KANDIDAATINTYÖ

Mitja Kärki

Ohjaaja: Timo Rahkonen

SÄHKÖTEKNIIKAN TUTKINTO-OHJELMA

2021

Kärki MA. (2021) Kandidaatintyö. Oulun yliopisto, Elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 18 s.

TIIVISTELMÄ

Tämän työn sisältö painottuu TMP117-lämpötilasensorin ohjelmistokirjaston kirjoittamiseen. Kirjasto kirjoitetaan sulautettujen järjestelmien kehittämiseen tarkoitetulle, avointa lähdekoodia hyödyntävälle Arduino-kehitysalustalle. Työssä käydään läpi lisäksi Arduino-ohjelmistokirjastojen yleinen rakenne ja esitellään Arduino-ekosysteemi. Komponentin käyttöönotto, tärkeimmät ominaisuudet ja sen kytkentä esitetään sanallisesti ja kuvallisesti. Tuotettua lähdekoodia selitetään Arduino-mikrokontrollerin ja lämpötilasensorin välisen yhteyden sekä kirjastoon toteutettujen ominaisuuksien osalta.

Lopuksi esitetään työssä saavutettuja tuloksia luodun esimerkkiohjelman suorituksen ja lähdekoodin kautta. Tuotettu sensorin perusominaisuudet toteuttava lähdekoodi on saatavilla kehitysympäristön periaatteen mukaisesti avoimena lähdekoodina.

Avainsanat: TMP117, Arduino, sulautetut järjestelmät, lämpötilasensori.

Kärki MA. (2021) Bachelor's Thesis. University of Oulu, Degree Program in Electrical Engineering, Bachelor's Thesis, 18 p.

ABSTRACT

This Bachelor's Thesis describes the process of writing a software library for the temperature sensor TMP117. The library is specifically written for the open source embedded system development platform Arduino. In addition, the platform and the general structure of the software libraries for it are briefly introduced. The temperature sensor's main features and deployment are also introduced along with the used schematic. Furthermore, the developed software is reviewed regarding the implemented sensor features and the communication between the Arduino board and the sensor.

An example program is created to showcase the software library. Its execution is shown using a screen capture of the output and a code snippet. The whole software library is available as open source respecting the development platform's philosophy.

Key words: TMP117, Arduino, embedded systems, temperature sensor.

SISÄLLYS

1. JOHDANTO

Avointa lähdekoodia hyödyntävässä sulautetun elektroniikan Arduino-kehitysympäristössä on tarjolla laaja valikoima ohjelmistokirjastoja ja ajureita erilaisille laitteille ja komponenteille. Laiteajurien tehtävä on toimia rajapintana Arduino-mikrokontrollerin ja oheislaitteen välillä helpottaen kehittäjän työtä.

Kynnys kehitysympäristön käyttöön on matala, joten se kannustaa ohjelmointitaidoiltaan varsin kokemattomiakin harrastelijoita luomaan projekteja sen suuremmin asiaan syventymättä. Vaikka käyttökokemuksen helppous on toki hyvä asia, aiheuttaa se sen, ettei harrastajista moni tule perehtyneeksi mikrokontrollereiden tai niihin liitettyjen laitteiden ohjaukseen kovinkaan syvällisesti edes ohjelmointitaitojen karttuessa.

Tämän opinnäytetyön tarkoituksena oli perehtyä tarkemmin Arduino-kehitysympäristön laiteajurien toimintaan luomalla sellainen Texas Instrumentsin TMP117-lämpötila-anturille. Laadittu laiteajuri kattaa työn luonteesta johtuen vain anturin tärkeimmät toiminnallisuudet.

2. LAITEAJURIN LUOMINEN ARDUINO-KEHITYSALUSTALLE

2.1. Arduino-kehitysympäristö

Etenkin elektroniikkaharrastelijoiden parissa tunnettu Arduino-kehitysympäristö on suosittu avoimuutensa, helppokäyttöisyytensä ja edullisuutensa takia. Harrastelijoiden lisäksi Arduinoa käytetään myös teollisuudessa ja opetuskäytössä, sillä markkinoilla olevien Arduino-tuotteiden laaja kirjo mahdollistaa monenlaiset sovellutukset [1].

Ekosysteemi koostuu tietokoneelle asennettavasta ohjelmointiympäristöstä (Arduino IDE) ja enimmäkseen Atmel-mikroprosessoreihin pohjautuvista piirilevyistä, joista molempien lähdetiedostot ovat avoimesti saatavilla. Valmiiksi käännettyä ohjelmistoa on saatavilla ilmaisena latauksena ja valmiita laitteita joidenkin kymmenien eurojen hintaan. Kolmannen osapuolen valmistamia Arduino-yhteensopivia klooneja voi ostaa vain muutamilla euroilla.

Arduino-piirilevyillä on mikrokontrollerin lisäksi FTDI-piiri, jolloin mikrokontrollerin ohjelmointi onnistuu USB-väylää käyttäen, eikä erillistä ohjelmointilaitetta tarvita. Tämä lisäys edelleen madaltaa kynnystä uusille käyttäjille.

Kehitysympäristön helppokäyttöisyys perustuu myös avoimeen lähdekoodiin. Erilaisia Arduinolla toteutettuja projekteja ja niiden ohjeita lähdekoodeineen on runsaasti saatavilla virallisen jälleenmyyjän lisäksi muun muassa foorumeilta, asiaan vihkiytyneiltä nettisivustoilta ja oheislaitteiden myyjiltä. Myös ohjelmointiympäristön mukana toimitetaan useita ohjelmistokirjastoja ja niiden käyttöesimerkkejä.

Arduino-ohjelmointikieli pohjautuu enimmäkseen C- ja C++ kieliin, jotka ovat suosittuja ja soveltuvia laiteläheiseen ohjelmointiin. Ohjelmatiedostoja kutsutaan kehitysympäristössä nimellä *Sketch*. Ainoat kääntäjän erityisvaatimukset Arduino IDE:n .ino-päätteiselle ohjelmatiedostolle ovat setup- ja loop-funktiot. Vaatimusten täyttyessä kehitysympäristö hyväksyy standardien mukaista C- tai C++ -koodia.

2.2. Ajurien rakenne

Ohjelmistokirjaston tehtävä yleisesti on luoda sen käyttäjälle eli ohjelmistokehittäjälle abstraktiotaso, joka piilottaa matalan tason toimenpiteet yhdistäen ne helppokäyttöiseksi kokonaisuudeksi. Tässä työssä kirjasto esimerkiksi sisällyttää yksittäisiin funktioihin kaikki yhdelle operaatiolle tarvittavat tiedonsiirtoväylän käskyt; loppukäyttäjän ei tarvitse perehtyä tiedonsiirtoprotokollan viestin aloitus-, viesti- ja lopetuskäskyihin, vaan funktio hoitaa viestinnän kokonaisuudessaan.

Arduino-kehitysympäristön helppokäyttöisyyden edistämiseksi tulee etenkin loppukäyttäjälle näkyvä rajapinta nimetä kansankielisesti ja olettaen käyttäjän vähäinen kokemus ohjelmoinnista ja sulautetusta elektroniikasta yleensä.

Arduino-kehitysympäristön laiteajuri koostuu vähintäänkin kahdesta, mutta yleensä useammasta tiedostosta. Kirjaston käyttäjälle on lisäksi suotavaa liittää mukaan käyttöä helpottavaa dokumentaatiota kuten esimerkkikytkentöjä ja -koodia sekä avainsanoja [3]. Laiteajuri toimitetaan aina lähdekoodina, mieluiten .zip-

pakattuna tiedostona. Ohjelmointiympäristö tukee graafisen käyttöliittymänsä kautta kyseiseen tiedostomuotoon pakattujen ohjelmistokirjastojen asennusta.

Minimivaatimukset toimivalle laiteajurille ovat .c- tai .cpp-päätteinen tiedosto, jossa varsinainen koodi sijaitsee, sekä .h-päätteinen otsikkotiedosto (engl. *header*), jossa esitellään funktiot, luokat ja objektit. Tuettuja ohjelmointikieliä laiteajurille ovat siis C ja C++. Tässä työssä kyseisten tiedostojen nimet ovat *tmp117.cpp* ja *tmp117.h*.

Arduinon kirjastonkirjoitusohjeen mukaan lähdekooditiedostojen lisäksi ohjelmistokirjastoon on hyvä liittää myös *keywords.txt*- ja esimerkkikooditiedostot sekä muuta mahdollista dokumentointia kuten kaaviot tai kuvat. Keywords.txt-tiedostoon määritellään kirjastossa käytetyt avainsanat kuten funktioiden nimet, jolloin ohjelmointiympäristö osaa värittää koodin luettavuuden parantamikseksi (engl. *syntax highlighting*). Lisäksi Arduinon kirjastomäärittelyn versiosta 2.2 eteenpäin kirjastossa voi olla *library.properties* -metadatatiedosto. Tiedostoon määritellään muun muassa kirjaston luojan nimi, lyhyt kuvaus ohjelmiston toiminnasta sekä mahdolliset ohjelmiston kääntämiseen vaadittavat ohjelmistoriippuvuudet. Satunnaiset, Arduinon ohjeistuksessa määrittelemättömät tiedostot kirjaston käytön tueksi tulee sijoittaa hakemistoon *extras*. Tässä työssä sen sisältö on kytkentäkaavio.

Esimerkkikoodi on ohjelmistoympäristössä käytettävä .ino-päätteinen tiedosto, jonka tarkoitus on saada käyttäjä helpommin alkuun ohjelmistokirjaston käytössä. Tässä työssä esimerkkitiedosto on nimeltään *tmp117.ino* ja sijaitsee ohjeistuksen mukaisesti polussa *tmp117/examples/*. Lisäksi työssä luotuun kirjastoon kuuluvat kirjaston tiedostorakenteen juuressa sijaitsevat *keywords.txt* ja *library.properties*. Kokonaisuudessaan kirjaston tiedostorakenne on siis seuraavanlainen:

```
tmp117/
tmp117/src
tmp117/src/tmp117.h
tmp117/src/tmp117.cpp
tmp117/library.properties
tmp117/keywords.txt
tmp117/extras
tmp117/extras/circuit_diagram.png
tmp117/readme.txt
tmp117/examples
tmp117/examples/tmp117.ino.
```

3. TEXAS INSTRUMENTS TMP117

3.1. Ominaisuudet

Työn ohjelmistokirjasto on kirjoitettu Texas Instruments:n TMP117-lämpötilasensorille. Sensori on digitaalinen ja pienimpään lämpötilaskaalaan asetettuna kykenee 16 bitin mittaustuloksellaan 0,0078125 celsiusasteen resoluutioon ja 0,1 celsiusasteen mittaustarkkuuteen [4].

Laitteessa on yhteensä kymmenen 16-bittistä rekisteriä, joita käytetään laitteen asetusten (esimerkiksi hälytyslämpötilat ja keskiarvoistus) määrittämiseen ja mittaustulosten tallentamiseen. Rekisterien kautta käytettävissä ovat myös kolme 16-bittistä EEPROM-aluetta, joihin voi tallentaa joko käynnistyksen yhteydessä ladattavia asetusarvoja tai yleiskäyttöistä tietoa.

Komponentin käyttöjännite on 1,8 – 5,5 volttia, joten se soveltuu helposti käytettäväksi ilman jännitteenmuunnoksia viiden voltin käyttöjännitteellä toimivan Arduino-mikrokontrollerin kanssa. WSON-koteloituna (engl. sanoista *Very-very-thin small-outline no-lead package*) TMP117:n ulkomitat ovat 2 mm * 2 mm ja piiri kuluttaa käytössä virtaa vain 3,5 µA, joten se soveltuu hyvin esimerkiksi mobiili- ja puettavan elektroniikan sovellutuksiin.

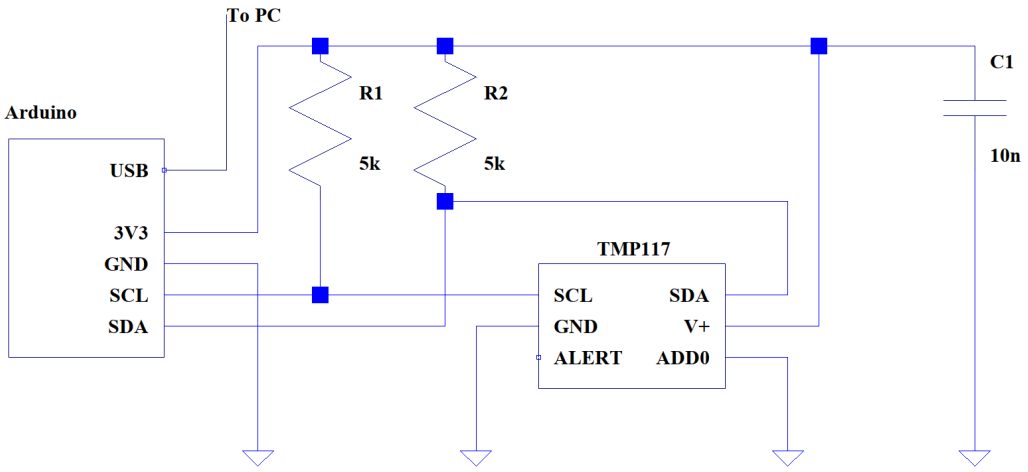
Sensorin tiedonsiirto tapahtuu joko yleisesti käytetyn I²C-väylän tai siitä johdetun SMBus-väylän kautta. Tässä työssä sensorin kanssa kommunikointi on toteutettu I²C-väylän kautta käyttämällä Arduinon Wire-kirjastoa.

3.2. Käyttöönotto

Tässä työssä sensori yhdistettiin Atmelin Atmega32U4-suorittimeen perustuvaan Arduino Micro -laitteeseen [5]. I²C-väylällä isäntälaitteena (engl. *master*) toimii mikrokontrolleri ja orjana (engl. *slave*) sensori.

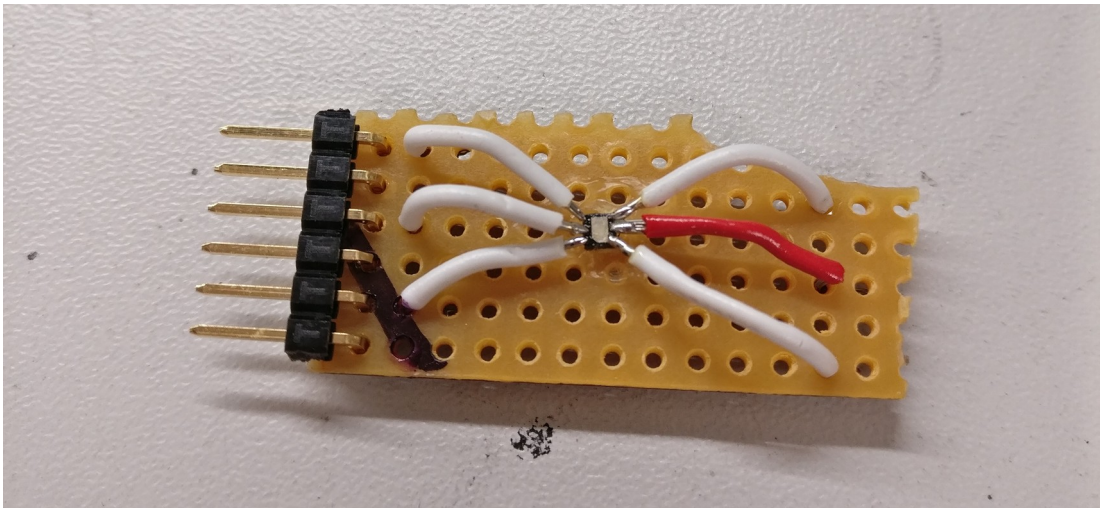
Sensorin kytkentä isäntälaitteeseen vaatii joitain yleisiä passiivikomponentteja toimiakseen. Koko piirin kytkennästä luotiin Arduinon aloittelijoillekin sopiva yksinkertaistettu kaavio LTSpice-piirisimulaatio-ohjelmalla (kuva 1). Kaavio liitettiin ohjelmistokirjastoon dokumentoinniksi ja kytkentäesimerkiksi. Kytkentä mukailee komponenttivalmistajan datalehdessä esittämää esimerkkikytkentää pienin muutoksin: ylösvetovastusten arvot pidettiin samoina, mutta käyttöjännitteen ja maatason välille kytkettiin lisäksi 10 nF suotokondensaattori. Käytetyn mikrokontrollerilaitteen SDA- ja SCL-pinnit eli I²C-väylän data- ja kellopinnit ovat piirilevyllä numerot 2 ja 3, joten sensorin vastaavat pinnit johdettiin niihin [6].

Sensorin käyttöjännite johdettiin Arduinon 3,3 voltin ulostulosta ja laitteen ja lämpötila-anturin maatasot kytkettiin yhteen. Koska Arduinon käyttöjännite saadaan USB-väylästä, ei erillistä jännitelähdettä ole kuvaan 1 merkitty. ALERT- eli hälytyspinnille ei ole tässä työssä käyttöä, joten se jätettiin kokonaan kytkemättä eli kellumaan. Hälytysominaisuus on tästä huolimatta käytössä I²C-väylän kautta luettavista hälytysrekistereistä.



Kuva 1. Yksinkertaistettu kytkentäkaavio isäntälaitteen ja sensorin välillä.

Pienen kokonsa takia sensorikomponentille tuli ensin rakentaa nauhakuparoidusta kytkentäpiirilevystä koekytkentälevyyn sopiva moduuli, joka on esitetty kuvassa 2. Kuvassa 1 esitetyt passiivikomponentit sekä moduuli kytkettiin koekytkentälevyyn. Sensorin pienen fyysisen koon takia moduulin rakennus käsin osoittautui haasteelliseksi – suositeltavaa olisi valmistaa komponentille piirilevy tai käyttää valmista kaupallisesti saatavilla olevaa moduulia.



Kuva 2. Sensori juotettuna kytkentäpiirilevyyn. Punainen johdin osoittaa käyttöjännitepinnin.

4. AJURI

4.1. Ominaisuudet

Vaikka työssä käytetty lämpötilasensori tukee lukuisia mittaustapoja ja -asetuksia kuten keskiarvoistustilan muuttamisen, tämän työn tarkoituksena ei ollut toteuttaa niitä kaikkien käyttöä, vaan perehtyä Arduino-ohjelmistokirjaston luontiin yleisesti. Lämpötilasensorin ominaisuuksista ohjelmistokirjasto toteuttaakin kolme perustoimintoa: lämpötilamittaus, hälytysrajojen asetus ja hälytystilan lukeminen. Koska kommunikointi sensorin kanssa tapahtuu samankaltaisilla käskyillä ominaisuudesta riippumatta, olisi muidenkin ominaisuuksien toteuttaminen kohtuullisen helppoa. Ohjelmistokirjaston tekstimuotoiset osat ovat ilmenevät kokonaisuudessaan liitteistä.

4.2 Osat

Työn ohjelmistokirjasto on laadittu Arduinon kirjoitushetkellä uusimman kirjastomäärittelyn (versio 2.2) mukaisesti ilman yhteensopivuutta vanhoihin versioihin, joissa vaadittu tiedostorakenne poikkeaa uusimmasta.

Kirjaston rakentaminen alkoi ensin luomalla oikeanlainen hakemisto- ja tiedostorakenne tyhjillä tiedostoilla. Tyhjiin tiedostoja lähdettiin täyttämään alkaen otsikko- ja lähdekooditiedostoista. Kun alustava toiminnallisuus oli toteutettu, luotiin sille testi eli esimerkkitiedosto. Lopullisen muotonsa ohjelmistokirjasto sai muutaman pääiteraation tuloksena, jonka jälkeen vaadittiin vielä useita pienempiä korjauksia ja optimointeja.

Lähdekoodi ja muut kirjastoon sisältyvät osat on kirjoitettu ja kommentoitu englanniksi, koska kirjasto julkaistaan Arduino-kehitysympäristön kaltaisesti avoimena lähdekoodina ohjelmistojen versionhallintaan tarkoitetulla GitHub-verkkosivustolla työn tekijän henkilökohtaisella käyttäjätilillä, jonka URL-osoite ilmenee liiteluettelosta. Näin työssä luotu ohjelmisto tavoittaa mahdollisimman suuren yleisön ja puolestaan tuottaa sisältöä Arduino-ekosysteemiin.

4.2.1 Otsikkotiedosto

Otsikkotiedostossa määritellään luokka *tmp117*, joka puolestaan jaetaan kahteen alaosioon: *public* eli julkiset, ohjelmistokirjastolle sekä sen käyttäjälle näkyvät jäsenet ja *private* eli vain ohjelmistokirjastolle itselleen käytettävissä olevat yksityiset jäsenet. Otsikkotiedostossa nimensä mukaisesti määritellään vain jäsenten nimet ja mahdolliset alkuarvot, ei niiden toiminnallisuutta.

Kirjastoissa käytäntönä on määrittää julkisista jäsenistä ensimmäisenä alustusfunktio (engl. *constructor*). Muita julkisia jäseniä ovat funktiot laitteen toiminnoille, aloitusfunktio *begin* ja vakiot *readAlert*-funktion palautusarvoille.

Laitteen tärkein ominaisuus eli lämpötilamittaus esitellään prototyypillä *double readTemperature(void);*. Koska lämpötilan lukeminen tapahtuu aina samalla

lailla eli samasta laitteen rekisteristä, ei funktiolla ole parametreja (*void*). Lämpötilarekisterin sisältö palautetaan käyttäjälle kaksinkertaisen tarkkuuden liukulukuna (*double*).

Hälytysrajat asetetaan niille varattuihin rekistereihin prototyypin *bool setAlertLimits(double low, double high);* mukaisesti. Funktion kaksi pakollista parametria ovat ala- ja ylärajan lämpötilat celsiusasteina. Paluuarvona käyttäjä saa totuusarvon (*bool*) rekisterien kirjoittamisen onnistumisesta – vain molempien rekisterien kirjoitusten onnistuminen palauttaa arvon *tosi*.

Kolmannen rajapintafunktion määrittelee prototyyppi *uint8_t readAlert(void);*. Kuten lämpötilan lukeminen, ei tämäkään funktio vaadi parametreja. Etumerkittömässä tavun kokoisessa kokonaislukumuuttujassa (*uint8_t*) palautetaan yksi kolmesta niin ikään julkisesti määritellystä paluuarvosta. Paluuarvo kertoo käyttälle hälytyksen tilan, joka on joko ei hälytystä, korkea lämpötila tai matala lämpötila (*noAlert*, *highAlert* tai *lowAlert*). Näiden paluuarvojen tarkempi määrittely ilmenee liitteestä 1.

Yksityisiä ovat matalan tason funktiot I²C-väylän ohjaamiseen (*i2cReadData* ja *i2cWriteData*) ja sensorin ominaisuuksia kuvaavat vakiot eli sellaiset elementit jotka pyritään piilottamaan ohjelmistokirjaston luomasta abstraktiotasosta. Loppukäyttäjän kannalta on yhdentekevää esimerkiksi sensorin sisäisten rekisterien osoitteet – käyttäjä on kiinnostunut vain rekisterien sisällöstä eli lämpötilalukemasta.

4.2.2 Lähdekooditiedosto

Tiedostossa *tmp117.cpp* toteutetaan otsikkotiedostossa nimettyjen sekä yksityisten että julkisten funktioiden toiminnallisuus. Tyhjän alustusfunktion jälkeen määritellään aloitusfunktio *begin*: funktio aloittaa I²C-liikenteen sensorin kanssa annetussa väylän osoitteessa ja kirjoittaa sensorin asetusrekisteriin oletusasetukset (jotka ovat eritelty otsikkotiedoston bittijonossa *defaultConfig*). Oletusasetukset ovat seuraavanlaiset: yhden sekunnin mittaussykli, kahdeksan näytteen keskiarvoistus ja hälytystila kytketty päälle. Asetusrekisterin selite löytyy datalehden taulukosta 6 (*Configuration Register Field Descriptions*).

Lämpötilan lukemiseen ja hälytysrajojen asettamiseen käytetyt funktiot ovat oikeastaan vain käskyjä suorittaa oikeanlainen sekvenssi tiedonsiirtoväylällä laitteen rekisterien lukemiseksi tai kirjoittamiseksi. Lisäksi ne kertovat tai jakavat kymmenjärjestelmässä määritellyn lämpötilan sensorin pienimmän merkitsevän luvun tarkkuudella eli resoluutiolla.

Hälytystilan tarkistusfunktiossa luetaan asetusrekisterin sisältö ja tarkistetaan bittioperaatioilla hälytyslippubittien tilat. Funktion palautusarvo on aina jokin kolmesta otsikkotiedostossa määritellystä hälytystilasta.

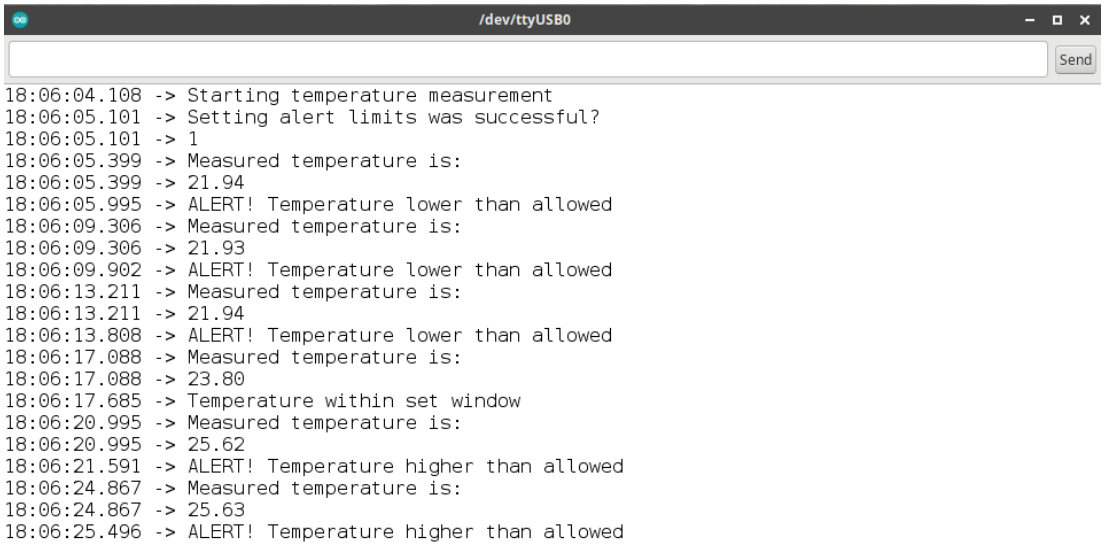
Yksityiset funktiot *i2cWriteData* ja *i2cReadData* käskyttävät sensoria yleisen I²C-standardin käskyjen mukaisesti [8]. Esimerkiksi ohjelmaleikkeessä 1 esitetyt tarvittavat *START*- ja *STOP*-käskyt on sisällytetty käytettyyn Wire-kirjastoon funktioina *beginTransmission* ja *endTransmission*. Koska väylällä siirrettävä paketti on aina kooltaan yksi tavu, tulee sensorin kaksitavuiset datapaketit siirtää kahdessa osassa. Lähetys sensorille tapahtuu siis seuraavassa järjestyksessä: aloituskäsky, kirjoitettavan rekisterin osoite, datapaketin merkitsevin osa, datapaketin vähiten merkitsevä osa ja viimeiseksi lopetuskäsky.

```
// Writes a 16-bit packet of data to a register on the
tmp117
  bool   tmp117::i2cWriteData(uint8_t   registerAddress,
uint16_t data)
  {
  // Counter for checking that the right amount of bytes
were written
  uint8_t byteCounter = 0;
  // First split the 16-bit data into two 8-bit variables
  uint8_t highByte = data >> 8;
  uint8_t lowByte = data & 0xff;
  Wire.beginTransmission(tmp117Address);
  Wire.write(registerAddress);
  // Write the actual data and check that right amount of
bytes were written
  byteCounter = byteCounter + Wire.write(highByte);
  byteCounter = byteCounter + Wire.write(lowByte);
  Wire.endTransmission();
  delay(2);
  // Successful write
  if (byteCounter == byteAmount)
  {
  return true;
  }
  // Unsuccessful write
  else {
  return false;
  }
}
```

Ohjelmaleike 1: funktio i2cWriteData

4.2.3 Esimerkkikoodi

Esimerkkikoodissa on TMP117-piirin avulla toteutettu yksinkertainen lämpövahti, jonka eri tiloja voi tarkkailla huoneenlämmössä esimerkiksi puhaltamalla lämmintä ilmaa sensoria kohti. Ilmoitukset vahdin tilasta ohjelma tulostaa noin kolmen sekunnin välein Arduino IDE:n sarjapäätteelle (engl. *serial monitor*). Kuvakaappaus sarjapäätteelle tulostuvista viesteistä on esitetty kuvassa 3. Kuvan ensimmäiset kolme riviä kertovat järjestelmän alustuksesta, jonka jälkeen siirrytään lämpötilamittaukseen. Lämpötilan kohotessa päätteelle tulostuu hälytystilan kaikki kolme mahdollista arvoa.



Kuva 3. Kuvakaappaus sarjapäätteestä.

Luokasta tmp117 luodaan ilmentymä eli olio nimellä *thermometer*. Arduinon syntaksin mukaisesti *Setup*-funktio suoritetaan vain kerran ohjelman aluksi. Siinä käynnistetään sarjapääteyhteys sekä yhdistetään lämpötilasensori I²C-väylään. Lopuksi asetetaan lämpövahdin lämpötilaikkunan rajat. Kirjaston sekä *begin*- että *setAlertLimits*-funktioiden palautusarvot kertovat kyseisen operaation onnistumisesta mahdollisien vikatilanteiden selvittämisen helpottamiseksi.

Ohjelman manuaaliseen keskeyttämiseen asti suoritettavassa *loop*-funktiossa tapahtuu varsinainen mittaaminen: lämpötilarekisterin sisältö luetaan ja tulostetaan sarjapäätteelle. Lopuksi luetaan hälytysrekisterin arvo ja tulostetaan se tulkattuna käyttäjälle. Vaikkei perinteistä toistorakennetta olekaan näkyvissä, Arduinon käytännön mukaisesti *loop* (Ohjelmaleike 2) suoritetaan aina uudelleen, tässä tapauksessa määritellyn kolmen sekunnin viiveen jälkeen. Suoritus päättyy siis vasta kun laitteen käyttöjännite katkaistaan.

```
  void loop() {
  // loop runs repeatedly
  // read and print temperature
  double temp = thermometer.readTemperature();
  Serial.println("Measured temperature is:");
  Serial.println(temp);
  // read and print alert status
  uint8_t alert = thermometer.readAlert();
  if (alert == thermometer.noAlert) {
  Serial.println("Temperature within set window");
  }
  else if (alert == thermometer.highAlert) {
  Serial.println("ALERT!    Temperature    higher    than
allowed");
  }
  else if (alert == thermometer.lowAlert) {
  Serial.println("ALERT!    Temperature    lower    than
allowed");
  }
  else {
  Serial.println("Reading alert status went wrong");
  Serial.println(alert);
  }
  delay(3000);
  }
```

Ohjelmaleike 2. Esimerkkikoodin funktio loop

Kokonaisuutena esimerkkikoodi on helposti laajennettavissa monimutkaisempiin kokonaisuuksiin, sillä se demonstroi käyttäjälle kaikkien julkisten objektien käytön – käyttäjälle ei jää välttämätöntä tarvetta lukea dokumentointia kirjaston ominaisuuksista.

Ohjelmistokirjaston tuettujen ominaisuuksien esittelemisen lisäksi esimerkkikoodi toimii testinä sekä ohjelmistokirjastolle itselleen sekä piirin kytkennälle: esimerkiksi virheellinen osoitevalintapinnin ADD0 kytkentä keskeyttää suorituksen ja tulostaa sarjapäätteelle ilmoituksen.

5. POHDINTA

Työlle asetetut tavoitteet Arduino-ohjelmistokirjaston luomisesta ja sen toiminnallisuudesta saavutettiin. Yleensä tieteellisissä julkaisuissa kuten kandidaatintöissä oletetaan lukijan olevan perillä alan perussanastosta ja -käsitteistä. Kuitenkin Arduino-kehitysympäristön käyttäjät usein vasta keräävät näitä tietoja käyttämällä kehitysympäristöä, jolloin ohjelmistolta vaaditaan yksinkertaista ja selkeää, mutta puolestaan edistyneempiä käyttäjiä varten tarkkaa dokumentointia. Etenkin työn ohjelmisto-osuutta kirjoittaessa tulikin ottaa huomioon käyttäjäkunnan taitojen kirjo kommentoimalla erityisesti esimerkkikoodi hyvin.

Koska tämä työ ei ollut osa mitään suurempaa projektia eikä mitään tiettyä tarvetta sensorin käytölle ollut, sensorin kaikkia ominaisuuksia ei ole työssä toteutettu. Luonteva jatkokehityskohde ohjelmistokirjastolle olisi loppujen ominaisuuksien toteuttaminen.

Työn parasta antia oli ei niinkään raaka C++-syntaksin opettelu, vaan kirjastokokonaisuuden luominen. Ohjelmointiympäristön yksinkertaisuus helpottaa aloittelijaa oppimaan uutta, mutta hieman kehittyneemmälle käyttäjälle yksinkertaisuus voi luoda oppimiseen rajoitteita. Esimerkkinä rajoittavasta yksinkertaisuudesta on ohjelmointiympäristöön sisällytetty kirjastonhallintatyökalu (*Library Manager*), jolla voi asentaa graafisen käyttöliittymän kautta ohjelmistokirjastoja Internetistä muutamalla hiiren napsautuksella. Kirjastojen hallinta ja rakenne näyttäytyy niin sanottuna mustana laatikkona piilottaen kirjaston toteutuksen.

Työ oli samalla myös katsaus Arduinon dokumentointien laatuun. Kirjaston tekninen määrittely ja ohjeistus löytyvät eri verkkotunnusten alta ja valitettavasti ovat keskenään linkitettyjä vain toiseen suuntaan [2, 3]. Sisällöltään ohjeistukset ovat onneksi selkeitä ja helppolukuisia.

Lämpötila-anturin valmistajan Texas Instruments:n komponentin dokumentointi eli datalehti ja I²C-väylän käyttöohje olivat puolestaan erittäin laadukkaita ja sisälsivät kaiken tarvittavan tiedon anturin käyttöönottoon ja sen kanssa kommunikointiin.

Sensorin käyttöönotossa esiintynyt komponentin pienestä koosta johtuva juotosongelma olisi voitu välttää käyttämällä valmista sensorimoduulia. Toisaalta pienen komponentin juottaminen tarjosi hyvää oppia.

6. YHTEENVETO

Kandidaatintyön aiheena oli kirjoittaa Arduino-ohjelmistokirjasto Texas Instruments TMP117-lämpötilasensorille. Ennen lähdekoodin kirjoittamista tuli tutustua Arduino IDE -ohjelmointiympäristön kirjastolta vaatimiin teknisiin seikkoihin ja kirjaston sekä ohjelmatiedostojen kirjoitusohjeeseen. Lisäksi työssä luotiin ohjelmistokirjaston käyttäjälle ohjeistus komponentin käyttöönottoon. Työn alussa esitellään myös Arduino-ekosysteemin osat ja niiden toiminta sekä käyttökohteet.

Kokonaisuutena työ oli onnistunut ja valmis standardien mukainen ohjelmistokirjasto julkaistiin avoimena lähdekoodina.

7. LÄHTEET

[1] About Us (luettu 12.10.2020) Arduino. URL: https://www.arduino.cc/en/Main/AboutUs

[2] Arduino Style Guide for Writing Libraries (luettu 12.10.2020) Arduino. URL: https://www.arduino.cc/en/Reference/APIStyleGuide

[3] Library specification (Luettu 14.12.2020) Arduino. URL: https://arduino.github.io/arduino-cli/library-specification/

[4] Data sheet (luettu 7.9.2020) Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/tmp117.pdf

[5] Arduino Micro (luettu 7.9.2020) Arduino. URL: https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardMicro

[6] Arduino Micro Schematic (luettu 7.9.2020) Arduino. URL: https://www.arduino.cc/en/uploads/Main/arduino-micro-schematic.pdf

[7] Writing a Library for Arduino (luettu 8.11.2020) Arduino. URL: https://www.arduino.cc/en/Hacking/LibraryTutorial

[8] Understanding the I²C Bus (luettu 7.9.2020) Texas Instruments. URL: https://www.ti.com/lit/an/slva704/slva704.pdf

8. LIITTEET

Liite 1 Otsikkotiedosto tmp117.h

Liite 2 Lähdekooditiedosto tmp117.cpp

Liite 3 Esimerkkikoodi tmp117.ino

Liite 4 Avainsanalista keywords.txt

Liite 5 Metadatatiedosto library.properties

Liite 6 Ohjeteksti readme.txt

Kaikki liitteet ovat saatavilla työn kirjoittajan GitHub-profiilissa URL-osoitteessa https://github.com/mitjakarki/tmp117.

Liite 1. Otsikkotiedosto tmp117.h

```
/*
tmp117.h - Library for TI tmp117 temperature sensor
Copyright (c) 2020 Mitja Kärki
University of Oulu
*/

// Ensure this library description is only included once
#ifndef tmp117_h
#define tmp117_h

#include "Arduino.h" // For using the delay()-function in the library
#include "Wire.h" // For i2c communication

// Declare the class
class tmp117
{
// User-accessible public interface
public:
// Functions
tmp117(); // Class constructor. Left empty in arduino libraries.
double readTemperature(void);
bool setAlertLimits(double low, double high);
uint8_t readAlert(void); // Checks if temperature is within the set alert limit
bool begin(uint8_t deviceAddress); // Library init function
// Variables
// readAlert return values
const uint8_t noAlert = 0b00; // temp measurement between alert limits
const uint8_t highAlert = 0b10; // temp measurement above high limit
const uint8_t lowAlert = 0b01; // temp measurement below low limit

// Library-accessible private interface
private:
// Functions
bool i2cWriteData(uint8_t registerAddress, uint16_t data);
uint16_t i2cReadData(uint8_t registerAddress);
// Variables
const uint8_t byteAmount = 2; // All data is 2 bytes of size
```

```
  uint8_t tmp117Address = 72; // Set 72 (ADD0 pin in GND)
as default i2c address
  float resolution = 0.0078125; // LSB resolution of
tmp117
  const uint16_t defaultConfig = 0b0000001000100000; //
Config register default (reset) values
  // tmp117 internal register addresses
  const uint8_t temperatureRegister = 0x00;
  const uint8_t configRegister = 0x01;
  const uint8_t highLimitRegister = 0x02;
  const uint8_t lowLimitRegister = 0x03;
  const uint16_t conversionDelay = 300; // default
conversion and averaging time in ms
  };
  #endif // tmp117
```

Liite 2. Lähdekooditiedosto tmp117.cpp

```
  /*
  tmp117.cpp - library for Texas Instruments tmp117
digital temperature sensor
  Copyright (c) 2020 Mitja Kärki
  University of Oulu
  */

  /*
  This library is for controlling Texas Instruments
tmp117 digital temperature sensor.
  The library was written as a part of my bachelor's
thesis.

  Constructed using the "Arduino style guide for writing
libraries".
  https://www.arduino.cc/en/Reference/APIStyleGuide
  */

  #include "tmp117.h"

  // Empty constructor
  tmp117::tmp117() {}

  // Begin function is used to initialise a library
instance in arduino sketches.
  bool tmp117::begin(uint8_t deviceAddress)
  {
  /* tmp117 address on the i2c bus can be set to 4
different values by
  connecting the tmp117 ADD0 pin to other pins.
  See tmp117 datasheet table 2 for details.*/
  if ((deviceAddress < 72) || (deviceAddress > 75))
  {
  return false;
  }
  /* See datasheet pin functions table for pin ADD0
connection defined addresses.
  Set to 0x48 (=72) for ground connection (default). */
  tmp117Address = deviceAddress;

  // Initialise library
  Wire.begin(); // Initialise wire library for i2c
communication
  delay(2); // 2 ms delay to allow the sensor to boot up
  // defaultConfig enables the alert functionality
  return i2cWriteData(configRegister, defaultConfig);
  }
```

```
  // Read temperature register
  double tmp117::readTemperature(void)
  {
  double temp; // Double precision float for final return
value
  temp = i2cReadData(temperatureRegister) * resolution;
  return temp;
  }

  // Set alert low and high limits
  bool  tmp117::setAlertLimits(double  lowLimit,  double
highLimit)
  {
  // Divide limit values by resolution and convert to
integers
  uint16_t lowLimitInt = lowLimit/resolution;
  uint16_t highLimitInt = highLimit/resolution;

  // Write alert temperature limits to register and check
if it was succesful
  return (i2cWriteData(lowLimitRegister, lowLimitInt) &&
  (i2cWriteData(highLimitRegister, highLimitInt)));
  }

  // Read configuration register contents to see if alert
flags are set
  uint8_t tmp117::readAlert(void)
  {
  // Store contents of configuration register here
  uint16_t configReg;
  uint8_t alertState;
  // Configuration register bitmasks
  uint16_t alertBit = 0b0000000000010000;
  uint16_t highAlertBit = 0b1000000000000000;
  uint16_t lowAlertBit = 0b0100000000000000;

  configReg = i2cReadData(configRegister);
  // Make sure that alert functionality is enabled
  // Alert mode is set if alert bit is set to 0 in config
reg
  if (alertBit & ~configReg)
  {
  // Check if alert flags are set
  if (configReg & highAlertBit)
  {
  alertState = highAlert;
  }
```

```
  else if (configReg & lowAlertBit)
  {
  alertState = lowAlert;
  }
  else {
  alertState = noAlert;
  }
  }
  else
  {
  alertState = noAlert;
  }
  // Clear status flags by reading the tmp117 config
register
  i2cReadData(configRegister);

  return alertState;
  }

  // Read given register contents
  uint16_t tmp117::i2cReadData(uint8_t registerAddress)
  {
  // The arduino wire library reads one byte at a time
from i2c line
  // so we need two one-byte variables to save the whole
16-bit temp value
  uint8_t msByte; // most significant byte of 16-bit temp
reading
  uint8_t lsByte; // least significant byte
  // Begin transmission with tmp117
  // Master needs to address the slave for slave to
transmit data.
  // Each i2c device has its own address on the bus
  // Master also needs to send START condition
  Wire.beginTransmission(tmp117Address);
  // Slave needs configuration from master.
  // Master needs to access slave's internal register map
by writing the
  Wire.write(registerAddress);
  // Master needs to send STOP condition to terminate the
transfer.
  // In arduino wire lib the endTransmission transmits
queued write()-bytes from master to slave
  Wire.endTransmission();
  // Some delay while the sensor converts temperature to
digital format and handles averaging
  delay(conversionDelay);
```

```
  // Request 2 bytes of data (=contents of a register)
from tmp117
  Wire.requestFrom(tmp117Address, byteAmount);
  if(Wire.available() == byteAmount);
  {
  msByte = Wire.read();
  lsByte = Wire.read();
  // Return data combined to a single 16-bit int
  return ((msByte << 8) | lsByte);
  }
  }

  // Writes a 16-bit packet of data to a register on the
tmp117
  bool   tmp117::i2cWriteData(uint8_t   registerAddress,
uint16_t data)
  {
  // Counter for checking that the right amount of bytes
were written
  uint8_t byteCounter = 0;
  // First split the 16-bit data into two 8-bit variables
  uint8_t highByte = data >> 8;
  uint8_t lowByte = data & 0xff;
  Wire.beginTransmission(tmp117Address);
  Wire.write(registerAddress);
  // Write the actual data and check that right amount of
bytes were written
  byteCounter = byteCounter + Wire.write(highByte);
  byteCounter = byteCounter + Wire.write(lowByte);
  Wire.endTransmission();
  delay(2);
  // Successful write
  if (byteCounter == byteAmount)
  {
  return true;
  }
  // Unsuccessful write
  else {
  return false;
  }
  }
```

Liite 3 Esimerkkikoodi tmp117.ino

```
/*
  Example code for tmp117 library

Showcase tmp117 library by creating a window detector for
monitoring room temperature.

  The circuit:
  - Texas Instruments TMP117 attached to i2c bus (pins
depend on arduino board version)
  - use TMP117 datasheet as reference:
https://www.ti.com/lit/ds/symlink/tmp117.pdf
  - you'll need 3 * 5 kOhm resistors
  - tmp117 VDD to arduino 3.3 volt output
  - tmp117 GND to arduino GND
  - tmp117 ADD0 to GND (to select address on i2c bus)
  - 10 nF capacitor between VDD and GND to smooth out
voltage source ripple
  Created in 2020
  By Mitja Kärki

  https://github.com/mitjakarki/tmp117

  */

  #include <tmp117.h>

  // create an instance of the tmp117 class
  tmp117 thermometer;

  const uint8_t tmp117Address = 72; // tmp117 ADD0 pin
connected to GND
  const double alertLowLim = 23.0; // limits in celsius
  const double alertHiLim = 25.0;

  void setup() {
  // setup runs once
  Serial.begin(9600);
  delay(2000);
  Serial.println("Starting temperature measurement");
  // run library init function and
  // check that the sensor address is correct
  if (!thermometer.begin(tmp117Address))
  {
  Serial.println("Invalid tmp117 address!");
  while (1) {
  delay(2000);
  }
  }
```

```
  // set alert low and high limits
  bool  alert  =  thermometer.setAlertLimits(alertLowLim,
alertHiLim);
  Serial.println("Setting alert limits was successful?");
  Serial.println(alert);
  }

  void loop() {
  // loop runs repeatedly
  // read and print temperature
  double temp = thermometer.readTemperature();
  Serial.println("Measured temperature is:");
  Serial.println(temp);
  // read and print alert status
  uint8_t alert = thermometer.readAlert();
  if (alert == thermometer.noAlert) {
  Serial.println("Temperature within set window");
  }
  else if (alert == thermometer.highAlert) {
  Serial.println("ALERT!    Temperature    higher    than
allowed");
  }
  else if (alert == thermometer.lowAlert) {
  Serial.println("ALERT!    Temperature    lower    than
allowed");
  }
  else {
  Serial.println("Reading alert status went wrong");
  Serial.println(alert);
  }
  delay(3000);
  }
```

Liite 4 Avainsanalista keywords.txt

```
#########################################
# Syntax Coloring Map For tmp117
#########################################

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

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

readTemperature   KEYWORD2
setAlertLimits    KEYWORD2
readAlertKEYWORD2
readAlertPin      KEYWORD2
begin    KEYWORD2

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

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

Liite 5 Metadatatiedosto library.properties

```
name=tmp117
version=1.0.4
author=Mitja Kärki <mitja.karki@student.oulu.fi>
maintainer=Mitja Kärki <mitja.karki@student.oulu.fi>
sentence=A basic library for temperature measurement
paragraph=Written as my bachelor’s thesis at the
University of Oulu
category=Sensors
url=https://github.com/mitjakarki/tmp117
architectures=avr
includes=tmp117.h
```

Liite 6 Ohjetiedosto readme.txt

```
This is an Texas Instruments tmp117 digital temperature
sensor library for Arduino. Tested working on Arduino IDE
1.8.13.

Installation
------------------------------------------------------------
----------------------

To install this library, use the Arduino IDE: Sketch >
Include Library > Add .ZIP Library... and browse and
select the library .ZIP.

When installed, this library should look like:

Arduino/libraries/tmp117                              (this
library's folder)
Arduino/libraries/tmp117/tmp117.cpp                   (the
library implementation file)
Arduino/libraries/tmp117/tmp117.h                     (the
library description file)
Arduino/libraries/tmp117/keywords.txt                 (the
syntax coloring file)
Arduino/libraries/tmp117/library.properties
(library metadata)
Arduino/libraries/tmp117/examples                     (the
examples in the "open" menu)
Arduino/libraries/tmp117/examples/tmp117.ino
(example code)
Arduino/libraries/tmp117/extras/circuit_diagram.png
(example circuit diagram)
Arduino/libraries/tmp117/readme.txt                   (this
file)
```