Opinnäytetyö AMK

Elektroniikka

Elektroniikkasuunnittelu

2018

Eero Holmberg

MODBUS-DATALOGGERI

– Ilmanvaihtokoneen tiedonkeruulaite

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Elektroniikka

Joulukuu 2018 | 38

Eero Holmberg

MODBUS-DATALOGGERI

Opinnäytetyön aiheena oli luoda Swegonin ilmanvaihtokoneiden kanssa kommunikoiva tiedonkeruulaite. Tiedonkeruulaite rakennettiin Arduino-levyllä. Tiedonsiirtoprotokollana oli Modbus, jonka mukaista esitystapaa käytettiin RS485-standardin mukaisella tiedonsiirtoväylällä.

Tavoitteena oli tutustua Modbusiin ja RS485-väylään. Hyödyntää tieto toimivaksi C++ -kielen ohjelmaksi Arduino-levylle. Työssä tutustuttiin myös Atmelin valmistamaan Atmega328 -mikrokontrolleriin ja erityisesti sen USART-piirin ja ajastimien toimintaan. Työssä hyödynnettiin mahdollisimman paljon mikrokontrollerin keskeytystoimintoja joustavan ja tehokkaan koodin luomiseksi.

Kommunikaatio ilmanvaihtokoneen ja tiedonkeruulaitteen välillä suoritettiin onnistuneesti. Viiden lämpötila-anturin lukemasta luotiin lokitiedosto Arduinoon kytketylle SD-kortille. Tiedonsiirron signaalin kulkua tutkittiin oskilloskoopilla. Toimiva ja helposti muokattava työkalu rakennettiin ilmanvaihtokoneiden toiminnan tutkintaan.

ASIASANAT:

Modbus, RS485, tiedonsiirto, dataloggeri, protokolla

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Electronics

December 2018 | 38

Eero Holmberg

MODBUS DATALOGGER

The main goal of this thesis was to produce and develop a datalogger which communicates with Swegon air handling units. The datalogger was constructed by using an Arduino board. The data transition protocol is Modbus, which uses the RS485 standard defined communication line.

The focus of the study is on the use of the Modbus and the RS485 communication line. Finally, the required knowledge from the study transferred into a working C++ program on the Arduino board. Atmel's Atmega328 microcontroller is also introduced in the thesis, especially its USART circuit and counters operations. The microcontroller's interrupt operations were used as much as possible to create flexible and effective code.

The communication between the air handling unit and the datalogger was accomplished successfully. From five different temperature-sensor's readings, a logfile was created into a SD card that was connected to the Arduino. Serial communications were observed with the oscilloscope. A functional and upgradeable tool for the communication between the Swegon air handling unit and the datalogger was developed.

KEYWORDS:

Modbus, RS485, communication, datalogger, protocol

SISÄLTÖ

LIITTEET

KUVAT

KÄYTETYT LYHENTEET ¶

ADU	Application data unit, Modbussin protokolaosuuteen lisätty siirtotavasta riippuvainen osuus.
ASCII	American Standard Code for Information Interchange, Modbusissa vaihtoehtoinen datan esitystapa sarjaliikenneväylälle.
CO2	Carbon dioxide, hiilidioksidi. Mitattava asia ilmanvaihtokoneissa.
CRC	Cyclic redundancy check. Virheentunnistusmenetelmä Modbus RTU protokolassa.
DE	Driver enable, RS485-piirin lähetystilan salliminen.
DI	Driver input, RS485-piirin lähetykseen menevän signaali.
FAT32	File allocation table 32-bits, tiedostojärjestelmä ja datan tallennusmuoto SD-kortilla.
MISO	Master input slave output. Johtimen nimi SPI-sarjaliikenneprotokolassa.
MOSI	Master output slave input. Johtimen nimi SPI-sarjaliikenneprotokolassa
mV	Millivoltti, tuhannesosa Voltista.
PDU	Protocol data unit, siirtotavasta riippumaton protokolaosuus Modbussista.
R	Resistor, passiivinen elektroniikan komponentti, vastus.
RH	Relative humidity, suhteellinen kosteussisältö. Mitattava asia ilmanvaihtokoneissa.
RE	Receiver enable, RS485-piirin vastaanottotilan salliminen.
RO	Receiver output, RS485-piirin vastaanotto

RS485	Protokola balansoidulle differientaaliselle sarjaliikenneväylälle.
RTU	Remote terminal unit, Modbussin sarjaliikenneprotokola siirtoväylälle.
RX	Receive, USART-piirin vastaanoton nasta.
SPI	Synchronous peripheral interface, synkroninen sarjaliikenneväylä mikrokontrollerin ja SD-kortin välillä.
TVS	Transient voltage supression, elektroninen suojakomponentti suuria transientjännitteitä vastaan.
TX	Transmit, USART-piirin lähetysnasta
UL	Unit load. Määritelty yhden RS485 kommunikointipiirin verkkoa kuormittava vaikutus.
USART	Universal synchronous asynchronous receiver transmitter.
VOC	Volatile organic compound, haihtuvia orgaanisia yhdisteitä. Ilmanvaihtokoneista mitattava asia.

1 JOHDANTO

Työn toimeksiantajayrityksenä on Swegon Kaarinan tehdas. Swegon Kaarina on ilmanvaihtoalan laitevalmistaja, jonka bisnesalueena on asuntoilmanvaihto. Tärkeimpiä kehitettäviä ja valmistettavia tuotteita ovat ilmanvaihtokoneet ja liesikuvut. Swegonin Ilmanvaihtokoneissa on Modbus-liityntä vakiona. Modbus mahdollistaa ilmanvaihtokoneen ohjaamisen, mittaus- ja asetusarvojen lukemisen digitaalisessa, luotettavassa ja tarkastetussa muodossa.

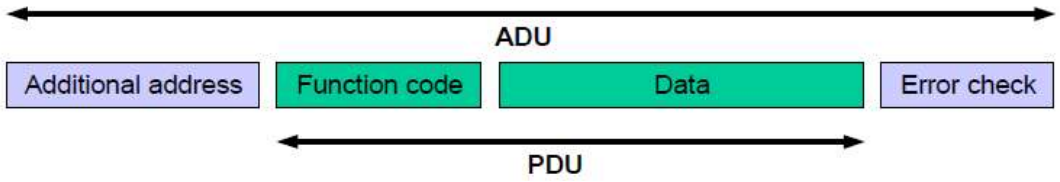
Swegonin Smart-tuoteperheen ilmanvaihtokoneet ovat täysin Modbus-yhteensopivia. Modbusia hyödyntäen voidaan luoda ilmanvaihtokoneen ulkopuolisia tehokkaita työkaluja koneen toiminnan analysoimiseksi ja ohjaamiseksi. Ulkopuolinen taloautomaatiojärjestelmä voi esimerkiksi lukea jokaisen väylään kytketyn ilmanvaihtokoneen hälytysrekisterit mahdollisten hälytyksien varalta etänä.

Ilmanvaihtokoneista voidaan vastaavasti kerätä myös datalokia. Ilmanvaihtokoneen mittausarvoista on nähtävissä koneen toiminta hyvin yksityiskohtaisesti. Ilmanvaihtokoneeseen on kytkettävissä vakiona olevien lämpötila-anturien lisäksi muun muassa RH-, CO2- ja VOC-anturi. Anturien lisäksi myös erilaisia koneeseen kytkettäviä laitteita on monia, kuten vesipattereita, ulkoisia kanavavastuksia ja -peltejä. Variaatioita on useita ja näiden kaikkien toiminta halutaan verifioida kenttä- ja laboratoriotesteissä.

Swegonilla on tarve luoda halpa dataloggeri, joka voidaan kytkeä Swegonin ilmanvaihtokoneeseen. Tämän opinnäytetyön aiheena on luoda kyseinen tiedonkeruulaite, joka kommunikoi luotettavasti ilmanvaihtokoneen kanssa Modbusia hyödyntäen. Dataloggerista luodaan prototyyppi Arduinon UNO-levyn ympärille. Data tallennetaan SD-kortille. Opinnäytetyössä perehdytään Modbusiin, RS485-väylään sekä Arduinon käyttämää, Atmelin valmistamaa Atmega328-mikrokontrolleriin.

2 MODBUS-PROTOKOLLA

Modbus-protokolla on vuonna 1979 julkaistu avoin ja lisenssimaksuton tiedonsiirtoprotokolla [1]. Modbus on yleisesti käytössä elektroniikkalaitteiden välisessä kommunikaatiossa ja mahdollistaa eri valmistajien laitteiden edullisen liittämisen keskenään. Protokolla on isäntärenkiprotokolla, jonka toiminnallisuudet ovat eri funktiokoodien alla. Funktiokoodit indikoivat, minkälainen toiminto suoritetaan ja missä muodossa datakehyksen tiedot ovat. Funktiokoodit ja datakehyksen tiedot muodostavat siirtolinjasta ja kommunikointitasosta riippumattoman protokolla osuuden PDU [2]. Siirtotapa määrittelee protokollaosuuden ulkopuolella olevat kehykset kuten esimerkiksi laitteen osoitteen tai virhetarkistusmenetelmän, joista muodostuu yhdessä PDU:n kanssa ADU [2, s.3]. Kuvassa 1 on esitettynä Modbus paketin rakenne yleisesti.



Kuva 1. Modbus-paketin rakenne yleisesti. [2, s.3]

2.1 Modbus data ja funktiokoodit

Modbus tukee kahta eri datamuotoa, 1 ja 16 bitin pituisia binääriarvoja. 16-bittinen eli kahden tavun kokoinen arvo jaotellaan aina kahdeksi yhden tavun arvoksi, joista suurempi esitetään ensin [2, s.5]. Jokaisella arvolla on oltava oma yksilöllinen paikkatieto laitteen muistissa, mihin rekisterinumerolla viitataan. Yhden bitin arvoista käytetään nimityksiä ”coils” ja ”discrete inputs”. Discrete input datatyyppi on vain luku-muotoinen ja coils arvoja voidaan myös kirjoittaa. 16-bittisistä arvoista käytetään nimityksiä input ja holding registers. Input registers ovat vain luku-muotoisia ja holding registereihin voidaan myös kirjoittaa [2, s.6].

Eri datamuodoille on määritelty erilaisia julkisia funktiokoodeja, jotka modbus.org-yhteisö on validoinut (kuva 2). Funktiokoodi esitetään viestissä aina riippumatta siitä, onko kyseessä kysymys vai vastaus.

				Function Codes		
				code	*Sub code*	*(hex)*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**
		Internal Bits Or Physical coils	Read Coils	**01**		**01**
			Write Single Coil	**05**		**05**
			Write Multiple Coils	**15**		**0F**
	16 bits access	Physical Input Registers	Read Input Register	**04**		**04**
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**
			Write Single Register	**06**		**06**
			Write Multiple Registers	**16**		**10**
			Read/Write Multiple Registers	**23**		**17**
			Mask Write Register	**22**		**16**
			Read FIFO queue	**24**		**18**
	File record access		Read File record	**20**		**14**
			Write File record	**21**		**15**
Diagnostics			Read Exception status	**07**		**07**
			Diagnostic	**08**	**00-18,20**	**08**
			Get Com event counter	**11**		**OB**
			Get Com Event Log	**12**		**0C**
			Report Server ID	**17**		**11**
			Read device Identification	**43**	**14**	**2B**
Other			Encapsulated Interface Transport	**43**	**13,14**	**2B**
			CANopen General Reference	**43**	**13**	**2B**

Kuva 2. Modbus, julkiset funktiokoodit. [2, s.11]

2.2 Rekisterin arvon lukeminen

Input ja holding rekisterien arvoja luetaan funktiokoodeilla 3 ja 4. Protokolla määrää funktion kysymyksen ja vastauksen muodon. Kysymyksen dataosuus alkaa kahden tavun pituisella aloitusrekisterin osoitteella, joita seuraa luettavien rekisterien määrä esitettynä 2 tavun pituisena lukuarvona. Vastauksen protokollaosuus muodostuu yhden tavun pituisesta luettavien datatavujen määrästä ja kahden tavun pituisista rekisterien arvoista. Jokaisen rekisterin arvo esitetään 2 tavun pituisena arvona, joista ensimmäinen tavu sisältää eniten merkitsevät bitit. Koska vastauksen datatavujen määrä esitetään yhden tavun pituisena arvona, rajoittuu kerralla kysyttävien rekisterien määrä 127:ään. [2, s.15-17]

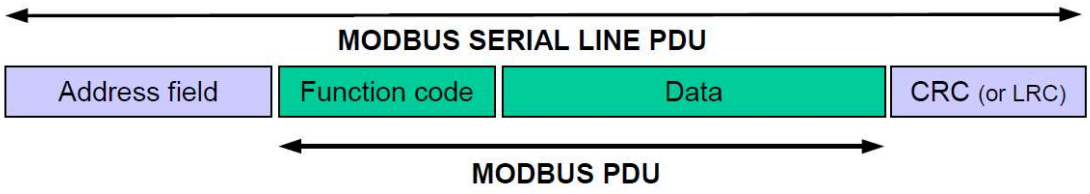
Esimerkiksi voidaan lukea kaksi peräkkäistä rekisteriä funktiokoodilla 3. Ensimmäinen luettava rekisteri olkoon 45067 eli heksadesimaaliarvo ”B00B”. Protokollaosuuden mukainen kysymyksen muoto on tällöin: ”03 B0 0B 00 02”. Rekisterin 45067 arvo olkoon

tässä esimerkissä 1 ja rekisterin 45067 arvo olkoon 2. Vastaus alkuperäiseen kysymykseen on tällöin: ”03 04 00 01 00 02”. Funktiokoodiin 3 vastataan neljällä tavulla, joista 00 01 vastaavat ensimmäistä rekisteriä ja seuraavat kaksi tavua toista rekisteriä. Jos kysyttäviä rekistereitä olisi 127, alkaisi vastaus tavuilla ”03” ja ”FF”. Suurin yhden tavun eli 8 bitin kokoinen heksadesimaaliarvo on ”FF”, mikä vastaa desimaaliarvoa 255.

2.3 Modbus sarjaväylällä

Modbus määrittelee kaksi eriävää siirtotapaa sarjaväylälle, RTU ja ASCII. Swegonin ilmanvaihtokoneet kommunikoivat Modbus RTU:n mukaisesti. Modbus RTU verkossa olevilla laitteilla on jokaisella yksilöllinen osoite. Osoitteen arvo on yhden tavun pituinen ja sijoittuu viestin alkuun. Viestin lopussa on kahden tavun pituinen CRC, joka muodostuu laskennallisesti viestin muista tavuista [3]. Osoitteen ja CRC:n välillä on siirtoväylästä riippumaton protokollaosuus. CRC eli cyclic redundacy check on yleisesti käytetty virheentunnistusmenetelmä tietotekniikassa [4]. Modbus-paketin muoto sarjaväylällä on esitetty kuvassa 3.

Kappaleen 2.2 esimerkin protokollaosuuteen lisätään nyt renkilaitteen osoite ja CRC-tavut. Renkilaite olkoon osoitteella 1. Nyt kahden peräkkäisen rekisteriarvon kysely, alkaen rekisteristä 45067 muodostuisi tavuista: ”01 03 B0 0B 00 02 09 93”. Rekisterien arvot olkoon 1 ja 2, jolloin vastaus muodostuisi tavuista: ”01 03 04 00 01 00 02 32 2A”. Protokollaosuuteen on nyt lisätty RTU:n määräämä virheentarkistus ja renkilaitteen osoite.



Kuva 3. Modbus-paketti sarjaväylällä [3, s.8].

2.4 Viestin ja tavun muodostus siirtoväylälle

Modbus RTUn jokainen tavu sisältää 8 databittiä, aloitusbitin, pariteettibitin sekä lopetusbitin. Eli yhden tavun siirtämiseen käytetään 11 bittiä. Mikäli pariteettibittiä ei käytetä, tulisi lopetus hoitaa kahdella bitillä [3, s. 12]. Viesti koostuu yhdestä kehyksestä, jonka pituus on maksimissaan 256 tavua, mikä sisältää renkilaitteen osoitteen, funktiokoodin, datan ja CRC-laskennan (kuva 4). Jokaisen kehyksen välissä oltava vähintään 3,5 merkkiä pitkä hiljainen jakso. Yksittäisen kehyksen sisällä ei saa olla yli 1,5 merkkiä pitkää taukoa [3, s. 13].

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low CRC Hi

Kuva 4. Modbus RTU-paketti [3, s.12]

2.5 CRC-laskenta

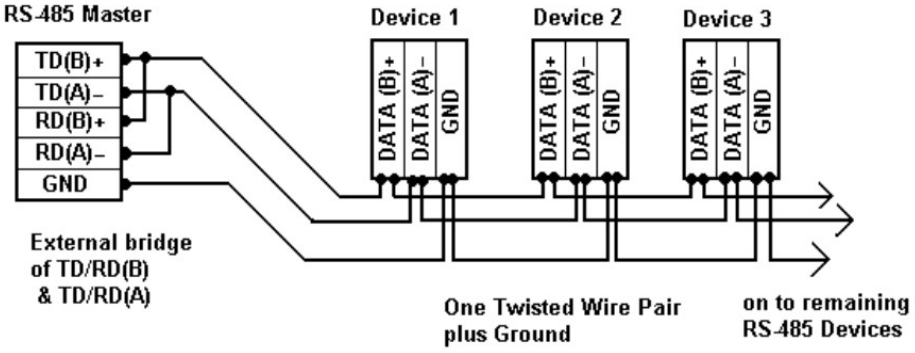
Modbus-protokolla käyttää CRC16-laskentamenetelmää virheellisesti tulkittujen tavujen havaitsemiseksi. CRC:n laskenta ottaa huomioon aina koko viestin sisällön riippumatta siirrettävän merkkijonon pituudesta. Jokaisen Modbus viestin lopusta löytyy kaksi CRC-tavua, joista vähiten merkitsevät 8 bittiä esitetään ensin [3, s.12]. Tavujen arvo on jakojäännös, joka muodostuu, kun viesti jaetaan tiedetyllä CRC-polynomilla binäärimuodossa. Lähetettävään viestiin lisätään CRC-tavut ja vastaanotettavasta datasta varmistetaan jakojäännöksen olevan 0 [5]. Modbusissa CRC-polynomina käytetään heksadesimaali arvoa 0xA001 [3, s.39].

3 TIEDONSIIRTOVÄYLÄ RS485

RS485 on vuonna 1983 hyväksytty standardi balansoidulle sarjaliikenneväylälle. Standardi määrittelee sarjaliikenneväylän elektronisten komponentit sekä sähköiset ominaisuudet eli fyysisen kerroksen. Väylästä on kahta eri versiota riippuen siitä, halutaanko väylässä tapahtuvan tietoliikennettä kahteen vai yhteen suuntaan. Kahden johtimen ratkaisussa vain yksi laite voi kommunikoida kerrallaan. Neljän johtimen väylässä on kaksi toisistaan eroteltua paria ja näin ollen kaksisuuntainen tietoliikenne on mahdollista. [6]

3.1 Verkon rakenne

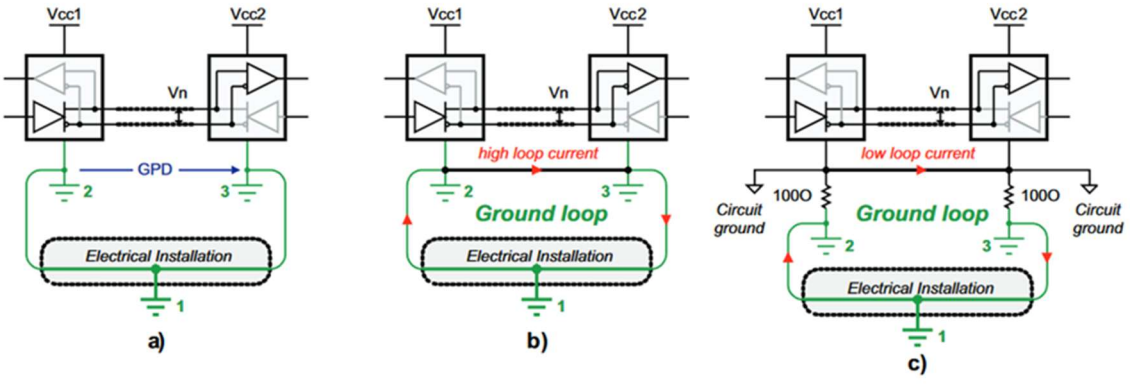
RS485-väylän laitteet suositellaan kytkettävän kuvan 5 mukaisella daisy chain -menetelmällä. Verkko muodostuu pääväylästä, johon kytketään kommunikoivat laitteet mahdollisimman lyhyillä johtimilla [6]. Pitkät johtimet lähetinvastaanottimen ja väylän välillä saattavat aiheuttaa heijastumia. Väylä tyypillisesti terminoidaan linjan molemmista päistä siten, että terminointivastuksen resistanssi vastaa linjan kaapelin ominaisimpedanssia. Standardi määrittelee väylään kytkettävien laitteiden maksimiksi 32 kappaletta, mikä perustuu oletukseen yhden standardinmukaisen kommunikointilaitteen verkkoa kuormittavasta vaikutuksesta, UL [7]. Komponenttivalmistajat myyvät kuitenkin verkkoa vähemmän kuormittavia kommunikointipiirejä, mikä mahdollistaa useampien laitteiden liittämisen RS485-verkkoon. Näin ollen myös Modbus protokollan tukemat 255 laitetta yhdessä verkossa ovat mahdollisia, mikäli käytetään 1/8 UL-luokiteltuja RS485-kommunikointipiirejä.



Kuva 5. RS485-verkko Daisy chain-menetelmällä [19].

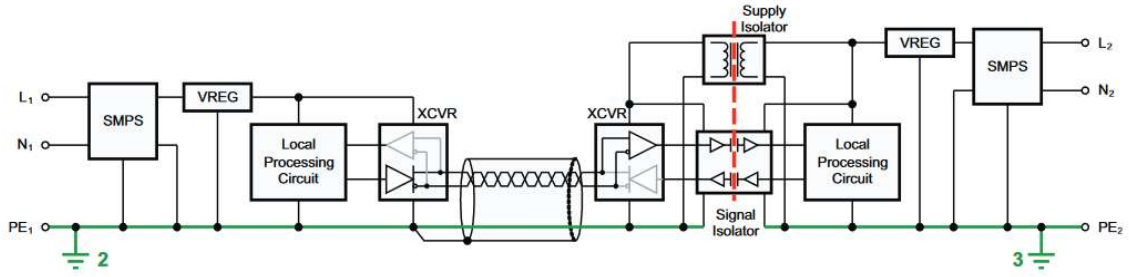
3.2 Maadoitus ja suojaus

RS485 ei määrittele tai ota kantaa väylän maadoitukseen [10], mutta yhteensopivan kommunikointilaitteen on pystyttävä toimimaan jännitealueella -7...12V. Pitkällä siirtolinjalla saattaa syntyä suuria potentiaalieroja suhteessa maahan. Väylän läheisyydessä saattaa olla myös suurivirtaisia verkkojännitettä kuormittavia ja häiritseviä laitteita [9]. Mikäli väylää ei ole lainkaan maadoitettu, saattaa yksittäisen kommunikointilaitteen käyttöjännitealue ylittyä ja näin ollen lakata toimimasta (kuva 6, kohta a). Väylää ei tule myöskään kytkeä suojamaahan useassa eri pisteessä, koska tällöin suojamaan jännitteen potentiaalierot purkautuvat väylän maajohtimen kautta (kuva 6, kohta b). Suuri virta datakaapelin ja kommunikointilaitteiden läpi saattaa rikkoa johtimen tai laitteet. Kolmantena huonona vaihtoehtona on rajoittaa virtaa piirikortin maan ja suojamaan välillä väylän eri pisteissä (kuva 6, kohta c). Tämä ei poista maankierron ongelmaa, mutta vähentää sitä ja on eräänlainen kompromissi. [8]



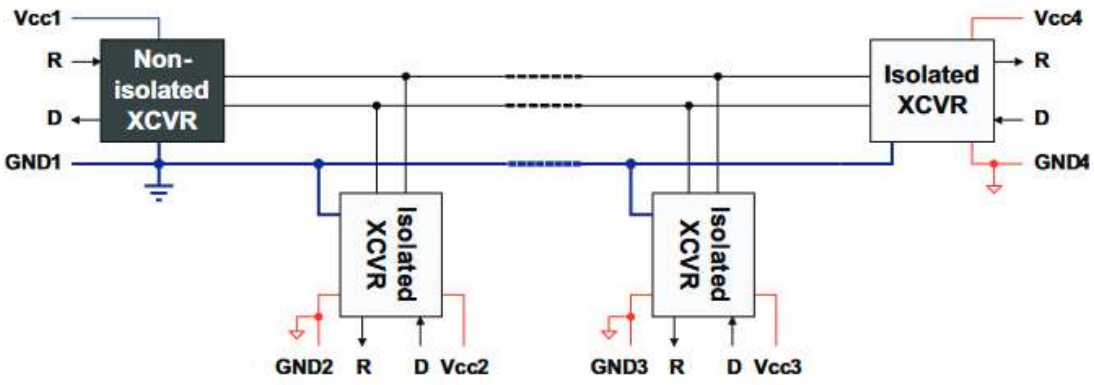
Kuva 6. RS485-väylän maadoituksen sudenkuopat [8].

Suurien maan potentiaalierojen varalta väylä maadoitetaan yhdestä pisteestä ja kaikki muut toimilaitteet väylällä erotetaan galvaanisesti paikallisesta maasta. Kuvan 7 kytkennässä vastaanottopiiri on lähetyspiirin potentiaalissa. RS485-sovittimen signaalijohtimet prosessointiyksikölle ja jännitelähde ovat erotettuna paikallisesta jännitepotentiaalista ja näin ollen potentiaalierot väylään kytkettävien laitteiden välillä eivät vaikuta siirtoväylään tai kommunikointilaitteisiin. Galvaaninen erotus poistaa maankierron virrat [9, s.13].



Kuva 7. Suojaerotettu RS485-väylä [8].

Kuvan 7 ratkaisua voidaan soveltaa kuvan 8 useamman laitteen verkossa. Väylän maadoitus tapahtuu yhdestä pisteestä ja muut väylään kytkettävät laitteet jätetään kellumaan niiden paikallisesta potentiaalista. Maadoitus tapahtuu yleensä isäntälaitteella ja renkilaitteiden osalta on huomioitava vain, että ne ovat suojaerotettuja paikallisesta potentiaalista käyttösähkön ja datalinjojen osalta.



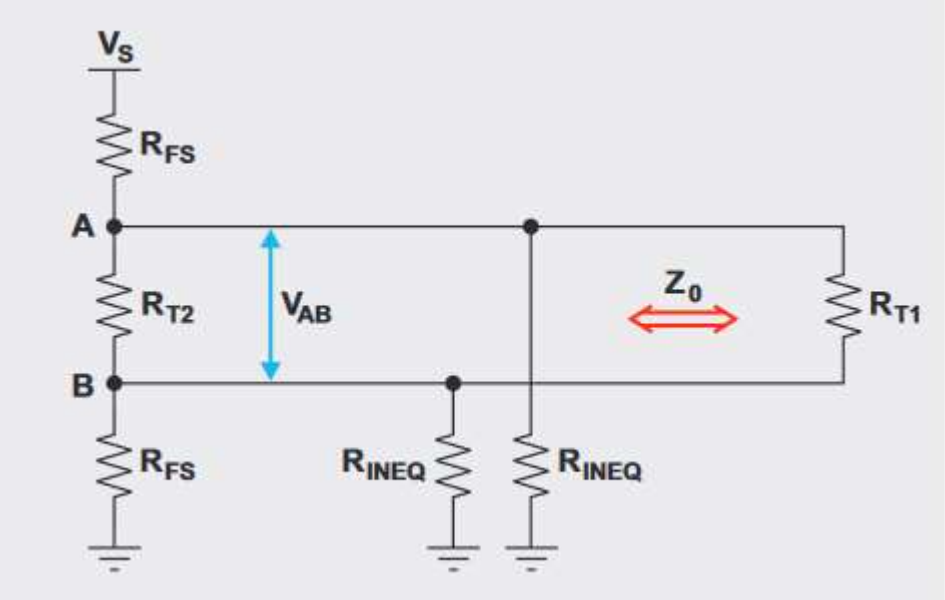
Kuva 8. Usean solmun suojaerotettu RS485-verkko [8].

Kaapeloinniksi suositellaan suojattua ja kierrettyä parikaapelia. Mahdollinen sähkömagneettinen säteily vaikuttaa molempiin linjoihin yhtäläisesti ja jännite-ero linjojen A ja B välillä pysyy likimain vakiona. Joissain RS485-väylän sovelluksissa

saattaa ilmetä impulssihäiriöitä esimerkiksi salaman iskun aiheuttamana, jolloin tulisi harkita esimerkiksi TVS-diodien kytkemistä väylän ja komponenttien suojaksi. [6]

3.3 Biasointi ja terminointi

RS485-standardinmukaisille kommunikointilaitteille on määritelty vastaanotettavan signaalin jännitetaso. Signaali, jonka jännite-ero väylän A:n ja B:n välillä on yli 200 mV tulkitaan tulos digitaaliarvoksi 1. Jännite-eron ollessa pienempi kuin -200 mV tulkitaan digitaaliarvoksi 0. Tämä tarkoittaa sitä, että jännite-eron ollessa itseisarvoltaan pienempi kuin 200 mV, väylän tilaa ei ole määritelty ja vastaanottopiirit voivat olettaa väylän tilan olevan 0 tai 1. Väylän ollessa odottavassa tilassa, linjat A ja B kelluvat, mistä johtuen vastaanotossa saattaa tapahtua virheellisiä tulkintoja. [11]



Kuva 9. RS485-väylän biasointi vastuksilla R_{FS} [11].

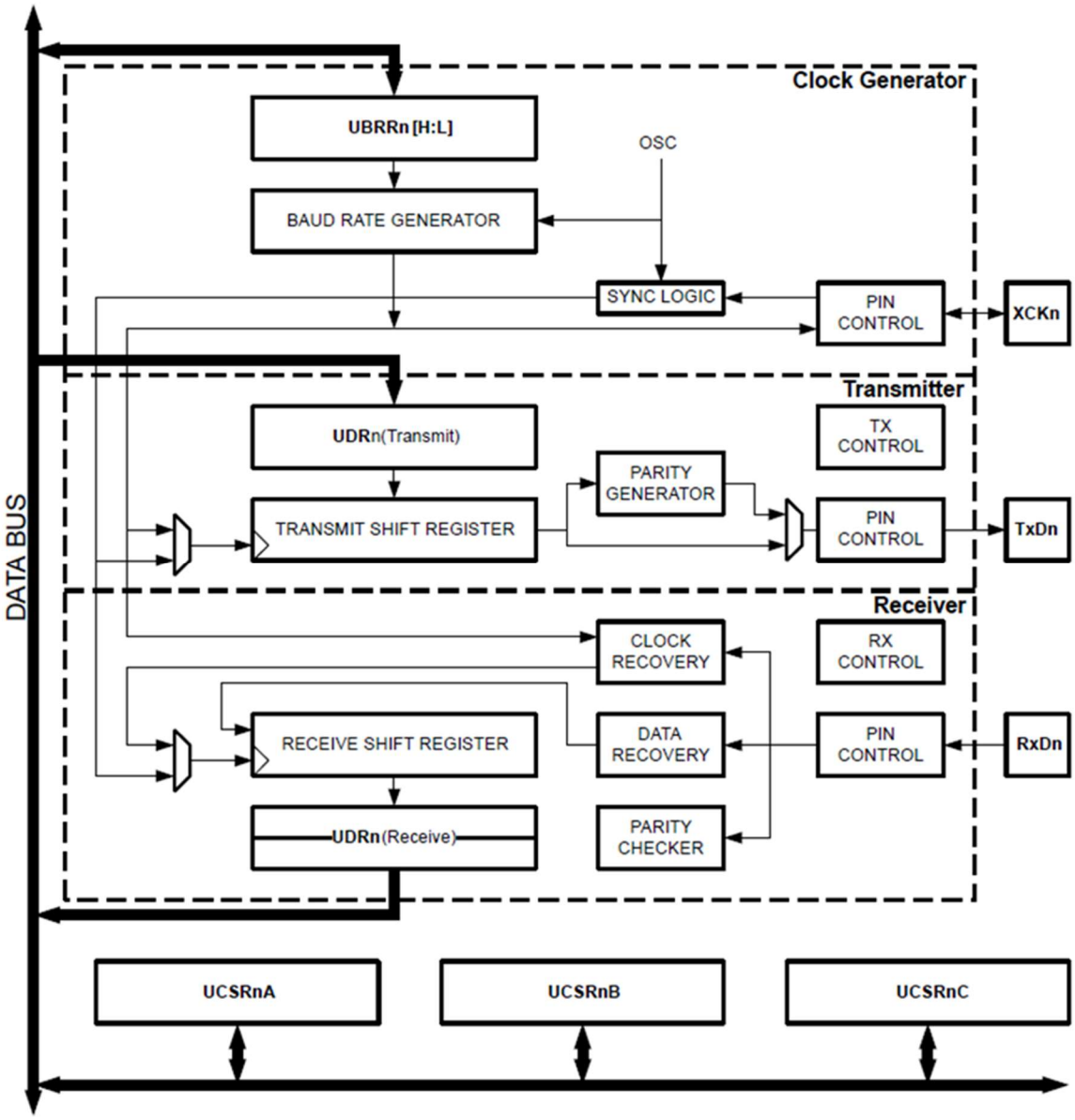
Häiriöiseen ympäristöön asennettava RS485-verkko suositellaan biasoimaan ulkoisilla ylös- ja alasvetovastuksilla kuvan 9 mukaisesti. Vastukset muodostavat jännitejaon ja pakottaa väylän tiedettyyn tilaan. Vastusarvo R_{FS} on riippuvainen verkon rakenteesta ja kytkettävistä vastaanottimista. Standardin mukaisen laitteen on pystyttävä luomaan 1,5 V:n jännite-ero linjojen A ja B välille. Vastaanotossa samainen jännite-ero on 200 mV. Verkkoa täytyy voida kuormittaa 32 UL:n (R_{INEQ}) verran, mikä vastaa resistanssi arvona noin 375 Ω [11]. Vastusarvot tulevat valita siten, että lähetyksen ja vastaanoton jännite-erot toteutuvat 32 standardinmukaisen laitteen kuormituksella.

4 ARDUINO

Arduino on avoimeen lähdekoodiin perustuva elektroniikka-alusta, sekä ohjelmointiympäristö. Arduino-levyt pohjautuvat Atmelin mikrokontrollereihin, joiden ympärille on lisätty tarpeenmukainen elektroniikka ja liittimet helpon käytettävyyden varmistamiseksi. Tarpeenmukaisella elektroniikalla tarkoitetaan esimerkiksi mikrokontrollerin vaatimia suodatuksia, jännitteen syöttöä ja ulkoista oskilaattoripiiriä. Arduino UNO-levy sisältää Atmelin atmega328P-mikrokontrollerin, mikä sisältää UART- ja SPI-piirin [20]. UART-piiriä käytetään tässä työssä ajamaan RS485-sovitinta ja SPI-piiriä SD-kortin kanssa kommunikointiin.

4.1 USART-piiri

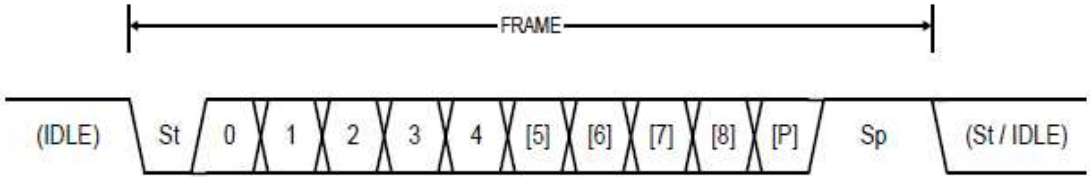
AVR atmega328P-mikrokontrollerissa on sisäänrakennettu USART-piiri. USART on jaettavissa kolmeen lohkoon, pulssigeneraattoriin, lähettimeen ja vastaanottimeen. Lähetin ja vastaanotin voidaan muokata tukemaan Modbus protokollan mukaisen tavun esitystapaa. Lähetys tapahtuu kirjoittamalla lähetyspuskuriin yhden tavun, jonka bitit piiri näytteistää sisäänrakennetun pulssigeneraattorin määräämällä nopeudella Arduinon TX-nastaan. USART-piirin lohkokaavio on esitettynä kuvassa 10. [12]



Kuva 10. USART-piirin lohkokaavio. [12, S.226]

USART-piirin ensimmäinen bitti eli aloitusbitti on aina 0, jota seuraa konfiguroinnista riippuen viidestä yhdeksään databittiä. Käyttötavasta riippuen piiri voidaan ohjelmoida laskemaan pariteettibitti ”odd”, ”even” tai ”none”. Kehykseen kuuluu aina vähintään yksi lopetusbitti. Piiri voidaan konfiguroida antamaan kaksi lopetusbittiä. USART-piirin

datakehys on esitettynä kuvassa 11. Kun kommunikointia ei ole, väylän tila on oltava 1. [12, s. 229]



Kuva 11. USART-piirin datakehys. [12, s. 229]

Datan vastaanotossa USART-piiri näytteistää vastaanotetut bitit normaalissa toimintamoodissaan 16-kertaisella nopeudella suhteessa väylän siirtonopeuteen, eli yhdestä bitistä otetaan 16 näytettä. Näytteistä kolme keskimmäistä määrää, tulkitaanko bitin arvoksi 1 vai 0 riippuen kumpaa kolmesta merkitsevästä näytteestä oli enemmän. Näin ollen 16 MHz:n kellotaajuudella maksimi siirtonopeus on normaalisti 1 Mb/s. Piiri on myös ohjelmoitavissa toimimaan kaksinkertaisella nopeudella, jolloin jokaisesta bitistä otetaan 8 näytettä ja maksimi siirtonopeus nousee 2 Mb/s:iin. [12, s.237-243]

4.2 SD-kortti

Arduinoon on helposti kytkettävissä erillinen SD-kortinlukija. Mikrokontrollerin ja SD-kortin välinen kommunikaatio tapahtuu SPI-väylällä. SPI on synkroninen kaksisuuntainen tiedonsiirtoväylä, joka on tarkoitettu etäisyydeltään lyhyeen tiedonsiirtoon. Synkronointipulssi osoittaa, koska väylän tilaa tulisi lukea. Lähetyspiirin ja vastaanottopiirin erillinen tahdistus, esimerkiksi 38400 tavua sekunnissa ei ole relevanttia. SPI-väylän vastaanottopiiri on näin ollen huomattavasti yksinkertaisempi kuin esimerkiksi UART-väylän vastaava. Jokaisella orjalaitteella on oma valintalinja ja yhteinen dataväylä. Dataväylän linjat ovat nimeltään MISO ja MOSI. [15]

SD-kortin tiedoston tallennusmuoto on FAT32. FAT32 on Microsoftin kehittämän ja käyttämä tiedostojärjestelmän. Tiedostojärjestelmän tarkoituksena on saada massamuistilaitteen, kuten SD-kortin sisältämä data helposti käsiteltävään ja luettavaan muotoon. Muistikortin maksimikapasiteetti rajoittuu kokoon 32 GB. 32 GB on Microsoftin luoma rajoitus muistikortin alustuksen yhteydessä, joka perustuu siihen, että FAT32:n suorituskyky heikkenee tiedostokoon kasvaessa. Suurempien datojen osalta tulisi käyttää muita tiedostojärjestelmiä [14]. SD-kortin muisti on pitkäkestoinen eli

palautettavissa ja luettavissa uudelleensähköistyksen jälkeen [13]. Arduinon oma julkinen kirjasto SD.h on luotu tukemaan FAT32-tiedostojärjestelmää ja tallennettava lokitiedosto on yhteensopiva Microsoft-käyttöjärjestelmän kanssa [16].

SD-kortille tallennettaan lokia tekstitiedostona, joka tallennetaan .csv-päätteellä. CSV-tiedosto voidaan avata esimerkiksi Microsoft Ofiicen Excel-tietojenkäsittelyohjelmalla. Tekstitiedostoon syötetään sarakkeen vaihdon merkiksi ”;”, ja rivin vaihdon merkiksi kirjoitetaan uudelle riville, mikä C++ -kielessä voidaan toteuttaa komennolla ”\n”. [17]

4.3 Ajastukset

Arduinon Atmega328P-mikrokontrollerissa on sisäänrakennettuna kolme ajastinta tai laskuria. Sisäänrakennettua laskuria käyttämällä voidaan toteuttaa riittävän tarkat ajastukset dataloggerille. Dataloggerin minimivaatimukset ajastuksille on määrittää loggaustiheys, kuinka nopeasti samainen arvo luetaan uudestaan. Samoin jokaiselle kyselylle tulisi määrittää, kuinka kauan vastausta odotetaan. Mikrokontrollerin laskuri päivittyy aina, kun mikrokontrollerin kellotaajuus tai ulkoinen oskillaattori päivittyy. Laskurille voidaan määritellä skaalausarvo 1, 8, 64, 256 tai 1026 [12, s.168]. Skaalausarvo 64 tarkoittaa tarkoittaa, että oskillaattorin 64 pulssia siirtää laskurin arvoa yhdellä. Laskuri voidaan ohjelmoida esimerkiksi luomaan keskeytys 10 ms:n välein.

4.4 Mikrokontrollerin keskeytystoiminnot

Interrupt eli keskeytys syntyy, kun järjestelmän osa tai toiminto vaatii erityishuomiota käyttöjärjestelmältä. Esimerkiksi mikrokontrollerin ajastin voidaan ohjelmoida luomaan keskeytyspyyntö tietyin väliajoin ja näin ollen voi toimia esimerkiksi kellon päivityksenä. Keskeytys tapahtuu mahdollisimman pian riippumatta, mitä toimintoa ohjelma on sillä hetkellä suorittamassa. Keskeytyksen alla voi olla oma ohjelman osa, mikä suoritetaan heti seuraavaksi. Interrupteja käytettäessä on huomioitava, että se nimensä mukaisesti aiheuttaa keskeytyksen laitteen toiminnalle. Keskeytyksiä ei tule tehdä liian usein, eikä keskeytys tule olla tarpeettoman pitkä. Tämä saattaa heikentää merkittävästi varsinaisen koodin suorituskykyä. [18]

5 TOTEUTUS

Koodikieleksi valittiin C++ ja ohjelmistoalustaksi Atmel Studio. Arduinon omia kirjastoja pyrittiin välttämään mahdollisimman paljon, jolloin toteutus olisi mahdollisimman käytännönläheinen ja mikrokontrolleria hyödyntävä. Modbus-pakettien luonti, lähetys ilmanvaihtokoneelle ja vastaanotto ilmanvaihtokoneelta onnistui ilman Arduino-kirjastoja. Kommunikointi RS485-sovittimelle toteutettiin mikrokontrollerin USART-piiriä hyödyntäen. SD-kortille kirjoittamiseen ja datan järjestelyyn FAT32-muotoon käytettiin valmiita arduinon oletuskirjastoja.

5.1 Modbus-pakettien luominen

Paketin luonti C++ -kielellä ja Atmel studio 7 ohjelmalla. Ensimmäinen tavu on orjalaitteen osoite, mikä oli tässä työssä määritelty arvoksi 1. Swegonin ilmanvaihtokoneen osoite on vaihdettavissa käyttöpaneelilla tai modbus rekisterillä ja tämän on vastattava kuvassa 12 luodun arvojoukon kohtaan ”FC[0]” syötettyä osoitetta. Koneen mittausarvot ovat luettavissa funktiokoodilla 4, mikä vastaa kuvassa kohtaa ”FC4[1]”. ”FC[2]” ja ”FC[3]” muodostuvat luettavan rekisterin 16-bittisestä järjestysnumerosta, joka jaetaan kahdeksi 8-bittiseksi arvoksi. ”FC[4]” on aina 0, koska Modbus-protokollan mukaan vastauksessa luettavien tavujen määrä esitetään 8-bittisenä arvona. Koska yksi rekisteri on 16-bittinen ja rekisterin arvo on 16-bittinen, voidaan lukea yhdellä kyselyllä maksimissaan 125 rekisteriarvoa kyseisellä funktiokoodilla [2]. ”FC[5]” on peräkkäisten rekisterien määrä. Jos useita peräkkäisiä rekisteriarvoja on useita, voidaan nämä sisällyttää yhteen kyselyyn. ”FC[6]” ja ”FC[7]” ovat paketin kaikista aikaisemmista arvoista laskettu 16-bittinen CRC-arvo, mikä on jaoteltu kahteen 8-bittiseen lukuun.

```
void packet(){
            FC4[0] = addr;
            FC4[1] = fnct;
            FC4[2] = (uint8_t)(regs[regno]>>8); //hi byte
            FC4[3] = (uint8_t)(regs[regno]); // lo byte
            FC4[4] = 0x00; // reg count hi, always 0
            FC4[5] = 0x01;

            while(regs[regno+1] == regs[regno]+1){
            //if next reg is +1, we can add it to same packet
                        FC4[5]++;
                        regno++;
```

```
            }
            bytecnt = 2*FC4[5];
            crc = 0xffff;
            for(i=0; i<=5; i++){
                crc = _crc16_update(crc, FC4[i]);// crc calc <util/crc16.h>
            }
            FC4[6] = (uint8_t)crc;
            FC4[7] = (uint8_t)(crc>>8);

            if(regno < (regcnt-1)){
                        regno++;
            }
            else regno = 0;
            PORTD |= (1<<PORTD2);
}
```

Kuva 12. Modbus-paketin luonti c++ -kielellä.

5.2 Modbus-pakettien liikuttaminen USART-väylällä

Pakettien luonnin jälkeen RS485-sovitin laitetaan lähetystilaan laittamalla mikrokontrollerin ”PORTD2” asentoon 1. Mikrokontrollerin porttia D2 vastaava digitaalinen ulostulo on kytketty RS485-sovittimen tilanohjaukseen. Paketin sisältö jaoteltiin 8 bittisiksi arvoiksi, jotka syötetään Atmega328-mikrokontrollerin UART-piirin puskuriin puskurin määräämällä nopeudella (kuva 13).

```
    packet();
    for(i=0;i<=7;i++){
        USART_Transmit(*(FC4+i));
        }
```

Kuva 13. Paketin syöttö USART-piirille.

Lähetys tapahtuu omassa kuvassa 14 esitetyssä funktiossa. USART-piirin puskuriin syötetään lähtevä merkkijono. Funktiossa odotetaan, kunnes puskurissa on tilaa uudelle tavulle, jolloin päällekirjoitus ei ole mahdollista.

```
void USART_Transmit (unsigned char data){
  while (!(UCSR0A & (1<<UDRE0)));
  UDR0 = data;
}
```

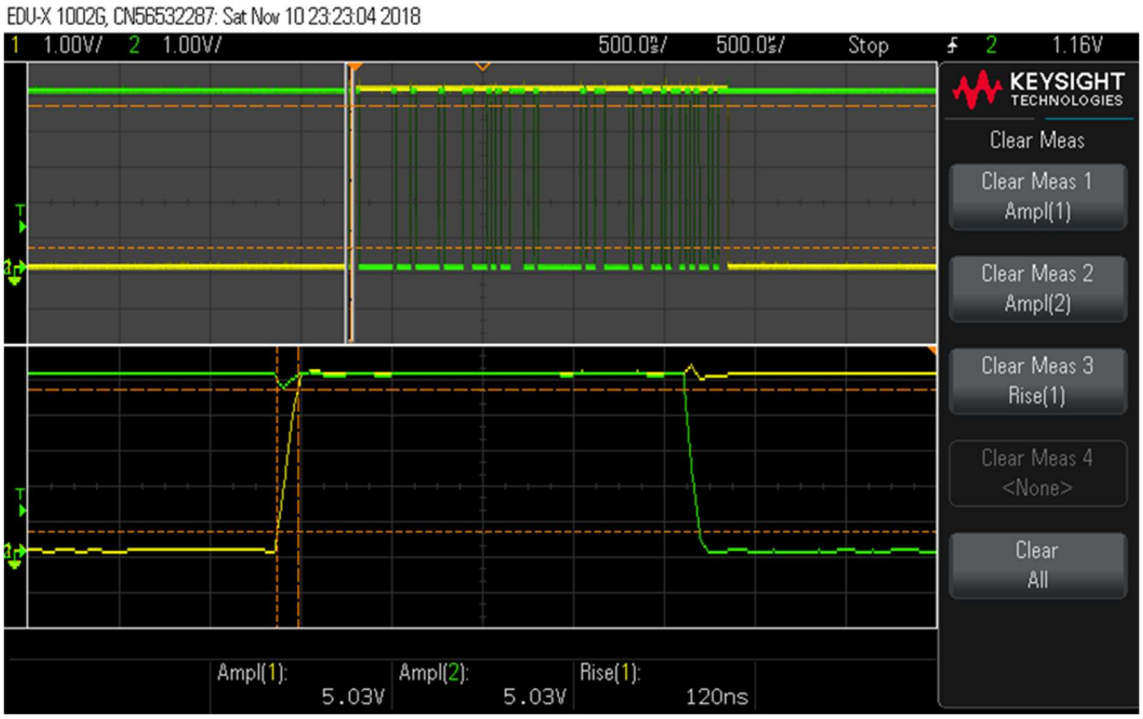
Kuva 14. Paketin syöttö USART-piirin puskuriin.

Keskeytys tehtiin myös USART-piirille, mikä aktivoituu, kun kaikki puskuriin syötetyt merkit ovat näytteistetty TX-nastalle, eikä mitään lähetettävää enää ole. Puskurin tyhjennettyä, RS485-sovitin laitetaan vastaanottotilaan asettamalle PORTD2 arvoon 0 kuvan 15 osoittamalla tavalla.

```
ISR (USART_TX_vect){
            PORTD &= ~(1<<PORTD2);
}
```

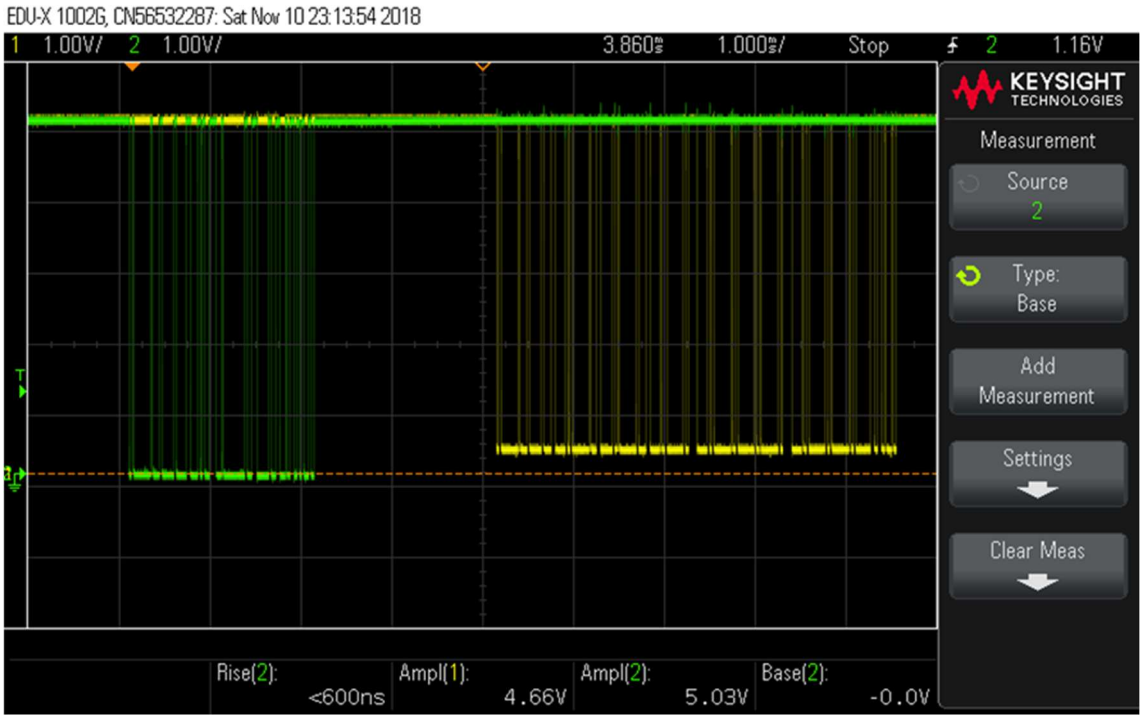
Kuva 15. RS485-ajurin ohjelmointi vastaanottotilaan.

Koodin toimivuus tarkistettiin oskilloskoopilla, jonka mittaustulos on esitettynä kuvassa 16. RS485-ajurin tilanohjaus on koko lähetyksen ajan aktiivisena ja tämän jälkeen palautuu vastaanottotilaan. Tilanohjaus on kuvassa 16 keltaisella ja USART-piirin lähettävä TX-nasta vihreällä.



Kuva 16. USART-piiriltä lähtevä signaali ja RS485-ajurin tilan ohjaus.

Lähetyksen jälkeen alkaa vastauksen odotus. Vastaus saapuu, jos saapuu, minkä vuoksi ohjelmalle on annettava maksimi aika vastauksen odotukselle. Jos viesti on mennyt perille ja vastaus saapuu, näytteistetään tämä USART-piirin puskuriin. Kuvan 17 kysymykseen saadaan vastaus noin 2 ms:n jälkeen. Vastauksen on lähettänyt ilmanvaihtokoneen piirikortti ja näin ollen kommunikaatio on onnistunut.



Kuva 17. USART-piirin lähetys esitettynä vihreällä ja vastaanotto keltaisella.

Ennen vastaanottoa suoritetaan vastaanotolle varattujen muuttujien nollaus kuvassa 18 esitetyllä tavalla. Samoin vastaanoton tavuille varattu merkkijono alustetaan komennolla ”memset”, jotta edellinen hyväksytty paketti ei hyväksytä tulevaa pakettia ennenaikaisesti. Samoin ”timeoutTimer” asetetaan ennalta määrättyyn arvoon.

```
x=0;
timeoutTimer = timeout;
memset(msgRX, NULL, sizeof (msgRX));

while(validate()!=2){
      if (timeoutTimer == 0){
          break;
      }
}
```

Kuva 18. Vastaanotto, rekisterin alustukset ja odotus.

Vastaanotto tapahtuu keskeytyksellä. USART-piiri ilmoittaa mikrokontrollerille, kun uusi tavu on vastaanotettu. Tämä tavu lisätään msgRX-merkkijonoon ja siirretään merkkijono pointterin arvoa yhdellä. Vastaanotolle luotu koodi on esitettynä kuvassa 19. Merkkijonon pituudeksi on määritelty 64, joka voidaan tarvittaessa muuttaa suuremmaksi.

```
ISR (USART_RX_vect){
   msgRX[x] = UDR0;
  if(x<64){
    x++;
  }
  else x=0;
}
```

Kuva 19. Vastaanotettu data haetaan puskurista.

MsgRX-merkkijonoa tutkitaan jatkuvasti kuvan 20 funktiolla, kunnes merkkijonon sisältö täyttää kaikki ehdot, mitä viestin onnistunut vastaanotto tarvitsee. Nämä ehdot ovat; odotusten mukainen orjalaite vastaa, oikealla funktiokoodilla ja tavujen määrä on odotetut mukainen. Jos edellä mainitut ehdot toteutuvat, asetetaan funktion palautettavaksi arvoksi 1. Mikäli myös CRC-laskenta menee oikein, palautettava arvo on 2. Vastaanoton merkkijono päivittyy keskeytyksien avulla. Heti kun vastaanoton merkkijono on oikean muotoinen ja hyväksytty, on vastaanotto onnistunut.

```
int validate(){
val = 0; //reset

if(msgRX[0] == FC4[0]
   && msgRX[1] == FC4[1]
   && msgRX[2] == bytecnt){
    val = 1; //right slave, function and expected bytecount
    crc = 0xffff;
    for(i=0; i<=bytecnt+2; i++){
        crc = _crc16_update(crc, msgRX[i]);
    }
    if(msgRX[bytecnt+3] == (uint8_t)(crc) && msgRX[bytecnt+4] ==
(uint8_t)(crc>>8)){
       val = 2; //crc ok
    }
}
    else{
         val = 3; //not valid
    }
return val;
}
```

Kuva 20. Vastaanotetun paketin hyväksyntä.

5.3 Ajastukset ja laskurit

Datalogger tarvitsee toimiakseen vähintään kaksi ajastusta. Loggausvälin sekä ajan, minkä verran vastausta odotetaan ja minkä jälkeen voidaan olettaa, että vastausta ei todennäköisesti tulla saaman. Nämä kaksi ajastusta luotiin käyttämällä mikrokontrollerin sisäistä ajastinta ”TIMER1”. Ajastin ohjelmoitiin päivittymään noin 10 ms:n välein.

```
            TCCR1A = 0; //rset
            TCCR1B = 0; //reset
            TCNT1 = 0; //reset
            OCR1A = 19999; //count to 19999 approx 10ms
            //TCCR1A |= (1<<COM1A0); //toggle on compare match debug purpose.
Pin PB1
            TCCR1B |= (1<<WGM12) | (1<<CS11); //prescaler 8 and WGM12 refers
to CTC mode clear timer on compare match
            TIMSK1 |= (1<<OCIE0A); //interrupt enable on compare match
```

Kuva 21. laskurin alustus.

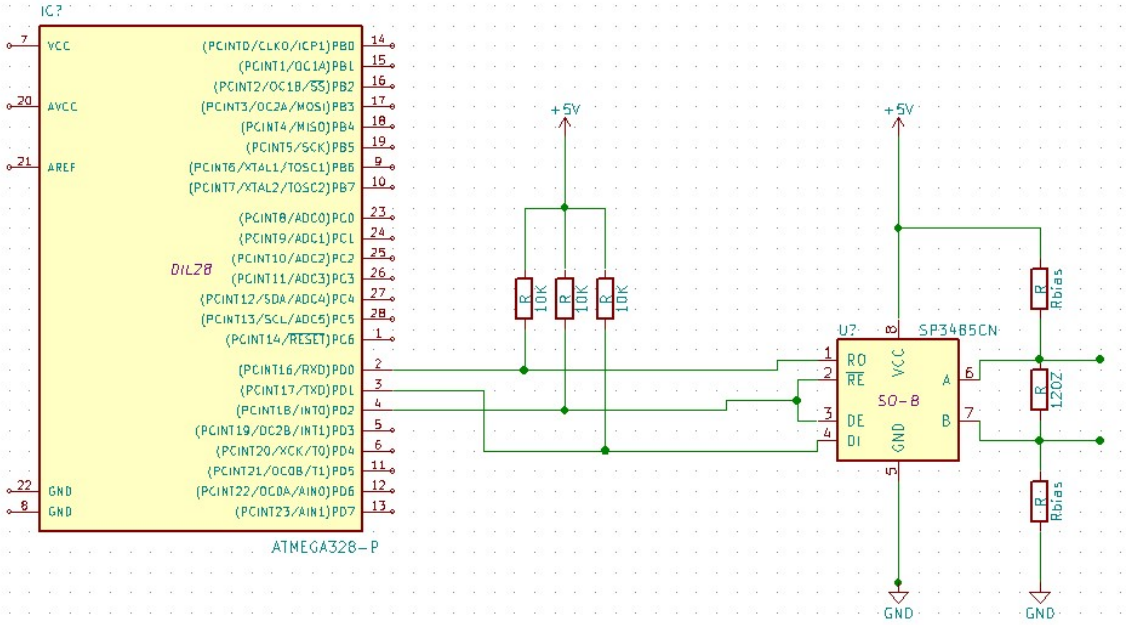
Muuttujien ”timeoutTimer” ja ”pollrateTimer” oli määriteltävä muotoon ”volatile unsigned int”, jolloin arvot päivittyivät keskeytystoiminnon aikaisessa koodiosiossa. Ilman kyseistä määritystä, kumpikaan muuttujista ei päivittynyt.

```
ISR (TIMER1_COMPA_vect){
            if(timeoutTimer!=0){
                        timeoutTimer--;
            }
            if(pollrateTimer!=0){
                        pollrateTimer--;
            }
}
```

Kuva 22. Ajastukset.

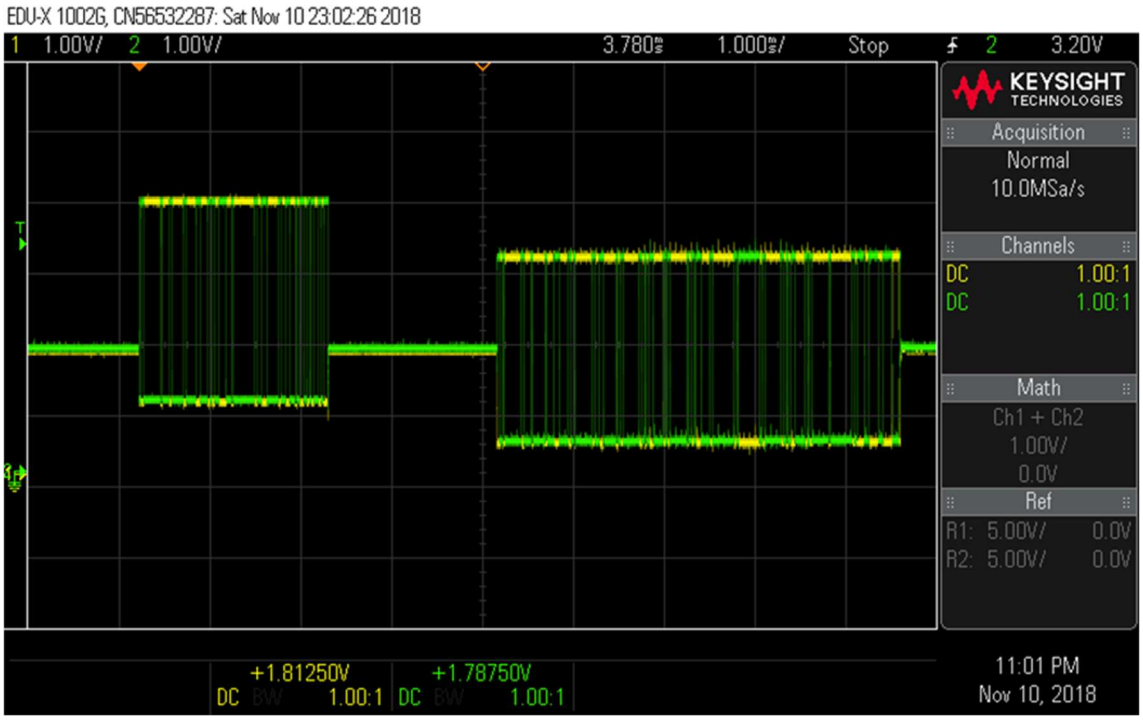
5.3 Pakettien siirto RS485-väylällä

RS485-sovitin kytkettiin arduinoon kuvan 23 mukaisesti. Sovittimen ja mikrokontrollerin väliset linjat kytkettiin 10 kΩ:in ylösvetovastuksien kautta käyttöjännitteeseen, jolloin linjat eivät pääse kellumaan. Ylösvetovastukset RX- ja TX-linjassa tarvitaan myös täyttämään USART-piirin määritys, lepotilassa USART-linjojen oltava digitaaliarvossa 1. Mikrokontrollerin RXD kytkettiin RS485-sovittimen RO-nastaan. TXD kytkettiin RS485-sovittimen DI-nastaan. Mikrokontrollerin PD2 kytkettiin ohjaamaan sovittimen tilaa eli invertoitua RE-nastaa, sekä DE-nastaa. Näin ollen RS485-sovitin on suoraan kytkettynä mikrokontrollerin UART-piiriin. Väylä maadoitetaan arduinon eli väylän isäntälaitteen potentiaaliin.



Kuva 23. Kytkentäkaavio.

Paketin luonti ja kysely ilmanvaihtokoneelle on suoritettu onnistuneesti. Ilmanvaihtokone vastaa odotuksien mukaisesti kyselyyn. Kuvassa 24 on esitettynä RS485-väylän linjat A ja B. Kuvan vasemmanpuoleinen paketti on kysely väylältä ja jälkimmäinen on ilmanvaihtokoneen vastaus. Oskilloskooppikuvasta nähdään, että sovittimen biasointi ei ole onnistunut. Väylä ei ole määritellyssä tilassa lähetyksen ja vastaanoton välisenä aikana. Käytetyssä valmiissa piirikortissa RS485-linjat A ja B olivat biasoitu 20 kΩ:in vastuksin, mikä on pienoinen mysteeri. Lähetyksen aikana väylällä näyttää olevan myös jonkinnäköinen epäsymmetria. Kommunikaatio on kuitenkin onnistunutta ja jännitetasot kommunikoinnin aikana ovat RS485-standardin mukaisia.



Kuva 24. RS485, differentiaalinen signaali.

5.4 Tallennus SD-kortille

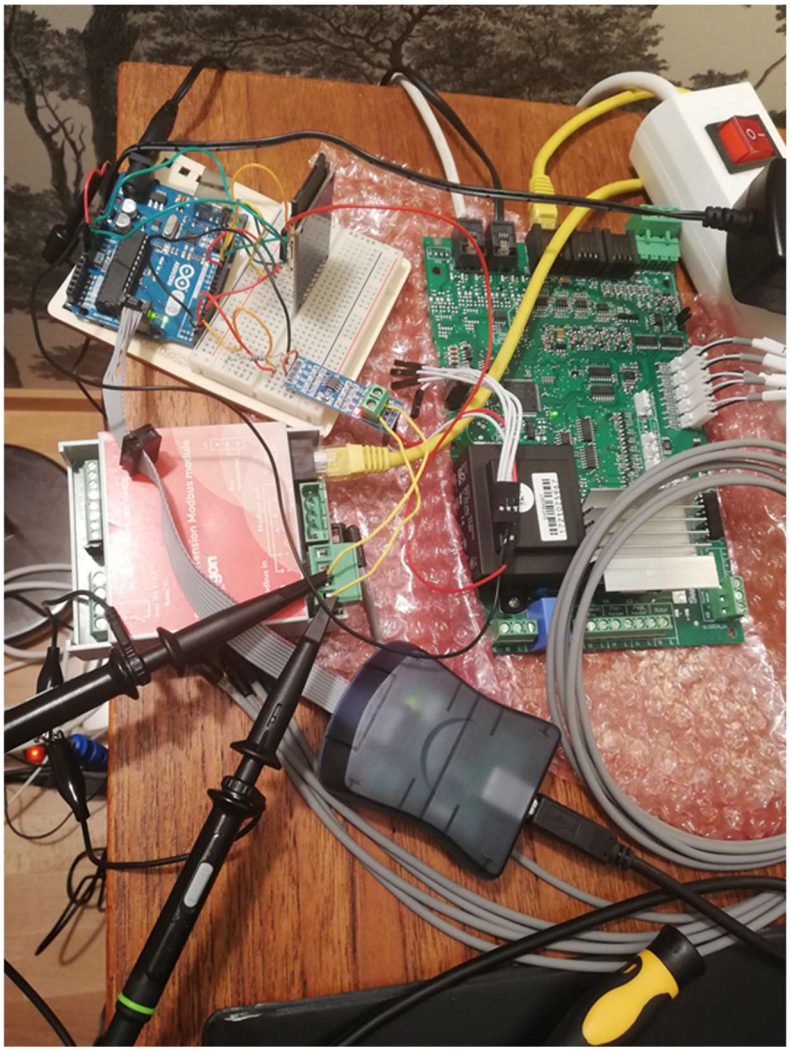
Tallennus SD-kortille tapahtui käyttämällä Arduinon valmiita kirjastoja SD.h ja SPI.h, jotka ovat erittäin vahvasti sidoksissa Arduino.h kirjastoon. Kirjastojen sisällä tapahtuu FAT32-muotoon tallennus SD-kortille. Onnistuneesti vastaanotettu viesti kirjoitetaan SD-kortille yhdistämällä Modbus-paketin tiedetyn rekisteriarvon suuret ja pienet 8 bittiä yhdeksi 16-bittiseksi arvoksi. Paketin sisältämät rekisterit erotellaan merkillä ”;”, jolloin sarakkeen vaihto on tunnistettavissa Microsoft Officen Excel-ohjelmistolla. Virheellisesti vastaanotettua vastausta ei kirjoiteta rekisteriin, mutta virhekoodi kirjoitetaan, jolloin muotoilut tiedostossa eivät häiriinny. Kun jokainen rekisteri on kirjoitettu SD-kortille, tehdään rivinvaihto ja odotetaan kunnes laskuri antaa luvan aloittaa prosessi alusta. SD-kortille tallennuksen funktio on esitettynä kuvassa 25.

```
void SDwrite(){
myFile = SD.open("e.csv", FILE_WRITE);
while(!myFile){
}
if (validate()==2){      //if valid, could be also val == 2
    for(i=3; i<=bytecnt+1; i+=2){
        temp = 0;
        temp |= msgRX[i] << 8 | msgRX[i+1];
        myFile.print(temp);
        myFile.print(";");
    }
}
else {
      for(i=3; i<=bytecnt+1; i+=2){   //not valid
          myFile.print(val); //print error
          myFile.print(";");
      }
}
if(regno == 0){    //all regs polled wait until next poll
   myFile.println("");
   while(pollrateTimer!=0);
}
myFile.close();
}
```

Kuva 25. SD-kortille kirjoitus.

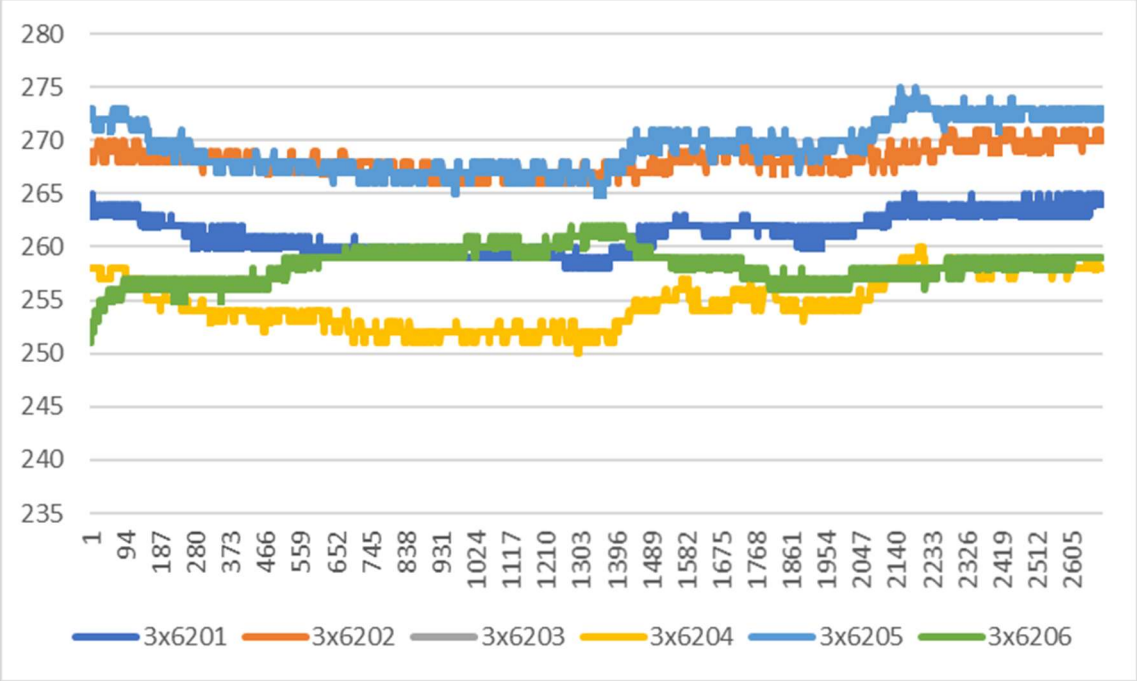
5.5 Datan analysointi

Dataloggerin toimintaa testattiin kuvan 26 mukaisella testijärjestelyllä. Swegon ilmanvaihtokoneen piirikorttiin oli kytkettynä viisi lämpötila-anturia. Ilmanvaihtokoneen piirikortille oli kytketty kytkentämoduuli ”Smart Extension Module”, jonka avulla voidaan kytkeytyä modbus protokollaa käyttävälle RS485-väylälle. Arduinoon kytkettiin erillinen SD-kortin lukija, sekä RS485-sovitin. Piirin ja mikrokontrolleriin ladattua ohjelman toimintaa tutkittiin Keysightin EDU-X 1002G-oskilloskoopilla, sekä SD-kortille tallennettavan tiedon avulla.



Kuva 26. Testijärjestely.

Modbussin avulla luettu mittausarvo kertoo tarkalleen sen, mitä ilmanvaihtokone näkee tai luulee näkevänsä ympäristöstään. Kuvan 27 lokitiedoston grafiikasta on nähtävissä, että joko lämpötila-antureissa on eroavaisuuksia, lämpötila-anturin lukemisessa on poikkeavuutta tai antureiden välillä on lämpötila-eroja. Kuvan 27 y-akselin lämpötila arvot ovat kerrottuna luvulla 10, jolloin esimerkiksi arvo 265 vastaa lämpötilaa 26,5 °C. Käytettävien Modbus-rekisterien arvot ovat aina kokonaislukuja. Tallennetusta lokista nähdään, että kommunikointi on onnistunut luotettavasti, eikä virheitä ole syntynyt. Virheiden merkiksi lokiin kirjoitettava arvo olisi ollut 0, 1 tai 3.



Kuva 27. SD-kortille tallennettu datan visualisointi.

Ilmanvaihtokoneesta on luettavissa paljon muitakin arvoja kuin lämpötiloja. Swegonin ilmanvaihtokoneista on saatavilla Modbusin avulla esimerkiksi tiedot koneen ohjauksista, tilatiedoista, mittauksista ja mahdollisista hälytyksistä. Mittauksia ja ohjauksia vertaamalla voidaan tutkia ja analysoida koneen ohjausta. Käytännön kohteessa, missä ilmanvaihtokone on asennettuna asuinhuoneistoon, voidaan kerätä tietoa asukkaan toiminnasta. Esimerkiksi suihkussa käynti näkyy asunnon kosteudentuottona ja on mitattavissa ilmanvaihtokoneen poistoilmalohkosta. Henkilömäärä asunnossa voidaan päätellä vastaavasti ilman hiilidioksidipitoisuudesta ja niin edelleen. Tilatietoja tutkimalla voidaan analysoida esimerkiksi sitä, kuinka usein asukas käyttää ilmanvaihtokoneen takkatoimintoa tai kuinka usein liesikuputoiminto on aktiivinen. Tietoja voidaan kerätä eri tarpeisiin.

6 JATKOKEHITYSIDEOITA

Dataloggeri vie lokitiedot nyt vain ja ainoastaan SD-kortille, mikä tekee datan keräämisestä suuressa mittakaavassa haastavaa. Loggerin parantamiseksi olisi lisättävä tehokas tapa tuoda data esimerkiksi tietokoneelle. Loggerin ja tietokoneen välisen väylän lisääminen mahdollistaisi esimerkiksi laitteen uudelleen konfiguroinnin. Datan lukeminen SD-kortilta erillistä väylää pitkin olisi myös selvä parannus laitteen käytettävyyteen. Ideaalinen tilanne olisi, jos tiedonkeruulaite olisi yhteydessä esimerkiksi pilvipalveluun ja ohjattavissa etänä. Laite tarvitsee ympärilleen rajapinnan käyttäjälle, jolloin rekisteriarvojen muutosta ei tarvitse tehdä koodiin ja datan siirto analysoitavaan muotoon olisi käytännöllisempää.

Elektroniikassa on paljon optimoitavaa. Arduino sisältää dataloggerille tarpeetonta elektroniikkaa, mikä on myös omalta osaltaan rajoite. Tarpeettoman elektroniikan poistamisella voidaan saavuttaa fyysisesti pienempi koko ja matalampi tehon kulutus. Pienillä muutoksilla loggeri olisi kytkettävissä ilmanvaihtokoneeseen ilman ulkoista virtalähdettä. Nyt Arduinon fyysinen koko on rajoite, eikä tiedonkeruulaitetta voi asentaa esimerkiksi ilmanvaihtokoneen sisälle. Myöskään Arduinoon helposti kytkettävä käyttöjännite ei ole saatavilla ilmanvaihtokoneesta. Arduinon suositeltava käyttöjännite on 7–12V, vaikka mikrokontrollerin vaatima käyttöjännite on tästä matalampi. [20]

Toteutus on kuitenkin toimiva ja ohjelmaa on helpohko jalostaa toimivammaksi kokonaisuudeksi. Ohjelmassa ei ole mitään aikakriittistä toimintoa, minkä aikana ei voida suorittaa muita toimintoja. Keskeytyksillä toteutetut ajastukset, lähetykset ja vastaanotot toimivat, vaikka koodi ohjelmoitaisiin tekemään jotain muuta, kuin varsinaista datalokia. Koodista löytyvät erilliset funktiot paketin luonnille, datan lähetykselle ja vastaanotolle, vastaanoton tarkistukselle sekä tallennukselle. Osioita koodista poistamalla, voidaan laitteesta modifioida versioita myös muihin tarkoituksiin kuin varsinaiseen tiedonkeruuseen. Esimerkiksi laitteesta voidaan modifioida versio, joka ohjaa ilmanvaihtokoneen tiloja tai ohjauksia.

7 YHTEENVETO

Swegonilla on tarve kerätä ilmanvaihtokoneen mittausdataa ja hyödyntää sitä tuotekehityksen tarpeisiin. Työn tarkoituksena oli suunnitella halpa Modbusiin kytkettävä tiedonkeruulaite, mikä voidaan kytkeä Swegonin ilmanvaihtokoneeseen tuottamaan datalokia itsenäisesti ilman erillistä tietokonetta. Laitteen suunnittelu vaatii tutustumista dataloggerin käyttämiin protokolliin, tiedonsiirtoväyliin ja tiedostojärjestelmiin.

Dataloggerin suunnittelu lähti tutkimalla, mitä Modbus tiedonsiirtoprotokolla sisältää. Swegonin asuntoilmanvaihdon koneet käyttävät funktiokoodeja 3 ja 4, joista molempien protokollaosuus on tiedon lukemisen kannalta sama. Datan esitystavaksi osoittautui sarjaliikenneväylän versio Modbus RTU. Modbus RTU sisältää CRC16-laskennan.

Swegonin ilmanvaihtokoneen Modbus-liityntä oli tehty RS485-väylään. Ennen RS485-liityntää oli signaali kuitenkin tuotettava ja tulkittava. Tähän valikoitui Atmelin Atmega328P-mikrokontrolleria hyödyntävä Arduino-levy. Mikrokontrollerissa oli sisäänrakennettu USART-piiri, jota käytettiin ohjaamaan ilmanvaihtokoneen ja tiedonkeruulaitteen välistä tiedonsiirtoväylää. Myös RS485-väylän suunnittelua käytiin tässä opinnäytetyössä pääpiirteittäin.

Kaikki Arduinon ja ilmanvaihtokoneen väliseen kommunikaatioon liittyvä koodi tuotettiin itse. SD-kortille tallentamiseen hyödynnettiin Arduinon omia kirjastoja, koska FAT-tiedostojärjestelmän luonti alusta alkaen olisi ollut kohtuuttoman suuri työ hyötyyn nähden. Rauta eli elektronisissa komponenteissa käytettiin valmiita ratkaisuja. Pääpaino työssä oli oikeastaan ohjelman tuottaminen.

Työssä luotiin toimiva, Swegonin ilmanvaihtokoneeseen kytkettävä dataloggeri. Koodi on helpohkosti muokattavissa ja esimerkiksi Modbus paketin luonti osaa yhdistää peräkkäiset rekisterit yhteen kyselyyn. Vastaanotettavalle viestille tehdään protokollaan perustuva tarkistus, sekä CRC-laskenta, eikä virheellisesti tulkittua dataa päästetä missään olosuhteessa SD-kortille. Toteutus on pyrkinyt hyödyntämään mahdollisimman tehokkaasti mikrokontrollerin sisäisiä piirejä, kuten laskureita, USART-piiriä ja erilaisia keskeytystoimintoja.

LÄHTEET

[1] Wikipedia, ”Modbus,” [WWW-sivu]. Saatavilla: https://fi.wikipedia.org/wiki/Modbus (luettu 20.11.2018)

[2] “Modbus Specification,” [PDF-tiedosto]. saatavilla: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (luettu 20.11.2018)

[3] “MODBUS over serial line specification and implementation guide V1.02,” [PDF-tiedosto]

Saatavilla: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (luettu 20.11.2018)

[4] Wikipedia, ”Cyclic redundancy check,” [WWW-sivu], saatavilla: https://en.wikipedia.org/wiki/Cyclic_redundancy_check (luettu 20.11.2018)

[5] ”CRC Generating and Checking”, [WWW-sivu] Saatavilla: http://ww1.microchip.com/downloads/en/appnotes/00730a.pdf (luettu 25.11.2018)

[6] ”The RS485 Design Guide”, [PDF-tiedosto], saatavilla: http://www.ti.com/lit/an/slla272c/slla272c.pdf (luettu 25.11.2018)

[7] “Guidelines for Proper Wiring of an RS-485 (TIA/EIA-485-A) Network” , [WWW-sivu], saatavilla: https://www.maximintegrated.com/en/app-notes/index.mvp/id/763 (luettu 25.11.2018)

[8] “Removing Ground Noise in Data Transmission Systems”, [PDF-tiedosto], Saatavilla: http://www.ti.com/lit/an/slla268/slla268.pdf (luettu 25.11.2018)

[9] “Interface Circuits for TIA/EIA-485 (RS-485) Application Report”, [PDF-tiedosto], saatavilla: http://www.ti.com/lit/an/slla036d/slla036d.pdf (luettu 25.11.2018)

[10] ”Wiring of RS485 Communications Networks”, [PDF-tiedosto], saatavilla: https://www.schneider-electric.com.au/library/SCHNEIDER_ELECTRIC/SE_LOCAL/APS/188266_2F4E/16798.pdf (luettu 25.11.2018)

[11] ”Passive failsafe for an idle bus”, [PDF-tiedosto], saatavilla: http://www.ti.com/lit/an/slyt324/slyt324.pdf (luettu 25.11.2018)

[12] ”8-bit microcontrollers ATmega328/P DATASHEET COMPLETE.” [PDF-tiedosto], Saatavilla: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf (luettu 15.11.2016)

[13] Wikipedia, ”Secure Digital”, [WWW-sivu], saatavilla: https://en.wikipedia.org/wiki/Secure_Digital (luettu 25.11.2018)

[14] ”How to Format a Large Hard Drive With FAT or FAT32”, [WWW-sivu], saatavilla: https://www.makeuseof.com/tag/format-large-hard-drive-fat-fat32/ (luettu 25.11.2018)

[15] ”Serial Peripheral Interface (SPI)”, [WWW-sivu], saatavilla: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all (luettu 25.11.2018)

[16] Wikipedia, ”File Allocation Table”, [WWW-sivu], saatavilla: https://en.wikipedia.org/wiki/File_Allocation_Table (luettu 25.11.2018)

[17] Wikipedia, ”Comma-separated values”, [WWW-sivu], saatavilla: https://en.wikipedia.org/wiki/Comma-separated_values (luettu 25.11.2018)

[18] Wikipedia, ”Interrupt”, [WWW-sivu], saatavilla: https://en.wikipedia.org/wiki/Interrupt (luettu 25.11.2018)

[19] “RS-485 CONNECTIONS FAQ CONNECTIONS FOR 2-WIRE or 4-WIRE”, [WWW-sivu], saatavilla: http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/RS-485-Connections-FAQ/faq_rs485_connections.pdf (luettu 28.11.2018)

[20] Arduino, [WWW-sivu], saatavilla: https://store.arduino.cc/arduino-uno-rev3 (luettu 28.11.2018).

Dataloggerin koodi

```
#include <Arduino.h> //modified, USART init part
#include <util/crc16.h>
#include <SPI.h>
#include <SD.h>

#define addr 1
#define fnct 4
#define pollrate 500 //x 10 milliseconds
#define timeout 30 // x 10 milliseconds

File myFile;
int temp, val;
unsigned int i, regcnt, regno, crc;
unsigned char FC4[8];
unsigned int regs[] = {6201, 6202, 6203, 6204, 6205, 6206};
unsigned char x, bytecnt;
volatile unsigned int timeoutTimer, pollrateTimer;
unsigned char msgRX[64];

ISR (USART_RX_vect){
   msgRX[x] = UDR0;
  if(x<64){
    x++;
  }
  else x=0;
}

ISR (USART_TX_vect){
            PORTD &= ~(1<<PORTD2);
}

void USART_Transmit (unsigned char data){
  while (!(UCSR0A & (1<<UDRE0)));
  UDR0 = data;
}

ISR (TIMER1_COMPA_vect){
            if(timeoutTimer!=0){
                        timeoutTimer--;
            }
            if(pollrateTimer!=0){
                        pollrateTimer--;
            }
}

void packet(){
            FC4[0] = addr;
            FC4[1] = fnct;
            FC4[2] = (uint8_t)(regs[regno]>>8); //hi byte
            FC4[3] = (uint8_t)(regs[regno]); // lo byte
            FC4[4] = 0x00; // reg count hi, always 0
            FC4[5] = 0x01;

            while(regs[regno+1] == regs[regno]+1){
                        FC4[5]++;
                        regno++;
```

```
            }
            bytecnt = 2*FC4[5];
            crc = 0xffff;
            for(i=0; i<=5; i++){
                crc = _crc16_update(crc, FC4[i]);
            }
            FC4[6] = (uint8_t)crc;
            FC4[7] = (uint8_t)(crc>>8);

            if(regno < (regcnt-1)){
                        regno++;
            }
            else regno = 0;
            PORTD |= (1<<PORTD2);
}

int validate(){
val = 0; //reset

if(msgRX[0] == FC4[0]
   && msgRX[1] == FC4[1]
   && msgRX[2] == bytecnt){
    val = 1; //right slave, function and expected bytecount
    crc = 0xffff;
    for(i=0; i<=bytecnt+2; i++){
        crc = _crc16_update(crc, msgRX[i]);
    }
    if(msgRX[bytecnt+3] == (uint8_t)(crc) && msgRX[bytecnt+4] ==
(uint8_t)(crc>>8)){
       val = 2; //crc ok
    }
}
    else{
         val = 3; //not valid
    }
return val;
}

void SDwrite(){
            myFile = SD.open("e.csv", FILE_WRITE);
            while(!myFile){
            }
            if (validate()==2){
                for(i=3; i<=bytecnt+1; i+=2){
                    temp = 0;
                    temp |= msgRX[i] << 8 | msgRX[i+1];
                    myFile.print(temp);
                    myFile.print(";");
                }
            }
            else {
                  for(i=3; i<=bytecnt+1; i+=2){
                      myFile.print(val);
                      myFile.print(";");
                  }
            }

            if(regno == 0){
               myFile.println("");
               while(pollrateTimer!=0);
```

```
            }
            myFile.close();
}
void setup() {

            cli();
            DDRD = 0b0001100; // PD2 for RS485 driver state, PD3 for SD card
chip select
            UCSR0B |= (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0) | (1<<TXCIE0);
//enable TX and RX
            UCSR0C = (3<<UCSZ00);//  //frame format 8data, 1 stop
            UBRR0 = 25; //baud rate

            TCCR1A = 0; //rset
            TCCR1B = 0; //reset
            TCNT1 = 0; //reset
            OCR1A = 19999; //count to 19999 approx 10ms
            //TCCR1A |= (1<<COM1A0); //toggle on compare match debug purpose.
Pin PB1
            TCCR1B |= (1<<WGM12) | (1<<CS11); //prescaler 8 and WGM12 refers
to CTC mode clear timer on compare match
            TIMSK1 |= (1<<OCIE0A); //interrupt enable on compare match

            sei();

            regcnt = sizeof(regs)/sizeof(uint16_t);
            regno = 0;

            SD.begin(4); //pin 4
            myFile = SD.open("log.csv", FILE_WRITE); //include to filehandler
            for(i=0; i<regcnt; i++){
                        myFile.print(regs[i]);
                        myFile.print(";");
            }
            myFile.println("");
            myFile.close();
}

void loop() {
     pollrateTimer = pollrate;
     packet();
     for(i=0;i<=7;i++){
         USART_Transmit(*(FC4+i));
     }
    x=0;
    timeoutTimer = timeout;
    memset(msgRX, NULL, sizeof (msgRX));
    while(validate()!=2){
          if (timeoutTimer == 0){
              break;
           }
          }
    SDwrite();
}
```