VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Matti Suominen

HAJAUTETUN TIEDONKERUUVERKON SUUNNITTELU

Protokollan ja ohjelmiston suunnittelu sekä toteutus

Tekniikka
2016

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Matti Suominen
Opinnäytetyön nimi	Hajautetun tiedonkeruuverkon suunnittelu
Vuosi	2016
Kieli	suomi
Sivumäärä	35
Ohjaaja	Jukka Matila

Tässä Vacon Oyj:lle tuotetussa tuotekehitysprojektissa tutkittiin ja toteutettiin taajuusmuuttajaan kytkettäville oheislaitteille kustannustehokas tiedonsiirtoprotokolla. Tavoitteena oli kehittää kasvaviin huoltotarpeisiin vastaava toteutus, jolla asiakastyytyväisyys pidettäisiin yllä.

Taajuusmuuttaja on tiedonsiirtoympäristönä häiriöaltis, eikä ympäristönä mahdollista rajattomien fyysisten väylien vetämistä pisteestä toiseen. Työssä tutkittiin olemassa olevia tiedonsiirtoprotokollia ja väyliä, joiden pohjalta päätös kehityksen suunnasta tehtiin.

Projektissa lopputuloksena saatiin prototyyppi, jonka jatkojalostamista jatketaan yhtiön sisällä. Prototyypin fyysinen ympäristö valikoitui jo aiemmin tehtyjen kokeiden ja päätösten pohjalta, mutta myös muita fyysisiä väyliä voidaan jatkossa hyödyntää helpoilla ohjelmistomuutoksilla.

Avainsanat taajuusmuuttaja; tiedonsiirto; Modbus; protokolla; tuotekehitys

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Matti Suominen
Title	Distributed Sensor Network Development
Year	2016
Language	Finnish
Pages	35
Name of Supervisor	Jukka Matila

In this research project which were made to Vacon Ltd were researched and developed in expensive data transmission protocol for frequency converter add on devices. Target was to develop implementation which responses to all time rising service needs to keep customer satisfaction high.

Frequency converter is a challenging environment for communication lines and as an environment doesn't provide possibilities to make new physical connections from point to point. In this project were researched currently active protocols and communication lines. Development line was based on these researches.

Output of this project were prototype which provides possibility to company to continue communication line and protocol development. Prototype's physical environment were selected based on premade tests. This selected physical environment doesn't restrict use of other possible environments and it is easy to change by providing new drivers for software.

Keywords frequency converter; data transferring; Modbus; protocol; R&D

SISÄLLYS

LYHENTEET JA TERMIT

Master	Sarjaliikenteessä verkon hallitsija – isäntä, hallitsee verkon liikenteen master – slave verkossa
Slave	Laite tai ohjelmakoodi joka toteuttaa isännän käskyjä
I2C	Inter-Integrated Circuit, tiedonsiirtoväylä
SDA	I2C:ssä käytettävä sarjamuotoinen dataväylä
SCL	I2C:ssä käytettävä kellolinja
EEPROM	Electronically Erasable Programmable Read-Only Memory, haihtumaton muistityyppi
SWD	Serial Wire Debug, ARM –suorittimella käytetty vaihtoehtoinen debuggausväylä

KUVIO- JA TAULUKKOLUETTELO

1 JOHDANTO

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja on Vacon Oyj, joka siirtyi osaksi tanskalaista Danfossia joulukuussa 2014 yrityskauppojen myötä /1/. Danfossin ja Vaconin yhteenlaskettu markkinaosuus maailmanlaajuisesti taajuusmuuttajissa on toiseksi suurin /2/.

Lopputyöaiheen sain saapuessani Vaconille harjoittelijaksi tuotekehitykseen kesällä 2015. Tuotekehityksen tehtävä on antaa tukea tuen piirissä oleville laitteille sekä kehittää uusia innovaatioita nykyisiin ja tuleviin tuotteisiin.

1.2 Projektin tavoite

Tässä tuotekehitysprojektissa kehitetään kustannustehokasta tapaa kerätä dataa mm. lämpötilatietoja, sekä tunnistaa verkkoon kytkettyjä laitteita. Kerätyn datan avulla voidaan tarjota asiakkaille esimerkiksi entistä laajempia ja ennakoivampia huoltoratkaisuja. Ennakoivat huoltoratkaisut hyödyttävät asiakasta, kun laitteen alhaallaoloaika on mahdollisimman lyhyt, kun huoltoa ei tilata sillä hetkellä jolloin huollon pitäisi jo olla.

Laitteistolla voidaan esimerkiksi mitata puhaltimen toiminta-aikaa sekä lämpötilaa, jolloin voidaan ennalta laskettujen määritelmien perusteella varoittaa asiakasta ennalta ja tarjota huoltoa. Tässä tapauksessa esimerkiksi voitaisiin kerätä tietoa ja hallita ilmajäähdytteisen taajuusmuuttajan puhaltimia. Puhaltimien pyörimisnopeutta voitaisiin säätää laitteesta kerätyn (lämpötila)datan perusteella ja täten voitaisiin pienentää energiankulutusta ja pidentää puhaltimen käyttöikää, kun puhallin ei pyöri koko ajan äärirajoilla.

Lopputuotteena saadaan prototyyppi, joka tarjoaa alustan palvelun jatkokehitykselle. Prototyypissä master kommunikoi kymmenen slave-laitteen kanssa ja laitteet voivat välillä sammua ja palata uudelleen verkkoon aiheuttamatta toimintahäiriöitä muille verkon laitteille.

1.3 Projektin vaiheet

Projektin ensimmäisessä vaiheessa tutustutaan olemassa oleviin tiedonsiirtoprotokolliin, sekä valitaan tuotekehityksen aikana käytettävä kehitysympäristö. Ensimmäiseen vaiheeseen kuuluu tiedonsiirtoprotokollan spesifiointi annettujen vaatimusten mukaisesti.

Toisessa vaiheessa simuloidaan verkon toimintaa Windows-ympäristössä simulaatio-ohjelmalla, jolla simuloidaan suunniteltuja spesifikaation mukaisia funktioita.

Kolmannessa vaiheessa protokollaa ja verkon toimintaa kehitetään aiemmin valitussa kehitysympäristössä. Vaiheeseen kuuluu myös luotettavuustestausta, jossa aiheutetaan verkkoon sähkömagneettisia häiriöitä.

Neljännessä vaiheessa verkon masterin toiminnallisuus siirretään lopulliseen ympäristöön ja suunnitellaan lähempänä lopputoteutusta oleva slave–node, jolla kyetään toteuttamaan ja verifioimaan vaadittu toiminnallisuus.

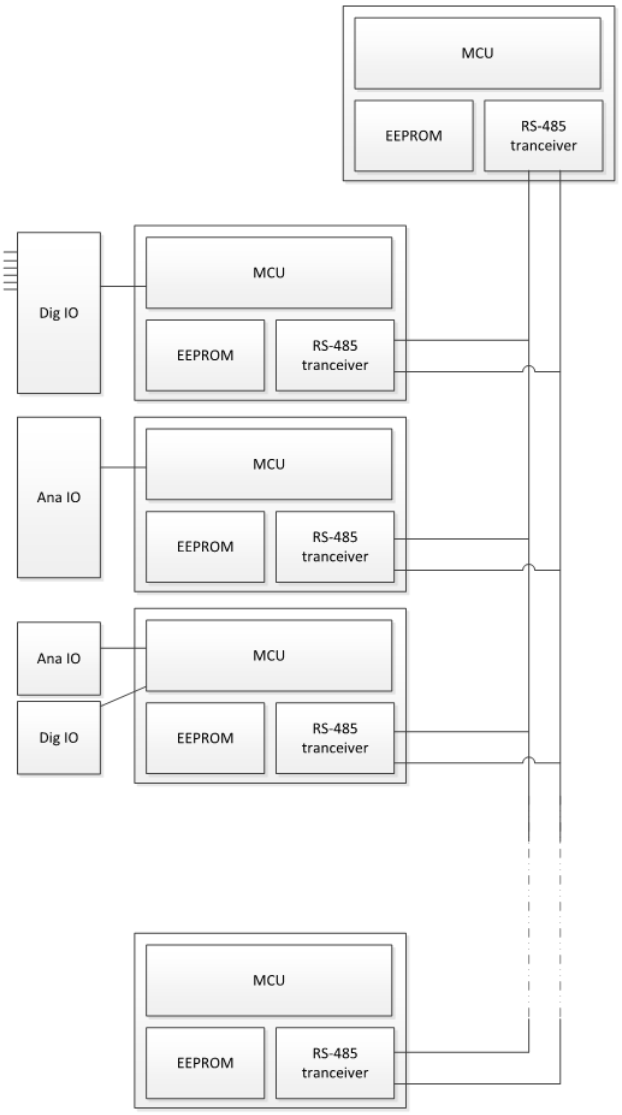
2 TIEDONSIIRTOTEKNIIKAT JA -PROTOKOLLAT

Tässä työssä kehitettiin häiriöisissä olosuhteissa toimivaa tiedonsiirtoverkkoa. Sähkömagneettiset häiriöt aiheuttavat tiedonsiirtoverkolle omia haasteita. Tässä luvussa tutustutaan olemassa oleviin tiedonsiirtoväyliin ja -protokolliin sekä oman tiedonsiirtoprotokollan kehitykseen.

2.1 Tiedonsiirtoväylät

Tiedonsiirtoväylät määrittävät fyysiset ominaisuudet tiedonsiirrolle. Fyysiset ominaisuudet määritellään standardeissa. Useisiin väyliin on saatavilla mikro-ohjaimeen kytkettäviä piirejä, joita ohjataan esimerkiksi mikro-ohjaimen sarjaliitynnällä. Tiedonsiirtoväyliä on niin langallisia, kuin langattomiakin. Tässä kappaleessa tutustutaan kolmeen langalliseen tiedonsiirtoväylään.

Väyliä voi olla myös eri topologioilla toteutettuja. Yleinen kenttäväylissä käytetty topologia on bus-verkko (kuvio 1), jossa laitteet on kytketty samoihin johtimiin. Muita yleisiä topologioita ovat usein lähiverkoissa käytetty tähti sekä rengas. Tähti-topologiassa laitteet on kytketty omalla kaapelilla esimerkiksi kytkimeen. Rengasverkossa laite on kytketty kaapeleilla vierekkäisiin laitteisiin siten, että kun viesti menee laitteesta tarpeeksi kauan kuvitellusti oikealle, saa laite lähettämänsä viestin vasemmalta.

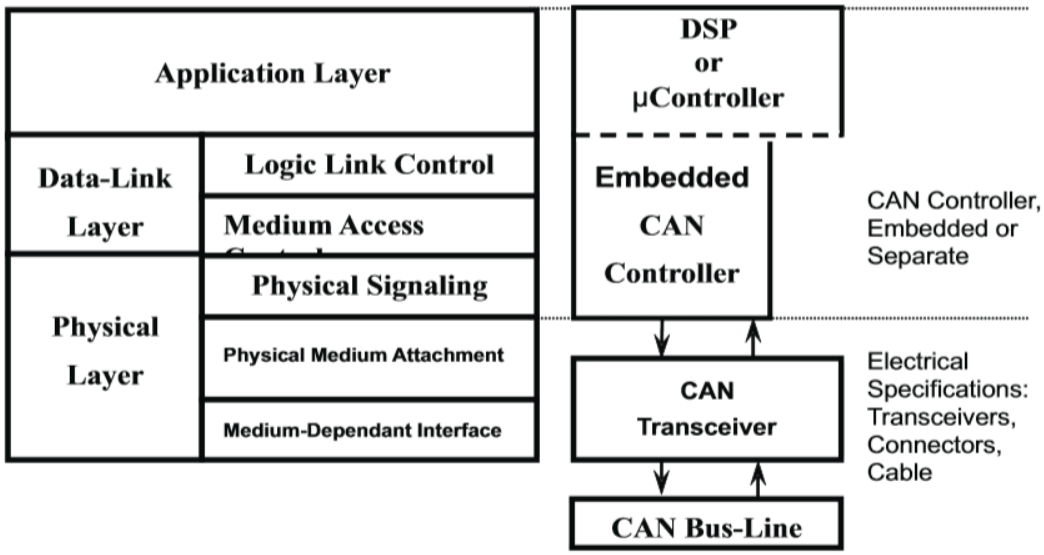


Kuvio 1. Bus-verkko ja siihen kytkettyjen laitteiden lohkokaavio.

2.1.1 CAN

CAN-väylä on alun perin autoteollisuuteen kehitetty minimalistinen tiedonsiirtoväylä, mutta on nykyään yleisesti teollisuudessa käytetty kenttäväylä. CAN-väylän löytää nykyään lähes jokaisesta modernista autosta, joiden OBD-lukijoiden väylänä CANia usein käytetään. CAN–väylän käyttö määrittää OSI-

mallin kaksi alinta tasoa (kuvio 2), CAN–väylän fyysiset ominaisuudet perustuvat ISO-11898 –standardiin /3/.



Kuvio 2. CAN -väylän arkkitehtuuri /4/.

CAN-väylän tiedonsiirtonopeus on 125 kb/s – 1 Mb/s. CAN-väylässä viestin vastaanottaja tunnistetaan joko 11 tai 29 bittisellä (Extended CAN) tunnisteella.

2.1.2 RS-422

RS-422 on RS-485:n tapaan balansoitu sarjaliikenneväylä, joka mahdollistaa 11 laitteen kytkemisen verkkoon. Tiedonsiirtonopeus RS-422 verkossa oi olla jopa 10 Mb/s. Haittapuolena tekniikassa on se, että ainoastaan yksi verkkoon kytketyistä laitteista voi lähettää, jolloin dataa ei ole mahdollista pyytää muilta laitteilta, eikä täten ollut mahdollista käyttää tässä työssä.

2.1.3 RS-485

RS-485 on balansoitu sarjaliikenneväylä, joka mahdollistaa jopa 32 laitteen liittymisen samaan verkkoon. RS-485–verkon fyysiset ominaisuudet on määritetty ANSI/TIA/EIA-485 -standardissa, jonka Telecommunications Industry

Association/Electronic Industries Alliance on julkaissut 1998. Erona RS-422-verkkoon on, että kaikki verkkoon kytketyt laitteet voivat toimia lähettäjänä. RS-485-verkon maksimi tiedonsiirtonopeus on jopa yli 10 Mb/s.

Tässä työssä päädyttiin käyttämään RS-485:ttä, koska kyseisellä tekniikalla oli aiemmin demottu vastaavaa kommunikointia point-to-point eli kahden pisteen välillä. Verkon prototyypissä päädyttiin tekemään topologialtaan bus-verkko (katso kuvio 1 sivulla 11), jossa laitteet ovat half-duplextilassa, jolloin ainoastaan yksi verkon laitteista lähettää kerrallaan. RS-485-verkon baudinopeuden, eli symbolinopeuden määrittää käytetty RS-485-ohjain. Tavallisesti nopeus on 9600 baud/s – 1 MBaud/s.

2.2 Tiedonsiirtoprotokollat

Protokollat määrittävät miten verkkoon kytketyt laitteet keskustelevat keskenään. Julkiset protokollat mahdollistavat eri laitevalmistajien valmistaa keskenään yhteensopivia tuotteita. Tiedonsiirtoprotokollia on olemassa eri käyttöympäristöihin ja –tarkoituksiin.

2.2.1 Modbus

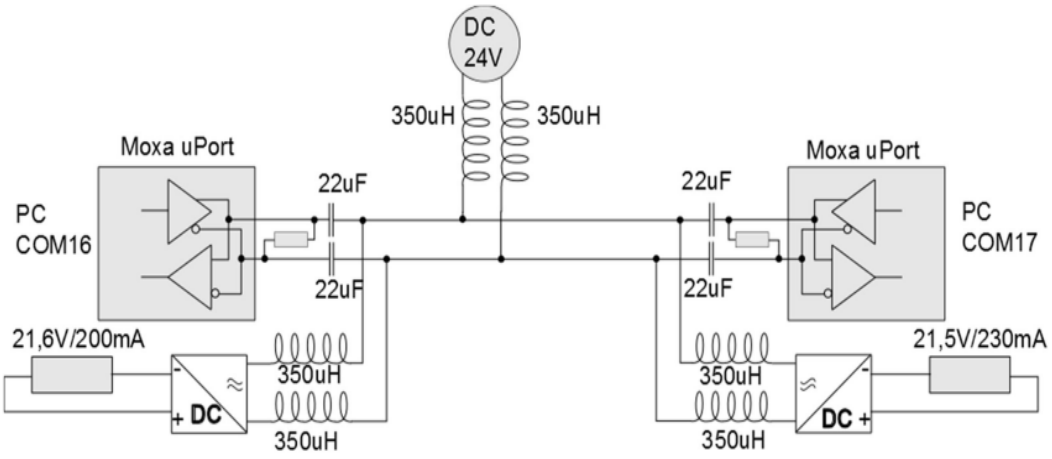
Tässä kappaleessa tutustutaan Modbus-protokollan eri variaatioihin. Modbus kaikkine variaatioineen on yleisesti käytetty tiedonsiirtoprotokolla teollisuudessa. Modbus protokollan on julkaissut Modicon vuonna 1979, alun perin PLC-laitteiden ohjaukseen /5/. Modbus–protokollaperhe tarjoaa laajan valikoiman eri tarkoituksissa ja ympäristöissä käytettäviä protokollia. Modbusin variaatiota ovat sarjaliikenteessä käytetyt RTU ja ASCII, sekä ethernetin päällä käytettävä TCP. Tämän lisäksi on kehitetty protokolla, joka mahdollistaa sarjaliikenteen ethernetin päällä, Modbus RTU Over TCP. Eri Modbus laitteille on saatavana myös yhdyslaitteita, jolloin esimerkiksi RTU- ja TCP-laitteita voidaan kytkeä samaan verkkoon.

3 KEHITYSYMPÄRISTÖ JA KÄYTETYT TYÖKALUT

Kehitysympäristö ja käytetyt työkalut valittiin osittain valmiina olleiden lisenssien ja aiemmin tehtyjen kokeilujen perusteella. Kehitysympäristönä käytettiin IAR:n Embedded Workbenchiä, aiemmin hankitun IAR:n ARM–kääntäjän ja debuggerin vuoksi. Kehitysympäristöt ovat ohjemointityökaluja, jotka usein sisältävät tarvittavat asiat koodin lataukseen ja debuggaukseen laitteessa.

3.1 Fyysinen ympäristö

Fyysinen kehitysympäristö rakennettiin Vaconilla aiemmin kokeillun demonstraattorin perusteella (Jyrki Mäki-Turja). Demonstraattorilla oltiin kokeiltu point-to-point yhteyttä kahden sarjaportin välillä DC-linjan päällä (kuvio 3.) Kytkennässä Moxa RS-485-lähetinvastaanottimien liitäntöihin ollaan kytketty kondensaattorit, jotka päästävät lävitseen ainoastaan nopeita pulsseja, tässä tapauksessa RS-485-lähettimen tilamuutokset. Kelat puolestaan ei vastaanota näitä nopeita vaihteluita, vaan jännite on tasainen ja voidaan syöttää esimerkiksi piirilevylle jännitetuloksi.



Kuvio 3. Jyrki Mäki-Turjan demonstraattori.

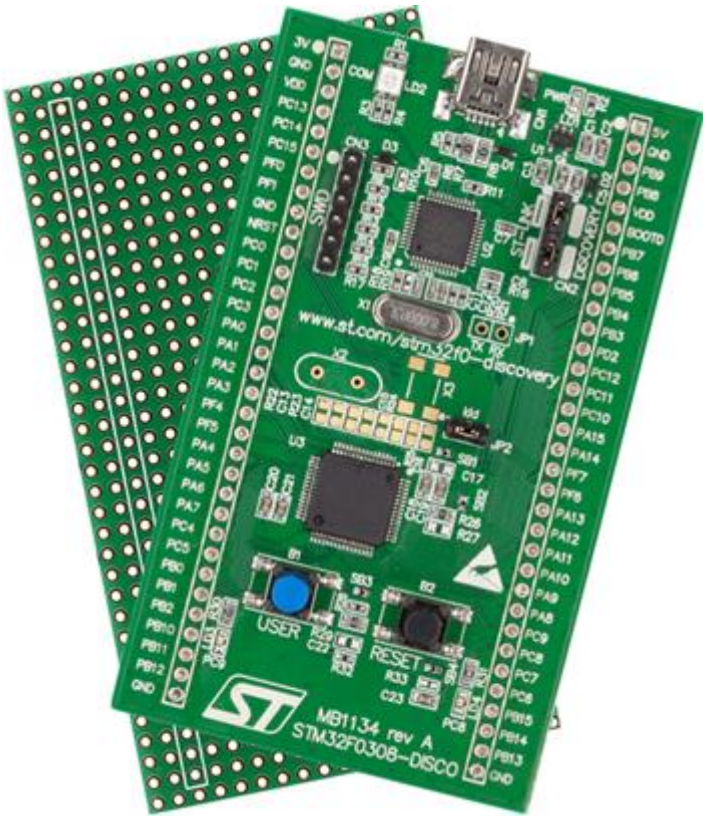
Fyysisen ympäristön käyttö vaikutti myös osaltaan ohjelmiston kehitykseen. Koska tiedonsiirto tapahtuu DC-linjassa, täytyy lähetetyt viestit balansoida, jotta

jännitetaso ei heittele ja vaikuta kytkettyjen laitteiden toimintaan. Lisää balansoinnista kappaleessa 4.6.

DC-linjassa kulkeva liikenne valittiin tähän työhön, koska tarkoituksena oli kehittää kustannustehokasta tapaa siirtää tietoa. Tässä mallissa ei ollut tarvetta ylimääräisille kaapeloinneille, koska jännitteistys oheislaitteille täytyisi tehdä joka tapauksessa, mikä osaltaan puoltaa kustannustehokkuutta.

3.1.1 STM32-kehitysalusta

Prototyypin tekemiseen valittiin STMicroelectronicsin valmistama 32F0308DISCOVERY-kehitysalusta (kuvio 4), jonka mikroprosessori STM32F0308R8 käyttää ARM Cortex M0-ydintä. Kehitysalusta valittiin helposti käytettävien I/O-pinnien sekä kehitysalustalla olevan debuggerin vuoksi.



Kuvio 4. STM32F0308-DISCO-kehitysalusta /6/.

Kehitysalustalla oli ohjelmointiin ja debuggaukseen käytettävä ST-LINK/V2–debug-työkalu ja käytetyllä prosessorilla oli työssä vaadittavat UART- ja I2C-oheislaitteet, sekä riittävästi sisään- ja ulostuloja, jotka yhdessä vaikuttivat kehitysympäristön valintaan. Kortin ohjelmisto ladattiin käyttämällä kortin usb-liitäntää.

STMicroelectronicsin kehitysalustoille löytyy kattava määrä esimerkkiprojekteja ja hyvin dokumentoitu kirjasto oheislaitteille, jotka helpottivat sovelluskehitystä.

3.1.2 RS-485 to Powerline–adapteri

Demonstraattorin (kuvio 3) pohjalta valmistettiin adapteri, joka voitaisiin kytkeä mikro-ohjaimen ja 24 voltin DC-jännitteen väliin (kuvio 5.) Adapterilla muutetaan mikro-ohjaimen sarjaliikenne differentiaaliseen RS-485-väylään, sekä tiedonsiirtoväylän 24 volttia mikro-ohjaimelle sopivaksi 5 voltin jännitteeksi.

Mikro-ohjaimella muutettiin 5 voltin jännite oheislaitteille sopivaksi 3,3 voltin jännitteeksi. Slave-laitteiden käyttämä muistipiiri on myös adapterilla, tässä tapauksessa I2C-ohjattu EEPROM.

Kuvio 5. Moduuli ja kehitysalusta.

3.2 Sovelluskehitys

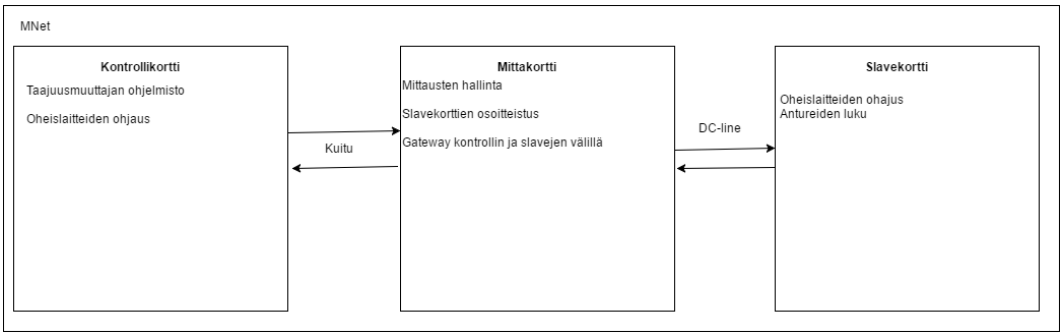
Sovelluksen kehitykseen käytettiin C–ohjelmointikieltä ja Windows simulaatio-ohjelma C++-kielellä. Työssä käytettiin Vaconin ohjelmointiohjeistusta, joka loi ääriviivat käytetylle ohjelmointityylille. Sovelluskehitykseen käytettiin IAR Embedded Workbenchia, Eclipseä ja Visual Studio -työkaluja.

Sovelluskehitystä varten oli luotu joukko vaatimuksia (katso taulukko 1 sivulla 20), jotka muutettiin sopiviksi ohjelmointitehtäviksi ja ohjelman osiksi. Vaatimusten täyttymistä seurattiin scrumin sprinteissä.

4 TIEDONKERUUOHJELMA

Tässä luvussa käsitellään ohjelmiston ja siihen liittyvien osien kehitystä. Ohjelmaa kehitettiin ketterällä menetelmällä ja työtä suunniteltiin kolmen viikon sprintti kerrallaan.

Ohjelmisto on jaettu kolmeen osaan (kuvio 6.) Master on kahdessa osassa kontrolli- ja mittakortilla ja kolmas osa on slave-korteilla.



Kuvio 6. Ohjelmiston hajautus.

Kaavion ”DC-line” on sama, kuin aiemmin esitelty, työssä käytettävä tiedonsiirtoväylä (kuvio 6.) Muuten kuviossa esitelty tämän työn kannalta oleellisimmat osat.

4.1 Mnet

Tässä työssä päädyttiin tekemään protokolla - Mnet, joka pohjautuu osin Modbusiin, sekä osittain Vaconin omaan HMI–protokollaan. Tähän päädyttiin haasteellisen ympäristön vuoksi. Ympäristöstä tulevat häiriöt saattavat aiheuttaa linjassa näkyviä virheitä. Näistä aiheutuvia haittavaikutuksia pyrittiin minimalisoimaan alhaisemmalla merkkinopeudella, jota ei kuitenkaan voitu käytetyn tekniikan vuoksi laskea alle 115 200 baudiin sekunnissa. Erona esiteltyyn Modbusiin on se, että protokollatasolla voidaan ajatella, että ollaan yhdistetty Modbus/RTU:sta ja Modbus/ASCIIsta asioita, kuten aloitus- ja lopetusmerkkien ajatus ASCIIsta ja tavunkokoiset komennot RTU:sta.

Modbusista poiketen Mnetissä ei ole vastaavaa osoitetaulua arvoille, vaan arvot on järjestyksessä IO-taulussa (kuvio 10.)

Kehitetylle protokollalle suunniteltiin kehyksen rakenne sekä joukko komentoja, joilla ohjattaisiin verkkoon kytkettyjä node –laitteita. Protokollan kehykseen kuuluu hyötykuorman lisäksi kahden tavun aloitus- ja lopetusmerkit sekä CRC16-tarkistussumma (kuvio 7.)

DLE	STX	Address	Command	Data	DLE	EOT/ETB	CRC16
1 byte	1 byte	1 byte	1 byte	0-64 bytes	1 byte	1 byte	2 bytes

Kuvio 7. Tiedonsiirtokehys.

4.1.1 Protokollan suunnittelu

Työn vaatimuksessa esitettiin, että mahdollisia slaveja tulee pystyä lisäämään verkkoon dynaamisesti jopa 10 kappaletta. Muuten verkon toiminnallisuus päätettiin toteuttaa master – slave –verkkona. Yhden masterin verkossa ei laitteilla ole epäselvyyttä kuka verkkoa hallinnoi. Vikatilanteiden kannalta useamman masterin verkko olisi kestävämpi, mutta tässä kehitysprojektissa päätettiin verkko toteuttaa yksinkertaisuuden vuoksi yhdellä masterilla.

Taulukko 1. Tiedonkeruuverkon vaatimusmäärittely.

Vaatimus	Selitys	Prioriteetti
Laitteiden lisääminen dynaamisesti	Laitteiden osoitteistus jo toiminnassa olevaan verkkoon, ilman että toiminta häiriintyy	1
Laitteiden ohjaus hitaassa aikatasossa	Verkon perus toiminnallisuus, laitteiden ohajus	1
Verkkoon voidaan kytkeä kymmenen oheislaitetta	Kymmenen laitetta tulee toimia verkossa ilman ongelmia	1
Virheentarkistus	Viestien virheentarkistus, viallisten viestien hylkäys	2

Master – slave–verkko itsessään on hyvin yksinkertainen, yksi master lähettää tiedossa oleville slave–laitteille kyselyitä, joihin slavet vastaavat. Master – slave-verkkoa voidaan tyypillisesti käyttää myös yhdensuuntaiseen kommunikointiin, jossa master jakaa komentoja kytketyille slave-laitteille joko yksitellen tai kaikille kerralla. Tässä verkossa toteutettiin kaikille laitteille menevä broadcast-viesti siten, että laiteosoitteena viestissä käytetään osoitetta 0 (kuvio 8.)

Suunnitellun protokollan mukaan slave-laitteiden tulee kysyä osoite verkon masterilta käynnistyksen yhteydessä. Osoitekysely on kolmivaiheinen. Kyselyssä slave aloittaa osoitepyynnöllä, joka lähetetään verkon broadcastosoitteeseen ja on tyypiltään ADDRESS_QUERY. Datana viestissä on kortin yksilöity 4 tavun kokoinen ID. Kättelyn toisessa vaiheessa master lähettää broadcastosoitteeseen kortin uuden viestintäosoitteen. Viesti koostuu tyypistä ADDRESS_ASSIGN sekä ID:stä (4 tavua) ja kortin osoitteesta (1 tavu). Slave varmistaa osoitteen vielä masterille lähettämällä omalla osoitteellaan viestin, jonka tyyppi on ADDRESS_VERIFY ja kortin tyyppi (1 tavu).

	address	command	data
node -> master	BROADCAST (1B)	ADDRESS_QUERY (1B)	UID (4B)
master -> node	BROADCAST (1B)	ADDRESS_ASSIGN (1B)	UID (4B) + NEW_ADDRESS (1B)
node -> master	NODE_ADDRESS (1B)	ADDRESS_VERIFY (1B)	NODE_TYPE(1B)

Kuvio 8. Osoiteviestit.

4.2 Toimintaperiaate

Välttääkseen päällekkäisten viestien lähettämisen, slave-laitteet odottavat satunnaisen ajan ennen ensimmäistä osoitekyselyä. Tämän lisäksi laitteet tarkistavat onko UART rekisterin vastaanotto aktiivinen. Satunnaista nukkumista ja pollausta jatketaan kunnes master on antanut osoitteen ko. laitteelle.

Kun nodet ovat saaneet osoitteen, verkon toimintaa ohjaa kontrollikortti. Kontrolli käy mittakortilta saamaansa osoitetaulua osoite kerrallaan läpi lähettäen jokaiseen osoitteeseen kyselyn inputien arvoista (kuvio 9.) Kontrolli vertaa saamiaan arvoja määriteltyihin raja-arvoihin ja tarvittaessa lähettää ohjausarvon nodelle. Node kuittaa saamansa ohjausarvon paluuviestillä.

Kuvio 9. Sekvenssikaavio.

Sekvenssikaaviosta voidaan nähdä ohjelman suoritusrakenne (kuvio 9.) Suoritusrakenne ohjelmistossa on yksinkertainen ja koostuu osoitetaulun käymisestä läpi ja sen lisäksi satunnaisesta, virran kytkentähetkeen painottuvasta nodejen osoitteistuksesta.

Verkossa käytettävät komennot (taulukko 2) ovat tasoltaan peruskom)entoja, joista osalle voidaan antaa tarkentavia komentoja. Esimerkiksi lue arvo – lue id; on kaksiosainen komento, jolla saadaan tarkennettua haluttua tehtävää.

Taulukko 2. Verkossa käytettävät komennot / funktiot.

ADDRESS_QUERY	Osoitteen kysyminen, nodelta masterille
ADDRESS_ASSIGN	Osoitteen antaminen, masterilta nodelle
ADDRESS_VERIFY	Osoitteen vahvistus, nodelta masterille
READ_VALUE	Lue arvo
WRITE_VALUE	Kirjoita arvo
SAVE_ADDRESS	Tallenna osoite eepromille

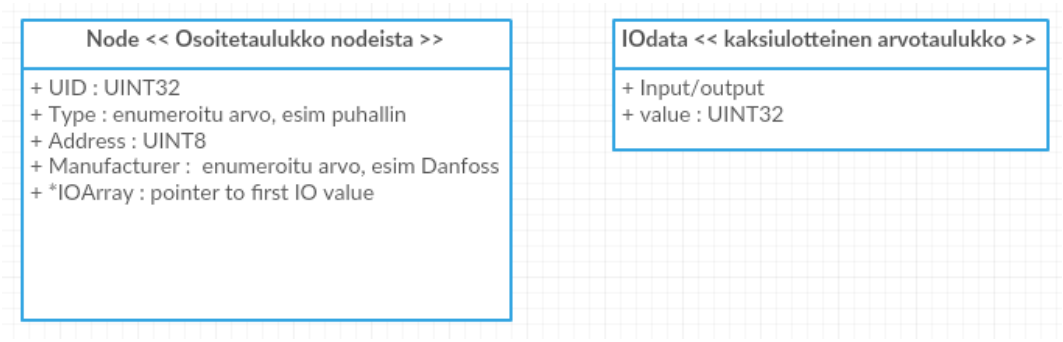
Osalle komennoista kehitettiin alikomennot (taulukko 3) paremman luettavuuden vuoksi. Käytännössä oltaisiin voitu käyttää myös jokaista komentoa itsenäisesti, mutta lopputulos olisi ollut sekava.

Taulukko 3. Komentojen alikomennot.

READ_VALUE	READ_ID	Laitteen ID:n tarkistusta varten
	READ_TYPE	Luetaan laitteen tyyppi EEPROMilta
	READ_IO	Viestikentässä voidaan määrittää kuinka monta, lähtökohtaisesti luetaan kaikki sisään- ja ulostulojen tila
WRITE_VALUE	WRITE_IO	Viestikentässä määritetään mihin IO ja mikä arvo
	WRITE_ADDRESS	Osoite kirjoitetaan talteen EEPROMille

4.3 Tietorakenne

Työn vaatimuksen mukaan yhdessä verkossa voi olla 10 + 1 laitetta. Jokaisessa slave–laitteessa on maksimissaan 16 x 32 bittistä dataa, jotka voidaan jokainen määritellä erikseen sisään- tai ulostuloiksi. Kuviossa 10 on esitelty kuinka nodejen tiedot on tallennettu masteriin. Node-taulussa on talletettu slave-kortin tunnistetiedot sekä ensimmäisen IO-datan osoite. IOdata on tyyppimääritelty strukti, joten arvoja voidaan lukea helposti loopissa. Sisääntulot voivat olla esimerkiksi AD-muuntimia tai lämpötilamittauksia. Ulostuloilla voidaan ohjata esimerkiksi PWM-tuulettimelle ajettavaa pulssinleveyttä.



Kuvio 10. Tietorakenne.

Jotta voidaan varmistua, että verkon master löytää kaikki kytketyt laitteet, masterille voidaan ladata asennusvaiheessa konfiguraatiotiedosto, josta voidaan tarkistaa ID:n perusteella slave–laitteiden ja niiden sisään- ja ulostulojen määrät sekä tyypit.

Koska slaveilla on paljon samankaltaista dataa, päätettiin käyttää tietomallissa taulukoituja strukteja, joiden sisältämä data voidaan määrittää muuttujalla.

Jokaiselle nodelle talletettiin yksinkertainen ID-tunniste eepromille (taulukko 4), joka kuvaa kyseisen noden toimintaa. ID-tunniste on yleensä jonkin laitteen, esimerkiksi kompressorin mukana toimitettava tunniste, josta käy ilmi laitteen tunnistetiedot (kuvio 11.) Tässä työssä fyysinen laatta on toteutettu kirjoittamalla arvot muistille.

Kuvio 11. Esimerkki Danfossin käyttämästä tunnisteesta /7/.

Taulukko 4. Noden ID-plate.

Nimi	Tyyppi
UID	unsigned integer (32 bit)
type	IO_Dev_type (enum)
address	unsigned char (8 bit) (jos tallennettu)
manufacturer id	unsigned char (valinnainen)

4.4 Master–ohjelman suunnittelu ja toteutus

Verkon master päätettiin toteuttaa siten, että ohjelmisto on jaettu taajuusmuuttajan kontrolli- ja mittakortille. Molemmilla korteilla toimii reaaliaikakäyttöjärjestelmä, joiden taskeina ohjelmaa ajetaan. Mittakortti on verkon yhteys taajuusmuuttajan aivoihin sekä sovellusrajapintaan, joka on tässä tapauksessa kontrollikortti. Mittakortti valittiin yhdyspisteeksi, koska se on yhteydessä muihin kytkettyihin IO laitteisiin samassa 24 voltin DC-linjassa, jolloin ylimääräisiä tiedonsiirtoväyliä ei tarvitse käyttää.

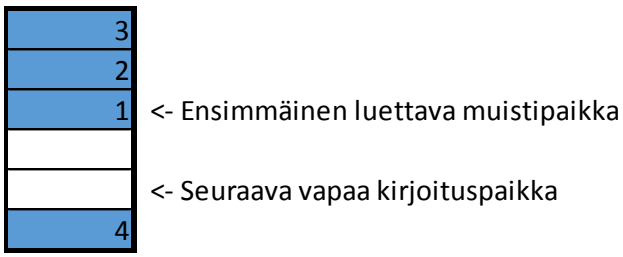
Koska mittakortti on ympäristönä kontrollikorttia rajallisempi, toteutettiin mittakortilla ainoastaan slave-osoitteiden hallinta. Muuten mittakortti toimi

ainoastaan yhdyskäytävänä kontrollin ja slavejen välillä hoitaen kommunikoinnin DC-balansoinnin (kappale 4.6). Kontrollin ja mittakortin välinen kommunikointi toteutettiin taajuusmuuttajan sisäiseen kommunikointiin suunnitellulla protokollalla ja väylällä.

Kontrollikortin ohjelmisto toteutettiin siten, että ohjelman lähtiessä ajamaan, kontrolli pyytää mittakortilta päivitetyn osoitetaulun. Tämä päivitys sisältää verkkoon kytkettyjen nodejen uniikin ID:n, tyypin ja verkko-osoitteen. Tämän jälkeen kontrolli lähettää kyselyitä verkkoon nodejen verkko-osoitteiden mukaan. Silloin kun kyseessä on inputeja sisältävä node, master vertaa uusia arvoja edellisiin arvoihin, sekä inputille määritettyihin mahdollisiin raja-arvoihin. Täten kontrollin ohjelmistolla olisi mahdollista ohjata kytkettyjä oheislaitteita.

4.5 Slave–ohjelman suunnittelu ja toteutus

Slavekortin ohjelmiston yksinkertaisuuden vuoksi ei työssä käytetty erillistä käyttöjärjestelmää, vaan ohjelma ajettiin eteenpäin yhdessä silmukassa. Slave lukee keskeytyksessä saapuvaa UART liikennettä ja täyttää tulevat viestit rengaspuskuriin (kuvio 12.) Rengaspuskuria luetaan pääohjelmassa ja lukuoperaation tulos määrittää mitä slave tekee seuraavaksi.



Kuvio 12. Rengaspuskuri.

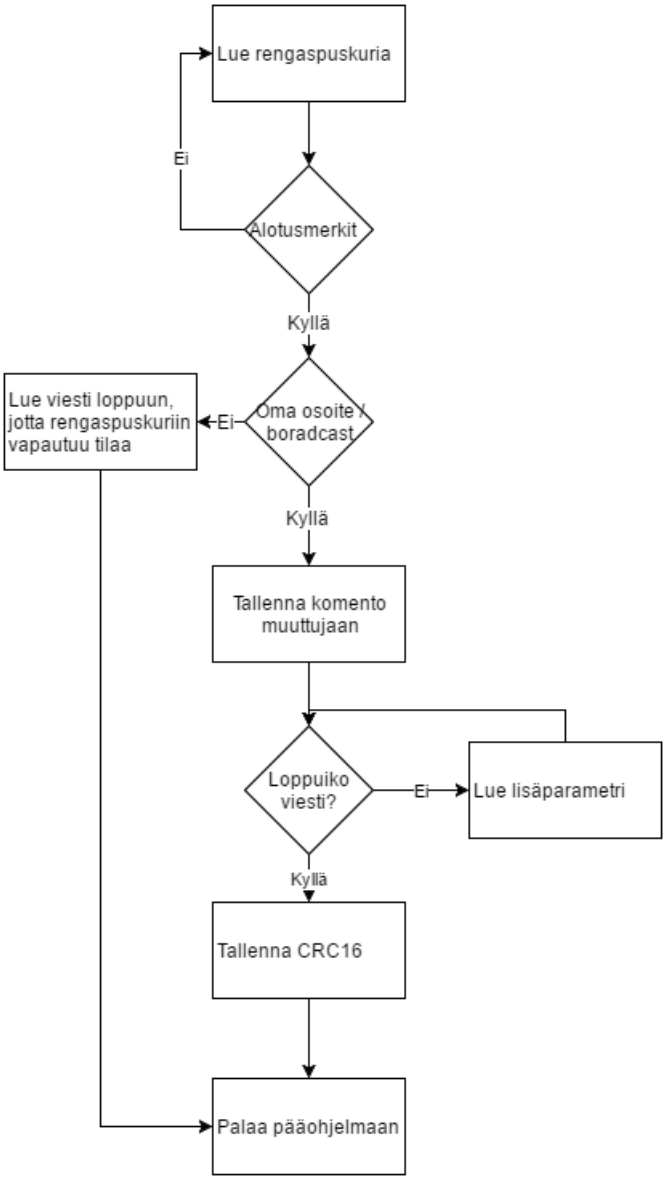
Rengaspuskuriin varataan haluttu määrä tilaa muistista. Tässä työssä käytettiin keskusmuistia datan tallennukseen nopean käsittelyajan vuoksi. Rengaspuskuria käytetään joko keskeytyksestä (kirjoitus) tai pääohjelmasta (luku). Luku-operaatiossa on tärkeää ottaa keskeytykset pois käytöstä, jottei kesken operaation luku- ja kirjoitusosoitteet mene sekaisin.

Slave käsittelee saapuneet viestit parserilla ja näiden perusteella määräytyy mitä kortti tekee. Saapuvilla komennoilla ohjataan IO-pinneihin kytkettyjä oheislaitteita masterin komentojen mukaan. Komennossa annetaan esimerkiksi PWM-ohjaimen prosentuaalinen arvo.

Jokaiselle tehtävälle komennolle implementotiin funktioit, joiden osoitteet tallennettiin slaven IO-taulun muistiin. Jos kyseessä on ulostulo, arvon muutos johtaa esimerkiksi PWM-ohjaimen arvon muutokseen.

4.5.1 Viestien parsinta

Saapuneet viestit parsitaan (kuvio 13), eli rengaspuskuria käydään läpi muistipaikka kerraallan ja kerätään saapunut viesti kokonaisuudessaan. Viestin parsinta aloitetaan, kun rengaspuskurista löytyy viestirakenteen aloitusmerkit.



Kuvio 13. Parserin flow-kaavio.

Parserille annetaan osoitteena paikka vastaanottajan ID:lle, viestin komennolle ja alikomennolle, sekä bufferi datalle ja 16 bittinen muuttujapaikka CRC tarkistussummalle. Tämän jälkeen jos viesti on ollut kyseiselle vastaanottajalle, tarkistetaan CRC ja toimitaan saadun komennon mukaan.

4.5.2 Prosessorin oheislaitteet

Prosessorin oheislaitteet vaativat omat ajurit ja rekistereiden alustukset. USART- ja I2C-väylille löytyi valmistajan tarjoamat ajurit (kuvio 14 ja 15), joita käytettiin tässä työssä. Ajurit tarjosivat rajapinnan merkkien lähettämiseen USARTilla ja I2C:llä. Näiden päälle luotiin ylemmän tason funktiot, joissa käytettiin ajurin funktioita.

```
typedef struct
{
  uint32_t USART_BaudRate;            /*!< This member configures the USART communication baud rate.
                                           The baud rate is computed using the following formula:
                                            - IntegerDivider = ((PCLKx) / (16 * (USART_InitStruct->USART_BaudRate)))
                                            - FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 1

  uint32_t USART_WordLength;          /*!< Specifies the number of data bits transmitted or received in a frame.
                                           This parameter can be a value of @ref USART_Word_Length */

  uint32_t USART_StopBits;            /*!< Specifies the number of stop bits transmitted.
                                           This parameter can be a value of @ref USART_Stop_Bits */

  uint32_t USART_Parity;              /*!< Specifies the parity mode.
                                           This parameter can be a value of @ref USART_Parity
                                           @note When parity is enabled, the computed parity is inserted
                                                 at the MSB position of the transmitted data (9th bit when
                                                 the word length is set to 9 data bits; 8th bit when the
                                                 word length is set to 8 data bits). */

  uint32_t USART_Mode;                /*!< Specifies wether the Receive or Transmit mode is enabled or disabled.
                                           This parameter can be a value of @ref USART_Mode */

  uint32_t USART_HardwareFlowControl; /*!< Specifies wether the hardware flow control mode is enabled
                                           or disabled.
                                           This parameter can be a value of @ref USART_Hardware_Flow_Control*/
} USART_InitTypeDef;
```

Kuvio 14. USART-rekisterin alustusparametrit /8/.

Taulukko 5. USARTin alustusparemetrit.

BaudRate	*115200 baud/s*
WordLength	*8 bittiä*
StopBits	*1*
Parity	*none*
Mode	*Receive ja transmit*
HardwareFlowControl	*ei*

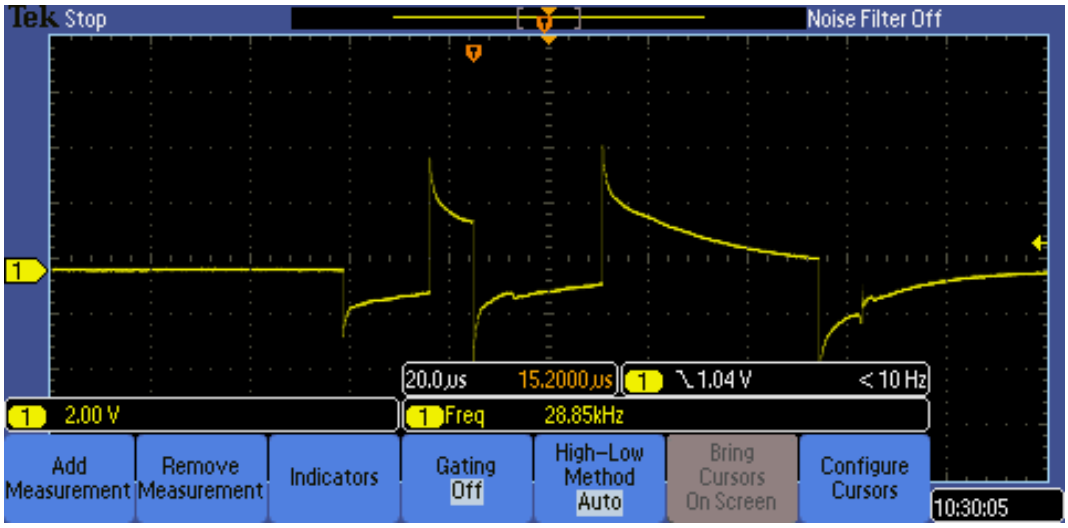
```
typedef struct
{
  uint32_t I2C_Timing;              /*!< Specifies the I2C_TIMINGR_register value.
                                       This parameter must be set by referring to I2C_Timing_Config_Tool*/

  uint32_t I2C_AnalogFilter;        /*!< Enables or disables analog noise filter.
                                       This parameter can be a value of @ref I2C_Analog_Filter*/

  uint32_t I2C_DigitalFilter;       /*!< Configures the digital noise filter.
                                       This parameter can be a number between 0x00 and 0x0F*/

  uint32_t I2C_Mode;                /*!< Specifies the I2C mode.
                                       This parameter can be a value of @ref I2C_mode*/

  uint32_t I2C_OwnAddress1;         /*!< Specifies the device own address 1.
                                       This parameter can be a 7-bit or 10-bit address*/

  uint32_t I2C_Ack;                 /*!< Enables or disables the acknowledgement.
                                       This parameter can be a value of @ref I2C_acknowledgement*/

  uint32_t I2C_AcknowledgedAddress; /*!< Specifies if 7-bit or 10-bit address is acknowledged.
                                       This parameter can be a value of @ref I2C_acknowledged_address*/
}I2C_InitTypeDef;
```

Kuvio 15. I2C-rekisterin alustusparametrit /8/.

Prosessorilla suoritettiin myös IO-pinnien tilamuutoksia. Pinneihin oli kytkettynä ledejä sekä releitä, AD-muunninta käytettiin lukemaan pinniin syötettävää jännitettä.

4.6 DC–balansointi

Ensimmäisen demonstraattorin perusteella oltiin todettu, että verkossa lähetettävä data tulee balansoida. Balansointi vaadittiin valitun tiedonsiirtokanavan vuoksi. Balansoimaton data olisi aiheuttanut DC-verkossa epätasaisuutta, joka olisi vaikuttanut verkon laitteiden jännitetasoihin, kun kytkettyjen laitteiden käyttämä jännite on 24 V. Myös adapterikorttien kondensaattorit vaativat kyseisen balansoinnin, jos viestissä olisi monta (>2) samaa bittiä peräkkäin, kondensaattoreiden purkautuminen aiheuttaisi tilan muuttumisen aiottua aiemmin (kuvio 16). Linjan palatessa nollatasoon aiottua aiemmin vastaanottava laite ei tunnista lähetettyä bittiä ja tavu rikkoutuu. Tavun rikkoutumista varten lähetetyssä viestissä on CRC16 tarkistusluku. Kuviossa 16 esitetään tavun rikkoutuminen tilan palatessa nollatasoon. Viestin osa on kaapattu oskilloskoopin differentiaaliprobella, joka summaa kahdesta linjasta saadun datan yhdeksi kuvaajaksi.

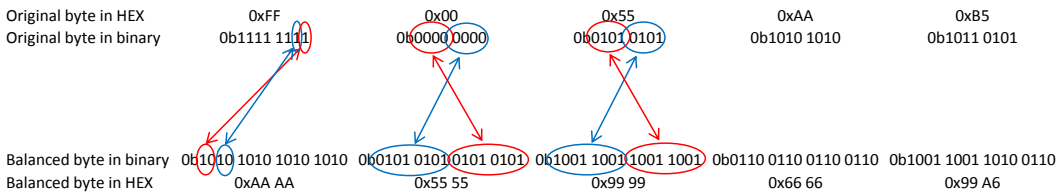


Kuvio 16. Tilan palaaminen nollatasoon.

4.6.1 Balansointi ohjelmallisesti

Balansoinnin tavoitteena oli, että samaa merkkiä ei lähetettäisi enempää, kuin kaksi kertaa peräkkäin (1 tai 0). Balansoinnin olisi todennäköisesti toteuttaa myös raudassa, mutta sitä ei tässä projektissa selvitetty.

Balansointi tehtiin kääntämällä lähetetystä tavusta jokaisen bitin perään bitin käänteinen arvo. Esimerkiksi 0000 olisi balansoituna 01010101 ja 11000010 olisi 1010010101011001. Lopullisessa balansointifunktiossa balansoitu arvo on ns. vähiten merkitsevä bitti ensin (kuvio 17.)



Kuvio 17. Balansointiesimerkkejä.

Balansoinnilla oli vaikutus myös saapuneiden viestien käsittelyyn, kun rengaspuskuria oli pakko laajentaa balansoinnista aiheutuvan kuorman vuoksi.

5 TESTAUS

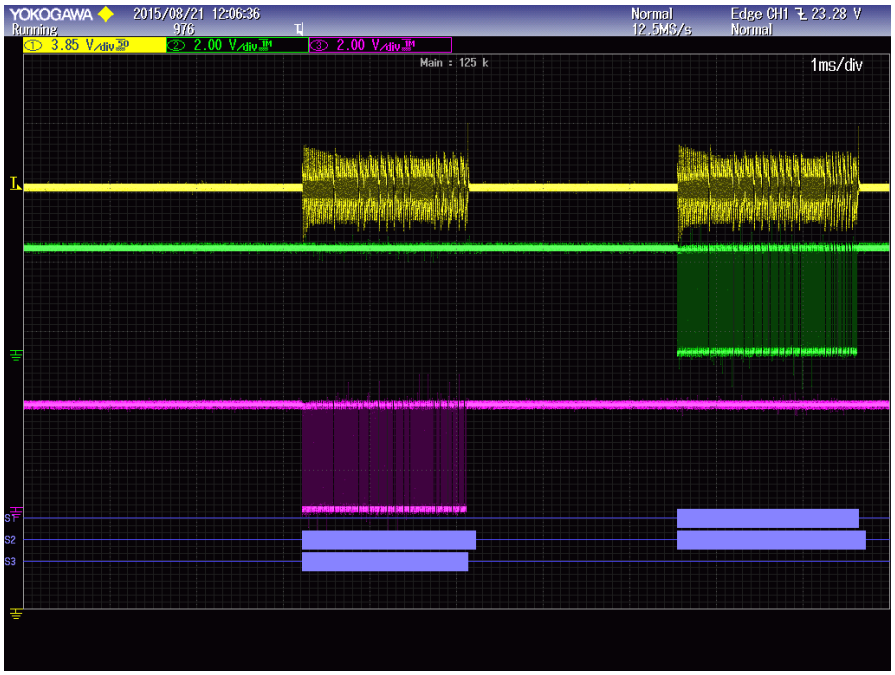
Tiedonsiirtoväylää testattiin ensimmäisen prototyypin jälkeen ja yleisesti verkon toimintaa testattiin jatkuvasti tuotekehityksen ohessa.

Väylää testattiin syöttämällä kierrettyyn parikaapeliin induktiivisesti sähköpulsseja. Testissä väylään kytkettiin kerrallaan 1 + 10 laitetta. Kaapeliin syötettiin induktiivisesti 0 - 40 kV jännitettä. Tässä vaiheessa ohjelmisto oli vielä alkuvaiheessa, ja ilmaantui epävakausongelmia. Seuraavaan kortin malliin päätettiin tämän pohjalta tehdä lisäys, jossa käytettäisiin suojattua parikaapelia, jonka vaippa johdettaisiin kortin alumiinikoteloon.

Ohjelmistoa ja ajoitusta testattiin lähettämällä verkossa viestejä. Varsinkin viestien parsinta on verkon toiminnan kannalta tärkeä, koska viestissä voi olla monia eri elementtejä, jotka tulee ottaa huomioon. Ajoituksella tarkoitetaan tässä kuinka pitkä tai lyhyt laitteen nukkuminen on ohjelmiston kannalta hyvä. Huomattiin, että pitkillä nukkumisilla rengaspuskuri täyttyy, eikä kaikkia viestejä saada luettua.

Parsintaa seurattiin lähettämällä verkossa eri komennoilla viestejä, kun PC oli debug-yhteydessä slave- ja master-kortteihin. Näin pystyttiin käymään merkki kerrallaan koko rengaspuskurin sisältö läpi. Todettiin että parsinta toimii ja tiputtaa virheelliset viestit pois.

Verkon toimintaa seurattiin aktiivisesti koko kehityksen ajan vertaamalla lähetettyjen viestien määrää saatujen vastausten määrään. Todettiin, että on sopivaa implementoida master lähettämään kysely kolme kertaa, ennen kuin slave tiputetaan osoitetaulusta. Kun slave ei saa viestejä viiteen sekuntiin, aloittaa slave uudelleen osoitekyselyn.



Kuvio 18. Kysely ja vastaus.

Oskilloskooppikuvassa (kuvio 18) erottuu hyvin sekä kysely (violetti), että vastaus (vihreä) ja 24 voltin siirtolinja, differentiaaliprobella kaapattuna (keltainen). Vihreä ja violetti kuvaaja on otettu mikro-ohjaimien UARTin TX-pinneistä. Kuviossa nähdään, kuinka nukkumisen jälkeen laite vastaa kyselyyn. Jos kuvassa näkyisi enemmän kyselyjä ja vastauksia, nähtäisiin kuinka niiden välinen aika vaihtelee, riippuen missä vaiheessa slaven nukkumissykliä master lähettää kyselyn.

6 YHTEENVETO

Työn tarkoituksena oli luoda uusi verkko, jonka tarkoitus oli lisätä taajuusmuuttajasta kerättävän tiedon määrää. Tämän lisäksi verkkoon kytkettyjä laitteita voidaan ohjata, hitaammassa aikatasossa, kuin tavallisempia suoraan sisään- ja ulostuloihin kytkettyjä laitteita.

Työssä käytettiin monipuolisesti koulussa opittuja asioita, kuten reaaliaikakäyttöjärjestelmää, c-kieltä ja mikro-ohjaimia, sekä opittiin uutta, kuten protokollan ja ohjelmiston suunnittelua. Tämän lisäksi saatiin käytännön kokemusta koulussa sivutuista aiheista, kuten scrumista. Apunani oli laaja asiantuntijaverkosto, joka auttoi erilaisten laiteongelmien kanssa ja ohjelmiston debuggauksessa.

Työn tekovaiheessa oli vielä moni asia avoinna, mikä vaikutti osaltaan työn etenemiseen, mutta opin, että se kuuluu osaltaan tuotekehitykseen. Työhön jäi vielä tehtäväksi lopullisten, oikeiden laitteiden ohjaukseen käytettävät funktiot, jotka täytyy suunnitella kun tiedetään tarkasti millaisia ja miten ohjattuja laitteita verkkoon kytketään.

Lopputuloksena saatiin tiedonsiirtoverkko, jossa slavet saavat osoitteet masterilta ja master lähettää kyselyitä ja kirjoittaa lukuja slaveille. Prototyypissä luettavat arvot olivat arvottuja, kyselyiden määrän mukaan kasvavia ja AD-muuntimelta luettuja kokonaislukuja. Tämän lisäksi verkon yli voitiin ohjata kortin IO:ta ja asettaa relelähdön tila aktiiviseksi tai ei-aktiiviseksi.

LÄHTEET

/1/ Kauppalehti. 2015. Verkkouutinen. Viitattu 25.10.2015.
http://www.kauppalehti.fi/uutiset/vacon-siirtyi-osaksi-danfossia/kr9xf9jd

/2/ Drives&Controls. 2015. Verkkouutinen. Viitattu 25.10.2015.
http://www.drivesncontrols.com/news/fullstory.php/aid/4802/Merged_Danfoss-Vacon_business_sets_its_sights_on_ABB.html

/3/ Corrigan, S. 2002, päivitetty 2008. Introduction to the Controller Area Network (CAN) s. 2. Viitattu 7.12.2015.
http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

/4/ Theremino. Verkkoartikkeli. Viitattu 12.3.2015.
http://www.theremino.com/en/technical/dpm-vs-can

/5/ Modbus Organization. Verkkoartikkeli. Viitattu 24.1.2016.
http://www.modbus.org/faq.php

/6/ STMicroelectronics. Tuotesivu. Viitattu 23.4.2016.
http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF253215?sc=internet/evalboard/product/253215.jsp

/7/ Danfoss. 2015. Verkkouutinen. Viitattu 24.1.2016.
http://commercialrefrigeration.danfoss.com/newsstories/cc/change-to-nameplates-on-commercial-compressors/?ref=17179899256

/8/ STMicroelectronics. STM320308 ajuripaketti. Viitattu 24.1.2016.
http://www.st.com/web/en/catalog/tools/PF259447