen kahdeksanbittisille mikrokontrollereille. Kääntäjä on konsolisovellus, joka toimii integroituna osana MPLAB ohjelmointiympäristöä. Kääntäjän hallinta, sekä linkittäjän kustomointi pystytään hoitamaan suoraan ohjelmointiympäristöstä käsin.

1.1.3 Cygwin

Cygwin on Microsoft Windowsille kehitetty yhteensopivuuskirjasto ja joukko ohjelmistoja, jotka mahdollistavat Windowsin käytön Unix-käyttöjärjestelmien tapaan. Monet Unix-ohjelmistot kääntyvät ja toimivat Cygwinin alaisuudessa ilman mitään muutoksia. Cygwiniä ylläpitävät ja kehittävät nykyisin useat eri tahot, muun muassa Red Hat. Cygwin on vapaa ohjelmisto.

1.1.4 Notepad++

Notepad++ on teksti- ja lähdekoodieditori Microsoft Windowsille. Notepad++:n merkittävistä hyödyistä Windowsin omaan Muistioon verrattuna on välilehtiin jaettu käyttöliittymä, mikä mahdollistaa useamman tiedoston pitämisen avoinna samaan aikaan tehokkaammin kuin Muistio. Notepad++ tukee myös useiden eri ohjelmointikielien syntaksin korostusta, mikä helpottaa koodin hahmottamista huomattavasti.

Notepad++ on ilmainen ohjelmisto, joka on ladattu SourceForge.net palvelimelta yli 27 miljoonaa kertaa. Se on myös kahdesti voittanut SourceForge Community Choise Awardin parhaalle kehitystyökalulle.

1.2 LAITTEISTO

Työnä tehty ohjelmisto toteutettiin piirille, jolta löytyy työn tekemisen kannalta merkittävimpinä komponentteina PIC18LF8722 mikrokontrolleri ja M25PX16 flash-muisti. Tiedonsiirtoon käytetään Modbus-sarjaliikenneprotokollaa RS-485-sarjaväylällä.

1.2.1 Microchip PIC18LF8722

PIC18LF8722 on Microchipin PIC18 -perheen kahdeksan bittinen prosessori, jossa on kilotavu EEPROM-, neljä kilotavua SRAM- ja 128 kilotavua flash-muistia.

Sovelluksen toivottiin pystyvän säilömään satoja mittaustuloksia, joten prosessorin sisäinen muisti ei ole mitenkään riittävä. Lisäksi tarvitaan flash-muisti, jolle varsinaiset mittaustulokset tallennetaan.

1.2.2 Micron M25PX16

Laitteessa oli Micronin M25PX16 Fash -muisti. Tallennustilaa muistilla on kaksi megatavua, josta osa saattaisi olla muiden funktioiden käytössä. Flash muisti koostuu 32:sta 64 kilotavun lohkosta (block), jotka koostuvat kuudestatoista neljän kilotavun alalohkosta (subblock). Flash muistille ei voida kirjoittaa uutta tietoa ennen kuin vanha tieto on poistettu. Tiedon poistaminen tapahtuu lohkoittain, alalohkoittain tai tarvittaessa tyhjentämällä koko flash-muisti kerralla (bulk erase).

1.2.3 Microchip PICkit 3

PICkit 3 on ohjelmointilaite Microchipin PIC -perheen prosessoreille. Laite yhdistetään tietokoneen USB-väylään ja sen avulla voidaan ohjelmoida käännetty ohjelma kohdelaitteelle. PICkit 3 tarjoaa myös mahdollisuuden syöttää piirin käyttöjännite ohjelmointilaitteen kautta, jolloin piirille ei tarvitse kytkeä erillistä jännitelähdettä. Ohjelmointilaitteesta löytyy myös 512 kilotavun verran flash muistia Programmer-to-Go toimintoa varten. Programmer-to-Go toiminnossa piiri pystytään ohjelmoimaan uudelleen suoraan ohjelmointilaitteelta ilman yhteyttä tietokoneeseen. Ominaisuus helpottaa erilaisten demonstraatioiden pitämistä, koska esityspaikalle ei tarvitse erikseen tuoda tietokonetta.

MPLAB kehitysympäristön kanssa yhdessä käytettynä PICkit 3 sallii myös reaaliaikaisen ongelmanratkaisumahdollisuuden tarjoamalla debug-moodin, rekisterikartan ja muita hyödyllisiä ominaisuuksia.

1.2.4 Modbus

Modbus on vuonna 1979 julkaistu sarjaliikenneprotokolla, joka on yleisesti käytössä elektroniikkalaitteiden välisessä kommunikoinnissa. Yksi syy Modbusin laajalle käytöl-

le on sen avoimuus ja lisenssimaksuttomuus. Modbus myös siirtää raakadataa ilman laitevalmistajan asettamia rajoituksia.

Jokaiselle Modbus-väylään liitettävälle laitteelle annetaan yksilöllinen osoite. Jokainen laite voi lähettää väylään Modbus-komennon, vaikkakin yleensä vain yksi master -laite tekee niin. Komento sisältää kohdelaitteen Modbus-osoitteen. Kaikki laitteet voivat vastaanottaa annetun komennon, mutta yksilöimällä kohdelaite osoitteen avulla, ainoastaan se suorittaa annetun komennon.

Modbusin julkaisi alunperin Modicon niminen yritys ohjelmoitavien logiikkapiiriensä käyttöön, mutta nykyään Modbusin kehityksestä vastaa Modbus Organization, joka koostuu itsenäisistä käyttäjistä ja Modbus yhteensopivien laitteiden toimittajista.

1.2.5 RS-485

RS-485 on differentiaalinen sarjaväylä, johon voidaan kytkeä useita väylälaitteita samanaikaisesti. Ainoastaan yksi väylälaite pystyy lähettää kerrallaan kolmijohtimisessa RS-485 väylässä. Viisijohtimisessa liikennöinti voi tapahtua kaksisuuntaisesti. RS-485:ttä käytetään teollisuussovelluksissa ja muissa automaatiojärjestelmissä, joissa väylälaitteiden etäisyydet ovat suuria, tarvittavat siirtonopeudet suuria tai ympäristö häiriöinen.

3. FLASH MUISTIN KÄYTTÖ SOVELLUKSESSA

Flash-muistin kirjoitus- ja lukuominaisuuksien asettamien rajoitusten vuoksi työn tekeminen alkoi pohtimalla, miten flash-muisti on mahdollisimman tehokkaasti käytettävissä anturidatan tallentamiseen.

Anturidataa tulee tietyin väliajoin laitteelle, jonka pitää tallentaa tieto ja pystyä pyydettäessä noutamaan viimeisimmät mittausarvot flash-muistista. Flash-muisti rajoittaa käyttöä kahdella tavalla. Jokainen muistipaikka tulee tyhjentää ennen uutta kirjoituskertaa, joten uutta mittaustulosta ei voi aina kirjoittaa vain samaan paikkaan. Flash-muistilla on myös rajallinen määrä kirjoituskertoja, ennen kuin se "kuluu puhki", eli ei pysty enää tallentamaan uutta dataa samaan muistipaikkaan. Useimpien tarjolla olevien flash-muistien luvataan kestävän vähintään satatuhatta kirjoituskertaa. Jos mittaustuloksia tulee kuitenkin esimerkiksi puolen minuutin välein, menisi sadantuhannen kirjoituskerran saavuttamiseen noin kuukausi.

Tässä kappaleessa esitellään joitain mieleen tulleita vaihtoehtoja flash-muistin käyttämiseen.

1.3 Rengaspuskuri

Ensimmäinen ajatus oli muodostaa tietyn kokoinen rengaspuskuri flash-muistiin ja täyttää sitä järjestyksessä uusilla mittaustuloksilla. Rengaspuskurissa muistista muodostetaan sarja muistipaikkoja, joita täytetään alusta lähtien ja puskurin täyttyessä palataan takaisin ensimmäiseen muistipaikkaan.

Sovelluksessa toteutus olisi ollut sellainen, että kun rengaspuskuri tulee täyteen, tyhjennetään alkupäästä lohkoja vanhimman säilytettävän mittaustuloksen lohkoon saakka. Rengaspuskuri-idean ongelmaksi muodostui kuitenkin tilanne, jossa tietyltä anturilta ei olisikaan tullut uutta mittaustulosta, jolloin puskuri täyttyisi siten, että vanhin säilytettävä mittaustulos olisikin lohkossa, jonne pitäisi päästä tallentamaan uusia mittaustuloksia. Myös mittaustulosten noutaminen rengaspuskurista olisi vaikeaa. Jokaisesta mittaustuloksesta olisi täytynyt tallentaa jonkinlainen aikaleima, itse mittaustulos, sekä tieto,

minkä mittalaitteen mistä anturista mittaustulos on lähtöisin. Samalla myös hakuajat mittaustuloksille olisivat kasvaneet, koska koko puskuri olisi käytävä aina läpi, kun halutaan hakea tiettyä tulosta.

Rengaspuskuri-idea hylättiin liian tehottomana ja hankalasti toteutettavana ratkaisuna.

1.4 Aikaleima flash-muistilla

Seuraava idea oli tallentaa jokaisesta mittausarvosta myös aikaleima, sekä mittalaitteen ja anturin ID. Ideassa jokaista mittaustulosta varten varattaisiin flash-muistilta tilaa peräkkäisistä muistipaikoista myös aikaleimalle ja anturin yksilöivälle ID-tiedolle.

Tällä tavalla pystyttäisiin säilömään kaikki tallennettava data flash-muistilla, jolloin RAM-muistiin ei tarvitsisi tallentaa mitään. Tallennus tehtäisiin aina seuraavaan tyhjään paikkaan ja tietoja noutaessa käytäisiin aina läpi koko tallennettu alue etsien uusin tallennettu data kullekin mittalaitteelle ja anturille.

Hyviä puolia tässä tavassa olisi muun muassa se, että kaikki olennainen data olisi flash-muistilla, jolloin RAM-muisti jäisi täysin ohjelmakoodin muun tiedon tallentamiseen. Ja kaikki mittausdata olisi samassa paikassa tallessa. Huonona puolena kuitenkin tuli vastaan hakuaikojen pituus. Jokaista pyydettyä mittausarvoa varten pitäisi koko flash-muistin tallennettu alue käydä lävitse löytääkseen uusimman mittausarvon kullekin anturille. Vaikka lukuajat eivät flash-muistissa hirveän pitkiä olekaan, on pitkät hakuajat kuitenkin mittaustulosten lisääntyessä merkittävä suorituskykyä laskeva tekijä.

Tämäkin idea hylättiin liian tehottomana.

1.5 Mittaustiedon sijainnin tallentaminen RAM-muistille

Kolmannessa ideassa käytettäisiin myös RAM-muistia apuna tiedon tallentamiseen. Ideana oli tallentaa RAM-muistiin taulukko, jonka indeksiä voidaan käyttää kyseessä olevan mittalaitteen ja anturin identifioimiseen. Flash-muistilla yksi lohko varattaisiin

aina yhdelle mittalaitteelle, jolloin mittalaitteen yksi antureiden tiedot löytyisi flash-muistin ensimmäisestä lohkosta.

Jokaisen RAM-muistilla olevan taulukon alkion arvo kertoisi vastaavasti tietyn anturin mittaustuloksen sijainnin flash-muistin kyseiselle mittalaitteelle varatussa lohkossa. RAM-muistilla olisi siis kolme tietoa tallessa. Mittalaitteen ID, anturin ID ja laskuri. Tiettyä kaavaa käyttäen näistä kolmesta tiedosta saataisiin koostettua osoite, missä osoitteessa mittaustulos sijaitsee flash-muistilla.

Tässä ideassa jouduttaisiin kuitenkin käyttämään huomattavasti tilaa RAM-muistilta, joka oli muutenkin jo melko pitkälti käytetty ohjelman muun toiminnallisuuden toteuttamisessa. Hakuajat olisivat kuitenkin huomattavasti nopeammat kuin ideassa, jossa kaikki data tallennettaisiin suoraan flash-muistille. Tieto siitä, missä mittaustulos sijaitsee, nopeuttaa tiedon hakua flash-muistilta, koska silloin koko flash-muistia ei tarvitse käydä lävitse.

Varsinaisessa ohjelmatoteutuksessa käytetään tätä ideaa. Laskurin tallentaminen RAM-muistille on siedettävä haitta lopullisessa toteutuksessa, koska tehokkuusedut ovat huomattavia muihin ideoihin verrattuna.

1.6 Toteutus ohjelmakoodissa

Toteutuksessa tallennetaan RAM-muistille ennalta määrätyn kokoinen taulukko, jonka koko määräytyy sen mukaan, montako mittalaitetta verkossa maksimissaan on, ja miten monta anturia kussakin mittalaitteessa voi maksimissaan olla. Koko flash-muistin ollessa käytössä mittaustulosten tallentamiseen, on mahdollisten mittalaitteiden määrä flash-muistipiirin alalohkojen kokonaismäärä. Koska käytössä olevassa piirissä muisti koostuu 32 lohkosta, joissa kussakin on 16 alalohkoa, tulee mittalaitteiden maksimimääräksi 512 mittalaitetta. Tässä tulee kuitenkin vastaan käytettävissä olevan RAM-muistin rajallisuus. Tallennustavassa flash-muisti ei rajoita yksittäisen mittalaitteen antureiden määrää, ainoastaan itse mittalaitteiden määrää. RAM-muistilla on käytettävissä tilaa kuitenkin vain noin kuudelle sadalle laskurille, jolloin näistä täytyy muodostaa jonkinlainen käyttötilannekohtainen raja-arvo.

Esimerkiksi jos tarvitaan ainoastaan yhtä mittaustulosta kultakin mittalaitteelta, esimerkiksi ainoastaan lämpötilatilastoja, voidaan verkkoon kytkeä kaikkiaan 512 mittalaitetta, joissa kussakin on yksi anturi. Tässä tapauksessa koko flash-muisti tulee käyttöön. Jos kuitenkin tarvitaan mittaustuloksia esimerkiksi lämpötilasta, hiilidioksidiarvoista ja kosteudesta, tulee kultakin mittalaitteelta kolmea mittaustulosta. Kolmella anturilla rajoittavaksi tekijäksi muodostuukin RAM-muistin tila, jonne mahtuu ainoastaan noin kuuden sadan anturin laskurit. Tässä tilanteessa verkossa voi olla siis noin kaksi sataa mittalaitetta.

4. OHJELMATOTEUTUS

Ohjelmatoteutuksessa tarvittavat ajurit flash-muistille oli valmiina ja Modbus-väylän kanssa kommunikointiinkin määriteltynä, joten työn tekemissä pystyttiin keskittymään täysin annettuun tehtävään. Ensimmäinen vaihe oli toteuttaa flash-muistille tallennus, sekä tallennuskohdan kartoittaminen. Tämän jälkeen toteutettiin flash-muistilta lukeminen. Kirjoitus- ja lukufunktioiden lisäksi tuli toteuttaa tapa, jolla flash-muistilla tietyssä alalohkossa oleva data saadaan säilytettyä myös silloin, kun alalohko joudutaan alustamaan sen täyttyessä.

Muistitoimintojen jälkeen toteutettiin Modbus-pyyntöihin vastaamiseksi funktiot, jotka noutavat pyydetyn tiedon flash-muistilta, pakkaavat sen Modbus-väylän haluamaan muotoon ja lähettävät viestin eteenpäin.

Tässä kappaleessa esitellään kunkin funktion perustoiminnallisuus ja ajatusprosessi funktion synnyn takana.

1.7 Laskurin tallentaminen ja lukeminen

Mittausarvojen tallentamiseen tarvitaan kahdenlaisia muistioperaatioita. Varsinaisen mittausarvon ja aikaleiman tallentaminen flash-muistille, sekä mittausarvon tallennuspaikan säilömiseen laskurin tallentaminen SRAM-muistilla olevaan taulukkoon. Mittausarvojen palauttamiseksi flash-muistilta tarvitaan myös oma operaatio.

SRAM-muistille luodaan maksimissaan kuusisataa alkioinen 16-bittisiä kokonaislukuja sisältävä taulukko, johon laskurin arvo tallennetaan. Jokainen taulukon alkio itsessään kertoo minkä mittalaitteen ja minkä mittalaitteen anturin mittausarvo alkion osoittamassa muistipaikassa on tallennettuna.

Jos käytössä on esimerkiksi kolme mittalaitetta, joissa kussakin on maksimissaan kolme anturia, näyttäisi laskuritaulukko taulukon 1 mukaiselta.

TAULUKKO 1. Laskuritaulukon rakenne

Indeksi	Sisältö
0	1. mittalaitteen, 1. anturin mittausarvon sijainti flash-muistin lohkossa
1	2. mittalaitteen, 1. anturin mittausarvon sijainti flash-muistin lohkossa
2	3. mittalaitteen, 1. anturin mittausarvon sijainti flash-muistin lohkossa
3	1. mittalaitteen, 2. anturin mittausarvon sijainti flash-muistin lohkossa
4	2. mittalaitteen, 2. anturin mittausarvon sijainti flash-muistin lohkossa
5	3. mittalaitteen, 2. anturin mittausarvon sijainti flash-muistin lohkossa
6	1. mittalaitteen, 3. anturin mittausarvon sijainti flash-muistin lohkossa
...	...

Laskuritaulukon indeksin määrää kolme tietoa. Mittalaitteen tunnistava id_node, anturin tunnistava id_sensor ja antureiden maksimimäärä MAX_SENSORS. Laskurin arvon määrää seuraavan tyhjän paikan sijainti flash-muistilla. Taulukkoon ei siis kirjaimellisesti tallenneta tietoa siitä, missä yksittäinen mittaustulos sijaitsee flash-muistilla, vaan tallennetun datan sijaintia seuraava ensimmäinen tyhjä muistipaikka. Käytännössä tämä tulee huomioida flash-muistille kirjoittaessa ja sieltä mittausarvoa palauttaessa. Hahmottamisen helpottamiseksi käytän kuitenkin laskurin arvosta sanamuotoa mittausarvon sijainti.

Aina kun uusi mittaustulos tallennetaan flash-muistille, tallennetaan myös sen sijainnin kertova laskurin arvo taulukkoon. Laskuritaulukon indeksi määräytyy siten, että anturin ID kerrotaan antureiden maksimimäärällä, mihin lisätään kyseisen mittalaitteen ID.

```
void counterWrite(int16_t *counter, uint16_t *id_node,
          uint16_t *id_sensor)
{
    counterArray[*id_sensor * MAX_SENSORS + *id_node]
          = *counter;
}
```

Vastaavasti mittausarvon lukeminen flash-muistilta lähtee liikkeelle siitä, että mittausarvon sijainti flash-muistilla luetaan laskuritaulukosta käyttäen samoja kolmea tietoa laskurin paikantamiseen.

```
void counterRead(uint16_t *id_node, uint16_t
          *id_sensor, int16_t *counter)
{
    *counter = counterArray[(*id_sensor *
          MAX_SENSORS) + *id_node];
}
```

Koska kaikista mittalaitteista ei välttämättä ole tullut mittausarvoja siihen mennessä, kun mittausarvoja pyydetään, alustetaan koko laskuritaulukko arvoon -1. Jos `counterRead`-funktio palauttaa arvon -1, tiedetään, ettei kyseisen indeksin mukaiselta anturilta ole mittausarvoa.

Sekä mittalaitteiden, että antureiden indeksointi alkaa nollasta. Ensimmäisen mittalaitteen node_id on siis nolla. Mittalaitteiden ensimmäisen anturin id_sensor on myös 0. Kun indeksointi aloitetaan nollasta, pitää laskuritaulukon indeksoimiseen käytettävä laskukaava paikkansa. Taulukon yksi esimerkin mukaisesti: maksimissaan kolme mittalaitetta (MAX_SENSORS = 3). Toisen mittalaitteen (id_node = 1) toinen anturi (id_sensor = 1) löytyy laskuritaulukon indeksistä 1 * 3 + 1 = 4.

1.8 Flash-muistille kirjoittaminen ja sen lukeminen

Flash-muistin ajurit oli valmiiksi kirjoitettuna ja käytettävissä. Toteutuksessa käytettiinkin ajureiden valmiita kirjoitus- ja lukufunktioita.

```
uint8_t M25PX16_read_bytes(uint32_t addr, /*@out@*/
          uint8_t* values, uint8_t size);

uint8_t M25PX16_write_bytes(uint32_t addr, uint8_t*
          values, uint8_t size);
```

Molemmissa operaatioissa arvot ovat uint8_t* muodossa ja mittaustulokset ovat liukulukuja. Operaatiot onnistuvat helpoiten käyttäen Ufloat32 -tyyppisiä muuttujia, jolloin liukulukumuuttujia ei tarvitse alkaa itse pilkkomaan tavun mittaisiksi.

```
typedef union Ufloat32_t
{
    unsigned char ub[4];
    unsigned short us[2];
    unsigned long l;
```

```
    floatRaw32 f;
} Ufloat32;
```

Ennen jokaista flash-muistia käyttävää komentoa on varmistettava, että flash-muisti on valmis ottamaan vastaan uuden komennon. Flash-muistin valmius tarkistetaan ohjelmassa käyttäen `waitFlashReady` -funktiota, joka käyttää hyödykseen flash-muistin ajureista löytyvää funktiota `M25PX16_is_ready()`, joka tarkastaa onko flash-muisti jo tekemässä jotain.

```
void waitFlashReady(void)
{
    while (M25PX16_is_ready() == 0)
    {
        delayUs(10);
    }
}
```

Jos flash-muistilta saadaan vastauksena, ettei se ole vielä valmis ottamaan vastaan uutta komentoa, odotetaan kymmenen mikrosekuntia ja kysytään uudelleen, joko voi uuden käskyn lähettää.

1.8.1 Flash-muistille kirjoittaminen

```
void flashWrite(uint16_t *id_node, uint16_t
          *id_sensor, Ufloat32 *measurement);
```

Ohjelmassa flash-muistille kirjoittaminen tapahtuu aina samassa järjestyksessä. Funktiokutsussa annetaan kirjoitusfunktiolle mittalaitteen ID, anturin ID ja mittausarvo. Ensimmäinen vaihe kirjoittamisessa on etsiä kyseiselle mittalaitteelle varatusta alalohkosta seuraava vapaa kirjoituspaikka laskuritaulukkoon tallennettujen tietojen perusteella. Seuraava vapaa kirjoituspaikka on suurin tallennettu arvo laskuritaulukossa kyseisen mittalaitteen antureille.

```
for (i = 0; i < MAX_SENSORS; i++)
{
    new_value = counterArray[(i * MAX_SENSORS) +
          *id_node];
    if (new_value > highest_value)
    {
        highest_value = new_value;
    }
}
```

Kun kirjoituspaikka löytyy, varmistetaan, että flash-muisti on valmiina ottamaan vastaan uuden käskyn. Kun flash-muisti on valmis, kutsutaan kirjoitusoperaatiota `M25PX16_write_bytes`, jolle annetaan kirjoitusosoite `addr`, mittausarvo `values`, sekä kirjoitettavien tavujen määrä `size`. Kirjoitusosoite `addr` lasketaan mittalaitteen ID:n, mittalaitteelle varatun tallennustilan koon ja laskuritaulukosta saadun ensimmäisen vapaan kirjoituspaikan perusteella.

```
waitFlashReady();
(void)M25PX16_write_bytes(((*id_node * SUBBLOCK) +
          highest_value), measurement->ub, 4);
highest_value += 4;
```

Samalla kirjoitetaan mittausarvon perään myös aikaleima, jota tarvitaan tarkistaessa, onko mittausarvo ehtinyt vanhentua palvelimen sitä pyytäessä. Mittausarvoille voidaan siis määrittää myös vanhenemisaika, jonka ylityttyä palvelimelle lähetetään vastauksena vain tieto siitä, että mittausarvo on epäkelpo.

```
timer.lr = RTimer_getSeconds16();
waitFlashReady();
(void)M25PX16_write_bytes(((*id_node * SUBBLOCK) +
          highest_value), timer.br , 2);
highest_value += 2;
```

Näiden kirjoitusoperaatioiden jälkeen kirjoitetaan vielä laskuritaulukkoon laskurin arvo, jolla pystytään paikantamaan juuri kirjoitettu mittausarvo flash-muistilta.

```
counterWrite(&highest_value, id_node, id_sensor);
```

Näiden kolmen operaation jälkeen flash-muistilla on tallessa mittausarvo ja aikaleima, sekä RAM-muistilla niiden sijainti flash-muistilla.

1.8.2 Flash-muistilta lukeminen

Flash-muistin lukemiseen käytetään `M25PX16_read_bytes` -funktiota, joka tarvitsee parametreina luettavan osoitteen, muuttujan, johon noudettava tieto tallennetaan ja luettavien tavujen määrän.

```
uint8_t M25PX16_read_bytes(uint32_t addr, /*@out@*/
          uint8_t* values, uint8_t size);
```

Lukuoperaatiota kutsutaan ainoastaan silloin kun Modbus-palvelin pyytää laitteelta mittausarvoa. Tästä syystä itse lukufunktiot ovat hyvin yksinkertaisia. Kun Modbus-palvelin pyytää mittaustulosta, haetaan laskuritaulukosta pyydetyn mittalaitteen ja anturin sijainti flash-muistilla. Nämä kolme tietoa tarvitaan flash-muistin lukemiseen. Flash-muistilta luetaan kaksi arvoa: mittausarvo ja aikaleima kyseiselle mittausarvolle.

```
void flashRetrieve(int16_t *counter, uint16_t
          *id_node, Ufloat32 *measurement, USHORTt
          *timestamp)
{
    waitFlashReady();
    (void)M25PX16_read_bytes(((SUBBLOCK *
          *id_node) + *counter - STO
          RE_SIZE), measurement->ub, 4);

    waitFlashReady();
    (void)M25PX16_read_bytes(((SUBBLOCK *
          *id_node) + *counter - STORE_SIZE
          + 4), timestamp->br, 2);
}
```

Kumpaakin lukuoperaatiota ennen varmistetaan, että flash-muisti on valmis ottamaan vastaan lukuoperaatiokäskyn. Noudettavan tiedon osoite saadaan kertomalla mittalaitteen ID mittalaitteen mittaustuloksille varatun alalohkon koolla ja lisäämällä tuloon laskurin arvo. Koska laskurissa on tallennettuna aina mittausarvoa seuraavan tyhjän muistipaikan sijainti, tulee edellisen kaavan tuloksesta vielä vähentää noudettavan datan määrä, jolloin päästään varsinaisen mittaustuloksen alkuun.

1.8.3 Alalohkon täyttyminen ja tietojen säilyttäminen alalohkon alustuksessa

Flash-muistille kirjoitettaessa tarkistetaan aina lopuksi, mahtuuko seuraava mittausarvo vielä samaan alalohkoon. Jos mittaustulos ei mahdu alalohkoon, täytyy kyseisen mittalaitteen uusimmista mittaustuloksista ottaa varmuuskopio, tyhjentää alalohko ja kirjoittaa varmuuskopioidut mittaustulokset uudelleen flash-muistille. Tämän funktion toteuttaminen ja testaaminen oli koko työn toteuttamisen hankalin ja pisin vaihe.

Alalohkon täyttyminen todetaan yksinkertaisella vertailuoperaatiolla. Jos laskurin arvo, eli seuraava vapaa kirjoituspaikka, on suurempi kuin alalohkon koko, josta vähennetään yhden tallennuskerran vaatima tila, alalohko on alustettava.

```
if (highest_value > (SUBBLOCK - STORE_SIZE))
```

Jos alustus tarvitaan, eli jos ehto täyttyy, varmuuskopioidaan kaikki kyseisen mittalaitteen uusimmat mittausarvot RAM-muistilla sijaitseviin väliaikaistaulukkoihin. Koska aikaleima on kokonaisluku ja mittausarvo on liukuluku, päästään helpoimmalla jos molemmille on oma väliaikaistaulukko.

```
for (i = 0; i < MAX_SENSORS; i++)
{
    counter = counterArray[(i * MAX_SENSORS) +
          *id_node];

    if ( counter != -1 )
    {
        flashRetrieve(&counter, id_node, measure
          ment, &retrieved_timestamp);

        tempArrayFloat[i] = measurement->f;
        tempArrayInt[i] = retrieved_timestamp.lr;
    }
}
```

Kun arvot on saatu varmuuskopioitua alustetaan flash-muistin alalohko. Ennen alustusta jälleen varmistetaan, että flash-muisti on valmis ottamaan vastaan alustuskäskyn.
Koska kaikissa mittalaitteissa ei aina ole kaikkia mahdollisia antureita, joudutaan ennen varmuuskopioiden palauttamista flash-muistille varmistaa, oliko kyseiseltä mittalaitteelta edes saatu mittaustuloksia. Tämä tieto on olennainen siksi, että varmuuskopioidut mittausarvot tulee saada oikeiden antureiden kohtiin laskuritaulukkoon. Mittausarvon olemassaolo tarkistetaan yksinkertaisella ehtolauseella, jossa verrataan kutakin laskuriarvoa -1:een, mihin laskuritaulukko aluksi alustettiin. Tarkistusta käytetään muutamassa eri kohdassa, joten sille oli järkevää luoda oma funktio.

```
uint8_t checkValidity(uint16_t *id_node, uint16_t
          *id_sensor, int16_t *counter)
{
    *counter = counterArray[(*id_sensor * MAX_SENSORS)
          + *id_node];

    if(*counter == -1)
    {
        return 0;
    }
    return 1;
}
```

Jokainen väliaikaisiin taulukkoihin tallennettu arvo siis palautetaan takaisin flash-muistille. Alustaessa myös laskureiden arvot kullekin mittausarvolle muuttuvat, joten laskuritaulukkoonkin on päivitettävä uudet arvot. Laskuritaulukon päivittämiseen käytetään yksinkertaisesti tietoa siitä, että mittausarvot palautetaan flash-muistille samassa järjestyksessä kuin ne sieltä luettiin, joten uudet laskureiden arvot tulevat kasvavassa järjestyksessä laskuritaulukkoon. Välistä on kuitenkin jätettävä ne anturit, joilta ei ole alustukseen mennessä saatu mittausarvoja tai joita ei ole kyseisessä mittalaitteessa olemassakaan.

```
for (i = 0; i < MAX_SENSORS; i++)
{
    if (checkValidity(id_node, &i, &counter))
    {
        counter = ((i + 1 - skipped) * STORE_SIZE);
          counterWrite(&counter, id_node, &i);
    }
    else
    {
        skipped++;
    }
}
```

Laskuritaulukko on nyt päivitetty vastaamaan uusia flash-muistille tulevia arvoja. Flash-muistille kirjoitetaan arvot väliaikaisista taulukoista järjestyksessä siten, että kunkin anturin mittaustiedon sijainti vastaa kirjoituksen jälkeen laskuritaulukon arvoja. Jälleen tarkastetaan ensin kunkin anturin kohdalla, onko siltä saatu mittaustuloksia. Jos mittaustulos löytyy, kirjoitetaan anturin indeksin mukainen mittausarvo ja aikaleima väliaikaisista taulukoista flash-muistille

```
for (i = 0; i < MAX_SENSORS; i++)
{
    if (checkValidity(id_node, &i, &counter))
    {
        measurement->f = tempArrayFloat[i];
        waitFlashReady();
        (void)M25PX16_write_bytes((SUBBLOCK * *id_node
            + j), measurement->ub, 4);
        j += 4;

        timer.lr = tempArrayInt[i];
        waitFlashReady();
        (void)M25PX16_write_bytes((SUBBLOCK * *id_node
             + j), timer.br, 2);
        j += 2;
    }
}
```

Näiden operaatioiden jälkeen flash-muistin kyseiselle mittalaitteelle varatun alalohkon alussa on kirjoitettuna kaikkien antureiden uusimmat mittausarvot ja niiden sijainnit löytyvät laskuritaulukosta omilta paikoiltaan. Tämä toteutus sijaitsee flash-muistin kirjoitusfunktion `flashWrite` lopussa.

1.9 Kommunikointi Modbus-väylän kanssa

Koska päätelaitteen on tarkoitus toimittaa mittaustiedot eteenpäin palvelimelle Modbus-väylän kautta, tuli ohjelmaan toteuttaa toiminnallisuus myös siihen, miten mittaustulokset saadaan toimitettua Modbus-väylälle. Laitteen pitää pystyä reagoimaan väylältä saapuviin pyyntöihin noutamalla pyydetyt mittaustulokset flash-muistista ja lähettämällä ne eteenpäin palvelimelle.

Toteutuksessa kommunikointi on järjestetty tarkastamalla onko uusia pyyntöjä tullut, selvittämällä, mitä arvoja pyynnössä haluttiin, noutamalla pyydetyt mittaustulokset flash-muistista ja vastaamalla pyyntöön lähettämällä pyydetyt mittaustulokset palvelimelle.

Toiminnallisuutta kirjoittaessa laitteen Modbus-ajurit, eivät olleet vielä toimintavalmiit, joten toteutus on kirjoitettu tukeutuen ajureiden määrittelydokumentaatioon. Varsinainen Modbus-viestien vastaanotto ja lähetys siis ei ollut vielä toteutettuna ohjelmaa kirjoittaessa.

1.9.1 Modbus-pyynnön tunnistaminen

Laitteen havaitessa uuden pyynnön saapuneen, kutsutaan funktiota `modbusRequestParser`, joka tunnistaa, mitä mittausarvoja palvelimelta tulleessa pyynnössä haluttiin saatavan verkosta.

Pyyntö sisältää tiedon rekisteripaikasta, eli mittalaitteen sijainnista Modbusrekisterissä, sekä tiedon siitä, miten monta tavua tästä rekisteripaikasta eteenpäin tahdotaan saada. Mittalaite ja anturi pystytään tunnistamaan pyynnössä olevasta rekisteriosoitteesta

```
if (*reg_address > 255)
{
    divider = *reg_address / 256;
}
*reg_address -= (256 * divider);

id_node = *reg_address;
id_sensor = divider;
```

Tällä tavalla saaduista mittalaitteen ja anturin tiedoista voidaan palauttaa flash-muistilta pyydetty mittausarvo. Koska pyyntö voi sisältää samanaikaisesti pyynnön useammasta mittaustuloksesta, tai vaikka kaikista mittaustuloksista, tulee viestiin vastatessa ottaa huomioon, miten montaa mittaustulosta pyydettiin.

Kutsumalla silmukassa funktiota, joka palauttaa pyydetyn mittausarvon flash-muistista, jokaista pyydettyä mittausarvoa kohden, saadaan vastattua kaikkiin pyynnössä saatuihin mittaustulosvaatimuksiin.

```
for (i = 0; i < *len; i++)
{
    modbusServe(&id_node, &id_sensor);
    id_node++;
}
```

1.9.2 Modbus-pyynnön toteuttaminen

Kun tiettyä mittalaitetta ja anturia vastaava pyyntö on tunnistettu, noudetaan pyydetty mittausarvo flash-muistista ja tunnistetaan, onko flash-muistista löytynyt mittausarvo validi.

```
counterRead(id_node, id_sensor, &counter);

if ( counter == -1 )
{
    modbusSend(no_value);
    return;
}
else
{
    flashRetrieve(&counter, id_node, &measurement,
          &time);
}

time.lr = RTimer_getSeconds16() - time.lr;
```

```
if (time.lr > TIMEOUT2S)
{
    modbusSend(value_timeout);
    return;
}
modbusSend(measurement);
```

Ensin noudetaan mittalaitetta ja anturi vastaava laskurin arvo laskuritaulukosta. Jos saatu laskurin arvo on -1, ei kyseisellä anturilta ole tullut mittausarvoa. Jos anturille on olemassa mittausarvo, eli laskurin arvo on suurempi kuin nolla, noudetaan mittausarvo flash-muistilta. Kun mittausarvo ja aikaleima on palautettu flash-muistilta, verrataan tallennettua aikaleimaa nykyiseen aikaan. Tällä tarkastetaan, ettei mittausarvo ole vanhentunut. Mikäli mittaustuloksen tallentamisesta on kulunut pidempi aika kuin säädetty timeout -aika sallii, vastataan palvelimelle ennalta määritetyllä timeout -arvolla. Jos arvo löytyy ja aikaleima on tarpeeksi tuore, lähetetään flash-muistilta palautettu mittausarvo eteenpäin.

1.9.3 Modbus-viestin valmistelu

Koska mittaustuloksia voidaan pyytää useampi kerralla, on järkevää myös vastauksessa lähettää kaikki pyydetyt mittausarvot kerralla. Tähän tarkoitukseen on toteutettu taulukko, johon kaikki `modbusServe` -funktiossa noudetut arvot tallennetaan ja lähetetään sitten kerralla palvelimelle.

```
void modbusSend(Ufloat32 value)
{
    if (globalLen > 0)
    {
          dataArray[valuesCounter] = value;
          globalLen--;
          valuesCounter++;
    }
    if( globalLen == 0 )
    {
          valuesCounter = 0;
          //This is where tha actual sending of the
          //data would be.
    }
    return;
}
```

Kun `modbusRequestParser` -funktion silmukka on käyty loppuun, löytyy kaikki pyydetyt mittausarvot `dataArray` -taulukosta ja ne pystytään lähettää eteenpäin palvelimelle.

Tätä vaihetta toteutuksesta ei pystytty testaamaan varsinaisella testilaitteella, koska modbus-ajurit olivat vielä toteuttamatta. Laitteen ulkopuolella testattuna tämä toteutus kuitenkin toimi.

5. POHDINTA

Ohjelmistotekniikassa mielestäni pätee kaikkein parhaiten ohje: Vain tekemällä oppii. Aina kun eteensä saa uuden projektin, varsinkin jos on ollut vähänkään taukoa ohjelmoinnista, tuntuu, että asioita joutuu palauttelemaan mieleen aika paljon. Kuitenkin kun kerran oppii jotain uutta, se on helposti palautettavissa mieleen, eikä asioita tarvitse alkaa opettelemaan uudelleen.

Kun sain opinnäytetyöni aiheeksi päätyneen projektin eteeni, olin aavistuksen pettynyt. Olen tiedostanut jo pidempään haluavani päästä tekemään jotain käyttäjäläheisempää ohjelmistoa työkseni, joten olisin mielelläni tehnyt myös opinnäytetyöni jostain tavoitteitani vastaavasta aiheesta. Nyt kun työ on valmis, täytyy kuitenkin myöntää, että tämä on ollut kaiken kaikkiaan hyvin positiivinen kokemus. Olen oppinut paljon uutta asioista, joihin en välttämättä tule enää koskaan törmäämään, mutta samalla tiedostan osaavani nyt myös nämä asiat.

Opinnäytetyön edetessä opin myös hyvin tärkeän asian ohjelmistokehityksestä. Aikataulut tuppaavat pettämään. Mitä tiukemmat aikataulut asettaa, sitä todennäköisemmin ne myös ylittyvät jossain vaiheessa. Tämän projektin yhteydessä asiaan törmäsi muun muassa niissä tilanteissa, kun jokin valmiina saatu asia ei toiminutkaan toivotulla tavalla ja joutui näitä, itselle vieraita, toteutuksia tutkia ja yrittää löytää niistä, miksi jokin ei toimi oikein.

Opin työn tekemisen aikana paljon laitteistoläheisestä ohjelmoinnista, vaikka kielenä tässä työssä olikin C++. Laitteisto asettaa yllättävän paljon rajoituksia siihen, miten asioita voi toteuttaa. Esimerkiksi ohjelmointikieli ja kääntäjävalinta oli ennalta määrätty laitteiston vaatimuksissa. Kääntäjä vastaavasti vaikutti työssäni esimerkiksi ohjelmointiympäristön valintaan ja tiettyihin syntaksiasioihin, joista ei varsinaisesti ollut spesifikaatioita saatavilla, mutta jotka oppi kyllä kääntäjän antamien virheilmoitusten kautta. Samaten ajoituksista riippuvaisessa sovelluksessa on yllättävän merkitsevää se, miten pitkään funktion suorittaminen kestää.

Myös testaamispuolella opin uusia temppuja. Esimerkiksi tapa, miten tutkia ajonaikaista tehokkuutta oskilloskoopin avulla oli minulle uutta. Se, että laitetaan oskilloskooppi

kiinni piirilevyn loistediodiin, sytytetään kyseinen komponentti tiettyyn funktioon siirtyessä ja katsotaan oskilloskoopista, miten pitkään kyseisen funktion suorittaminen kestää, antoi aivan uuden näkökulman ohjelmiston tehokkuuden testaamiseen.

Työn valmistuttua ohjelma on ollut jonkin aikaa teollisuudessa jo käytössä ja saamani palautteen mukaan toteutus on myös toiminut asetettujen vaatimusten mukaisesti. Toteutus on tarpeeksi tehokas, jottei siitä tule ajoitusten kanssa pullonkaulaa, sekä tarpeeksi vakaata, että laitetta pystytään käyttämään teollisuudessa.

LÄHTEET

MicroChip. 2004. PIC18F8722 Family Data Sheet. [online]. Luettu 01.05.2012. http://kevin.org/frc/PIC18F8722_ds.pdf

Numonyx. 2010. M25PX16 flash memory data sheet. [online]. Luettu 01.05.2012 http://www.micron.com/~/media/Documents/Products/Data%20Sheet/NOR%20Flash/5983M25PX16.ashx

MicroChip. 2012. PICKit 3 In-Circuit Debugger Features. [online]. Luettu 01.05.2012 http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en538340&redirects=pickit3

Flash Memory - Wikipedia. Wikipedia. [online]. Luettu 27.05.2012 http://en.wikipedia.org/wiki/Flash_memory

Modbus Organization. 2012. Modbus. [online]. Luettu 27.05.2012 http://www.modbus.org/