Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut ohjelmistot

2015

Jere Sinisalo

OLOSUHDESEURANTA-JÄRJESTELMÄ

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

2015 | 56

Jari-Pekka Paalassalo, Tiina Ferm

Jere Sinisalo

OLOSUHDESEURANTAJÄRJESTELMÄ

Opinnäytetyön tarkoituksena oli kehittää turkulaiselle mittaus- ja kalibrointialan yritykselle olosuhdeseurantajärjestelmä, jolla mitataan yrityksen kalibrointilaboratorion lämpötilaa, kosteutta ja ilmanpainetta. Mittaustuloksia piti myös pystyä tarkastelemaan jälkikäteen.

Työn käytännön osuus koostuu mittalaitteen ja asiakasohjelman kehityksestä sekä rakentamisesta. Varsinainen mittalaite päätettiin toteuttaa käyttäen Raspberry Pi-mikrotietokonetta ja sen I2C-väylään kytkettäviä olosuhdeantureita. Mittaustulokset tallennetaan laitteelle tietokantaan käyttäen MySQL-tietokantapalvelinta. Mittaustulosten tarkastelua varten kehitettiin asiakassovellus, jonka avulla mittaustulokset saadaan käyttäjälle taulukko- tai graafimuodossa. Lopuksi laitteelle suoritettiin testejä sekä laitteen anturit kalibroitiin ja säädettiin.

Työn tuloksena saatiin kehitettyä ja rakennettua toimiva olosuhdeseurantajärjestelmä, joka täytti toimeksiantajan vaatimukset. Olosuhdeseurantajärjestelmä otettiin käyttöön ja sen kehitystä tullaan jatkamaan myös käyttöönoton jälkeen.

ASIASANAT:

mikrotietokone, Linux, Raspberry Pi, I2C, olosuhdeseuranta, anturi, MySQL

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Software

2015 | 56

Jari-Pekka Paalassalo, Tiina Ferm

Jere Sinisalo

ENVIRONMENTAL MONITORING SYSTEM

The purpose of this thesis was to create an environmental monitoring system for a Finnish calibration company located in Turku. The environmental monitoring system is used for monitoring temperature, humidity and absolute pressure in the company's calibration laboratory. It should be also possible to examine measurement data afterwards.

The practical part of the thesis consists of developing and creating an instrument and a client software. The instrument was chosen to be built on Raspberry Pi microcomputer using environmental sensors connected to its I2C bus. Measurement data are saved in the instrument's MySQL database. The Client software was developed to allow users to inspect saved measurements in numeric and graphical form. Finally, different tests were performed for the instrument and sensors were calibrated and adjusted.

As a result, a functioning environmental system was developed and built. The system fulfilled all required criteria and was taken into use. Developing the system will be continued in the future.

KEYWORDS:

microcomputer, Linux, Raspberry Pi, I2C, environmental monitoring, sensor, MySQL

SISÄLTÖ

LIITTEET

KUVAT

KÄYTETYT LYHENTEET

ACK	Kuittausmerkki, jonka ASCII numero on 6.
ARM	Advanced RISC Machines. Sulautetuissa järjestelmissä yleisesti käytetty 32-bittinen mikroprosessoriarkkitehtuuri.
CRON	Cron on ajastuspalvelu Unix-pohjaisille käyttöjärjestelmille.
I2C	Kaksisuuntainen isäntä-orjaperiaatteella toimiva tiedonsiirtoväylä.
MLC	Multi-level Cell. Tarkoitetaan yleensä flash-muistin tyyppiä, jossa yhdellä muistisolulla voi olla neljä eri arvoa.
NAK	Negatiivinen kuittausmerkki, jonka ASCII numero on 21.
RAM	Random Access Memory eli työmuisti tai käyttömuisti.
RTC	Real Time Clock eli reaaliaikakello.
SBC	Single Board Computer eli yhden piirikortin tietokone.
SLC	Single-level Cell. Eräs flash-muistin tyypeistä, jossa yhdellä muistipaikalla voi olla vain 2 eri arvoa.
SSD	Solid-State Drive on Flash-muistilla toteutettu massamuisti tietokoneilla, joka ei sisällä mekaanisesti liikkuvia osia.
TLC	Triple-level Cell. Mahdollistaa jopa kahdeksan eri arvoa yhdelle muistipaikalle.
UART	Universal Asynchronous Receiver/Transmitter. Sarjaliikennepiiri, joka muun muassa mahdollistaa sarjaliikenteen tietokoneen ja oheislaitteen välillä.

1 JOHDANTO

1.1 Työn tarkoitus ja tavoitteet

Opinnäytetyön tavoitteena oli kehittää turkulaiselle mittaus- ja kalibrointialan yritykselle olosuhdeseurantajärjestelmä, joka mittaisi riittävän tarkasti yrityksen kalibrointilaboratorion lämpötilaa ja kosteutta, sekä ilmanpainetta. Kalibrointitilan olosuhteilla on suuri merkitys tiloissa tehtäville mittauksille, josta syystä niiden seuranta on välttämätöntä. Edellä mainittujen olosuhteiden lisäksi laitteen tulisi mitata myös ulkolämpötilaa ja -kosteutta. Mittaustulokset pitäisi tallentaa automaattisesti, ja niitä täytyy pystyä tarkastelemaan helposti.

1.2 Asiakkaan vaatimukset

Asiakkaan kanssa keskusteltiin työn vaatimuksista, ja päädyttiin seuraaviin vaatimuksiin:

- Laitteen on kyettävä toimimaan itsenäisesti ja tallennettava mittaustuloksia laitteen muistiin, vaikka tietoverkko ei olisi toiminnassa.
- Tallennettuja mittaustuloksia on pystyttävä tarkastelemaan helposti taulukko- ja graafimuodossa.
- Laitteen näytön käyttöliittymän tulee olla selkeä ja mittaustulokset pitää pystyä lukemaan näytöltä pidemmänkin etäisyyden päästä.
- Laitteen on pystyttävä palautumaan sähkökatkoksen jälkeen.
- Laitteen antureiden on oltava kalibroitavissa ja viritettävissä, sekä jokaiselle anturille on oltava mahdollista antaa yksilölliset kalibrointikertoimet.

1.3 Työn rakenne

Opinnäytetyö koostuu teoriaosuudesta, käytännön toteutuksen ja testauksen osuudesta ja yhteenvedosta. Teoriaosuudessa käsitellään lyhyesti mitattavia suureita ja niiden mittausmenetelmiä. Lisäksi teoriaosuus käsittelee työn toteutuksen kannalta keskeisiä aiheita, kuten työssä käytetyn tiedonsiirtoväylän perusteita ja tallennusmedian käyttöön liittyviä haasteita ja rajoitteita.

Varsinainen käytännön toteutuksen osa on jaettu kolmeen osaan, joista ensimmäinen käsittelee järjestelmän laitteistoa. Tässä osassa esitellään lyhyesti eri laitteistokomponentit ja niiden tekniset ominaisuudet sekä mainitaan niiden valintaperusteista. Järjestelmän ohjelmistoa käsittelevässä osassa puolestaan perehdytään laitteiston ohjelmistojen konfigurointiin ja kuvataan laitteelle kirjoitettujen mittausohjelmien toimintaa. Lisäksi osiossa käsitellään järjestelmää varten kirjoitetun asiakassovelluksen toiminta ja ominaisuudet. Viimeisessä käytännön toteutuksen osassa käsitellään järjestelmän testausta ja järjestelmän antureiden kalibrointitoimenpiteitä.

Yhteenvedossa käydään läpi työn lopputulos, esille tulleita ongelmia ja niiden ratkaisuja sekä järjestelmän kehityskohteita. Loppuun on vielä liitetty laitteelle kirjoitettujen ohjelmien lähdekoodit.

2 TEORIA

2.1 Lämpötilan mittaus

Lämpötila on maailmassa yksi mitatuimmista suureista. Lämpötilan mittaaminen on tärkeää, koska monet suureet ovat riippuvaisia lämpötilasta. Esimerkiksi ilmavirtauksia mitattaessa lämpötilalla on suora vaikutus ilman tiheyteen. Lisäksi lämpötila vaikuttaa myös välillisesti erilaisten mittauslaitteistojen tarkkuuteen tai ryömintään (lämpötilakompensointi).

Kappaleen tai aineen lämpötilaa voidaan mitata joko kosketusmittauksella tai koskemattomalla mittauksella. Kosketusmittauksella tarkoitetaan esimerkiksi vastuslämpömittareita, termoelementtejä tai perinteisiä lasilämpömittareita, jotka ovat kosketuksissa mitattavan väliaineen kanssa. Koskemattomassa mittauksessa taas voidaan käyttää esimerkiksi IR-lämpömittareita, jotka eivät ole kosketuksissa mitattavan aineen tai kappaleen kanssa. [1]

Lämpötilaa mitattaessa täytyy mittarin antaa tasaantua mittauskohteen lämpötilaan, jotta luotettavan mittaustuloksen saaminen on mahdollista. Huoneilman lämpötilaa mitattaessa kiinteällä anturilla, ei tämä yleensä tuota ongelmia. [1]

Lisäksi mittaustulokseen vaikuttaa lämpötilamittarin itselämpiäminen, joka tarkoittaa sitä, että elektronisen lämpötila-anturin mittausvirta lämmittää anturia. Opinnäytetyössä käytetyissä lämpötila-antureissa saattaa valmistajan mukaan esiintyä itselämpiäimistä, mikäli antureilta luetaan mittaustuloksia liian nopeaan tahtiin. [1]

2.2 Kosteuden mittaus

Huoneilman kosteutta mitataan erityisesti eri alojen tuotantotiloissa, koska ilmankosteudella on suuri vaikutus erilaisten bakteerien kasvustoon sekä staattisen sähkön varautumiseen. Ilmankosteuden vaikutus ilman tiheyteen täytyy myös huomioida ilmavirtauksia mitattaessa.

Yleisimmin huoneilman kosteus ilmaistaan suhteellisena kosteutena (%RH), joka kertoo, kuinka monta prosenttia absoluuttinen kosteus on vallitsevan lämpötilan kyllästyskosteudesta. Jos ilmaan haihdutetaan vettä yli kyllästyskosteuden, alkaa ilmassa oleva vesi tiivistyä pisaroiksi. Vaihtoehtoisesti ilman kosteutta voidaan ilmaista absoluuttisella kosteudella, joka ilmaisee ilman absoluuttisen vesimäärän.

Ilmankosteutta voidaan mitata useilla eri menetelmillä:

- Hiuskosteusmittari eli mekaaninen mittaus: Hiuskosteusmittarin toiminta perustuu hiuksen venymiseen tai kutistumiseen kosteuden muuttuessa, jolloin hiukseen kiinnitetyn viisarin asento muuttuu.
- Psykrometri: Perustuu menetelmään, jossa käytetään kahta lämpötilamittaria. Toisen anturin ympärillä on kostea kangas ja sitä tuuletetaan vakiopuhalluksella. Nesteen haihtuminen on riippuvainen ympärillä vallitsevasta kosteudesta ja nesteen haihtuessa anturi jäähtyy. Kuivan ja kostean anturin lämpötilaerosta voidaan laskea ilman kosteus.
- Resistiivinen kosteusmittari: Mittaa resistiivisyyden muutosta kosteuden muuttuessa
- Kapasitiivinen kosteusmittari: Mittaa kapasitanssin muutosta ilmassa olevien vesimolekyylien imeytyessä anturin ohueen polymeerikalvoon
- Kastepisteanturi: Kastepisteanturissa on peili, jonka pintaan tiivistyy vesipisaroita sitä tarpeeksi jäähdytettäessä

- NIR-kosteusmittari: Kosketukseton menetelmä kosteuden mittaamiseen, joka hyödyntää mittauksessa IR-aluetta. [2]

2.3 Ilmanpaineen mittaus

Ilmanpaineen mittaamisen yleinen käyttökohde on sääolosuhteiden ennustaminen. Ilmanpaineen muutosta vertailemalla on mahdollista ennustaa säätä. Ilmanpaineen noustessa on yleensä tiedossa tyynempää poutasäätä, kun taas ilmanpaineen laskeminen kertoo sään huonontumisesta ja tuulien voimistumisesta.

Sääolosuhteiden ennustamisen lisäksi ilmanpainetta mitataan, koska sillä on suuri vaikutus erilaisten ilmanvirtausmittareiden kalibroinneissa. Edellä mainitusta syystä kyseisen suureen mittaaminen on erityisen tärkeää toimeksiantajalle.

Ilmanpainetta voidaan mitata muun muassa nestepatsasmanometrilla tai mekaaniseen muodonmuutokseen perustuvilla painemittareilla. Nestepatsasmanometrin toiminta perustuu nestepatsaan liikkumiseen putkessa hydrostatiikan lakien mukaisesti. Ilmanpainetta mitattaessa putken toinen pää on yhdistetty tyhjiöön, jolloin putken toisessa päässä vallitseva ilmanpaine liikuttaa nestepatsasta. Näin voidaan lukea putken mitta-asteikosta nestepatsaan pinnan kohdalta senhetkinen ilmanpaine. [3]

Mekaaniseen muodonmuutokseen perustuvissa painemittareissa mitataan tuntopään mekaanista muutosta ilmanpaineen muuttuessa. Tuntopää voi olla esimerkiksi kalvo, joka venyy paineen muuttuessa. Elektronisissa painemittareissa mekaanista muodonmuutosta voidaan mitata esimerkiksi käyttämällä hyödyksi piezoresistiivisyyttä. Piezoresistiivisyydellä tarkoitetaan puolijohteen tai metallin resistiivisyyden muutosta, kun siihen kohdistetaan mekaanista rasitusta. [3]

2.4 I2C-väylä

I2C-väylä on alun perin Philipsin vuonna 1982 kehittämä kaksisuuntainen sarjaliikenneväylä, joka tunnetaan myös nimillä I^2C- tai IIC-väylä. Väylä koostuu kahdesta linjasta, joista toinen on datalinja (SDA) ja toinen kellopulssilinja (SCL). Väylä toimii isäntä-orjaperiaatteella ja jokainen väylään kytkettävä orjalaite yksilöidään 7-bittisellä tai mahdollisesti harvinaisemmalla 10-bittisellä osoitteella. [4]

I2C-väylä tukee erilaisia tiedonsiirtonopeuksia, joista standard mode (100 kb/s) ja low-speed mode (10 kb/s) ovat yleisimpiä. Uusimmat versiot väylästä käyttävät jopa 3.4 Mb/s tiedonsiirtonopeuksia ja 16-bittisiä osoitteita. [5]

I2C-väylä sopii tarkoituksiin, joissa väylän kustannukset ja yksinkertaisuus menevät nopeuden edelle esim. EEPROM-muistien, AD-muuntimien ja erilaisten antureiden väylänä.

I2C-väylän toimintaperiaate lyhyesti

Isännän kirjoittaessa väylälle, vastaa osoitteen omaava orjalaite ACK-bitillä (vetämällä SDA-linjan maihin). Puolestaan orjalaitteen kirjoittaessa väylälle, kuittaa isäntä kirjoituksen lopuksi ACK-bitillä. Alla oleva luettelo esittää lyhyesti isännän ja orjalaitteen toiminnan väylällä [4]:

1. Isäntä lähettää orjalaitteen osoitteen ja R/W-bitin (0 = kirjoitusoperaatio) väylälle.
2. Kirjoitetaan väylälle orjalaitteen rekisterin osoite, johon halutaan kirjoittaa.*
3. Kirjoitetaan data edellä mainittuun rekisteriin.**
4. Isäntä lähettää orjalaitteen osoitteen ja R/W-bitin (1 = lukuoperaatio) väylälle.

5. Kirjoitetaan väylälle orjalaitteen rekisterin osoite, josta halutaan lukea.*
6. Luetaan data edellä mainitusta rekisteristä.**

(*) Joissain orjalaitteissa rekisteriä ei tarvitse erikseen määritellä ennen luku- tai kirjoitustapahtumaa, vaan voidaan hypätä tämän kohdan yli.

(**) Tarvittaessa voidaan kirjoittaa tai lukea useampia tavuja peräkkäin.

2.5 Tallennusmedian elinikä

Yksi opinnäytetyön haasteista oli käytettävän tallennusmedian eliniän pidentäminen. Raspberry Pi -mikrotietokoneet käyttävät tallennustilanaan flash-muistiin perustuvia SD-kortteja, joiden elinikä eli uudelleenkirjoituskertojen määrä on rajallinen. Uudelleenkirjoituskertojen määrä riippuu käytettävästä muisti-tekniikasta ja ”wear level” -menetelmästä. Useimmiten tunnettujen valmistajien muistikortit kestävät useampia uudelleenkirjoituskertoja.

Erilaisia muistitekniikoita ovat muun muassa SLC, MLC ja TLC, jotka kuvaavat muistisolun rakennetta. SLC-tekniikkaa käyttävissä muisteissa muistisolulla voi olla ainoastaan kaksi jännitearvoa, kun taas MLC- ja TLC-muistitekniikoissa yhdellä muistisolulla voi olla useampia jännitearvoja. MLC- ja TLC-muistien etuina on edullisuus ja muistitiheys, mutta lyhyempi elinikä johtuen muistisolujen jännitearvojen erottelukyvyn huonontumisesta.

SLC-tekniikkaa käyttävien muistien uudelleenkirjoituskertojen määrä on yleensä noin kymmenkertainen MLC-tekniikkaa käyttäviin muisteihin ja lähes satakertainen TLC-tekniikkaa käyttäviin muisteihin verrattuna [6]. SLC-tekniikkaa käyttävät muistit ovat myös nopeampia ja kuluttavat vähemmän virtaa, mutta eivät ole kalliin hintansa takia yleisiä kulutuselektroniikassa.

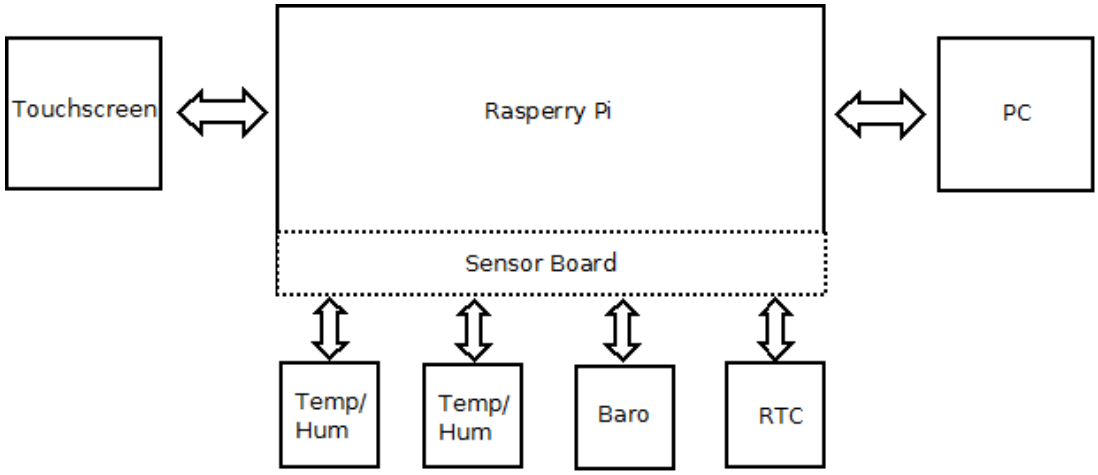
Useiden flash-muistien elinikää pyritään parantamaan erilaisilla ”wear level” -menetelmillä, jotka tarvitsevat toimiakseen muistikortille integroidun mikrokontrollerin. Edellä mainittuja toimintoja ei ole kuitenkaan spesifioitu SD-korttien standardissa, vaan niiden käyttäminen on täysin kiinni valmistajasta. Näin ollen ”wear level” -menetelmiä on yleensä käytössä vain kalliimmissa SD-korteissa. Dynamic wear level -menetelmää käyttävissä muisteissa kirjoitettava data kirjoitetaan aina sellaisiin vapaisiin muistipaikkoihin, joissa on vähiten kirjoituskertoja. Static wear level -menetelmä puolestaan on monimutkaisempi ja saattaa vaikuttaa negatiivisesti muistikortin suorituskykyyn. Jälkimmäisessä menetelmässä siirretään pitkään samana pysynyttä dataa muistipaikkoihin, joissa on eniten kirjoituskertoja. Static wear level -menetelmä pidentää huomattavasti tehokkaammin muistin elinikää. [7]

Tallennusmedian elinikää onkin tarpeen pidentää käyttäjän omilla toimenpiteillä:

- Siirretään lokien kirjoitus RAM-muistille tai poistetaan lokien kirjoitus kokonaan käytöstä.
- Käytetään kapasiteetiltaan suurempaa muistikorttia.
- Poistetaan swap eli sivutusmuisti käytöstä.
- Poistetaan tiedostojärjestelmän journalointi käytöstä.
- Lisätään /etc/fstab-tiedostoon noatime-, nodiratime-optiot.

3 LAITTEISTO

Järjestelmän laitteisto koostuu mikrotietokoneesta, anturipiirikortista, antureista, kosketusnäytöstä, SSD-levystä ja virtalähteestä, sekä niiden kaapeloinneista. Lisäksi järjestelmään voidaan laskea tietoverkko, sekä PC-tietokone, jolla tarkastellaan mittaustuloksia. Alla yksinkertaistettu kuva järjestelmän osista.



Kuva 1. Kuva järjestelmästä.

3.1 Mikrotietokone

Järjestelmän pohjaksi päätettiin valita laajaan laitteisto- ja ohjelmistotuen omaava Raspberry Pi -mikrotietokone, joka on suuressa suosiossa harrastajien keskuudessa edullisen hintansa ja kattavan tuen ansiosta. Valintaan vaikutti myös aikaisempi kokemus kyseisen mikrotietokoneen käytöstä. Mikrotietokoneella on mahdollista ajaa useita eri Linux-distribuutioita sekä RISC OS-käyttöjärjestelmää. Raspberry Pi on hieman luottokorttia suurempi yhden piirikortin mikrotietokone (SBC), joka on varustettu 700 MHz ARM-pohjaisella prosessorilla. Tähän mennessä mikrotietokoneesta löytyy kolmea eri mallia Model A, Model B ja Model B+.

Opinnäytetyössä käytetty malli on Model B, joka on varustettu 512 MB keskusmuistilla. Kyseisen mikrotietokoneen katsottiin olevan ominaisuuksiltaan riittävä opinnäytetyön aihetta varten.

Liitäntäväylät

Mikrotietokoneesta löytyy seuraavat liitännät ja väylät.

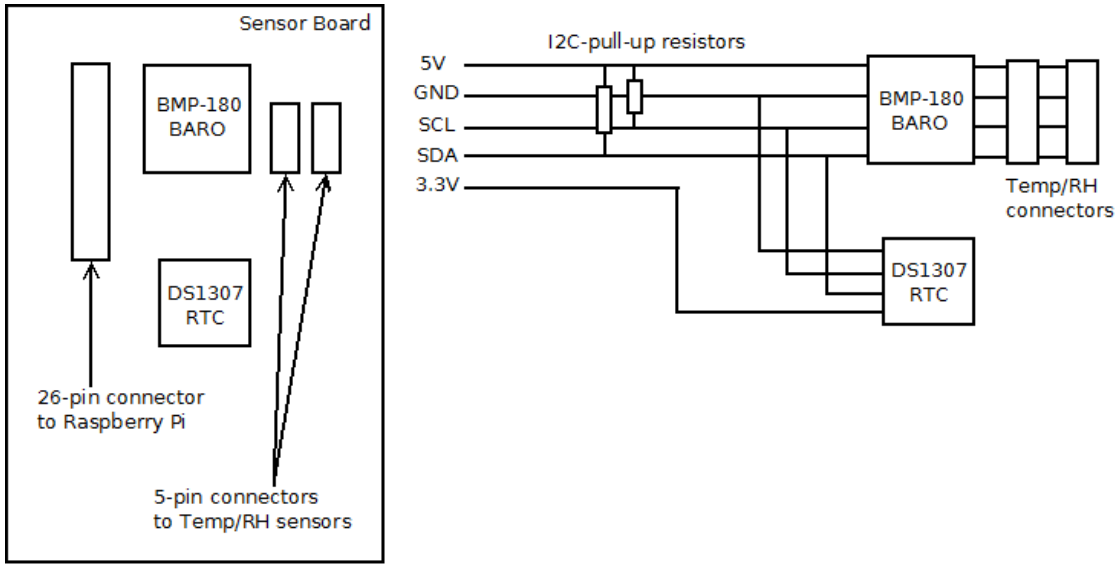
- 2 x USB 2.0-liitäntä, 10/100 Ethernet-liitäntä
- RCA- ja HDMI-videoulostulot, 3.5 mm -audioulostulo
- 8 x GPIO-liitäntä, 5V-jännitelähtö, 3,3V-jännitelähtö
- UART-, I2C- ja SPI-väylät.
- 15-pinninen CSI-liitäntä kameramoduulille.

3.2 SSD-levy

Laitteeseen asennettiin SSD-levy järjestelmän eliniän ja suorituskyvyn parantamiseksi. SSD-levy on Kingstonin valmistama ja se on kapasiteetiltaan 64 GB. Levy kytketään mikrotietokoneen USB-väylään ja se tarvitsee oman virransyöttönsä.

3.3 Anturipiirikortti

Kaikki anturit ja liitännät antureille on laitettu omalle piirikortilleen, jotta niiden vaihtaminen vikatilanteessa olisi mahdollisimman yksinkertaista. Lisäksi piirikortille on juotettu tarvittavat ylösvetovastukset I2C-väylää varten ja virtaliittimet. Piirikortille jätettiin tyhjää tilaa tulevaisuuden varalle, mikäli laitteeseen halutaan liittää lisää antureita tulevaisuudessa. Seuraavalta sivulta löytyvästä kuvasta selviää piirikortin rakenne virtaliittimiä lukuun ottamatta.



Kuva 2. Dialla piirretty kuva ja piirikaavio anturipiirikortista.

3.4 Näyttö

Laitteen näytöksi päätettiin valita 4DSystemsin valmistama uLCD-32PTU -kosketusnäyttö. Näyttö on 3.2”:n LCD-näyttö 240x320 pikselin resoluutiolla ja 65 000 värin väriavaruudella. Näytössä on oma prosessorinsa ja näyttöä voidaan ohjata sarjaportin kautta. Näytön piirikortilta löytyy myös muistia, I2C-väylä, 13 GPIO-liitäntää, 2 sarjaporttia ja 8 kappaletta 16-bittisiä ajastimia.

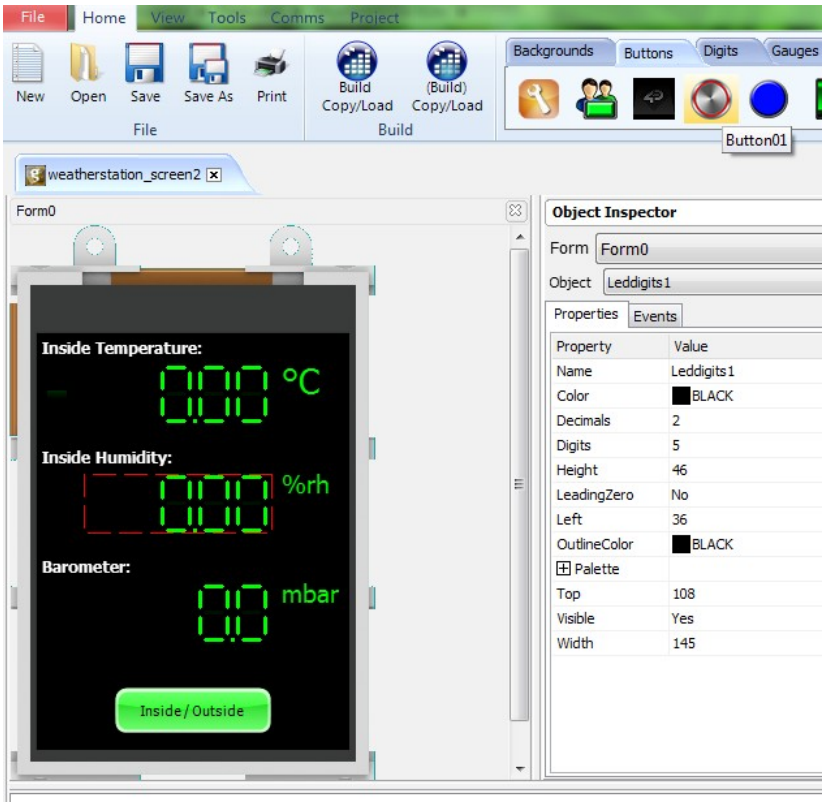
Näytön kehitystyökalut

Näyttöä voidaan ohjelmoida usealla eri tavalla, joista helpoin on käyttää täysin graafista Visi-Genie -työkalua. Visi-Genie mahdollistaa näytön käyttöliittymän suunnittelun ilman, että käyttäjän tarvitsee osata 4DGL-koodausta. Kaikki graafiset elementit viedään näytölle drag&drop-periaatteella ja elementtien ominaisuuksien muuttaminen onnistuu helposti graafista työkalua käyttämällä. Kun näytön käyttöliittymä on valmis, siirretään käännetty tiedosto microSD-muistikortille ja kytketään muistikortti näyttöön. Tämän jälkeen näytöllä olevien elementtien ohjaaminen onnistuu sarjaportin kautta lähetettävillä komennoilla.

Ajan säästämiseksi päätettiin laitteen näytön käyttöliittymä toteuttaa kokonaan käyttäen Visi-Genie -työkaluja.

Näytön käyttöliittymän suunnittelu

Näytön käyttöliittymän osalta vaatimukseksi oli asetettu helppokäyttöisyys ja selkeys, joten käyttöliittymästä päätettiin tehdä mahdollisimman selkeä. Käyttöliittymä koostuu kahdesta lähes samanlaisesta lomakkeesta, joissa molemmissa on segmenttinäytöt lämpötilalle, kosteudelle ja ilmanpaineelle. Toisella lomakkeella näytetään sisätilan lämpötila ja kosteus, kun taas toisella lomakkeella näytetään vastaavat tiedot rakennuksen ulkopuolelta. Näiden lomakkeiden välillä voidaan vaihdella yksinkertaisesti näytön alareunassa olevaa painiketta painamalla. (Kuva 3.) Käyttöliittymän kehityksen jälkeen otettiin graafisten elementtien osoitteet talteen ja siirryttiin kirjoittamaan aliohjelmaa näytön ohjaamiseksi mikrotietokoneen sarjaportin kautta.



Kuva 3. Näytön käyttöliittymän kehitys Visi-Geniellä.

3.5 Lämpötila- ja kosteusanturit

Lämpötilan ja -kosteuden mittaamiseen valittiin Sveitsiläisen Hygrochipin valmistama HYT-271 kosteus- ja lämpötila-anturi. Anturin kosteusmittaus perustuu kapasitiiviseen mittausmenetelmään ja lämpötilan mittauksessa käytetään PTAT-menetelmää. PTAT-menetelmä perustuu transistorin sisällä olevan diodin päästöjännitteen muutokseen lämpötilan muuttuessa. Anturi pitää sisällään mikropiirin, joka muuttaa analogisen mittausarvon digitaaliseen muotoon ja mahdollistaa sen lukemisen I2C-väylän kautta.

Anturin mittausalue lämpötilalle on -40...125 °C ja suhteelliselle kosteudelle 0...100 %RH. Lämpötilamittauksen tarkkuudeksi luvataan ±0,2 °C (0...60 °C) ja resoluutioksi 0,015 °C. Kosteusmittaukselle tarkkuutta luvataan ±1,8 %RH (0...80 %RH).

Anturin käyttöjännitealue on 2,7 – 5,5 V ja mittausdatan siirto tapahtuu I2C-väylän välityksellä. Antureiden osoite on vaihdettavissa ohjelmallisesti, joten niitä on mahdollista liittää useampia samalle I2C-väylälle. Alla valmistajan datalehdestä löytyvä kuva anturista.

Kuva 4. HYT-271 Lämpötila- ja kosteusanturi [8].

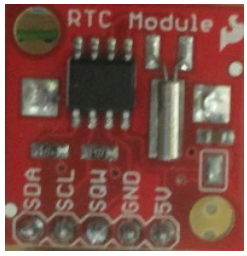
3.6 Ilmanpaineanturi

Ilmanpaineanturiksi valikoitui Boschin valmistama BMP180-ilmanpaineanturi. Anturilla voidaan mitata absoluuttista painetta alueella 300 - 1 100 hPa. Anturin suhteellinen tarkkuus ilmanpainealueella (950 - 1050 hPa) on ±0,12 hPa, joka riittää käyttötarkoitukseen erittäin hyvin. Anturissa on myös sisäänrakennettu lämpötila-anturi, jolla korjataan paineanturin mittaustuloksia. Anturi kytketään I2C-väylään ja sen käyttöjännitealue on 1,8 - 3,6 V [9].

Kuva 5. Ilmanpaineanturipiirikortti.

3.7 Reaaliaikakello

Raspberry Pi -mikrotietokone ei sisällä omaa reaaliaikakelloa (RTC) kustannussyistä, joten päätettiin laitteistoon lisätä erillinen reaaliaikakello. (Kuva 6.) Reaaliaikakellon tehtävänä on pitää oikea kellonaika tallessa, vaikka mikrotietokoneesta katkeaisi virransyöttö. Tällöin voidaan myös varmistua, että järjestelmä saa oikean kellonajan sähkökatkoksenkin jälkeen, vaikka tietoverkko olisi kaatunut.

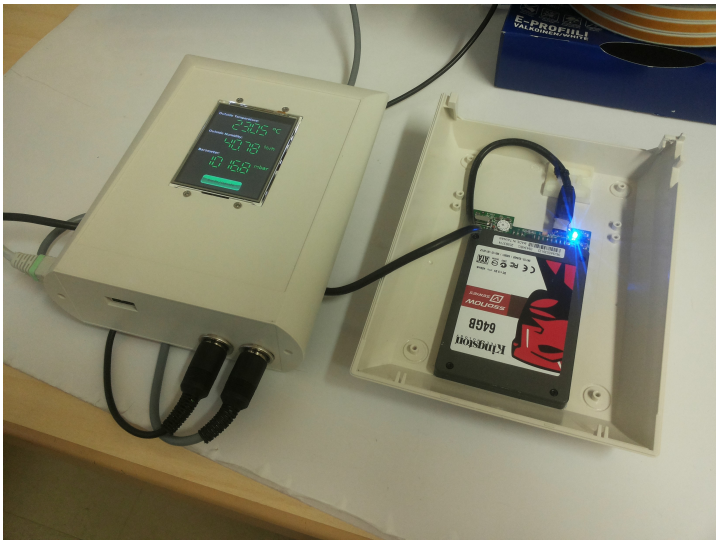


Kuva 6. Reaaliaikakellon piirikortti.

Opinnäytetyöhön valittiin I2C-väylään kytkettävä Dallas Semiconductorin valmistama DS1307-reaaliaikakello. Ylimääräisen juottamisen ja rakentelun välttämiseksi päätettiin käyttää valmista piirikorttia, jossa on valmiina kaikki reaaliaikakellon toimintaan tarvittavat komponentit. Reaaliaikakello on varmistettu CR1225-paristolla, jonka varauksen luvataan kestävän yhdeksän vuotta ilman ulkoista virransyöttöä. [10]

3.8 Kotelointi ja kokoonpano

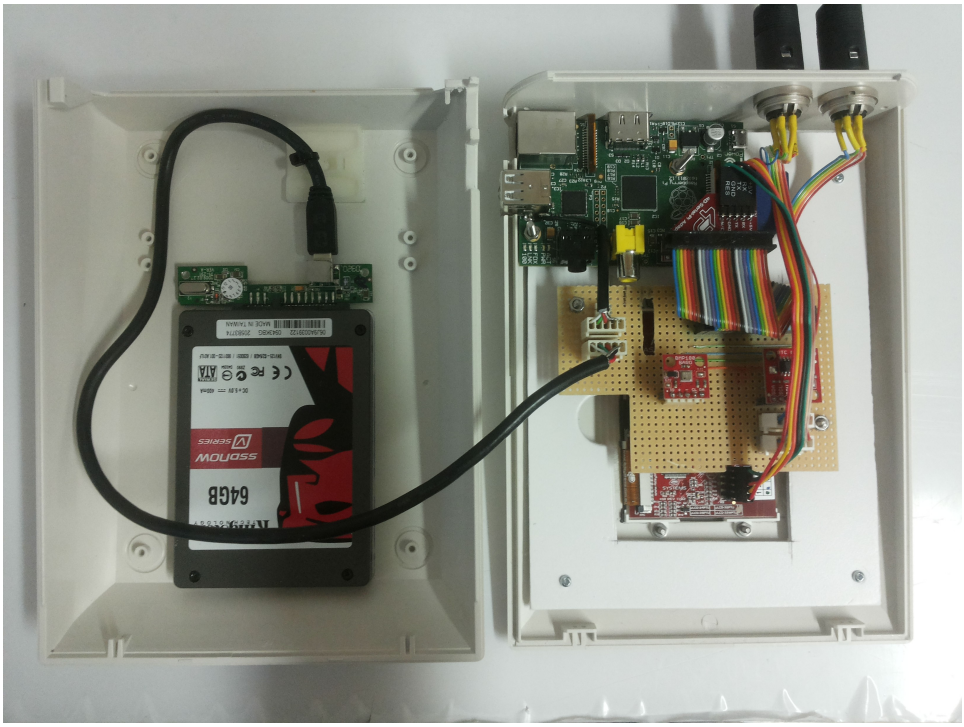
Laitteen koteloksi hankittiin OKW:n valmistama 200 x 150 x 72 mm:n kokoinen muovikotelo. Koteloon takaosaan kiinnitettiin 5-napaiset din-liittimet lämpötila- ja kosteusantureita varten sekä virtaliitin. Koteloon tehtiin myös reiät hdmi-, USB- ja Ethernet-liitäntöjä varten. Kotelon päällikanteen tehtiin reikä laitteen kosketusnäytölle. (Kuva 7.)



Kuva 7. Kuva keskeneräisestä laitteesta.

Järjestelmän osien sijoittaminen kotelon sisälle oli haastavaa, sillä kompaktin kotelon sisällä ei ollut juurikaan ylimääräistä tilaa. Osat sijoitettiin niin, että tulevaisuudessa laitteen osien vaihtaminen ja järjestelmän ohjelmistojen päivitys olisi mahdollisimman yksinkertaista. Lisäksi yritettiin huomioida eniten

lämpöä tuottavien osien sijoittaminen ylemmäs kotelossa, jolloin lämpö ei poistuessaan lämmittäisi kotelon sisällä olevaa ilmanpaineanturia ja reaaliaikakelloa. Vaikkakaan lämpenemisellä ei pitäisi olla suurta vaikutusta ilmanpaineanturin tarkkuuteen, sillä anturi on lämpötilakompensoitu.



Kuva 8. Osien sijoittelu laitteen sisällä.

3.9 Virransyöttö

Laite käyttää virtalähteenä ulkoista 5 V 2 A virtalähdettä. Laitteelle päätettiin hankkia ulkoinen virtalähde, jottei virtalähde lämmittäisi turhaan laitteiston muuta elektroniikkaa. Virransyöttö mikrotietokoneelle toteutettiin itse tehdyllä kaapelilla, joka juotettiin mikrotietokoneeseen. Kaapeli sisältää virransyötön lisäksi myös USB-dataliitännät, jotka on juotettu kiinni mikrotietokoneen USB-porttiin. Edellä mainittuun kaapeliratkaisuun päädyttiin, jottei liitäntää SSD-levylle tarvitse ottaa kotelon ulkopuolella olevista USB-liittimistä ja tuoda takaisin sisälle koteloon.

4 OHJELMISTO

4.1 Mikrotietokoneen konfigurointi

Mikrotietokoneen käyttöjärjestelmäksi asennettiin Raspbian Wheezy 3.12 -käyttöjärjestelmä. Raspbian on ilmainen Debian Linuxiin pohjautuva käyttöjärjestelmä, joka on optimoitu Raspberry Pi -alustalle.

4.1.1 I2C-väylän käyttöönotto

Oletuksena Raspbianissa I2C-väylän käyttö on estetty, joten se joudutaan erikseen avaamaan seuraavilla toimenpiteillä.

1. Ensimmäisenä lisätään */etc/modules* tiedostoon rivit:

   i2c-bcm2708

   i2c-dev

2. Kommentoidaan */etc/modprobe.d/raspi-blacklist.conf* -tiedostosta rivi:

   blacklist i2c-bcm2708

3. Käynnistetään mikrotietokone uudelleen ja asennetaan vielä *i2c-tools*, jolla on kätevä testata i2c-laitteiden toimintaa. (Kuva 9.)

```
pi@raspberrypi ~/testit/HYT271 $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- 28 -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi ~/testit/HYT271 $
```

Kuva 9. i2cdetect tunnistaa väylälle kytketyt laitteet.

4.1.2 Sarjaportin käyttöönotto

Jotta sarjaporttia voisi käyttää ohjelmassa, täytyy suorittaa seuraavat toimenpiteet.

1. Estetään sarjaportin kautta kirjautuminen kommentoimalla /etc/inittab -tiedostosta seuraava rivi: *T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100*
2. Lisäksi kannattaa estää mikrotietokonetta lähettämästä Boot-infoa sarjaportin kautta poistamalla */boot/cmdline.txt* tiedostosta kaikki ttyAMA0 viittaukset.
3. Lopuksi suoritetaan mikrotietokoneen uudelleenkäynnistys *sudo reboot* -komennolla.

4.1.3 Reaaliaikakellon käyttöönotto

Aluksi on syytä tarkistaa, että käyttöjärjestelmän kello on oikeassa ajassa ja aikavyöhyke on asetettu oikein.

1. Asetetaan oikea aikavyöhyke komennolla: *sudo dpkg-reconfigure tzdata*
2. Ladataan RTC-moduuli komennolla: *sudo modprobe rtc-ds1307*
3. Suoritetaan seuraava komento pääkäyttäjän oikeuksilla: *echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device*
4. Kirjoitetaan RTC-kellolle oikea aika ja päivämäärä komennolla: *sudo hwclock -w*
5. Lisätään tiedostoon */etc/modules* seuraava rivi: *rtc-ds1307*
6. Lopuksi lisätään vielä alla olevat rivit tiedostoon */etc/rc.local*

   echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device

   sudo hwclock -s

Sen jälkeen voidaan tarkistaa *”sudo hwclock -r”* -komennolla reaaliaikakellolta aika ja päivämäärä. Yllä olevien toimenpiteiden jälkeen järjestelmän pitäisi jokaisen käynnistyksen yhteydessä hakea oikea aika ja päivämäärä reaaliaikakellolta.

4.1.4 Toimenpiteet tallennusmedian eliniän pidentämiseksi

Koska SD-korttien ”wear level”-menetelmät ja kapasiteetti ovat yleensä huomattavasti huonompia kuin SSD-levyissä, päätettiin varsinainen tiedostojärjestelmä sijoittaa SSD-levylle. SD-korttia tarvitaan edelleen laitteen boottaamiseen, mutta sinne ei juurikaan kirjoiteta mitään.

SSD-levyn elinikää pyritään pidentämään siirtämällä lokien kirjoitus RAM-muistiin tmpfs käyttäen. Lisäksi */etc/fstab*-tiedostoon lisättiin noatime- ja nodiratime-optiot ja poistettiin järjestelmän swap eli sivutusmuisti käytöstä. Tiedostojärjestelmän journalointi päätettiin ainakin toistaiseksi jättää päälle, koska järjestelmälle ei ole varavirtaa, jolloin käyttövirran katkeaminen saattaisi aiheuttaa tiedostojärjestelmän vikaantumisen.

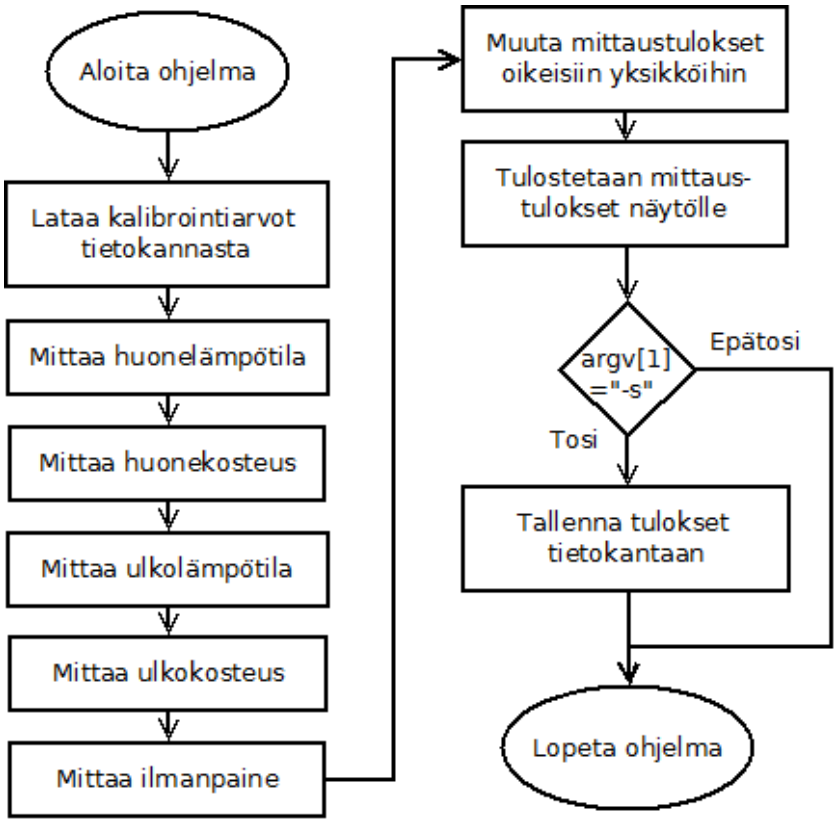
Alla olevasta kuvasta selviää järjestelmän uusi tiedostorakenne, jossa SD-korttia käytetään vain varsinaisen tiedostojärjestelmän käynnistämiseen. Lisäksi kuvassa näkyy tmpfs avulla RAM-muistille siirretyt tiedostopolut.

```
pi@raspberrypi ~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs           57G  3.0G   51G   6% /
/dev/root        57G  3.0G   51G   6% /
devtmpfs        215M     0  215M   0% /dev
tmpfs            44M  220K   44M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs            88M     0   88M   0% /run/shm
/dev/mmcblk0p1   56M  9.7M   47M  18% /boot
tmpfs            30M     0   30M   0% /var/tmp
tmpfs           100M  104K  100M   1% /var/log
```

Kuva 10. Ruutukaappaus tiedostorakenteesta.

4.2 Pääohjelma

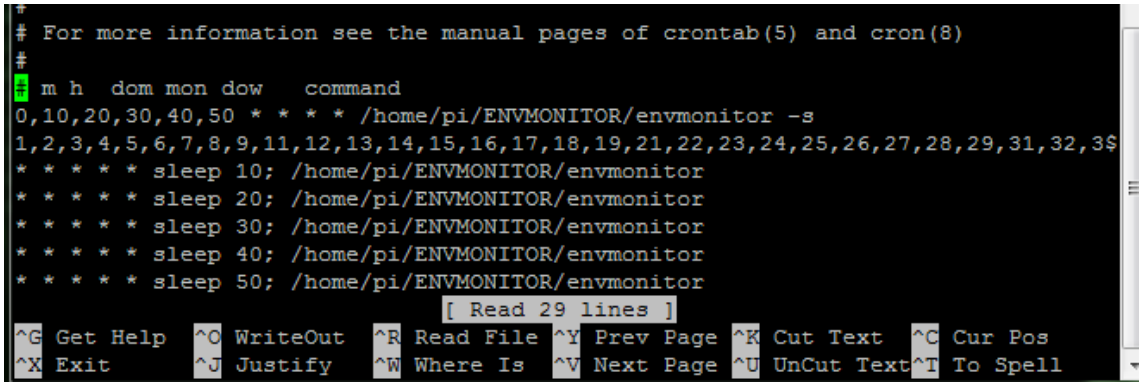
Pääohjelma ja kaikki aliohjelmat on kirjoitettu c-kielellä. Pääohjelma aloittaa alustamalla muuttujia, jonka jälkeen lataa jokaisen anturin kalibrointikertoimet niille varattuihin muuttujiin. Sen jälkeen ohjelma hakee mittaustulokset antureilta käyttäen antureille tehtyjä aliohjelmia ja muuntaa tulokset oikeisiin yksikköihin. Kun mittaustulokset ovat muunnettu oikeisiin yksikköihin, tulostetaan ne laitteen näytölle käyttäen LCD-näyttöä ohjaavaa aliohjelmaa. Lopuksi tarkastellaan pääohjelman kutsuparametreista, että tallennetaanko tulokset tietokantaan. Tietokantaan tallentamisessa käytetään myös sille tehtyä aliohjelmaansa. Alla vuokaavio selventämään pääohjelman toimintaa.



Kuva 11. Vuokaavio pääohjelman toiminnasta.

Pääohjelmaa ajetaan ajastettuna Cronin avulla. Pääohjelma ajetaan 10 sekunnin välein ilman parametreja ja 10 minuutin välein käyttäen ”-s”-

parametria, jolloin mittaustulokset tallentuvat tietokantaan. Crontab-tiedostosta tuli hieman monimutkainen, sillä oli tärkeää, ettei pääohjelmaa ajeta samaan aikaan ilman parametria ja parametrin kanssa. (Kuva 12.)



Kuva 12. Ruutukaappaus crontab-tiedostosta.

4.3 Aliohjelmat

Ohjelmisto koostuu pääohjelman lisäksi viidestä aliohjelmasta. Aliohjelmat on pyritty jakamaan mahdollisimman selkeästi laitteisto- ja toimintokohtaisesti. Jokaiselle anturityypille, sekä näytölle on oma aliohjelmansa. Tämän lisäksi kalibrointiarvojen lukemiselle tietokannasta ja mittaustulosten tallentamiselle tietokantaan on molemmille erilliset aliohjelmansa.

4.3.1 Lämpötilan ja kosteuden mittaaminen

Aliohjelma sisältää vain yhden funktion *getTempOrHum()*, joka ottaa parametreiksi anturin osoitteen ja muuttujan (*tOrH*), jolla määrätään palautetaanko lämpötila vai kosteus. Jos muuttujan *tOrH* arvo on 0, niin funktio palauttaa anturin lämpötilan ja muussa tapauksessa kosteuden. Funktio palauttaa arvon *long*-tyypin arvona.

Alussa funktio alustaa tarvittavat muuttujat ja avaa I2C-väylän. Väylän avaamisen jälkeen funktio käskee anturia aloittamaan mittauksen, jonka jälkeen funktio lukee anturilta tulevat neljä tavua. Vastaanotetusta neljästä tavusta

funktio laskee mitatun kosteuden ja lämpötilan, jonka jälkeen funktio palauttaa halutun tuloksen pääohjelmalle.

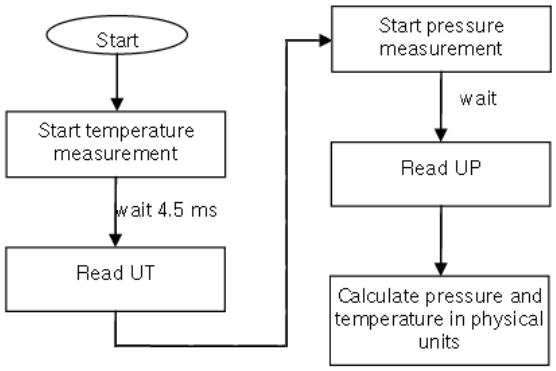
4.3.2 Ilmanpaineen mittaaminen

Aliohjelma sisältää vain yhden funktion *readPressure()*, joka ottaa parametreiksi anturin osoitteen ja halutun mittaustarkkuuden (*oss*). Funktio palauttaa todellisen paineen *long*-tyypin arvona.

Aluksi funktio alustaa tarvittavat muuttujat, jonka jälkeen se avaa I2C-väylän ja yrittää lukea paineanturilta tehtaalla määritetyt kalibrointiarvot. Paineanturin sisäiset kalibrointiarvot (11 kpl) ovat 16-bittisiä kokonaislukuja ja niitä tarvitaan todellisen ilmanpaineen ja lämpötilan laskemiseen. Kalibrointiarvot luettuaan ohjelma lähettää ohjausrekisteriin (*0xF4*) komennon *0x2E*, jolloin paineanturi aloittaa lämpötilan mittaamisen. Tämän jälkeen ohjelma odottaa 15 ms ja lukee lämpötilan mittaustuloksen datarekisteristä *0xF6*.

Lämpötilalukeman saatuaan ohjelma lähettää käskyn lukea paineanturilta ilmanpaineen (*0x34+oss<<6*) ja odottaa 50 ms, jonka jälkeen lukee jälleen tuloksen datarekistereistä 0xF6, 0xF7 ja 0xF8.

Lopuksi ohjelma suorittaa suuren määrän laskutoimituksia, joilla edelliset ”raaka-arvot” saadaan muutettua todellisiksi mittausarvoiksi ja palauttaa todellisen ilmanpaineen pääohjelmalle.



Kuva 13. Valmistajan vuokaavio ilmanpaineen mittauksesta [11].

4.3.3 Näytön päivitys

Näytön päivitykseen tarkoitettu aliohjelma sisältää kaksi funktiota, joista vain yksi näkyy pääohjelmalle ja kolme proseduuria.

uLCD_init() proseduurin tarkoituksena on asettaa sarjaportin asetukset näytölle sopivaksi ja avata yhteys.

uLCD_sync() funktio kirjoitettiin, koska Raspberry Pi -mikrotietokoneella on tapana lähettää välillä ”roskaa” sarjaportin kautta. Funktio lähettää näytölle dataa sarjaportin kautta, kunnes näyttö vastaa NACK komennolla. Funktion suorituksen jälkeen näytölle pystyy lähettämään dataa normaalisti.

sendCommand() funktio lähettää sarjaportin kautta näytölle dataa. Parametreiksi annetaan lähetettävä data ja datan pituus. Funktio palauttaa arvon 1, jos näyttö tunnistaa komennon ja vastaa *ACK*. Jos näyttö ei tunnista lähetettyä komentoa (*NACK*), niin funktio palauttaa arvon 0.

uLCD_setValue() proseduuri näkyy pääohjelmalle, ja se ottaa parametreiksi näytöllä olevan graafisen elementin osoitteen ja sille syötettävän arvon. Proseduuri muuntaa myös *long*-tyypin osoitteen ja arvon *sendCommand()* -funktiolle sopivaksi dataksi (kuusi peräkkäistä tavua) ja laskee niille tarkistussumman. Tarkistussumma lasketaan suorittamalla jokaiselle tavulle XOR-operaatio.

uLCD_close() proseduuri sulkee sarjaportin.

4.3.4 Mittaustulosten tallennus tietokantaan

Mittaustulokset antureilta tallennetaan tietokantaan kymmenen minuutin välein. Tietokanta koostuu kahdesta taulusta; *measurementdata*, johon tallennetaan kaikki antureiden mittaustulokset ja *calibrationdata*, joka sisältää jokaiselle anturille kalibrointikertoimet.

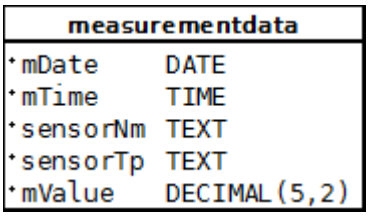
Mittaustulosten tallentamisessa tietokantaan käytetään sille tarkoitettua aliohjelmaansa. Aliohjelma sisältää neljä proseduuria.

mysql_connect() proseduurin tehtävänä on muodostaa yhteys MySQL-tietokantaan.

mysql_disconnect() proseduurin tehtävänä on katkaista yhteys tietokantaan, kun tarvittavat kirjoitusoperaatiot on suoritettu.

mysql_write() proseduurilla lähetetään parametrina annettu merkkijono tietokannalle.

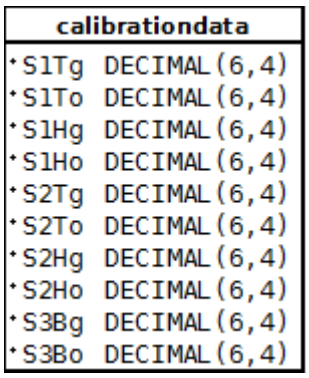
savemeasurement() on aliohjelman ainoa ulospäin näkyvä proseduuri. Se ottaa parametreikseen anturin nimen, tyypin ja mittaustuloksen. Näistä se muodostaa SQL-kyselyn, jonka se lähettää mysql_write() -proseduurille parametrina. Lopuksi proseduuri sulkee yhteyden tietokantaan.



Kuva 14. Tietokannan measurementdata-taulu.

4.3.5 Kalibrointiarvojen lataaminen tietokannasta

Pääohjelman ensimmäinen tehtävä on kutsua aliohjelmaa, joka lataa jokaiselle anturille kalibrointikertoimet tietokannan *calibrationdata*-taulusta. (Kuva 15.)



Kuva 15. Tietokannan calibrationdata-taulu.

Taulun kentät on nimetty käyttäen seuraavaa periaatetta. Ensimmäiset kaksi merkkiä ilmaisevat anturin tunnuksen, jonka jälkeen ilmaistaan yhdellä merkillä mitattava suure ja lopuksi merkitään onko kyseessä gain-kerroin vai offset-arvo.

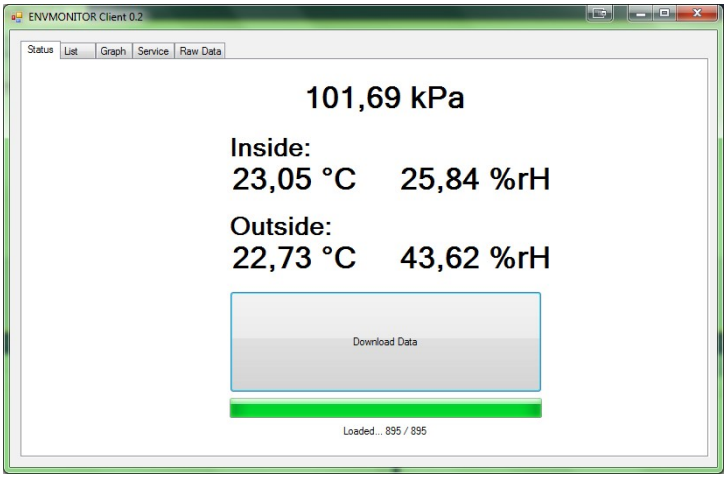
4.4 Tietokannan varmuuskopiointi

Tietokanta varmuuskopioidaan yrityksen tiedostopalvelimelle kerran viikossa käyttäen Cronin avulla ajastettua Linux-komentosarjaa.

4.5 Asiakassovellus

Yksi kehitettävän mittausjärjestelmän vaatimuksista oli mittaustulosten helppo tarkastelu graafien ja taulukoiden muodossa, joko erillisellä sovelluksella tai web-käyttöliittymän kautta.

Asiakassovellus päätettiin kehittää käyttäen Microsoftin Visual Studio -työkaluja aikaisempien hyvien kokemuksien ansiosta. Ohjelmointikieleksi valittiin C# ja sovellus käyttää valmiiksi löytyviä Visual Studion graafi- ja taulukko-komponentteja. Lisäksi MySQL-tietokantayhteyttä varten ladattiin erillinen kirjasto, joka hoitaa tietokantayhteyden asiakassovellukselta mittalaitteelle.

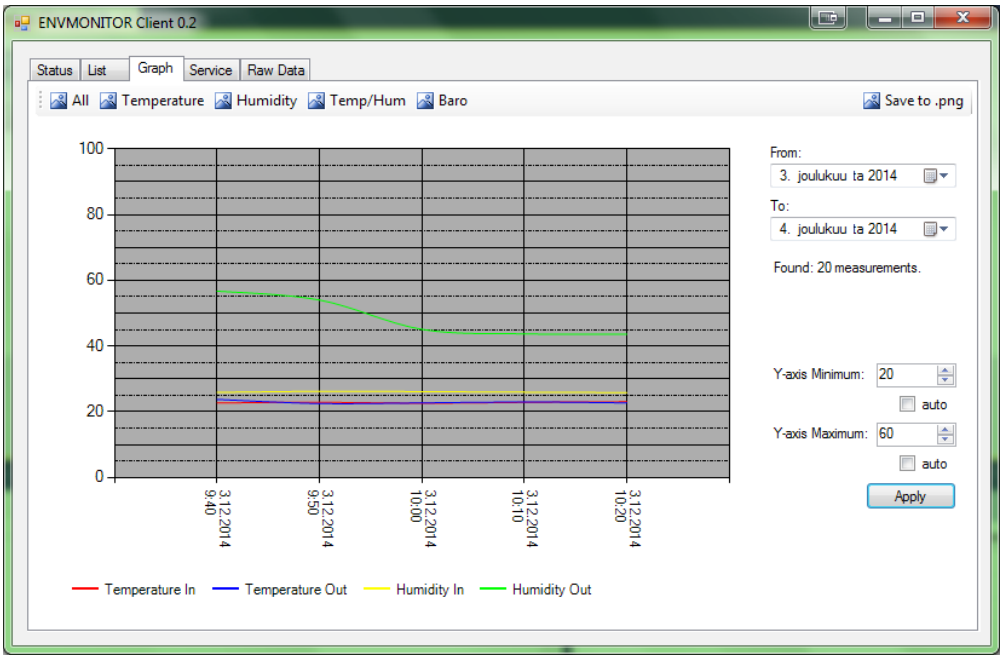


Kuva 16. Asiakassovelluksen pääikkuna.

Sovelluksen toiminta ja ominaisuudet

Aluksi sovellus luo yhteyden tietokantaan ja lataa tietokannasta antureilta tallennetut mittaustulokset. Mittaustulokset ladataan *measurementdata*-taulusta ja tallennetaan niille varattuihin muuttujiin. Tämän jälkeen mittaustuloksia on mahdollista tarkastella taulukko- tai graafimuodossa halutulla aikavälillä.

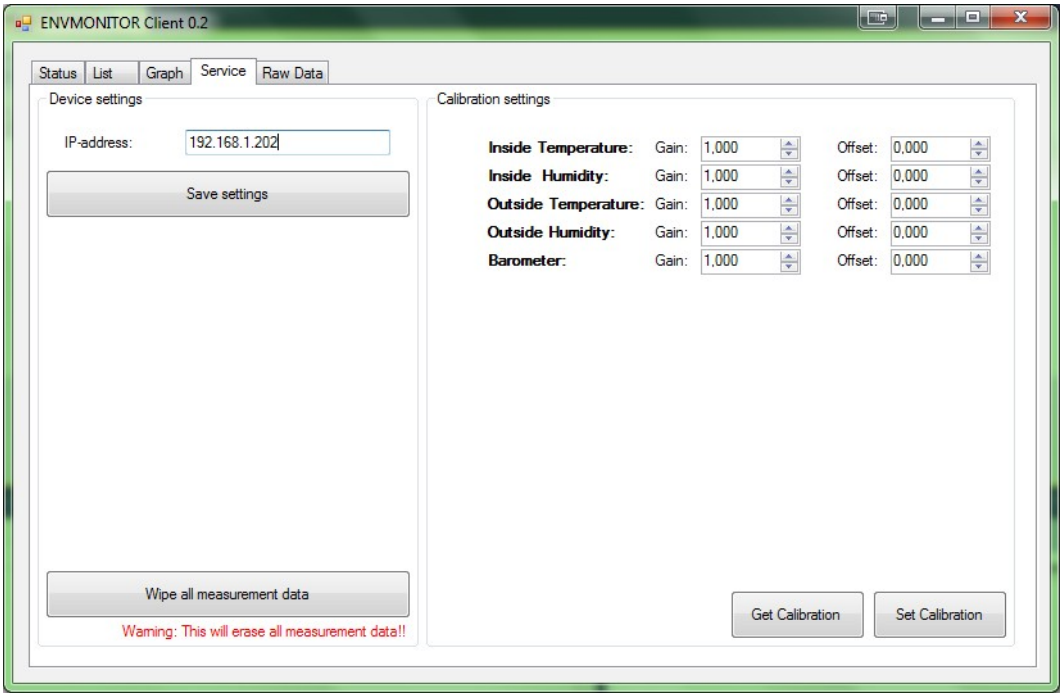
Asiakassovelluksen graafinäkymässä on mahdollista erotella mitä käyriä halutaan piirtää sekä muuttaa graafin akselin arvoja. Lisäksi graafi voidaan tallentaa kuvana png-tiedostomuotoon. (Kuva 17.)



Kuva 17. Kuva asiakassovelluksen graafinäkymästä.

Taulukkovälilehdellä voidaan tarkastella yksittäisiä mittaustuloksia eri antureilta ja viedä ne csv-tiedostoksi Exceliä varten.

Asiakassovelluksen huoltovälilehdellä on mahdollista lukea antureille asetetut kalibrointikertoimet tai kirjoittaa antureille uudet kalibrointikertoimet. Edellä mainittujen toimintojen lisäksi samalta välilehdeltä löytyy mahdollisuus tyhjentää kaikki laitteelle tallennetut mittaustulokset. (Kuva 18.)



Kuva 18. Asiakassovelluksen huoltovälilehti.

Asiakassovelluksen ensimmäisellä käyttökerralla täytyy laitteen ip-osoite syöttää huoltovälilehdellä olevaan IP-address-kenttään ja tallentaa asetukset. Tämän jälkeen ohjelma osaa jatkossa käynnistyessään hakea tallennetut asetukset automaattisesti.

5 TESTAUS JA KALIBROINTI

5.1 Testaus

Järjestelmän vakautta ennen käyttöönottoa on yritetty testata käyttämällä normaalia suurempaa mittaustulosten näytteenottonopeutta. Opinnäytetyön aikataulu ei kuitenkaan salli kovin pitkää testivaihetta, joten laitteen toiminnan seurantaa jatketaan käyttöönoton jälkeen ja suoritetaan tarvittaessa mahdollisia korjaustoimenpiteitä järjestelmän vakauden parantamiseksi. Laitteen antureiden toimintaa tullaan testaamaan jokaisen kalibroinnin yhteydessä, jotta voidaan varmistua mittaustulosten oikeellisuudesta.

5.2 Kalibrointi ja viritys

Yksi järjestelmän vaatimuksista oli antureiden viritysmahdollisuus, sillä vaikka anturit onkin kalibroitu ja viritetty tehtaalla, täytyy niiden olla viritettävissä myös jatkossa. Laitteen anturit ovat viritettävissä asiakassovelluksen avulla ja jokaiselle anturille on mahdollista antaa gain- ja offset-kalibrointiarvot. Vastaavaa menetelmää käytetään yleisesti AD-muuntimien virityksessä. Menetelmässä alkuperäinen mittaustulos kerrotaan ensin gain-kertoimella, jonka jälkeen tulokseen lisätään offset-arvo.

Lämpötila- ja kosteusantureiden kalibrointi suoritettiin käyttäen apuna Weiss WK11/180-sääolosuhdekaappia, jossa on mahdollista ajaa lämpötiloja -40...180 °C välillä. Laitteen lämpötila- ja kosteuslukemia verrattiin referenssimittarin lukemiin ja kirjattiin lukemat ylös. Referenssimittarina toimi Vaisalan lämpötila-kosteusmittari, joka on kalibroitu Vaisalan akkreditoidussa kalibrointilaboratoriossa.

Ilmanpainemittarin kalibrointi suoritettiin alipainekammiossa käyttäen apuna Druckin valmistamaa referenssipainemittaria, joka on kalibroitu Suomen kansallisessa mittanormaalilaboratoriossa.

Kalibroinnin jälkeen laskettiin kalibroitaville antureille uudet kalibrointiarvot ja lähetettiin ne laitteelle. Tämän jälkeen suoritettiin vielä mittaukset kahdessa mittauspisteessä, jotta voitiin varmistua uusien kalibrointiarvojen oikeellisuudesta.

6 YHTEENVETO

Järjestelmä täytti sille asetetut vaatimukset ja toimeksiantaja oli tyytyväinen lopputulokseen. Laitteelle kaavailtua partikkelimittausominaisuutta ei saatu toteutettua ajan puitteissa, sillä sopivaa mittaria ei ole saatu yritykselle vielä hankittua.

Järjestelmää kehitettäessä tuli vastaan muutamia odottamattomia haasteita, joista ehkä haastavin oli tallennusmedian rajallinen kirjoituskertojen määrä. Nähtäväksi jääkin, mikä järjestelmän lopullinen elinikä tallennusmedian osalta tulee olemaan nykyisellä ratkaisulla. Lisäksi ongelmia tuottivat joidenkin valmiiden I2C-kirjastojen puutteet, joten antureiden aliohjelmat kirjoitettiin uudelleen käyttäen Linuxin mukana tulevia I2C-kirjastoja. Vaikka siitä aiheutuikin lisää opiskelua ja ohjelmointia, saatiin lopputuloksena toimivat ajurit, joiden ei pitäisi olla yhtä laitteistoriippuvaisia.

Järjestelmää tullaan kehittämään tulevaisuudessa sekä asiakassovelluksen että varsinaisen laitteen osalta. Laitteen osalta suurimpia kehityskohteita on partikkelimittausominaisuuden lisääminen järjestelmään, tallennusmedian eliniän pidentäminen ja uusien ominaisuuksien lisääminen kosketusnäytön käyttöliittymään. Asiakassovelluksessa puolestaan on vielä paljon kehitettävää käyttöliittymän ja tietokantakyselyjen optimoinnin osalta. Lisäksi asiakassovellukseen täytyy lisätä mahdolliset uudet anturit, joita järjestelmään tullaan lisäämään tulevaisuudessa.

.

LÄHTEET

[1] Mikes Metrologia: Lämpötilan mittaus, Tekijä: Thua Veckström, pdf-dokumentti, viitattu 2.11.2014, Saatavilla: http://metrologia.fi/mikes/Oppaat/J4_2005_Lampotilan_mittaus.pdf

[2] Ilmatieteenlaitos: Lämpötila ja kosteus, www-dokumentti, viitattu 2.11.2014, Saatavilla: http://ilmatieteenlaitos.fi/lampotila-ja-kosteus#15

[3] Mikes Metrologia: Paineen mittaus, Tekijä: Sari Saxholm ja Markku Rantanen, pdf-dokumentti, viitattu 16.2.2015, Saatavilla: http://www.metrologia.fi/mikes/Oppaat/J1_2011_Paineen_mittaus.pdf

[4] Robot Electronics: Using the I2C Bus, www-dokumentti, viitattu 18.12.2014, Saatavilla: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

[5] Wikipedia: I2C, www-dokumentti, viitattu 18.12.2014, Saatavilla: http://en.wikipedia.org/wiki/I%C2%B2C

[6] Wikipedia: Flash Memory, www-dokumentti, viitattu 13.11.2014, Saatavilla: http://en.wikipedia.org/wiki/Flash_memory

[7] Micron: NAND Wear-Leveling Techniques, pdf-dokumentti, viitattu 11.11.2014, Saatavilla: http://www.micron.com/-/media/documents/products/technical%20note/nand%20flash/tn2942_nand_wear_leveling.pdf

[8] Hygrochip: Digital Humidity Sensor HYT-271, pdf-datalehti, viitattu 9.10.2014, Saatavilla: http://www.farnell.com/datasheets/1719946.pdf

[9] Bosch Sensortec: BMP180 Technical data, www-dokumentti, viitattu 9.10.2014, Saatavilla:http://www.bosch-sensortec.com/en/homepage/products_3/environmental_sensors_1/bmp180_1/bmp180

[10] Sparkfun Real Time Clock Module, www-dokumentti, viitattu 12.11.2014, Saatavilla: https://www.sparkfun.com/products/12708

[11] Bosch: BMP180 Digital Pressure Sensor, pdf-datalehti, viitattu 10.10.2014, Saatavilla:http://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-09.pdf

Pääohjelman lähdekoodi

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "logger.h"
#include "BMP180.h"
#include "HYT271.h"
#include "loadcal.h"
#include "ULCD.h"

#define S1 0x28 //HYT271 #1
#define S2 0x29 //HYT271 #2
#define S3 0x77 //BMP180

#define getTemp 0
#define getHum 1

int main(int argc, char **argv) {

    printf("This is Enviromental Monitoring System ver 1.0\n");

    //Check arguments
    int save=0;
    if (argc==2){
        if (strncmp(argv[1],"-s",2)==0){
            save=1;
        }
```

```
}

//Load Calibration Values
double Gain[5];
double Offset[5];
int i;
for (i=0;i<5;i++){
    Gain[i]=loadcalibration(i*2);
    Offset[i]=loadcalibration(i*2+1);
    printf("G:%f O:%f\n",Gain[i],Offset[i]);
}

//Measure
long rawTin = getTempOrHum(S1,getTemp);
double Tin = (rawTin / 100.0);
long rawHin = getTempOrHum(S1,getHum);
double Hin = (rawHin /100.0);
long rawTout = getTempOrHum(S2,getTemp);
double Tout = (rawTout / 100.0);
long rawHout = getTempOrHum(S2,getHum);
double Hout = (rawHout /100.0);
long rawBaro = readPressure(S3,3);
double Baro = (rawBaro / 1000.0);

//Calculate
Tin = Gain[0]*Tin + Offset[0];
Hin = Gain[1]*Hin + Offset[1];
Tout = Gain[2]*Tout + Offset[2];
Hout = Gain[3]*Hout + Offset[3];
Baro = Gain[4]*Baro + Offset[4];

//Print to uLCD
long lcdTin=Tin*100;
long lcdHin=Hin*100;
long lcdTout=Tout*100;
long lcdHout=Hout*100;
```

```
    long lcdBaro=Baro*100;

    uLCD_init();
    uLCD_sync();
    uLCD_setValue(uTin,abs(lcdTin));
    uLCD_setValue(uHin,abs(lcdHin));
    uLCD_setValue(uTout,abs(lcdTout));
    uLCD_setValue(uHout,abs(lcdHout));

    if (lcdTin < 0){
        uLCD_setValue(uNegTin,1);
    }else{
        uLCD_setValue(uNegTin,0);
    }

    if (lcdTout < 0){
        uLCD_setValue(uNegTout,1);
    }else{
        uLCD_setValue(uNegTout,0);
    }

    uLCD_setValue(uBaro,lcdBaro);
    uLCD_setValue(uBaro2,lcdBaro);
    uLCD_close();

    //Save all to database
    if(save){
        savemeasurement("S1T", "TEMP", Tin);
        savemeasurement("S1H", "HUMI", Hin);
        savemeasurement("S2T", "TEMP", Tout);
        savemeasurement("S2H", "HUMI", Hout);
        savemeasurement("S3B", "BARO", Baro);
        printf("All saved!\n");
    }
    printf("Exiting!\n");
    return 0;
}
```

Lämpötila- ja kosteusanturin lähdekoodi

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <math.h>
#include "HYT271.h"


long getTempOrHum(int address, int tOrH) {

    int fd;
    char *fileName = "/dev/i2c-1"; // I2C bus name
    unsigned char buf[10]; //Buffer for data
    long temperature;
    long humidity;

     if ((fd = open(fileName, O_RDWR)) < 0) { // Open port for reading and
writing
        printf("Failed to open i2c bus!\n");
        //exit(1);
        return (-254);
    }

    if (ioctl(fd, I2C_SLAVE, address) < 0) {
        printf("Unable to get bus access to talk to slave!\n");
        //exit(1);
        return (-253);
    }
    buf[0] = 0;
```

```
if ((write(fd, buf, 1)) != 1) { // Send register we want to read from
     printf("Write Error!\n");
     //exit(1);
     return (-252);
}

if (read(fd, buf, 4) != 4) { // Read data into buffer
     printf("Read Error!\n");
     //exit(1);
     return (-251);
}
else {
     unsigned char humhighByte = buf[0];
     unsigned char humlowByte = buf[1];
     unsigned int rawhumidity = ((humhighByte <<8) + humlowByte);
     rawhumidity = rawhumidity &= 0x3FFF;
     humidity = 10000/pow(2,14)*rawhumidity;
     unsigned char temphighByte = buf[2];
     unsigned char templowByte = buf[3];
     templowByte=(templowByte &= 0x3F);
     unsigned int rawtemperature = (temphighByte <<6)+templowByte;
     rawtemperature = rawtemperature &= 0x3FFF;
     temperature = 16500.0/pow(2,14)*rawtemperature-4000;

     if (tOrH==0){
          return temperature;
     }else{
          return humidity;
     }

}

return -256;
```

Ilmanpaineanturin lähdekoodi

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <math.h>
#include <stdint.h>
#include "BMP180.h"

long readPressure(int address, int16_t oss) {

    int fd;
    char *fileName = "/dev/i2c-1"; // I2C bus name
    unsigned char buf[24]; //Buffer for data
    //Uncorrected pressure and temperature
    long UT,UP;
    //Cal coefficients:
    int16_t AC1,AC2,AC3,B1,B2,MB,MC,MD;
    uint16_t AC4,AC5,AC6;

     if ((fd = open(fileName, O_RDWR)) < 0) { // Open port for reading and
writing
         printf("Failed to open i2c bus!\n");
         exit(1);
    }

    if (ioctl(fd, I2C_SLAVE, address) < 0) {
         printf("Unable to get bus access to talk to slave\n");
         exit(1);
    }
```

```
// **** Read Calibration Coefficients ****

buf[0] = 0xAA; //Start reading from EEPROM address 0xAA

if ((write(fd, buf, 1)) != 1) { // Send register we want to read from
    printf("Write Error!\n");
    exit(1);
}

if (read(fd, buf, 24) != 24) { // Read data to buffer
    printf("Read Error!\n");
    exit(1);
}
else {
    AC1 = ((buf[0]<<8) + buf[1]);
    AC2 = ((buf[2]<<8) + buf[3]);
    AC3 = ((buf[4]<<8) + buf[5]);
    AC4 = ((buf[6]<<8) + buf[7]);
    AC5 = ((buf[8]<<8) + buf[9]);
    AC6 = ((buf[10]<<8) + buf[11]);
    B1 =  ((buf[12]<<8) + buf[13]);
    B2 =  ((buf[14]<<8) + buf[15]);
    MB = ((buf[16]<<8) + buf[17]);
    MC = ((buf[18]<<8) + buf[19]);
    MD = ((buf[20]<<8) + buf[21]);
}

// **** Read Temperature ****

buf[0] = 0xF4; //control register address
buf[1] = 0x2E; //start temperature measurement command

if ((write(fd, buf, 2)) != 2) { // Send command to register
    printf("Write Error!\n");
    exit(1);
}
```

```
usleep(15000); //wait for measurement to complete
buf[0] = 0xF6; //read measurement data from register 0xF6
if ((write(fd, buf, 1)) != 1) {
    printf("Write Error!\n");
    exit(1);
}

if (read(fd, buf, 2) != 2) { // Read  data to buffer
    printf("Read Error!\n");
    exit(1);
}
else {
    UT = ((buf[0]<<8) + buf[1]); //Uncorrected temperature
}

// **** Read Pressure ****

buf[0] = 0xF4; //control register address
buf[1] = 0x34+(oss<<6); //start pressure measurement (oss=3)
if ((write(fd, buf, 2)) != 2) {
    printf("Write Error!\n");
    exit(1);
}

usleep(50000); //wait measurement to complete

buf[0] = 0xF6; //Read measurement data from register 0xF6
if ((write(fd, buf, 1)) != 1) {
    printf("Write Error!\n");
    exit(1);
}

if (read(fd, buf, 3) != 3) { // Read back data into buf[]
    printf("Read Error!\n");
    exit(1);
}
```

```
    else {
          UP = ((buf[0]<<16) + (buf[1]<<8) + buf[2])>>(8-3); //Uncorrected
pressure
    }

    // **** Final Calculations For  correction ****
        //For Temperature
        long X1 = (UT-AC6) * AC5 / pow(2,15);
        long X2 = MC * pow(2,11) / (X1 + MD);
        long B5 = X1 + X2;
        long T = (B5 + 8) / pow(2,4);

        //For Pressure
        long B6 = B5 - 4000;
        X1 = (B2 * (B6* B6 / pow(2,12))) / pow(2,11);
        X2 = AC2 * B6 / pow(2,11);
        long X3 = X1 + X2;
        long B3 = (((AC1*4+X3) << oss)+2) / 4;
        X1 = AC3 * B6 / pow(2,13);
        X2 = (B1 * (B6 * B6 / pow(2,12))) / pow(2,16);
        X3 = ((X1 + X2) + 2) / pow(2,2);
        unsigned long B4 = AC4 * (unsigned long)(X3 + 32768)/pow(2,15);
        unsigned long B7 = ((unsigned long)UP - B3) * (50000 >> oss);
        long P;
        if (B7 < 0x80000000) {
            P = (B7 * 2) / B4;
        } else {
            P = (B7 / B4) * 2;
        }
        X1 = (P / pow(2,8)) * (P / pow(2,8));
        X1 = (X1 * 3038) / pow(2,16);
        X2 = (-7357 * P) / pow(2,16);
        P = P + (X1 + X2 + 3791) / pow(2,4);
    return P;
}
```

Kosketusnäytön lähdekoodi

```
//Version 5
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include "ULCD.h"

//Addresses for uLCD components
#define uTin 0x010F00
#define uHin 0x010F01
#define uBaro 0x010F02
#define uBaro2 0x010F05
#define uTout 0x010F03
#define uHout 0x010F04
#define uNegTin 0x011300
#define uNegTout 0x011301

int serial1 = -1;

int sendCommand(char buf[], int len){
    int retVal=-1;
    //Write buffer to serial port
                                        printf("%02x,%02x,%02x,%02x,%02x,
%02x\n",buf[0],buf[1],buf[2],buf[3],buf[4],buf[5]);
    if (serial1 != -1)
    {
        if (write(serial1, buf, len) < 0)
        {
            printf("Serial Port Write Error!\n");
        }
    }
    usleep(100000); //wait for response
```

```
    //Read response
    if (serial1 != -1)
    {
        unsigned char rxbuf[100];
        int rxlen = read(serial1, rxbuf, 1);
        if (rxlen < 0)
        {
            printf("Serial Port Read Error!\n");
        }
        else if (rxlen == 0)
        {
            printf("No Data!\n"); //No data received
        }
        else
        {
            if(rxbuf[0]==0x06){
                retVal=1;
                printf("ACK\n");
            }else{
                retVal=0;
                printf("NACK\n");
            }
        }
    }
    return retVal;
}

void uLCD_init() {

        serial1  =  open("/dev/ttyAMA0",  O_RDWR  |  O_NOCTTY  |  O_NDELAY);
//Open in non blocking read/write mode
    if (serial1 == -1)
    {
        printf("Error Opening Serial Port!");
    }
```

```
    struct termios options;
    tcgetattr(serial1, &options);
    options.c_cflag = B9600 | CS8 | CLOCAL | CREAD;
    options.c_iflag = IGNPAR;
    options.c_oflag = 0;
    options.c_lflag = 0;
    tcflush(serial1, TCIFLUSH);
    tcsetattr(serial1, TCSANOW, &options);
}

void uLCD_setValue(long addr, long val) {

    unsigned char addr1 = (addr>>16) & 0xFF;
    unsigned char addr2 = (addr>>8) & 0xFF;
    unsigned char addr3 = addr & 0xFF;
    unsigned char val1 = (val>>8) & 0xFF;
    unsigned char val2 = val & 0xFF;
    unsigned char cs = (addr1^addr2^addr3^val1^val2);

    int i=0;
    while(i!=1){
        char buf[] = {addr1,addr2,addr3,val1,val2,cs};
        i=sendCommand(buf,6);
    }
}

int uLCD_sync(){
    int retVal=-1;
    int i=0;
    unsigned char baf[] = {0xFF};
    for (i=0;i<10;i++){
        int a =sendCommand(baf,1);
        if(a==0){
            retVal=1;
            break;
        }
```

```
        usleep(10000);
    }
    return retVal;
}

void uLCD_close(){
    close(serial1);
}
```

Kalibrointiarvojen lukemisen lähdekoodi

```
#include <mysql/mysql.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "loadcal.h"

#define DBNAME "measurements"
#define DBUSER "logger"
#define DBPASS "mittaaja"

MYSQL *mysql1;
MYSQL_RES *res;
MYSQL_ROW row;

double mysql_read (char command[]) {
     double value;
     if(mysql1 != NULL){
          if (mysql_query(mysql1,command)){
               printf("Write Error!\n");
               return;
          }
          res = mysql_use_result(mysql1);
          while ((row = mysql_fetch_row(res)) != NULL)
               value = atof(row[0]);
          mysql_free_result(res);
          return value;
     }
}

double loadcalibration(int col){
     char command[50];
     char buf[10];
     switch (col) {
```

```
case 0:
     strcpy(buf,"S1Tg");
     break;
case 1:
     strcpy(buf,"S1To");
     break;
case 2:
     strcpy(buf,"S1Hg");
     break;
case 3:
     strcpy(buf,"S1Ho");
     break;
case 4:
     strcpy(buf,"S2Tg");
     break;
case 5:
     strcpy(buf,"S2To");
     break;
case 6:
     strcpy(buf,"S2Hg");
     break;
case 7:
     strcpy(buf,"S2Ho");
     break;
case 8:
     strcpy(buf,"S3Bg");
     break;
case 9:
     strcpy(buf,"S3Bo");
     break;

}

strcpy(command,"");
strcat(command,"SELECT ");
```

```
    strcat(command,buf);
    strcat(command," FROM calibrationdata");
    mysql_connect();
    double value;
    value = mysql_read(command);
    mysql_disconnect();
    return value;
}
```

Kalibrointiarvojen tallentamisen lähdekoodi

```
#include <mysql/mysql.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include "logger.h"
#include <string.h>

#define DBNAME "measurements"
#define DBUSER "logger"
#define DBPASS "mittaaja"

MYSQL *mysql1;

void mysql_connect (void) {
    mysql1 = mysql_init(NULL);
    if (mysql1!=NULL){
                                                if(mysql_real_connect(mysql1,
"localhost",DBUSER,DBPASS,DBNAME,0,NULL,0) != NULL){
            printf("Connected!\n");
        }else{
            printf("Error!!\n");
        }
    }else{
        printf("Error!\n");
    }
}

void mysql_disconnect (void){
    mysql_close(mysql1);
    printf("Disconnected!\n");
}
```

```
void mysql_write (char command[]) {
    if(mysql1 != NULL){
        if (mysql_query(mysql1,command)){
            printf("Write Error!\n");
            return;
        }
    }
}

void savemeasurement(char nm[], char tp[], double mval){
    char command[200];
    char buffer[16];
    strcpy(command,"");
                        strcat(command,"INSERT      INTO      measurementdata
(mDate,mTime,sensorNm,sensorTp,mValue) VALUES ");
    strcat(command,"(CURRENT_DATE(),NOW(),\'");
    strcat(command,nm);
    strcat(command,"\',\'");
    strcat(command,tp);
    strcat(command,"\',");
    sprintf(buffer,"%f",mval);
    strcat(command,buffer);
    strcat(command,")");
    mysql_connect();
    mysql_write(command);
    mysql_disconnect();
}
```