Antti Harhanen

Rengaslämpötilan mittausjärjestelmä

Mittausjärjestelmä ja sen tarkkuuden määritys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Auto- ja kuljetustekniikka

Insinöörityö

4.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Antti Harhanen Rengaslämpötilan mittausjärjestelmä 21 sivua + 7 liitettä 4.5.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Auto- ja kuljetustekniikka
Suuntautumisvaihtoehto	Autosähkötekniikka
Ohjaaja(t)	Autoelektroniikan lehtori Vesa Linja-aho

Insinöörityössä esitellään oppilastyönä valmistettu rengaslämpötilan mittausjärjestelmä.

Insinöörityön tavoitteena oli määrittää aiemmin oppilastyönä valmistetun mittausjärjestelmän tarkkuus, korjata sen toiminnassa ilmenneet puutteet sekä suunnitella mittausjärjestelmästä pienoisversio ohjelmineen Metropolia Ammattikorkeakoulun automaatiolaboratoriota varten.

Insinöörityö on rakenteeltaan kolmiosainen. Ensimmäisessä osassa esitellään insinöörityön aihe ja lähtökohdat sekä I2C-väylätekniikan perusteet sekä käydään lävitse aiheeseen liittyvää termistöä. Toisessa osassa esitellään rengaslämpötilan mittausjärjestelmän tekninen rakenne sen tärkeimpien kokonaisuuksien osalta. Kolmannessa osassa tutkitaan rengaslämpötilan mittausjärjestelmän mittaustarkkuutta kaupallisen infrapunalämpömittarin avulla.

Insinöörityön tuloksena saatiin suunniteltua valmistuskustannuksiltaan edullinen rengaslämpötilan mittausjärjestelmä, jonka mittaustarkkuus tunnetaan ja jota voidaan käyttää Metropolia Ammattikorkeakoulun tulevissa projekteissa mittavälineenä. Tämä insinöörityö sisältää myös tarvittavan dokumentoinnin mittausjärjestelmän jatkokehitykselle.

Avainsanat	infrapuna, lämpötila, mittaus, rengas

Author(s) Title Number of Pages Date	Antti Harhanen Tire Temperature Monitoring System 21 pages + 7 appendices 4 May 2014
Degree	Bachelor of Engineering
Degree Programme	Automotive and Transport Engineering
Specialisation option	Automotive Electronics Engineering
Instructor(s)	Vesa Linja-aho, Senior Lecturer in Automotive Electronics

This thesis introduces an infra red tire temperature monitoring system designed as a part of Helsinki Metropolia University of Applied Sciences' CDIO project. The goal of this thesis was to fix the problems that occurred during the original tire temperature monitoring system project and to measure the accuracy of the system. As a part of the thesis a smaller version of the system was designed for the use of the Automation Laboratories of Helsinki Metropolia University.

The thesis consists of three parts. The first part introduces the I2C Bus technology and its terminology. The second part displays the structure of the Tire Temperature Monitoring System and its main components. In the third part the accuracy of the system is tested.

As a result of this thesis an affordable tire temperature monitoring system was designed for the use of Helsinki Metropolia University. The system can be used in future projects and has potential for further development. This thesis includes also the full documentation of the Tire Temperature Monitoring System.

Keywords	infra red, temperature, measuring, tire

Sisällys

3. /Wire/keywords.txt
4. /Wire/utility/twi.c
5. /Wire/utility/twi.h
6. /LCD_C0220BiZ/LCD_C0220BiZ.h
7. /LCD_C0220BiZ/LCD_C0220BiZ.cpp
8. /LCD_C0220BiZ/keywords.txt
9. /LCD_C0220BiZ/lcd.h
10. /LCD_C0220BiZ/ST7036.h
11. /LCD_C0220BiZ/ST7036.cpp
12. /I2Cmaster/i2cmaster.h
13. /I2Cmaster/keywords.txt
14. /I2Cmaster/twimaster.cpp

Liite 7. Mittauspöytäkirja

Lyhenteitä ja käsitteitä

Lähetin	laite joka lähettää dataa väylään
Vastaanotin	laite joka vastaanottaa dataa väylältä
Isäntä	tiedonsiirron aloittava laite, luo kellosignaalit ja tekee tiedonsiirron lopetukset
Orja	laite jonka kanssa isäntä vaihtaa tietoja
Multi-master	järjestelmä sallii useamman kuin yhden isännän ilman että siirrettävä data korruptoituu tai väyläviestintä muuten häiriintyy. Arbitraatio varmistaa, että vain yksi isäntä lähettää kerralla.
Synkronisaatio	väylään kytkettyjen isäntien kellosignaalien tahdistaminen, tarvitaan vain useampia isäntiä sisältävissä järjestelmissä.
Drain	avoimen kollektorin piiri
Arbitraatio	lähetysjärjestyksen määrittäminen isäntien kesken. Koska I2C-väylä on drain-tyyppinen (yhden laitteen kytkiessä linjan maahan, pysyy sen jännite alhaalla, riippumatta muista laitteista), ykköstä ja nollaa lähetettäessä, nollan ensin lähettänyt isäntä voittaa arbitraation. Tarvitaan vain useampia mastereita sisältävissä järjestelmissä
Vdd	Voltage drain drain, tässä yhteydessä järjestelmän käyttöjännite
I_L	Input Refence Low = looginen nolla
I_H	Input Reference High = looginen yksi
SDA	Signal Data = datasignaali
SCL	Signal Clock = kellosignaali
bitti	binääriluku, 0 tai 1
tavu	kahdeksan bitin joukko
R/W	Read/Write- eli luku/kirjoitus-bitti
ACK	Acknowledge-bitti seuraa jokaista lähetettyä 8 bitin tavua. Sen avulla orja ilmoittaa isännälle olevansa valmis vapauttamaan väylän uutta tiedonsiirtoa varten. Isäntä vapauttaa SDA-väylän jonka jälkeen orjan on mahdollista laskea SDA-jännite nollaksi. Tämä 9:nen

kellosignaalipulssin aikana tapahtuva jännitteenmuutos I_H:sta I_L:n on ACK-bitti.

NACK — Not Acknowledge-bitti syntyy kun SDA-väylän jännite pysyy ylhäällä 9:nen kellosignaalipulssin ajan. Kertoo isännälle, että:

1. Väylällä ei ole laitetta kutsutussa osoitteessa.
2. Vastaanotin ei kykene vastaanottamaan tai lähettämään johtuen siitä että se suorittaa tehtävää.
3. Tiedonsiirron aikana lähetetty NACK, vastaanotin vastaanottaa dataa tai komentoja joita se ei tunne tai ymmärrä.
4. Tiedonsiirron aikana lähetetty NACK, vastaanotin ei kykene vastaanottamaan enempää tavuja dataa.
5. Isännän on lähetettävä tiedonsiirron päättymistä merkitsevä signaali.

Nackin jälkeen isäntä voi lähettää joko Stop-komennon tai Repeated Start -komennon riippuen jatketaanko tiedonsiirtoa laitteiden välillä vai ei.

CRC — Circular Redundancy Correction -tarkistustavu lasketaan viestin bittijonosta. Tämä mahdollistaa viestien oikeellisuuden tarkistuksen. Mikäli orjalaitteen saaman viestin sisällöstä laskema CRC-bitti ei vastaa viestin osana vastaanotettua CRC-bittiä on vastaanotetussa viestissä virhe.

Lyhenteet ja käsitteet I2C-väylän spesifikaation ja käyttöohjeen UM10204:n [2, s.6-32] mukaisesti.

1 Johdanto

Tämä opinnäytetyö perustuu Metropolia Ammattikorkeakoulun CDIO-projektin (Conceive, Design, Implement, Operate) osana suunniteltuun rengaslämpötilan mittausjärjestelmään. Järjestelmän tilasi Metropolian tuntiopettaja Pasi Oikarinen. Toinen projektia ohjanneista oli autoelektroniikan lehtori Vesa Linja-aho. Projektin tavoitteena oli suunnitella ja valmistaa edullinen rengaslämpötilan mittausjärjestelmä, joka kykenee mittaamaan auton kaikkien renkaiden pintalämpötilan kolmesta pisteestä ±1 °C:n tarkkuudella.

Opinnäytetyön tekijä oli osana opiskelijaprojektia ohjelmoinut Arduino Uno -mikrokontrollerin toimimaan infrapunatunnistimia käyttävän rengaslämpötilan mittausjärjestelmän ohjainlaitteena. Järjestelmä hyödyntää I2C-väylätekniikkaa.
Rengaslämpötilan mittausjärjestelmän mittaustarkkuutta ei ollut määritetty eikä sitä ollut kalibroitu. Lisäksi mittausjärjestelmän virtapiiri ei kyennyt toimimaan suunnitelluilla neljällä mitta-anturisarjalla. Järjestelmään oli näin ollen myös suunniteltava väylänjakaja, jotta kaikki mitta-anturisarjat saataisiin toimimaan samanaikaisesti. Opinnäytetyön tavoitteena oli korjata nämä mittausjärjestelmässä havaitut puutteet sekä määrittää sen mittaustarkkuus. Lisäksi opinnäytetyössä suunniteltiin mittausjärjestelmästä pienoisversio (liite 3) Metropolia Ammattikorkeakoulun automaatiolaboratoriota varten.

Tämän opinnäytetyön teoriaosassa tarkastellaan I2C-väylätekniikkaa ja sen toiminnan perusteita. Lisäksi esitellään Arduino Uno -mikrokontrolleri. Mittausjärjestelmän mittaustarkkuus on määritetty vertaamalla sen mittaustuloksia teollisesti valmistetun infrapunalämpömittarin kanssa. Mittausjärjestelmän päivitystaajuus arvioitiin laskennallisesti.

2 Yleistä I2C-väylätekniikasta

I2C on yksi pienelektroniikassa yleisesti käytössä olevista väylätekniikoista. I2C-väylän tiedonsiirtonopeus vaihtelee 100 kb/s ja 5 Mb/s välillä riippuen väylän ohjainlaitteen kehitysasteesta. I2C-väylä on avoimen kollektorin rakenne, eli se vaatii ylösvetovastuksen jännitteen vaihtelujen aikaansaamiseksi

Järjestelmässä voi olla useita isäntälaitteita ja orjalaitteita. Näiden määrää rajaa vain osoitteiden 7-bittisyys, minkä vuoksi käytettävissä olevia väyläosoitteita on ainoastaan 127 kappaletta. On olemassa myös 10-bittisiä I2C-laitteita, mutta ne ovat toistaiseksi melko harvinaisia, sillä tarve niiden olemassaololle on syntynyt vasta viime vuosien kuluessa. Vain yksi laite kerrallaan voi lähettaa tietoa väylässä ja väylään kirjoittaminen tapahtuu muuttamalla SDA- ja SCL-linjojen jännitettä. Väylän maksimikapasitanssin ei tulisi olla yli 400 pF, jotta tiedonsiirto toimii normaalisti. Tämä rajoittaa väylän pituuden muutamaan metriin. Järjestelmää koottaessa todettiin tämän merkitsevän maksimissaan kahta kuuden metrin mittaista johdinta kolmen anturin anturisarjoilla.

2.1 SDA- ja SCL-logiikkatasot

Nollabitin ja bitin yksi tasot eivät ole I2C-väylätekniikassa kiinteät, vaan ne on määritetty Vdd:n mukaan. Nollabitiksi tulkitaan kaikki alle 30 % Vdd:stä olevat signaalit kun taas bitiksi yksi kaikki yli 70 % Vdd:stä olevat signaalit. Tähän on päädytty jotta väyläjärjestelmä tukisi mahdollisimman moneen eri tekniikkaan (CMOSiin, NMOSiin ja bipolaariseen) perustuvia laitteita. 0-bitti on dominantti avoimen kollektorin rakenteen vuoksi.

On olemassa joitakin laitteita joille on määritetty kinteät I_L = 1,5V ja I_H = 3,0V. Nämä niin sanotut Legacy-laitteet ovat kuitenkin poistumassa käytöstä ja harvinaisia.

2.2 Tiedonsiirto I2C-väylässä

Kaikkien I2C-väylään lähetettävien tavujen tulee olla 8-bittisiä. Järjestelmään on kuitenkin mahdollista kytkeä myös muita, esimerkiksi 4- ja 16-bittisiä laitteita. Tällöin luettu ja kirjoitettu tieto tulee jakaa osiin ja tallentaa välimuistiin ennen datan varsinaista käsittelyä.

I2C:hen pohjautuvissa järjestelmissä isäntä aloittaa tiedonsiirron lähettämällä START-komennon väylään. START-komento suoritetaan muuttamalla SDA-johtimen jännitearvo I_H:sta I_L:ksi SCL:n ollessa I_H:ssa. Vastaavasti isännän tiedonsiirron päättävä STOP-komento suoritetaan muuttamalla SDA I_L:sta I_H:ksi SCL:n ollessa I_H:ssa (kuva 1). START-komentoa seuraa kutsuttavan orjan osoite ja R/W-bitti riippuen siitä

odotetaanko kirjoitetaanko johonkin orjan sisäisistä osoitteista vai luetaanko jokin tietty orjan sisäinen osoite.

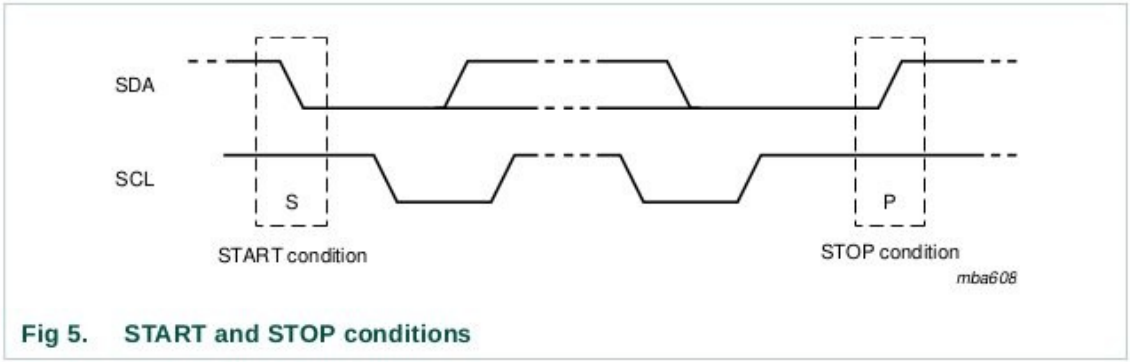


Fig 5. START and STOP conditions

Kuva 1. I2C-väylän Start ja Stop komennot [4, s. 9].

3 Rengaslämpötilan mittausjärjestelmä

Rengaslämpötilan mittausjärjestelmä on suunniteltu toimimaan 12 MLX90614-lämpötilatunnistimella. Tunnistimet on jaettu neljään sarjaan, niin että kutakin ajoneuvon pyörää kohden on kolme tunnistinta. Anturisarjat toimivat kukin omissa fyysisesti erillisissä väylissään, jotka luodaan väylänjakajalla. Tämä mahdollistaa pidempien johtimien käytön. Rengaslämpötilan mittausjärjestelmä tulostaa kunkin anturisarjan keskimääräisen lämpötilan Arduinon PWMOut-pinneihin pulssileveysmoduloituna 5 V:n datasignaalina.

3.1 Mikrokontrolleri

Arduino Uno -mikrokontrolleri perustuu ATmega328-piirille ja sen käyttöjännite on 5 V. Mikrokontrollerin virtalähteen jännite voi olla 6–20 V sen sisään rakennetun regulaattorin ansiosta. Mikrokontrollerin prosessorin kellotaajuus on 16 MHz ja siinä on sisään rakennettuna 32 kB Flash-muistia, 2 kB SRAM-muistia sekä 1 kB EEPROM-muistia.

Piirilevyltä löytyy 14 digitaalista I/O-pinniä sekä 6 analogista sisääntuloa. Osan digitaalisista pinneistä ulostulojännite on pulssileveysmoduloitua kanttiaaltoa. Analogiset pinnit A4 ja A5 ovat I2C-väylän SDA- ja SCL-pinnit ja niissä on tiedonsiirron mahdollistamista varten sisään rakennetut ylösvetovastukset jotka voi kytkeä päälle ja

pois ohjelmallisesti. Virran ulostuloja on kaksi 5 V ja 3,3 V. Digitaalisten pinnien ulostulojännite on 5 V.

Arduino Unon ohjelmointia varten on mahdollista ladata ilmainen käyttöliittymä, Arduino IDE, joka toimii yleisimmillä käyttöjärjestelmillä (Linux, Windows ja Mac). Ohjelmointikieli perustuu C++:aan.

3.2 Väylänjakaja

Johtimien pituuden maksimoimiseksi ja neljän saman kiinteän väyläosoitteen omaavan LCD-näytön käytön mahdollistamiseksi tulee fyysinen väyläyhteys jakaa neljään osaan. Tätä tarkoitusta varten suunniteltiin erityinen väylänjakaja.

Renkaan lämpötilan mittausjärjestelmää varten suunniteltiin kuuteen npn-transistoriin perustuva väylänjakaja. Transistoriksi valikoitui 2N2222a yleisyytensä ja kestävyytensä vuoksi. Väylänjakaja toimii seuraavasti: transistorit avataan täysin johtavaan tilaan johtamalla niiden kannalle 5 V:n jännite Arduinon digitaalisilta ulostuloilta. Avaamalla ja sulkemalla transistoreja voidaan päättää mikä neljästä väylän osasta on käytössä. Transistorien käyttö kuitenkin rajoittaa järjestelmän nopeutta, koska 2n2222a:n täysi aukeaminen vaatii noin 24 ms [6, s. 2] jolloin neljän anturisarjan tietojen kirjoittaminen neljälle erilliselle näytölle vie noin 0,1 sekuntia. Tästä syystä järjestelmän päivitystaajuus jää korkeimmillaankin tasolle 10 näytettä/sekunti. Rengaslämpötilaa mitattaessa tämä on kuitenkin riittävä näytteenottotaajuus.

Transistorien käyttö väylän jakamiseen ei ole ihanteellinen ratkaisu käytettäessä I2C-väyläprotokollaa. Väyläviestinnän rakenne perustuu SDA- ja SCL-signaalien tasomuutoksiin, joten tilanne, jossa molemmat väylät siirtyvät yhtä aikaa I_L:sta I_H:hin, johtaa väyläviestinnän katkeamiseen. Tästä johtuen väylien fyysinen aktivointi tapahtuukin mittausjärjestelmässä nostamalla SCL-signaali I_H-tasoon ennen SDA-signaalia. Väyläviestinnässä tämä vastaa STOP-komentoa. On myös otettava huomioon, että transistorien johtavaksi saattamisen vaatimat jännitteenjakosillat laskevat väylään lähetetyn signaalin jännitetasoa noin 0,4–0,8 volttia. 5 V:n jännitettä käyttävässä I2C-väylässä I_H:n jännite saa vaihdella 3,5–5 V, joten voidaan todeta, että vaimeneminen on hyväksyttävällä tasolla.

3.3 Virtapiiri

I2C-väylätekniikan ansiosta järjestelmän vaatimien komponenttien lukumäärä on melko pieni. Rengaslämpötilan mittausjärjestelmän virtapiiri muodostuu väylänjakajasta, LCD-näyttöjen kirkkautta muuttavasta säätövastuksesta, painonapin kytkennästä, infrapuna-anturien häiriönpoistokondensaatoreista ja I2C-väylän vaatimista ylösvetovastuksista. Ylösvetovastusten tehtävä on muodostaa I2C-väylään havaittavissa oleva jännite. Niiden mitoittamiseksi on väyläprotokollassa [2, s. 55] esitetty menetelmä:

Määritetään järjestelmän kapasitanssin aiheuttama viive jännitteen noustessa arvosta V_{IL} arvoon V_{IH}.

$$V_{IL} = 0{,}3 * V_{DD} \quad (1),$$

$$V_{IH} = 0{,}7 * V_{DD} \quad (2),$$

tiedetään että:

$$V(t_1) = 0{,}3 * V_{DD} = V_{DD}\left(1 - e^{(-t_1/RC)}\right) \quad (3).$$

Ratkaistaan t_1,

$$t_1 = 0{,}3566749 * RC \quad (4).$$

$$V(t_2) = 0{,}7 * V_{DD} = V_{DD}\left(1 - e^{(-t_2/RC)}\right) \quad (5)$$

ratkaistaan t_2,

$$t_2 = 1{,}2039729 * RC \quad (5).$$

Tästä seuraa, että mittausjärjestelmän aikavakio

$$T = t_2 - t_1 = 0{,}8473 * RC \quad (6)$$

$$R_p(max) = T / 0{,}8473 * C \quad (7)$$

missä R on ylösvetovastuksen arvo ja C järjestelmän kokonaiskapasitanssi.

Lämpötilan mittausjärjestelmässä käytetty I2C-väylä on Standard-tasoa ja sen suurin sallittu aikavakio on 1000 ns.

Näiden laskelmien ja protokollan $R_{p(max)}$- ja $R_{p(min)}$- taulukoiden [2, s. 55] perusteella ylösvetovastuksiksi valikoituivat 4,7 kΩ vastukset, koska järjestelmän tarkkaa kapasitanssia ei tunneta, mutta sen voidaan olettaa olevan suuri johtimien pituuden vuoksi.

3.4 Emissiivisyys ja sen määrittäminen

Emissiivisyys on suhdeluku, joka kuvaa aineen pinnan kykyä luovuttaa vastaanottamaansa säteilyenergiaa. Aineesta palaavan säteilyn intensiteettiä verrataan teoreettisen tilanteeseen jossa aine ei säteile ollenkaan takaisin, eli on ”täysin musta” [5, s. 461–463].

Aineiden kyky sitoa säteilyä vaihtelee suuresti niiden ominaisuuksien mukaan. Tilanteessa jossa aineen sisäinen lämpötila on ympäristön vallitsevaa lämpötilaa korkeampi, se säteilee enemmän kuin sitoo säteilyä. Myös säteilyn aallonpituudella on merkitystä tutkittaessa emissiivisyyttä, sillä aineen emissiivisyys muuttuu mitattavan säteilyn aallonpituuden mukaan. Tästä syystä onkin aiheellista huomauttaa että tässä opinnäytetyössä emissiivisyydestä puhuttaessa viitataan aineen emissiivisyyteen infrapuna-aallonpituudella.

Tuntemattoman emissiivisyyden määrittämisessä voidaan käyttää hyödyksi pinnoitusmateriaalia jonka emissiivisyys tunnetaan hyvin, esimerkiksi maalarinteippiä (E = 0,95). Riittävän suuri alue mitattavasta kohteen pinnasta käsitellään tunnetun emissiivisyyden omaavalla aineella jolloin on mahdollista verrata kahden eri pinnan välistä lämpötilaeroa. Tunnetun emissiivisyyden omaavan pinnan läheisyydessä voidaan käyttää mittaustarkkuuden parantamiseksi kontaktimittausta, esimerkiksi lämpöparia. Mitataan mitattavan kappaleen pintalämpötiloja tunnetun emissiivisyyden alueelta ja pinnoittamattomalta alueelta. Mitattujen lämpötilojen erosta voi päätellä mittaako tuntemattoman pinnan lämpötilaa liian suurella (näyttää liian korkeita

lämpötiloja) vai liian pienellä emissiivisyysarvolla (mitatut lämpötilat liian alhaisia). Parhaan tuloksen saa erilaisilla haarukoimismenetelmillä. Kun infrapunalla mitatun lämpötilan ja kontaktimittauksen lämpötilan ero on mahdollisimman pieni ollaan lähellä aineen emissiivisyysarvoa.

Mitattavan aineen emissiivisyyden tarkkaa määritystä ei ollut mahdollista suorittaa osana tätä opinnäytetyötä sen vaatimien erikoislaitteiden puuttumisesta johtuen.

3.5 Melexis MLX 90614 -infrapunatunnistin

Opinnäytetyössä käytettiin MLX 90614-infrapunatunnistimia. Nämä tunnistimet pystyvät lähettämään mittaamiaan lämpötiloja digitalisoituna tietona muun muassa I2C-väylän välityksellä. Merkittävin etu kyseisten infrapunatunnistimien käytössä niiden edullisuuden lisäksi on se, että niiden väyläosoitteet ja emissiivisyysarvot ovat ohjelmallisesti muutettavissa. MLX 90614:n kotelointi on tyyppiä TO-39, eli ne ovat myös suhteellisen pienikokoisia. Tunnistimen halkaisija on 12 mm.

4 Järjestelmän mittaustarkkuuden määrittäminen ja kalibrointi

Koska mittajärjestelmän infrapuna-anturit on mahdollista kalibroida eri materiaaleille ohjelmallisesti on tätä ominaisuutta hyödynnetty osana mittajärjestelmään ohjelmaa, jolloin toiminto ei vaadi erillistä ohjelmaa vaan ainoastaan yhden muuttujan muuttamisen. Tämän jälkeen ohjelma syöttää halutun emissiivisyysarvon kaikille antureille yhden kerran aina ohjelman käynnistyessä. Kalibraatioarvojen muokkaamiseksi vaaditaan tietokone johon on asennettu Arduino-ohjelmisto.

4.1 Mittausmenetelmä

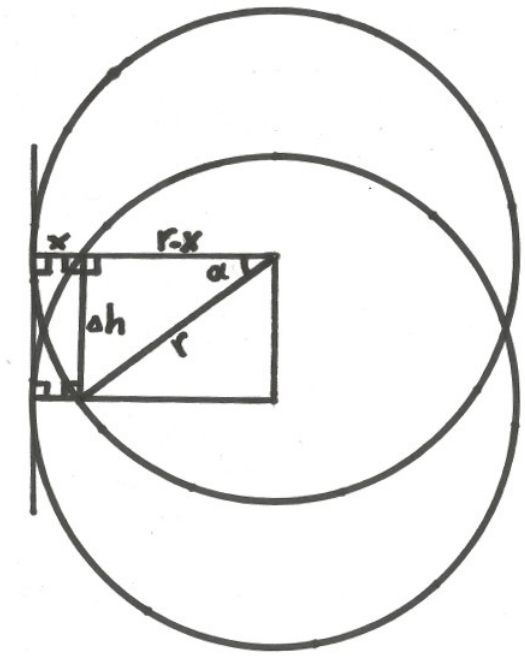
Suoritetuissa mittauksissa käytettiin mittausvälineinä rullamittanauhaa (tarkkuus ±0,1 mm) ja Milwaukee Laser TEMP-GUN™ M12™ -infrapunalämpömittaria. Anturiasetelma kytkettiin statiiviin siten että sen etäisyyttä mitattavaan kohteeseen voitiin muuttaa (kuva 2). TEMP-GUNia käyttäen pyrittiin määrittämään mahdollisimman tarkasti renkaan pintalämpötila. Infrapunasäteilyä hyödyntävän lämpötilamittauksen mittaustarkkuus paranee mikäli mitattavan kohteen lämpötila on nostettu ensin korkeammaksi kuin ympäristön vallitseva lämpötila. Lisäksi mittausten tarkkuuden

parantamiseksi mitattavan kohteen, tässä tapauksessa autonrenkaan pinta päällystettiin tunnetun emissiivisyyden omaavalla aineella eli maalarinteipillä.

Kuva 2. Mittausjärjestelyt.

Rengasta lämmitettiin lämpölähteellä renkaan sisäpuolelle puhaltamalla noin 60°C lämpötilaan saakka. Renkaan lämmityksen aikana valokuvattiin mittausjärjestelmän ja TEMP-GUNin näyttämät lämpötilat ja ne kirjattiin näiden valokuvattujen tulosten perusteella yhden anturin osalta mittauspöytäkirjaan (liite 7).

Mittaukset toistettiin 5 cm:n, 10 cm:n ja 15 cm:n etäisyydeltä niin, että TEMP-GUNin etäisyys pysyi kaikkien mittausten ajan samana tulosten vertailtavuuden saavuttamiseksi. Tämä on tarpeen koska anturin ollessa kiinnitettynä auton runkoon, anturin ja renkaan kulutuspinnan välinen etäisyys muuttuu jatkuvasti jousituksen sisään- ja ulosjoustojen aikana. Kuvassa 3 on esitetty trigonometriset perusteet kaavalle 13 jonka avulla on mahdollista renkaan pyöreän muodon vaikutus mittausetäisyyteen , kun mittausjärjestelmän tunnistimet pysyvät paikoillaan ja rengas liikkuu y-akselin suuntaisesti joustojen mukana.



Kuva 3. Jouston pituuden vaikutus mittausetäisyyteen, trigonometriset perusteet.

Kuvan 3 perusteella voidaan todeta, että

$$\sin\alpha = \frac{\Delta h}{r} \quad (8)$$

mistä seuraa

$$\alpha = \arcsin\frac{\Delta h}{r} \quad (9)$$

Saadun kulman α avulla on mahdollista määrittää mittausetäisyyden muutos:

$$\cos\alpha = \frac{r-x}{r} \quad (10)$$

mistä ratkaistaan x

$$r - x = r\cos\alpha \quad (11),$$

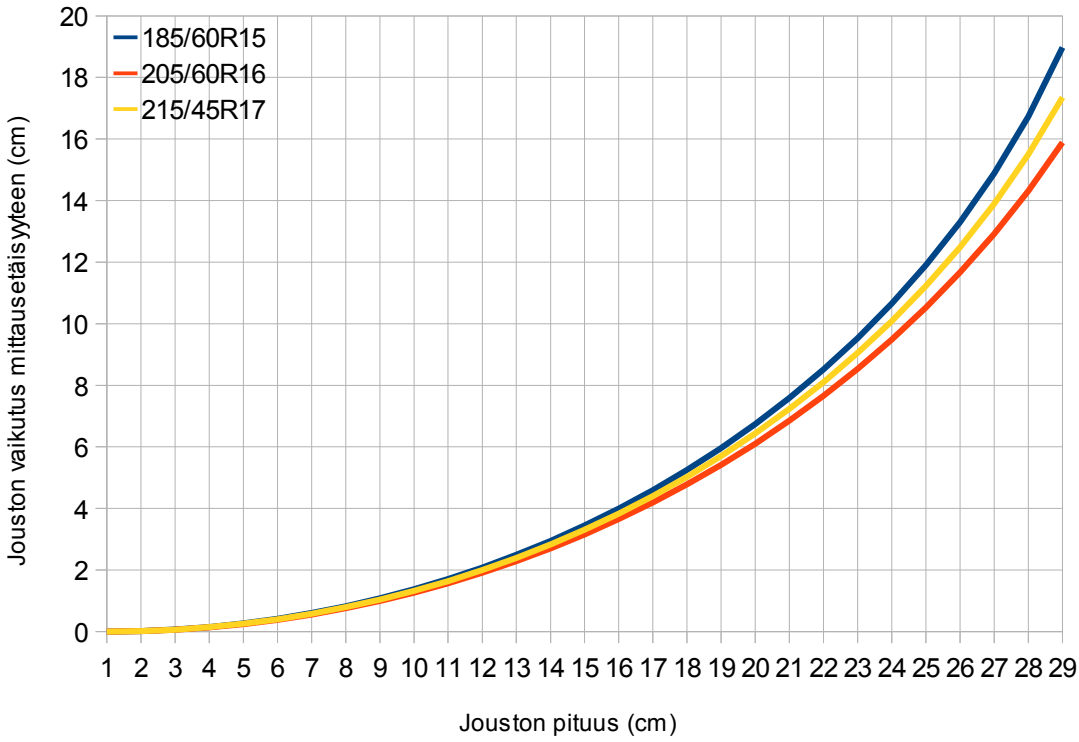
$$x = r - r\cos\alpha \quad (12).$$

Koska kulma α tunnetaan, seuraa:

$$x = r - r\cos\left(\arcsin\frac{\Delta h}{r}\right) \quad (13).$$

Kaavaa 13 käyttäen on mahdollista todeta, että esimerkiksi 185/60R15-koon renkaan kohdalla tämä merkitsee 12,5 cm:n sisäänjoustossa noin 1 cm:n muutosta mittausetäisyydessä, mikäli anturi on sijoitettu nollatilanteessa ajoneuvon akselin tasalle. Kuvassa 4 on esitetty graafisesti tämä jouston pituuden ja mittausetäisyyden välinen riippuvuus kolmelle vuonna 2014 yleiselle rengaskoolle. Kaavan 13 avulla

muodostettua kuvaajaa voidaan hyödyntää tehokkaasti myös tilanteissa, joissa mittauspiste ei ole pyörän navan tasolla. Tämä onnistuu siirtämällä y-akselia oikealle jolloin jouston pituuden aiheuttama muutos mittausetäisyyteen on sisäänjoustossa suurempi kuin ulosjoustossa. Kuvaajasta voidaan myös todeta, että mitä lähemmäs pyörännavan tasoa anturisarja on sijoitettavissa, sitä vähemmän mittausetäisyyden muutos vaikuttaa mittaustuloksiin.



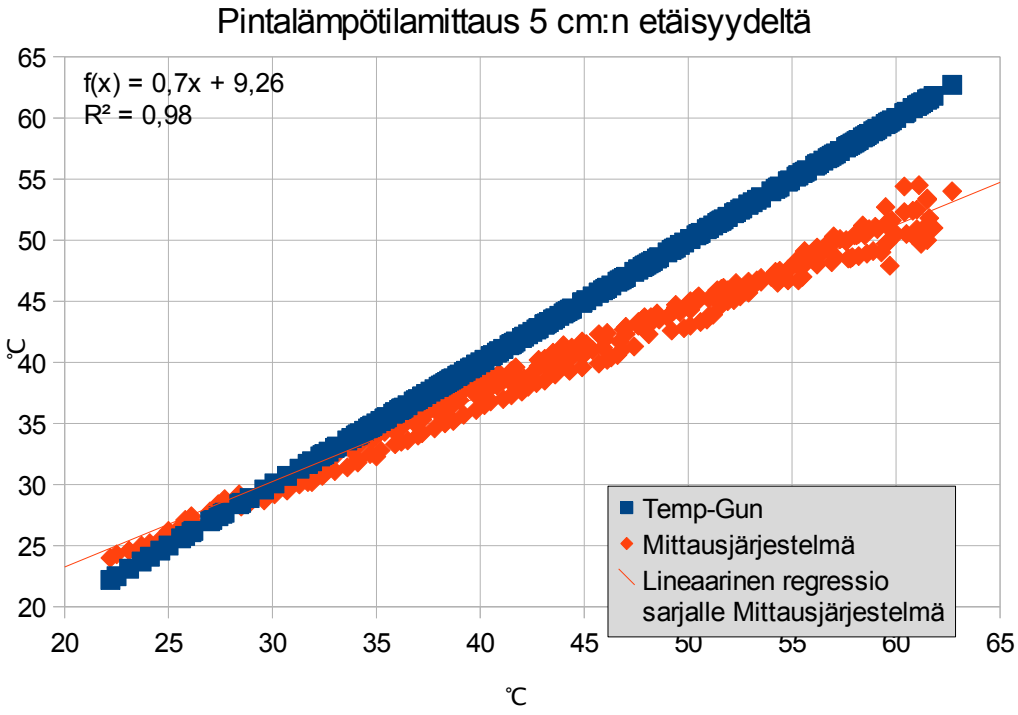
Kuva 4. Jouston pituuden vaikutus mittausetäisyyteen.

Mikäli mitta-anturien sijoitus renkaan navan tasolle jousituksen ollessa perusasennossa on mahdotonta, pystyy kuvaajasta päättelemään myös jouston vaikutuksen tässä tapauksessa. Koska anturien sijoittelumahdollisuudet vaihtelevat ajoneuvoittain kuten myös jouston liikerata, on todettava ettei kyseinen riippuvuuslaskelma ole suoraan sovellettavissa kaikkiin tapauksiin. Esimerkiksi yhdysheilurijousituksella varustetun ajoneuvon jouston liikerata on ympyrämäinen suhteessa kiinteästi sijoitettuun mitta-anturiin, kun taas erillisjousitetussa ajoneuvossa ylä- ja alatuennan rakenne määrittelee suoraviivaisen liikkeen kulman suhteessa ajoradan pintaan.

4.2 Mittaustulokset

Mittaustulosten laadun määrittämisessä käytettiin hyödyksi PNS- eli pienimmän neliösumman keinoa. PNS-keino on yleisesti käytössä olevan lineaaristen mallien estimointimenetelmä. Mallin tarkkuutta arvioidaan regressiosuoran selitysasteen (R^2-luvun) perusteella [3, s. 291]. Selitysaste havainnollistaa sitä, miten hyvin saadut mittaustulokset asettuvat regressiosuoran suhteen. Selitysaste vaihtelee välillä 0-1, missä arvo 1 merkitsee tilannetta jossa kaikki mittaustulokset asettuvat regressiosuoralle.

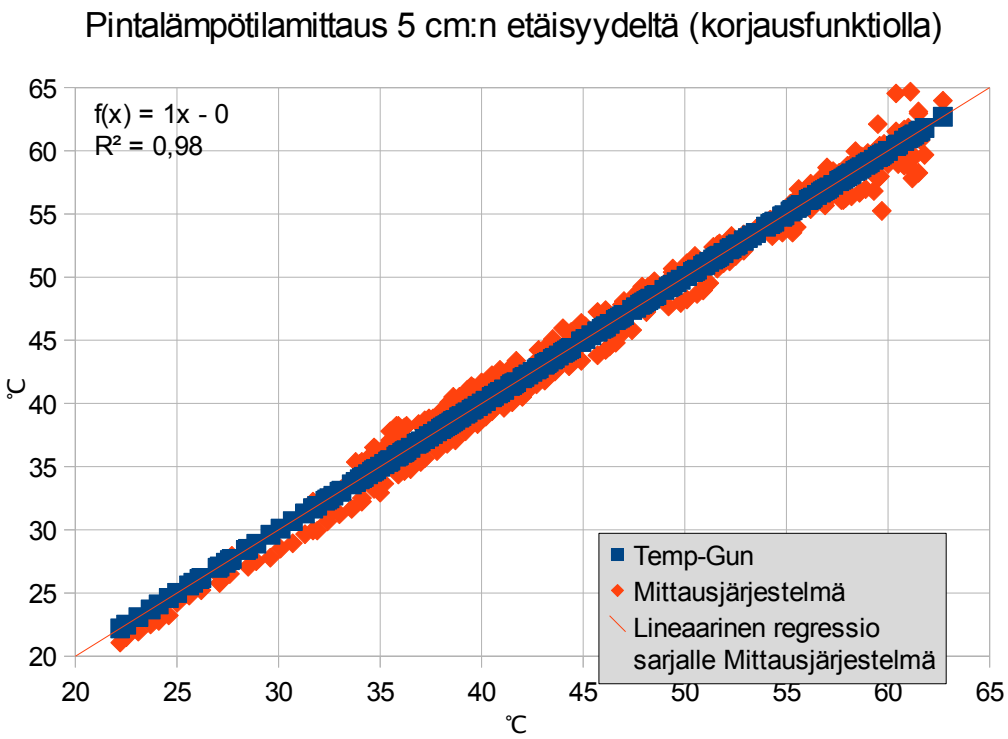
Kuvassa 5 on graafisesti esitettynä mittausjärjestelmän yhden anturin mittaustulokset. Temp-Gunilla mitatut tulokset sovitettiin avulla kuvaajaan niin että ne muodostavan suoran joka kulkee koordinaatiston origon kautta. Mittauspisteistä muodostetun regressiosuoran funktio on f(x). R^2-arvosta voidaan päätellä että mittausjärjestelmän mittaustarkkuuden olevan hyvä kun mittausetäisyys on 5 cm.



Kuva 5. Pintalämpötilamittaus 5 cm:n etäisyydeltä.

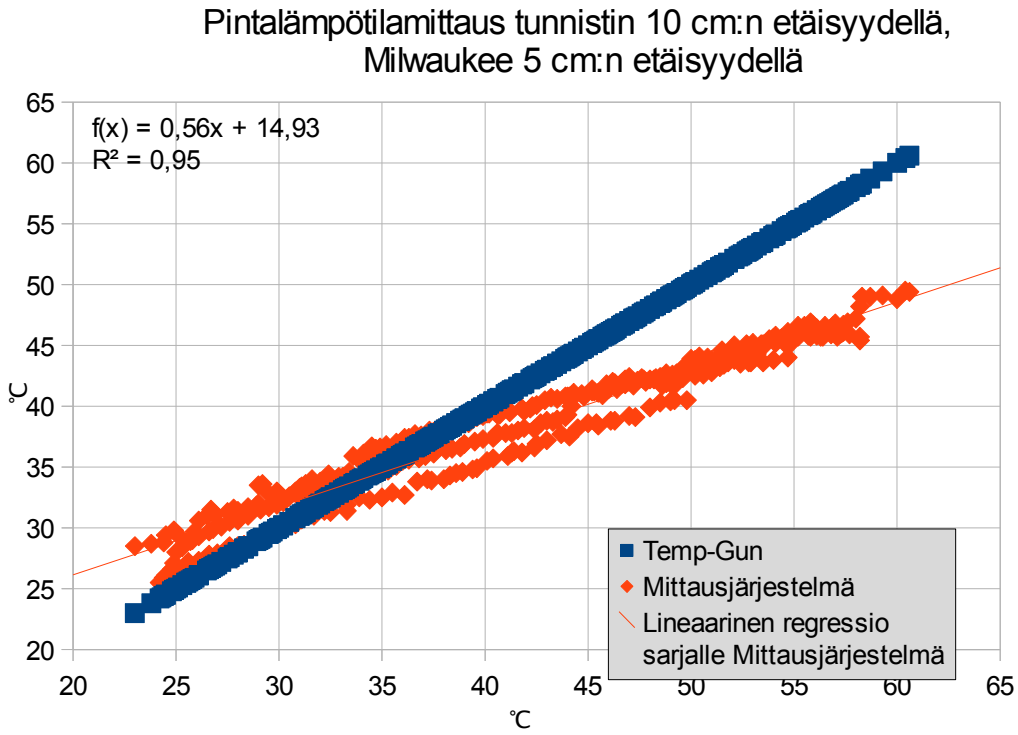
Vaikka saatujen mittaustulosten lineaarinen riippuvuus on selitysasteen perusteella hyvä, pystyy mittaustulosten kuvaajasta päättelemään, että rengaslämpötilan mittausjärjestelmän mittaamissa arvoissa on systemaattinen virhe. Tämän virheen

korjaaminen onnistuu ensimmäisen asteen polynomifunktiolla, mikäli ilmiötä voidaan pitää lineaarisena. Korjausfunktion kulmakerroin on mittausjärjestelmän mittaustulosten lineaarisen regression funktion kulmakertoimen käänteisluku (tavoitteena on saada sovitettua lineaarisen regression kuvaaja kulkemaan yhdensuuntaisesti Temp-Gunin mittaustulosten kuvaajan kanssa). Kun Temp-Gunin kuvaaja ja mittausjärjestelmän mittaustulosten lineaarisen regression funktion kuvaaja ovat yhdensuuntaiset, sovitetaan lineaarisen regression kuvaaja kulkemaan kuvaajan origon kautta lisäämällä kuvaajien yhdensuuntaistamisen tuottaman lukujoukon lineaarisen regression funktion vakion vastaluku korjausfunktion vakioksi (kuva 6) .



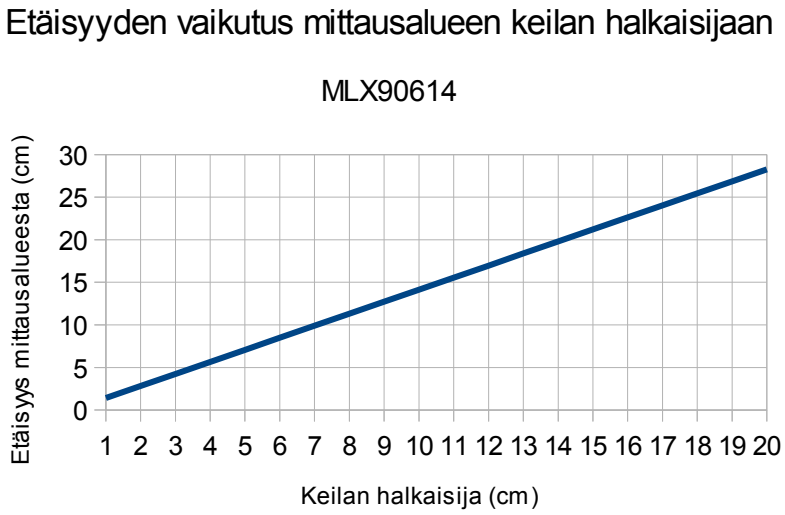
Kuva 6. Pintalämpötilamittaus 5 cm:n etäisyydeltä, mittaustulokset sovitettu kuvaajaan korjausfunktion avulla.

Mittaukset toistettiin myös niin, että Temp-Gunilla mitattiin pintalämpötilaa 5 cm:n etäisyydellä, kun mittausjärjestelmän tunnistin siirrettiin 10 cm: etäisyydelle kohteesta (kuva 7). Mittaustuloksista ja niiden lineaarisen regression funktiosta on havaittavissa, että mittaustulosten selitysaste huononee ja mitatut lämpötilat ovat systemaattisesti matalampia kuin lyhyemmän etäisyyden päästä saadut mittaustulokset. Tämä voi osaltaan johtua myös MLX90614-infrapunatunnistimen mittausalueen leveästä (90°) keilasta (kuva 8).



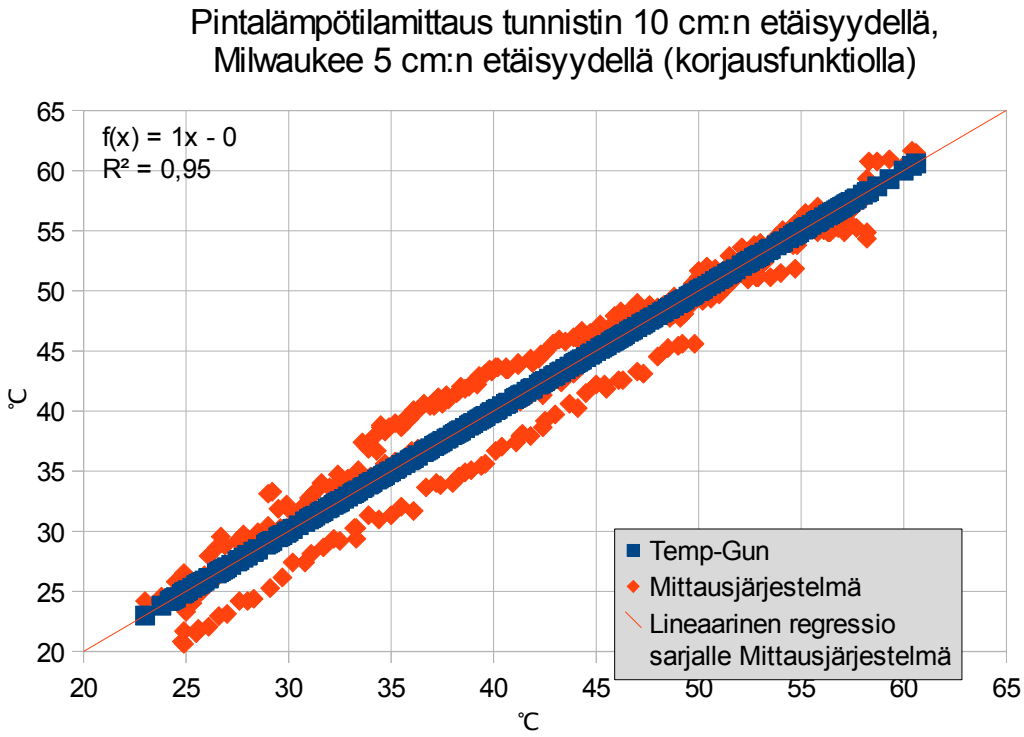
Kuva 7. Pintalämpötilamittaus, tunnistin 10 cm: etäisyydellä, Milwaukee 5 cm: etäisyydellä.

Temp-Gunin infrapunatunnistimen kennon mittausalueen keila on huomattavasti MLX90614:n vastaavaa kapeampi (mittausalueen keilan halkaisijan suhde mittausetäisyyteen verrattuna on 1:40).



Kuva 8. Mittausetäisyyden vaikutus MLX90614:n mittausalueen keilan halkaisijaan.

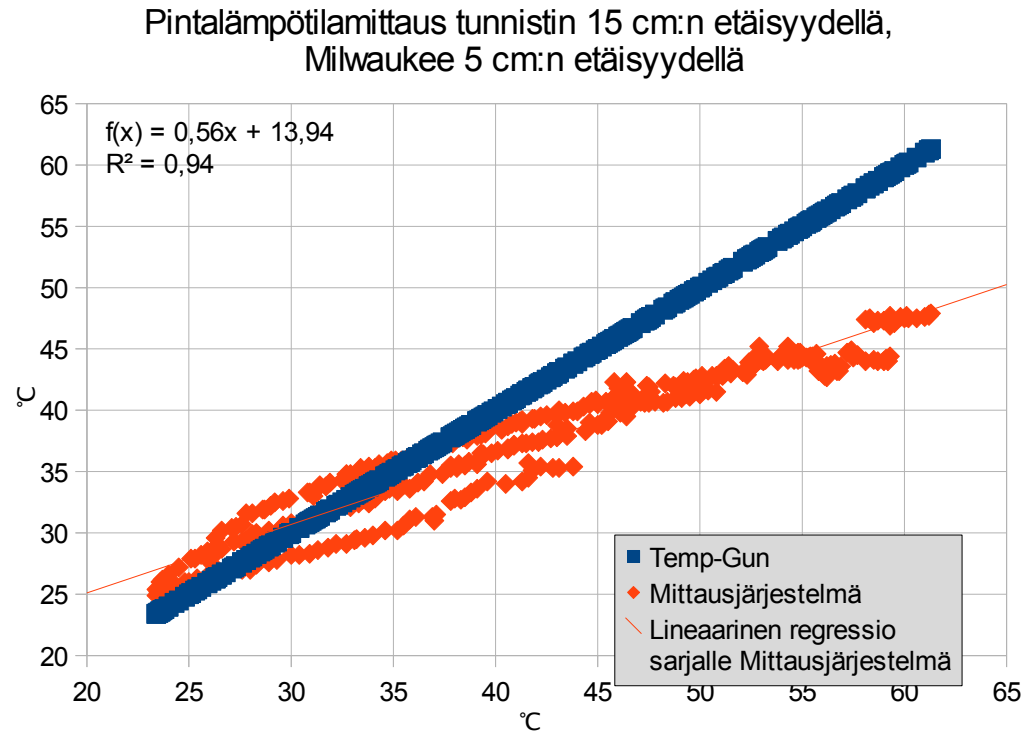
Kun 10 cm:n etäisyydeltä otetut mittaustulokset sovitettiin aiemmin kuvatulla tavalla saatiin kuvan 9 mukainen kuvaaja.



Kuva 9. Pintalämpötilamittaus tunnistin 10 cm:n etäisyydellä, Milwaukee 5 cm:n etäisyydellä, mittaustulokset sovitettu kuvaajaan korjausfunktion avulla.

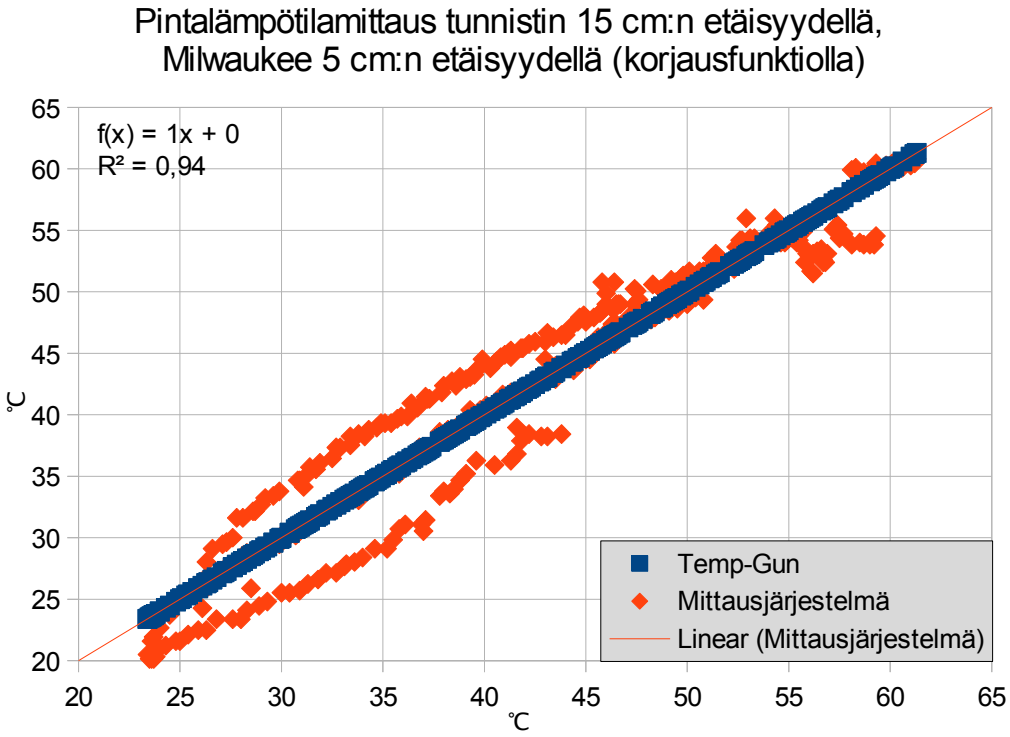
Verrattuna 5 cm:n etäisyydeltä suoritettuihin mittauksiin, selitysaste on laskenut 0,98:sta 0,95:teen ja lineaarisen regression kulmakerroin 0,7stä 0,56:teen. Voidaan siis todeta, että etäisyyden kasvaessa mittausjärjestelmän havaitsemat lämpötilat laskevat ja niiden luotettavuus heikkenee.

Kun mittausetäisyys kasvaa 15 cm:iin (kuva 10) voidaan mittausjärjestelmän mittaamien lämpötilojen havaita olevan edelleen matalampia kuin esimerkiksi 10 cm:n etäisyydeltä mitattaessa (regressiosuoran kulmakerroin pysyy samana, mutta sen vakio laskee arvosta 14,93 arvoon 13,94). Lisäksi mittaustulosten hajonta kasvaa erityisesti matalilla lämpötiloilla ja lineaarisen regression selitysaste laskee 0,95:stä 0,94:ään.



Kuva 10. Pintalämpötilamittaus tunnistin 15 cm:n etäisyydellä, Milwaukee 5 cm:n etäisyydellä.

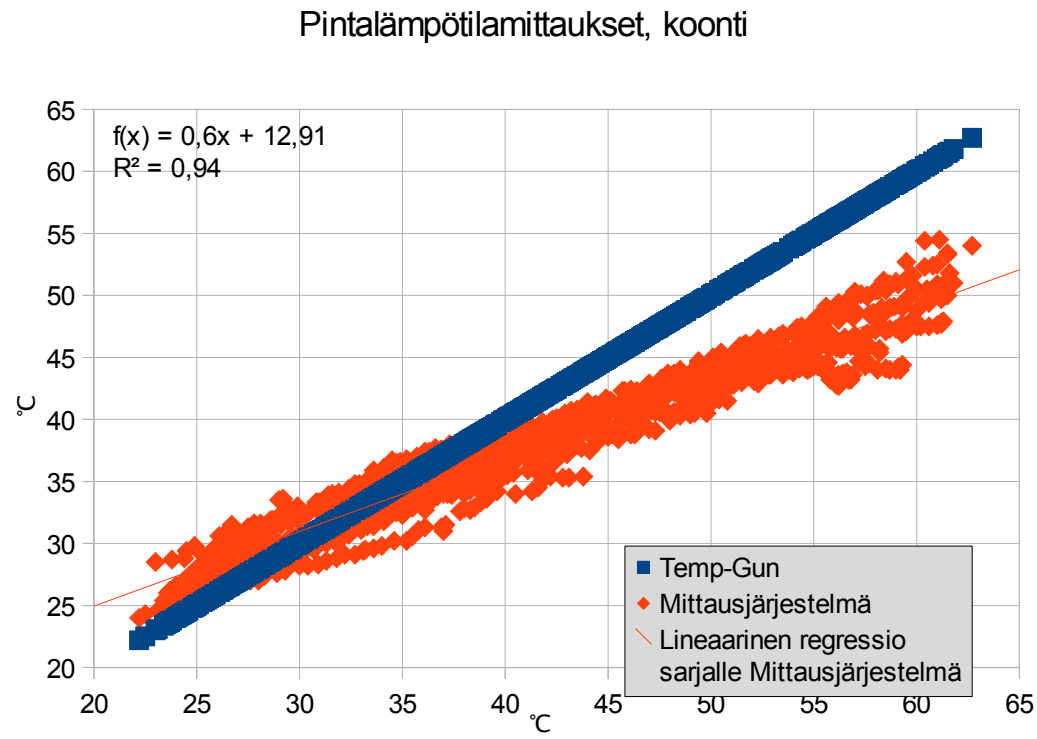
Kuvassa 11 on esitettynä 15 cm:stä suoritetut mittaukset korjausfunktiolla sovitettuna.



Kuva 11. Pintalämpötilamittaus, tunnistin 15 cm:n etäisyydellä, Milwaukee 5 cm:n etäisyydellä, tulokset sovitettu korjausfunktion avulla.

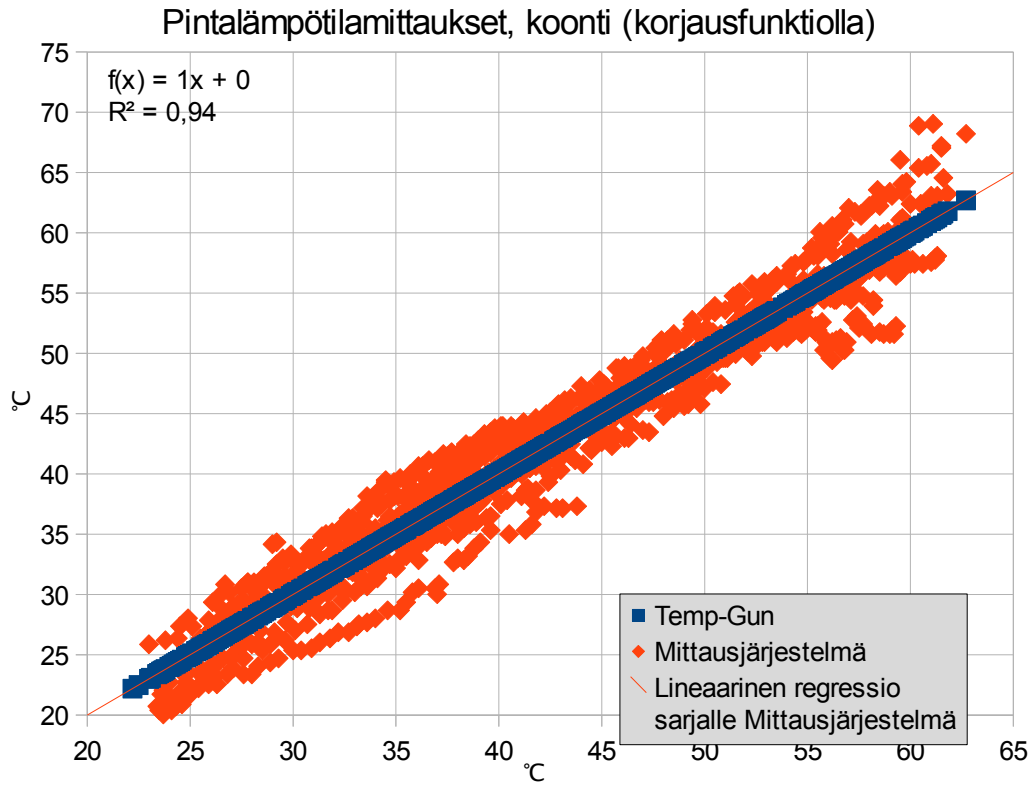
Jouston aiheuttaman etäisyyden muutoksen kaavasta voidaan todeta, että renkaan pinnan kaarevuuden aiheuttama mittaustuloksen muutos pysyy alle 10 cm:ssä yleisimmillä rengaskoilla suurissakin (0–23 cm:n) joustoissa, anturin ollessa sijoitettuna jouston liikeradan linjan kanssa kohtisuoralla linjalla, pyörännavan korkeudella. Näin ollen voidaan todeta että rengaslämpötilan mittausjärjestelmän mittaustarkkuus voidaan sovittaa etäisyydelle 5–15 cm.

Kuvaan 12 on koottu kaikki rengaslämpötilan mittausjärjestelmän mittaustulokset järjestelmän mittaustarkkuuden määritystä varten. Regressiosuoran selitysaste on hyvä (0,94).



Kuva 12. Koonti pintalämpötilamittauksista.

Tämän mittaustulosten koonnin regressiosuoran funktion perusteella määritetään mittausjärjestelmälle korjausfunktio, jonka mukaan korjattujen mittaustulosten regressiosuora on esitetty kuvassa 13.



Kuva 13. Pintalämpötilamittausten koonti korjatuilla mittaustuloksilla.

Arvioitaessa PNS-keinolla saatua mallia, tulee tarkastella myös selitysasteen tilastollista merkitsevyyttä. Suoran tapauksessa malli selittää mittaustulosten vaihtelua merkitsevästi, jos sen kulmakerroin poikkeaa riskitasolla 0,05 merkitsevästi nollasta [3, s. 292].

Määritetään lineaarisen regression t_0 käyttäen kaavaa 7 [3, s. 292]:

$$t_0 = \frac{|b_n|}{[\sigma_n]} \quad (7)$$

, missä b_n on suoran kulmakerroin ja σ_n kulmakertoimen keskihajonnan estimaatti.

Mittaustuloksia arvioitiin taulukkolaskentaohjelman linest-funktiota käyttäen (taulukko 1).

Taulukko 1. Linest-funktion arvot mittaustuloksista

b_n	1	$4{,}84*10^{-11}$	b_{n-1}
σ_n	0,00807...	0,356...	σ_{n-1}
r^2	0,939...	2,649...	σ_y
F	15342,942...	988	df

SS_{reg}	107686,797...	6934,419...	SS_{resid}

Taulukon perusteella laskien saadaan testisuureen arvoksi $t_0 \approx 124$. Verrattaessa saatua t_0 arvoa taulukkoon "t-jakauman pisteitä t_p, joille $P(T \geq t_p) = p$" [3, s. 303], voidaan todeta että, koska vapausasteiden ollessa 988 saatu testisuureen arvo ylittää riskitason rajan arvon, saadut mittaustulokset ovat tilastollisesti merkitseviä.

Milwaukee ilmoittaa Temp-Gunin mittaustarkkuudeksi ±1,5 %, mikä merkitsee mittausalueella 0–100 °C epätarkimmillaan noin ±1,5 °C tarkkuutta. Rengaslämpötilan mittausjärjestelmän mittaustulosten keskihajonta niiden lineaarisen regression suhteen mukaan σ_y = 2,65 °C kuvaa hyvin mittausjärjestelmän sisäisen virheen maksimia. Näiden kahden järjestelmän virheiden summa kuvaa hyvin mittausjärjestelmän mittaustarkkuutta, joka on epätarkimmillaan ±4,1 °C.

4.3 Mittaustaajuus

Rengaslämpötilan mittausjärjestelmän mittaustaajuus riippuu käytössä olevien anturisarjojen lukumäärästä. Mittausjärjestelmän ohjelmasta voidaan delay-komentojen perusteella laskea mittausjärjestelmän sisältävän viivettä noin 440 ms. Järjestelmän mittaustaajuuden voidaan siis arvioida olevan 1–2 Hz.

5 Yhteenveto

Opinnäytetyössä suunniteltiin väylänjakaja oppilastyönä valmistettuun rengaslämpötilan mittausjärjestelmään. Kyseinen järjestelmä käyttää infrapunasäteilyä lämpötilan määrittämiseen. Mittausjärjestelmän mittaustarkkuus määritettiin vertaamalla sen tuottamia mittaustuloksia kaupalliseen infrapunalämpömittariin ja sen mittaustaajuudesta tehtiin arvio.

Rengaslämpötilan mittausjärjestelmälle asetettu tavoitetarkkuus ±1 °C oli jo lähtökohtaisesti epärealistinen otettaessa huomioon, että Milwaukee Laser TEMP-GUN™ M12™:n mittatarkkuus lämpötila-alueella 0–100 °C on ±1,5 %. Mittausjärjestelmän tuottamien mittaustuloksien lineaarisen regression todettiin 5–15 cm:n etäisyydellä olevan selitysasteeltaan hyvä ja saatujen mittaustulosten olevan tilastollisesti merkitseviä. Näin ollen voidaan todeta rengaslämpötilan

mittausjärjestelmän mittaustarkkuuden olevan vähintään ±4,1 °C mittausalueella 0–65 °C. Teoreettiseksi mittaustaajuudeksi arvioidaan 1–2 Hz.

Insinöörityötä voidaan pitää onnistuneena, vaikka sen tuloksena valmistettu rengaslämpötilan mittausjärjestelmä täytti asetetut tavoitteet vain osittain. Tavoitteena ollutta mittaustarkkuutta ei saavutettu. Merkittävin syy tähän oli valittujen tunnistimen mittausalueen keilan liian suuri leveys. MLX90614xA kykenee valmistajan mukaan ±1 °C:n mittaustarkkuuteen ympäristön lämpötilan ollessa 0–50 °C ja mitattavan kohteen lämpötilan ollessa 0–120 °C. Esimerkiksi saman valmistajan anturi MLX90614xCF:n mittausalueen keilan kulma on 10 °, mikä parantaa mittaustulosten luotettavuutta. Lisäksi vertailumittauksissa olisi tullut käyttää tarkempaa mittavälinettä.

Mittausjärjestelmän mittaustaajuutta rajoittaa suurimmassa määrin väylänjakajassa käytettyjen transistorien nopeus. Mikäli järjestelmän nopeutta haluttaisiin parantaa, tulisi väylänjakaja suunnitella uudelleen tai järjestelmän rakennetta muokata siten, että tarve pitkille johtimille poistuu. Yksi rengaslämpötilan mittausjärjestelmälle asetetuista vaatimuksista oli sen helppo siirrettävyys ajoneuvosta toiseen, joten anturisarjojen johtimien pituuden optimointi tulee kysymykseen vain jos järjestelmä asennetaan kiinteästi johonkin ajoneuvoon. Toisena ratkaisuna ongelmaan voidaan pitää siirtymistä langattomaan tiedonsiirtoon anturisarjojen ja ohjainyksikön välillä.

Itse ohjainyksikön kokoa voisi pienentää huomattavasti suunnittelemalla mittausjärjestelmälle integroidun piirilevyn, joka sisältää kaikki mittausjärjestelmän osat. Siirtyminen yhteen suureen näyttöön vähentäisi tarvittavien komponenttien lukumäärää, mutta CDIO-projektin alkuvaiheessa ei ollut saatavilla riittävän suuria LCD-näyttöjä edulliseen hintaan, joten päädyttiin käyttämään neljää pienikokoista mutta edullista näyttöä.

Myös mittausjärjestelmän helppokäyttöisyyttä voisi parantaa. Toistaiseksi mittajärjestelmän parametrien muuttaminen vaatii sen kytkemistä tietokoneyhteyteen. Emissiivisyysarvon muuttaminen ilman tietokoneyhteyttä on yksi tärkeimmistä tulevaisuuden parannuksista. Lisäksi laitteeseen kytkettyjen anturisarjojen automaattinen tunnistus olisi mahdollista toteuttaa pelkällä ohjelmistopäivityksellä.

Lähteet

1 Arduino Products and Reference. Verkkodokumentti. Arduino. <www.arduino.cc>. Luettu 1.1.2013

2 I2C-bus specification and user manual , UM10204. 5.-9.2012. Verkkodokumentti. NXP. <www.nxp.com>. Luettu 1.3.2013.

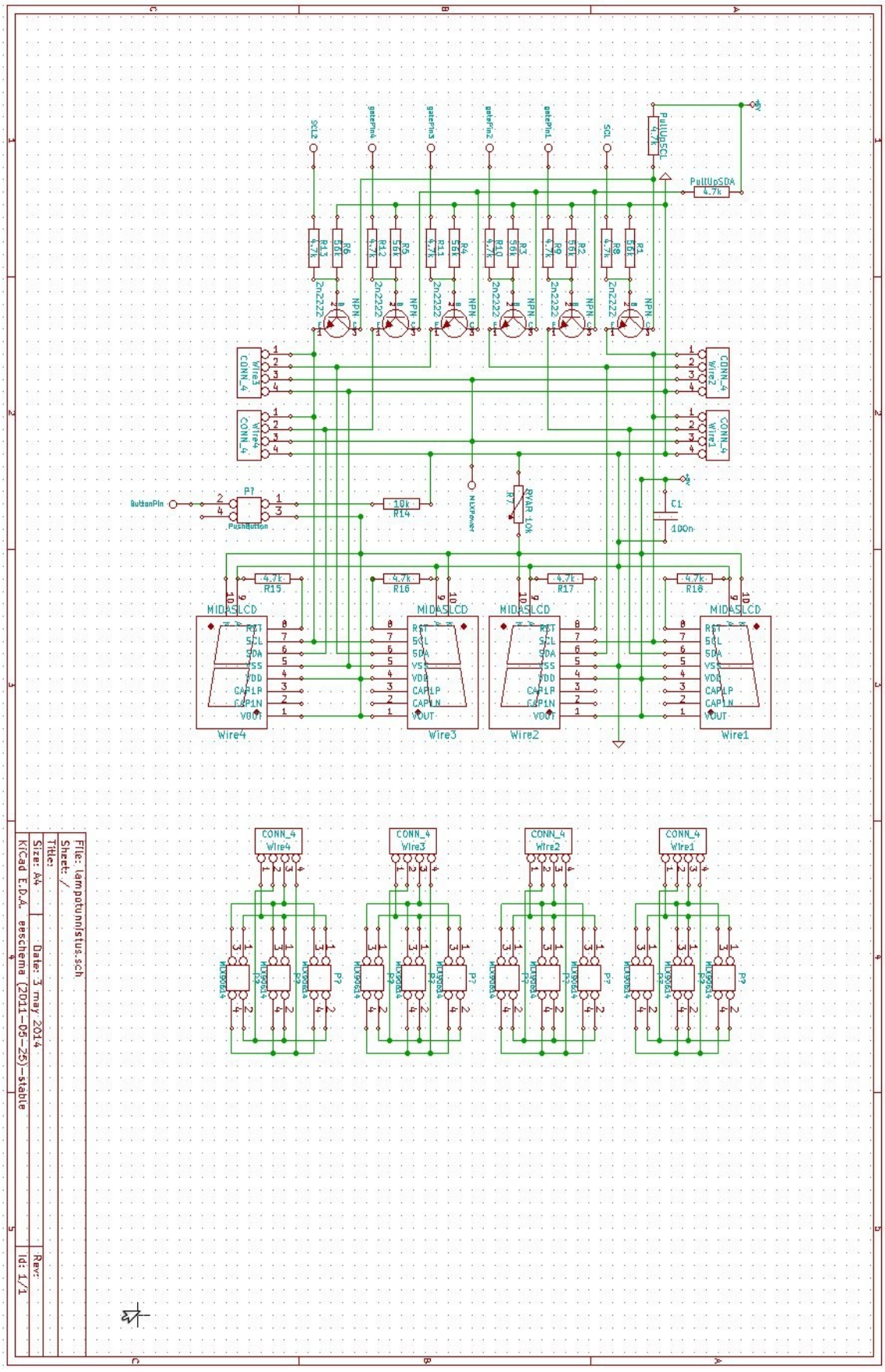
3 Laininen, Pertti. 1998. Todennäköisyys ja sen tilastollinen soveltaminen. Helsinki: Otatieto.

4 Melexis MLX90614 family Single and Dual Zone Infra Red Thermometer in TO-39 datasheet. Datalehti. Melexis.

5 Suvanto, Kari. 2010. Tekniikan fysiikka 1. Helsinki: Edita Publishing.

6 2N2222A datasheet. Datalehti. Microsemi.

Rengaslämpötilan mittausjärjestelmän kytkentäkaavio



Rengaslämpötilan mittausjärjestelmän komponenttien kustannukset

Osa/Part	Lkm/Amount	Yht./Total
MCCOG21605C6W-BNMLW (LCD-näyttö)	4	62.25 €
MLX90614ESF-BAA (Infrapunatunnistin/IR-Sensor)	2	39.43 €
MLX90614ESF-BAA (Infrapunatunnistin/IR-Sensor)	11	186.87 €
100nF kalvokondensaattori/condensator	11	4.09 €
2n2222a transistori/transistor	4	3.22 €
4,7Kohm vastus/resistor	10	1.70 €
SN74CBT3253D multiplexer	2	2.08 €
3ETL9-15.0 katkaisin/switch	2	5.93 €
91A1A-B28-D15Lpotentiometer, 10K	1	6.50 €
CABLE, 4 CORE, 0.22MM, 25M, 440V, 1A; Re	1	56.35 €
VEROBOARD, PCB, 121.92MM X 101.6MM; Boar	1	11.63 €
MCCOG21605C6W-BNMLW (LCD-näyttö)	2	31.12 €
PLUG, PANEL, 4WAY; Connector Type:Circul	4	19.34 €
SOCKET, 4WAY, FREE; Connector Type:Circu	4	29.76 €
Arduino Uno	1	27.58 €
	Yhteensä/Total(alv/tax 24%)	487.85 €

Infrapunalämpömittari: järjestelmän pienoisversio ja sen kokoamisohjeet

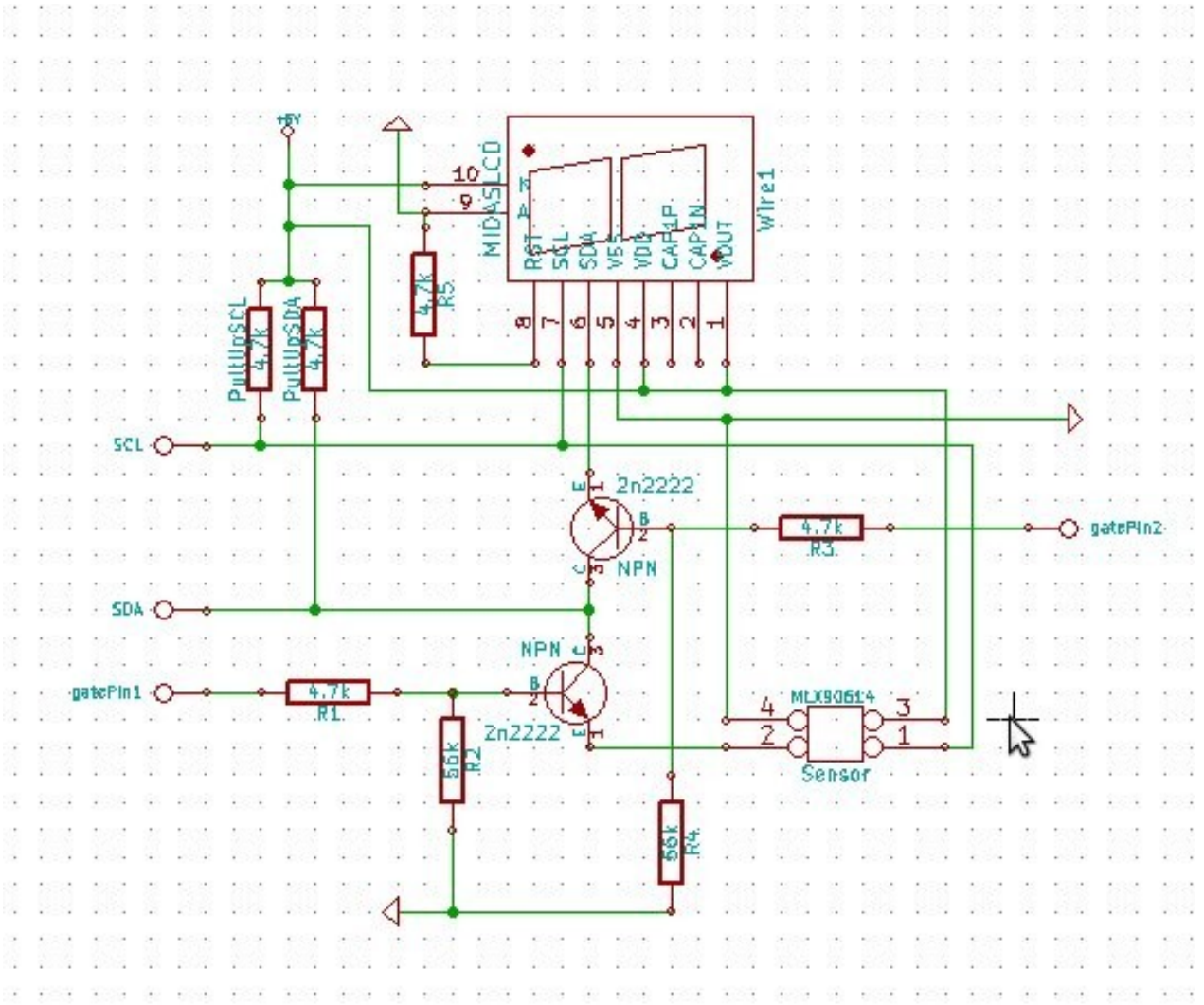
Infrapunalämpömittari

Metropolian automaatiotekniikan laboratorion opetuskäyttöön suunniteltu yksinkertaistettu malli rengaslämpötilan mittausjärjestelmästä sisältää lähes samat ominaisuudet kuin täysikokoinen järjestelmä. Malli on suunniteltu toimimaan osana opetuskäyttöä sulautettujen järjestelmien havainnollistamisessa. Oppilaat voivat koota järjestelmän koekytkentälevylle ja testata sen toimintaa käytännössä. Ajatuksena on havainnollistaa myös väylänjakajan toimintaperiaatetta asettamalla infrapuna-anturi ja lcd-näyttö samaan väyläosoitteeseen.

Näiden ohjeiden perusteella on mahdollista rakentaa LCD-näytöllinen lämpömittari joka hyödyntää väylätekniikkaa.

Vaadittavat komponentit:

1. MCCOG21605C6W-BNMLW (LCD-näyttö)
2. MLX90614ESF-BAA (Infrapunatunnistin)
3. 5kpl 4,7kΩ vastuksia
4. 2kpl 56kΩ vastuksia
5. 2kpl 2n2222 transistoreja
6. Arduino Uno
7. Koekytkentälevy
8. Johtimia



Kokoa järjestelmä kytkentäkaavion mukaisesti.

Lataa liitteenä oleva ohjelma mikrokontrolleriin.

Mikäli saat virheilmoituksia, varmista että koneeseen on asennettu tarvittavat kirjastot (ohjelman koodin kolme ensimmäistä riviä sisältävät niiden nimet).

Voit muuttaa anturin emissiivisyysarvoa sovittaaksesi sen mittaamaan eri materiaaleja mahdollisimman tarkasti.

```
#include "Wire.h"
#include "ST7036.h"
#include "i2cmaster.h"


ST7036 lcd = ST7036 ( 2, 16, 0x3E<<1 ); //16 characters and 2 line display, 7 byte address


//MODIFIABLE
const int gatePin1 = 1;       //muxer control pins
const int gatePin2 = 2;
```

```
const float emissivityValue = 0.95; // Melexis IR Sensors Default Emissivity is 0xFFFF, HEX, 65535, DEC
const int MLX = 0x00; //universal address for all I2C devices

//DON'T TOUCH
long emissivity = 65535*emissivityValue;
byte emissivityLSB = emissivity & 0xFF;
byte emissivityMSB = ((emissivity)>>8) & 0xFF;
double tempData = 0x0000;



  void setup() {
  //Serial.begin(9600); //Removing slashes will enable Serial connection
  //Serial.println("Setup...");

  //Serial.begin(9600);
  //Serial.println("emissivity check");
  //Serial.println(emissivityMSB,HEX);
  //Serial.print(emissivityLSB,HEX);

  pinMode(A4, INPUT);             //set SDA and SCL pins to analog input with
  pinMode(A5, INPUT);
  digitalWrite(A4, LOW);          //if LOW internal pull-ups turned off
  digitalWrite(A5, LOW);

  pinMode(gatePin1, OUTPUT);       //set Pins to OUTPUT mode
  pinMode(gatePin2, OUTPUT);

  delay(100);

  digitalWrite(gatePin1, LOW);
  digitalWrite(gatePin2, HIGH);   //Open connection to LCD
  delay(30);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensor");
  lcd.setCursor(1, 0);
  lcd.print("Setup...");
  delay(5);

  digitalWrite(gatePin1, HIGH);
  digitalWrite(gatePin2, LOW);   //Open connection to MLX90614

  ChangeEmissivity (MLX, emissivityLSB, emissivityMSB);


  digitalWrite(gatePin1, LOW);
  digitalWrite(gatePin2, HIGH);
  delay(30);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensor");
  lcd.setCursor(1, 0);
  lcd.print("Setup: Ready");
  delay(100);
  digitalWrite(gatePin1, LOW);
  digitalWrite(gatePin2, LOW);
  }
```

```
  void ChangeEmissivity (int dev, int lowbyte, int hibyte) // Use this to Adjust Sensor Emissivity
  {
  int tester = hibyte+lowbyte;
  //Serial.begin(9600);
  //Serial.print("Sensor ");
  //Serial.print(dev>>1, HEX);
  //Serial.println(" emissivity");
  //Serial.print((hibyte), HEX);
  //Serial.println((lowbyte), HEX);
  ReadEmis(dev);
  ZeroEmis(dev);
  ReadEmis(dev);
  ChangeEmis(dev, lowbyte, hibyte);
  while((ReadEmis(dev)) != (tester)){
  ZeroEmis(dev);
  ChangeEmis(dev, lowbyte, hibyte);
  delay(10);
  }
  }


  word ZeroEmis(int dev) {

  for (int a = 0; a != 256; a++) {
    i2c_start_wait(dev + I2C_WRITE);  //send start condition and write bit
    i2c_write(0x24);                  //send command for device to return address
    i2c_write(0x00);                  // send low byte zero to erase
    i2c_write(0x00);                  //send high byte zero to erase
    if (i2c_write(a) == 0) {          //checks all possible PECS
      i2c_stop();                     //Release bus, end transaction
      delay(100);
      //  Serial.print("Found correct CRC: 0x");
      //  Serial.println(a, HEX);
      return a;
      }
    }
  i2c_stop();                       //Release bus, end transaction
  //Serial.println("Correct CRC not found");
  return -1;
  }

  word ChangeEmis(int dev, int NewELo, int NewEHi) {

    for (int a = 0; a != 256; a++) {
    i2c_start_wait(dev + I2C_WRITE);  //send start condition and write bit
    i2c_write(0x24);                  //send command for device to return address
    i2c_write(NewELo);                  // send new low byte
    i2c_write(NewEHi);                  //send new high byte
    if (i2c_write(a) == 0) {
      i2c_stop();
      delay(100);
      //   Serial.print("Found correct CRC: 0x");
      //   Serial.println(a, HEX);
      return a;
      }
    }
  i2c_stop();                       //Release bus, end transaction
```

```
  //Serial.println("Correct CRC not found");
  return -1;
  }

  int ReadEmis (int dev){
    int data_low = 0;
    int data_high = 0;
    int pec = 0;

    i2c_start_wait(dev+I2C_WRITE);
    i2c_write(0x24);

    // read
    i2c_rep_start(dev+I2C_READ);
    data_low = i2c_readAck(); //Read 1 byte and then send ack
    data_high = i2c_readAck(); //Read 1 byte and then send ack
    pec = i2c_readNak();
    i2c_stop();

    //Serial.begin(9600);
    //Serial.print("Emissivity: ");
    //Serial.println(dev>>1, HEX);
    //Serial.println(data_high,HEX);
    //Serial.println(data_low,HEX);

    return int (data_low+data_high);

    delay(100);
  }



  float readtemp(int dev) {


  float celcius;

  int data_LSB = 0;
  int data_MSB = 0;
  int pec = 0;

  i2c_start_wait(dev+I2C_WRITE);

  i2c_write(0x07); //read temperature memory


  i2c_rep_start(dev+I2C_READ);
  data_LSB = i2c_readAck(); //Read 1 LSbyte and then send ack
  data_MSB = i2c_readAck(); //Read 1 MSbyte and then send ack
  pec = i2c_readNak();
  i2c_stop();

   //This converts high and low bytes together and processes temperature, MSB
is a error bit and is ignored for temperatures
    double tempFactor = 0.02; // 0.02 degrees per LSB (measurement resolution
of the MLX90614)
  double tempData = 0x0000; // zero out the data
  int frac; // data past the decimal point
```

```
  // This masks off the error bit of the high byte, then moves it left 8 bits and adds the low byte.
  tempData = double(((data_MSB & 0x007F) << 8) + data_LSB);
  tempData = (tempData * tempFactor)-0.01;

  celcius = (tempData - 273.15);

  //Serial.println((tempData - 273.15);

  //Serial.print("Celcius: ");
  //Serial.println(celcius);

  //Serial.print("Fahrenheit: ");
  //Serial.println(celcius);
  return celcius;
  }


  void loop() {

  float celcius1;

  digitalWrite(gatePin1, HIGH);
  digitalWrite(gatePin2, LOW);
  delay(30);
  celcius1 = readtemp(MLX);

  digitalWrite(gatePin1, LOW);
  digitalWrite(gatePin2, HIGH);
  delay(30);
    lcd.init();
    lcd.setCursor(0, 0);
    lcd.print("Temperature        ");
    lcd.setCursor(1, 0);
    lcd.print(celcius1,1);
    delay(5);

  }
```

Käyttöohje rengaslämpötilan mittausjärjestelmälle

Rengaslämpötilan mittausjärjestelmän käyttöohje

Rengaslämpötilan mittausjärjestelmä koostuu kahdesta pääkomponentista: anturisarjasta ja keskusyksiköstä. Yksittäisen väyläjohtimen päähän sijoitetut infrapunatunnistimet muodostavat anturisarjan. Keskusyksikkö koostuu väylänjakajasta ja Arduino Uno -mikrokontrollerista.

Valmistelut

Ohjelman lataaminen mikrokontrolleriin vaatii USB-kaapelin ja tietokoneen johon on asennettu Arduino IDE-käyttöliittymä.

1. Asenna koneeseen Arduino IDE-ohjelmisto.

Rengaslämpötilan mittausjärjestelmä vaatii Arduino IDE:n kirjastopäivityksen toimiakseen. Kirjastot kokonaisuudessaan ovat tämän ohjekirjan liitteinä. Nämä kolmannen osapuolen ohjelmoimat kirjastot voivat olla saatavilla myös sähköisessä muodossa.

2. Lataa kirjastotiedostot seuraavat kirjastotiedostot Arduino IDE:n libraries-hakemistoon:

1. **"Wire"**
2. **"LCD_C0220BIZ"**
3. **"I2Cmaster"**

Mikäli kirjastoja ei ole saatavilla, ne läytyvät myös tämän dokumentin liitteinä. Kukin liitteistä sisältää tiedostonimen, jolla luotu tiedosto tulee tallentaa Arduino IDE:n libraries-kansioon.

Ohjelman lataaminen mikrokontrolleriin

3. Valitse Arduino IDE:n (kuva 1) File liukuvalikosta Open ja avaa TireTemp.ino

Kuva 14: Arduino IDE -käyttöliittymä

Tee tarvittavat muutokset (yksittäisten anturien osoitteet, anturisarjojen järjestys ja lukumäärä, emissiivisyyssäätö).

4. Lataa ohjelma mikrokontrolleriin valitsemalla File-valikosta Upload tai pikanäppäimellä Ctrl+U.

Ohjelman muokkaaminen

Järjestelmän ohjelmaa ei ole suojattu mitenkään ja kaikkia sen osia voi muokata. Käytön kannalta oleellisimmat muokattavat ohjelman osat ovat anturisarjojen lukumäärä, niissä olevat anturit väyläosoitteineen ja anturien emissiivisyys.

Käytettävyyden kannalta tärkeimmät muokattavat osat on sijoitettu koodin alkuun muuttujiksi.

Muuttujat

Seuraavassa kappaleessa on esitelty ohjelman tärkeimmät muuttujat ja niiden käyttötarkoitus.

gatePin1...4 = syötetty arvo kertoo ohjelmalle mihin mikrokontrollerin digital out -pinniin muuttujalla viitataan. Näillä hallitaan SDa-signaalin kulkua väylänjakajan lävitse.

gatePinSCL...2 = kuten edellä mutta SCl-signaalin osalta.

buttonPin = määrittää mihin digital out -pinniin painonappi on kytketty.

dataPin1...4 = määrittää minkä pinnin kautta järjestelmä syöttää ulos pulssileveysmoduloituna järjestysnumeroltaan vastaavan anturisarjan keskilämpötilan.

set1...4 = kertoo järjestelmälle tietyn anturisarjan olevan käytössä. Järjestysnumero vastaa väylänjakajan SDa-väyläulostuloa.

dev1...12 = määrittää mistä väyläosoitteestä tiettyä anturia kutsutaan.

config (true...false) = kertoo järjestelmälle jos emissiivisyyden muutos on käytössä.

emissivityValue = emissiivisyysarvo joka tullaan syöttämään antureihin (0.01...0.99).

adjustZero1...12 = anturikohtainen korjausfunktion kerroin.

adjustZero1offset...12offset = anturikohtainen korjausfunktion vakio.

adjustZero ja adjustZerooffset muodostavat yhdessä korjausfunktion joka mahdollistaa lineaarisen virheen poistamisen.

Painonappi

Keskusyksikön painonappi on kaksitoiminen. Lyhyt painallus muuttaa keskusyksikön LCD-näyttöjen näkymän yksittäisten anturien lämpötiloista anturisarjan mittaamaan keskilämpötilaan.

Pitkä painallus katkaisee hetkellisesti jännitteen (VDD) anturisarjoilta.

HUOMIO! Mikäli näyttöjen arvoissa on selkeä virhe voit käyttää pitkää painallusta anturisarjojen väyläyhteyden korjaamiseksi.

Emissiivisyyden määritys

Ohjeet yhdelle infrapunamittarille ja lämpöparille

1. Teippaa maalarinteipillä 10cm x10cm alue tutkittavan aineen pinnasta (mikäli mahdollista). Sijoita lämpöpari tämän pinnan alle.

2. Lämmitä tutkittavan aineen lämpötila reilusti yli ympäristössä vallitsevan lämpötilan (ΔT>10°C). 3. Mittaa tutkittavan aineen lämpötilaa infrapunalämpömittarilla teipatun alan kohdalta noin 5 cm etäisyydeltä tai kuten infrapunalämpömittarin valmistaja on mittausetäisyyden määrittänyt.

Lämpöparin ja infrapunalämpömittarin välisestä lämpötilaerosta on mahdollista todeta kuinka hyvin kontaktimittaus onnistuu. Jos lämpötilaero on pienempi kuin 0,1'C (mittarien näyttämä suurimman osan ajasta sama, lämpötilan muutos näkyy toisessa mittarissa viiveellä tutkittavan aineen jäähtyessä), voidaan kontaktimittauksen sovitusta pitää riittävänä.

4. Lämmitä kohde uudestaan. Verrataan kontaktilämpömittauksen ja teippaamattoman pinnan lämpötilaeroa.

Mitattujen lämpötilojen erosta voi päätellä mittaako tuntemattoman pinnan lämpötilaa liian suurella (näyttää liian korkeita lämpötiloja) vai liian pienellä emissiivisyysarvolla (mitatut lämpötilat liian alhaisia).

5. Säädä mittalaitteen emissiivisyysarvoa saadun tiedon perusteella kunnes sen mittaustulokset ovat mahdollisimman lähellä kontaktimittauksen lämpötila-arvoja. Parhaan tuloksen saa erilaisilla haarukoimismenetelmillä.

Rengaslämpötilan mittausjärjestelmän koodi kommentteineen

```
//I would like to thank following individuals and online-communities for inspiration and advice

//LCD control example
//by Andy of workshopshed.com ( http://www.workshopshed.com/2012/07/a-little-venture-with-electronics-lcd.html )
//Melexis IR Sensor Temperature reading example
//by bildr.blog (bildr community) ( http://bildr.org/2011/02/mlx90614-arduino/ )
//Melexis IR Sensor Emissivity adjuster and
//I2C Slave address Changer example
//by arduino.cc forum username: paulrd ( http://arduino.cc/forum/index.php?topic=54170.0 )

//-Antti Harhanen

#include "Wire.h"
#include "ST7036.h"
#include "i2cmaster.h"


ST7036 lcd = ST7036 ( 2, 16, 0x3E<<1 ); //16 characters and 2 line display, 7 byte address


const int gatePin1 = 2;       //muxer control pins
const int gatePin2 = 4;
const int gatePin3 = 7;
const int gatePin4 = 8;
const int gatePinSCL = 10;
const int gatePinSCL2 = 11;
const int MLXPower = 12;

const int buttonPin = 13;     //pushbutton pin number

int button = 0;               //Button subprogram variable
int flip = 0;

const int dataPin1 = 3;
const int dataPin2 = 5;       //pins for datalogger DP1 = LF, DP2 = RF, DP3 = LR, DP = DP4
const int dataPin3 = 6;       //data is in 8-bit PWM
const int dataPin4 = 9;


//Sensors I2C address (7Byte)

//The sensor arrays
//set 1 (LF)
const boolean set1 = false;     //check true if a set is available
const int dev1 = 0x39<<1;
const int dev2 = 0x40<<1;
const int dev3 = 0x41<<1;
//set 2 (RF)
const boolean set2 = false;
const int dev4 = 0x42<<1;
const int dev5 = 0x46<<1;
const int dev6 = 0x47<<1;
```

```
//set 3 (LF)
const boolean set3 = false;
const int dev7 = 0x43<<1;
const int dev8 = 0x44<<1;
const int dev9 = 0x45<<1;
//set 4 (RR)
const boolean set4 = true;
const int dev10 = 0x48<<1;
const int dev11 = 0x49<<1;
const int dev12 = 0x50<<1;


//SENSOR EMISSIVITY
const boolean config = true;

const float emissivityValue = 0.95; // Melexis IR Sensors Default Emissivity is 0xFFFF, HEX, 65535, DEC

long emissivity = 65535*emissivityValue;

byte emissivityLSB = emissivity & 0xFF;
byte emissivityMSB = ((emissivity)>>8) & 0xFF;

double tempData = 0x0000;

// Melexis IR Sensors correction factor for calibrating the sensors
double adjustZero1 = 1;  //default = 1
double adjustZero1offset = 0;  // offset or calibration function, default = 0
double adjustZero2 = 1;
double adjustZero2offset = 0;
double adjustZero3 = 1;
double adjustZero3offset = 0;
double adjustZero4 = 1;
double adjustZero4offset = 0;
double adjustZero5 = 1;
double adjustZero5offset = 0;
double adjustZero6 = 1;
double adjustZero6offset = 0;
double adjustZero7 = 1;
double adjustZero7offset = 0;
double adjustZero8 = 1;
double adjustZero8offset = 0;
double adjustZero9 = 1;
double adjustZero9offset = 0;
double adjustZero10 = 1;
double adjustZero10offset = 0;
double adjustZero11 = 1;
double adjustZero11offset = 0;
double adjustZero12 = 1;
double adjustZero12offset = 0;



   void setup() {
     //Serial.begin(9600); //Removing slashes will enable Serial connection and disable independent mode
   //Serial.println("Setup...");

   //Serial.begin(9600);
   //Serial.println("emissivity check");
```

```
  //Serial.println(emissivityMSB,HEX);
  //Serial.print(emissivityLSB,HEX);

  pinMode(A4, INPUT);             //set SDA and SCL pins to analog input with
  pinMode(A5, INPUT);
  digitalWrite(A4, LOW);          //if LOW internal pull-ups turned off
  digitalWrite(A5, LOW);

  pinMode(gatePin1, OUTPUT);       //set Pins to OUTPUT mode
  pinMode(gatePin2, OUTPUT);
  pinMode(gatePin3, OUTPUT);
  pinMode(gatePin4, OUTPUT);
  pinMode(gatePinSCL, OUTPUT);
  pinMode(gatePinSCL2, OUTPUT);

  pinMode(dataPin1, OUTPUT);
  pinMode(dataPin2, OUTPUT);
  pinMode(dataPin3, OUTPUT);
  pinMode(dataPin4, OUTPUT);

  pinMode(MLXPower, OUTPUT);

  pinMode(buttonPin, INPUT);       //set button as input

  delay(100);

  digitalWrite(MLXPower, HIGH);

    digitalWrite(gatePinSCL, HIGH); //There must be a delay between opening
SCL and SDA
  delay(5);                    //if SCL and SDA go from 0 to 1 at the same time it's
    digitalWrite(gatePin1, HIGH);    //not a command. If SCL goes up first it's
"STOP"
  delay(30);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensors 1");
  lcd.setCursor(1, 0);
  lcd.print("Setup...");
  delay(5);

  if(set1 == true && config ==true){
  ChangeEmissivity (dev1, emissivityLSB, emissivityMSB);
  ChangeEmissivity (dev2, emissivityLSB, emissivityMSB);
  ChangeEmissivity (dev3, emissivityLSB, emissivityMSB);
  }

  delay(5);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensors 1");
  lcd.setCursor(1, 0);
  lcd.print("Setup: Ready");
  delay(5);

  digitalWrite(gatePin1, LOW);
  digitalWrite(gatePin2, HIGH);
  delay(30);
  lcd.init();
  lcd.setCursor(0, 0);
```

```
lcd.print("Temp. Sensors 2");
lcd.setCursor(1, 0);
lcd.print("Setup...");
delay(5);

if(set2 == true && config ==true){
ChangeEmissivity (dev4, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev5, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev6, emissivityLSB, emissivityMSB);
}

delay(5);
lcd.init();
lcd.setCursor(0, 0);
lcd.print("Temp. Sensors 2");
lcd.setCursor(1, 0);
lcd.print("Setup: Ready");
delay(5);

digitalWrite(gatePinSCL, LOW);
digitalWrite(gatePin2, LOW);
digitalWrite(gatePinSCL2, HIGH);
delay(5);
digitalWrite(gatePin3, HIGH);
delay(30);
lcd.init();
lcd.setCursor(0, 0);
lcd.print("Temp. Sensors 3");
lcd.setCursor(1, 0);
lcd.print("Setup...");
delay(5);

if(set3 == true && config ==true){
ChangeEmissivity (dev7, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev8, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev9, emissivityLSB, emissivityMSB);
}

delay(5);
lcd.init();
lcd.setCursor(0, 0);
lcd.print("Temp. Sensors 3");
lcd.setCursor(1, 0);
lcd.print("Setup: Ready");
delay(5);

digitalWrite(gatePin3, LOW);
digitalWrite(gatePin4, HIGH);
delay(5);
lcd.init();
lcd.setCursor(0, 0);
lcd.print("Temp. Sensors 4");
lcd.setCursor(1, 0);
lcd.print("Setup...");
delay(5);

if(set4 == true && config ==true){
ChangeEmissivity (dev10, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev11, emissivityLSB, emissivityMSB);
ChangeEmissivity (dev12, emissivityLSB, emissivityMSB);
```

```
  }

  delay(5);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensors 4");
  lcd.setCursor(1, 0);
  lcd.print("Setup: Ready");
  delay(5);

  digitalWrite(gatePinSCL2, LOW);
  digitalWrite(gatePin4, LOW);
  delay(100);
  }

   void ChangeEmissivity (int dev, int lowbyte, int hibyte) // Use this to Adjust
Sensor Emissivity
  {
  int tester = hibyte+lowbyte;
  //Serial.begin(9600);
  //Serial.print("Sensor ");
  //Serial.print(dev>>1, HEX);
  //Serial.println(" emissivity");
  //Serial.print((hibyte), HEX);
  //Serial.println((lowbyte), HEX);
  ReadEmis(dev);
  ZeroEmis(dev);
  ReadEmis(dev);
  ChangeEmis(dev, lowbyte, hibyte);
  while((ReadEmis(dev)) != (tester)){
  ZeroEmis(dev);
  ChangeEmis(dev, lowbyte, hibyte);
    digitalWrite(MLXPower, LOW);      //MLX VDD must be cycled to 0V after
changing
  delay(10);                    //emissivity values
  digitalWrite(MLXPower, HIGH);
  }
  }


  word ZeroEmis(int dev) {

  for (int a = 0; a != 256; a++) {
   i2c_start_wait(dev + I2C_WRITE);  //send start condition and write bit
   i2c_write(0x24);                 //send command for device to return address
   i2c_write(0x00);                 // send low byte zero to erase
   i2c_write(0x00);                 //send high byte zero to erase
   if (i2c_write(a) == 0) {         //checks all possible PECS
     i2c_stop();                   //Release bus, end transaction
     delay(100);
     //  Serial.print("Found correct CRC: 0x");
     //  Serial.println(a, HEX);
     return a;
     }
   }
  i2c_stop();                    //Release bus, end transaction
  //Serial.println("Correct CRC not found");
  return -1;
  }
```

```
word ChangeEmis(int dev, int NewELo, int NewEHi) {

  for (int a = 0; a != 256; a++) {
  i2c_start_wait(dev + I2C_WRITE);  //send start condition and write bit
  i2c_write(0x24);                  //send command for device to return address
  i2c_write(NewELo);                // send new low byte
  i2c_write(NewEHi);                //send new high byte
  if (i2c_write(a) == 0) {
    i2c_stop();
    delay(100);
    //   Serial.print("Found correct CRC: 0x");
    //   Serial.println(a, HEX);
    return a;
    }
  }
 i2c_stop();                        //Release bus, end transaction
 //Serial.println("Correct CRC not found");
 return -1;
 }

 int ReadEmis (int dev){
  int data_low = 0;
  int data_high = 0;
  int pec = 0;

  i2c_start_wait(dev+I2C_WRITE);
  i2c_write(0x24);

  // read
  i2c_rep_start(dev+I2C_READ);
  data_low = i2c_readAck(); //Read 1 byte and then send ack
  data_high = i2c_readAck(); //Read 1 byte and then send ack
  pec = i2c_readNak();
  i2c_stop();

  //Serial.begin(9600);
  //Serial.print("Emissivity: ");
  //Serial.println(dev>>1, HEX);
  //Serial.println(data_high,HEX);
  //Serial.println(data_low,HEX);

  return int (data_low+data_high);

  delay(100);
 }

 void datawriter(int dataPin, float average) {
 float averageout;
   averageout = map(average, 0, 100, 0, 255); //maps temperature 0-100C to
PWM output range
 analogWrite(dataPin, averageout);
 }

 void writelcd(boolean data, int set) {   //match data and set e.x. (set1,1)

 float celcius1;
 float celcius2;
 float celcius3;

 float average;
```

```
int PinSCL;
int Pin;
int dataPin;

int sens1;
int sens2;
int sens3;

if(data == false) {
  if(set == 1){
    PinSCL = gatePinSCL;
    Pin = gatePin1;

    }
  if(set == 2){
    PinSCL = gatePinSCL;
    Pin = gatePin2;

    }
  if(set == 3){
    PinSCL = gatePinSCL2;
    Pin = gatePin3;

    }
  if(set == 4){
    PinSCL = gatePinSCL2;
    Pin = gatePin4;

    }

  digitalWrite(PinSCL, HIGH);
  delay(30);
  digitalWrite(Pin, HIGH);
  delay(30);
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Temp. Sensors ");
  lcd.setCursor(0, 14);
  lcd.print(set);
  lcd.setCursor(1, 0);
  lcd.print("NO DATA");
  delay(5);

  digitalWrite(PinSCL, LOW);
  digitalWrite(Pin, LOW);

  }else{
  if(data == true) {

  if(set == 1){
    PinSCL = gatePinSCL;
    Pin = gatePin1;
    dataPin = dataPin1;
    sens1 = dev1;
    sens2 = dev2;
    sens3 = dev3;
    }
  if(set == 2){
    PinSCL = gatePinSCL;
```

```
  Pin = gatePin2;
  dataPin = dataPin2;
  sens1 = dev4;
  sens2 = dev5;
  sens3 = dev6;
  }
if(set == 3){
  PinSCL = gatePinSCL2;
  Pin = gatePin3;
  dataPin = dataPin3;
  sens1 = dev7;
  sens2 = dev8;
  sens3 = dev9;
  }
if(set == 4){
  PinSCL = gatePinSCL2;
  Pin = gatePin4;
  dataPin = dataPin4;
  sens1 = dev10;
  sens2 = dev11;
  sens3 = dev12;
  }

digitalWrite(PinSCL, HIGH);
delay(5);
digitalWrite(Pin, HIGH);
delay(30);

celcius1 = readtemp(sens1);
celcius2 = readtemp(sens2);
celcius3 = readtemp(sens3);

average=(celcius1+celcius2+celcius3)/3;
datawriter(dataPin, average);


  if(button == 0){

lcd.init();
lcd.setCursor(0, 0);
lcd.print("Out.  Mid.  Ins.");
lcd.setCursor(1, 0);
lcd.print(celcius1,1);
lcd.setCursor(1, 6);
lcd.print(celcius2,1);
lcd.setCursor(1, 12);
lcd.print(celcius3,1);
delay(5);

digitalWrite(gatePinSCL, LOW);
digitalWrite(Pin, LOW);

}else{

lcd.init();
lcd.setCursor(0, 0);
lcd.print("Average         ");
lcd.setCursor(1, 0);
lcd.print(average,1);
delay(5);
```

```
  digitalWrite(PinSCL, LOW);
  digitalWrite(Pin, LOW);

  }

  }else{
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("Parameter");
  lcd.setCursor(1, 0);
  lcd.print("Error");
  delay(5);

  digitalWrite(PinSCL, LOW);
  digitalWrite(Pin, LOW);
  }
  }
}

float readtemp(int dev) {


float celcius;

int data_LSB = 0;
int data_MSB = 0;
int pec = 0;
double corrFactor1 = 0; //correction factor of each separate sensor
double corrFactor2 = 0;

i2c_start_wait(dev+I2C_WRITE);

i2c_write(0x07); //read temperature memory


i2c_rep_start(dev+I2C_READ);
data_LSB = i2c_readAck(); //Read 1 LSbyte and then send ack
data_MSB = i2c_readAck(); //Read 1 MSbyte and then send ack
pec = i2c_readNak();
i2c_stop();

 //This converts high and low bytes together and processes temperature, MSB is a error bit and is ignored for temperatures
 double tempFactor = 0.02; // 0.02 degrees per LSB (measurement resolution of the MLX90614)
double tempData = 0x0000; // zero out the data
int frac; // data past the decimal point

 // This masks off the error bit of the high byte, then moves it left 8 bits and adds the low byte.
tempData = double(((data_MSB & 0x007F) << 8) + data_LSB);
tempData = (tempData * tempFactor)-0.01;

if (dev == dev1){corrFactor1 = adjustZero1; corrFactor2 = adjustZero1offset;}
if (dev == dev2){corrFactor1 = adjustZero2; corrFactor2 = adjustZero2offset;}
if (dev == dev3){corrFactor1 = adjustZero3; corrFactor2 = adjustZero3offset;}
if (dev == dev4){corrFactor1 = adjustZero4; corrFactor2 = adjustZero4offset;}
if (dev == dev5){corrFactor1 = adjustZero5; corrFactor2 = adjustZero5offset;}
if (dev == dev6){corrFactor1 = adjustZero6; corrFactor2 = adjustZero6offset;}
```

```
   if (dev == dev7){corrFactor1 = adjustZero7; corrFactor2 = adjustZero7offset;}
   if (dev == dev8){corrFactor1 = adjustZero8; corrFactor2 = adjustZero8offset;}
   if (dev == dev9){corrFactor1 = adjustZero9; corrFactor2 = adjustZero9offset;}
          if (dev == dev10){corrFactor1 = adjustZero10; corrFactor2 =
adjustZero10offset;}
          if (dev == dev11){corrFactor1 = adjustZero11; corrFactor2 =
adjustZero11offset;}
          if (dev == dev12){corrFactor1 = adjustZero12; corrFactor2 =
adjustZero12offset;}

   celcius = (tempData - 273.15)*corrFactor1 + corrFactor2;
   // float celcius = (celcius*1.8) + 32; //display in fahrenheits

   //Serial.println((tempData - 273.15)*corrFactor1 + corrFactor2);

   //Serial.print("Celcius: ");
   //Serial.println(celcius);

   //Serial.print("Fahrenheit: ");
   //Serial.println(celcius);
   return celcius;
   }

   void Button(){// pushbutton subprogram

   long time = 0;
   while(digitalRead(buttonPin) == HIGH)
   {time++;
   delay(1);
   //Serial.begin(9600);
   //Serial.println(time);
   }
   if(time>15)      //adjusting time sets the trigger for pushbutton
   {if(time>1000){digitalWrite(MLXPower, LOW);    //RESET MLX
   delay(100);
   digitalWrite(MLXPower, HIGH);}
   else
   {if(flip == 0){flip = 1; button = 1;}else{flip = 0; button = 0;}}
   }}



   void loop() {
   writelcd(set1,1);
   writelcd(set2,2);
   Button();
   writelcd(set3,3);
   writelcd(set4,4);
   Button();
   delay(0);
   }
```

Tarvittavat Arduino IDE -ohjelmakirjastot tiedostoineen

1. /Wire/Wire.h

```
/*
  TwoWire.h - TWI/I2C library for Arduino & Wiring
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef TwoWire_h
#define TwoWire_h

#include <inttypes.h>
#include "Stream.h"

#define BUFFER_LENGTH 32

class TwoWire : public Stream
{
  private:
    static uint8_t rxBuffer[];
    static uint8_t rxBufferIndex;
    static uint8_t rxBufferLength;

    static uint8_t txAddress;
    static uint8_t txBuffer[];
    static uint8_t txBufferIndex;
    static uint8_t txBufferLength;

    static uint8_t transmitting;
    static void (*user_onRequest)(void);
    static void (*user_onReceive)(int);
    static void onRequestService(void);
    static void onReceiveService(uint8_t*, int);
  public:
    TwoWire();
    void begin();
    void begin(uint8_t);
    void begin(int);
    void beginTransmission(uint8_t);
    void beginTransmission(int);
    uint8_t endTransmission(void);
    uint8_t requestFrom(uint8_t, uint8_t);
```

```
    uint8_t requestFrom(int, int);
    virtual size_t write(uint8_t);
    virtual size_t write(const uint8_t *, size_t);
    virtual int available(void);
    virtual int read(void);
    virtual int peek(void);
                virtual void flush(void);
    void onReceive( void (*)(int) );
    void onRequest( void (*)(void) );

    using Print::write;
};

extern TwoWire Wire;

#endif
```

2. /Wire/Wire.cpp

```
/*
  TwoWire.cpp - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

extern "C" {
  #include <stdlib.h>
  #include <string.h>
  #include <inttypes.h>
  #include "twi.h"
}

#include "Wire.h"

// Initialize Class Variables //////////////////////////////////////////////////

uint8_t TwoWire::rxBuffer[BUFFER_LENGTH];
uint8_t TwoWire::rxBufferIndex = 0;
uint8_t TwoWire::rxBufferLength = 0;

uint8_t TwoWire::txAddress = 0;
uint8_t TwoWire::txBuffer[BUFFER_LENGTH];
uint8_t TwoWire::txBufferIndex = 0;
uint8_t TwoWire::txBufferLength = 0;

uint8_t TwoWire::transmitting = 0;
void (*TwoWire::user_onRequest)(void);
void (*TwoWire::user_onReceive)(int);

// Constructors ////////////////////////////////////////////////////////////////

TwoWire::TwoWire()
{
}

// Public Methods //////////////////////////////////////////////////////////////

void TwoWire::begin(void)
{
  rxBufferIndex = 0;
```

```
  rxBufferLength = 0;

  txBufferIndex = 0;
  txBufferLength = 0;

  twi_init();
}

void TwoWire::begin(uint8_t address)
{
  twi_setAddress(address);
  twi_attachSlaveTxEvent(onRequestService);
  twi_attachSlaveRxEvent(onReceiveService);
  begin();
}

void TwoWire::begin(int address)
{
  begin((uint8_t)address);
}

uint8_t TwoWire::requestFrom(uint8_t address, uint8_t quantity)
{
  // clamp to buffer length
  if(quantity > BUFFER_LENGTH){
    quantity = BUFFER_LENGTH;
  }
  // perform blocking read into buffer
  uint8_t read = twi_readFrom(address, rxBuffer, quantity);
  // set rx buffer iterator vars
  rxBufferIndex = 0;
  rxBufferLength = read;

  return read;
}

uint8_t TwoWire::requestFrom(int address, int quantity)
{
  return requestFrom((uint8_t)address, (uint8_t)quantity);
}

void TwoWire::beginTransmission(uint8_t address)
{
  // indicate that we are transmitting
  transmitting = 1;
  // set address of targeted slave
  txAddress = address;
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
}

void TwoWire::beginTransmission(int address)
{
  beginTransmission((uint8_t)address);
}

uint8_t TwoWire::endTransmission(void)
{
  // transmit buffer (blocking)
```

```
  int8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
  // reset tx buffer iterator vars
  txBufferIndex = 0;
  txBufferLength = 0;
  // indicate that we are done transmitting
  transmitting = 0;
  return ret;
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
size_t TwoWire::write(uint8_t data)
{
  if(transmitting){
  // in master transmitter mode
    // don't bother if buffer is full
    if(txBufferLength >= BUFFER_LENGTH){
      setWriteError();
      return 0;
    }
    // put byte in tx buffer
    txBuffer[txBufferIndex] = data;
    ++txBufferIndex;
    // update amount in buffer
    txBufferLength = txBufferIndex;
  }else{
  // in slave send mode
    // reply to master
    twi_transmit(&data, 1);
  }
  return 1;
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
size_t TwoWire::write(const uint8_t *data, size_t quantity)
{
  if(transmitting){
  // in master transmitter mode
    for(size_t i = 0; i < quantity; ++i){
      write(data[i]);
    }
  }else{
  // in slave send mode
    // reply to master
    twi_transmit(data, quantity);
  }
  return quantity;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::available(void)
{
  return rxBufferLength - rxBufferIndex;
}
```

```
// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::read(void)
{
  int value = -1;

  // get each successive byte on each call
  if(rxBufferIndex < rxBufferLength){
    value = rxBuffer[rxBufferIndex];
    ++rxBufferIndex;
  }

  return value;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
int TwoWire::peek(void)
{
  int value = -1;

  if(rxBufferIndex < rxBufferLength){
    value = rxBuffer[rxBufferIndex];
  }

  return value;
}

void TwoWire::flush(void)
{
  // XXX: to be implemented.
}

// behind the scenes function that is called when data is received
void TwoWire::onReceiveService(uint8_t* inBytes, int numBytes)
{
  // don't bother if user hasn't registered a callback
  if(!user_onReceive){
    return;
  }
  // don't bother if rx buffer is in use by a master requestFrom() op
  // i know this drops data, but it allows for slight stupidity
  // meaning, they may not have read all the master requestFrom() data yet
  if(rxBufferIndex < rxBufferLength){
    return;
  }
  // copy twi rx buffer into local read buffer
  // this enables new reads to happen in parallel
  for(uint8_t i = 0; i < numBytes; ++i){
    rxBuffer[i] = inBytes[i];
  }
  // set rx iterator vars
  rxBufferIndex = 0;
  rxBufferLength = numBytes;
  // alert user program
  user_onReceive(numBytes);
}
```

```
// behind the scenes function that is called when data is requested
void TwoWire::onRequestService(void)
{
  // don't bother if user hasn't registered a callback
  if(!user_onRequest){
    return;
  }
  // reset tx buffer iterator vars
  // !!! this will kill any pending pre-master sendTo() activity
  txBufferIndex = 0;
  txBufferLength = 0;
  // alert user program
  user_onRequest();
}

// sets function called on slave write
void TwoWire::onReceive( void (*function)(int) )
{
  user_onReceive = function;
}

// sets function called on slave read
void TwoWire::onRequest( void (*function)(void) )
{
  user_onRequest = function;
}

// Preinstantiate Objects //////////////////////////////////////////////////////

TwoWire Wire = TwoWire();
```

3. /Wire/keywords.txt

```
#######################################
# Syntax Coloring Map For Wire
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

#######################################
# Methods and Functions (KEYWORD2)
#######################################

begin	KEYWORD2
beginTransmission	KEYWORD2
endTransmission	KEYWORD2
requestFrom	KEYWORD2
send	KEYWORD2
receive	KEYWORD2
onReceive	KEYWORD2
onRequest	KEYWORD2

#######################################
# Instances (KEYWORD2)
#######################################

Wire	KEYWORD2

#######################################
# Constants (LITERAL1)
#######################################
```

4. /Wire/utility/twi.c

```
/*
  twi.c - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <math.h>
#include <stdlib.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/twi.h>
#include "Arduino.h" // for digitalWrite

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif

#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#include "pins_arduino.h"
#include "twi.h"

static volatile uint8_t twi_state;
static uint8_t twi_slarw;

static void (*twi_onSlaveTransmit)(void);
static void (*twi_onSlaveReceive)(uint8_t*, int);

static uint8_t twi_masterBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_masterBufferIndex;
static uint8_t twi_masterBufferLength;

static uint8_t twi_txBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_txBufferIndex;
static volatile uint8_t twi_txBufferLength;

static uint8_t twi_rxBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_rxBufferIndex;
```

```
static volatile uint8_t twi_error;

/*
 * Function twi_init
 * Desc     readys twi pins and sets twi bitrate
 * Input    none
 * Output   none
 */
void twi_init(void)
{
  // initialize state
  twi_state = TWI_READY;

  // activate internal pullups for twi.
  digitalWrite(SDA, 1);
  digitalWrite(SCL, 1);

  // initialize twi prescaler and bit rate
  cbi(TWSR, TWPS0);
  cbi(TWSR, TWPS1);
  TWBR = ((F_CPU / TWI_FREQ) - 16) / 2;

  /* twi bit rate formula from atmega128 manual pg 204
  SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
  note: TWBR should be 10 or higher for master mode
  It is 72 for a 16mhz Wiring board with 100kHz TWI */

  // enable twi module, acks, and twi interrupt
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);
}

/*
 * Function twi_slaveInit
 * Desc     sets slave address and enables interrupt
 * Input    none
 * Output   none
 */
void twi_setAddress(uint8_t address)
{
  // set twi slave address (skip over TWGCE bit)
  TWAR = address << 1;
}

/*
 * Function twi_readFrom
 * Desc     attempts to become twi bus master and read a
 *          series of bytes from a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes to read into array
 * Output   number of bytes read
 */
uint8_t twi_readFrom(uint8_t address, uint8_t* data, uint8_t length)
{
  uint8_t i;

  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length){
    return 0;
```

```
  }

  // wait until twi is ready, become master receiver
  while(TWI_READY != twi_state){
    continue;
  }
  twi_state = TWI_MRX;
  // reset error state (0xFF.. no error occured)
  twi_error = 0xFF;

  // initialize buffer iteration vars
  twi_masterBufferIndex = 0;
  twi_masterBufferLength = length-1;  // This is not intuitive, read on...
  // On receive, the previously configured ACK/NACK setting is transmitted in
  // response to the received byte before the interrupt is signalled.
  // Therefor we must actually set NACK when the _next_ to last byte is
  // received, causing that NACK to be sent in response to receiving the last
  // expected byte of data.

  // build sla+w, slave device address + w bit
  twi_slarw = TW_READ;
  twi_slarw |= address << 1;

  // send start condition
     TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) |
_BV(TWSTA);

  // wait for read operation to complete
  while(TWI_MRX == twi_state){
    continue;
  }

  if (twi_masterBufferIndex < length)
    length = twi_masterBufferIndex;

  // copy twi buffer to data
  for(i = 0; i < length; ++i){
    data[i] = twi_masterBuffer[i];
  }

  return length;
}

/*
 * Function twi_writeTo
 * Desc     attempts to become twi bus master and write a
 *          series of bytes to a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes in array
 *          wait: boolean indicating to wait for write or not
 * Output   0 .. success
 *          1 .. length to long for buffer
 *          2 .. address send, NACK received
 *          3 .. data send, NACK received
 *          4 .. other twi error (lost bus arbitration, bus error, ..)
 */
uint8_t twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait)
{
  uint8_t i;
```

```
  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length){
    return 1;
  }

  // wait until twi is ready, become master transmitter
  while(TWI_READY != twi_state){
    continue;
  }
  twi_state = TWI_MTX;
  // reset error state (0xFF.. no error occured)
  twi_error = 0xFF;

  // initialize buffer iteration vars
  twi_masterBufferIndex = 0;
  twi_masterBufferLength = length;

  // copy data to twi buffer
  for(i = 0; i < length; ++i){
    twi_masterBuffer[i] = data[i];
  }

  // build sla+w, slave device address + w bit
  twi_slarw = TW_WRITE;
  twi_slarw |= address << 1;

  // send start condition
      TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) |
_BV(TWSTA);

  // wait for write operation to complete
  while(wait && (TWI_MTX == twi_state)){
    continue;
  }

  if (twi_error == 0xFF)
    return 0;      // success
  else if (twi_error == TW_MT_SLA_NACK)
    return 2;      // error: address send, nack received
  else if (twi_error == TW_MT_DATA_NACK)
    return 3;      // error: data send, nack received
  else
    return 4;      // other twi error
}

/*
 * Function twi_transmit
 * Desc     fills slave tx buffer with data
 *          must be called in slave tx event callback
 * Input    data: pointer to byte array
 *          length: number of bytes in array
 * Output   1 length too long for buffer
 *          2 not slave transmitter
 *          0 ok
 */
uint8_t twi_transmit(const uint8_t* data, uint8_t length)
{
  uint8_t i;
```

```
  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length){
    return 1;
  }

  // ensure we are currently a slave transmitter
  if(TWI_STX != twi_state){
    return 2;
  }

  // set length and copy data into tx buffer
  twi_txBufferLength = length;
  for(i = 0; i < length; ++i){
    twi_txBuffer[i] = data[i];
  }

  return 0;
}

/*
 * Function twi_attachSlaveRxEvent
 * Desc     sets function called before a slave read operation
 * Input    function: callback function to use
 * Output   none
 */
void twi_attachSlaveRxEvent( void (*function)(uint8_t*, int) )
{
  twi_onSlaveReceive = function;
}

/*
 * Function twi_attachSlaveTxEvent
 * Desc     sets function called before a slave write operation
 * Input    function: callback function to use
 * Output   none
 */
void twi_attachSlaveTxEvent( void (*function)(void) )
{
  twi_onSlaveTransmit = function;
}

/*
 * Function twi_reply
 * Desc     sends byte or readys receive line
 * Input    ack: byte indicating to ack or to nack
 * Output   none
 */
void twi_reply(uint8_t ack)
{
  // transmit master read ready signal, with or without ack
  if(ack){
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
  }else{
                        TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT);
  }
}

/*
 * Function twi_stop
 * Desc     relinquishes bus master status
```

```
 * Input    none
 * Output   none
 */
void twi_stop(void)
{
  // send stop condition
      TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) |
_BV(TWSTO);

  // wait for stop condition to be exectued on bus
  // TWINT is not set after a stop condition!
  while(TWCR & _BV(TWSTO)){
    continue;
  }

  // update twi state
  twi_state = TWI_READY;
}

/*
 * Function twi_releaseBus
 * Desc     releases bus control
 * Input    none
 * Output   none
 */
void twi_releaseBus(void)
{
  // release bus
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT);

  // update twi state
  twi_state = TWI_READY;
}

SIGNAL(TWI_vect)
{
  switch(TW_STATUS){
    // All Master
    case TW_START:     // sent start condition
    case TW_REP_START: // sent repeated start condition
      // copy device address and r/w bit to output register and ack
      TWDR = twi_slarw;
      twi_reply(1);
      break;

    // Master Transmitter
    case TW_MT_SLA_ACK:  // slave receiver acked address
    case TW_MT_DATA_ACK: // slave receiver acked data
      // if there is data to send, send it, otherwise stop
      if(twi_masterBufferIndex < twi_masterBufferLength){
        // copy data to output register and ack
        TWDR = twi_masterBuffer[twi_masterBufferIndex++];
        twi_reply(1);
      }else{
        twi_stop();
      }
      break;
    case TW_MT_SLA_NACK:  // address sent, nack received
      twi_error = TW_MT_SLA_NACK;
      twi_stop();
```

```
  break;
 case TW_MT_DATA_NACK: // data sent, nack received
  twi_error = TW_MT_DATA_NACK;
  twi_stop();
  break;
 case TW_MT_ARB_LOST: // lost bus arbitration
  twi_error = TW_MT_ARB_LOST;
  twi_releaseBus();
  break;

// Master Receiver
 case TW_MR_DATA_ACK: // data received, ack sent
  // put byte into buffer
  twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
 case TW_MR_SLA_ACK:  // address sent, ack received
  // ack if more bytes are expected, otherwise nack
  if(twi_masterBufferIndex < twi_masterBufferLength){
    twi_reply(1);
  }else{
    twi_reply(0);
  }
  break;
 case TW_MR_DATA_NACK: // data received, nack sent
  // put final byte into buffer
  twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
 case TW_MR_SLA_NACK: // address sent, nack received
  twi_stop();
  break;
// TW_MR_ARB_LOST handled by TW_MT_ARB_LOST case

// Slave Receiver
 case TW_SR_SLA_ACK:   // addressed, returned ack
 case TW_SR_GCALL_ACK: // addressed generally, returned ack
 case TW_SR_ARB_LOST_SLA_ACK:   // lost arbitration, returned ack
 case TW_SR_ARB_LOST_GCALL_ACK: // lost arbitration, returned ack
  // enter slave receiver mode
  twi_state = TWI_SRX;
  // indicate that rx buffer can be overwritten and ack
  twi_rxBufferIndex = 0;
  twi_reply(1);
  break;
 case TW_SR_DATA_ACK:       // data received, returned ack
 case TW_SR_GCALL_DATA_ACK: // data received generally, returned ack
  // if there is still room in the rx buffer
  if(twi_rxBufferIndex < TWI_BUFFER_LENGTH){
    // put byte in buffer and ack
    twi_rxBuffer[twi_rxBufferIndex++] = TWDR;
    twi_reply(1);
  }else{
    // otherwise nack
    twi_reply(0);
  }
  break;
 case TW_SR_STOP: // stop or repeated start condition received
  // put a null char after data if there's room
  if(twi_rxBufferIndex < TWI_BUFFER_LENGTH){
    twi_rxBuffer[twi_rxBufferIndex] = '\0';
  }
  // sends ack and stops interface for clock stretching
  twi_stop();
```

```
      // callback to user defined callback
      twi_onSlaveReceive(twi_rxBuffer, twi_rxBufferIndex);
      // since we submit rx buffer to "wire" library, we can reset it
      twi_rxBufferIndex = 0;
      // ack future responses and leave slave receiver state
      twi_releaseBus();
      break;
    case TW_SR_DATA_NACK:       // data received, returned nack
    case TW_SR_GCALL_DATA_NACK: // data received generally, returned nack
      // nack back at master
      twi_reply(0);
      break;

    // Slave Transmitter
    case TW_ST_SLA_ACK:         // addressed, returned ack
    case TW_ST_ARB_LOST_SLA_ACK: // arbitration lost, returned ack
      // enter slave transmitter mode
      twi_state = TWI_STX;
      // ready the tx buffer index for iteration
      twi_txBufferIndex = 0;
      // set tx buffer length to be zero, to verify if user changes it
      twi_txBufferLength = 0;
      // request for txBuffer to be filled and length to be set
      // note: user must call twi_transmit(bytes, length) to do this
      twi_onSlaveTransmit();
      // if they didn't change buffer & length, initialize it
      if(0 == twi_txBufferLength){
        twi_txBufferLength = 1;
        twi_txBuffer[0] = 0x00;
      }
      // transmit first byte from buffer, fall
    case TW_ST_DATA_ACK: // byte sent, ack returned
      // copy data to output register
      TWDR = twi_txBuffer[twi_txBufferIndex++];
      // if there is more to send, ack, otherwise nack
      if(twi_txBufferIndex < twi_txBufferLength){
        twi_reply(1);
      }else{
        twi_reply(0);
      }
      break;
    case TW_ST_DATA_NACK: // received nack, we are done
    case TW_ST_LAST_DATA: // received ack, but we are done already!
      // ack future responses
      twi_reply(1);
      // leave slave receiver state
      twi_state = TWI_READY;
      break;

    // All
    case TW_NO_INFO:   // no state information
      break;
    case TW_BUS_ERROR: // bus error, illegal stop/start
      twi_error = TW_BUS_ERROR;
      twi_stop();
      break;
  }
}
```

5. /Wire/utility/twi.h

```
/*
  twi.h - TWI/I2C library for Wiring & Arduino
  Copyright (c) 2006 Nicholas Zambetti.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef twi_h
#define twi_h

  #include <inttypes.h>

  //#define ATMEGA8

  #ifndef TWI_FREQ
  #define TWI_FREQ 100000L
  #endif

  #ifndef TWI_BUFFER_LENGTH
  #define TWI_BUFFER_LENGTH 32
  #endif

  #define TWI_READY 0
  #define TWI_MRX   1
  #define TWI_MTX   2
  #define TWI_SRX   3
  #define TWI_STX   4

  void twi_init(void);
  void twi_setAddress(uint8_t);
  uint8_t twi_readFrom(uint8_t, uint8_t*, uint8_t);
  uint8_t twi_writeTo(uint8_t, uint8_t*, uint8_t, uint8_t);
  uint8_t twi_transmit(const uint8_t*, uint8_t);
  void twi_attachSlaveRxEvent( void (*)(uint8_t*, int) );
  void twi_attachSlaveTxEvent( void (*)(void) );
  void twi_reply(uint8_t);
  void twi_stop(void);
  void twi_releaseBus(void);

#endif
```

6. /LCD_C0220BiZ/LCD_C0220BiZ.h

```
// ----------------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: No
//
// @file LCD_C0220BiZ.h
// NHD C0220BiZ display class definition.
//
// @brief Based on the LCD API 1.0 by dale@wentztech.com
//        This library implements the driver to drive the Newhaven Display
//        NHD-C0220BiZ-FSW-FBW-3V3M. The display is build around the ST7036
//        i2c LCD controller. This is a 3.3V display.
//        I2C displays based on the ST7632 should also be compatible.
//
//        Other compatible displays:
//          - NHD-C0220BiZ-FSW-FBW-3V3M
//          - NHD-C0220BiZ-FS(RGB)-FBW-3VM
//        Non tested but should be compatible with no or little changes
//          - NHD-C0216CiZ-FSW-FBW-3V3
//          - NHD-C0216CiZ-FN-FBW-3V
//
// @author F. Malpartida - fmalpartida@gmail.com
// ----------------------------------------------------------------------------

#ifndef LCD_C0220BIZ_h
#define LCD_C0220BIZ_h

#define _LCDEXPANDED                                                    //      If
defined turn on advanced functions

#include <Arduino.h>
#include "ST7036.h"

#define _LCD_C0220BIZ_VERSION "1.2.0"
#define _LCD_API_VERSION      "1.0"

class LCD_C0220BIZ : public ST7036
{

public:

  /**
   Constructor for the display class

   @return None

   LCD_C0220BIZ( );
   */
  LCD_C0220BIZ( );
```

```
    /**
     Constructor for the display class with backlight allowcation pin.

     @param backlightPin  initiales the backlight pin.

     @return None

     LCD_C0220BIZ(int8_t backlightPin );
     */
    LCD_C0220BIZ( int8_t backlightPin );

};

#endif
```

7. /LCD_C0220BiZ/LCD_C0220BiZ.cpp

```
// ----------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: No
//
// @file LCD_C0220BiZ.cpp
// Display class implementation of the LCD API 1.0
//
// @brief Based on the LCD API 1.0 by dale@wentztech.com
//        This library implements the driver to drive the Newhaven Display
//        NHD-C0220BiZ-FSW-FBW-3V3M. The display is build around the ST7036
//        i2c LCD controller. This is a 3.3V display.
//        I2C displays based on the ST7032i should also be compatible.
//
//        Other compatible displays:
//          - NHD-C0220BiZ-FSW-FBW-3V3M
//          - NHD-C0220BiZ-FS(RGB)-FBW-3VM
//        Non tested but should be compatible with no or little changes
//          - NHD-C0216CiZ-FSW-FBW-3V3
//          - NHD-C0216CiZ-FN-FBW-3V
//
//
// @author F. Malpartida - fmalpartida@gmail.com
// ----------------------------------------------------------------------
#include "LCD_C0220BiZ.h"

// Class private constants and definition
// ----------------------------------------------------------------------
const int     NUM_LINES          = 2;     // Number of lines in the display
const int     NUM_COLUMNS        = 20;    // Number of columns in the display
const int     I2C_ADDRS          = 0x78;  // I2C address of the display

// Static  member variable definitions
// ----------------------------------------------------------------------

// Static file scope variable definitions
// ----------------------------------------------------------------------

// Private support functions
// ----------------------------------------------------------------------

// CLASS METHODS
// ----------------------------------------------------------------------

// Constructors:
// ----------------------------------------------------------------------
LCD_C0220BIZ::LCD_C0220BIZ(  ):ST7036  (  NUM_LINES,  NUM_COLUMNS,
I2C_ADDRS )
{
```

```
}

LCD_C0220BIZ::LCD_C0220BIZ(int8_t backlightPin ) :
   ST7036 ( NUM_LINES, NUM_COLUMNS, I2C_ADDRS, backlightPin )
{

}
```

8. /LCD_C0220BiZ/keywords.txt

```
#######################################
# Syntax Coloring Map For Ultrasound
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

LCD_C0220BIZ                    KEYWORD1

#######################################
# Methods and Functions (KEYWORD2)
#######################################
init          KEYWORD2
setDelay      KEYWORD2
command       KEYWORD2
write         KEYWORD2
clear         KEYWORD2
home          KEYWORD2
on            KEYWORD2
off           KEYWORD2
cursor_on     KEYWORD2
cursor_off    KEYWORD2
blink_on      KEYWORD2
blink_off     KEYWORD2
setCursor     KEYWORD2
status        KEYWORD2
load_custom_character   KEYWORD2
keypad        KEYWORD2
setBacklight  KEYWORD2
setContrast   KEYWORD2

#######################################
# Constants (LITERAL1)
#######################################
```

9. /LCD_C0220BiZ/lcd.h

```
// ---------------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: No
//
// @file lcd.h
// LCD API 1.0 interface declaration class.
//
// @brief Based on the LCD API 1.0 by dale@wentztech.com
//        This class implements the LCD API abstract library class from
//        which all LCDs inherite.
//
// @author F. Malpartida - fmalpartida@gmail.com
// ---------------------------------------------------------------------------

#ifndef LCD_h
#define LCD_h
#include "Arduino.h"
#include "Print.h"


#define _LCDEXPANDED                                                    //      If
defined turn on advanced functions

#define _LCD_API_VERSION      "1.0"

class LCD : public Print
{

public:

  /**
   Send a command to the display

   @param value[in] Command to be sent to the display

   @return None

   void command(uint8_t value);
   */
                virtual void command(uint8_t value) = 0;

  /**
   Initialise the display. Once created the object, this is the next operation
   that has to be called to initialise the display into a known state. It
   assumes that the I2C bus is not initialised and hence initialise the Wire
   interface.

   Clear the display
```

```
Set contrast levels
Set the cursor at origens (0,0)
Turn on the entire display

void init();
*/
		virtual void init() = 0;

		/**
Set a different delay to that in the library. It may be needed to delay
sending commands or characters one after the other.

@param cmdDelay[in] Delay after issuing a command
@param charDelay[in] Delay after issuing a character to the LCD

@return None

void setDelay(int,int);
*/
		virtual void setDelay(int,int) = 0;

/**
This is the write method used by the Print library. It allows printing
characters to the display and new lines: print, println. It will write the
value to the display and increase the cursor.

@param value[in] character to write to the current LCD write position

@return None

virtual void write(uint8_t);
*/
		virtual size_t write(uint8_t) = 0;

/**
This is the write method used by the Print library. It allows printing
characters to the display and new lines: print, println. It will write the
value to the display and increase the cursor.

@param buffer[in] buffer to write to the current LCD write position
@param size[in] size of the buffer

@return None

virtual void write(uint8_t, size_t);
*/
virtual size_t write(const uint8_t *buffer, size_t size) = 0;

/**
Clear the display and set the cursor to 0,0

void clear();
*/
		virtual void clear() = 0;

/**
Set the cursor to 0,0

void home();
*/
```

```
		virtual void home() = 0;

/**
 Switch the display on. This is the default state when the display is
 initialised. See. init() method

 void on();
 */
		virtual void on() = 0;

/**
 Switch the display off.

 void off();
 */
		virtual void off() = 0;

/**
 Turn on the cursor "_".

 void cursor_on();
 */
		virtual void cursor_on() = 0;

/**
 Turn off the cursor. This is the default state when the display is
 initialised.

 void cursor_off();
 */
		virtual void cursor_off() = 0;

/**
 Activate cursor blink.

 void blink_on();
 */
		virtual void blink_on() = 0;

/**
 Deactivate cursor blinking. This is the default state when the display is
 initialised.

 void blink_off ();
 */
		virtual void blink_off() = 0;

/**
 Set the cursor at the following coordinates (Line, Col). Initial value after
 initialization is (0,0).

 @param Line[in] Line where to put the cursor, range (0, max display lines-1)
 This display only take (0, 1)
 @param Col[in]  Colum where to put the cursor, range (0, max width+1)

 @return None

 void setCursor(uint8_t Line, uint8_t Col );
 */
		virtual void setCursor(uint8_t Line, uint8_t Col ) = 0;
```

```
                //
  // EXPANDED FUNCTIONALITY METHODS
  // ------------------------------------------------------------------------

#ifdef _LCDEXPANDED

                /**
   Provides the state of the LCD. This value is updated every command is sent
   to the LCD or a character or a buffer is writen to the display.

   @return 0 OK, 1 if data was too big to be transmitted, 2 NACK on address
   transmission, 3 NACK on data transmission, 4 other error.

   uint8_t status();
   */
                virtual uint8_t status() = 0;

  /**
   Load a custom character on the display. After adding a new character to
   the character set, the coordinates are set to (0, 0). This method should
   be called during initialization.

   @param char_num[in] Character to load onto the display, this display supports
   upto 16 user defined characters.
   @param rows[in] Bitmap defining the character, the display assumes an array
   of 8 bytes per character.

   @return None.

   uint8_t load_custom_character(uint8_t char_num, uint8_t *rows);
   */
                virtual void load_custom_character(uint8_t char_num, uint8_t
*rows) = 0;

                /**
   NOT SUPPORTED

   uint8_t keypad();
   */
                virtual uint8_t keypad() = 0;

                void printstr(const char[]);

  /**
   Sets the backlight level. If the backlight level is connected to a PWM  pin,
   new_val will set a light level range between 0 and 255. If it is connected
   to a normal GPIO, from 0 to 127 it will be off and from 128 to 255 the
   backlight will be on. Backlight pin allocation on constructor.

   @param new_val[in] Backlight level of the display. Full range will only be
   available on pins with PWM support.

   @return None.

   uint8_t setBacklight();
   */
                virtual void setBacklight(uint8_t new_val) = 0;

  /**
```

```
    Sets the LCD contrast level.

    @param new_val[in] The contrast range (0 to 255) has been mapped to 16
    contrast levels on the display.

    @return None.

    uint8_t setContrast();
    */
                virtual void setContrast(uint8_t new_val) = 0;


#endif

private:

};

#endif
```

10./LCD_C0220BiZ/ST7036.h

```
// ------------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: No
//
// @file ST7036.h
// NHD C0220BiZ display class definition.
//
// @brief Based on the LCD API 1.0 by dale@wentztech.com
//        This library implements the driver to any I2C display with the ST7036
//        LCD controller.
//        I2C displays based on the ST7632 should also be compatible.
//
//        Other compatible displays:
//          - NHD-C0220BiZ-FSW-FBW-3V3M
//          - NHD-C0220ST7036BiZ-FS(RGB)-FBW-3VM
//        Non tested but should be compatible with no or little changes
//          - NHD-C0216CiZ-FSW-FBW-3V3
//          - NHD-C0216CiZ-FN-FBW-3V
//
// @author F. Malpartida - fmalpartida@gmail.com
// ------------------------------------------------------------------------

#ifndef ST7036_h
#define ST7036_h


#define _LCDEXPANDED                                                    //        If
defined turn on advanced functions
#include <Arduino.h>
#include <inttypes.h>
#include <Wire.h>
#include "Print.h"
#include "LCD.h"


#define _ST7036_VERSION       "1.2.0"
#define _LCD_API_VERSION      "1.0"

class ST7036 : public Print
{

public:

  /**
   Constructor for the display class

   @param num_lines[in] Number of lines in the display
   @param num_col[in]   Number of columns in the display
```

```
 @param i2cAddr[in]   i2c address of the display

 @return None

 ST7036(uint8_t num_lines, uint8_t num_col, uint8_t i2cAddr );
 */
ST7036(uint8_t num_lines, uint8_t num_col, uint8_t i2cAddr );

/**
 Constructor for the display class with backlight allowcation pin.

 @param num_lines[in] Number of lines in the display
 @param num_col[in]   Number of columns in the display
 @param i2cAddr[in]   i2c address of the display
 @param backlightPin  initiales the backlight pin.

 @return None

 ST7036(uint8_t num_lines, uint8_t num_col, uint8_t i2cAddr );
 */
ST7036(uint8_t num_lines, uint8_t num_col, uint8_t i2cAddr,
        int8_t backlightPin );

/**
 Send a command to the display

 @param value[in] Command to be sent to the display

 @return None

 void command(uint8_t value);
 */
                void command(uint8_t value);

/**
 Initialise the display. Once created the object, this is the next operation
 that has to be called to initialise the display into a known state. It
 assumes that the I2C bus is not initialised and hence initialise the Wire
 interface.

 Clear the display
 Set contrast levels
 Set the cursor at origens (0,0)
 Turn on the entire display

 void init();
 */
                void init();

                /**
 Set a different delay to that in the library. It may be needed to delay
 sending commands or characters one after the other.

 @param cmdDelay[in] Delay after issuing a command
 @param charDelay[in] Delay after issuing a character to the LCD

 @return None

 void setDelay(int,int);
 */
```

```
		void setDelay(int,int);

/**
 This is the write method used by the Print library. It allows printing
 characters to the display and new lines: print, println. It will write the
 value to the display and increase the cursor.

 @param value[in] character to write to the current LCD write position

 @return None

 virtual void write(uint8_t);
 */
		virtual size_t write(uint8_t);

/**
 This is the write method used by the Print library. It allows printing
 characters to the display and new lines: print, println. It will write the
 value to the display and increase the cursor.

 @param buffer[in] buffer to write to the current LCD write position
 @param size[in] size of the buffer

 @return None

 virtual void write(uint8_t, size_t);
 */
virtual size_t write(const uint8_t *buffer, size_t size);

/**
 Clear the display and set the cursor to 0,0

 void clear();
 */
		void clear();

/**
 Set the cursor to 0,0

 void home();
 */
		void home();

/**
 Switch the display on. This is the default state when the display is
 initialised. See. init() method

 void on();
 */
		void on();

/**
 Switch the display off.

 void off();
 */
		virtual void off();

/**
 Turn on the cursor "_".
```

```
 void cursor_on();
 */
                void cursor_on();

/**
 Turn off the cursor. This is the default state when the display is
 initialised.

 void cursor_off();
 */
                void cursor_off();

/**
 Activate cursor blink.

 void blink_on();
 */
                void blink_on();

/**
 Deactivate cursor blinking. This is the default state when the display is
 initialised.

 void blink_off ();
 */
                void blink_off();

/**
 Set the cursor at the following coordinates (Line, Col). Initial value after
 initialization is (0,0).

 @param Line[in] Line where to put the cursor, range (0, max display lines-1)
 This display only take (0, 1)
 @param Col[in]  Colum where to put the cursor, range (0, max width+1)

 @return None

 void setCursor(uint8_t Line, uint8_t Col );
 */
                void setCursor(uint8_t Line, uint8_t Col );

                //
// EXPANDED FUNCTIONALITY METHODS
// ------------------------------------------------------------------------

#ifdef _LCDEXPANDED

                /**
 Provides the state of the LCD. This value is updated every command is sent
 to the LCD or a character or a buffer is writen to the display.

 @return 0 OK, 1 if data was too big to be transmitted, 2 NACK on address
 transmission, 3 NACK on data transmission, 4 other error.

 uint8_t status();
 */
                uint8_t status();

/**
```

```
    Load a custom character on the display. After adding a new character to
    the character set, the coordinates are set to (0, 0). This method should
    be called during initialization.

    @param char_num[in] Character to load onto the display, this display supports
    upto 16 user defined characters.
    @param rows[in] Bitmap defining the character, the display assumes an array
    of 8 bytes per character.

    @return None.

    uint8_t load_custom_character(uint8_t char_num, uint8_t *rows);
    */
                    void load_custom_character(uint8_t char_num, uint8_t *rows);

                    /**
    NOT SUPPORTED

    uint8_t keypad();
    */
                    uint8_t keypad();

                    void printstr(const char[]);

   /**
    Sets the backlight level. If the backlight level is connected to a PWM  pin,
    new_val will set a light level range between 0 and 255. If it is connected
    to a normal GPIO, from 0 to 127 it will be off and from 128 to 255 the
    backlight will be on. Backlight pin allocation on constructor.

    @param new_val[in] Backlight level of the display. Full range will only be
    available on pins with PWM support.

    @return None.

    uint8_t setBacklight();
    */
                    void setBacklight(uint8_t new_val);

   /**
    Sets the LCD contrast level.

    @param new_val[in] The contrast range (0 to 255) has been mapped to 16
    contrast levels on the display.

    @return None.

    uint8_t setContrast();
    */
                    void setContrast(uint8_t new_val);


#endif

private:
                    uint8_t _num_lines;
    uint8_t _num_col;
    uint8_t _i2cAddress;
    int     _cmdDelay;
    int     _charDelay;
```

```
  bool    _initialised;
  uint8_t _status;
  int8_t  _backlightPin;

};

#endif
```

11./LCD_C0220BiZ/ST7036.cpp

```
// ------------------------------------------------------------------------
// Created by Francisco Malpartida on 20/08/11.
// Copyright 2011 - Under creative commons license 3.0:
//        Attribution-ShareAlike CC BY-SA
//
// This software is furnished "as is", without technical support, and with no
// warranty, express or implied, as to its usefulness for any purpose.
//
// Thread Safe: No
// Extendable: No
//
// @file LCD_C0220BiZ.cpp
// Display class implementation of the LCD API 1.0
//
// @brief Based on the LCD API 1.0 by dale@wentztech.com
//        This library implements the driver to any I2C display with the ST7036
//        LCD controller.
//        I2C displays based on the ST7032i should also be compatible.
//
//        Other compatible displays:
//          - NHD-C0220BiZ-FSW-FBW-3V3M
//          - NHD-C0220BiZ-FS(RGB)-FBW-3VM
//        Non tested but should be compatible with no or little changes
//          - NHD-C0216CiZ-FSW-FBW-3V3
//          - NHD-C0216CiZ-FN-FBW-3V
//
//
// @author F. Malpartida - fmalpartida@gmail.com
// ------------------------------------------------------------------------
#include <Arduino.h>                    //all things wiring / arduino
#include <Wire.h>
#include <string.h>                                     //needed for strlen()
#include <inttypes.h>

#include "LCD.h"
#include "ST7036.h"

// Class private constants and definition
// ------------------------------------------------------------------------
const int    CMD_DELAY         = 1;  // Command delay in miliseconds
const int    CHAR_DELAY        = 0;  // Delay between characters in miliseconds
const int    PIXEL_ROWS_PER_CHAR = 8;  // Number of pixel rows in the LCD
character
const int    MAX_USER_CHARS      = 16; // Maximun number of user defined
characters

// LCD Command set
```

```
const uint8_t DISP_CMD       = 0x0;  // Command for the display
const uint8_t RAM_WRITE_CMD  = 0x40; // Write to display RAM
const uint8_t CLEAR_DISP_CMD = 0x01; // Clear display command
const uint8_t HOME_CMD       = 0x02; // Set cursos at home (0,0)
const uint8_t DISP_ON_CMD    = 0x0C; // Display on command
const uint8_t DISP_OFF_CMD   = 0x08; // Display off Command
const uint8_t SET_DDRAM_CMD  = 0x80; // Set DDRAM address command
const uint8_t CONTRAST_CMD   = 0x70; // Set contrast LCD command
const uint8_t FUNC_SET_TBL0  = 0x38; // Function set - 8 bit, 2 line display 5x8,
inst table 0
const uint8_t FUNC_SET_TBL1  = 0x39; // Function set - 8 bit, 2 line display 5x8,
inst table 1

// LCD bitmap definition
const uint8_t CURSOR_ON_BIT  = ( 1 << 1 );// Cursor selection bit in Display on
cmd.
const uint8_t BLINK_ON_BIT    = ( 1 << 0 );// Blink selection bit on Display on
cmd.

// Driver DDRAM addressing
const uint8_t dram_dispAddr [][3] =
{
  { 0x00, 0x00, 0x00 },  // One line display address
  { 0x00, 0x40, 0x00 },  // Two line display address
  { 0x00, 0x10, 0x20 }   // Three line display address
};

// Static  member variable definitions
// ----------------------------------------------------------------------------

// Static file scope variable definitions
// ----------------------------------------------------------------------------

// Private support functions
// ----------------------------------------------------------------------------

// CLASS METHODS
// ----------------------------------------------------------------------------

// Constructors:
// ----------------------------------------------------------------------------
ST7036::ST7036(uint8_t num_lines, uint8_t num_col,
            uint8_t i2cAddr )
{
  _num_lines    = num_lines;
                  _num_col      = num_col;
  _i2cAddress   = ( i2cAddr >> 1 );
  _cmdDelay     = CMD_DELAY;
  _charDelay    = CHAR_DELAY;
  _initialised  = false;
  _backlightPin = -1;

}

ST7036::ST7036(uint8_t num_lines, uint8_t num_col,
            uint8_t i2cAddr, int8_t backlightPin )
{
  _num_lines    = num_lines;
                  _num_col      = num_col;
  _i2cAddress   = ( i2cAddr >> 1 );
```

```
  _cmdDelay     = CMD_DELAY;
  _charDelay    = CHAR_DELAY;
  _initialised  = false;
  _backlightPin = backlightPin;

  // If there is a pin assigned to the BL, set it as an output
  // ----------------------------------------------------------
  if ( _backlightPin != 0 )
  {
    pinMode ( _backlightPin, OUTPUT );
  }

}

// Functions: modifiers (set), selectors (get) and class methods
// ---------------------------------------------------------------------------
void ST7036::init ()
{
  size_t retVal;
  // Initialise the Wire library.
  Wire.begin();

  Wire.beginTransmission ( _i2cAddress );
  Wire.write ( (byte)0x0 );   // Send command to the display
  Wire.write ( FUNC_SET_TBL0 );
  delay (10);
  Wire.write ( FUNC_SET_TBL1 );
  delay (10);
  Wire.write ( 0x14 );  // Set BIAS - 1/5
  Wire.write ( 0x73 );  // Set contrast low byte
  Wire.write ( 0x5E );  // ICON disp on, Booster on, Contrast high byte
  Wire.write ( 0x6D );  // Follower circuit (internal), amp ratio (6)
  Wire.write ( 0x0C );  // Display on
  Wire.write ( 0x01 );  // Clear display
  Wire.write ( 0x06 );  // Entry mode set - increment
  _status = Wire.endTransmission ();

  if ( _status == 0 )
  {
    _initialised = true;
  }
}


void ST7036::setDelay (int cmdDelay,int charDelay)
{
                    _cmdDelay = cmdDelay;
                    _charDelay = charDelay;
}


void ST7036::command(uint8_t value)
{
  // If the LCD has been initialised correctly, write to it
  if ( _initialised )
  {
    Wire.beginTransmission ( _i2cAddress );
    Wire.write ( DISP_CMD );
    Wire.write ( value );
    _status = Wire.endTransmission ();
```

```
    delay(_cmdDelay);
  }
}


size_t ST7036::write(uint8_t value)
{
  // If the LCD has been initialised correctly write to it
  // ------------------------------------------------------
  if ( _initialised )
  {

    // If it is a new line, set the cursor to the next line (1,0)
    // ------------------------------------------------------------
    if ( value == '\n' )
    {
      setCursor (1,0);
    }
    else
    {
      Wire.beginTransmission ( _i2cAddress );
      Wire.write ( RAM_WRITE_CMD );
      Wire.write ( value );
      _status = Wire.endTransmission ();
      delay(_charDelay);
    }
  }
}

size_t ST7036::write(const uint8_t *buffer, size_t size)
{
  // If the LCD has been initialised correctly, write to it
  // ------------------------------------------------------
  if ( _initialised )
  {
    Wire.beginTransmission ( _i2cAddress );
    Wire.write ( RAM_WRITE_CMD );
    Wire.write ( (uint8_t *)buffer, size );
    _status = Wire.endTransmission ();
    delay(_charDelay);
  }
}


void ST7036::clear()
{
  command (CLEAR_DISP_CMD);
}


void ST7036::home()
{
  command ( HOME_CMD );
}


void ST7036::on()
{
  command ( DISP_ON_CMD );
}
```

```
void ST7036::off()
{
  command ( DISP_OFF_CMD );
}


void ST7036::cursor_on()
{
  command ( DISP_ON_CMD | CURSOR_ON_BIT );
}

void ST7036::cursor_off()
{
  command ( DISP_ON_CMD & ~(CURSOR_ON_BIT) );
}

void ST7036::blink_on()
{
  command ( DISP_ON_CMD | BLINK_ON_BIT );
}

void ST7036::blink_off()
{
  command ( DISP_ON_CMD & ~(BLINK_ON_BIT) );
}


void ST7036::setCursor(uint8_t line_num, uint8_t x)
{
  uint8_t base = 0x00;

  // If the LCD has been initialised correctly, write to it
  // ------------------------------------------------------
  if ( _initialised )
  {
    // set the baseline address with respect to the number of lines of
    // the display
    base = dram_dispAddr[_num_lines-1][line_num];
    base = SET_DDRAM_CMD + base + x;
    command ( base );
  }
}

#ifdef _LCDEXPANDED
uint8_t ST7036::status(){

                return _status;
}


uint8_t ST7036::keypad ()
{
  // NOT SUPPORTED
  return 0;
}


void ST7036::load_custom_character (uint8_t char_num, uint8_t *rows)
```

```
{
  // If the LCD has been initialised correctly start writing to it
  // ----------------------------------------------------------------------
  if ( _initialised )
  {
    // If it is a valid place holder for the character, write it into the
    // display's CGRAM
    // ----------------------------------------------------------------------
    if ( char_num < MAX_USER_CHARS )
    {
       // Set up the display to write into CGRAM - configure LCD to use func table
0
      Wire.beginTransmission ( _i2cAddress );
      Wire.write ( DISP_CMD );
        Wire.write ( FUNC_SET_TBL0 ); // Function set: 8 bit, 2 line display 5x8,
funct tab 0
      delay ( _cmdDelay );

      // Set CGRAM position to write
            Wire.write ( RAM_WRITE_CMD + (PIXEL_ROWS_PER_CHAR *
char_num) );
      _status = Wire.endTransmission ();

          // If we have changed the function table and configured the CGRAM
position
      // write the new character to the LCD's CGRAM
      // ----------------------------------------------------------------------
      if ( _status == 0 )
      {
            write ( rows, PIXEL_ROWS_PER_CHAR ); // write the character to
CGRAM

        // Leave the LCD as it was - function table 1 DDRAM and set the cursor
        // position to (0, 0) to start writing.
        command ( FUNC_SET_TBL1 );
        setCursor ( 0,0 );
      }
    }
  }
}


void ST7036::setBacklight(uint8_t new_val)
{
  // Set analog write to the pin, the routine already checks if it can
  // set a PWM or not.
  // ----------------------------------------------------------------------
                if ( _backlightPin != -1 )
  {
    analogWrite ( _backlightPin, new_val );
  }
}


void ST7036::setContrast(uint8_t new_val)
{
  // Only allow 15 levels of contrast
                new_val = map ( new_val, 0, 255, 0, 15 );

  command(CONTRAST_CMD + new_val);
```

```
}
#endif // _LCDEXPANDED
```

12. /I2Cmaster/i2cmaster.h

```
#ifndef _I2CMASTER_H
#define _I2CMASTER_H   1
/************************************************************************
* Title:    C include file for the I2C master interface
*           (i2cmaster.S or twimaster.c)
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device
* Usage:    see Doxygen manual
************************************************************************/

#ifdef DOXYGEN
/**
 @defgroup pfleury_ic2master I2C Master library
 @code #include <i2cmaster.h> @endcode

 @brief I2C (TWI) Master Software Library

 Basic routines for communicating with I2C slave devices. This single master
 implementation is limited to one bus master on the I2C bus.

   This I2c library is implemented as a compact assembler software implementation of the I2C protocol
 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for all AVR with built-in TWI hardware (twimaster.c).
 Since the API for these two implementations is exactly the same, an application can be linked either against the
 software I2C implementation or the hardware I2C implementation.

 Use 4.7k pull-up resistor on the SDA and SCL pin.

 Adapt the SCL and SDA port and pin definitions and eventually the delay routine in the module
 i2cmaster.S to your target when using the software I2C implementation !

 Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile when using the TWI hardware implementaion.

 @note
     The module i2cmaster.S is based on the Atmel Application Note AVR300, corrected and adapted
    to GNU assembler and AVR-GCC C call interface.
    Replaced the incorrect quarter period delays found in AVR300 with
    half period delays.

 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury

 @par API Usage Example
    The following code shows typical usage of this library, see example test_i2cmaster.c

 @code

 #include <i2cmaster.h>
```

```
 #define Dev24C02   0xA2          // device address of EEPROM 24C02, see datasheet

 int main(void)
 {
    unsigned char ret;

    i2c_init();                     // initialize I2C library

    // write 0x75 to EEPROM address 5 (Byte Write)
      i2c_start_wait(Dev24C02+I2C_WRITE);       // set device address and write mode
    i2c_write(0x05);                  // write address = 5
    i2c_write(0x75);                  // write value 0x75 to EEPROM
    i2c_stop();                      // set stop conditon = release bus


    // read previously written value back from EEPROM address 5
      i2c_start_wait(Dev24C02+I2C_WRITE);       // set device address and write mode

    i2c_write(0x05);                  // write address = 5
      i2c_rep_start(Dev24C02+I2C_READ);         // set device address and read mode

    ret = i2c_readNak();                // read one byte from EEPROM
    i2c_stop();

    for(;;);
 }
 @endcode

*/
#endif /* DOXYGEN */

/**@{*/

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
#error "This library requires AVR-GCC 3.4 or later, update to newer AVR-GCC compiler !"
#endif

#include <avr/io.h>

/** defines the data direction (reading from I2C device) in i2c_start(),i2c_rep_start() */
#define I2C_READ    1

/** defines the data direction (writing to I2C device) in i2c_start(),i2c_rep_start() */
#define I2C_WRITE   0


/**
 @brief initialize the I2C master interace. Need to be called only once
 @param  void
 @return none
 */
extern void i2c_init(void);
```

```
/**
 @brief Terminates the data transfer and releases the I2C bus
 @param void
 @return none
 */
extern void i2c_stop(void);


/**
 @brief Issues a start condition and sends address and transfer direction

 @param    addr address and transfer direction of I2C device
 @retval   0   device accessible
 @retval   1   failed to access device
 */
extern unsigned char i2c_start(unsigned char addr);


/**
  @brief  Issues a repeated start condition and sends address and transfer direction

 @param   addr address and transfer direction of I2C device
 @retval  0 device accessible
 @retval  1 failed to access device
 */
extern unsigned char i2c_rep_start(unsigned char addr);


/**
 @brief Issues a start condition and sends address and transfer direction

 If device is busy, use ack polling to wait until device ready
 @param    addr address and transfer direction of I2C device
 @return   none
 */
extern void i2c_start_wait(unsigned char addr);


/**
 @brief Send one byte to I2C device
 @param    data  byte to be transfered
 @retval   0 write successful
 @retval   1 write failed
 */
extern unsigned char i2c_write(unsigned char data);


/**
 @brief    read one byte from the I2C device, request more data from device
 @return   byte read from I2C device
 */
extern unsigned char i2c_readAck(void);

/**
 @brief    read one byte from the I2C device, read is followed by a stop condition
 @return   byte read from I2C device
 */
```

```
extern unsigned char i2c_readNak(void);

/**
 @brief    read one byte from the I2C device

 Implemented as a macro, which calls either i2c_readAck or i2c_readNak

 @param    ack 1 send ack, request more data from device<br>
           0 send nak, read is followed by a stop condition
 @return   byte read from I2C device
 */
extern unsigned char i2c_read(unsigned char ack);
#define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();


/**@}*/
#endif
```

13. /I2Cmaster/keywords.txt

```
#######################################
# Syntax I2Cmaster
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

#######################################
# Methods and Functions (KEYWORD2)
#######################################
i2c_init                        KEYWORD2
i2c_stop                        KEYWORD2
i2c_start                       KEYWORD2
i2c_rep_start   KEYWORD2
i2c_start_wait  KEYWORD2
i2c_write                       KEYWORD2
i2c_readAck                     KEYWORD2
i2c_readNak                     KEYWORD2
i2c_read                        KEYWORD2

#######################################
# Constants (LITERAL1)
#######################################
```

14. /I2Cmaster/twimaster.cpp

```
/*************************************************************************
**
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>

#include <i2cmaster.h>


/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

/* I2C clock in Hz */
#define SCL_CLOCK 50000L


/*************************************************************************
 Initialization of the I2C bus interface. Need to be called only once
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */

}/* i2c_init */


/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

                // send START condition
                TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

                // wait until transmission completed
                while(!(TWCR & (1<<TWINT)));

                // check value of TWI Status Register. Mask prescaler bits.
                twst = TW_STATUS & 0xF8;
                if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;
```

```
                // send device address
                TWDR = address;
                TWCR = (1<<TWINT) | (1<<TWEN);

                // wail until transmission completed and ACK/NACK has been
received
                while(!(TWCR & (1<<TWINT)));

                // check value of TWI Status Register. Mask prescaler bits.
                twst = TW_STATUS & 0xF8;
                if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) )
return 1;

                return 0;

}/* i2c_start */


/*************************************************************************
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************/
void i2c_start_wait(unsigned char address)
{
   uint8_t   twst;


   while ( 1 )
   {
                   // send START condition
                   TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

                // wait until transmission completed
                while(!(TWCR & (1<<TWINT)));

                // check value of TWI Status Register. Mask prescaler bits.
                twst = TW_STATUS & 0xF8;
                if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

                // send device address
                TWDR = address;
                TWCR = (1<<TWINT) | (1<<TWEN);

                // wail until transmission completed
                while(!(TWCR & (1<<TWINT)));

                // check value of TWI Status Register. Mask prescaler bits.
                twst = TW_STATUS & 0xF8;
                if      (       (twst       ==       TW_MT_SLA_NACK      )||(twst
==TW_MR_DATA_NACK) )
                {
                   /* device busy, send stop condition to terminate write operation
*/
                     TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

                     // wait until stop condition is executed and bus released
                     while(TWCR & (1<<TWSTO));
```

```
                    continue;
                }
                //if( twst != TW_MT_SLA_ACK) return 1;
                break;
    }

}/* i2c_start_wait */


/*************************************************************************
 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible
          1 failed to access device
*************************************************************************/
unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );

}/* i2c_rep_start */


/*************************************************************************
 Terminates the data transfer and releases the I2C bus
*************************************************************************/
void i2c_stop(void)
{
    /* send stop condition */
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

                // wait until stop condition is executed and bus released
                while(TWCR & (1<<TWSTO));

}/* i2c_stop */


/*************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
            1 write failed
*************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

                // send data to the previously addressed device
                TWDR = data;
                TWCR = (1<<TWINT) | (1<<TWEN);

                // wait until transmission completed
                while(!(TWCR & (1<<TWINT)));

                // check value of TWI Status Register. Mask prescaler bits
                twst = TW_STATUS & 0xF8;
                if( twst != TW_MT_DATA_ACK) return 1;
                return 0;
```

```
}/* i2c_write */


/*************************************************************************
 Read one byte from the I2C device, request more data from device

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readAck(void)
{
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
                while(!(TWCR & (1<<TWINT)));

   return TWDR;

}/* i2c_readAck */


/*************************************************************************
 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readNak(void)
{
                TWCR = (1<<TWINT) | (1<<TWEN);
                while(!(TWCR & (1<<TWINT)));

   return TWDR;

}/* i2c_readNak */
```

Mittauspöytäkirja

milwaukee	sensori	milwaukee	sensori	milwaukee	sensori	milwaukee	sensori	milwaukee	sensori	milwaukee	sensori	milwaukee	sensori	milwaukee	sensori
61.1	54.5	23.8	28.7	52.5	45.4	39.3	36.5	39	36.9	52.4	43.5	57.3	46	31.2	33.5
60.4	54.4	24.4	28.8	52.9	45.7	39.4	36.4	39.8	37.3	52.8	43.7	57.4	46.1	31.4	33.9
62.7	54	24.5	29.4	52.9	46.3	39.8	36.5	40.2	37.7	52.9	43.7	57.7	45.9	31.7	33.8
61.5	53.4	24.9	29.8	53.3	46.7	40.1	36.7	40.8	38.1	53.5	43.6	58.2	45.7	31.9	34.1
61.5	53.3	25.1	29.4	54.3	46.5	40.6	36.8	41.5	38.3	54	43.8	58.2	45.4	32.5	34.3
59.8	51.6	25.9	29.7	54.8	46.7	40.6	36.9	42.1	38.7	54.7	44	24.8	26.6	32.7	34.8
59.6	51.5	26.1	30.6	55.3	46.7	40.9	37.1	42.5	39.1	54.7	44	24.9	27.1	32.9	34.8
59.6	51.1	26.5	31	55.5	47	40.9	37.2	43.3	39.4	54.8	45.1	25	28	33.4	34.9
59.1	50.9	26.7	31.5	56.2	48	41.3	37.3	43.5	39.8	55.2	45.7	25.3	28.4	33.4	35.3
58.1	50.4	26.9	31.1	56.9	48.2	41.5	37.3	43.8	39.9	55.7	45.8	25.5	28.8	33.8	35.4
57.6	49.9	27.5	31.3	57.9	48.6	41.8	37.4	44.1	40.1	55.8	45.7	25.7	28.9	34.1	35.3
56.8	49.5	27.8	31.6	58.2	48.7	42.1	37.4	44.4	40.4	56.1	45.8	25.8	29	34.3	35.6
56.5	49.1	28	31.4	58.6	48.9	42.3	37.6	44.9	40.5	56.3	45.7	26.1	29.3	34.7	35.7
56.3	49	28	31.3	58.9	49.1	42.7	37.7	45.4	41	56.4	45.7	26.6	29.7	34.9	35.9
55.7	48.5	28.3	31.3	59.3	49	43	37.8	45.9	41.2	56.8	45.9	26.8	29.9	35.1	35.9
55.7	48.2	28.5	31.5	59.5	49.7	43.1	38	46.7	41.7	57.1	45.7	27.2	30.1	35.4	35.9
55.5	47.9	28.5	31.7	59.6	49.8	43.5	37.9	47	42.3	57.2	45.9	27.6	30.5	35.7	36.1
54.8	47.6	28.9	31.8	60	50.5	43.5	38.5	47.6	43	23.4	24.9	28	30.6	35.9	36.2
54.2	47.4	29	32	60.5	50.5	43	38.8	48.3	43.4	23.5	25.2	28.5	31	36.2	36.2
53.5	46.9	29.5	32.8	60.8	50.4	43	38.8	48.6	43.5	23.6	25.2	29.1	31.5	36.4	36.8
52.9	46.6	29.9	33	61	50.6	44.4	38.3	49	43.7	23.8	25.3	29.5	31.7	36.7	36.6
52.3	46.5	30	32.8	61	50.9	44.6	39	49.6	44.1	24	25.7	29.6	31.9	37.1	37.1
51.7	46.1	30.8	33	61.3	50.9	45	38.8	50.1	44.3	24.3	25.8	29.9	31.9	37.2	37
51.6	46	31	33.3	61.6	51.8	45.2	39	50.5	45.1	24.8	26	30.2	32.1	37.3	37
51.4	45.9	31.1	33.4	24.2	25.5	45.2	38.8	51	45.2	25	26	30.5	32.6	37.9	37.3
50.5	45.4	31.3	33.6	24.3	25.4	45.5	39.1	51.5	45.5	25.4	26.3	31	32.5	38	37.6
50.4	45.2	31.6	34	24.4	25.7	45.6	39.8	51.9	45.9	25.9	26.5	31.2	32.8	38.4	37.8
50.1	45	32	33.8	24.4	25.9	46.1	39.8	52.6	46	26.3	26.5	31.5	32.7	38.6	37.6
49.4	44.7	32.5	34.1	24.5	26.1	46.4	39.5	53.1	46.2	26.8	27	31.9	33.1	38.8	38
49.4	44.5	32.4	34.4	24.9	26.5	46.5	40.4	53.5	46.6	27.6	27	32.3	33.3	39.1	37.9
49.3	44.2	32.9	34.2	24.9	26.5	46.2	40.2	54	47	28	27	32.7	33.4	39.3	38
48.5	44	33.1	34.2	24.9	26.5	46.3	40.4	54.4	47.5	28.3	27.4	32.9	33.7	39.5	38.1
47.9	43.7	33.4	34.6	25.5	27	46.7	40.3	54.9	47.7	28.9	27.6	33.2	33.9	39.6	38.3
48.2	43.7	34.3	35.5	25.6	27.2	47.1	40.7	55.3	47.9	29.3	27.8	33.7	33.9	39.9	38.8
48	43.5	33.9	35.6	26.1	27.3	47.3	40.6	57.7	48.5	30	28.2	34	34.3	40.3	38.4
47.8	43.5	33.6	35.9	26.6	27.8	47.5	40.6	57.8	48.5	30.4	28.2	34.6	34.7	40.6	38.7
47.6	43.3	34.1	36.1	27	27.9	47.8	40.7	55.2	48.3	30.9	28.3	34.7	34.9	40.8	38.9
47.9	43	34.7	36.4	27.6	28.5	48.2	40.6	56.1	48.5	31.3	28.6	35.2	35	41	39
47	42.9	34.5	36.6	28	28.5	48.4	40.7	55.7	48.8	31.8	28.8	35.3	34.7	41.2	39.1
46.9	42.7	34.5	36.7	28.3	28.6	48.8	41	57.1	48.6	32.2	29.1	35.7	35.1	41.3	38.9
46.1	42.4	34.7	36.5	29.1	29.1	49.1	41	59.7	47.9	32.7	29.1	36	35.5	41.3	39.2
45.7	42.3	34.9	36.6	29.7	29.6	49.5	41.1	55.6	49.1	33.1	29.4	36.3	35.6	41.8	39.3
46	41.9	35.2	36.8	30.2	30.3	49.3	41.3	56.2	49.4	33.2	29.5	36.8	35.7	41.9	39.3
44.9	41.7	35.5	36.6	30.8	30.3	49.7	41.5	56.7	49.7	33.6	29.6	37.1	35.9	42.2	39.5
45.1	41.5	35.7	37	31.1	30.7	49.7	41.9	57	50.3	34	29.8	37.5	36.1	42.5	39.6
44	41.4	35.9	37	31.7	31	50	41.3	57.3	50.1	34.6	30.2	38.1	36.4	43	39.6
44.4	41.2	36.1	37.4	32.2	31.4	50.8	41.5	57.6	50	35.2	30.2	38.4	36.5	43.1	40
43.5	40.8	36.3	37.4	32.5	31.3	50.4	41.6	58	50.4	35.5	30.6	38.8	36.6	43.4	39.8
44.2	40.6	36.6	37.7	33.2	31.9	51.2	43	58.4	51.2	35.8	31.1	39	36.9	43.8	39.9
43.1	40.3	36.9	37.6	33.3	31.4	52.3	42.9	58.5	50.4	36.1	31.3	39.5	37.1	44	39.9
42.8	40.2	37.1	37.6	33.3	31.9	52.1	43.1	58.2	50.5	37	31	39.9	37.3	44.3	40.3
43.2	40.1	37.3	38	33.9	32.5	52.3	43.2	58.7	50.9	36.9	31.3	40.4	37.4	44.6	40.5
43.1	39.9	37.5	37.7	34.4	32.3	52.6	44.2	59	51.1	37.1	31.5	40.6	37.8	44.7	40.7
43.1	39.8	37.5	37.9	35	32.5	52.6	43.6	59.5	52.7	37.8	32.6	41	37.8	45	40.5
42.8	39.7	37.7	38.1	35.5	32.9	52.9	45.2	61	52.5	38.3	32.7	41.3	37.8	44.9	40.8
41.7	39.6	37.8	37.9	36.1	32.7	53.1	44.1	60.8	52.4	38	32.8	41.6	38	45.4	40.7
41.5	39.3	38.2	38.2	36.7	33.8	53.8	44	60.4	52.3	38.5	32.9	41.9	38.2	45.7	40.9
41.6	39.2	38.3	38.3	37.2	34	54	44.5	61	50.4	38.8	33.3	42.4	38.1	45.8	41
41.4	39	38.4	38.5	37.4	33.9	54.1	44.6	61.2	49.7	39.1	33.6	42.7	38.6	46	41.3
40.9	39.1	38.6	38.4	38	34	54.3	45.2	61.8	51	39.6	34.2	43	38.8	46.3	41.1
40.5	38.8	38.8	38.5	38.3	34.3	54.4	44.9	61.5	50	40.5	34	43.3	38.7	46.4	41.1
40.9	38.5	39	38.7	38.6	34.5	54.8	44.7	22.2	24	41.3	34.2	43.5	38.9	46.5	41.3
41.1	38.5	39.2	38.6	38.9	34.6	54.9	44.2	22.5	24.3	41.6	34.5	43.9	39.1	46.6	41.3
40	38.4	39.3	39	39.4	34.8	54.9	44.4	23.1	24.6	41.6	34.5	44	39.3	46.7	41.3
40.6	38.1	39.6	39.1	39.6	34.9	54.8	44.5	23.7	25	41.8	35.1	44.2	40	47.6	41.5
39.6	38.1	39.8	39.3	40.1	35.5	54.8	44.6	24.1	25.2	41.6	35.7	44.3	41	47.4	42
39.5	38.2	39.9	39.2	40.4	35.7	55.2	44.3	24.6	25.5	42.2	35.4	45.2	41.4	46	41.8
39.3	37.9	40.1	39.4	41.1	35.9	54.9	44.7	25	26.2	42.8	35.3	45.7	40.9	47.4	41.3
39.3	38	40.2	39.4	41.3	36.2	55.4	44.1	25.6	26.6	43.1	35.3	45.8	41.2	46.4	42.3
38.9	37.6	40.6	39.3	41.4	36.3	55.3	44.3	26.2	26.9	43.8	35.4	45.9	41.8	45.8	42.3
38.6	37.6	40.6	39.4	41.8	36.2	55.8	43.2	27.1	27.3	23.4	25.4	46.4	41.4	47.5	41.9
38.6	37.4	40.7	39.3	42.4	36.6	56.1	42.8	27.6	27.8	23.6	25.4	46.8	41.8	48.3	42.2
38.4	37.3	41.2	39.6	42.5	36.9	56.2	42.7	28.5	28.2	23.7	25.2	47.2	41.7	48.8	41.9
38.6	37.2	41.2	39.5	43	37.2	55.7	43.9	28.9	28.5	23.5	25.4	47.7	42.2	49	42
38.9	37	41.8	39.8	43.7	37.7	55.9	43.6	29.6	28.7	23.7	25.5	47.8	42	48.7	42
38	36.8	41.9	39.6	44.1	37.5	56.2	43.6	30.1	29.2	23.6	26	48.2	42.2	49.2	42.4
38.1	36.8	42.2	39.8	44.5	38.2	56.4	43.7	30.7	29.5	23.7	26.2	48.5	42.3	49.4	42.3
38.2	36.5	42.3	40	45	38.6	56.7	43.2	31.3	30	24	26.6	48.8	42.7	49.8	42.6
37.4	36.4	42.5	40.1	45.4	38.6	56.7	43.3	31.7	30.2	24.5	27.2	49	42.4	49.7	42.5
37.6	36.4	42.9	40.5	45.5	38.4	56.8	43.2	31.9	30.2	25.1	27.9	49.7	43	50.1	42.8
37.2	36.3	43.2	40.7	46.1	38.8	56.9	43.6	32.4	30.7	25.3	27.9	49.7	43.3	50.1	42.3
36.9	36.1	43.5	40.6	46.3	38.8	57.5	44.3	32.5	31.1	25.6	28.2	50.1	43.6	50.6	42.8
36.1	35.8	43.9	40.8	47	39.2	57.4	44.9	33	31.1	25.9	28.5	50	43.6	50.8	42.8
35.8	36	44	40.8	47.3	39.1	58.3	47.5	33.6	31.4	26.3	28.4	50.2	43.8	50.9	42.8
35.9	36	44.1	40.8	48	39.9	58.5	47.1	34.1	31.8	26.6	28.7	50.4	44.1	51.1	42.8
36.3	36	44.3	40.8	48.5	40.3	58.7	47.3	34.1	32	27.2	29.3	50.8	43.4	51.2	43
36.2	35.9	44.3	41.1	49	40.4	59.3	46.9	34.7	32.5	27.6	29.3	51	43.8	51.5	43
35.5	35.7	44.7	41	49.2	40.5	59.4	47.2	35	32.3	27.8	29.4	51.4	44.1	51.5	43.4
36.1	35.4	44.8	40.8	49.8	40.5	59.4	47.3	35.2	32.8	28.1	29.6	51.5	44.6	51.6	43.3
36.1	35.4	45.2	41.2	48.6	41.7	59.5	47.1	35.9	33.3	28.1	29.9	52.1	45	51.4	43.6
35.5	35.2	45.3	41.2	48.8	41.9	58.1	47.4	36.2	33.5	28.3	30	52.7	45.1	51.2	43.4
35.4	35	45.5	41.1	49.1	41.7	59	47.3	36.5	33.6	28.9	30.2	53	45.2	52.5	43.6
34.8	34.7	45.9	41.5	49.3	41.9	59.3	47.7	37	34	29.6	30.6	53.4	45.1	52.6	43.6
34.7	34.8	46	41.4	48.8	42.5	59.8	47.6	37.2	34.2	29.9	30.4	53.8	45.5	52.7	43.8
35.4	34.6	46.2	41.9	49.6	42.3	60.2	47.5	37.8	34.6	30	30.8	54	45.6	52.7	43.8
35.4	34.3	46.2	42	50.2	42.5	60.1	47.7	38.3	35	30.4	30.8	54.1	45.8	52.7	44.2
34.4	34.2	46.3	41.9	50.6	42.6	60	47.5	38.7	35.2	30.7	30.8	54.4	45.6	53	44.1
34.4	34.2	46.6	41.8	51	42.8	60.1	47.6	39.2	35.7	31	31.1	54.7	46.1	52.4	43.9
34.1	34	46.7	42	51.1	43	60.6	47.5	39.8	36.1	31.2	31.5	54.9	46	53.2	44.1
34	33.9	46.8	42.2	51.4	43.2	61	47.6	40.2	36.5	31.4	31.5	55.2	46.6	53.3	44.3
33.8	34	47	42.4	51.6	43.5	61.2	47.8	40.5	36.8	31.7	31.6	55.3	46.2	53.1	44.3
34.8	33.7	47.2	41.9	52.9	43.6	61.3	47.9	41.1	37	31.9	31.8	55.5	46.6	53.1	44.2
25.8	27.1	47.4	42.1	52.8	43.6	61.2	47.7	41.5	37.3	32.3	32.1	55.8	46.9	53	44.2
26.1	27.4	47.6	42.3	52.7	44	28.5	28.4	42	37.6	32.9	32.1	55.9	46.7	53.1	44
27	27.8	48	42.2	53	44.2	26.1	27.5	42.3	38	33.3	32.4	56.5	46.6	54.8	44.1
27.4	28.4	48.3	42.3	52.3	44.5	26.4	28.5	42.7	38.3	33.5	32.7	56.9	46.6	54.6	44.1
27.7	28.8	48.5	42.4	52.5	44.5	26.4	29	43.1	38.5	33.8	32.4	57	46.8	54.3	44.1
28.4	29.2	48.6	42.4	53.2	44.3	26.3	29.6	43.6	39	33.9	32.9	57.4	46.7	55.4	44.1
31.7	31.8	49	42.4	53.5	44.9	26.6	30.2	44.3	39.3	34.1	32.8	57.6	46.9	55	44.3
32.3	32.1	49.1	42.6	53.5	45.1	27.1	30.4	44.9	39.6	34.2	33	58	47.2	55.4	44.4
32.7	32.2	49.3	42.7	54.1	45.2	27.3	30.5	45.7	39.9	34.6	33.4	58.2	48.2	55.5	44.2
33.1	32.5	49.3	42.7	54.6	45.1	27.6	30.7	46.1	40.2	34.8	33.6	58.3	49	55.5	44.4
33.8	32.8	49.5	42.7	54.9	45.6	27.8	31.6	46.3	40.4	35.2	33.4	58.7	49	55.7	44.6
34.2	33.3	49.7	43.2	55.1	46.1	28.1	31.6	46.6	40.6	35.5	33.9	59.3	49.1	56.6	43.8
34.6	33.9	50.1	43.1	55.3	46.2	28.6	31.9	46.8	41	35.8	33.6	60	48.8	56	43.5
35.2	34.3	50	43.9	55.2	46.4	28.7	31.9	47.4	41.3	36.2	34.1	60.4	49.5	58.1	44
35.9	34.3	51.6	44.2	55.8	46.2	29	32.2	48.1	42.3	36.5	34.2	60.6	49.4	58.5	44.1
36.2	34.7	50.8	44	55.8	46.2	29.2	32.5	49.2	42.6	36.8	34.8	29	33.5	58.7	44
36.8	35.1	51.2	43.6	56.1	46.4	29.6	32.6	49.8	42.8	37.4	34.8	29.2	33.6	59	44
37.2	35.3	51.4	43.6	56.3	46.2	29.9	32.8	50.1	43	37.8	35.3	23	28.5	59.1	44.2
37.4	35.8	51.6	43.4	56.6	46.3	31.1	33	50.6	43.3	37.8	35.5	57.7	44.5	59.2	44
38	35.8	51.8	43.6	56.7	46.2	30.9	33.3	50.9	43.5	38.1	35.3	57.2	44.7	59.3	44.4
38.5	36.2	52.1	43.8	57	46.1	30.8	33.3	51	43.8	38.2	35.6	51.6	44.7	38.7	35.8
				52.2	45.1	39.2	36	51.2	43.9	38.5	35.5	51.9	45	39.1	35.6