OULUN AMMATTIKORKEAKOULU

Jarkko Korkiakoski

SDI-12 SARJALIIKENNEPROTOKOLLA

SDI-12 SARJALIIKENNEPROTOKOLLA

Jarkko Korkiakoski
Opinnäytetyö
Kevät 2020
Sähkö-ja automaatiotekniikan
tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Sähkö-ja automaatiotekniikan tutkinto-ohjelma, automaatiotekniikka

---

Tekijä: Jarkko Korkiakoski
Opinnäytetyön nimi suomeksi: SDI-12-sarjaliikenneprotokolla
Opinnäytetyön nimi englanniksi: SDI-12 Serial Communication Protocol
Työn ohjaajat: Satu Vähänikkilä, Mika Viitala
Työn valmistumislukukausi ja -vuosi: Kevät 2020
Sivumäärä: 38

Työ tehtiin yhteistyössä THT Control Oy:n kanssa osana S1000-logiikan kehittämisen kartoittamista. Työn tarkoituksena oli tutustua SDI-12-sarjaliikenneprotokollaan ja sen mahdolliseen käyttöön automaatiotekniikassa. Työn teoriaosuudessa tutustuttiin SDI-12-sarjaliikenneprotokollan historiaan, kehitykseen johtaneisiin syihin ja protokollan toimintaan.

Työosuudessa luotiin Arduino-ohjelma, joka pyytää mittaukset mikroprosessorianturilta käyttäen SDI-12-standardin mukaisia käskyjä. Nämä mittaukset onnistuttiin tallettamaan muuttujiin, jotka luettiin S1000-logiikalla rekistereihin käyttäen Modbus-väylää. Logiikalta mittaukset voidaan tuoda visuaalisesti esitettävään muotoon Cromi-valvomojärjestelmään.

Seuraava tavoite kehityksessä on luoda ohjelmaratkaisu, joka tekisi anturin ja logiikan välisen keskustelun sulavammaksi. Tässä voidaan hyödyntää erilaisia SDI-12- ja Modbus-liikenteen ajoituksia. Lopullinen tavoite on tuoda mittaukset suoraan S1000-logiikalle ilman Arduino-kehityslautaa. Tämä voidaan toteuttaa esimerkiksi lisämoduulilla, joka logiikkaan liitettäessä käyttäytyisi SDI-12-masterin tavoin.

Asiasanat: SDI-12, Sarjaliikenne, Arduino, Automaatio

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Electrical and Automation Engineering, Option of Automation Engineering

---

Author: Jarkko Korkiakoski
Title: SDI-12 Serial Communication Protocol
Name of Supervisor: Satu Vähänikkilä, Mika Viitala
Year: Spring 2020
Pages: 38

This thesis was done in co-operation with THT Control as a part of further development planning for S1000 logic controller. The aim of this thesis was to get familiar with SDI-12 serial digital interface standard and its possible uses in automation technology. The first part of this thesis consists of theory, history and the main reasons for the development of this standard. The second part consists of a practical example of integrating a microprocessor-based sensor to S1000 logic controller and Cromi monitoring software.

In the practical part of the thesis the microprocessor-based sensor was successfully integrated into Cromi monitoring software using S1000 logic controller and Arduino UNO board. Arduino UNO board was responsible for taking the measurements from the sensor and storing them into Modbus holding registers for the S1000 to read as a Modbus master device. The measurements were then sent to Cromi monitoring software using Modbus TCP and presented visually.

The next step in the development is improving some aspects of the program that handles the measurements in Arduino IDE to reduce the number of timeout errors on the S1000 logic controller side. Improvements in the timing of the measurement readings by S1000 logic controller would make detecting connection problems easier. The overall goal of the development would be to get rid of Arduino board and develop a module for the S1000 logic controller that could act as an SDI-12 master.

Keywords: SDI-12, Serial Communication, Arduino, Automation

SISÄLLYS

SANASTO

Cromi	THT Control Oy:n kehittämä modulaarinen valvomo-ohjelmisto
Lokilaite	Laite, joka pystyy tallentamaan lokitietoja
Modbus	Modiconin vuonna 1979 kehittämä tietoliikenne protokolla
Moduuli	Lisäosa, jolla on oma toiminnallinen tehtävä
PLC	Ohjelmoitava logiikka (Programmable logic controller)
Puskuri	Väliaikainen muistipaikka
RTU	Remote terminal unit. Mikroprosessorilaite, joka toimii rajapintana fyysisten laitteiden ja valvomon välissä

1 JOHDANTO

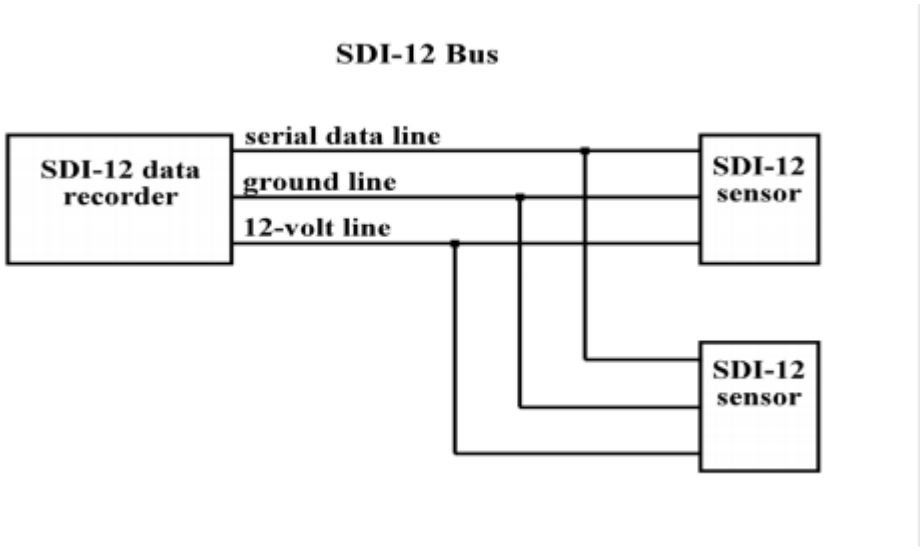
Työn tarkoituksena on tutustua SDI-12-sarjaliikenneprotokollaan. Työssä pyritään käymään läpi protokollan toiminta ja sen tämänhetkiset käyttökohteet ja -tarkoitukset. Teoriaosuudessa tarkastellaan standardin kehittämiseen johtaneita syitä ja historiaa.

Työosuudessa tarkoituksena on tuoda mikroprosessoriantureista koostuvan sääaseman mittauksia Cromi-valvomoon. Tarkoituksena on pohtia SDI-12-protokollaa käyttävien mikroprosessoriantureiden käyttömahdollisuuksia osana automaatiojärjestelmää. Työ on tehty yhteistyössä THT Control Oy:n kanssa ja se on osa S1000-logiikkaohjaimen kehityksen kartoittamista.

2 SDI-12

SDI-12 (Serial/Digital Interface at 1200 baud) on asynkroninen sarjaliikenneprotokolla, joka kehitettiin vuonna 1998 USGS/HIF:n (U.S. Survey's Geological Survey's Hydrologic Instrumentation Facility) ja yksityisistä yrityksistä koostuvan joukon yhteistyöllä. Tarkoituksena oli kehittää matalavirtainen rajapinta älykkäiden mikroprosessoriantureiden ja lokilaitteen välille. (1)

Koska protokolla tulisi käytettäväksi syrjäisillä seuduilla, haluttiin sen olevan mahdollisimman yksinkertainen. Sen sijaan, että jokaisesta lokilaitteesta ja anturista tulisi tutkia niiden käyttämät kytkennät, ohjelmoinnit ja laitteiden yhteensopivuus, voitaisiin todeta laitteiden käyttävän SDI-12-protokollaa, ja täten olevan yhteensopivat. SDI-12-protokollaa käyttävät sensorit kytketään kolmella johtimella: sarjaliikenne, maadoitus ja jännite. SDI-12 muodostaa monipistepiirin (Kuva 1): yhtä lokilaitetta kohti voi olla useampi SDI-12-sensori mittaamassa eri suureita. (2, s. 2)



KUVA 1. SDI-12 -väylä (2, s. 3)

2.1 Kommunikointi

Protokolla noudattaa master-slave-verkon konfiguraatiota, missä lokilaite on master, joka lähettää käskyjä ja kysyy dataa antureilta, joilla jokaisella on oma yksilöllinen tunniste. Lokilaitteen käskyt ja antureiden vastaukset kulkevat yhtä sarjaliikennejohdinta pitkin käyttäen ASCII -merkkejä.

Anturit pysyvät matalavirtaisessa valmiustilassa odottamassa lokilaitteen herätyskäskyä. Lokilaitteen antaessa herätyskäskyn kaikki sensorit heräävät ja odottavat tarkennusta, mitä sensoria lokilaite tavoittelee. Tämän jälkeen lokilaite antaa käskyn tietylle sensorille, jolloin kaikki muut sensorit palaavat matalavirtaiseen valmiustilaan. Sensori, jolta tietoa on vaadittu, lähettää lokilaitteelle vastauksena ajan, jonka jälkeen mittaustieto on valmis lähetettäväksi. Mikäli mittaus on heti valmiina, pyytää lokilaite mittaustiedon lähetettäväksi välittömästi. Jos mittaus ei ole valmiina, lokilaite odottaa, kunnes sensori ilmoittaa mittauksen olevan valmis, jonka jälkeen lokilaite pyytää mittaustiedon lähetettäväksi.

2.1.1 SDI-12 käskyt ja vastaukset

Jotta SDI-12-protokollan periaate antureiden ja lokilaitteiden yhteensopivuudesta pysyisi, ovat myös laitteiden väliset käskyt ja vastaukset standardoituja. Käskyt alkavat aina sensorin tunnisteella ja loppuvat ”!”-merkkiin, jota käytetään merkkaamaan käskyrivin päättymistä. Standardinmukaiset vastaukset sisältävät pyydetyt arvot ja vastausrivi päätetään <CR><LF> (carriage return, line feed), joka tarkoittaa osoittimen siirtämistä rivin alkuun ja rivinvaihtoa. Protokollan standardikäskyjen tulee toimia kaikissa SDI-12-protokollaa käyttävissä lokilaitteissa ja antureissa eikä niiden tule tehdä muutoksia sensorin kalibraatioasetuksiin. Standardikäskyjen lisäksi on olemassa käskyjä, joilla voidaan muuttaa sensorin konfiguraatioasetuksia, esimerkiksi käskeä sensori kalibroimaan itsensä. Standardin ulkopuoliset käskyt ovat yleensä sensorin valmistajan tekemiä ja tarkoitettu vain tietyn sensorin käytettäväksi. Näidenkin käskyjen tulee noudattaa standardin mukaista käsky- ja vastausmuotoa. (2. s. 7)

Voimme käyttää käsky- ja vastausrakenteen esimerkkinä ”I!”-käskyä, jota käytetään myös myöhemmin työosuudessa.

”I!”-käsky käskee sensoria lähettämään tietoja itsestään, kuten sensorin yhteensopivuustason, mallinumeron ja firmware versionumeron.

Käsky → Vastaus

aI! → allccccccccmmmmmmvvvxxx . . . xxx

Vastaus muodostuu seuraavalla tavalla:

a sensorin tunniste

ll SDI-12 versionumero

cccccccc kahdeksan merkkiä, joista selviää myyjän tiedot, usein yrityksen nimi tai lyhenne

mmmmmm kuusi merkkiä, joista selviää sensorin mallinumero

vvv kolme merkkiä kertomaan sensorin versionumeron

xxx … xxx vapaaehtoinen kenttä, maksimissaan 13 merkkiä pitkä, josta selviää sensorin sarjanumero tai muu tarkentava tieto sensorista, joka ei ole olennainen lokilaitteen toiminnan kannalta. (2. s. 10)

2.1.2 Baudinopeus ja signaalimuoto

Baudi on yksikkö, joka ilmoittaa tiedonsiirtotapahtumien määrän sekunnissa. Baudinopeutta käytetään usein sarjaliikenteessä kuvastamaan tiedonsiirtonopeutta. Nykyään yleisemmin käytetty yksikkö kuvastamaan tiedonsiirtonopeutta on bps (bits per second), joka on tarkempi ja havainnollistavampi tapa kuvastaa tiedon siirtymisen nopeus. Yksi tiedonsiirtotapahtuma saattaa siirtää kerralla useita bittejä, esimerkiksi modeemeilla, joilla tiedonsiirtonopeus on 9600 bps, on signalointinopeus vain 2400 baudia. (3)

Koska sarjaliikenteessä tieto lähetetään bitti kerrallaan, täytyy vastaanottajan tietää mistä biteistä viesti koostuu, esimerkiksi mistä viesti alkaa ja mihin viesti loppuu. Tämän takia lähettäjän ja vastaanottajan on sovittava tiedon siirtämiseen käytettävä signaalimuoto. Signaalit yleensä alkavat startbitillä, jota seuraavat databitit. Viimeistä databittiä voidaan joissain tapauksissa käyttää pariteettibittinä. Pariteettibitti asetetaan joko parilliseksi tai parittomaksi ja vastaanottaja käyttää pariteettibittiä tarkistaakseen, tuliko viesti virheettömänä perille. Pariteetilla tarkoitetaan signaalin 1-bittien yhteenlasketun summan parillisuutta. Mikäli sovittu pariteetti on parillinen (even parity), mutta viesti saapuu perille parittomana (odd parity) tiedetään, että jokin bitti on korruptoitunut. Esimerkiksi viesti ”1001” lähetetään ja pariteetiksi on sovittu parillinen pariteetti, lisätään loppuun pariteettibitti 0, jotta viestin 1-bittien parillinen määrä pysyy (1+0+0+1+0 = 2). Viestin loppua merkitsee stop-bitti. (4)

SDI-12 baudinopeus on nimensä mukaisesti 1200 baudia. Signaalin muoto on

- -1 start-bitti
- -7 databittiä, joista vähiten tärkeä bitti lähetetään ensin
- -1 pariteettibitti, parillinen pariteetti
- -1 stop-bitti.

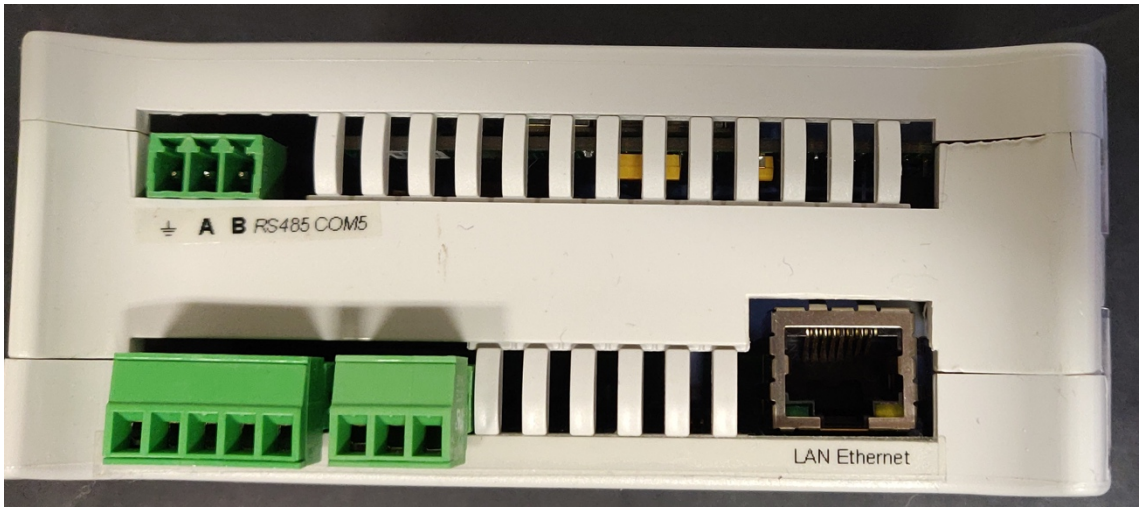
2.2 Käyttökohteet

SDI-12 sopii hyvin syrjäisiin kohteisiin, joista tarvitaan paljon mittaustietoa kustannustehokkaasti. Yleisimmät käyttökohteet SDI-12-protokollaa noudattaville antureille ovat vesihuollon, geotekniikan ja agrikulttuurin tietoverkostojen rakentamisessa. Esimerkiksi USGS käyttää yli 4000:ää SDI-12-sensoria dataverkostoissaan. (5)

3 S1000

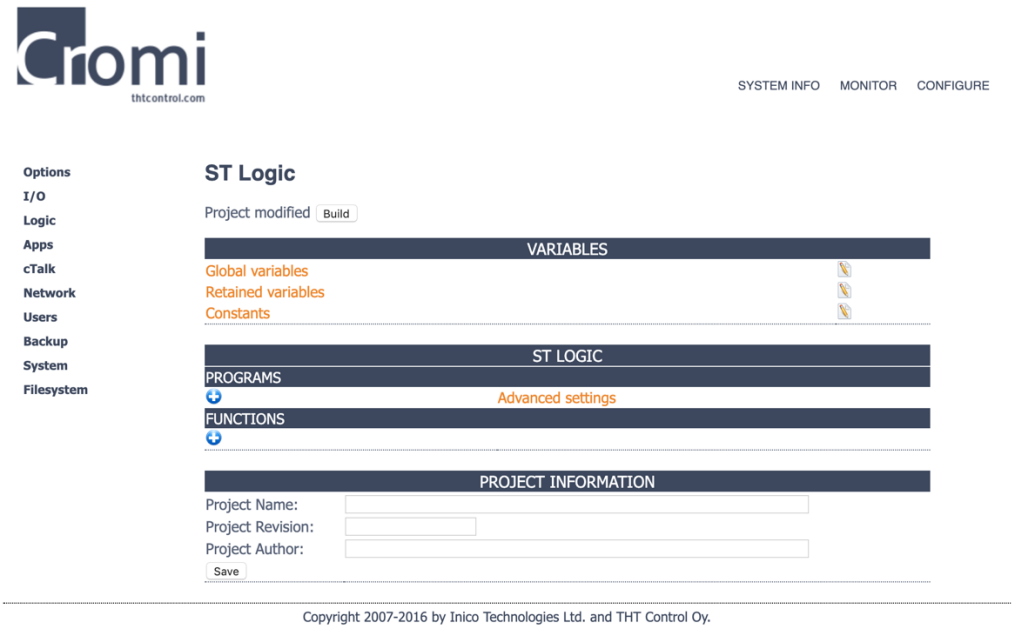
S1000 on pienikokoinen ja älykäs RTU (Smart RTU), joka yhdistää RTU:n vahvuudet kommunikoinnin ja monitoroinnin alalta sekä PLC:n prosessointikyvyn. Kontrolleriin voidaan lisätä useita erilaisia lisämoduuleja, jotka laajentavat kontrollerin käyttömahdollisuuksia. Lisämoduulit voivat tuoda kontrollerille esimerkiksi lisää analogisia ja digitaalisia I/O -paikkoja, 3G/4G -yhteyden tai RS485 sarjaliikenneportin (Kuvat 2 ja 3), jota tulemme tässä työssä käyttämään.

KUVA 2. S1000 kontrolleri + RS485/WLAN -lisämoduuli



KUVA 3. S1000 -kontrolleri + RS485/WLAN -lisämoduuli

S1000-kontrollerin ominaisuuksiin kuuluu selainpohjainen konfigurointi sekä logiikkaohjelmien tekoalusta. S1000 käyttää logiikkaohjelmien luomiseen IEC-61131-3 standardiin kuuluvaa Structured Text -ohjelmointikieltä. Kontrollerin selainpohjainen käyttöliittymä sisältää myös kääntäjän, joten kontrollerin ohjelmoiminen ei vaadi lisäohjelmien asentamista. Painamalla sivuvalikosta löytyvää ”Logic” -linkkiä saadaan auki logiikkaohjelmasivu (Kuva 4). Sivulla on listattuna muuttujat, vakiot, ohjelmat ja funktiot. Listojen yläpuolella on ”Build” -nappi, jolla ohjelman saa käännettyä.



KUVA 4. S1000 Logic -valikko

”FUNCTIONS”-listauksen alapuolella olevasta lisäysmerkistä painamalla saadaan auki funktion kirjoitusnäkymä (Kuva 5).

KUVA 5. Funktion kirjoitusnäkymä

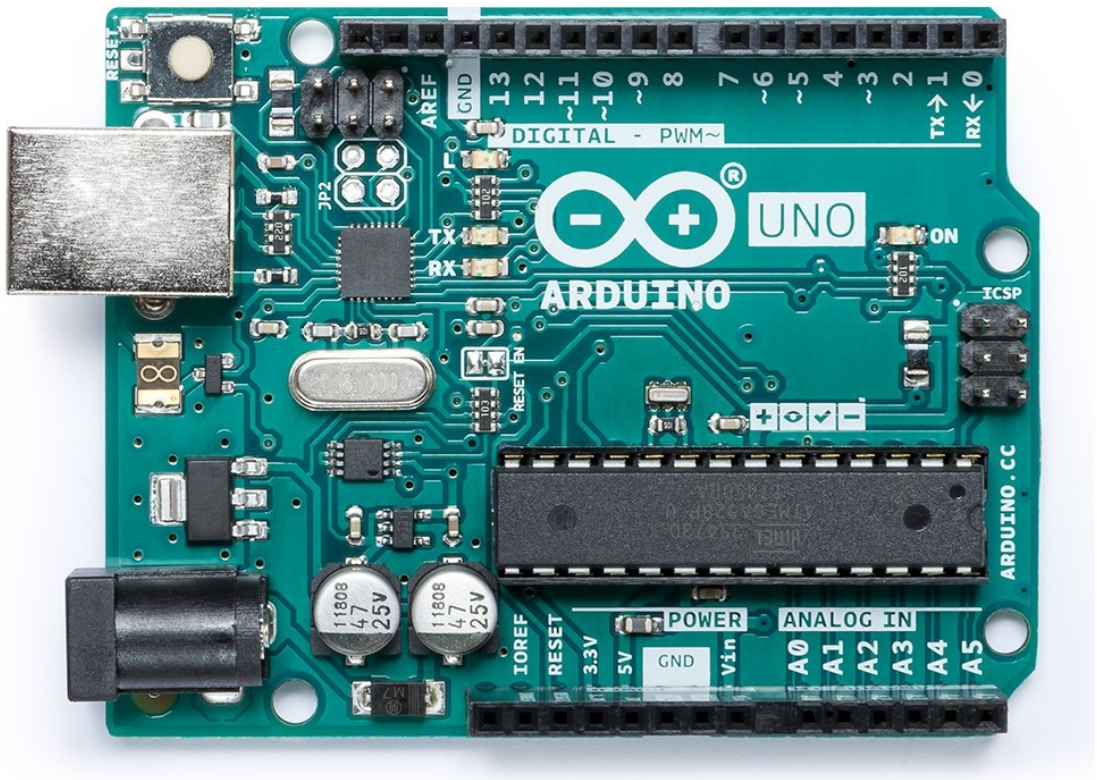
”PROGRAMS”-listauksen alapuolella olevasta lisäysmerkistä painamalla saadaan auki ohjelmien kirjoitusnäkymä (Kuva 6). Kun ohjelmat ja logiikat on kirjoitettu ja tallennettu, ne löytyvät listattuna ”Logic”-sivulta.

KUVA 6. S1000 logiikkaohjelman kirjoitusnäkymä

4 ARDUINO

Arduino on avoimen lähdekoodin elektroninen kehitysalusta, joka perustuu helposti käytettävään laitteistoon ja ohjelmistoon. Kehitysalustat sisältävät mikrokontrollerin, jolle voi tehdä erilaisia ohjelmia Arduino IDE -ohjelmiston kautta. Arduino tukee C- ja C++ -ohjelmointikieliä.

Työssä käytettiin Arduino Uno -kehitysalustaa (Kuva 7). Arduino Uno on ensimmäinen Arduino.cc:n kehittämä kehitysalusta, joka sisältää USB-ohjelmointiportin (6). Kehitysalustassa on 14 digitaalista I/O-pinniä, joista kuutta voidaan käyttää pulssinleveysmodulaatiossa ja kuusi analogista I/O-pinniä. Jokaista digitaalista ja analogista pinniä voidaan käyttää niin tulona kuin lähtönäkin, riippuen miten pinnien toiminta on ohjelmassa määritetty.



KUVA 7. *Arduino UNO -kehitysalusta (6)*

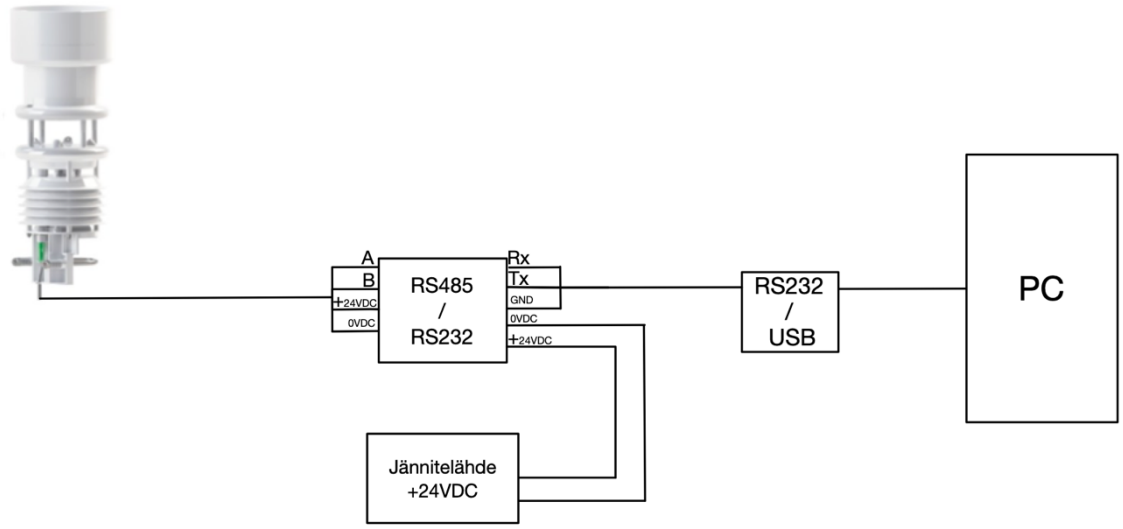
5 SÄÄASEMA

Sääasemalla tarkoitetaan laitetta, jolla pystytään mittaamaan useita eri sääsuureita. Työssä käytetiin Lufft:n WS -sarjan 601-UMB älykästä sääasemaa (Kuva 8.), jonka mittauksiin kuuluu muun muassa lämpötila, suhteellinen kosteus, ilmanpaine ja tuulen nopeus. Sääasema tukee usean eri protokollan lisäksi myös SDI-12 protokollaa, minkä takia kyseistä sääasemaa haluttiin käyttää työssä. (7)

KUVA 8. WS601-UMB Sääasema (7)

6 TYÖN SUORITUS

Työn suoritus aloitettiin WS601-UMB sääaseman konfiguroinnilla. Konfiguroinnissa haluttiin muuttaa sääaseman kommunikaatiostandardia RS485-standardista SDI12-standardin mukaiseksi. Sääasema kytkettiin tietokoneeseen RS485:n mukaisella kytkennällä (Kuva 9) ja konfiguraatiomuutoksiin käytettiin sääaseman valmistajan omaa ohjelmistoa nimeltä UMB Config Tool.



KUVA 9. Anturin konfigurointikytkentä

6.1 Arduino-ohjelmakoodi

Kun sääasema oli saatu SDI-12-tilaan, oli seuraavana tavoitteena saada luettua sääaseman mittaukset. Mittausten lukemiseen käytettiin Arduino Uno -kehitysalustaa ja Arduino IDE -ohjelmistoa. Arduino-kehitysalustaan ladattiin SDI-12 kirjasto ja ohjelma, joka saa Arduinon käyttäytymään kuin lokilaite. Tämän jälkeen sääasema kytkettiin ohjelman pinnimääritysten mukaisesti kehitysalustaan.

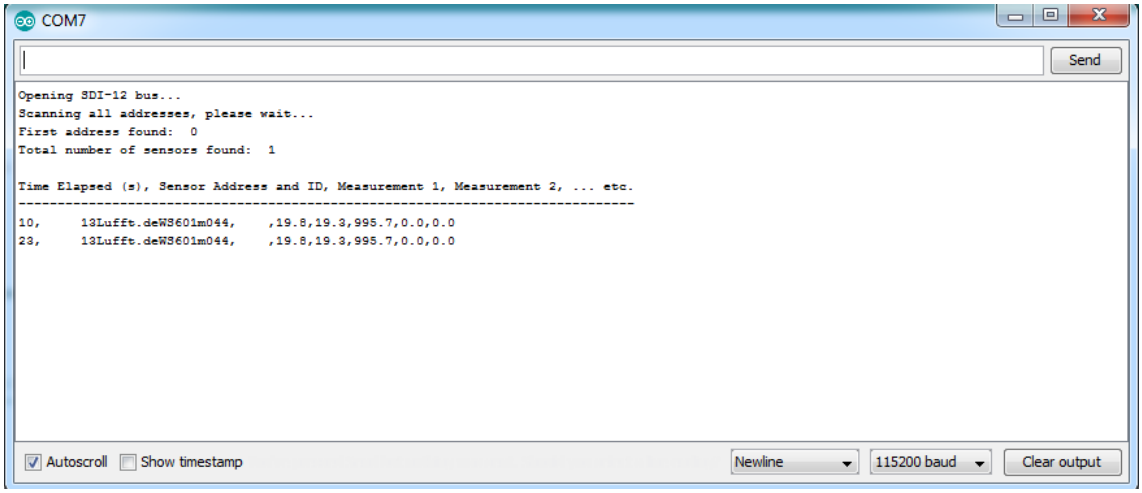
Arduinon ohjelmakoodi lähettää kolmea eri käskyä anturille ja tallentaa anturin vastauksen puskuriin, joka sitten tulostetaan sarjamonitoriin käyttäjän nähtäväksi. Puskuri tyhjennetään jokaisen tulostuksen jälkeen, jotta puskurissa on tilaa uudelle vastaukselle.

Kolme Arduinon lähettämää käskyä ovat ”I!”, ”M!” ja ”D0!” (Kuva 10). Nämä käskyt saavat sensorin lähettämään tiedot itsestään (I!), aloittamaan mittauksen (M!) ja lähettämään valmiit mittaustiedot (D0!).

```
// gets identification information from a sensor, and prints it to the serial port
// expects a character between '0'-'9', 'a'-'z', or 'A'-'Z'.
void printInfo(char i){
  String command = "";
  command += (char) i;
  command += "I!";
  mySDI12.sendCommand(command);
  // Serial.print(">>>");
  // Serial.println(command);
  delay(30);

  printBufferToScreen();
}

void takeMeasurement(char i){
  String command = "";
  command += i;
  command += "M!"; // SDI-12 measurement command format  [address]['M'][!]
  mySDI12.sendCommand(command);
  delay(30);

  // wait for acknowlegement with format [address][ttt (3 char, seconds)][number of measurments available, 0-9]
  String sdiResponse = "";
  delay(30);
  while (mySDI12.available())  // build response string
  {
    char c = mySDI12.read();
    if ((c != '\n') && (c != '\r'))
    {
      sdiResponse += c;
      delay(5);
    }
  }
  mySDI12.clearBuffer();
```

KUVA 10. I!-ja M!-käskyt

Kun Arduino-ohjelma on lähettänyt käskyt anturille, sarjamonitoriin tulostuvat kaikki pyydetyt tiedot ja mittaukset (Kuva 11).



KUVA 1. Arduino sarjamonitori

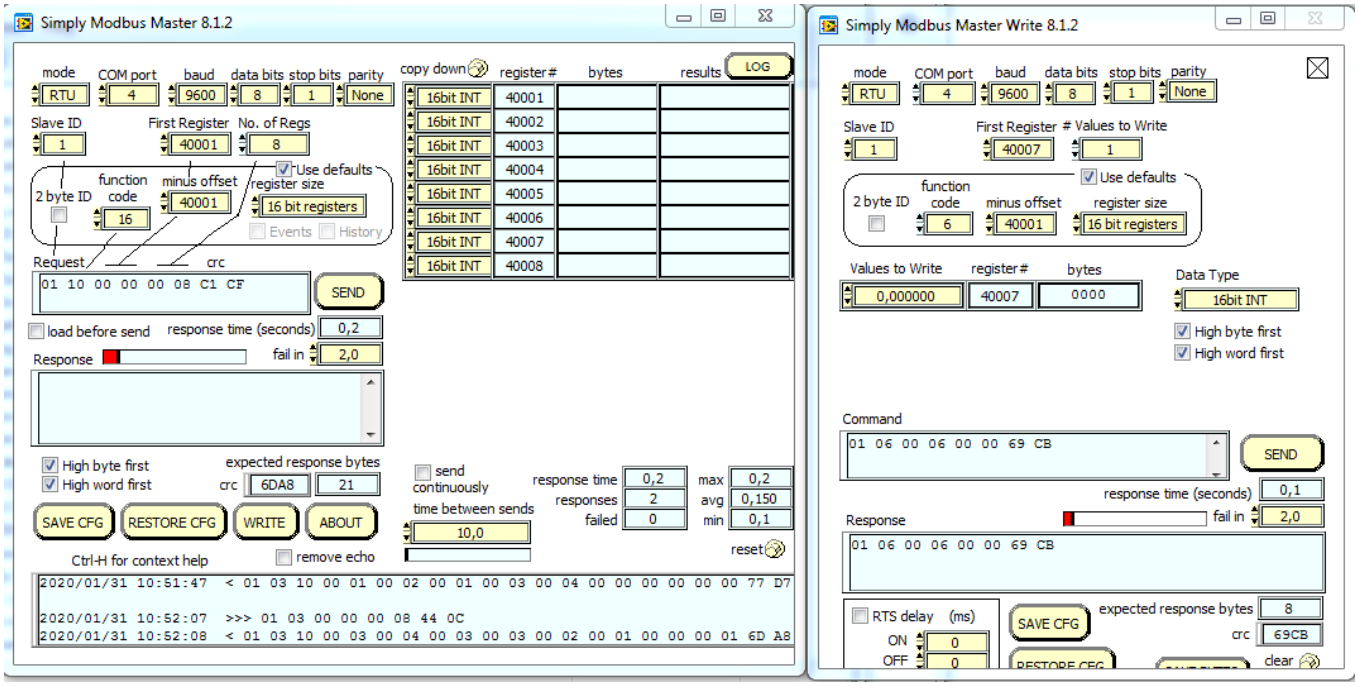
Mittaukset (measurement 1, measurement 2, ...) vasemmalta oikealle ovat: lämpötila, suhteellinen kosteus, suhteellinen ilmanpaine, tuulen nopeus ja maksimi tuulen nopeus.

6.2 Modbus-väylän testaaminen

Kun anturin mittaukset oli saatu luettua Arduinon kautta, aloitettiin työn suorituksen seuraava vaihe. Koska työssä haluttiin saada mittaukset S1000-kontrollerilta luettavaan muotoon, päätettiin mittaukset tallentaa Modbus-rekistereihin. Tätä varten Arduinolle täytyi ladata uusi ohjelmakoodi, joka mahdollistaa kehitysalustan ja PC:n välille master-slave-yhteyden, jossa Arduino toimii slave-laitteena.

Kehitysalustaan täytyi myös lisätä RS485-lisäkortti (Kuva 12), joka yhteyden mahdollistamisen lisäksi suojaa kehitysalustaa galvaanisella erotuksella. Arduino kehitysalustaan yhdistettiin LED-valo ja nappi, joiden tilat kirjoitettiin modbus-rekistereihin ohjelmakoodissa. Modbus-rekistereiden lukuun käytettiin Simply Modbus Master -ohjelmaa (Kuva 13), joka on yksinkertainen testiohjelma, jonka avulla PC voi käyttäytyä modbus RTU masterin tapaisesti. Testiohjelman ilmaisessa demoversiossa kyselyt ja kirjoitukset on rajattu kuuteen yhtä käyttökertaa kohti, mutta tämä oli enemmän kuin tarpeeksi yhteyden toimivuuden varmistamiseksi.

KUVA 12. Arduino UNO ja RS485 -lisäkortti (8)



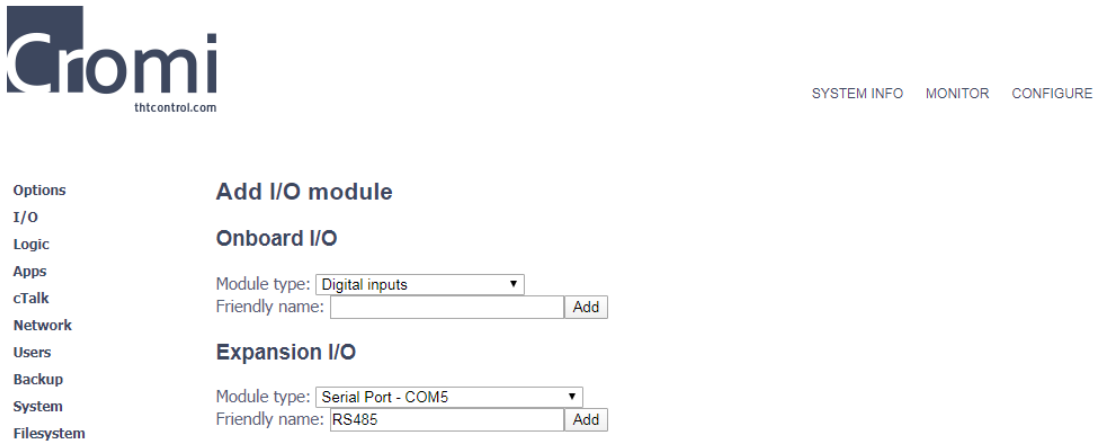
KUVA 13. Simply Modbus Master testausohjelma

Kun modbus-yhteys oli testattu, seuraavana tavoitteena saada nämä samat rekisterit luettua S1000-kontrollerilla. Kehitysalusta kytkettiin suoraan S1000-logiikan RS485-lisäkorttiin kiinni ja logiikalle konfiguroitiin Modbus-yhteys.

Sivuvalikosta valitaan ”I/O”, josta avautuu IO-konfiguraationäkymä ja listaus konfiguroiduista IO-moduuleista. Tässä vaiheessa lista oli vielä tyhjillään (Kuva 14).

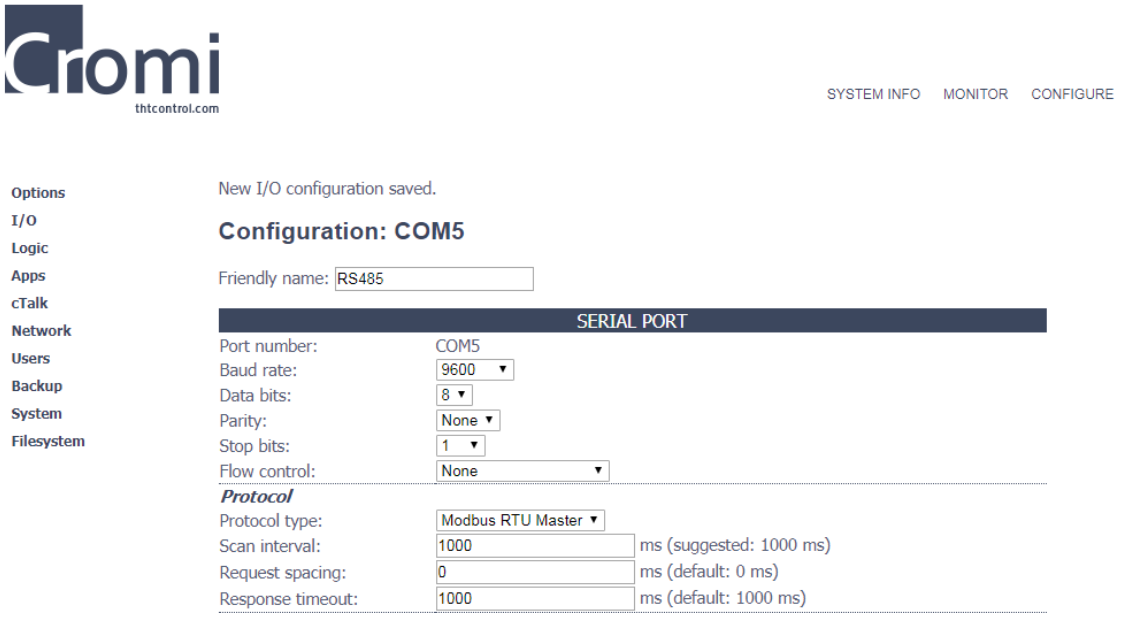
KUVA 14. S1000 IO -konfiguraationäkymä

”Add new I/O module”-linkkiä painamalla avautuu valikko, josta voidaan valita IO -moduuli, joka konfiguroidaan. Tässä tapauksessa konfiguroimme RS485-lisä-moduulin eli expansion I/O:n (Kuva 15). Konfiguroitavalle IO-moduulille voidaan antaa myös nimi, joka selventää käyttäjälle sen käyttötarkoitusta ohjelmassa.



KUVA 15. S1000-moduulin lisääminen

IO-moduuli lisätään painamalla ”Add” ja avaamalla moduuli moduulilistasta pääs-tään tarkastelemaan sen asetuksia. Tässä tapauksessa päästiin määrittelemään RS485-sarjaportin signaalimuodot ja protokollan tyyppi (Kuva 16). Työssä halut-tiin S1000-laitteen toimivan RTU Master -laitteena.



KUVA 16. S1000 sarjaportin konfigurointi

Tässä kohtaa työtä oli testattu tiedonsiirtyminen väliltä anturi – kehitysalusta ja kehitysalusta – S1000. Seuraavaksi haluttiin saada nämä kaksi väliä toimimaan yhtäaikaisesti ja lukea anturin mittaustiedot S1000-logiikalla.

6.3 Arduino-ohjelmakoodin työstäminen

Ohjelmakoodissa anturilta saatu mittaustieto on talletettu sarjapuskuriin String-muuttujana muotoon: ”XX.X,XX.X,XX.X, ..”, joten ensimmäinen asia, joka haluttiin tehdä, oli saada tämä vastaus parsittua osiin. Koska viestin sisältämät mittaukset on erotettu toisistaan pilkuilla, voimme käyttää parsimiseen funktiota, joka lukee String-muuttujan sisältöä merkki kerrallaan ja lopettaa, kun luettava merkki on erottimeksi nimetty merkki. Erottimen kohdalla luetut merkit tallennetaan uuteen String-muuttujaan (Kuva 17).

```
String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
```

KUVA 17. Mittaustietojen parsimisfunktio

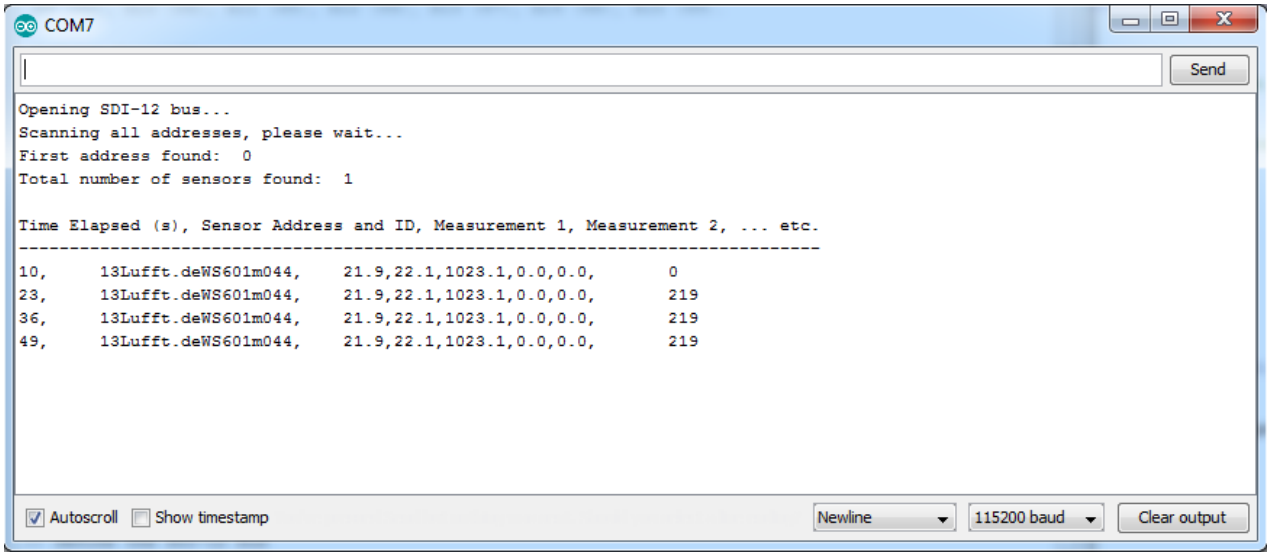
Koska Arduino-ohjelmakoodin puskuri tyhjennettiin jokaisen vastauksen ja käskyn välissä, haluttiin mittaustieto ottaa talteen ennen kuin se katoaa. Tähän käytettiin väliaikaista String -muuttujaa nimeltä vastausBuffer, johon talletetaan tieto, jonka Arduino on juuri saanut anturilta kysyttyään mittaustietoja. Lopullinen String -muuttuja, mihin tieto talletettiin parsimista varten, nimettiin vastausBuffer2 ja sen sisällön viimeiseksi merkiksi lisättiin pilkku (Kuva 18), jotta parsimisfunktio ei yritä lukea tietoa yli rajojen.

```
  printBufferToScreen();
  vastausBuffer2 = vastausBuffer;
  vastausBuffer2 += ",";
  mySDI12.clearBuffer();
String  vastausBuffer = "";
```

KUVA 18. Puskurin tietojen tallettaminen muuttujaan

Tämän jälkeen String vastausBuffer2 pystyttiin syöttämään getValue()-funktioon. Ulostulona funktiosta saatiin testiksi ensimmäisen mittauksen eli lämpötilamittauksen. Koska mittaus sisälsi yhden desimaalipaikan, kerrottiin mittaus kymmenellä, jotta siitä saataisiin helposti käsiteltävä kokonaisluku. Desimaali saadaan mittaukseen takaisin skaalaamalla se takaisin kymmenen kertaa pienemmäksi valvomon päädyssä.

Parsimisfunktion ulostulo testattiin lisäämällä parsitun mittauksen tulostaminen sarjamonitoriin (Kuva 19).



KUVA 19. Sarjamonitori ja parsittu lämpötilamittaus

Suureksi ongelmaksi ohjelmien yhdistämisessä paljastui se, että ohjelmissa oli päällekkäisyyksiä. Sarjaliikenteen tarkka ajoittaminen vaatii keskeytyksiä samoille pinneille kummassakin ohjelmassa, eli ne eivät voi toimia yhtä aikaa häiritsemättä toinen toistaan.

Mikäli SDI-12- ja Modbus-liikenne saataisiin käyttämään eri sarjaporttia, voisi kummallakin liikenteellä olla omat keskeytyksensä ilman ristiriitoja. Valitettavasti kehitysalustassa, jota työn tekemisessä käytettiin, ei ollut kuin yksi sarjaportti. Fyysisen sarjaportin sijaan on mahdollista käyttää ohjelmallista sarjaporttia, SoftSerial, jonka avulla kehitysalustan vapaita pinnejä voisi konfiguroida toimimaan TX- ja RX -pinneinä. SoftSerial kuuluu Arduinon vakiokirjastoihin, mutta tässä tapauksessa sen käyttäminen olisi mahdotonta, sillä kumpikaan ohjelmakirjasto, joita työhön käytettiin ei tukenut ohjelmallisen sarjaportin käyttöä.

Jotta sama ohjelma voisi liikennöidä sekä anturille, että logiikalle, jouduttiin ohjelmaan lisäämään tuhoajafunktio ”mySDI12.~SD12()” (Kuva 20). Tuhoajafunktion tehtävänä on pysäyttää SDI-12-liikennöinti siksi aikaa, kun mittaukset luetaan kehitysalustalta logiikalla.

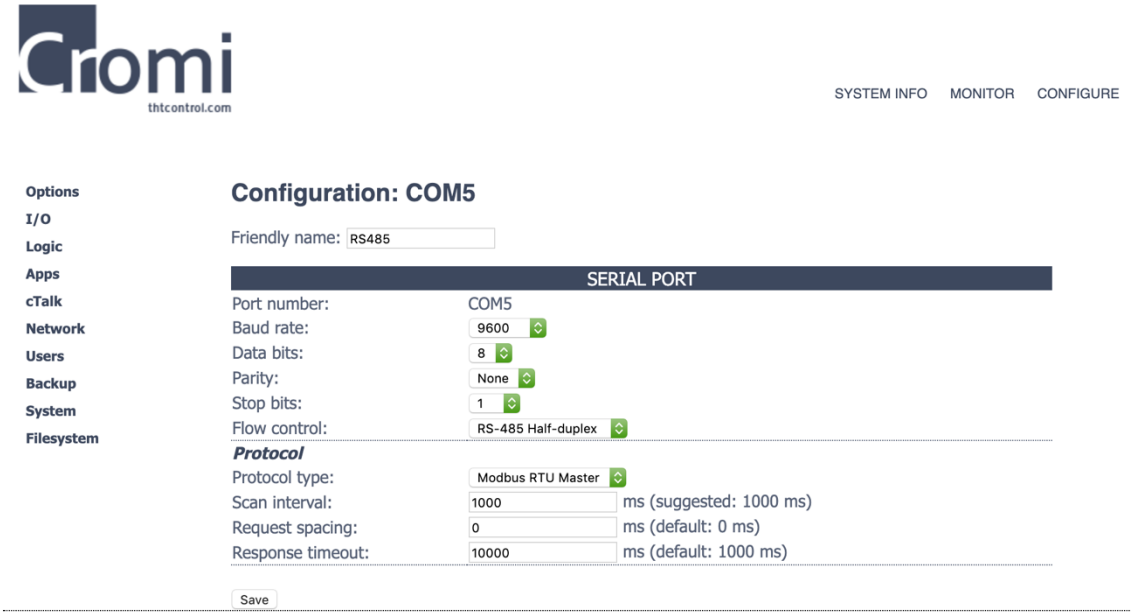
```
  mySDI12.~SDI12();
  delay(500);
  modbus_configure(&Serial, 9600, SERIAL_8N2, 1, 6, 5, holdingRegs);

modbus_update_comms(9600, SERIAL_8N2, 1);

  modbus_update();
     holdingRegs[0] = Lampotila;
     holdingRegs[1] = Kosteus;
     holdingRegs[2] = Ilmanpaine;
     holdingRegs[3] = Tuulennopeus;
     holdingRegs[4] = TuulennopeusMax;

  delay(10000); // wait ten seconds between measurement attempts.
```

KUVA 2. Modbus-yhteyden avaaminen ja rekistereiden päivittäminen

Kun Modbus-rekisterit on päivitetty ja kymmenen sekunnin viive on kulunut, ohjelma siirtyy takaisin alkuun, jossa SDI12-liikenne laitetaan päälle ja uusien mittausten pyytäminen alkaa.

6.4 Logiikan konfigurointi

Kun mittaukset oli saatu parsittua ja talletettua Modbus -rekistereihin, oli jäljellä enää logiikan konfigurointi (Kuva 21). Sarjaliikenteen bittimuoto on 8N1 ja tiedonsiirtonopeus 9600 baudia. Logiikka toimii master-laitteena, joka skannaa rekistereitä sekunnin välein. Timeout-ajaksi laitettiin 10 sekuntia yhden sekunnin sijaan, sillä SDI-12-liikenteen ollessa päällä Modbus-liikenne ei toimi, vaan aiheuttaa timeout virheen.

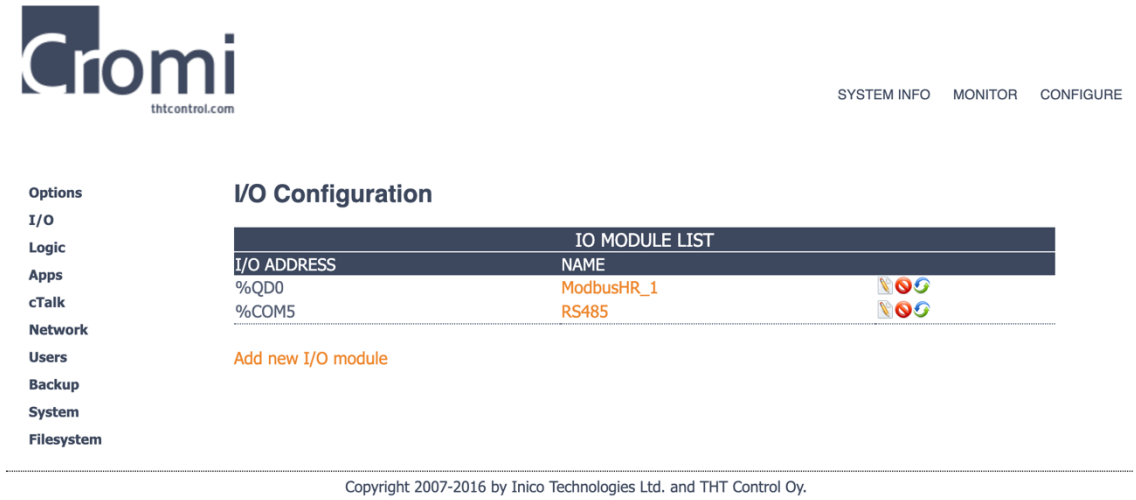


KUVA 21. COM5 -portin konfigurointi

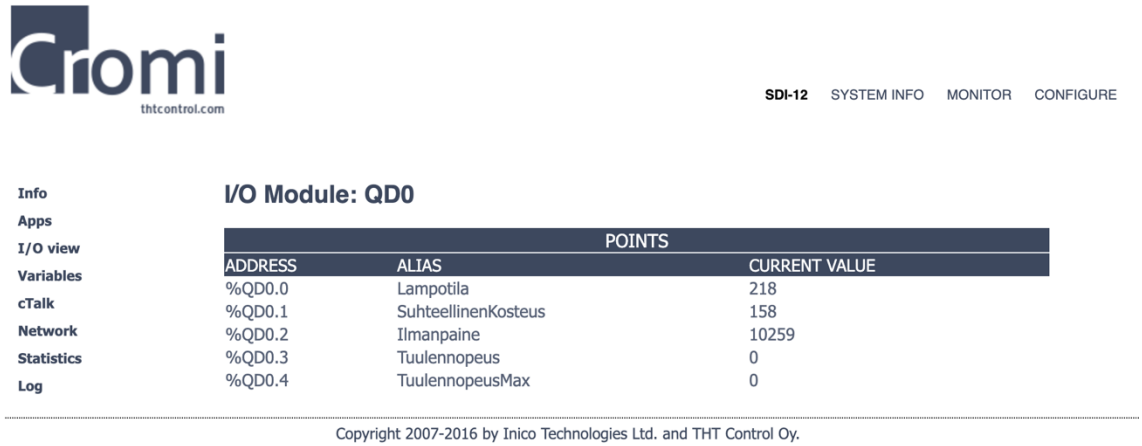
Modbus-rekistereille annettiin slave-osoite, josta tietoa luetaan, rekistereiden määrä, rekisterin tyyppi ja maksimimäärä mittauksia kyselyä kohti. Rekisterit myös nimettiin mittausten monitoroinnin helpottamiseksi (Kuva 22).

KUVA 22. Modbus-rekistereiden konfigurointi

Kun I/O oltiini konfiguroitu (Kuva 23), käynnistettiin logiikka uudelleen Run-tilassa, jotta voitiin monitoroida mittauksia ja Modbus-liikennettä (Kuva 24).

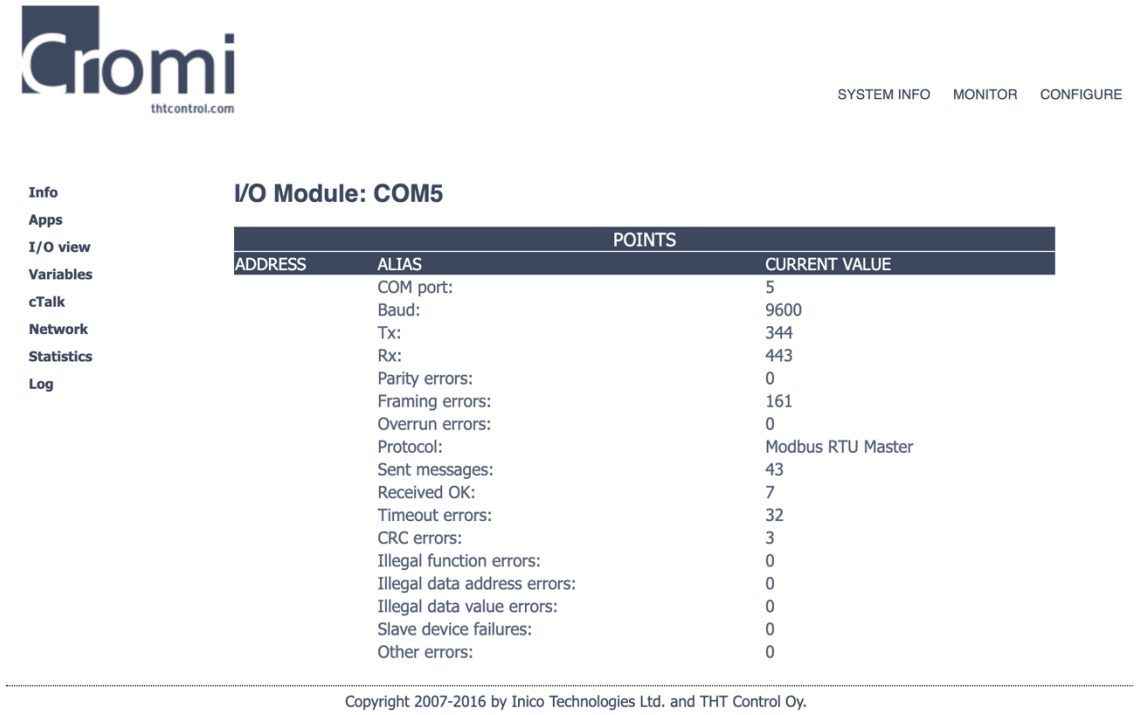


KUVA 23. S1000 I/O -sivu



KUVA 24. S1000 Modbus-rekistereiden monitorointinäkymä

Modbus-liikennettä monitoroidessa pystytään näkemään SDI12-liikenteen aiheuttama häiriö (Kuva 25). 43 lähetetystä kyselystä vain seitsemän osui väliin, jossa Modbus-liikenne on avoin ja SDI12-liikenne katkaistuna.

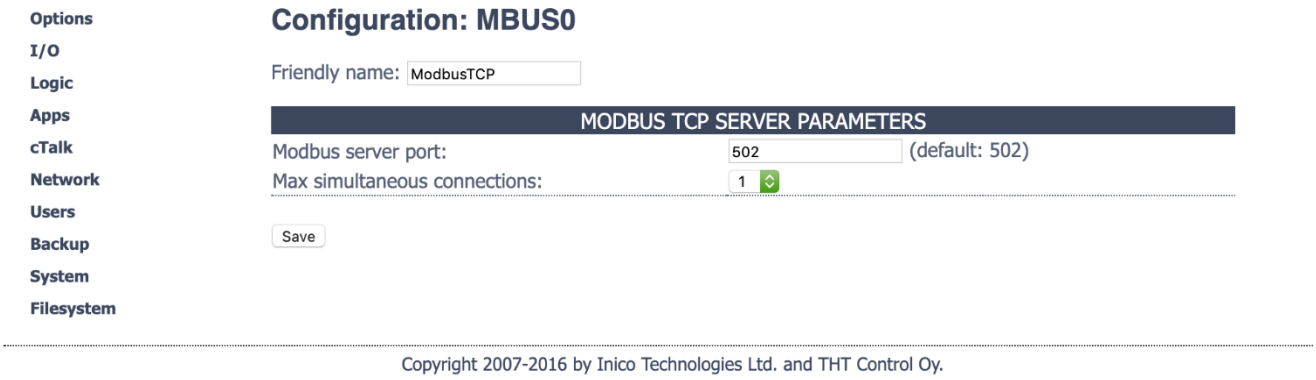


KUVA 25. COM5 -portin monitorointinäkymä

6.5 Valvomonäkymän luonti

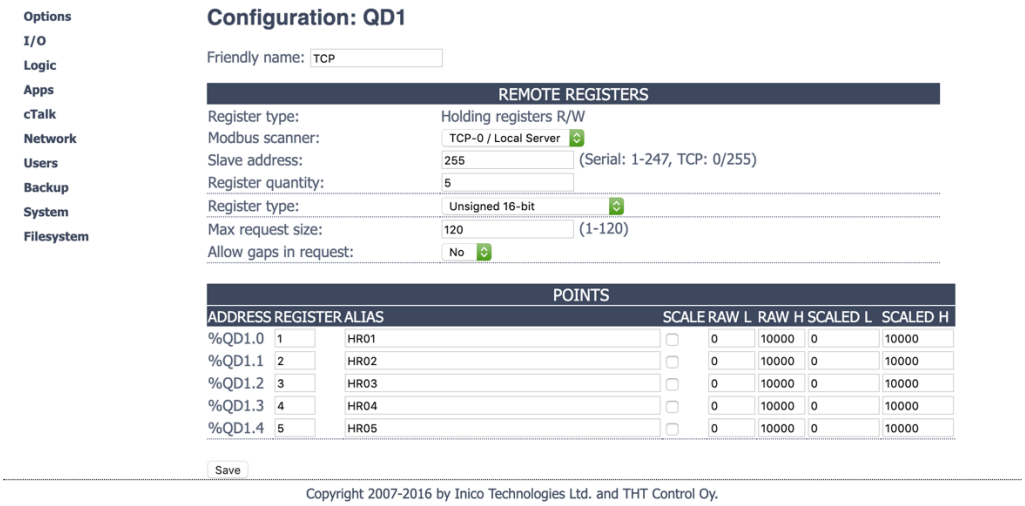
Kun mittaukset olivat luettavissa logiikalla, alettiin työstämään liikennettä valvomoon. Tätä varten S1000-logiikalle täytyi konfiguroida Modbus TCP -serveri (Kuva 26). Modbus TCP mahdollistaa liikenteen toimimisen ethernetin välityksellä. Tarkoituksena oli tuoda mittaukset lähiverkon sisällä valvomokoneelle (agent), josta mittaukset sitten voidaan lukea päävalvomokoneelta (main).

Aluksi logiikalle lisättiin Modbus TCP -serveri, joka mahdollistaa tietojen lähettämisen ethernetin välityksellä client -kohteisiin.



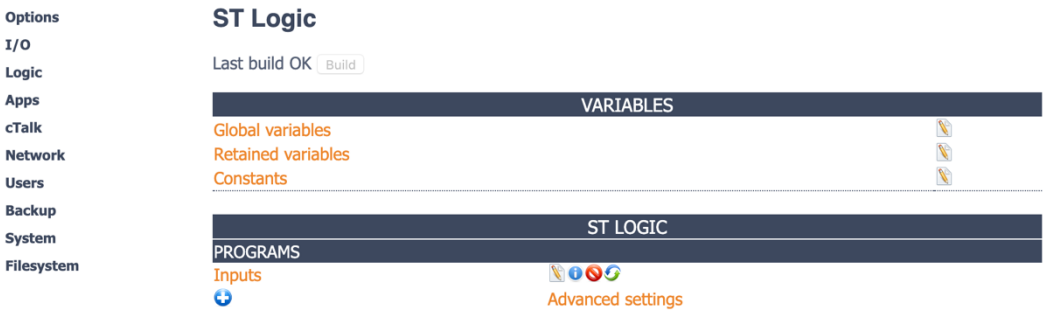
KUVA 26. Modbus TCP server

Tämän jälkeen luotiin uusi IO -konfiguraatio Modbus-rekistereille, joista Modbus TCP Server skannaa tietoja (Kuva 27). Holding-rekisterit nimettiin yksinkertaisesti HR0-HR4 (holding register).



KUVA 27. *Modbus TCP holding rekisterit*

Seuraavaksi haluttiin kopioida aikaisemmin tehtyjen Modbus RTU -rekistereiden tiedot Modbus TCP -rekistereihin. Tämä toteutettiin tekemällä logiikan ohjelmapuolelle ”Inputs”-niminen ohjelma, jossa rekisterisiirto suoritettiin (Kuvat 28 ja 29).



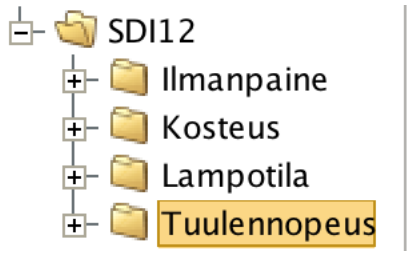
KUVA 28. Inputs -ohjelma

KUVA 29. Rekisterisiirto

Viimeisenä logiikan IP vaihdettiin valvomokoneen kanssa samalle alueelle ja kytkettiin ethernet-kaapelilla ethernet-kytkimeen, johon valvomokone oli myös kytkettynä.

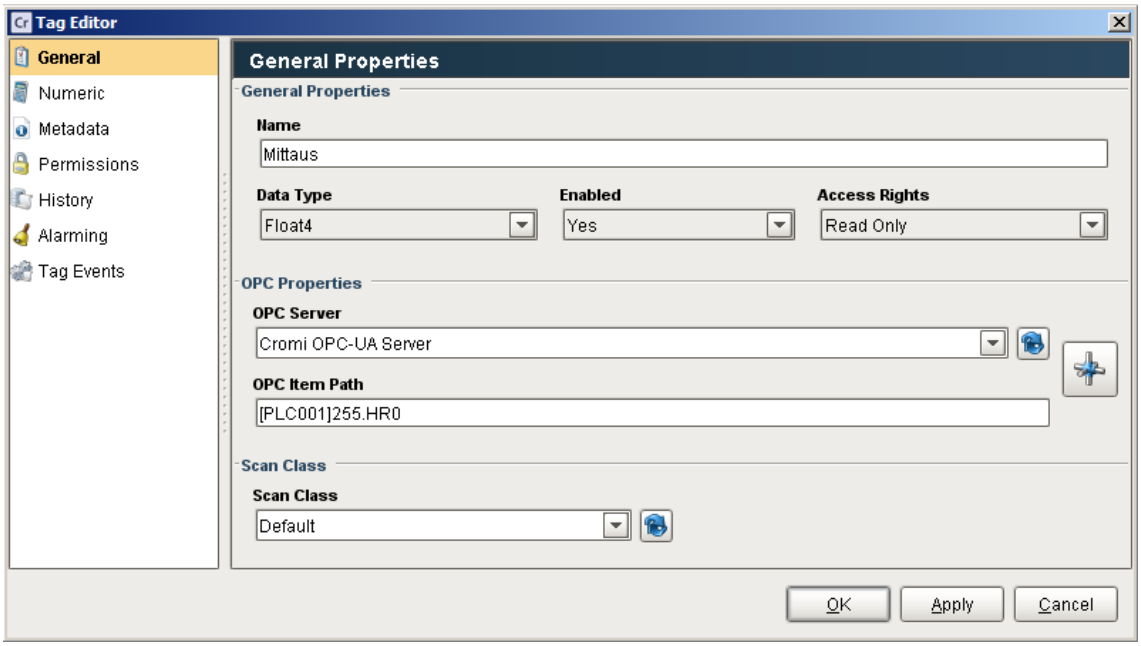
Kun logiikalle oli tehty tarvittavat konfiguroinnit, alettiin tuomaan mittaukset logiikalta valvomokoneelle (agent).

Cromi-valvomoon luotiin ensin selkeä kansiorakenne mittaustageja varten (Kuva 30).



KUVA 30. Valvomon kansiorakenne

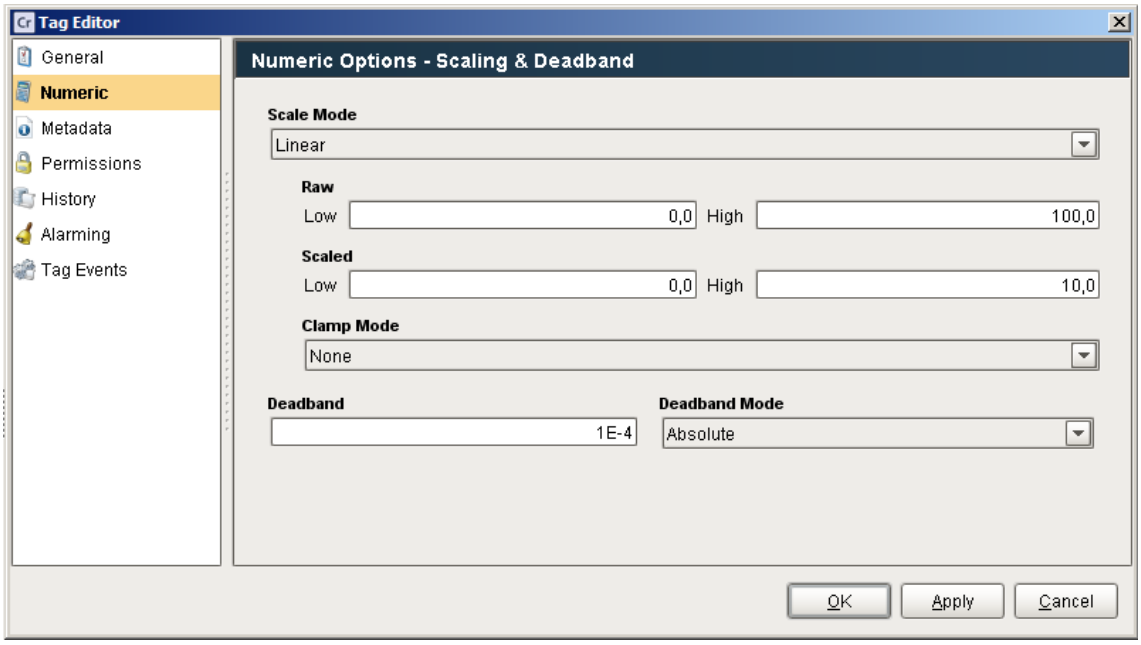
Tämän jälkeen kansioihin sijoitettiin niihin kuuluvat mittaustagit ja niille annettiin osoite, josta tieto noudetaan (Kuva 31).



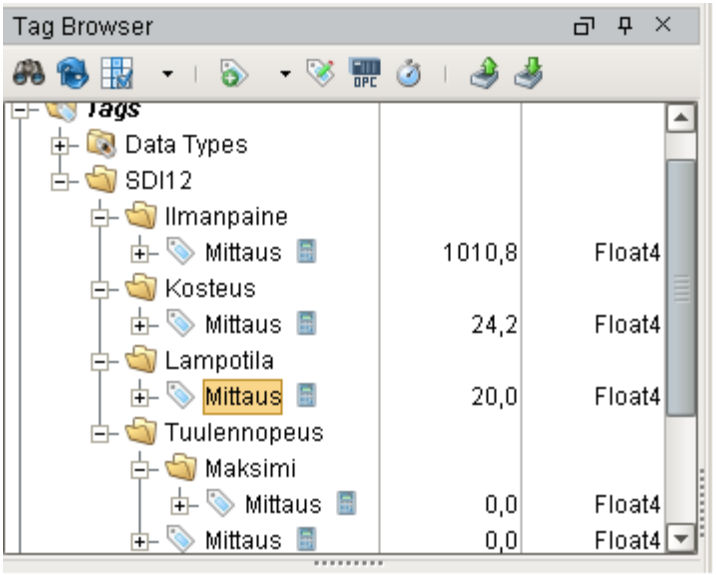
KUVA 31. Tag editor, lämpötilamittaus

Kun mittaustiedot on siirretty, ne voidaan esittää kokonaisluvun sijaan taas float-tyyppisenä. Näin mittauksiin saatiin näkymään desimaalipaikka, josta hankkiuduttiin eroon aikaisemmin.

Desimaalipaikka saadaan takaisin asettamalla skaalattu arvo kymmenen kertaa pienemmäksi kuin raaka arvo (Kuva 32). Tämän jälkeen mittausten tuonti oli valmis (Kuva 33).

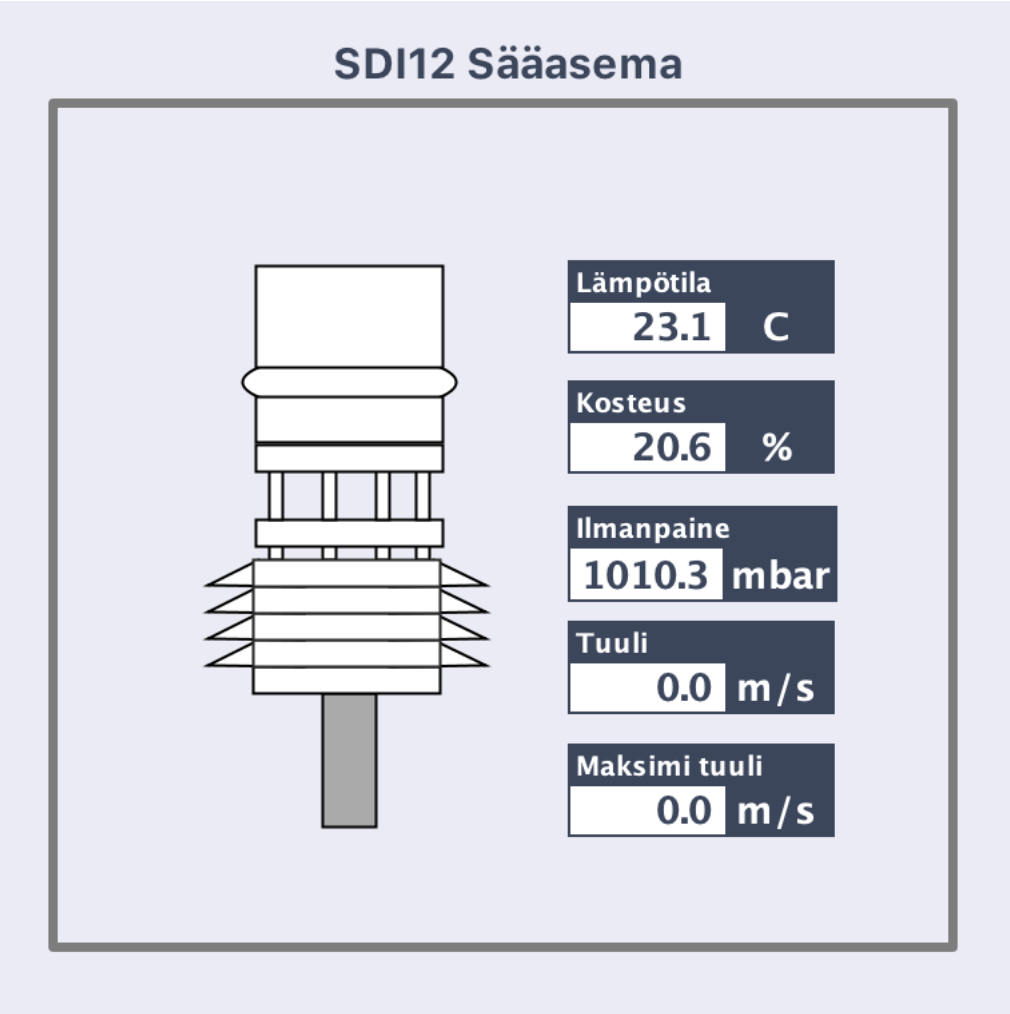


KUVA 32. Mittausarvon skaalaus



KUVA 33. Valmis kansiorakenne

Nyt mittaustagit pystyttiin esittämään visuaalisesti valvomokuvassa (Kuva 34).



KUVA 34. Sääaseman valvomokuva

7 LOPPUTULOS

Työssä oli tarkoitus tutustua SDI-12-standardiin, sen käyttötarkoituksiin ja historiaan. Työosuudessa tarkoituksena oli saada luettua mittaustietoja mikroprosessorianturilta käyttäen SDI-12-sarjaliikenneprotokollaa S1000-logiikkaohjaimelle ja Cromi-valvomoon.

Lopputuloksena työstä meillä on Arduino-ohjelmakoodi, jota pystytään käyttämään SDI-12-protokollaa käyttävien antureiden mittatietojen tuomiseen Modbus-väylää pitkin logiikalle. Mittauksia tulee noin 20 sekunnin välein, joten mihinkään erityisen tarkkaan prosessiin näitä mittauksia ei pysty käyttämään, mutta ne sopivat hyvin historiatietojen keräämiseen ja esimerkiksi taulukoiden tekemiseen anturin mittapaikalla vallitsevista olosuhteista. Logiikalta historiatiedot saa tuotua suoraan valvomoon visuaalisesti esitettävään muotoon ja osalle mittauksista voidaan antaa esimerkiksi hälytysrajat.

8 POHDINTA

SDI-12-protokolla ei ollut tuttu ennestään, vaan koko työ alkoi ensin aineiston etsimisellä ja sen opiskelulla.

Työssä tuli vastaan useita haasteita ja ongelmia, joihin ratkaisujen etsiminen oli erittäin opettavaista ja mielenkiintoista. Opinnäytetyölle rajatun ajan vuoksi ohjelmaan jää vielä kehittämisen varaa. Modbus-liikenteen katkonainen toimiminen kyseissä ohjelmassa aiheuttaa rutkasti timeout-virheitä. Nämä virheet voivat kuormittaa lokitietoja ja niistä olisi hyvä päästä eroon.

Ohjelmaa voisi kehittää esimerkiksi luomalla bitin S1000-logiikalle, joka tilaa muuttamalla voisi kytkeä Modbus-liikenteen aktiiviseksi esimerkiksi tietyin aikavälein. Näin Modbus-väylän kysely voitaisiin helpommin ajoittaa osumaan aina oikeaan kohtaan, eikä logiikan tarvitsisi kysellä slave-laitteelta mittauksia silloin kun niitä ei ole saatavilla. Ideaali ratkaisu olisi päästä eroon Arduino-kehitysalustasta laitteiden välillä ja kehittää SDI-12-protokollaa käyttävä moduuli, jonka saa liitettyä osaksi S1000-logiikkaa.

LÄHTEET

1. SDI-12 historiaa. Saatavissa: http://www.sdi-12.org/history Hakupäivä 16.12.2019

2. SDI-12 standardi. Saatavissa: http://sdi-12.org/specification.php Hakupäivä 16.12.2019

3. ”Tiedonsiirron peruskäsitteitä”. Saatavissa: http://www.cs.tut.fi/etaopetus/ti-tepk/luku19/peruskasitteet.html Hakupäivä 13.4.2020

4. ”Parity bit”. Saatavissa: https://www.computerhope.com/jargon/p/paritybi.html Hakupäivä 13.4.2020

5. "In what applications are SDI-12 devices used?". Saatavissa: http://www.sdi-12.org Hakupäivä 16.12.2019

6. Arduino UNO. Saatavissa: https://store.arduino.cc/arduino-uno-rev3/ Hakupäivä 23.1.2020

7. ”WS601-UMB Smart Weather Sensor”. Saatavissa: https://www.lufft.com/products/compact-weather-sensors-293/ws601-umb-smart-weather-sensor-1831/ Hakupäivä 13.4.2020

8. Arduino UNO RS485 -shield. Saatavissa: https://www.hwhardsoft.de/english/projects/rs485-arduino/ Hakupäivä 13.2.2020