Opinnäytetyö AMK

Elektroniikan koulutusohjelma

Elektroniikkasuunnittelu

2016

Niko Lahtinen

# KÄYTTÖLIITTYMÄN KEHITYS SULAUTETULLE JÄRJESTELMÄLLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ AMK

TURUN AMMATTIKORKEAKOULU

Elektroniikan koulutusohjelma | Elektroniikkasuunnittelu

2016 | 35

Ohjaajat: Lehtori Henry Gylen, Toni Rumpunen

Niko Lahtinen

# KÄYTTÖLIITTYMÄN KEHITYS ELEKTRONIIKAN SULAUTETULLE JÄRJESTELMÄLLE

Tässä työssä kehitetään fyysinen käyttöliittymä valokuituvahvistimelle. Käyttö liittymää ohjataan mikrokontrollerilla. Käyttöliittymällä voidaan selata sekä vaihtaa valokuituvahvistimen asetuksia. Työ keskittyy mikrokontrollerien ohjelmointiin, sekä FreeRTOS-käyttöjärjestelmän käyttöön. FreeRTOS-käyttöjärjestelmää käytetään usein elektroniikan sulautetuissa järjestelmissä, jossa on useita samanaikaisia ohjelmia käytössä. Käyttöjärjestelmä helpottaa suurempien ohjelmisto kokonaisuuksien käyttöä.

Työssä käydään läpi kehitys alustojen ominaisuuksia. Käytetyt kehitys alustat ovat Arduino Yún sekä STM32F407-pohjainen kehitysalusta. Työ on tehty C- ja C++ -kielillä sekä Arduinon omalla ohjelmointikielellä.

Työssä käydään läpi siinä käytettyjä komponentteja, open source -ohjelmia sekä ohjelmointiin käytettyjä työkaluja. Kehitystyö tehtiin Arduino Idellä sekä Red Suite 5:llä.

ASIASANAT:

FreeRTOS, LwIP, STM32F407, C-ohjelmointi, Sulautettu järjestelmä

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Electronics | Electronic Design

2016 | 35

Instructors: Lecturer Henry Gylen, Toni Rumpunen

Niko Lahtinen

# THE DEVELOPMENT OF AN OPERATING INTERFACE FOR AN EMBEDDED SYSTEM

The purpose of this thesis was to develop a physical user interface for an optical amplifier. The interface will be controlled with a microcontroller. This software will be used to adjust and verify the setting of the optical amplifier. This thesis focuses on development of the software for microcontroller and using the FreeRTOS operating system. The FreeRTOS operating system is often used in embedded systems, which use simultaneously multiple programs. The software is easier to develop and debug when using the equivalent operating system.

This thesis looks into the used components, open source programs and the tools used to program the microcontrollers and discusses features of certain development boards. The development boards used in this development were Arduin Yún and a STM32F407 based development board. The software was created using C, C++ and Arduino's own programming language. The result of this thesis is a software that communicates with the optical amplifier using Ethernet connection.

KEYWORDS:

FreeRTOS, LwIP, STM32F407, C programming, embedded systems

# SISÄLTÖ

**KÄYTETYT LYHENTEET** **6**

**1 JOHDANTO** **8**

**2 KEHITYSALUSTAT, OHEISLAITTEET JA NIIDEN OHJELMOINTI** **9**

2.1 LCD-näyttö 9

2.2 Käytetyt kehitysalustat 11

2.3 Kehitysalustojen ohjelmointi 13

**3 OPEN SOURCE –OHJELMAT** **15**

3.1 Open Source lisenssit 15

3.2 FreeRTOS 15

3.3 LwIP 16

**4 KEHITYSYMPÄRISTÖN LUONTI** **17**

4.1 DHCP-palvelin 17

4.2 Uuden projektin luonti Red Suite 5:een 18

4.3 Koodi tiedoston luonti Red Suite 5:ssä 19

4.4 Koodin kääntäminen ja makefile 19

4.5 Virheenpaikantimen käyttö 19

**5 LAITTEEN KEHITYS** **20**

5.1 C-kielen perusfunktiot 20

5.2 Arduin Yún prototyyppi 21

5.3 STM32F407 kehitysalusta 24

5.4 Main funktio 24

5.5 Näppäinten ja näytön ohjelma 25

5.6 Yhteyden muodostaminen 27

5.7 UDP-kommunikointi 28

5.8 Ohjelma kokonaisuus 30

**6 TESTAUS, JATKOKEHITYS SEKÄ ONGELMAT** **32**

6.1 Kehitystyön ongelmat 32

6.2 Jatkokehitys 32

6.3 Lopuksi 33

# KUVAT

Kuva 1. Kuvassa esimerkki I2C kommunikoinnista [4] 10
Kuva 2. Kuvassa vasemmalla Hitachi HD44780 piiri LCD-näytöllä sekä näppäimillä ja oikella I2C LCD-näyttö 10
Kuva 3. Kuvassa näkyy 7-segmenttinäytöllä toteutettavat kirjaimet ja numerot. [5] 11
Kuva 4. Kuvass Arduino Yún kehitysalusta [6] 11
Kuva 5. Kuvassa STM32F407 pohjainen kehitysalusta 12
Kuva 6. Kuvassa Raspberry pi 2 model b 13
Kuva 7. Kuvassa Red probe + [10] 14
Kuva 8. Kuvassa DHCP-palvelimen toiminta.[14] 17
Kuva 9. Kuvassa uuden tyhjän makefile C- ja C++ -projektin luonti. 18

# KUVIOT

Kuvio 1. Kuviossa on LCD-näytön kytkentä ATmega328p mikroprosessorin kanssa. [17] 21
Kuvio 2. Kuviossa näppäimen kytkentä. 22

# KÄYTETYT LYHENTEET

| | |
|---|---|
| ARM | Advanced RISC Machines, kehittyneet RISC-pohjaiset mikroprosessorit |
| C | C-ohjelmointikieli |
| C++ | C++ -ohjelmointikieli |
| Copyleft | Käyttäjänoikeus |
| CPU | Central Processing Unit, keskusprosessori |
| DHCP | Dynamic Host Configuration Protocol, osoitteen välityspalvelin |
| Flash | Flash –muisti, puolijohdemuisti |
| FreeRTOS | Free Real Time Operating System, C-kielen käyttöjärjestelmä |
| GCC | GNU Compiler Collection, GNU kääntäjäkokoelma |
| GNU | GNU's Not Unix, käyttöjärjestelmä |
| GPIO | General Purpose input/output, yleinen input/outputnasta |
| IDE | Integrated development environment, integroitu ohjelman-kehitysalusta |
| IP | Internet Protocol, internetprotokolla |
| I2C | Inter-Integrated Circuit, viestintä stantardi digitaalipiireille |
| LCD | Liquid Crystal Display, Nestekidenäyttö |
| lwIP | Lightweight TCP/IP, pienikokoinen internetprotokollaohjelma |
| MCU | Memory controller unit, muistin hallintayksikkö |
| MHz | Megahertsi |
| PCB | Protocol Controll Block, protokollan hallintaosa |
| Python | Ohjelmointi kieli |

| | |
|---|---|
| SD | Secure Digital, salattu digitaalinen tallennus |
| TCP | Transmission Control Protocol, viestintäprotokolla tietokoneille |
| UDP | User Datagram Protocol, yhteydetön viestintäprotokolla |

# 1 JOHDANTO

Elektroniikan sulautetuissa järjestelmissä on usein rajoituksena muistin määrä. Tässä työssä tarkastellaan pienikokoisen käyttöjärjestelmän ja sovellusten käyttöä sulautetussa järjestelmässä. Työn tavoitteena on luoda valokuituvahvistimelle fyysinen käyttöjärjestelmä LCD-näytön ja näppäinten avulla. Opinnäytetyö on tehty Teleste Oyj:lle.

Sulautettujen järjestelmien käyttöjärjestelmiä käsitteleviä lopputöitä on hyvin vähän. Pekka Kulpakon tekemässä lopputyössä käytettiin VDK-nimistä käyttöjärjestelmää. Työ keskittyi enemmän IC-piirien testaukseen ja tutkimiseen. Samoin LwIP:tä oli käytetty vain kahdessa työssä, joista toinen oli edellä mainittu. [1]

Työn vaatimuksia ovat näytön ja näppäimien lisäksi laitteen kanssa kommunikointi käyttäen ethernet-liitäntää. Laitteen pitää pystyä lukemaan sekä kirjoittamaan valokuituvahvistimelle tietoa. Näppäimistö, näyttö ja ethernet-liikennöinti pitää sisällyttää yhdelle kehitysalustalle, jonka toimeksiantaja on tarjonnut. Nämä ohjelmat eivät saa häiritä muita alustalla toimivia ohjelmia. Tämän takia käytössä on FreeRTOS-käyttöjärjestelmä.

Työssä ei tutkita mahdollisia tuotantoon liittyviä asioita. Työ keskittyy pääosassa sulautetun järjestelmän kehitykseen. Työn tehtävänä on selvittää tuotteen mahdollista käytettävyyttä sekä käyttöjärjestelmän käyttöä sulautetussa laitteessa.

# 2 KEHITYSALUSTAT, OHEISLAITTEET JA NIIDEN OHJELMOINTI

Kehitysalustan tarkoitus on olla riittävän monipuolinen, jotta voidaan tehdä helposti uusia tuotteita. Erilaisia alustoja löytyy hyvin tarkoista vaatimuksista hyvin yleisiin alustoihin. Hyvin varustellut alustat ovat hyviä kehitystyöhön, mutta lopullisessa tuotteessa niissä on usein ylimääräisiä ominaisuuksia.

## 2.1 LCD-näyttö

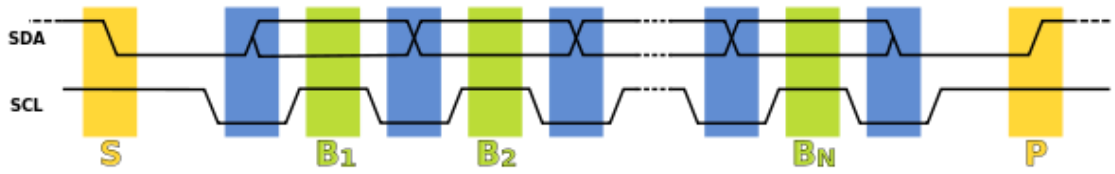
Nestekidenäyttö muodostuu valoa polarisoivasta nesteestä, jota voidaan ohjata sähkön avulla. Nesteen molemmilla puolilla on läpinäkyvät polarisoivat levyt. Polarisoidut levyt ovat suorakulmassa toisiinsa nähden. Näytön taustalla on taustavalo. Kun pikseliin johdetaan sähköä, sen neste polarisoituu ja muodostaa näytölle tumman pisteen, josta taustavalo ei pääse läpi. [2]

### Hitachi HD44780

Toimeksiantaja tarjosi kahdella eri ajurilla varusteltuja LCD-näyttöjä. Toinen oli Hitachi HD44780. Hitatchi HD44780 on pistematriisinäyttö. Piirissä on 16 nastaa, joilla ohjataan näyttöä. Register select ja read/write-nastan avulla määrätään, mikä kuvio tulostetaan mihinkin kohtaan näyttöä. Hitachille löytyy Arduino IDE:stä valmis kirjasto. Tämän tyyppinen LCD-näytön ajuripiiri on hyvin yleinen, joten sitä ohjaava valmis koodi on helppo löytää. [2]

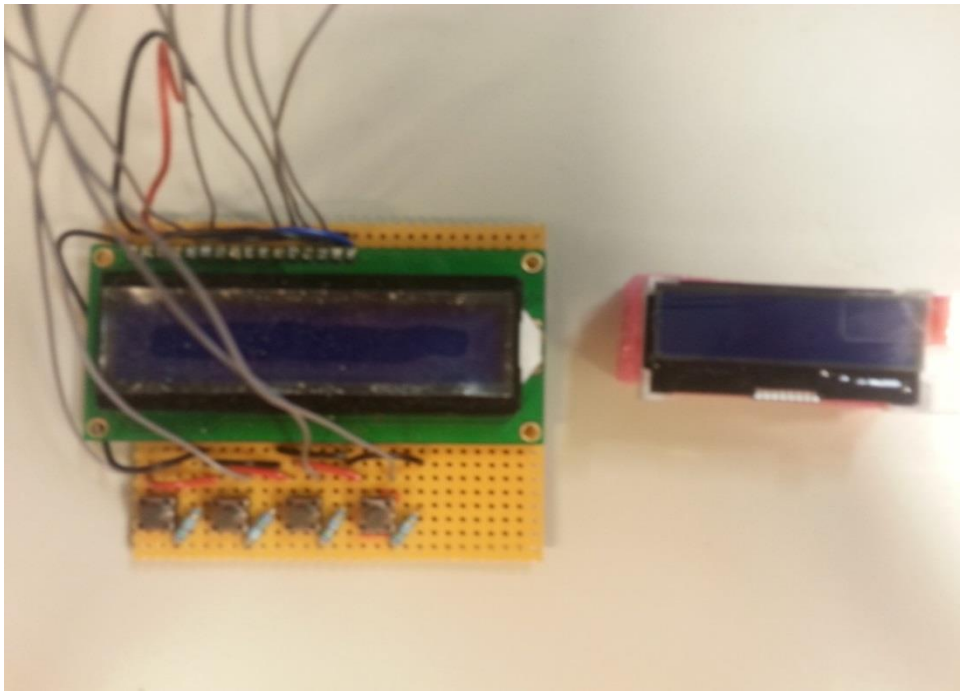
### I2C-näyttö

Toinen näyttö toimi I2C:n standardilla. I2C:n etuna on sen vaatima johtojen määrä. Datan kuljetukseen tarvitaan vain kaksi johtoa, serial data line sekä serial clock line. Samalla voidaan ketjuttaa useampi I2C-laite peräkkäin. Verrattuna I2C:hen Hitachin malli vaatii kuusi johtoa datan siirtämiseen. I2C:n perusnopeus on 100 kbit/s. I2C on hyvin yleinen, joten siihenkin löytyy valmiit ohjelmat helposti. [3]




Kuva 1. Kuvassa esimerkki I2C kommunikoinnista [4]

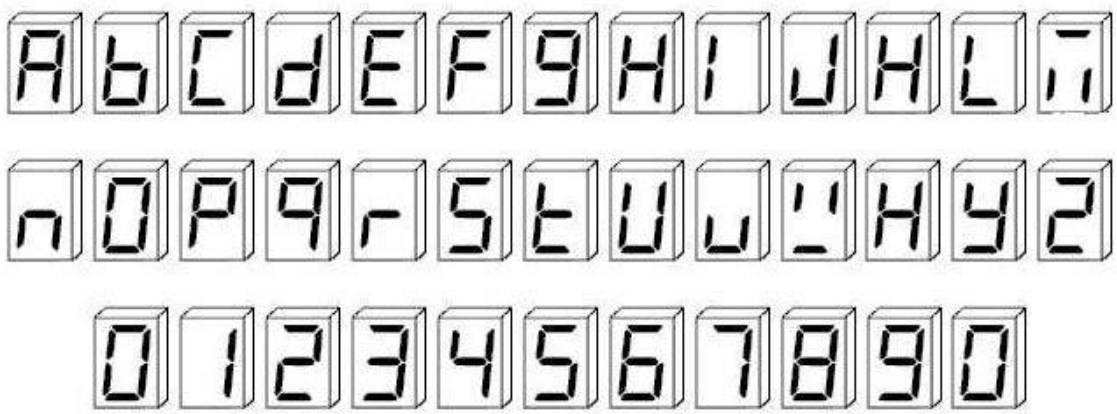
Yllä olevassa kuvassa alempi linja kuvaa serial clock line:ä, joka määrää milloin bitit luetaan. Ylempi linja on serial data line, jossa määrätään luetun bitin arvo. I2C lukee bitit aina kello pulssin ollessa ylhäällä.



Kuva 2. Kuvassa vasemmalla Hitachi HD44780 piiri LCD-näytöllä sekä näppäimillä ja oikella I2C LCD-näyttö

**7-segmenttinäyttö**

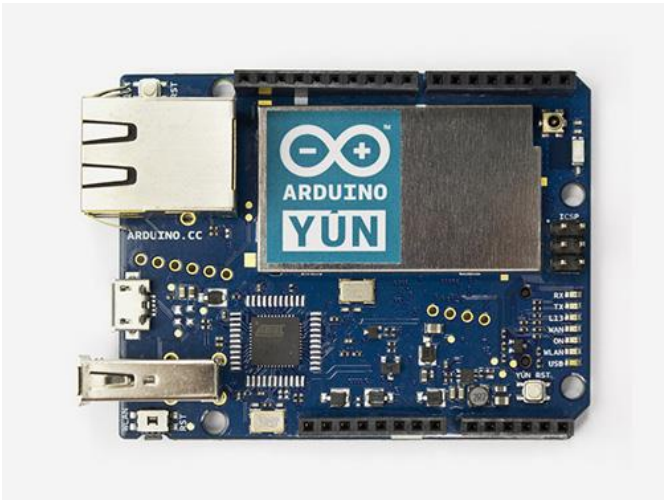
7-segmenttinäyttö on yksinkertainen komponentti, jossa valaistaa halutut segmentit ledeillä. Koska segmenttejä on vain 7, kirjainten näyttäminen voi olla epäselvää.



Kuva 3. Kuvassa 7-segmenttinäytöllä toteutettavat kirjaimet ja numerot. [5]

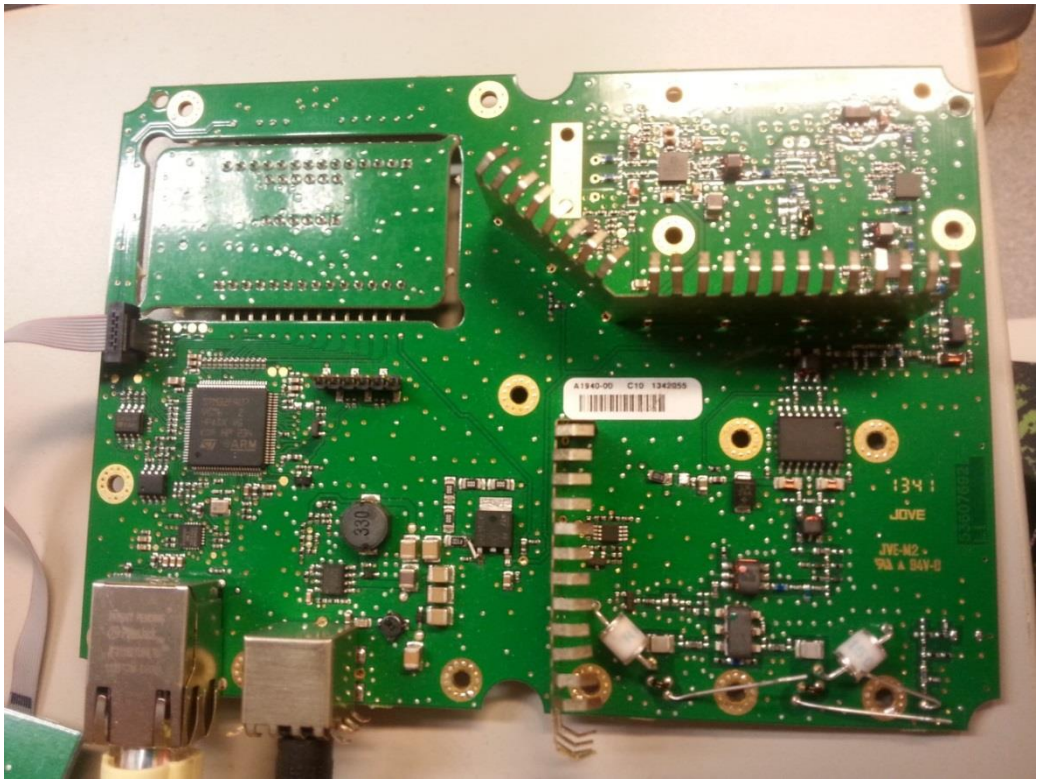
## 2.2 Käytetyt kehitysalustat

Työssä käytettiin kahta eri kehitysalustaa. Ensimmäisessä prototyypissä oli käytössä kuvassa 4 näkyvä Arduino Yún. Arduinot ovat avoimeen lähdekoodin perustuvia tuotteita. Yúnin ominaisuutena on 2 eri prosessoria. Molemmilla prosessoreilla on oma hallittava puoli laitteesta. ATmega32u4-prosessoria käytetään kehitysalustan käyttöön. Atheros AR9331 -prosessorilla käytetään OpenWRT –käyttöjärjestelmä ä. [6]




Kuva 4. Kuvass Arduino Yún kehitysalusta [6]

Lopullinen tuote tehtiin STM32F407-pohjaiselle alustalle. Alusta on huomattavasti monipuolisempi kuin Arduino Yún. Alustassa on integroitu ethernet-portti sekä 4 kappaletta 7-segmenttinäyttöjä. [7]

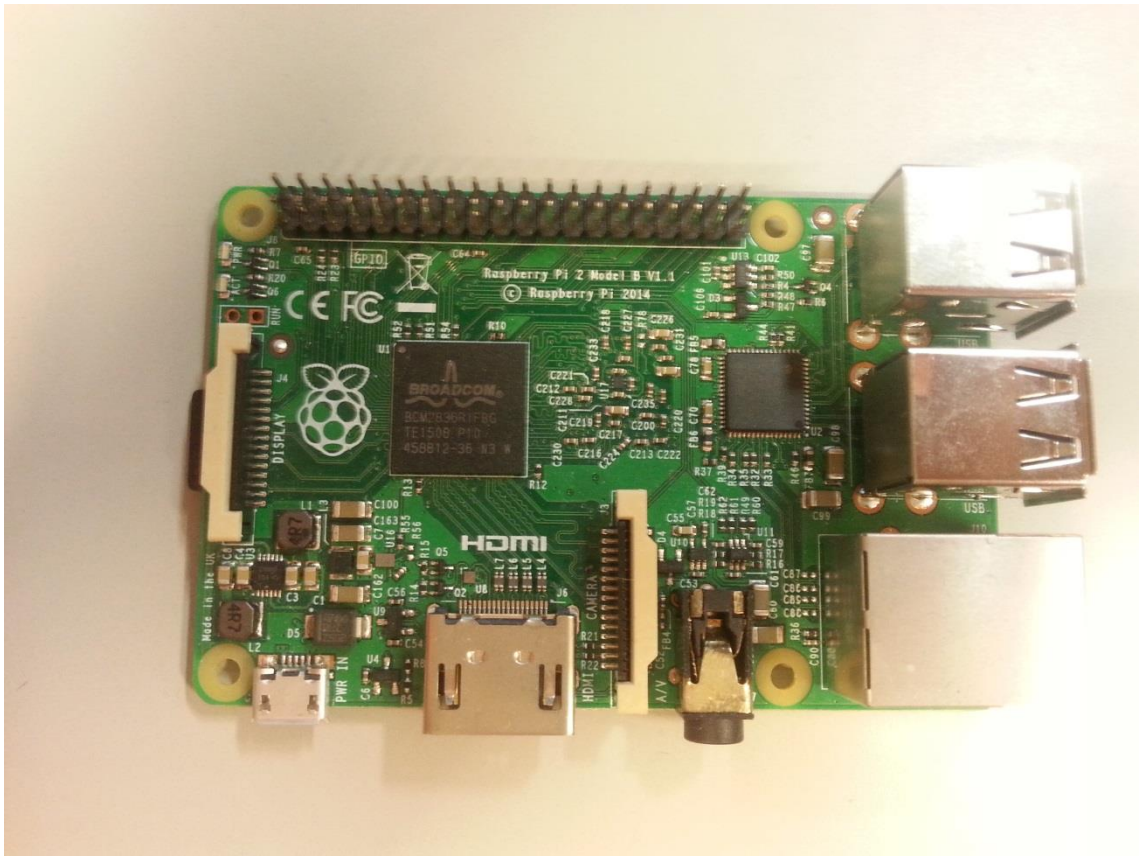


Kuva 5. Kuvassa STM32F407-pohjainen kehitysalusta

Raspberry pi on open source -pohjainen Linux-tietokone. Pienen kokonsa ja nopean configuroinnin ansiosta sitä on hyvä käyttää tukena erilaisissa projekteissa ohjelmointikehityksen apuna. ARM-pohjaisen prosessorin avulla Raspberryllä voidaan käyttää ARM GNU/Linux -jakeluversioita. [8]

Raspberryn oleellisia ominaisuuksia ovat:

- 900 Mhz neliydin ARM Cortex-A7 CPU
- 1 GB RAM –muisti
- HDMI-portti
- Ethernet–portti
- VideoCore IV 3D graphics core
- Micro SD –korttipaikka




Kuva 6. Kuvassa Raspberry pi 2 model b

## 2.3 Kehitysalustojen ohjelmointi

Arduino IDE on avoimen lähdekoodin ohjelma, jolla voidaan ohjelmoida Arduino-kehitysalustoja. Arduino IDE tukee omaa yksinkertaisempaa ohjelmointikieltä. Tämä ohjelmointikieli on hyvin yksinkertainen, mutta hidas ja iso verrattuna C- ja C++ -kieliin. Ohjelma tukee oman ohjelmointikielen lisäksi C- ja C++ -kieliä.

Red Suite 5 on Code Redin tarjoama IDE. Red Suite 5 on ohjelmiston kehitysaalusta ARM -pohjaisille MCU:ille. Red Suite 5:n ominaisuuksia on virheiden paikannus ja suora lataus on-chip flash-muistille. [9]

Red probe + on nopea USB-Fyhteydellä toimiva virheiden paikannin. Tuote on tarkoitettu toimimaan yhdessä Red Suite 5:n kanssa. Red probe +:aa käytetään ohjelman kirjoittamiseen kehitysalustalle, sekä sillä voidaan luoda pysäytyspisteitä koodiin. Pysäytyspisteillä voidaan tarkastella laitteen arvoja sekä koodin toimintaa tietyssä kohtaa. Laite pystyy suorittamaan yksittäisiä rivejä koodissa, jolloin näkee

suoraan, mitä laitteessa tapahtuu. Laitteen päällä olevat kolme lediä kertovat yhteyksien tilan. Jos ledi palaa, niin yhteys on olemassa. Laite ei erittele toimiiko yhteys. [10]


Kuva 7. Kuvassa Red probe + [10]

# 3 OPEN SOURCE –OHJELMAT

Open Source –ohjelmat ovat ilmaisia. Kuka tahansa pääsee vapaasti käyttämään sekä muuttamaan näitä ohjelmia. Ohjelma luokitellaan avoimen lähdekoodin ohjelmaksi, sen lisenssin perusteella. [11]

## 3.1 Open Source lisenssit

Kaupallisessa käytössä open source –ohjelmissa pitää ottaa huomioon kunkin ohjelman lisenssi. Open source –lisenssit voidaan jakaa karkeasti kolmeen luokkaan.

- Permissive license
- Weak copyleft license
- Strong copyleft license

Permissive -lisenssissä koodia käyttävä yritys saa vapaasti käyttää ja myydä tämän lisenssin alla olevia tuotteita vapaasti eteenpäin ilman, että heidän tarvitsee julkaista omaa koodiaan. Weak copyleft –lisenssin tuotteiden koodia saa muokata ja myydä, mutta kehitetty koodi pitää julkaista tuotteen ostajalle tai linkata dynaamisesti erilaiseen heikkoa käyttöoikeutta käyttävään kirjastoon. Strong copyleft –lisenssi vaatii samoja käyttöoikeuksia tuotteen tekijälle, sekä ostajalle. Tuote pitää toimittaa samalla lisenssillä, kuin käytetty open source –ohjelma. [11]

## 3.2 FreeRTOS

FreeRTOS on C-kielinen käyttöjärjestelmä. Käyttämällä käyttöjärjestelmää pystytään ajamaan monta eri ohjelmaa hallitusti samaan aikaan. Tässä työssä samalla prosessorilla toimi useita sovelluksia samanaikaisesti. Käyttöjärjestelmä helpottaa samanaikaisten ohjelmien toiminnan seuraamista. Ilman käyttöjärjestelmää koodista tulee sekava ja sen virheiden paikannus muuttuu haastavaksi, kun käytetään samanaikaisesti useita ohjelmia. Tavallisesti FreeRTOS:in binääri kuva on noin 6 kt:sta 12 kt:uun. [12]

```
void vFlashCoRoutine( CoRoutineHandle_t xHandle,
                      UBaseType_t uxIndex )
{
```

```
    // Co-routines must start with a call to crSTART().
    crSTART( xHandle );

    for( ;; )
    {
        // Delay for a fixed period.
        crDELAY( xHandle, 10 );

        // Flash an LED.
        vParTestToggleLED( 0 );
    }

    // Co-routines must end with a call to crEND().
    crEND();
}
```

Yllä esimerkki FreeRTOS-käyttöjärjestelmän koodista. [12]

Esimerkissä olevassa koodissa luodaan funktio *vFlashCoRoutine*, joka vilkuttaa laitteessa olevaa lediä. FreeRTOS on kirjoitettu tiettyjen ohjeiden mukaisesti. Funktioiden ja muutujien ensimmäinen kirjain kertoo niiden ominaisuuden. Esimerkiksi *vParTestToggleLED( 0 );,* v sana edessä tarkoittaa lopun olevan funkitio. Muita yleisiä merkkejä ovat *x*, joka tarkoittaa *size_t* tapaista muuttujaa ja *q*, joka viittaa tehtäväjonoon. Tämä koodin kirjoitus järjestelmä helpottaa koodin luettavuutta.

## 3.3 LwIP

LwIP on pienikokoinen TCP/IP protokolla paketti. Koska LwIP vie alle 10 kt tilaa, se sopii hyvin sulautettuihin järjestelmiin. Kuten FreeRTOS, myös LwIP on avoimen lähdekoodin ohjelma. Tätä sovellusta voidaan käyttää käyttöjärjestelmän kanssa tai ilman. LwIP tukee useita ethernet-liikenteeseen kuuluvia protokollia, kuten IP, UDP, TCP ja DHCP. [13]

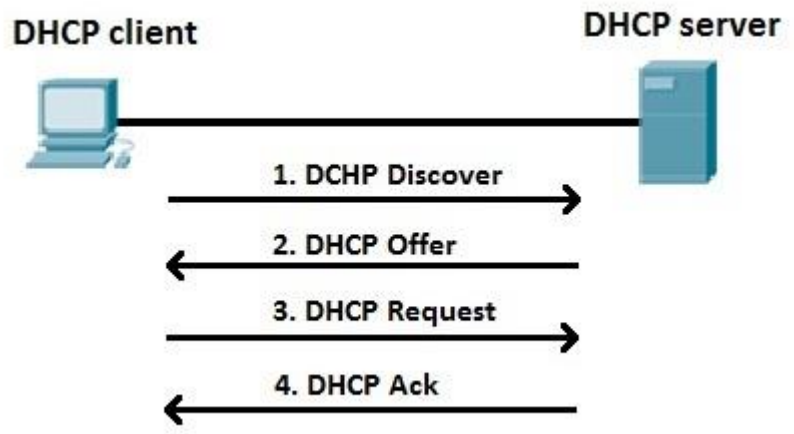
# 4 KEHITYSYMPÄRISTÖN LUONTI

## 4.1 DHCP-palvelin

DHCP-palvelin on lähiverkossa toimiva osoitteen jakaja. Se jakaa IP-osoitteita lähiverkon laitteille. Palvelimelle kerrotaan osoite avaruus, josta se jakaa osoitteita eteenpäin.

DHCP-palvelin asennettiin aluksi Arduino Yúnin OpenWrt käyttöjärjestelmälle. Kuitenkin jostiain syystä laite ei toiminut halutulla tavalla. Ongelmana on mahdollisesti Arduinolle asennettu oma käyttöjärjestelmä, joka sotkee network interface -asetuksia.

Arduinon jälkeen käyttöön otettiin Raspberry pi 2. Molemmissa tapauksissa käyttöjärjestelmä kirjoitetaan levyn näköistiedostona SD-kortille toisen tietokoneen avulla.




Kuva 8. Kuvassa DHCP-palvelimen toiminta.[14]

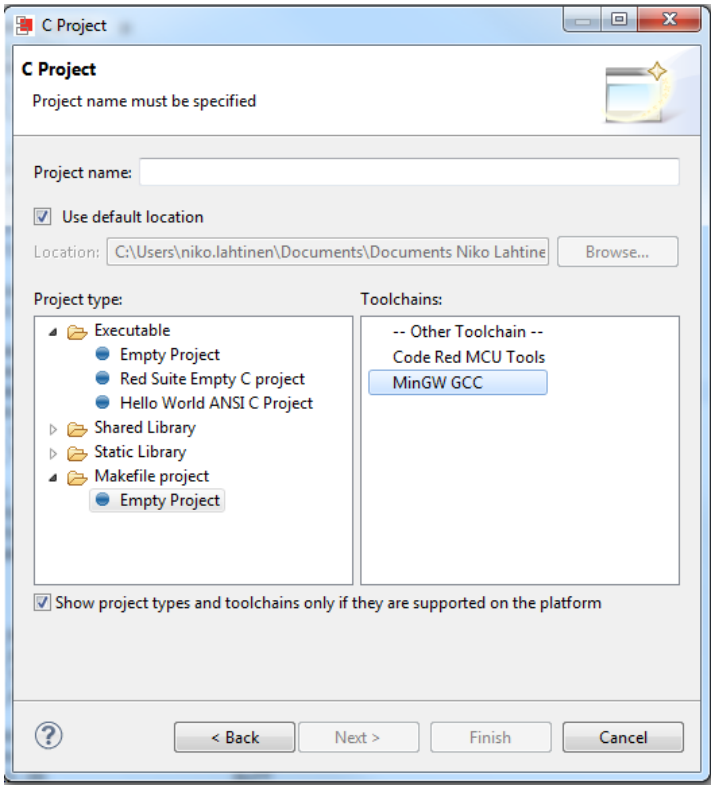
**Internet Systems Consortium DHCP server**

DHCP-palvelimena käytettiin Internet Systems Consortiumin tarjoamaa ilmaista ohjelmaa. Palvelimen käyttöön otossa määritellään Raspberryn asetuksiin ensin staattinen IP–osoite. Tämän jälkeen palvelimen asetuksiin määritetään DHCP-palvelimen IP–osoiteavaruus. Lopuksi vaihdetaan tämä ohjelma käynnistymään

Raspberryn käynnistyessä. Staattisen IP–osoitteen asettamiseen meni huomattavasti luultua pidempään. Uudessa Raspberryn käyttöjärjestelmässä oli muutettu IP–osoitteiden asetusten muutoksia. Asetus piti vaihtaa erilliseen kansioon, sekä graafisesta käyttöliittymästä. Palvelin käynnistyy kirjoittamalla Raspberryn terminaaliin komento *sudo service isc-dhcp-server start*. [15]

## 4.2 Uuden projektin luonti Red Suite 5:een

Uusi C- ja C++ -projekti alkaa valitsemalla valikosta uusi projekti. Tämän jälkeen projektille määritetään projektin tyyppi, esimerkiksi makefile-projekti. Lisäksi valitaan oikea työkalu koodin kääntämiseen, tässä tapauksessa MinGW GCC, sekä koodin käyttämien kirjastojen sijainti. Kun valitaan tyhjä makefile-projekti, Red Suite 5 antaa käyttäjän luoda itse makefile:n, sekä lähde tiedostot. Seuraavassa valikossa voidaan asettaa käännöstyökaluille tarkempia asetuksia, kuten koodin kielen versio. Red Suite 5:llä voidaan myös tehdä uusi projekti valmiiden asetusten pohjalta, jolloin ohjelma kysyy mille mikrokontrollerille ohjelma tehdään. Tämän jälkeen Red Suite 5 tuo projektiin mukaan automaattisesti mikroprosessorin vaatimat kirjastot.




Kuva 9. Kuvassa uuden tyhjän makefile C- ja C++ -projektin luonti.

### 4.3 Koodi tiedoston luonti Red Suite 5:ssä

Ohjelman kirjoitus aloitetaan luomalla uusi lähdetiedosto. Tiedosto pitää nimetä, sen käyttötarkoituksen mukaan. C-kielellä tehdyt tiedostot ovat .c-päätteisiä. Uutta tiedostoa luotaessa asetetaan hiiri ensin kansioon johon se tehdään ja sen jälkeen valitaan hiiren oikealla näppäimellä uusi tiedosto. Red Suite 5 tunnistaa automaattisesti käytetyn ohjelmointikielen tiedoston päättestä. C-kielen pääte on .c, C++ -kielen pääte on .cpp ja kirjastotiedoston pääte on .h.

### 4.4 Koodin kääntäminen ja makefile

Valmis C-koodi pitää kääntää ensin mikroprosessorille ymmärrettävään muotoon. Tämän jälkeen ohjelma voidaan kirjoittaa mikroprosessorille. Tässä työssä tehty koodi käännettiin Red Suite 5 –ohjelmalla. Ohjelmassa on valmiiksi C- ja C++ -kääntäjät. Koodin kääntämisessä käytetään apuna makefile:ä. Makefile on tiedosto johon kirjoitetaan käännettävän koodin asetukset, kuten käännöksen nimi ja kirjastojen sijainti. Työkalu palkista painetaan build eli kokoa. Tähän toimintoon voidaan tallettaa valmiiksi eri kokonaisuuksia projektista. Työn helpotuksen vuoksi on luotu valmiiksi virheen paikannus käännös, sekä lopullinen käännös, josta puuttuu virheen paikannuksen vaatimat ohjelmat.

### 4.5 Virheenpaikantimen käyttö

Virheenpaikanninta käytetään, kun halutaan tarkastella tiettyä osaa ohjelmassa. Ohjelmaan laitetaan pysähdyspiste ja käynnistetään virheenpaikannusominaisuus Red Suite 5 -ohjelmassa. Ohjelma alkaa alusta ja pysähtyy haluttuun pisteeseen. Red Suite 5 pystyy näyttämään kaikki muuttujat tässä pysähdyspisteessä. Muuttujien avulla voidaan tarkistaa, että ohjelma toimii halutulla tavalla. Ohjelmaa voidaan myös ajaa yksi rivi kerrallaan.

# 5 LAITTEEN KEHITYS

## 5.1 C-kielen perusfunktiot

If-lause tarkistaa jonkin ehdon, jonka jälkeen ohjelma suorittaa lauseen sisällä olevan toiminnon. Jos ehto ei täyty, niin ohjelma hyppää toiminnon yli. If-lause on hyvin helppo ja nopea käyttää.

```
If ( a > 10 ) {

Lcd.print(”Adjust”);

}
```

Yllä esimerkki if-lauseesta. Lause tarkistaa onko muuttuja a suurempi kuin 10. Jos a on suurempi kuin kymmenen, tulostetaan näytölle teksti *Adjust*. If-lausetta käytetään näppäinten painalluksien prosessointiin.

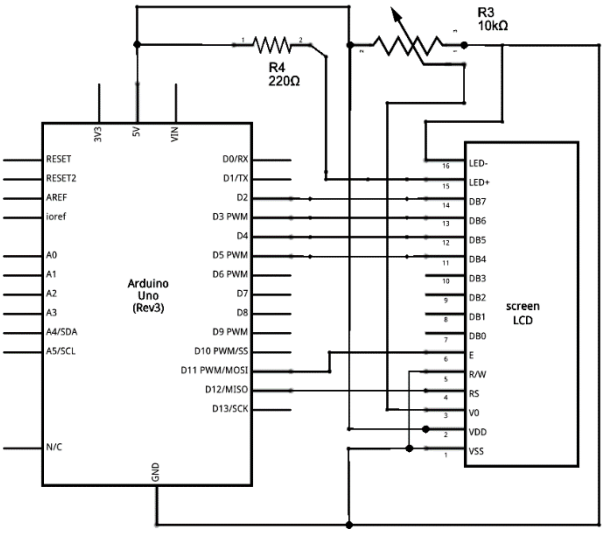
Työssä käytetään switch case -lausetta yksinkertaisen valikon luomiseen. Alussa tarkistetaan esimerkiksi muuttujan arvo, jonka jälkeen siirrytään sen osoittamaan paikkaan, jossa suoritetaan tietty toiminto.

```
switch(expression) {

   case constant-expression  :

      statement(s);

      break; /* optional */

   case constant-expression  :

      statement(s);

      break; /* optional */

   /* you can have any number of case statements */

   default : /* Optional */

   statement(s);

}
```

Yllä esimerkki switch case -lauseesta. [16]

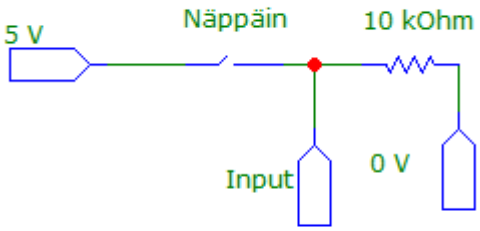
## 5.2 Arduin Yún prototyyppi

Ensimmäinen prototyyppi tehtiin Arduino Yún:illa. Tässä versiossa oli vain näyttö, näppäimet ja valikko. Tässä käytettiin hyväksi Arduinon omaa ohjelmointikieltä, josta löytyy helposti kirjastot LCD-näytölle, sekä näppäimille.




Kuvio 1. Kuviossa on LCD-näytön kytkentä ATmega328p mikroprosessorin kanssa. [17]

Nappulat kytketään luettavaksi, kun jännite on joko 0 V tai 5 V. On parempi kytkeä nappula siten, että taso on normaalisti 0 V ja painalluksessa 5 V. Näin saadaan säästettyä vuotovirtaa ja ongelmia ei synny kelluvien jännitteiden kanssa. Maan ja luettavan portin väliin laitetaan suuri vastus, vähintään 10 kΩ. Näppäintä painaessa luettava portti yhdistyy 5 V:n porttiin. Kun luettavan portin jännite on 5 V, ohjelma rekisteröi näppäimen painetuksi. Vastus varmistaa, että näppäintä lukevan portin jännite on 0 V, kun näppäintä ei paineta.




Kuvio 2. Kuviossa näppäimen kytkentä.

**Prototyypin koodin kehitys**

Arduinon prototyypissä käytettiin Arduinon tarjoamia valmiita kirjastoja, sekä Arduinon omaa ohjelmointikieltä. Arduinon oma kieli on helppoa kirjoittaa, mutta käytössä se on hidas ja iso verrattuna C-kieleen. Arduino koodi perustuu kahdesta osasta, setup ja loop. Setup osassa määritellään miten ja mitä portteja käytetään. Setup-funktio käydään aina läpi kerran laitteen käynnistyessä. Loop-osassa on koodi jota suoritetaan jatkuvasti.

```
void setup() {

pinMode(13, OUTPUT);
```

Osio, jossa määritietään ohjelman vaatimat nastat. Nasta 13 määritetään ulosmenoksi

```
}

void loop() {

digitalWrite(13, HIGH);

digitalWrite(13, LOW);
```

Osio, joka suorittaa toimintoa jatkuvasti. Nastan 13 arvoa muutetaan viidestä voltista nollaan volttiin.

```
}
```

Yllä esimerkki Arduinon ohjelman rakenteesta.

Arduinolla luotiin nopeasti perusvalikko näytölle ja näppäimille. Loop-osassa odotettiin, että jokin nappula painetaan. Sen jälkeen tehtiin ehdon mukainen tulostus näytölle. Valikon valinta toimi switch case periaatteella. Kun näppäintä painetaan, ohjelma tarkistaa, mikä näppäin on painettuna ja tekee näppäimen määräämän komennon.

```
if(backstate == HIGH ){

    if(a>=30){

      a=3;

      }else if(a>=20){

      a=2;

      }else if(a>=10){

      a=1;

      }

}
```

Yllä olevassa esimerkissä on osa Arduinossa käytetystä if-lauseesta. Tätä käytetään valikon selaamiseen. Aluksi laite tunnistaa, mikä näppäin on painettu. Tämän jälkeen ohjelma muuttaa muuttujan a arvoa ehtojen mukaisesti. Backstate on tietyn näppäimen nimi ja *HIGH* on sen tämän hetkinen arvo.

Switch case -lauseella tulostetaan näytölle haluttu teksti.

```
switch (a) {

    case 1:

      lcd.print("Adjustments");

      delay(100);

      break;

       case 10:

              lcd.print("Input: -5dBm");

              delay(100);

              break;
```

```
}
```

Yllä olevassa osassa tarkisetetaan muuttujan *a* arvo ja tämän jälkeen ohjelma siirtyy muuttujan määräämään kohtaan.

## 5.3 STM32F407-kehitysalusta

STM32F407-kehitysalustalla käytettiin hyväksi siinä valmiiksi olevaa koodia. Kehitysalustalla oli valmiiksi koodi näytölle, näppäimille sekä LwIP-ohjelmalle. Kehitysalustassa on valmiiksi kolme näppäintä sekä 4 kappaletta 7-segmenttinäyttöjä. Alustassa olevaa kolmea näppäintä käytetään asetusten selaamiseen ja vaihtamiseen. Näytön sekä näppäintä ohjaamiseen käytetään STMicroelectronicsin tarjoamaa kirjastoa. Kirjastoissa on valmiiksi määritelty funktio, joka asettaa tietyt prosessorin nastat näytön käyttöön. Kirjaistoissa on myös valmiiksi määriteltynä, mikä kirjain vastaa mitäkin yhdistelmää näytössä ja prosessorissa.

Koodista suurin osa on FreeRTOS:in vaatimia osia. Jotta ohjelma toimisi kuten haluttu, pitää sille tehdä käyttöjärjestelmän vaatimat osat. Käyttöjärjestelmä toimii aluksi kuten tavaillinen C-kielen ohjelma. Tiedostossa main.c käynnistetään kaikki vaadittavat ominaisuudet kutsumalla funktioita ja luomalla käyttöjärjestelmälle tehtäviä sekä jonoja. Jotta IP-yhteys toimisi, pitää maintiedostossa olla kutsu, joka käynnistää vaadittavat osat IP-yhteydelle. Lisäksi pitää luoda jono, johon sovellukset voivat lähettää pyyntöjä IP-sovellukselle.

LwIP:n käyttöönotossa pitää määritellä sen käyttämiin kirjastoihin halutut asetukset. Kirjastoissa määritellään mitä protokollia ja miten niitä käytetään. Tässä työssä LwIP asetettiin toimimaan DHCP-palvelimen asiakkaana.

## 5.4 Main funktio

Ohjelma alkaa funktiosta main. Mikroprosessorin alustus suoritetaan toisessa main.c -tiedostossa. Ohjelman käynnistyessä ensin käynnistetään itse mikroprosessoria ja lisälaitteita ohjaavat ohjelmat. Tämän jälkeen siirrytään FreeRTOS:in käyttämään main.cpp-tiedostoon. Main.cpp-tiedostossa kutsutaan käyttöjärjestelmän main funktiota.

```
Int main(void)
{

xQueueHandle qWeb2Appl = xQueueCreate( 1, sizeof( QMSG ) );
```

Luodaan jono ethernetistä applikaatioille.

```
xQueueHandle qAppl2Web = xQueueCreate( 1, sizeof( QMSG ) );
```

Luodaan jono applikaatioilta ethernettiin.

```
display_init();
```

Funktiossa display_init(); asetetaan näytön vaatimat nastat sekä luodaan näytölle tehtävä käyttöjärjestelmälle.

```
vWebInit();
```

Funktio kutsuu web-asetuksia sekä luo uuden tehtävän käyttöjärjestelmälle.

```
vTaskStartScheduler();
```

Lopuksi käyttöjärjestelmä käynnistää tehtävähallinnan. Tämä funktio alkaa käyttää edellä määriteltyjen funktioiden jonojen tietoja.

```
return 0;

}
```

## 5.5 Näppäinten ja näytön ohjelma

Main funktiossa on valmiiksi määriteltynä display_init sekä local_ui_init funktiot. Näissä kahdessa luodaan tehtävät käyttöjärjestelmälle sekä määritetään nastojen tehtävä prosessorille. Jotta näyttö toimisi, sille pitää luoda käyttöjärjestelmän vaatima tehtävä. Tehtävää voidaan kutsua esimerkiksi display_init -funktiossa.

```
xTaskCreate (vDisplay);
```

xTaskCreate luo funktiolle vDisplay oman tehtävä jonon.

Toimeksiantajan koodissa on valmiiksi koodi näytölle ja näppäimille. Tämän takia ei tehdä uusia funktioita vaan muokataan olemassa olevaa. Näppäimet ovat valmiiksi määriteltyinä.

```
if (mUpButton.isPressed())
            {
                        do
                        {
                        if (++mViewMenuSelect < VIEW_MENU_LAST)
                            mViewMenuSelect = VIEW_MENU_FIRST;
```

If lause tarkistaa, missä kohtaa valikkoa ollaan.

```
                        DisplayMenuText();
```

Valikon kohta määrää mitä tulostetaan näytölle.

```
                        vTaskDelay((portTickType) 500);
                        }
 while (!mDownButton.isReleased());
```

If-lause tarkistaa mikä näppäin on painettuna, jonka jälkeen se suorittaa do-lauseessa olevan tehtävän. Lopussa odotetaan, että näppäintä ei enää paineta. Do-lauseen tehtävä voisi olla sama kuin Arduinon prototyypissä. Tässä tapauksessa muuttuja a on korvattu *uint8_t* tyypisellä *mViewMenuSelect* muuttujalla.

Valikon rakenteeseen tulee pieniä muutoksia käytön puolesta, sillä nyt on käytössä kolme näppäintä neljän sijasta. Prototyypissä näppäimet olivat select, back, up sekä down. Kolmella näppäimellä valikko toteutetaan ilman down-näppäintä. Valikko muuttujalle asetetaan arvo, jonka jälkeen hypätään takaisin valikon alkuun. Esimerkiksi viisi eri kohtaa, jolloin valikko alkaa arvosta 1 ja up-näppäin lisää arvoon 1. Kun muuttujan arvo on 5 ja painetaan näppäintä up, palataan kohtaan yksi.

### Valikko

Yksinkertaisin valikko on laittaa kaikki asetukset samalle tasolle. Arduinolla tehdyssä prototyypissä oli 3 tasoa valikossa. Valikon esineistä luodaan lista.

```
static char const * view_menu[] =
{
            "CFG ",
            "GAIN",
            " RF ",
            "OPTI",
            "PROP",
            "MODU",
```

```
};
```

Valikossa pitää käyttää lyhenteitä, sillä käytössä on vain 4 merkkiä kerrallaan.

*VIEW_MENU_LAST* -muuttuja muodostetaan *view_menu*-taulukosta.

```
uint8_t const VIEW_MENU_LAST  = (sizeof(view_menu)/sizeof(view_menu[0]));
```

Yllä oleva toiminto laskee *view_menu*-taulukon esineiden määrän. Arduinon prototyypissä merkittiin kaikki tapaukset ja verrattiin niitä nykyiseen kohtaan. *View_menu*-funktiolle ei tarvitse erikseen määrittää jokaista kohtaa, vaan ohjelma laskee sen suoraan listasta.

Pelkkä näytölle tulostus ei riitä asetuksien lukemiseen ja vaihtamiseen. Näytölle arvojen tulostus tapahtuu view_menu:n kohtien arvojen avulla. *View_menu*-taulukon kohdille voidaan laskea arvo. Esimerkisksi kohdan ”CFG ” arvo on yksi. Ohjelmalle määritetään muuttujat näiden arvojen perusteella. Näitä muuttujia kutsutaan myöhemmin, kun halutaan tulostaa tekstiä näytölle.

```
uint8_t const DISPLAY_VIEW_CONFIGURATION     =  1;
uint8_t const DISPLAY_VIEW_GAIN     =  2;

 switch (mViewMenuSelect)
{
  case DISPLAY_VIEW_OPTLEVEL: set_display_text(CONFIGURATION);  break;
  case DISPLAY_VIEW_GAIN:     set_display_text(GAIN);  break;

}
```

Switch-lause tarkistaa *mViewMenuSelect* -muuttujan arvon ja siirtyy yläpuolella määritettyyn kohtaan.

## 5.6 Yhteyden muodostaminen

Laitteeseen muodostetaan yhteys ethernet-liittymän kautta. Ethernet-sovittimen kanssa käytetään LwIP ohjelmaa. Main funktiossa kutsutaan funktiota *vWebInit*. *vWebInit* funktiossa kutsutaan myöhemmin funktiota *LwIP_Init*, joka asettaa varsinaiset IP-asetukset. Valmiisiin IP-asetusten koodiin tehdään muutama muutos, jotta ohjelma toimii DHCP-palvelimen asiakkaana.

```
void LwIP_init(void){

int a, b, c, d, f;

if (ETH_link_xSemaphore == NULL ){

vSemaphoreCreateBinary(ETH_link_xSemaphore);

}

ETH_BSP_Config();

autoip_start;

}
```

Yllä olevassa esimerkissä on LwIP:n käynnistysfunktio. Alussa luodaan muuttujat IP-osoitteelle. Sen jälkeen varmistetaan, että *ETH_link_xSemaphore* on olemassa. Semafori ohjaa ethernet-yhteyden liikennettä. *ETH_BSP_Config* funktio määrittää ethernet-portin nastat prosessorille. Lopuksi autoip_start käynnistää IP-osoitteen haun ja lopullisen toiminnan.

LwIP vaatii myös, että sen opt.h-tiedostoon tehdään määritelmämuutos. Tiedostossa on lueteltuna LwIP:n asetukset.

```
#indef          LWIP_AUTOIP

#define        LWIP_AUTOIP    1

#endif
```

Yllä olevassa esimerkissä on määritelty *LWIP_AUTOIP* sallituksi. Muuttamalla yksi nollaksi toiminto kiellettäisiin.

## 5.7 UDP-kommunikointi

Kommunikointiin käytetään UDP-paketteja. UDP-paketteja voidaan seurata ja lähettää helposti Python-ohjelmalla. UDP-paketeille luodaan websocket. Websocket asetetaan vakioksi tietokoneella. Jotta tietokoneella voidaan vastaanottaa kehitysalustan lähettämiä paketteja, tietokoneen UDP-kuunteluportti pitää tuntea. Tietokone pystyy vastaamaan paketteihin, sillä UDP-paketti sisältää lähetys portin osoitteen.

```
import socket’
```

```
DEST_IP =     ”192.168.0.11”

DEST_PORT =   2500

MESSAGE =     bytes.fromhex(”test”)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)void

sock.sendto(MESSAGE, (DEST_IP, DEST_PORT))

IP, PORT =    sock.getsockname()

sock =        socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind((IP, PORT))

while True:

              data, addr = sock.recvfrom(1024)

              print("Recieved data: ", data[7:], "\n")

              quit()
```

Yllä olevassa esimerkissä on Python-ohjelma. Ohjelma lähettää heksabitteinä tekstin test. Tämän jälkeen se odottaa vastausta. Vaihtamalla ohjelman järjestystä se saadaan ensin kuuntelemaan vastausta, jonka jälkeen se vastaa viestiin. Ohjelmaa käytettiin valokuituvahvistimen simulointina.

### UDP-kommunikointi LwIP:llä

```
struct udp_pcb * udp_new(void)
```

Yllä olevassa esimerkissä luodaan uusi *udp_pcb*. PCB-paketti tunnistaa UDP-yhteyden. Tämän jälkeen viestejä voidaan joko vain lähettää, tai luoda pysyvä yhteys. UDP-pakettien lähettämiseen ja vastaanottamiseen tarvitaan uusi yhteys.

```
err_t udp_connect(struct udp_pcb * pcb, struct ip_addr * ipaddr, u16_t port)
```
[18]

Yllä olevassa esimerkissä luodaan uusi UDP-yhteys ja sen muuttujat ipddr sekä port määritetään autoip_start funktiossa. Ipaddr ja port ovat osoitteet joihin paketit lähetetään.

```
err_t udp_send(struct udp_pcb * pcb, struct pbuf * p)
```

Yllä oleva koodi lähettää udp_connect:ssa määriteltyyn osoitteeseen paketin pbuf.

Lähetyksen jälkeen luodaan kuunteluportti.

```
void udp_recv(struct udp_pcb * pcb,

              void (* recv)(void * arg, struct udp_pcb * upcb,

                                        struct pbuf * p,

                                        struct ip_addr * addr,

                                        u16_t port),

              void * recv_arg)
```
[19]

Yllä olevassa esimerkissä luodaan kuunteluosoite. Osoitteen pitää olla sama, kuin mistä paketti lähetettiin. Vastaanotettu data osoitetaan *recv*-muutujaan.

## 5.8 Ohjelma kokonaisuus

Ohjelma koostuu kahdesta tiedostosta. Localui.cpp sekä udp_serv.c. Localui-tiedostossa on ohjelman käyttö, joka kutsuu tarvittaessa udp_serv.c tiedoston funktioita. Jokaisella uudella UDP-pyynnöllä luodaan uusi UDP-osoite. Käytön jälkeen osoite suljetaan. Main-funktiossa pitää käynnistää käyttöliittymälle sekä LwIP:lle omat tehtävät. Tämän jälkeen ne alkavat toimia itsenäisesti.

```
if (mUpButton.isPressed())
             {
                          do
                          {
                          if (++mViewMenuSelect < VIEW_MENU_LAST)
                              mViewMenuSelect = VIEW_MENU_FIRST;


             DisplayMenuText();
```

Edellisellä sivulla olevassa esimerkissä odotetaan, kunnes nappulaa painetaan. Tämä kutsuu *DisplayMenuText*-funktiota. *DisplayMenuText* hakee tulostettavan tiedon valmiilta listalta tai kysyy sitä UDP-paketin kanssa.

```
void vUdp_send(char * P){

err_t udp_connect(struct udp_pcb * pcb, struct ip_addr * ipaddr, u16_t port)

err_t udp_send(struct udp_pcb * pcb, struct pbuf * p)
```

```
void udp_recv(struct udp_pcb * pcb,
              void (* recv)(void * arg, struct udp_pcb * upcb,
                                        struct pbuf * p,
                                        struct ip_addr * addr,
                                        u16_t port),
              void * recv_arg)
}
```

Yllä olevassa esimerkissä on UDP-paketin lähetys sekä vastaanotto. Lopuksi funktio palauttaa näytölle vastatun paketin.

# 6 TESTAUS, JATKOKEHITYS SEKÄ ONGELMAT

Laitteen käyttötestausta ei suoritettu. Prototyypin asennus valokuituvahvistimeen vaatisi oman piirilevyn suunnittelun tai valmiin näyttöpiirin tilausta. Työssä ei haluttu keskittyä erilaisten valmiiden näyttöpiirien tarkasteluun. Markkinoilla olisi ollut valmiiksi laitteeseen sopivia näyttöpiirejä, joissa on näppäimet mukana. Prototyypin kanssa käytetty näyttö ei olisi sopinut fyysisesti valokuituvahvistimeen.

## 6.1 Kehitystyön ongelmat

Toimeksiantajan tarjoamalla piirilevyllä oli valmiiksi toiselle laitteelle suunniteltu ohjelmisto. Ohjelmistosta olisi pitänyt poistaa kaikki ylimääräiset ohjelmat muistin säästämiseksi sekä ohjelman yksinkertaistamiseksi. Toisaalta osa ohjelmisto tarjosi hyvän pohjan, johon soveltaa uutta.

## 6.2 Jatkokehitys

Tuotantoa varten voitaisiin valita paremmin tähän tarkoitukseen sopiva mikrokontrolleri. Tähän riittäisi paljon suppeampi levy. Tärkeimmät ominaisuudet olisivat ethernet-portti ja useita vapaita nastoja. Vapaisiin nastoihin voitaisiin liittää haluttu nestekidenäyttö, näppäimet ja muita ylimääräisiä komponentteja, kuten lämpöanturi.

Verkko yhteyttä voitaisiin parantaa lisäämällä yksi ethernet-portti. Kahdella ethernet-portilla voitaisiin luoda kaksi eri yhteyttä. Tällä järjestelmällä voitaisiin luoda helposti yhteys kehitysalustalle sekä alustalta valokuituvahvistimelle. Kehitysalustalle voisi näin lisätä etäkäyttöliittymän ethernetin avulla.

Toinen vaihtoehto olisi lisätä ethernet-reititin laitteen sisälle. Näin saataisiin luotua yhteys suoraan kehityspiirilevylle sekä valokuituvahvistimelle.

Fyysisen käyttöliittymän parhaita puolia ovat ongelmanratkaisutilanteet. Kun valokuituvahvistin ei vastaisi ethernet-viesteihin, voitaisiin näppäimillä etsiä tarvittava tieto tai asettaa haluttu toiminto, jonka jälkeen vika olisi korjattu. Tämän voi välttää lisäämällä reset-näppäimen laitteen taakse, joka nollaisi ip-asetukset tunnettuihin asetuksiin.

Fyysinen käyttöliittymä voitaisiin korvata täysin käyttämällä ethernet-liikennettä. Tässä on etuna kustannussäästöt valmistuksessa. Fyysisen käyttöliittymän tarve tämän tapaisessa järjestelmässä ei ole suuri. Laitetta todennäköisesti käytetään etänä toiselta tietokoneelta.

## 6.3 Lopuksi

Vaikka toimeksiantaja tarjosi valmiin pohjan työn tekoon, niin se osoittautui silti hyvin haastavaksi. Koska kyseessä oli valmiiksi suuri systeemi, niin työn tekeminen vaati paljon opettelua. Puhtaan prototyypin suunnittelu sujui helposti ja nopeasti. Arduinolla luotu prototyyppi toimi hyvin ja sillä kyettiin arvioimaan, miltä lopullinen tuote voisi näyttää.

Koska työssä ei otettu huomioon laitteen valmistusta tuotannossa, jäi työstä pois eri komponenttien tarkastelu. Työ keskittyi ohjelman luomiseen, eikä erilaisten osien valitsemiseen. Projektissa käytetyn kehitysalustan aiheuttamista rajoitteista johtuen lopuksi ei voitu täysin tehdä alussa sovittua laitetta. Jos kehitysalusta mahdollistaisi, niin LCD-näytön lisääminen olisi yksinkertainen prosessi. Sama data vain lähetettäisiin suoraan näytölle. Oleellisin osa kehitettyä ohjelmaa on kommunikointi laitteen kanssa. Laitteelta saatavia tietoja voidaan helposti soveltaa moneen eri tarkoitukseen.

Jos tässä työssä tehty applikaatio olisi ollut ainoa laitteella pyörivä sovellus, niin käyttöjärjestelmää ei olisi tarvinnut käyttää.

# LÄHTEET

[1] Ala-aseman prosessori- ja ohjelmistokehitysympäristön kartoitus [www-dokumentti]. Saatavilla: https://www.theseus.fi/handle/10024/1398 2.5.2016

[2] https://fi.wikipedia.org/wiki/Nestekiden%C3%A4ytt%C3%B6 5.6.2016

[3] https://arduino-info.wikispaces.com/LCD-Blue-I2C 2.5.2016

[4] https://en.wikipedia.org/wiki/I%C2%B2C 3.6.2016

[5] http://www.twyman.org.uk/Fonts/ 2.5.2016

[6] https://www.arduino.cc/en/Main/ArduinoBoardYun 1.5.2016

[7] http://www2.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f4-series/stm32f407-417.html?querycriteria=productId=LN11 2.5.2016

[8] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ 9.5.2016

[9] Code red ”Getting Started with Red Suite 5” [www-dokumentti]. Saatavilla: http://support.code-red-tech.com/CodeRedWiki/NewInVersion5?action=AttachFile&do=get&target=Red_Suite_5_Getting_Started.pdf 1.5.2016

[10] https://www.embeddedartists.com/products/tools/tool_cr_redprobe.php 1.5.2016

[11] https://opensource.org/licenses 21.5.2016

[12] http://www.freertos.org/ 1.5.2016

[13] http://lwip.wikia.com/wiki/LwIP_Wiki 1.5.2016

[14] http://study-ccna.com/dhcp-dns/ 3.6.2016

[15] https://www.isc.org/downloads/dhcp/ 3.6.2016

[16] Brian W. Kernighan, Dennis M. Ritchie. ”The C Programming language”, Prentice hall PTR

[17] https://www.arduino.cc/en/Tutorial/HelloWorld 2.5.2016

[18] http://lwip.wikia.com/wiki/Raw/UDP 5.6.2016

[19] https://www.arduino.cc/en/Reference/SwitchCase 3.6.2016