ETÄOHJATTU LED-MATRIISINÄYTTÖ

Ville Syrjänen

Opinnäytetyö
Elokuu 2015
Kone- ja tuotantotekniikka
Kone- ja laiteautomaatio

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Kone- ja tuotantotekniikka
Kone- ja laiteautomaatio

VILLE SYRJÄNEN:
Etäohjattu LED-matriisinäyttö

Opinnäytetyö 54 sivua, joista liitteitä 16 sivua
Syyskuu 2015

Tämän opinnäytetyön tarkoituksena oli luoda etäohjattu LED-matriisinäyttö, jolla on mahdollista näyttää kuvia ja videoita lähiverkon kautta. Näyttö soveltuu käytettäväksi esimerkiksi messutilaisuuksissa ja muissa vastaavissa tapahtumissa mainosnäyttönä tai katseenvangitsijana. Työllä ei ollut erillistä tilaajaa, vaan se sai alkunsa tekijän omasta halusta perehtyä ohjelmoitavien logiikkapiirien ja lähiverkon toimintaan.

Tässä opinnäytetyössä keskityttiin komponenttien valintaan vaikuttaviin asioihin sekä kotelon suunnitteluun ja valmistukseen. Työssä perehdyttiin myös tärkeimpien komponenttien sisäiseen toimintaan, sekä ohjelmien tekemiseen FPGA-piiriä ja tietokonetta varten. Työn tuloksena oli toimiva näyttökokonaisuus, joka vastasi työn tavoitteita.

Asiasanat: LED-matriisi, etäohjattu, fpga

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Mechanical and Production Engineering
Machine Automation

VILLE SYRJÄNEN:
Remotely Controlled LED-Matrix Display

Bachelor's thesis 54 pages, appendices 16 pages
September 2015

The purpose of this thesis was to create a remotely controlled LED-matrix display which is able to show images and video via a local network. This display can be used as an advertising display or an eye-catcher for trade fairs and other similar events. This study was not commissioned by any company. Instead, the subject originated from the author's own desire to learn about programmable logic and local networks.

This project focused on the design and manufacture of the housing and the factors that affected the choice of components. Moreover, the internal workings of the main components, as well as writing programs for an FPGA integrated circuit and the computer that drives the display were further points of interest discussed here. The result of this work was a working LED-matrix display which corresponds with the main goals of this thesis.

Key words: led-matrix, remotely, controlled, fpga

SISÄLLYS

LYHENTEET JA TERMIT

ARP	Address resolution protocol
BGA	Ball grid array
CLB	Configurable logic block
FPGA	Field-programmable gate array
I/O	Input/Output
ICMP	Internet control message protocol
IGMP	Internet group management protocol
IOB	Input/output block
IP	Internet protocol
LED	Light emitting diode
MAC	Media access control
PLC	Programmable logic controller
PWM	Pulse-width modulation
RGB	Red green blue
SDA	Serial data out
SDI	Serial data in
SIPO	Serial in, parallel out
TCP	Transmission control protocol
TQFP	Thin quad flat pack
UDP	User datagram protocol

1 JOHDANTO

Tässä opinnäytetyössä oli tarkoitus valmistaa LED-matriisinäyttö jota ohjataan etäohjauksella lähiverkon kautta. Näyttö pystyy toistamaan videota ja kuvia. LED-matriisinäyttöä voidaan käyttää esimerkiksi mainostamiseen messuilla ja muissa vastaavissa tilaisuuksissa. Näyttöä voidaan käyttää myös katseenvangitsijana ja yleisöhoukuttimena. Työn aiheen valinta kohdistui matriisinäyttöön, sillä se oli oivallinen tilaisuus oppia käyttämään ohjelmoitavia logiikkapiirejä ja samalla kehittämään elektroniikka osaamista.

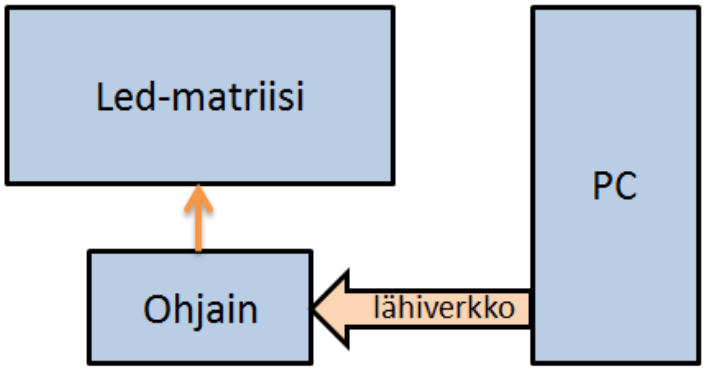
Opinnäytetyön tavoitteena oli tutustua ohjelmoitavaan logiikkaa ja lähiverkon toimintaan. Logiikan ja lähiverkon ohella tuli tutustuttua myös tuotekehitykseen ja elektroniikkaan. Työssä ei käytetty automaatiokäytössä yleisesti käytössä olevia PLC logiikoita, vaan erillistä piirilevylle asennettavaa FPGA-logiikkapiiriä. Kummatkin kuitenkin toimivat samalla periaatteella ja käyttäjän tehtäväksi jää kuvailla piirin logiikan toiminta. FPGA-piirin käyttöä puoltavat sen edullisempi hinta ja pienempi koko. FPGA-piiri oli myös helpompi integroida tarvittavaan ohjaukseen, sillä sen käyttöjännite on 3.3V kuten suurimmalla osalla muissakin komponenteissa. PLC logiikat ovat yleisemmin 24V käyttöjännitteelle.

Opinnäytetyössä on käyty läpi tärkeimpien komponenttien valinta ja valintaan vaikuttavat asiat. Työssä perehdytään myös joidenkin komponenttien sisäiseen toimintaan ja rakenteeseen. Teorian jälkeen kerrotaan kuinka ensimmäisestä testiversiosta päästään lopullisen piirilevyn suunnitteluun ja valmistamiseen. Tämän jälkeen käydään läpi kotelon rakentaminen ja lopuksi perehdytään ohjelmointiin ja siihen käytettäviin sovelluksiin.

2 TYÖN MÄÄRITTELY

LED-matriisinäyttö on oivallinen tapa rakentaa näyttävä valaistus tai mainostaulu. Matriiseja voidaan myös käyttää isojen videoseinien tekoon. Matriisien käyttöä tavallisien nestekidenäyttöjen sijasta tukee se, että ne on helppo räätälöidä kyseessä olevaa sovellusta varten. Työn aiheena olevan kokoluokan matriisinäyttöjen ohjaus on yleensä toteutettu mikrokontrollerilla ja kuvan syöttäminen tapahtuu joko sarjaportin tai USB yhteyden välityksellä. Näyttö voi olla myös ns. ”Stand alone” malli missä kuvat tai lyhyt videonpätkä tulee ohjaimen omasta muistista. Opinnäytetyöni eroaa edellä mainituista näytöistä sillä, että siinä on logiikka ohjaus ja etäkäyttö lähiverkon kautta.

Valmistamani kokonaisuus rakentuu LED-matriisista sekä sen ohjaimesta. Näyttö liitetään lähiverkkoon jonka välityksellä sille voi lähettää kuvia ja videota. Näytön ohjainpiiri luo TCP-serverin (Transmission Control Protocol), johon esim. tietokoneella voidaan liittyä ja tätä kautta lähettää dataa näytölle. Ohjainpiiri vastaanottaa dataa ja tallentaa sen puskurimuistiin, josta se sitten ajetaan näytölle. Ohjainpiiri vastaa serverin toiminnasta, datan vastaanottamisesta, puskurimuistista sekä matriisin ohjaamisesta. Systeemin karkea lohkokaavio on esitetty kuviossa 1.



KUVIO 1. Lohkokaavio

Tietokoneen tai muun vastaavan lähiverkkoon siirretyn laitteen ohjelmiston tehtävä on skaalata haluttu kuva oikean kokoiseksi LED-matriisia varten, sekä myös muuttaa värit oikeanlaisiksi. Tietokoneelle tai mobiililaitteelle on mahdollista tehdä ohjelmisto, joka toistaa LED-matriisin kautta kuva- tai videotiedostoja tai esimerkiksi suoraa näytönkaappaus kuvaa tietokoneen ruudulta. Tässä työssä keskityn kuitenkin vain tietokoneen

ohjelmistoon, joka pystyy kaappaamaan reaaliaikaista kuvaa tietokoneen näytöltä ja lähettämään sitä LED-näytölle. Valitsemaani ohjelmistoratkaisuun päädyin siksi, että tämänlaisella ohjelmistolla voidaan helposti toistaa mitä tahansa videoita minkä tahansa suoratoisto palvelun kautta. Jos kuitenkin halutaan toistaa videoita tai kuvia suoraan tietokoneelta eikä internetistä, voidaan ottaa näytönkaappaus tietokoneen omasta mediatoistimesta.

Näytön tekniset tavoitteet määräävät pitkälti käytettyjen osien valinnan, kuten esimerkiksi LED-matriisin ja sen ohjauspiirin valinnan. Tärkeimmät tavoitteet koskevat itse näytön toimintaa ja sen käyttömukavuutta. Näytön päivitystaajuuden tulisi olla vähintään 100 Hz jotta kuva olisi tasainen eikä silmä erota minkäänlaista välkettä. Videokuvan kuvataajuuden olisi myös hyvä olla vähintään 25 kuvaa/s jotta video olisi tasaista. Käytön tulisi myös olla helppoa ilman turhaa asetusten säätämistä ja konfigurointia.

3 TÄRKEIMPIEN KOMPONENTTIEN VALINTA

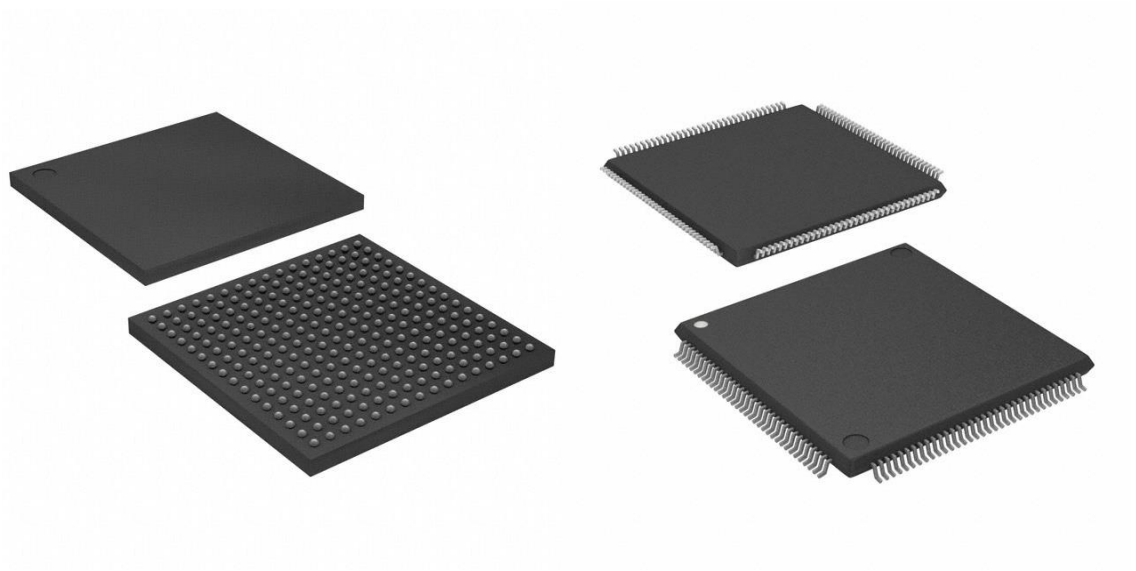
3.1 LED-matriisi

LED-matriisin valintaan vaikutti pääosin ledien riittävä määrä ja hinta. Matriisin rakentamiseen olisi voinut käyttää valmiita 8x8 ledin kokoisia elementtejä, mutta koska elementit ovat halvimmillaan noin 4,5 euroa kappaleelta, olisi 64x32 kokoiselle matriisille tullut hintaa 144 euroa. Seuraava vaihtoehto olisi rakentaa matriisi itse erillisistä RGB-ledeistä, jolloin tarvittava RGB-ledien määrä olisi 64x32 matriisille 2048 lediä. Käytettäessä pintaliitos ledejä, olisi hinnaksi pelkille ledeille tullut noin 42 euroa. Tarvitsee kuitenkin muistaa, että jos matriisi rakennetaan erillisistä ledeistä tai edellä mainituista elementeistä, tarvitsee lisäksi hankkia ledien vaatimat ohjauspiirit. Ledit sekä ledien vaatima elektroniikka olisi myös vaatinut erittäin suuren piirilevyn, jonka teettäminen olisi maksanut helposti päälle 100 euroa. Hinnan ja käytettävyyden puolesta parhaimmaksi vaihtoehdoksi osoittautui Adafruit Industries yhtiön tuotteisiin kuuluva LED-matriisi. Kyseiset paneelit sisältävät RGB-ledien lisäksi niiden ohjaamiseen tarvittavan elektroniikan, mikä helpottaa huomattavasti suunnittelua ja vähentää tarvittavien osien määrää. Kuvassa 1 on esitetty 16x32 kokoinen matriisi kuvattuna takaapäin, kuvassa näkyy paneelin LED-ohjaimet. Adafruit myy verkkosivuillaan 16x32 ja 32x32 kokoisia paneeleita, joten paneeleita tarvittaisiin vähintään kaksi jotta saavutettaisiin haluttu 64x32 ledin kokonaismäärä. Lopulta kuitenkin löysin Kiinassa sijaitsevan myyjän valikoimista 64x32 kokoisen paneelin, joka vastaa Adafruit yhtiön tuotteita ja hinta toimituskuluineen oli hieman alle 100 euroa.

KUVA 1. 16x32 paneeli takaa (Adafruit, muokattu)

3.2 Ohjauspiiri

Yksi ohjauspiirin kriteereistä oli se, että sen täytyy olla riittävän nopea päivittämään LED-matriisia jotta ihmissilmä ei huomaisi minkäänlaista välkettä kuvassa. Ohjauspiirin tulee myös pystyä hoitamaan datan lukeminen lähiverkosta, sekä sen tulee pystyä reaaliajassa kääntämään data haluttuun muotoon. Ensimmäisenä mieleen tulisi toteuttaa ohjaus mikro-ohjaimella. Mikro-ohjaimen käyttöä puoltavat se, että niitä on saatavissa useaa eri mallia eri ominaisuuksilla ja eri hintaryhmissä. Mikro-ohjaimia on myös suhteellisen helppo ohjelmoida C++ ohjelmointikieltä käyttäen. Tässä sovelluksessa kuitenkin vaaditaan ohjaukselta paljon, sillä ohjaimen tulisi pystyä hoitamaan montaa eri toimintoa lähes samanaikaisesti. Rinnakkaisen toiminnan tarpeellisuus johtuu siitä, että vaikka sekventaalisesti suoritettava ohjelma pystyisikin ylläpitämään matriisin tarvitsemaa päivitysnopeutta, joutuisi se aina keskeyttämään matriisin päivityksen datan lukemisen ja kääntämisen ajaksi. Rinnakkaisen toiminnan takia valitsinkin ohjelmoitavan logiikkapiirin. Ohjelmoitavaksi logiikaksi valitsin Alteran valmistaman Cyclone sarjaan kuuluvan FPGA-piirin. Valintani kohdistuin kyseisen valmistajan tuotteisiin, koska heidän tuotteitaan oli helposti saatavissa ja valmistajan verkkosivuilta oli tarjolla paljon opetusmateriaalia piirin käyttöönottoa varten. Sovellukseeni päädyin valitsemaan Cyclone II sarjaan kuuluvan EP2C5T144C7N piirin. Valintani kohdistui kyseiseen piiriin siksi, koska siinä oli riittävästi sisäisiä logiikkaelementtejä ja käyttömuistia tarvittavan toiminnan aikaansaamiseksi. Vaikka Cyclone II sarja onkin Alteran vanhempaa mallistoa, on se kuitenkin vielä tuotannossa ja sitä voi vielä käyttää valmistajan mukaan uusiin sovelluksiin (Altera). Cyclone II sopi paremmin käyttööni kuin uudet, koska se on valmistettu TQFP koteloon ja on näin ollen vielä mahdollinen juottaa käsin, toisin kuin uudempien piirien BGA kotelot. TQFP kotelossa liitospinnit ovat kotelon ympärillä, tämän takia se on mahdollista juottaa käsin. BGA kotelossa pinnit ovat kotelon pohjassa ja ne ovat erittäin pieniä puolipallon muotoisia kontakteja, joka tarkoittaa että juottamiseen olisi käytettävä juotostahnaa ja uunia. BGA kotelon haittana on myös pinnien tiheys sillä tämä vaatisi huomattavasti tarkemmin tehdyn piirilevyn, joka olisi tullut paljon kalliimmaksi teettää. TQFP ja BGA tyyppien koteloiden erot ovat esitetty kuviossa 2. BGA kotelo on kuvassa vasemmalla ja TQFP kotelo oikealla.



KUVIO 2. BGA ja TQFP koteloiden erot (Digikey, muokattu)

3.3 Kommunikointi

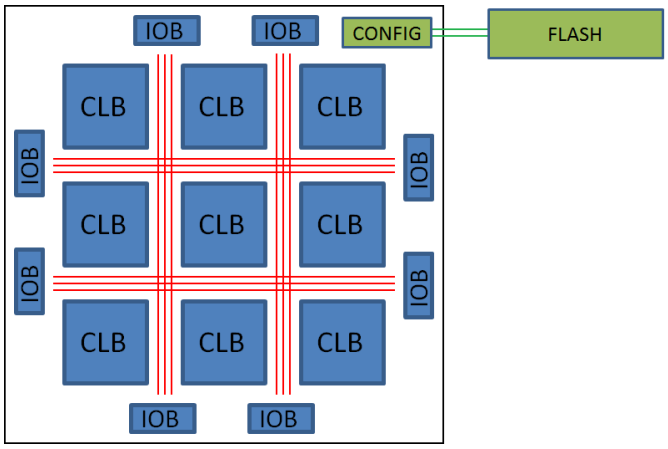
Lähiverkossa tapahtuva kommunikointi olisi ollut mahdollista hoitaa sisäisesti FPGA-piirissä, mutta päädyin kuitenkin käyttämään erillistä mikropiiriä kommunikointiin. Erillisen piirin käyttö helpotti huomattavasti ohjelmiston kehitystä, sillä suurin osa kommunikoinnista on valmiiksi hoidettu piirin sisällä. Tarvittavan toiminnon suorittamiseen oli tarjolla useita piirejä ja suurin osa niistä olisi käynyt toteutukseen. Valinnassani kuitenkin päädyin valmistajan WIZnet Products valmistaman W5500 piirin, sillä se oli halpa ja valmistajalta löytyi erittäin hyvin infoa ja opastusta piirin käyttöön. Piiriä varten valmistajalla oli myös tarjolla valmis kirjasto mikro-ohjaimia varten. Valmista kirjastoa en pystynyt käyttämään FPGA-piirin kanssa, mutta kirjaston koodeja tutkimalla oli helppo saada selville tarvittavat komennot. Merkittävä tekijä oli myös se, että piiristä oli tarjolla valmis testialusta jossa piirin lisäksi oli kaikki sen käyttöön vaadittava elektroniikka. Testialustan kytkentäkaavio ja osaluettelo oli myös vapaasti saatavilla valmistajan sivuilta, joten se oli helppo sisällyttää lopulliseen piirilevyyn ja piirin toiminnasta pystyi olemaan varma.

4 KOMPONENTTIEN TOIMINTA

4.1 FPGA-piiri

FPGA-piiri on ohjelmoitava elektroniikkakomponentti. Käyttäjän on mahdollista implementoida minkä tahansa digitaalisten piirien toiminta yhden FPGA-piirin sisään, rajana on yleensä vain mielikuvitus ja toiminnon monimutkaisuus. FPGA-piiri koostuu sen sisällä olevista logiikkaelementeistä, joita piiristä ja valmistajasta riippuen voi olla jopa satojatuhansia. Logiikkaelementtien toteuttama toiminto ja logiikka elementtien väliset kytkennät ovat määriteltävissä laitteistokuvauskielen avulla. Jotkut FPGA-piirit saatattavat sisältää myös ei-ohjelmoitavia osia, kuten esimerkiksi suotimia, käyttömuistia tai valmiita suorittimia. Suurin osa FPGA-piireistä unohtaa konfigurointinsa heti kun virta katkaistaan. Käytännössä tämä tarkoittaa sitä, että piiri tarvitsee toimiakseen ulkoisen ohjelmamuistin josta se saa aina käynnistyessään tarvittavat tiedot logiikkaelementtien kytkemiseen. Käynnistystä varten piirissä on valmiina sisäinen lohko, joka aina piirin käynnistyessä lataa ohjelmamuistista tarvittavat tiedot logiikkaelementtien kytkemistä varten. (Wolf 2004:105-112.)

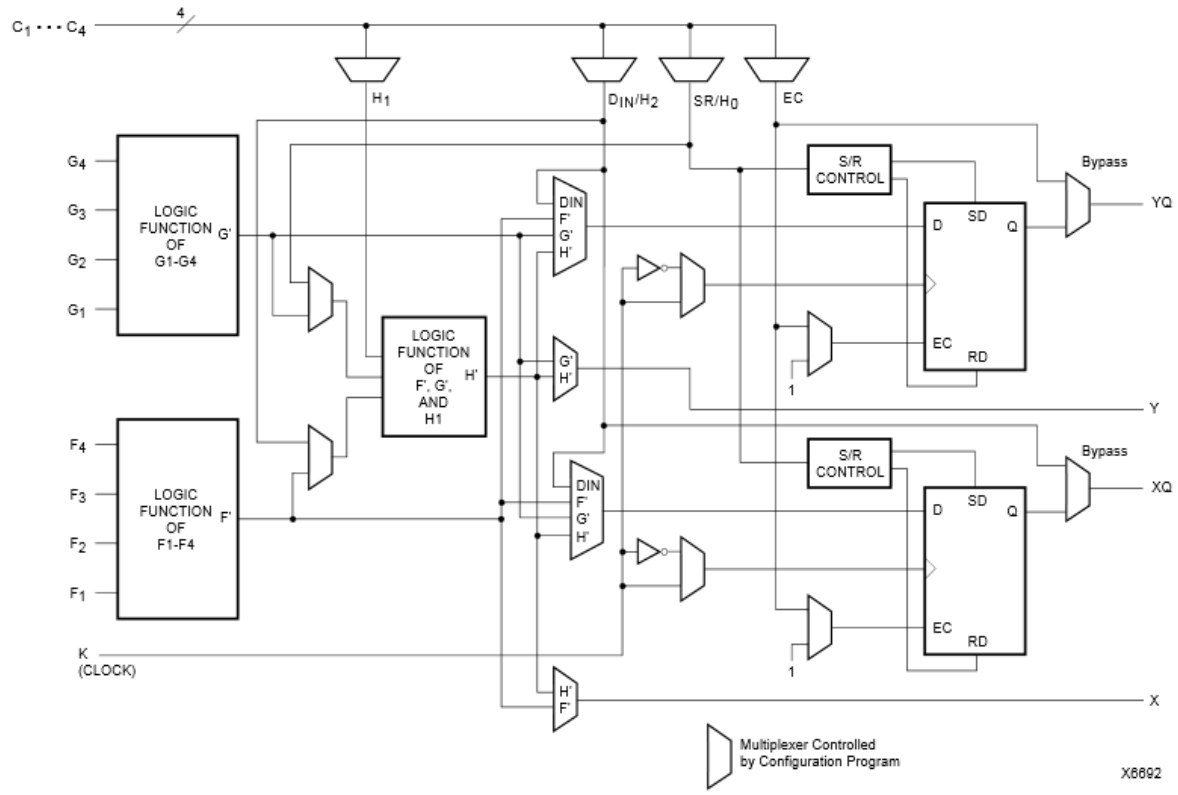
Kuviossa 3 on esitetty karkeasti FPGA-piirin sisäinen rakenne. Kuviossa kirjaimilla CLB merkityt lohkot kuvaavat logiikkaelementtejä, joita muokkaamalla ja yhdistelemällä saadaan aikaan laitteistokuvauskielellä määritelty logiikka. Kirjaimin IOB merkityt lohkot liittävät sisään- ja ulostulot logiikkaan. Logiikka- ja I/O-elementit ovat yhteydessä toisiinsa sisäisten johteiden avulla, nämä johteet sisältävät suuren määrän kytkinmatriiseita, joiden avulla saadaan haluttua toimintoa vastaava kytkentä.



KUVIO 3. FPGA-piirin sisäinen rakenne

4.1.1 Logiikkaelementit

FPGA-piirin sisältämät logiikkaelementit koostuvat yleensä kytkinmatriisista missä on useita sisääntuloja, logiikkaelementissä on myös joitakin valintakomponentteja kuten multiplekseri, kiikkuja ja LUT (LookUp-Table). (Wolf 2004:113.) Kyseisten komponenttien takia yksi logiikkaelementti on hyvin muunneltavissa eri tarkoituksia varten. Yhdestä elementistä voidaan rakentaa esimerkiksi osa kombinatorista logiikkaa, sitä voidaan myös käyttää RAM-muistina tai siirtorekisterinä. Logiikkaelementtien toiminta ja sisäinen kytkentä määräytyy piirin käynnistyksen yhteydessä, jolloin se konfiguroi itsensä käyttäen ohjelmamuistissa olevaa tietoa. Kuviossa 4 on esitetty Xilinx nimisen valmistajan XC4000 sarjaan kuuluvien piirien logiikkaelementin lohkokaavio.

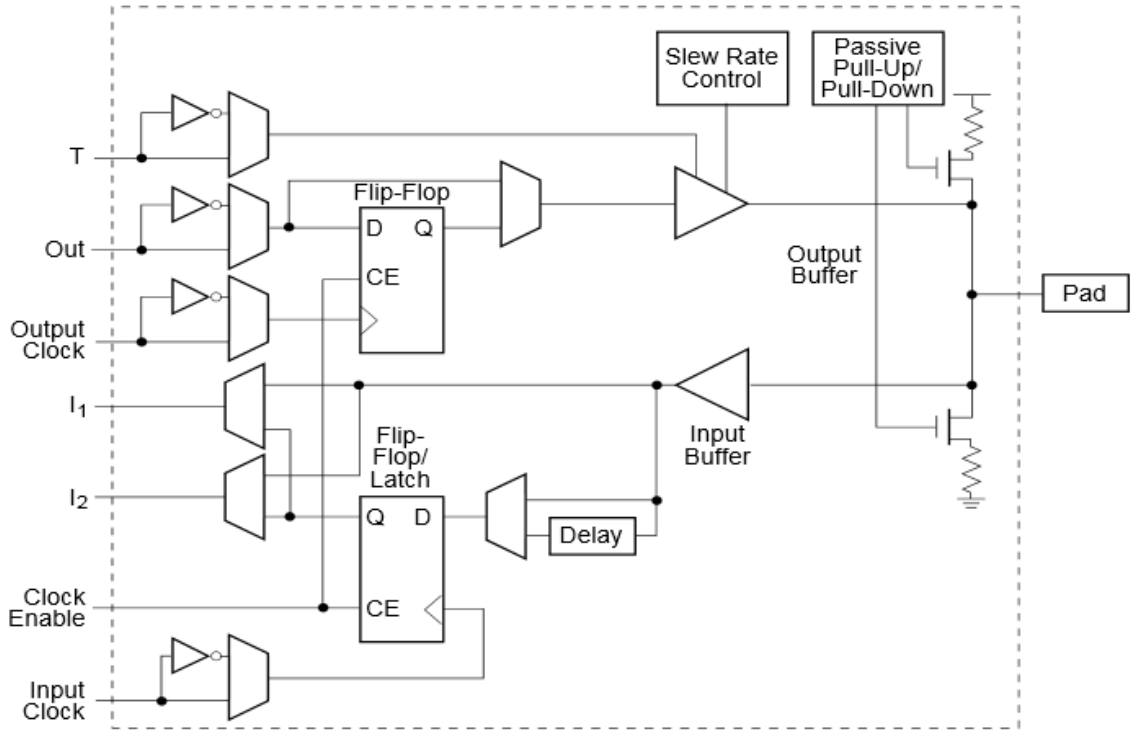


KUVIO 4. Logiikkaelementin lohkokaavio (Xilinx)

4.1.2 I/O-elementit

Konfiguroitavat I/O-elementit tarjoavat linkin sisäisen logiikan ja ulkoisten lähtöjen välillä. Jokainen ulkoinen lähtö on määriteltävissä sisään- tai ulostuloksi, tai samaan aikaan kummaksikin jolloin se voi vuorotellen lähettää ja vastaanottaa dataa. Ulkoisiin lähtöihin on määriteltävissä sisäiset ylös- ja alasvetovastukset, jolloin niitä ei erikseen

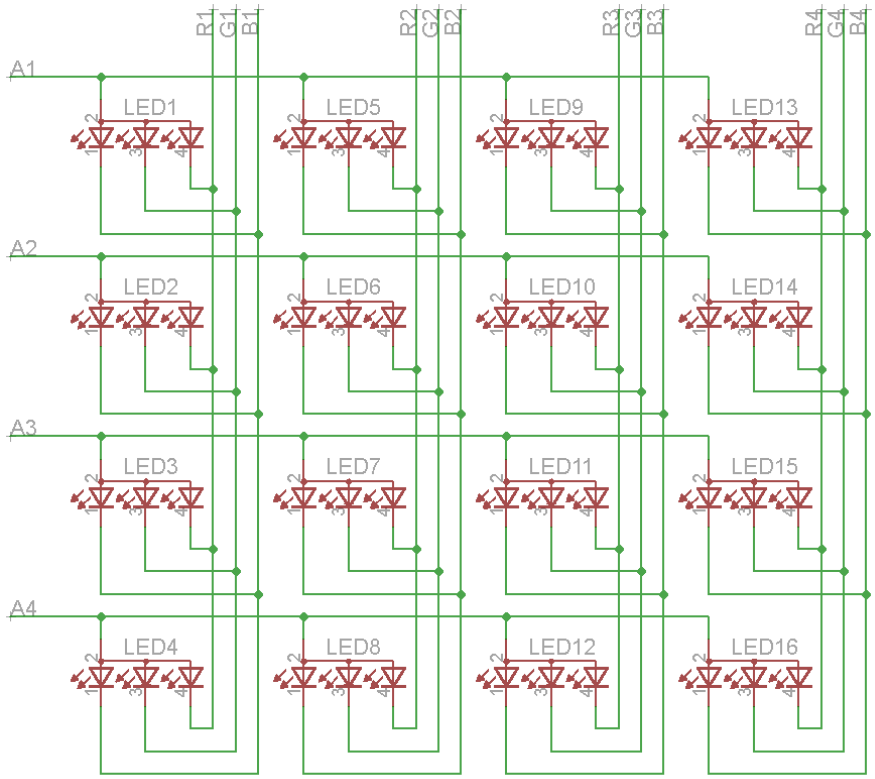
tarvitse lisätä ulkoiseen elektroniikkaan. Lähtöjen jänniterajat ovat myös määriteltävissä joko 1.2V tai 3.3V logiikalle. Kuviossa 5 on esitetty Xilinx XC4000E sarjan I/O-elementin lohkokaavio. I/O-elementin sisääntulot I1 ja I2 voidaan reitittää logiikalle joko suoraan sisääntulopuskurilta tai vaihtoehtoisesti sisääntulorekisterin kautta. Mikäli sisääntulo tulee suoraan puskurilta, lukee logiikka sen heti kun sisääntulon taso muuttuu. Reitittäessä sisääntulo rekisterin kautta, saa logiikka sisääntulon arvon vasta silloin kun rekisterille syötetään kellopulssi. Ulostulojen signaalit voidaan siirtää joko sellaisenaan puskurin kautta lähtöön, ne voidaan myös invertoida tai reitittää rekisterin kautta.



KUVIO 5. I/O-elementin lohkokaavio (Xilinx)

4.2 LED-matriisi

LED-matriisi on elementti missä useampi LED-valo on yhdistetty riveihin ja sarakkeisiin muodostaen näin suorakulmaisen taulukon. LED-matriiseita on saatavilla joko yksi tai monivärisinä. Ledien suuresta määrästä johtuen niitä ei voida pitää samanaikaisesti päällä sillä tämä vaatisi paljon virtaa, sekä ennen kaikkea erittäin paljon ulostulokanavia ohjaavalta piiriltä. LED-matriisin sisäinen kytkentä on esitetty kuviossa 6. Kuviosta voidaan nähdä, että kaikkien samalla rivillä olevien ledien anodit ovat kytketty yhteen, kuten myös samassa sarakkeessa olevien samanväristen ledien katodit.

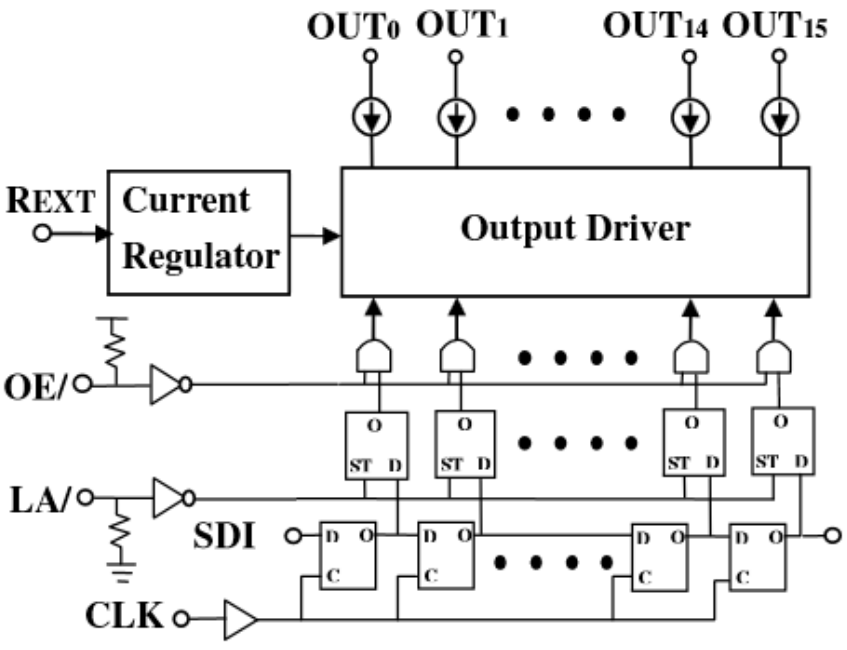


KUVIO 6. Matriisin sisäinen kytkentä.

Ledien ohjaaminen tapahtuu siten, että jokainen sarake on kytketty LED-ajurinpiiriin omaan lähtöönsä. Jokaiselle värille on oma LED-ajurinsa siksi, että värin kirkkaus voidaan säätää sopivaksi. Kirkkauden säätö on tarpeellinen koska ledien fyysisien ominaisuuksien takia kunkin värin valovoima on erisuuruinen, vaikka ledien läpi kulkeva virta olisikin sama. Ajurit ovat yleensä 16-kanavaisia, joten tämä tarkoittaa sitä, että jokaista väriä kohden on yleensä useampi peräkkäin ketjutettu ajuri. Rivien ohjaus tapahtuu dekooderipiirin avulla, joka ottaa sisäänsä binäärilukuja ja kytkee lukua vastaavaan sarakkeeseen jännitteen. Kuvan luominen tapahtuu siten, että LED-ajureille syötetään ensimmäisen rivin data. Datan kirjoittamisen jälkeen tieto siirretään ledeille ja samaan aikaan dekooderi ohjaa jännitteen ensimmäiselle riville jolloin ledit syttyvät. Tämän jälkeen ajureille ajetaan seuraavan rivin data. Toisen rivin datan saavuttua, kytketään se ledeille samalla hetkellä kun dekooderi vaihtaa jännitteen kyseiselle riville. Samalla periaatteella käydään läpi kaikki rivit, jonka jälkeen kierto aloitetaan alusta. Tällä periaatteella ohjatussa matriisissa palaa kerrallaan vain yhden rivin ledit, joka alentaa huomattavasti virrankulutusta. Kierron ollessa tarpeeksi nopea, ei ihmissilmä voi erottaa välkettä ja aivoihin muodostuu illuusio yhtenäisestä kuvasta.

4.2.1 LED-ohjaimien toiminta

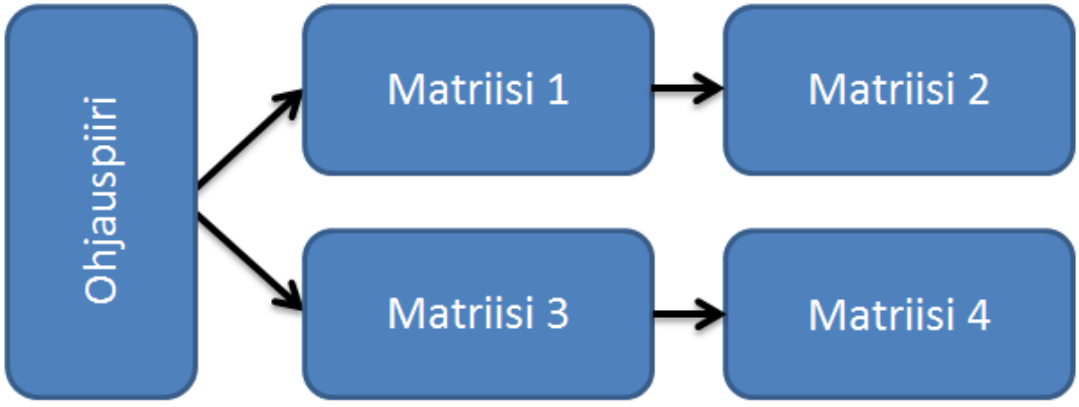
LED-matriisin käyttämät ajurit ovat 16-kanavaisia siirtorekisterin tyyppisiä ajureita, joissa on lisäksi virtaregulaattori säätämässä ledeille sopivan virran. Piirin sisäinen toiminta on esitetty kuviossa 7. Piirin siirtorekisteri on SIPO-tyyppin (Serial in, parallel out) rekisteri. SIPO tarkoittaa sitä, että data tulee rekisteriin sisään sarjamuotoisena ja lähtee ulos rinnakkaisena, eli vain muutamalla datalinjalla voidaan ohjata montaa eri lähtöä. Tämän seurauksena data tuodaan sisään bitti kerrallaan kuviossa esiintyvään SDI tuloon ja siirretään eteenpäin jokaisella CLK tuloon tulevan kellopulssin nousevalla reunalla. Rekisterin saatua kaikkien ledien data, siirretään tieto led-ohjaukselle LA/ tulon laskevalla reunalla. Ledien kirkkautta voidaan hallita PWM-signaalilla, joka tuodaan OE/ tuloon. PWM ohjauksen periaatteena on se, että valodiodi sytytetään ja sammutetaan jatkuvasti erittäin nopeasti. Pulssien taajuus on aina sama, mutta muuttamalla pulssinleveyttä, eli prosenttia kuinka kauan LED on päällä, voidaan määrittää valon kirkkaus. Ohjaimien rakenteen ansiosta niitä on mahdollisuus ketjuttaa peräkkäin lähes rajaton määrä. Ketjuttamisen ainoa este on tiedonsiirtonopeus, sillä mitä enemmän ajureita on peräkkäin, sitä kauemmin datan siirto kestää.



KUVIO 7. Led-ohjaimen lohkokaavio (StarChips Technology)

4.2.2 Matriisin toiminta

Käyttämäni LED-matriisi on 64x32 kokoinen ja se on jaettu neljään32x16 ledin matriisiksi. Matriisi on monivärinen ja siinä on käytetty RGB-ledejä. RGB-LED on puolijohde komponentti, missä samaan koteloon on istutettu punainen, sininen ja vihreä LED-elementti, yhdistelemällä näitä kolmea väriä saadaan aikaiseksi kaikki mahdolliset värit. Suuren LED määrän takia jokainen pienempi 32x16 osa on jaettu kahteen pienempään osaan, ensimmäinen osa on 1-8 rivien ledit ja toinen osa on 9-16 rivien ledit. Rivejä ohjaa yksi 8-kanavainen dekooderi, mikä tarkoittaa että ensimmäisen ja yhdeksännen rivin ledit ovat samaan aikaan päällä, sama toistuu myös toisen ja kymmenen rivin ledeillä jne. aina kahdeksanteen ja kuudenteentoista riviin. Rivien yhdistämisen ansiosta useampi rivi on päällä samaan aikaan ja tällä saadaan luotua tasaisempi kuva jonka välke on vaikeampi huomata. Erilliset 32x16 matriisit ovat yhdistetty toisiinsa kuvion 8 osoittamalla tavalla.

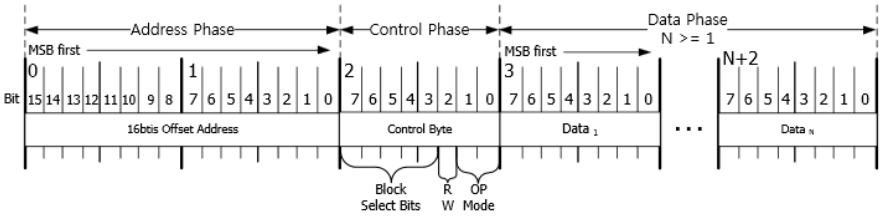


KUVIO 8. Matriisien kytkentä.

Jokainen matriisi tarvitsee toimiakseen 12 signaalia. LED-ajureita varten on 3 yhteistä signaalia, CLK, OE ja LA. CLK on kellosignaali joka saa datan liikkumaan eteenpäin ajureissa, OE signaali taas määrittää onko ledit päällä vai ei, sekä LA siirtää datan siirtorekisteriltä ledien ulostuloajuriin. LED-ajureille tulee myös yhteensä kuusi datasignaalia, kaksi kutakin väriä kohden, toinen rivejä 1-8 varten ja toinen rivejä 9-16. Näiden lisäksi rivejä ohjaava dekooderi tarvitsee 3-bittisen tulon jolla se ohjaa 8 eri ulostuloa. Kaikkien 4 matriisin CLK, OE, LA ja dekooderin signaalit on kytketty yhteen. Matriisien 1 ja 2 datasignaalit on ketjutettu peräkkäin, kuten myös matriisien 3 ja 4.

4.3 W5500 Ethernet-ohjain

W5500-piiri toimii linkkinä ohjainpiirin ja lähiverkon välillä. Piiri tukee TCP, UDP ja ICMP protokollia, joiden avulla datan siirto tapahtuu. Piiri tukee myös osoitteiden selvittämiseen ja useiden asiakkaiden liittymiseen käytettäviä protokollia. Näitä protokollia ovat esimerkiksi IPv4, ARP ja IGMP. Piirissä on valmiina sisäinen 32 kilotavun puskurimuisti lähiverkossa kulkevia paketteja varten. W5500 on varustettu kahdeksalla itsenäisellä kannalla kommunikointia varten ja jokaiselle kannalle voidaan määrittää tietty osa muistista lähtevää ja tulevaa dataa varten. Mikäli liikennettä ei tarvitse ohjata kuin yhden kannan kautta, voidaan muut sulkea ja osoittaa kaikki muisti tällä yhdelle kannalle. W5500 piiri hoitaa automaattisesti lähiverkon kautta tulevien datapakettien käsittelyn ja tallentaa datan puskurimuistiin mistä käyttäjän on se helppo lukea. Piirille tarvitsee vain kertoa MAC ja IP osoite, mikä tietoliikenneprotokolla on käytössä sekä asettaa kantojen muistivaraukset ja avata halutut kannat. Tämän jälkeen tarvitsee vain seurata onko dataa vastaanotettu ja lukea se. W5500 kommunikoi ohjauspiirin kanssa käyttäen SPI väylää ja sen maksimi kellonopeus on valmistajan mukaan 80 MHz. SPI väylän käyttämässä datakehyksessä on vakiomäärä osoitebittejä, mutta dataa kirjoittaessa tai luettaessa kehykseen pituus voi vaihdella. (WIZnet.) Kuviossa 9 on esitetty tiedonsiirtoon käytettävän datakehyksen rakenne.



KUVIO 9. SPI kehyksen rakenne (WIZnet)

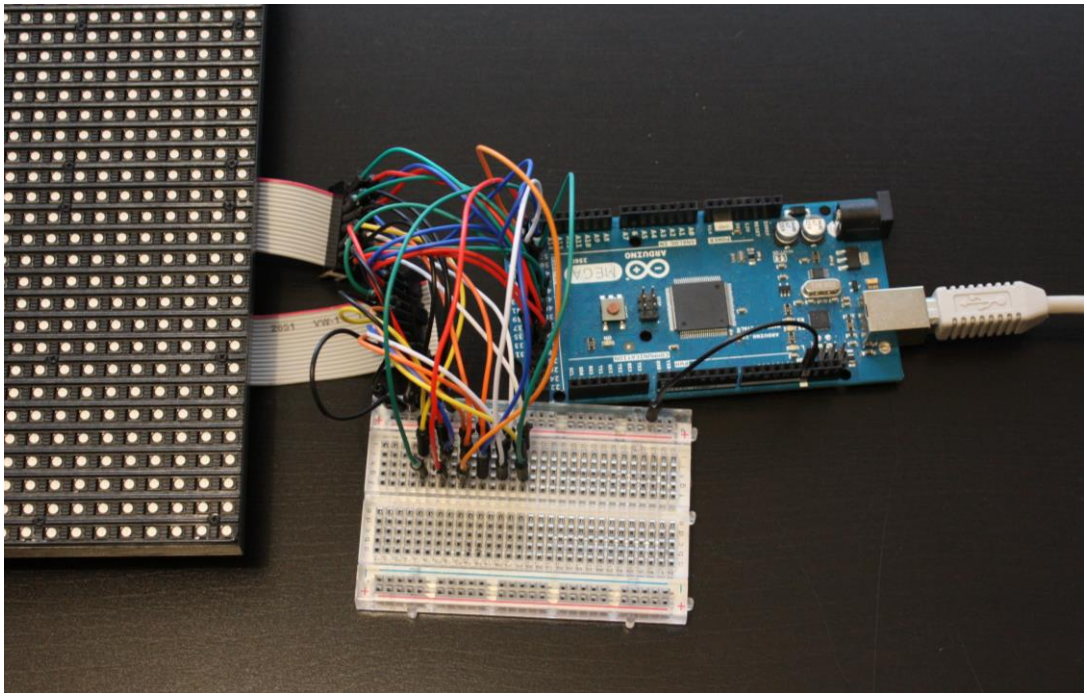
SPI kehys koostuu 16 bitistä joilla määritetään datan osoite, 8 bitistä jotka määrittävät halutun toiminnan ja data osuudesta joka on pituudeltaan vähintään 8 bittiä. Osoite vaihe määrittää dataa luettaessa ja kirjoittaessa puskurimuistin muistipaikan tai muistipaikan piirin sisäisessä rekisterissä jonka avulla piiri konfiguroidaan ja sen statusta voidaan seurata. Toiminnan määrittelevässä vaiheessa piirille kerrotaan halutaanko manipuloida yleisiä asetuksia, kannan asetuksia, halutaanko lukea kannan tai piirin status vai halutaanko lukea tai kirjoittaa dataa puskurimuistiin. Toiminnan määrittelevässä vaiheessa kerrotaan myös onko käytössä vakio datan pituus vai vaihtuva pituus. Dataosuudessa joko luetaan tai kirjoitetaan data aikaisemmin määrättyyn osoitteeseen.

5 TOTEUTUS

Työn toteuttamisen aloitin määrittelemällä halutut ominaisuudet ja valitsemalla komponentit. Komponenttien valinnan jälkeen oli aika hankkia FPGA-piirin ja W5500-piirin koealustat, kuten myös itse LED-matriisi prototyyppiä ja testailua varten. LED-matriisi ja FPGA-piirin koealustan tilasin eBay verkkohuutokaupasta hyvän saatavuuden ja hinnan takia. W5500-piirin koealustan tilasin TME nimisestä elektroniikkaliikkeestä joka sijaitsee Puolassa.

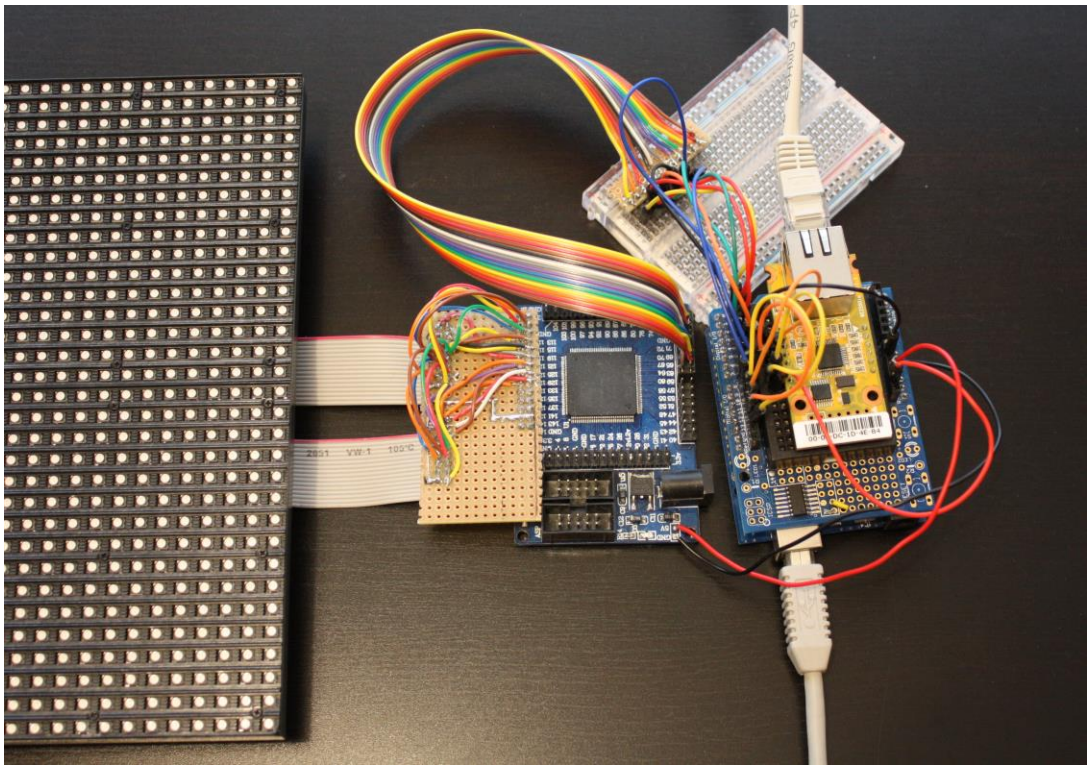
5.1 Prototyyppi

Prototyypin tekemisen aloitin LED-matriisin toimintaan tutustumalla. Matriisin tarvitsemasta datasta ja sen syöttötavasta ei ollut tarjolla riittävän yksityiskohtaista tietoa, joten ensimmäiseksi oli tutustuttava siihen. Matriisin ohjausta kehittäessä käytin Arduino-merkkistä mikro-ohjain alustaa, sillä Arduinon vaatima koodi on huomattavasti nopeampi kehittää kuin FPGA-piirin vaatima. Saatuani valmiiksi toimivan mallin matriisin vaatimasta koodista Arduinolle, oli se helppo rakentaa uudestaan FPGA-piirin vaatimaksi laitteistokuvauskieleksi. Kuvassa 2 on esitetty LED-matriisi joka on kytketty Arduino Mega mikro-ohjain alustaan.



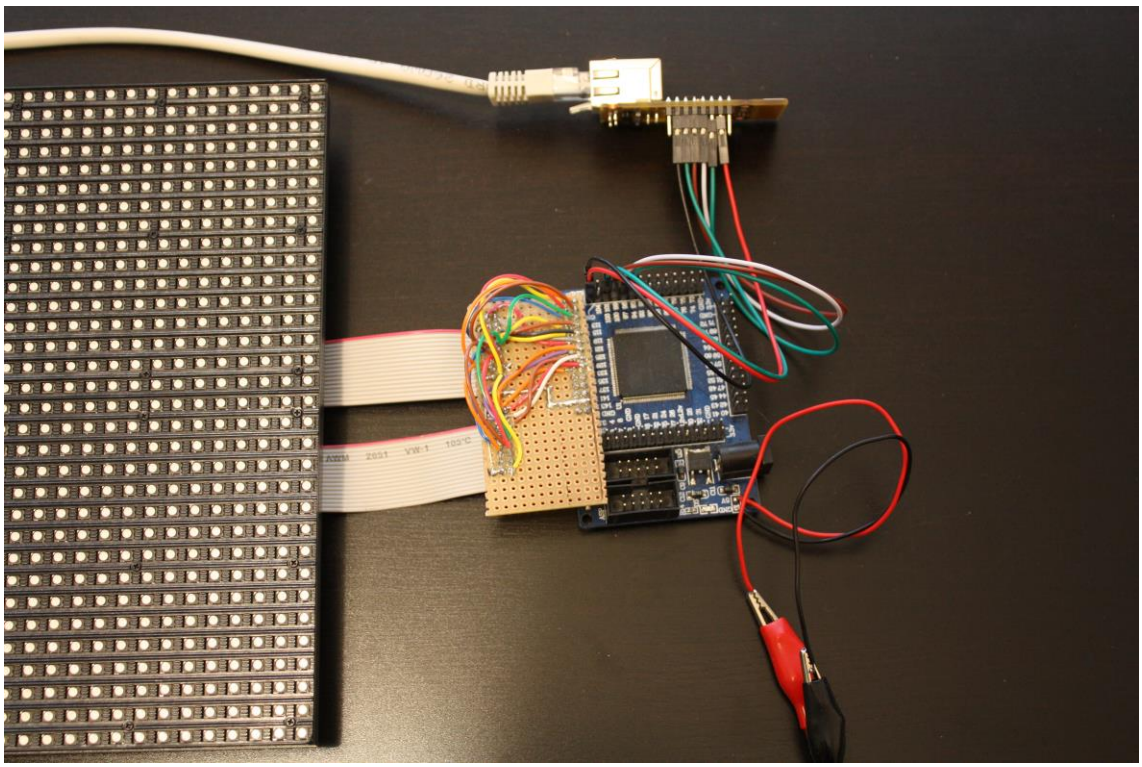
KUVA 2. LED-matriisi ja Arduino Mega

Matriisin vaatiman koodin valmistuttua oli seuraavaksi vuorossa tutustua W5500-piirin toimintaan. Piirille löytyi valmis aliohjelmakirjasto Arduinolle, joten helpoin tapa oli aloittaa testaus sen kanssa. Työ jatkui samaan tapaan kuin matriisin kanssa, ensin toimiva ohjelma Arduinon avulla, joka sitten muutetaan sopivaksi FPGA-piirille. Testiohjelma lukee datan lähiverkosta ja syöttää sen matriisille. Kuvassa 3 on esitetty testikokoonpano missä näkyy osa matriisista, FPGA-piirin testialusta, Arduino UNO, sekä W5500-piirin testialusta.



KUVA 3. Testikokoonpano matriisille.

Lopullisen tuotteen olisi voinut toteuttaa kyseisen kokoonpanon kaltaisesti, missä FPGA-piiri ohjaa matriisia ja mikro-ohjain lähiverkkokommunikointia W5500-piirin avulla. Tämä kuitenkin olisi tarkoittanut sitä, että elektroniikka olisi vaatinut enemmän komponentteja ja olisi näin ollen tullut kalliimmaksi. Koska mahdollisuutena oli sisällyttää mikro-ohjaimen toiminta samaan FPGA-piiriin kuin matriisin ohjaus, tein viimeisen prototyyppiversion poistamalla Arduinon kokoonpanosta. Ohjauksien integrointi yksinkertaisti huomattavasti prototyyppiä, tämän voi nähdä kuvassa 4, missä on esitetty matriisi ja W5500-piirin koealusta, mitkä ovat liitetty FPGA-piiriin.



KUVA 4. FPGA ohjaamassa matriisia ja W5500-piiriä

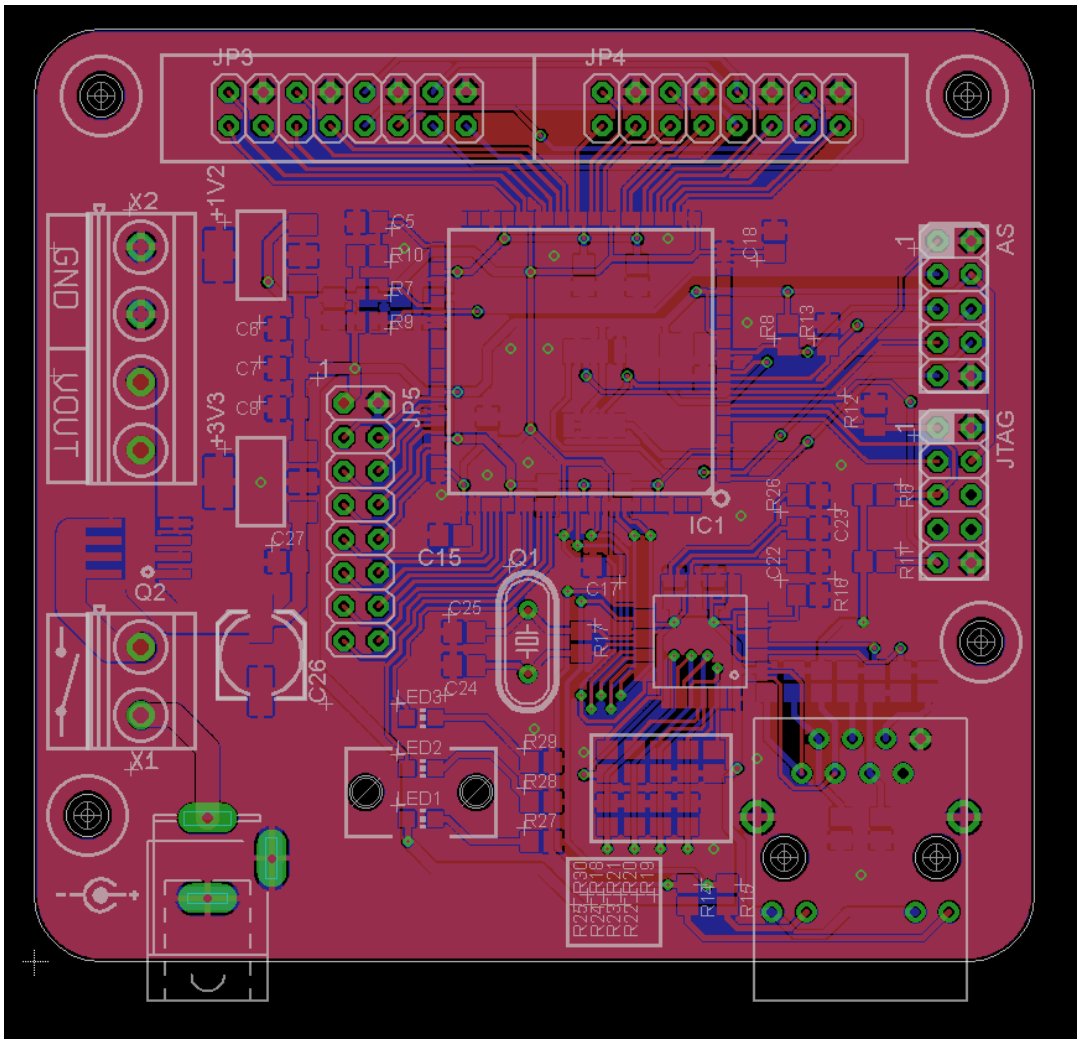
Tässä vaiheessa prototyyppiä alkoi lopullisen ohjelmakoodin kirjoittaminen ja sen testaaminen. Ohjelma oli hyvä saada toimivaksi tässä vaiheessa, sillä jos ohjelma olisi toimiakseen vaatinut ylimääräistä elektroniikkaa, oli se helppo lisätä ja testata tässä vaiheessa. Samalla pystyi myös helposti arvioimaan lopullisen virrankulutuksen jonka ansiosta elektroniikkaan kuuluvat jänniteregulaattorit oli helppo mitoittaa.

5.2 Piirilevy

Lopullisen ohjainlevyn suunnitteluun käytin pohjana valmistajien julkaisemia kytkentäkaavioita komponenttien koealustoista. Tämä helpotti huomattavasti suunnittelua, sillä kaikkien tärkeiden vastuksien ja kondensaattorien arvot olivat jo valmiiksi mitoitettu. Valmiiden kaavioiden käyttö myös varmisti myös lopullisen tuotteen luotettavan toiminnan ja vähensi mahdollisten virheiden määrää.

Piirilevyn suunnitteluun käytin yrityksen CadSoft kehittämää EAGLE ohjelmistoa. Ohjelmisto on pääosin maksullinen, mutta siitä on saatavissa myös ilmainen versio, missä tosin on rajoitteita levyn koolle ja kerroksien määrälle. Tämän työn vaatiman piirilevyn suunnitteluun riitti ilmainen versio, sillä ulkomitat jäivät alle 100x80 millimetriin ja kerroksia tarvittiin vain kaksi kappaletta. Levyn suunnitelma on esitetty kuviossa 10.

Kuviossa purppura kerros on levyn yläpuolen kupari ja sininen alapuolen kupari. vihreät osat ovat komponenttien vaatimat reiät, sekä reiät jotka kuljettavat signaalin levyn yläpuolelta alapuolelle. Valkoiset merkinnät ovat komponenttien paikkoja, sekä levyyn printattavat kuviot ja komponenttien nimitykset.

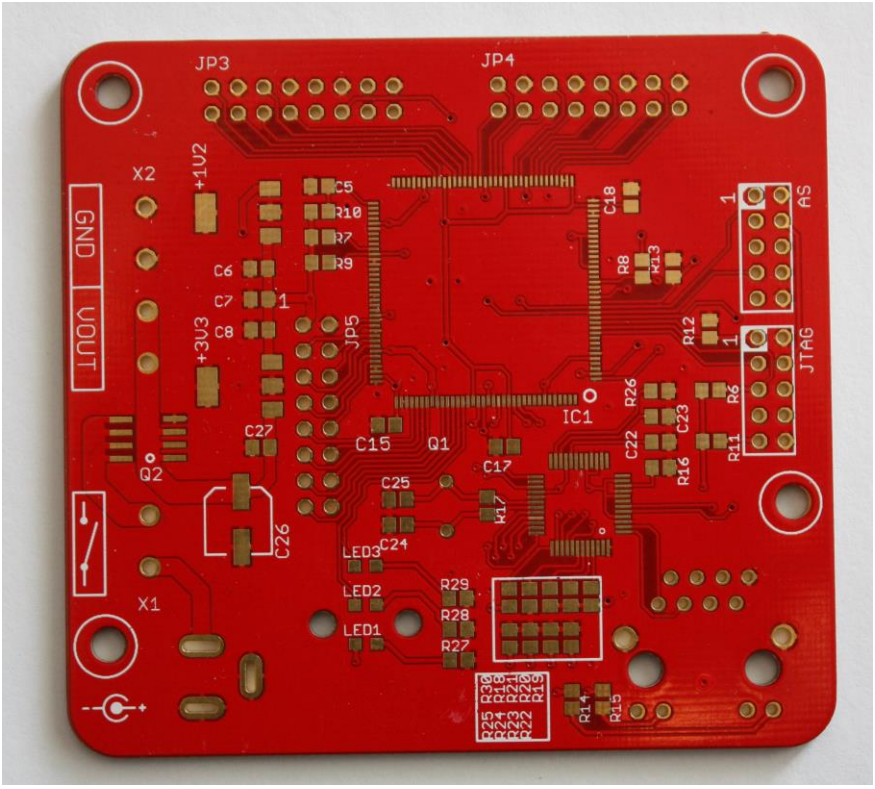


KUVIO 10. Piirilevyn suunnitelma ylhäältäpäin.

Kuviossa näkyvät JP3 ja JP4 liittimet ovat matriisia varten, X2 liitin on matriisin jännitteen syöttöä varten sekä X1 on liitin virtakytkintä varten. JTAG ja AS on FPGA-piirin ohjelmointia varten, AS liittimen avulla ohjelmoidaan FPGA-piirin ulkoinen ohjelmamuisti ja JTAG on testiliitin piiriä varten. Kuviosta erottuvat myös paikat jänniteregulaattoreille, paikat on merkitty tekstein +3V3 ja +1V2. Jänniteregulaattorit muuttavat sisään tulevan 5V jännitteen piireille tarvittavaksi 3,3V ja 1,2V suuruisiksi jännitteiksi. Kuviossa näkyvä Q2 p-kanavan MOSFET transistori toimii suojana jos virran napaisuus on väärinpäin. P-MOSFET toimii samoin tavoin kuin suojadiodi jos transistorin hila on kytketty maahan. Syy miksi valitsin transistorin diodin sijasta on transistorin pienempi

jännitehäviö, tällä saavutetaan parempi hyötysuhde sekä pienempi lämmöntuotto. Levyn piirikaavio löytyy liitteestä 1 sekä täydellinen osaluettelo liitteestä 2.

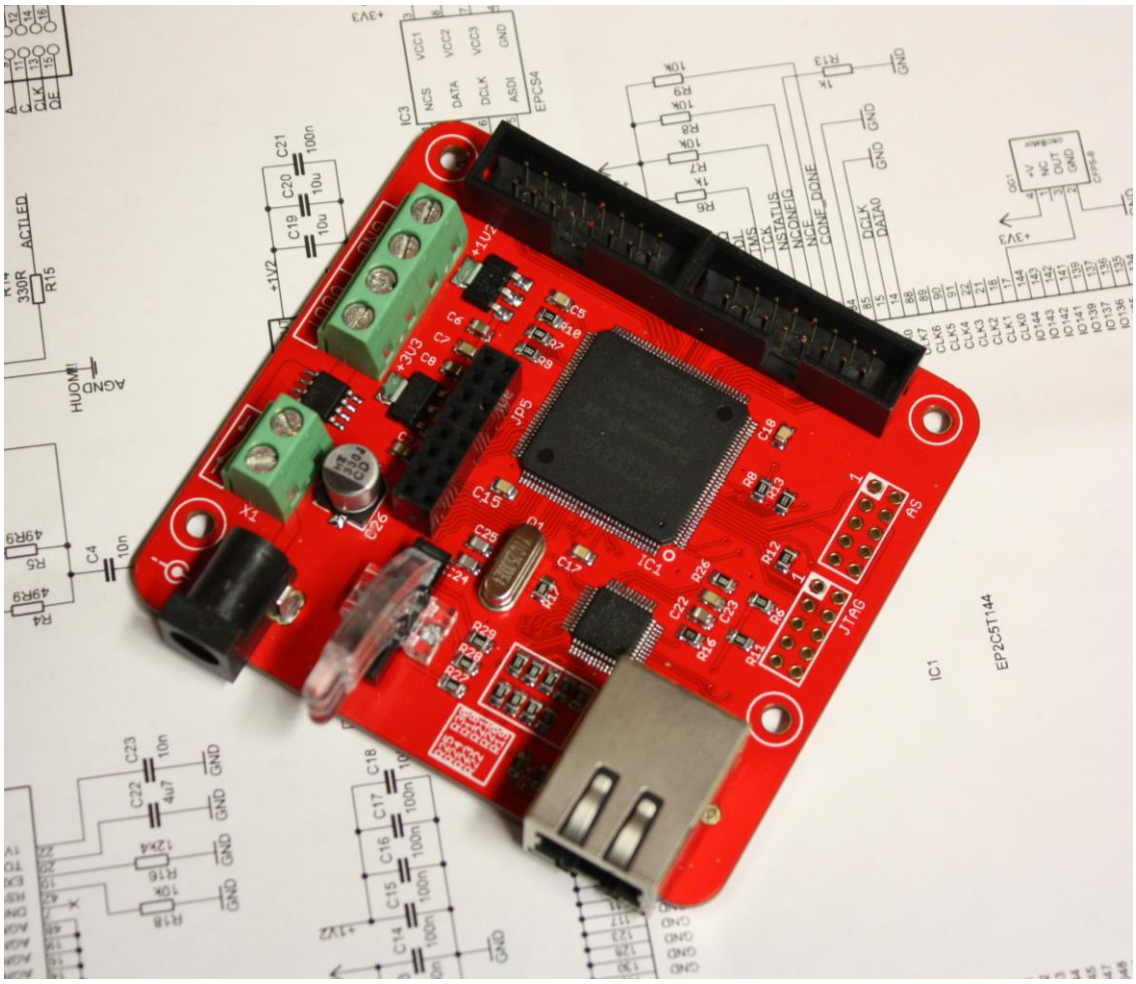
Saatuani valmiiksi levyn valmistusta varten vaaditut tiedostot, teetin levyn Kiinassa sijaitsevassa Elecrow yhtiössä. Elecrow valmistaa pieniä piirilevyeriä asiakkaan tiedostojen perusteella. Elecrow on yksi halvimmista paikoista minkä löysin, ainoana ongelmana oli tuotannon hitaus, sillä tilauksen valmistuminen vie yleensä noin 1-2 viikkoa. Tuotantoajan lisäksi odottamista lisää vielä toimitusaika, joka vaihtelee reilusta viikosta kuukauteen toimitustavasta riippuen. Tilaamani piirilevyn toimitus tilaushetkestä siihen, että sain levyt käsiini, vei noin kuukauden. Levylle tulevat komponentit tilasin Puolassa sijaitsevasta elektroniikka komponentteja myyvästä yrityksestä nimeltään Transfer Multisort Elektronik. TME oli oivallinen paikka hankkia komponentit, sillä toimitusaika varastossa oleville tuotteille on vain muutama päivä. Kuvassa 5 on esitetty teettämäni piirilevy ilman komponentteja.



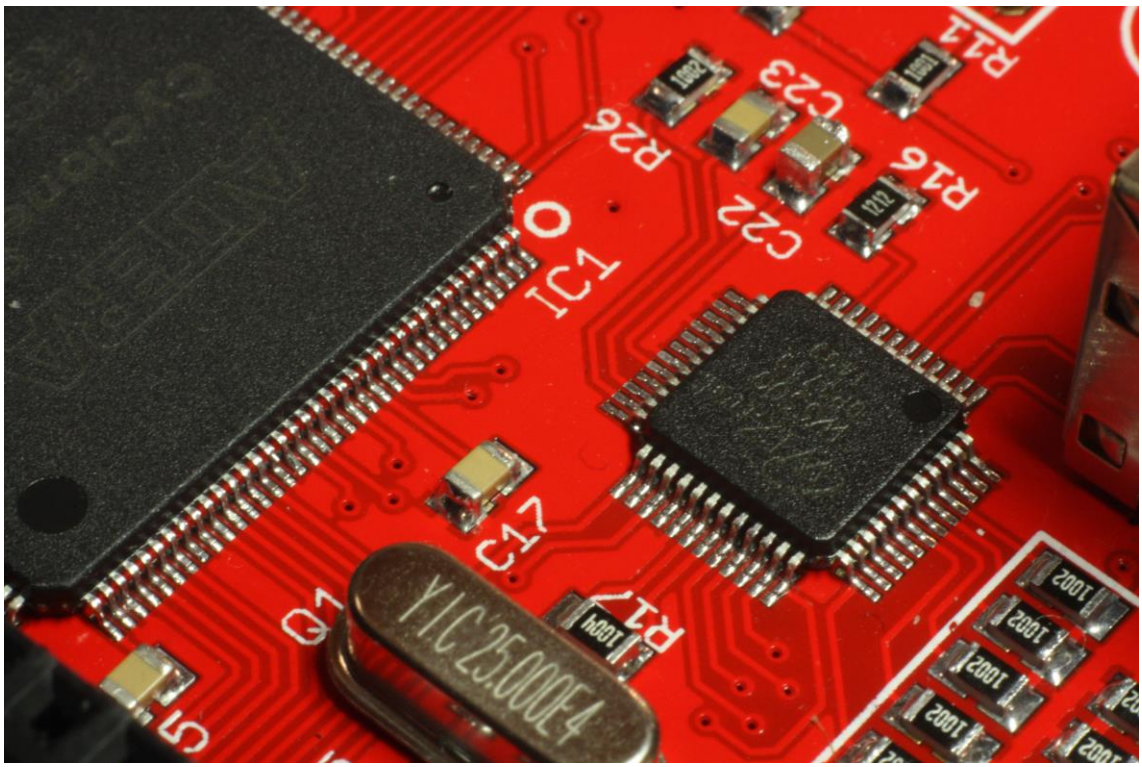
KUVA 5. Piirilevy ilman komponentteja

Saatuani levyn oli aika aloittaa komponenttien juottaminen paikoilleen. Teollisuudessa tämänlaiset levyt juotettaisiin käyttäen ensin stensiiliä millä levitetään juotostahna komponenteille. Juotostahnan jälkeen mekaaninen ”pick and place” kone latoo komponentit

levylle ja sen jälkeen levy kuljetetaan uuniin missä juotostahna sulaa ja juottaa komponentit paikoilleen. Kotioloissa on myös mahdollista kasata levy samankaltaisella tyylillä. Päädyin kuitenkin juottamaan osat levyyn käsin, sillä tarvitsin vain yhden kappaleen ja sabluunan teettäminen sekä sopivan uunin hankkiminen olisi nostanut kuluja huomattavasti. Kuvassa 6 on esitetty piirilevy komponenttien kanssa, sekä kuvassa 7 on lähikuva juotoksista.



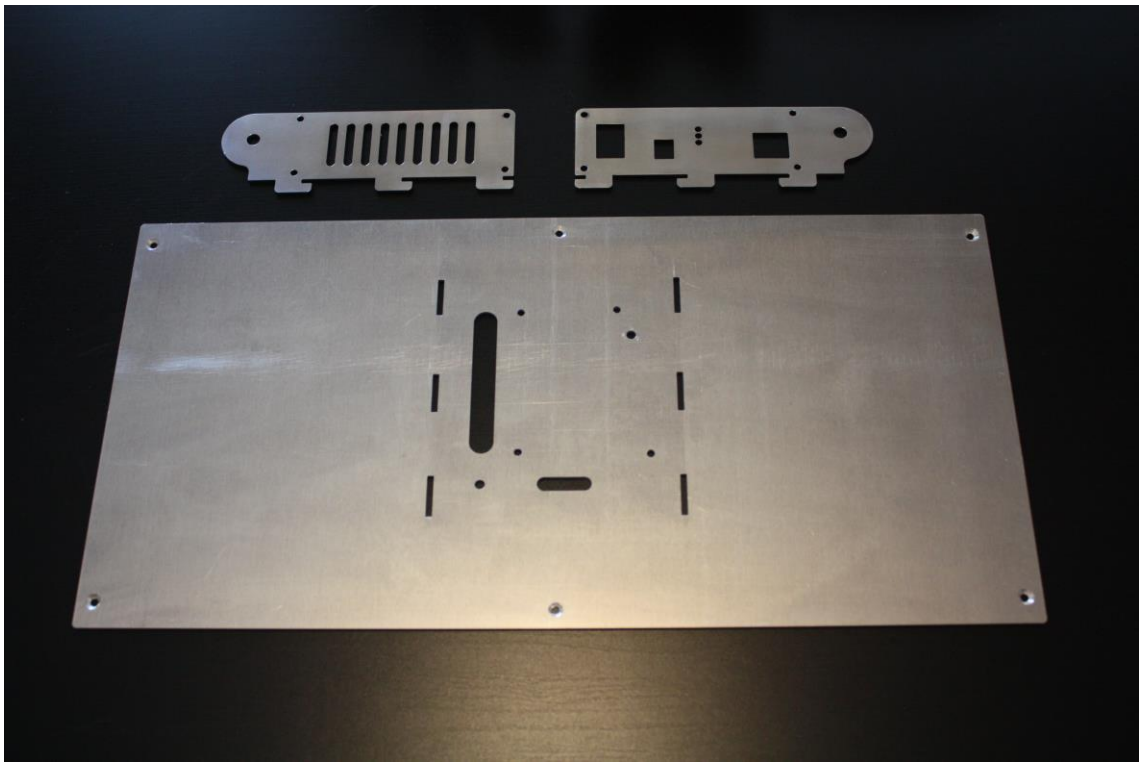
KUVA 6. Valmis piirilevy.



KUVA 7. Lähikuvaa juotoksista.

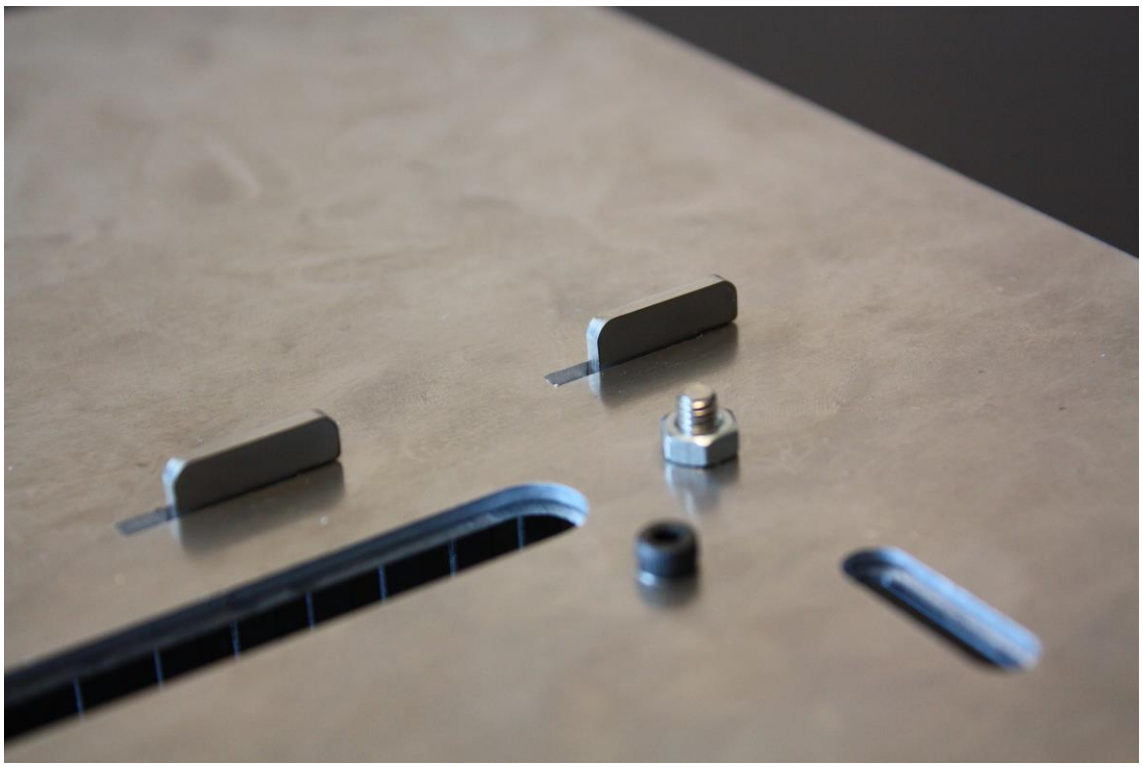
5.3 Kotelon valmistus

Kotelon suunnittelussa pyrin siihen, että se olisi valmistettavissa mahdollisimman vähillä työvaiheilla ja helppo kasata. Helpottaakseni työtäni pyrin käyttämään valmista koteloa ohjauspiiriä varten. Ohjauspiirin kotelo oli ainoa valmiina hankittu osa ja muut osat ovat valmistettu alumiinista ja ruostumattomasta teräksestä piirustuksieni perusteella. Ohjauspiirin kotelon valinnassa päädyin alumiinista valmistettuun 4-osaiseen koteloon, johon kuuluu päätylevyt ja kaksiosainen profiilista tehty runko. Profiilista tehdyn kotelon käyttöön päädyin siksi, koska se oli helppo liittää muihin kotelon osiin tekemällä siihen toisenlaiset päätylevyt. Kotelo oli yleismallinen ja näin ollen ihan normaalia hyllytavaraa, joten se oli myös nopeasti saatavilla. Päätylevyt ovat leikattu 2 mm vahvuisesta ruostumattomasta teräksestä laserleikkurilla ja niihin on samalla tehty paikka jalalle, reiät liittimille ja merkkivaloille, sekä lisäksi muutama aukko parempaa jäähdytystä varten. Päätyihin on lisätty myös hakaset takalevyyn kiinnitystä varten. Takalevy on leikattu 2 mm alumiinista ja siihen on tehty reiät kiinnitystä ja johtoja varten, sekä myös lovet, mihin päätyjen hakaset sopivat. Kuvassa 8 on esitetty laserleikatut päädyt ja takalevy.



KUVA 8. Laserleikatut osat koteloon

Kuten jo aiemmin totesin, on päätylevyjen kiinnitys takalevyyn toteutettu kuvassa 8 nähtävillä hakasilla. Hakasien lisäksi vaihtoehtoina olisi ollut myös tehdä kaikki osat samasta materiaalista ja hitsata ne yhteen. Hitsaus olisi tehnyt kotelosta tukevamman, mutta koska matriisi ja takalevy eivät paina paljoa, on hakaset riittävän tukeva kiinnitysmenetelmä. Hakasien käyttöä puolsi myös se että hitsaamisen tuoma lämpö ja sauman kutistuminen hitsauksen jälkeen aiheuttaa lieviä muodon muutoksia levyissä. Kyseiset muodonmuutokset vaarantaisivat osien yhteensopivuuden ja aiheuttaisivat lisää työtä. Hakaset ottavat vastaan kaiken painon, mutta eivät yksinään riitä pitämään koteloa kasassa. Hakasien irtoaminen on estetty yhdellä M4 pultilla joka pitää ohjauspiirin profiilikotelon ja takalevyn yhdessä, sekä lisäksi on myös kaksi M3 pulttia jotka kiinnittävät piirilevyn koteloon. Kuvassa 9 on esitetty hakasien kiinnitys ja pultit.



KUVA 9. Hakaset ja varmistuspultti

Kotelon päätylevyt ja takalevy on valmistettu laserleikkurilla koska osien yhteensopivuus vaati hyvää tarkkuutta ja samalla se vähensi viimeistelyn tarvetta. Kotelon jalka ei vaatinut yhtä hyvää tarkkuutta kuin muut osat, joten sen osat päätin leikata vesileikkurilla koska siihen oli mahdollisuus. Kuvassa 10 on nähtävissä osien leikkaamista Alikon valmistamalla vesileikkurilla.

KUVA 10. Osien vesileikkausta

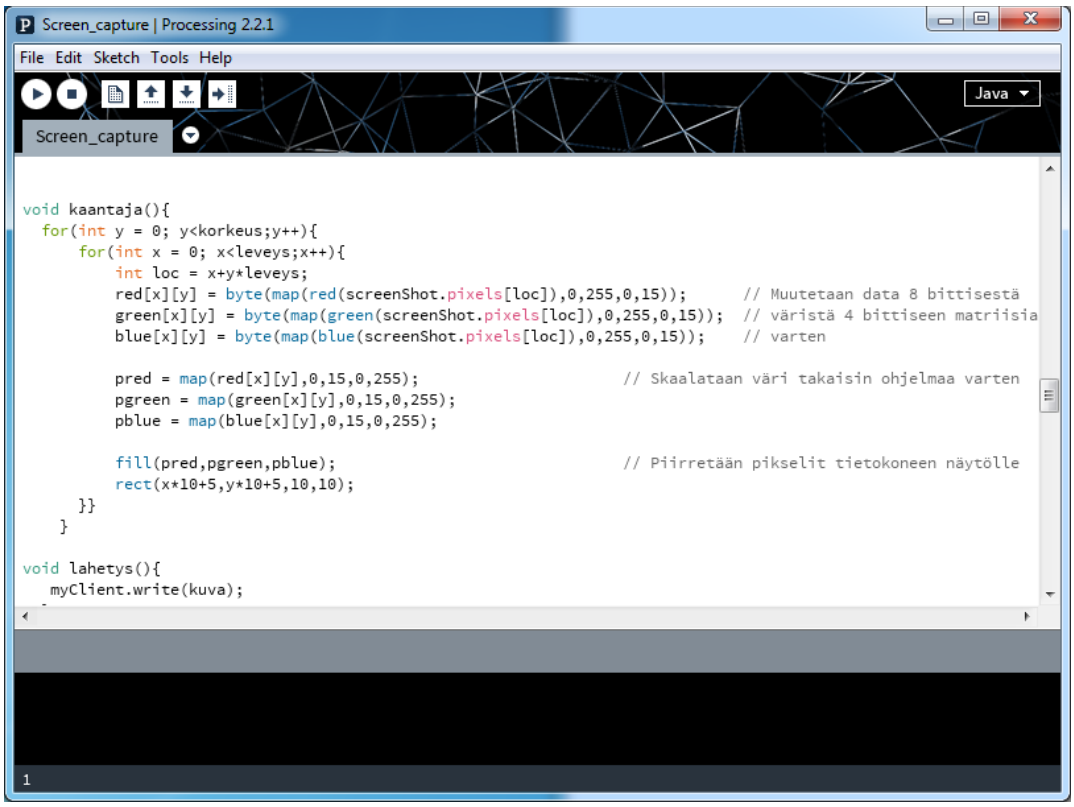
Valmis Kotelo on muotoilultaan varsin yksinkertainen ja kevytrakenteinen. Osat sopivat paikoilleen juuri niin kuin oli tarkoitettukin ja se oli erittäin helppo kasata. Valmis kotelo on esitetty kuvissa 11 ja 12.

KUVA 11. Näyttö edestä

KUVA 12. Näyttö takaa

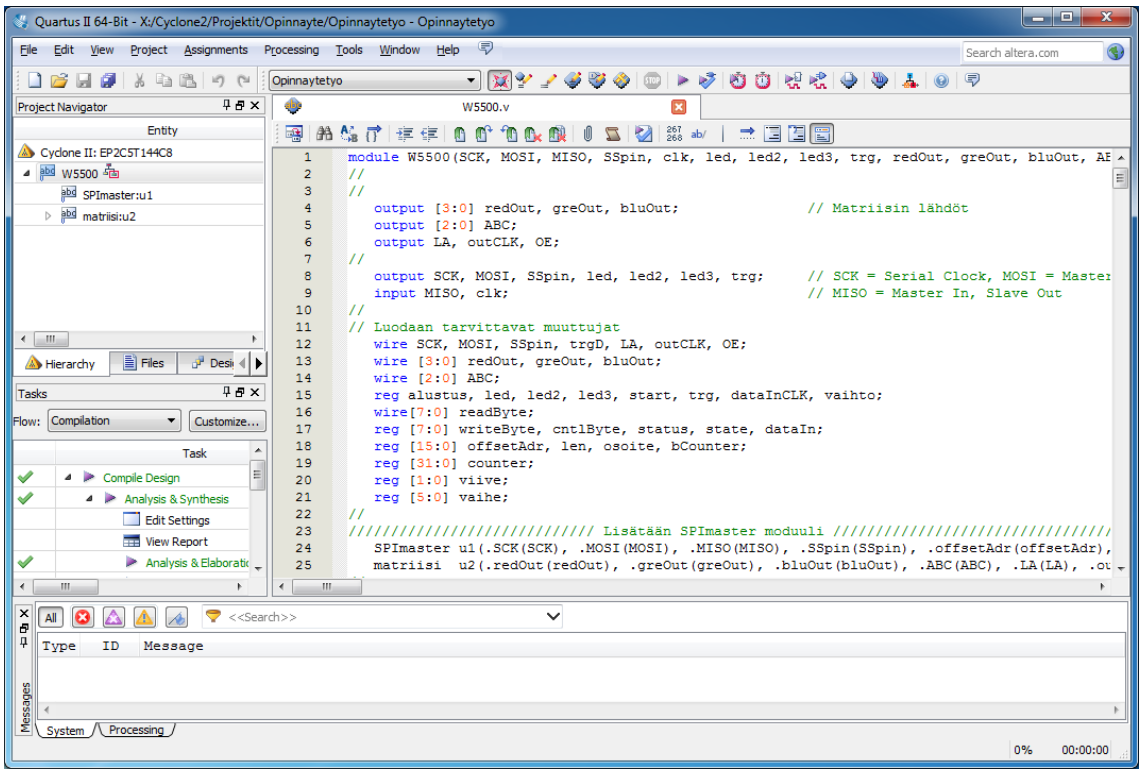
6 OHJELMOINTIYMPÄRISTÖT

Tietokoneen ohjelman tekemiseen käytin ilmaista Processing-nimistä ohjelmointiympäristöä. Processing on ohjelmointikieli, ohjelmointiympäristö ja internet yhteisö joka sai alkunsa vuonna 2001. Processing ympäristö on ns. ”Open Source” ohjelmisto, eli sen lähdekoodi on vapaasti saatavilla ja muokattavissa. Processing ohjelmointikieli pohjautuu C++ ja java ohjelmointikieliin ja siihen on saatavilla yli 100 eri tarkoitukseen tehtyä aliohjelmakirjastoa jotka on hyvin dokumentoituja. Ohjelmointiympäristön lisäksi Processing tarjoaa myös erittäin laajan käyttäjäfoorumin sekä kattavan wiki-aineiston esimerkkikoodeista ja aliohjelmien käytöstä. Processing ympäristö tarjoaa myös kääntäjän, joka kääntää käyttäjän kirjoittaman koodin Java- sovellukseksi. Kääntäjiä on myös saatavilla muille alustoille. Processing ohjelmointiympäristö on varsin yksinkertainen ja se koostuu tekstieditorista sekä muutamasta painikkeesta tärkeimpiä toimintoja varten. Ympäristössä voidaan suorittaa testiajo kirjoitetulle ohjelmalle tai se voidaan kääntäjän avulla tallentaa itsenäiseksi ohjelmakseen joka voidaan ajaa ilman Processing ympäristöä. (Processing.) Kuvassa 13 on esitetty ohjelmointi ympäristö.



KUVA 13. Processing ohjelmointiympäristö

FPGA-piirin ohjelman tekemiseen ja piirin ohjelmointiin käytin piirin valmistajan tarjoamaa ilmaisversiota Quartus II ohjelmistosta. Suurin ero ilmaisversion ja maksullisen välillä on tuettujen piirien määrässä. Ilmaisversio ei tue Alteran isoimpia mallisarjoja ja siitä puuttuu osa työkaluista. Ilmaisversion ominaisuudet kuitenkin riitti työhöni paremmin kuin hyvin. Ohjelmisto on hyvin laaja ja se tukee useaa eri laitteistokuvauskieltä, kuten esimerkiksi käyttämääni Verilog kieltä. Quartus ohjelmisto tarjoaa käyttäjälle tekstieditorin lisäksi työkalut ohjelman simulointia ja piirin ohjelmointia varten. Ohjelmistosta löytyy myös hyvät työkalut ohjelman syntetisointia ja kääntämistä varten. Laitteistokuvauskielen syntetisoinnilla tarkoitetaan vaihetta, jossa laitteistokuvauskielelle kirjoitettu ohjelma muutetaan logiikka-elementtien kytkennöiksi. Syntetisointi on pakollinen vaihe, sillä FPGA-piirin sisäisen rakenteen takia se ei aja erillistä ohjelmaa vaan se tarvitsee konfigurointi ohjeet logiikka-elementtien järjestämistä varten. Syntetisoinnin jälkeen ohjelma vielä käännetään piirin hyväksymään datamuotoon. Ohjelmisto sisältää myös analysointityökaluja joiden avulla kirjoitettu ohjelma voidaan optimoida mikäli ajoitukset tai logiikka-elementtien määrä on merkittävä tekijä. Quartus ohjelmistoa voidaan käyttää myös ASIC-piirin (Application Specific Integrated Circuit) suunnitteluun, jolloin simulointi ja analysointi työkalut ovat erittäin isossa roolissa. (Altera.) Kuvassa 14 on esitetty Quartus II ohjelmiston ympäristö.



KUVA 14. Quartus II ohjelmisto

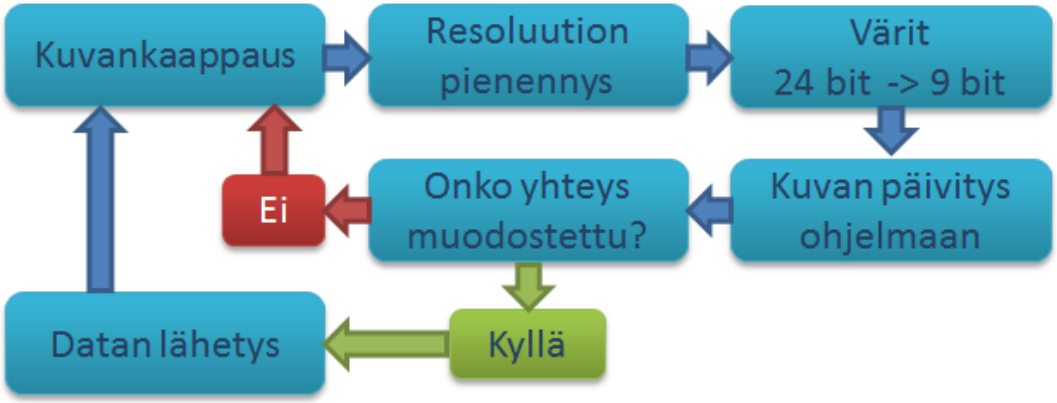
7 OHJELMOINTI

7.1 Tietokoneen ohjelma

Tietokoneen ohjelmakierron alussa tarkistetaan onko painikkeita painettu yhteyden muodostamista tai katkaisua varten. Seuraavaksi ohjelma tarkistaa, paljonko aikaa on kulunut siitä, kun kuva on päivitetty. Jos viive on ylittynyt, siirtyy ohjelma kuvankaappaus ja datan prosessointi aliohjelmiin. Näytönkaappaus aliohjelmassa ohjelma tallentaa kuvan joka sillä hetkellä on ennalta määrätyllä alueella tietokoneen näytöllä. Tallentamisen jälkeen kuvan resoluutio pienennetään näytölle sopivaksi. Aliohjelma tallentaa kuvan 24 bittiseksi dataksi, missä jokaista päävärin kirkkautta varten on 8 bittiä, eli yksi tavu dataa. LED-matriisi tarvitsee toimiakseen 12 bittisen värimaailman, joten jokaisen värin arvo skaalataan väliltä 0-255 välille 0-15. Kun jokaisen pikselin väri on skaalattu, tulostetaan kuva käyttöpaneeliin. Seuraavaksi kuva valmistellaan lähetystä varten, joten jokaisen pikselin data kasataan yhdeksi pitkäksi merkkijonoksi. Datan muokkaamisessa ongelmaksi tulee se, että yhtä pikseliä varten dataa on 12 bittiä ja normaali tavu on vain 8 bittiä. Ongelman voisi ratkaista tallentamalla jokainen pikseli omaksi kahden tavun mittaiseksi kokonaisluvuksi. Ongelmana tässä kuitenkin on se, että kuvan vaatima datamäärä olisi huomattavasti isompi kuin sen tarvitsisi olla ja se hidastaisi datan lähettämistä. Kyseiseen ongelmaan löysin ratkaisun tallentamalla datan bitti kerrallaan, jolloin tavujen vakiopituus ei ole ongelma. Kyseinen ratkaisu mahdollisti myös sen, että datamäärä pysyy mahdollisimman pienenä. Tämä ratkaisu toimi hyvin myös siksi, että samalla kun tavut puretaan, voidaan pikselien data kasata oikeassa järjestyksessä matriisia varten jolloin vältytään ylimääräiseltä datan muokkaamiselta matriisin puolella. Datan muokkaamisen ja tallentamisen jälkeen ohjelma tarkistaa onko yhteys matriisiin luotu, jos yhteys on olemassa, lähetetään data eteenpäin matriisille, muutoin aloitetaan ohjelmakierto alusta. Täydellinen ohjelmakoodi löytyy liitteestä 3.

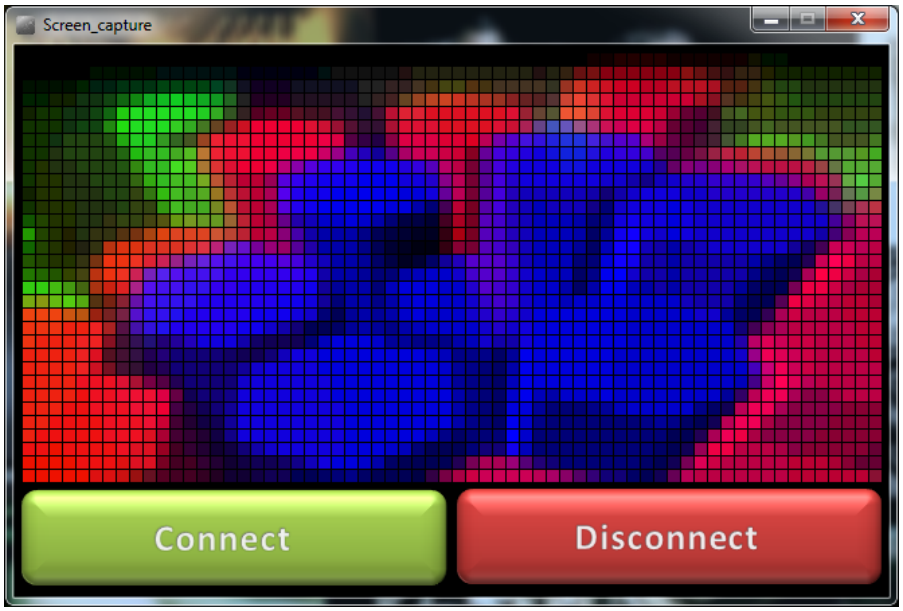
Ohjelman saatua käskyn muodostaa yhteys, yrittää se luoda yhteyden serveriin, jonka matriisi on luonut käytettyyn lähiverkkoon. IP osoite ja käytetyn portin numero on valmiiksi asetettu itse ohjelmakoodiin, koska tässä sovelluksessa käytössä on vain yksi näyttö. Tarvittaessa voitaisiin ohjelmaa muokata siten, että ennen yhdistämistä käyttäjän tulee syöttää ohjelmaan hallintapaneelin kautta haluttu IP osoite. Yhteyden katkaisua pyydettäessä ohjelma lähettää matriisiin ensin dataketjun, joka muuttaa kaikki pikselit mustiksi, eli sammuttaa kaikki ledit. Tämän jälkeen ohjelma odottaa 500 millisekuntia

jolla varmistetaan että kaikki data on siirretty. Viiveen jälkeen yhteys katkaistaan ja ohjelma jää jatkamaan normaalia kiertoa lähettämättä dataa matriisille. Kuviossa 11 on esitetty ohjelmakierron toimintaa kuvaava lohkokaavio.



KUVIO 11. Tietokoneen ohjelman lohkokaavio

Ohjelman käyttöpaneeli on varsin yksinkertainen ja pitää sisällään vain kaksi painiketta. Painikkeet ovat yhteyden muodostamista ja katkaisua varten. Paneelin painikkeet ovat BMP kuvatiedostoja jotka ohjelma lisää paneeliin. Ohjelma tunnistaa hiiren klikkauksen, mikäli se tehdään painikkeen sisällä ja toteuttaa näin halutun toiminnan. Käyttöpaneelissa on myös LED-matriisin pikseleitä edustava ruutu, joka toistaa matriisille lähetettyä kuvaa. Ruudulle kuva tulostetaan saman datan perusteella mitä itse LED-näytölle lähetetään ja tämän ansiosta on mahdollista tarkkailla näytettävää kuvaa tietokoneelta käsin. Kuvassa 15 on esitetty ohjelman käyttöpaneeli toiminnassa.

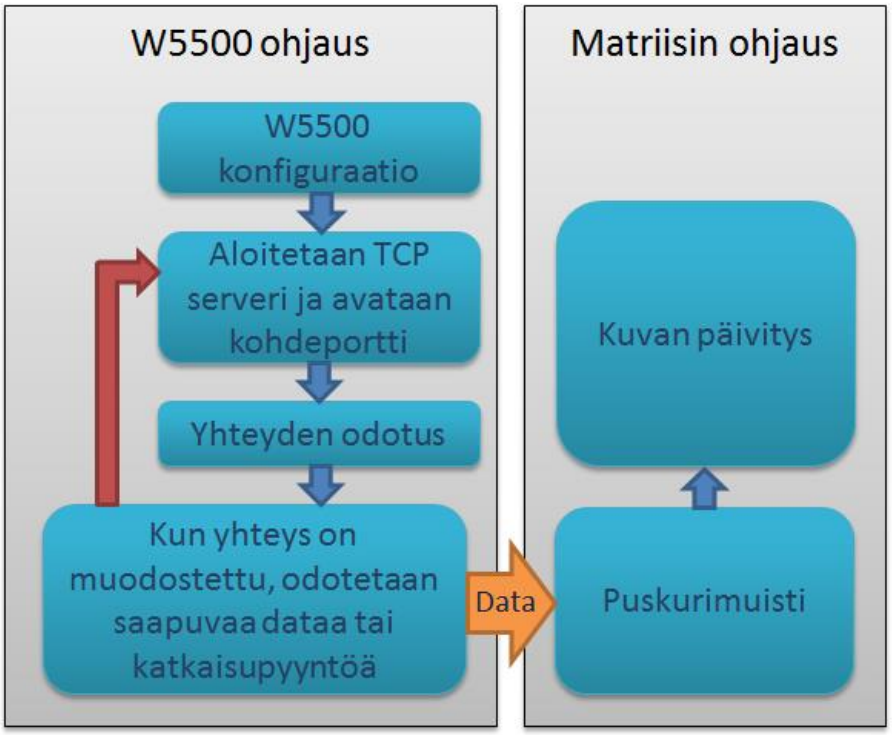


KUVA 15. Tietokoneen ohjelman käyttöpaneeli

7.2 FPGA-piirin ohjelma

FPGA-piirin ohjelma on kirjoitettu Verilog nimisellä laitteistokuvauskielellä, jota käytetään laitteiston tai logiikan toiminnan kuvaamiseen. Laitteistokuvauskielellä tehty logiikan ohjelma ei etene sekventaalisesti rivi kerrallaan kuten esimerkiksi mikroprosessorien ohjelmat. Tämä käyttäytyminen luo lisävaikeutta ohjelman tekemiseen ja ohjelmaa kirjoittaessa tarvitsee olla tarkkana että vain halutut operaatiot suoritetaan kerralla. Mikäli useampi operaatio samaa muuttujaa varten suoritetaan samaan aikaan, voi logiikkaketjujen välille tulla kilpailutilanne ja näin ollen lopputulos ei aina ole haluttu. Rinnakkainen toiminta tuo huomattavasti lisävaikeutta ohjelmointiin, mutta se tarkoittaa myös että yhden kellopulssin aikana voidaan suorittaa huomattavasti enemmän toimintoja.

FPGA-piirin ohjelma pitää sisällään kolme erillistä ns. moduulia. Moduulit toimivat täysin rinnakkain eivätkä näin ollen hidasta tai häiritse toistensa toimintaa. Ensimmäinen moduuli ohjaa W5500-piiriä ja hoitaa dataliikenteen, toinen moduuli ohjaa LED-matriisia sekä puskurimuistia ja kolmas moduuli hoitaa SPI-väylän liikenteen. W5500 moduuli toimii päällimmäisenä ja näin ollen ohjaa muiden moduuleitten toimintaa. FPGA-piirin ohjelman toiminta on esitetty kuviossa 12.



KUVIO 12. Ohjelmakierron karkea lohkokaavio

7.2.1 W5500 moduuli

Moduuleista monimutkaisin on W5500-piirin ohjaamiseen ja datan vastaanottamiseen käytetty moduuli. Tämä kyseinen moduuli konfiguroi lähiverkko-ohjaimen aina näytön käynnistyessä. Konfiguroinnin jälkeen moduuli luo lähiverkkoon TCP-serverin, jonka jälkeen se jää odottamaan asiakkaan liittymistä serverille. Tietokoneen ottaessa yhteyttä serveriin, moduuli jää odottamaan lähetettyä dataa. Moduulin saatua tieto W5500-piiriltä siitä, että sen puskurimuistissa on vastaanotetun kuvan datapaketti, aloittaa se datan lukemisen ja sen syöttämisen matriisin ohjaukselle. Datapaketin käsiteltyään moduuli jää odottamaan seuraavaa datapakettia tai asiakkaan yhteydenkatkaisua. Asiakkaan katkaistua lähiverkkoyhteys, moduuli sammuttaa W5500-piirin TCP-serverin ja alustaa sen. Alustuksen jälkeen asiakkaan odottaminen aloitetaan uudelleen. W5500-piiriä ohjaavan moduulin ohjelma on nähtävissä liitteessä 4.

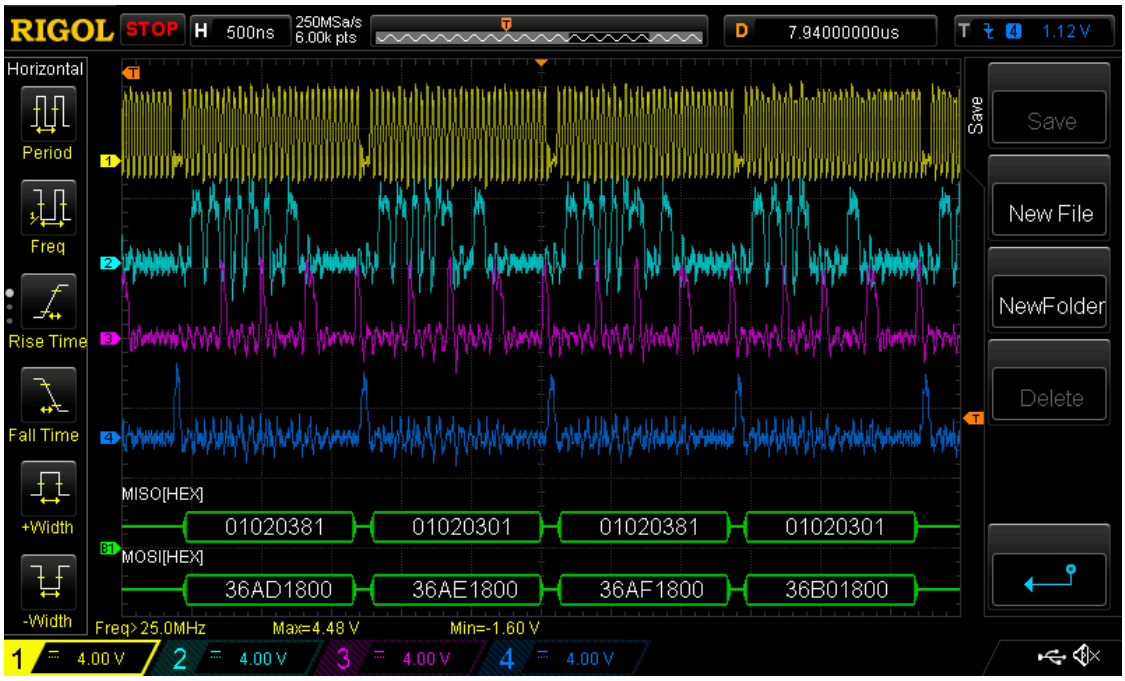
Moduulin lopussa on lista kaikista tarvittavista komennoista mitä W5500-piiri tarvitsee toimiakseen. Lista toimii siten, että ohjelman edetessä laskurin arvo kasvaa ja sen arvon perusteella listasta suoritetaan haluttu rivi joka muuttaa osoitemuuttujan, kontrollitavun ja lähetetyn datan halutuksi. Tämän jälkeen tiedot lähetetään SPI moduulin kautta piirille. Lista oli helpoin tapa toteuttaa piirin konfigurointi, sillä lähetetty data on ennalta määrätty ja voitiin tallentaa kiinteiksi muuttujiksi. Datan lukemista varten listaa ei voida käyttää juoksevan osoite numeroinnin takia. Komennot puskurissa olevan datan tarkistamista varten ja sen osoitteen lukemista varten ovat kuitenkin aina vakiot, joten ne voitiin listaan lisätä.

7.2.2 SPI moduuli

W5500-piiri on yhteydessä ohjaimeen SPI-väylän välityksellä. SPI-väylän käyttämiseen löytyy yleensä omat aliohjelmat mikroprosessoreille, mutta koska kyseessä oli ohjelmoitava logiikka, ei valmista koodia ollut saatavilla. SPI-väylä toimii ns. master-slave arkkitehtuurilla, missä yhdessä väylässä on yksi ohjaava laite eli master ja yksi tai useampi käytettävä laite eli slave. Ohjelman moduuli ohjaa SPI-väylän datalähtöjä joita on yhteensä neljä kappaletta, SCLK, MOSI, MISO ja SS. Väylän kellolinja on SCLK joka on yhteinen kaikille väylässä oleville laitteille ja sitä ohjaa väylän master. Datalinjat MOSI (Master Out, Slave In) ja MISO (Master In, Slave Out) ovat myös yhteisiä jokaiselle väylässä olevalle laitteelle. MOSI linjaa pitkin ohjaava laite lähettää dataa ohjatta-

ville laitteille ja MISO linjaa pitkin se vastaanottaa sitä. Jokaista ohjattavaa laitetta kohden on oma SS (Slave Select) lähtö ja sillä valitaan minkä käytettävän laitteen kanssa halutaan keskustella.

Ohjelman SPI moduuli vastaanottaa W5500 moduulilta halutut komennot ja lähettää ne eteenpäin lähiverkkoa ohjaavalle W5500-piirille. Piirin SPI kehys koostuu neljästä tavusta, joista kaksi ensimmäistä ovat osoitetieto. Kolmas on kontrollivaihe joka määrittää mitä osaa piiristä halutaan käyttää, esimerkiksi halutaanko muokata asetuksia, lukea dataa muistista tai kirjoittaa sitä. Viimeinen tavu voi olla joko muistiin kirjoitettava data tai halutut piirin asetukset, jolloin se on piirille lähetettävää dataa. Viimeinen tavu voi olla myös piiriltä luettava data, jolloin ohjaava piiri vastaanottaa tavun W5500-piiriltä. Kuviossa 13 on esitetty SPI väylän datalinjat ja niiden kautta kulkevia datapaketteja.

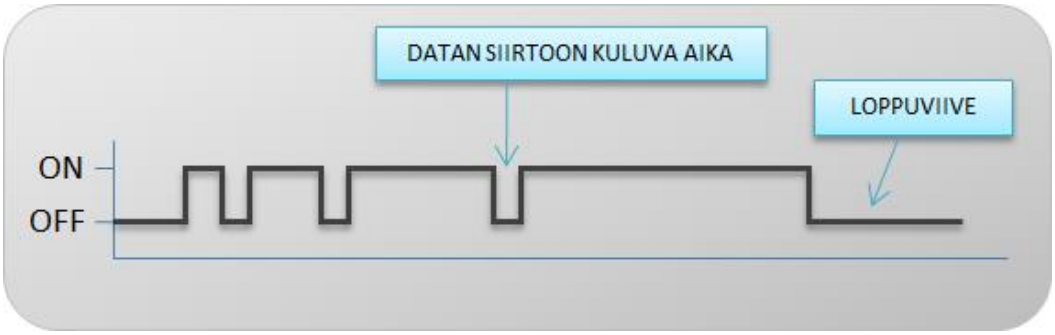


KUVIO 13. SPI-väylän data

Kuviossa oskilloskoopin kanava 1 on SPI-väylän kellolinja SCLK, kanava 2 on MOSI, kanava 3 on MISO ja kanava 4 on SS. Alimpana on nähtävissä linjoissa kulkeva data heksadesimaaleina. Kuviossa piiri vastaanottaa kuvaa ja MOSI linjassa menevän datan kahdesta ensimmäisestä tavusta, eli neljä ensimmäistä merkistä on nähtävissä kuinka piiri käy lävitse muistipaikkojen osoitteita. Kuvion vasemmassa alalaidassa on myös nähtävissä kellopulssin taajuus, joka on tässä kokoonpanossa 25 MHz. SPI moduulin ohjelma on liitteessä 5.

7.2.3 Matriisin moduuli

LED-matriisia ohjaava moduuli koostuu matriisin ohjaimesta ja puskurimuistista. Datan käsittelyn ja FPGA-piirin RAM-muistipaikkojen koon takia oli helpompi toteuttaa puskurimuisti usealla eri datamatriisilla. Datamatriiseista jokainen pitää sisällään 1024 neljä bittistä muuttujaa, jolloin yhden datamatriisin kooksi tulee 4096 bittiä. Yksi datamatriisi on samankokoinen kuin FPGA-piirin muistiblokki, mikä mahdollistaa muistin optimaalisen käytön. Jokaista LED-ohjainta varten on oma datamatriisi, joka pitää sisällään datan kahta eri kuvaa varten. Datamatriisi sisältää siis kuvan, jota kyseisellä hetkellä ajetaan näytölle, sekä puskurimuistissa olevan kuvan johon kirjoitetaan lähiverkosta vastaanotettua dataa. Uuden kuvan saavuttua puskurimuistiin, alkaa moduuli ajaa sitä näytölle, jolloin edellisen kuvan muistipaikat siirtyvät puskurimuistiksi. Värien kirkkautta varten jokaista lediä kohden on yhteensä 12 bittiä dataa, eli neljä bittiä kutakin pääväriä kohden. Eri kirkkauksien aikaansaamiseksi ledejä pidetään päällä bittien määräämä osuus ennalta määrätystä ajasta. Ledeille ajetaan ensin vähiten merkitsevien (LSB, Least Significant Bit) bittien data, joka määrää onko LED päällä ensimmäisen ajanjakson, sama toistetaan kaikkien bittien kohdalla. Ensimmäisen bitin määräämä aika on 1/15 lopullisesta ajasta, seuraava bitti on 2/15 osa lopullisesta ajasta, kolmas bitti on 4/15 osa ja neljäs on 8/15 osa ajasta. Tällä tavoin kirkkautta voidaan säätää binäärilukujen tavoin, jolloin 4 bitin avulla saadaan mahdolliseksi 16 eri kirkkautta täysin mustasta täyteen kirkkauteen. Bittien määräämien aikajakson jälkeen on vielä lyhyt viive, jolloin kaikki ledit ovat sammuksissa. Tätä viivettä muuttamalla voidaan määrätä LED-matriisin lopullinen kirkkaus. Ledien ohjaukseen käytetty tapa on esitetty kuviossa 14, jossa y-akseli esittää onko LED päällä vai ei ja x-akseli kuvaa aikaa.



KUVIO 14. Kirkkauden hallinta.

8 POHDINTA

Opinnäytetyö oli hyvin opettavainen ja tavoitteet täyttyivät. Työtä tehdessä ohjelmoitava logiikka tuli tutuksi ja samalla tuli opittua perusteet itselleni uudesta laitteistokuvauskielestä. Työtä tehdessä tuli opittua myös lähiverkon toiminnasta, mikä oli yhtenä tavoitteena. Uusien asioiden lisäksi kertyi lisää tietoa ja taitoa elektroniikkasuunnittelusta sekä komponenttien valinnasta. Kotelon suunnittelun ja toteutuksen osalta ei mitään uutta sen sijaan tarttunut matkaan, koska kokemusta kyseisestä aihepiiristä löytyy jo valmiiksi.

Teknisesti LED-matriisinäyttö on täysin toimiva ja juuri sellainen kuin oli tavoitteena. Elektroniikka toimii kuten pitääkin ja myös kotelon suunnittelu onnistui. Kotelo oli helppo kasata ja varsin kevyt, kuten alun perin oli tarkoituksenakin. Logiikan ohjelma on toimiva ja ajaa asiansa. Lopulliseksi näytönpäivitystaajuudeksi tuli 625 Hz, mikä on enemmän kuin riittävästi tasaisen kuvan aikaansaamiseksi. Tietokoneen ohjelma ei aivan täytä tarkoitustaan. Ohjelmassa toimii kuvankaappaus ja datan lähetys. Videokuvaa toistettaessa tietokoneen ohjelma ei pysy perässä ja tämän takia kuvataajuus ei ole riittävän korkea tasaisen videokuvan aikaansaamiseksi.

Kehitysideoina todettakoon, että tietokoneen ohjelma tulisi korjata ja optimoida nopeampaan kuvan lähetykseen. Vaikka logiikan ohjelma toimiikin, olisi hyvä silti parannella ja hienosäätää sitä. Yhtenä kehitysmahdollisuutena logiikan ohjelmalle voisi olla esimerkiksi tarkastusbittien lisääminen kuvan dataan, jotta mahdolliset virheet datassa eivät haittaisi kuvaa. Elektroniikan kehityksenä voisi lisätä jännitesäätimen virransyöttöön, jolloin näyttö ei olisi niin tarkka jännitteen suuruudelle. Tämän hetkinen jännitealue näytölle on välillä 4.2 – 5.5 volttia. Lisäämällä jännitesäädin virransyöttöön saataisiin alue välille 6 – 12 volttia, jolle välille useimmat muuntajat sijoittuvat. Ylimääräinen säädin tosin lisäisi lämmöntuottoa ja huonontaisi hyötysuhdetta.

LÄHTEET

Adafruit Industries. Medium 16x32 RGB LED matrix panel. Luettu 29.3.2015. www.adafruit.com/products/420

Altera. 2015. Cyclone Series. Luettu 26.3.2015. www.altera.com/products/fpga/cyclone-series.html

DigiKey. Pruducts. Luettu 15.4.2015. www.digikey.fi/product-search/en

Wolf, W. 2004. FBGA-Based System Design. 1. painos.

Xilinx. 2015. XC4000 Datasheet. PDF-tiedosto. Luettu 28.3.2015. www.xilinx.com/support/documentation/data_sheets/4000.pdf

StarChip Technology Inc. 2014. SCT2042 Datasheet. PDF-tiedosto. Luettu 27.3.2015. www.starchips.com.tw/pdf/datasheet/SCT2024V01_03.pdf

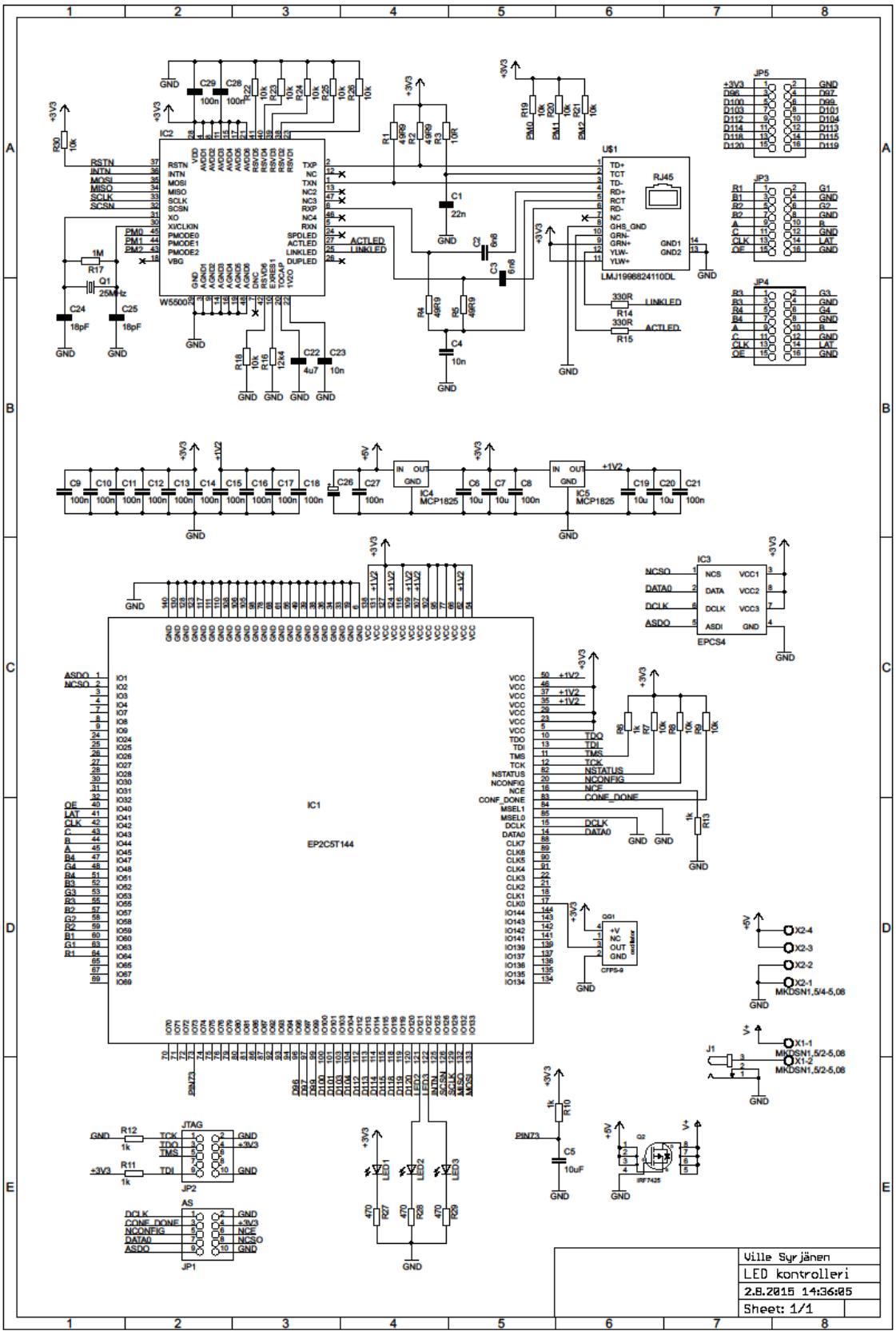
WIZnet. 2013. W5500 Datasheet. PDF-tiedosto. Luettu 29.3.2015. www.wizwiki.net/wiki/lib/exe/fetch.php?media=products:w5500:w5500_ds_v106e_141230.pdf

Processing. Overview. Luettu 22.5.2015. www.processing.org/overview/

Altera. 2015. Quartus II Software. Luettu 26.3.2015. www.altera.com/products/design-software/fpga-design/quartus-ii/overview.html

LIITTEET

Liite 1. Kytkentäkaavio



Liite 2. Piirilevyn osaluettelo

Tunnus	Nimike	Arvo/Malli
C1	Kondensaattori	22 nF
C2, C3	Kondensaattori	6,8 nF
C4	Kondensaattori	10 nF
C5-C7	Kondensaattori	10 µF
C8-C18	Kondensaattori	100 nF
C19, C20	Kondensaattori	10 µF
C21	Kondensaattori	100 nF
C22	Kondensaattori	4,7 µF
C23	Kondensaattori	10 nF
C24, C25	Kondensaattori	18 pF
C26	Kondensaattori	330 µF
C27-C29	Kondensaattori	100 nF
R1, R2	Vastus	49,9 Ω
R3	Vastus	10 Ω
R4, R5	Vastus	49,9 Ω
R6	Vastus	1 kΩ
R7-R9	Vastus	10 kΩ
R10-R13	Vastus	1 kΩ
R14, R15	Vastus	330 Ω
R16	Vastus	12,4 kΩ
R17	Vastus	1 MΩ
R18-R26	Vastus	10 kΩ
R27-R29	Vastus	470 Ω
R30	Vastus	10 kΩ
JP1-JP2	Piikkirima	2x5
JP3-JP5	Piikkirima	2x8
IC1	FPGA-piiri	EP2C5T144
IC2	Ethernet-piiri	W5500
IC3	Ohjelmamuisti	EPCS4
IC4	Regulaattori 3,3V	MCP1826S-3302
IC5	Regulaattori 1,2V	MCP1825S-1202
LED1 -LED3	Valodiodi	KP-2012SRC
Q1	Oskillaattori	25 MHz
Q2	P-Mosfet	IRF7425
Q3	Oskillaattori	50 MHz
J1	Virtaliitin	5,5/2,1
X1	Liitin	2-5,08
X2	Liitin	4-5,08
U$1	Liitin	RJ-45

Liite 3. Ohjelmakoodi tietokoneelle 1(3)

```
import java.awt.image.BufferedImage;
import java.awt.*;
import processing.net.*;
Client myClient;
static int leveys = 64;            // Kiineteä muuttujat matriisin leveydelle ja korkeudelle
static int korkeus = 32;

int koko = 3072;

PImage screenShot;

long aika = 1000;
int Osoite = 0;
int viive = 0;
boolean yhdistetty = false;
byte[][] red = new byte[leveys][korkeus];
byte[][] green = new byte[leveys][korkeus];
byte[][] blue = new byte[leveys][korkeus];
byte[] kuva = new byte[koko];
byte[] test = new byte[koko];
float pred = 0;
float pgreen = 0;
float pblue = 0;

PImage yhdista;
PImage katkaise;

void setup() {
  yhdista = loadImage("Yhdista.bmp");
  katkaise = loadImage("Katkaise.bmp");
  size(650, 410);
  screenShot = getScreen();
  screenShot.resize(64,32);
  background(0,0,0);
  for(int i=0;i<koko;i++){
       test[i] = 0;
     }
  image(yhdista,5,330);
  image(katkaise,328,330);

}

void draw () { // Ohjelmakierto jossa kutsutaan tarvittavat aliohjelmat ja tarkistetaan viive

  screenShot = getScreen();
  screenShot.resize(leveys,korkeus);

  if(mousePressed){
    if(mouseX<322 && mouseY<405 && mouseX>5 && mouseY>330){    //Suoritetaan jos yhdistetään
      myClient = new Client(this, "192.168.1.177", 8888);
      yhdistetty = true;
      viive = millis()+500;
      while(viive>millis()){
      }}
    if(mouseX<645 && mouseY<405 && mouseX>328 && mouseY>330){  //Suoritetaan jos yhteys katkaistaan
      viive = millis()+700;
      while(viive>millis()){
      }
      myClient.write(test);    // Lähetettään matriisille kokonaan musta kuva
      viive = millis()+700;    // ledien sammuttamiseksi ennen yhteyden katkaisua
      while(viive>millis()){
      }
      myClient.stop();
      yhdistetty = false;
  }}
  if(millis()-aika >= 30){
```

```
      kaantaja();
        if(yhdistetty == true){
          myClient.write(kuva);
        }
      tallennus();
      aika = millis();
      }}

PImage getScreen() {                          // Kuvankaappaus
  GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
  GraphicsDevice[] gs = ge.getScreenDevices();
  DisplayMode mode = gs[0].getDisplayMode();
  Rectangle bounds = new Rectangle(300, 300, 600, 450);
  BufferedImage desktop = new BufferedImage(mode.getWidth(),
  mode.getHeight(), BufferedImage.TYPE_INT_RGB);
  try {
    desktop = new Robot(gs[0]).createScreenCapture(bounds);
  }
  catch(AWTException e) {
    System.err.println("Naytonkaappaus ei onnistunut.");
  }
  return (new PImage(desktop));
}

void kaantaja(){
  for(int y = 0; y<korkeus;y++){
      for(int x = 0; x<leveys;x++){
          int loc = x+y*leveys;
        // Muutetaan data 8 bittisestä väristä 4 bittiseen matriisia varten
          red[x][y] = byte(map(red(screenShot.pixels[loc]),0,255,0,15));
          green[x][y] = byte(map(green(screenShot.pixels[loc]),0,255,0,15));
          blue[x][y] = byte(map(blue(screenShot.pixels[loc]),0,255,0,15));
        // Skaalataan väri takaisin ohjelmaa varten
          pred = map(red[x][y],0,15,0,255);
          pgreen = map(green[x][y],0,15,0,255);
          pblue = map(blue[x][y],0,15,0,255);
       // Piirretään pikselit tietokoneen näytölle
          fill(pred,pgreen,pblue);
          rect(x*10+5,y*10+5,10,10);
      }}}

void tallennus(){             // datan muunnin ja tallennus funktio
  Osoite = 0;
   for(int y=0;y<8;y++){
     for(int x=0;x<64;x++){
        bitWrite(Osoite,0,bitRead(red[x][y],1));
        bitWrite(Osoite,1,bitRead(red[x][y+8],1));
        bitWrite(Osoite,2,bitRead(red[x][y+16],1));
        bitWrite(Osoite,3,bitRead(red[x][y+24],1));
        bitWrite(Osoite,4,bitRead(red[x][y],0));
        bitWrite(Osoite,5,bitRead(red[x][y+8],0));
        bitWrite(Osoite,6,bitRead(red[x][y+16],0));
        bitWrite(Osoite,7,bitRead(red[x][y+24],0));
        Osoite++;
     }}
   for(int y=0;y<8;y++){
     for(int x=0;x<64;x++){
        bitWrite(Osoite,0,bitRead(red[x][y],3));
        bitWrite(Osoite,1,bitRead(red[x][y+8],3));
        bitWrite(Osoite,2,bitRead(red[x][y+16],3));
        bitWrite(Osoite,3,bitRead(red[x][y+24],3));
        bitWrite(Osoite,4,bitRead(red[x][y],2));
        bitWrite(Osoite,5,bitRead(red[x][y+8],2));
        bitWrite(Osoite,6,bitRead(red[x][y+16],2));
        bitWrite(Osoite,7,bitRead(red[x][y+24],2));
        Osoite++;
```

```
  }}
 for(int y=0;y<8;y++){
   for(int x=0;x<64;x++){
      bitWrite(Osoite,0,bitRead(green[x][y],1));
      bitWrite(Osoite,1,bitRead(green[x][y+8],1));
      bitWrite(Osoite,2,bitRead(green[x][y+16],1));
      bitWrite(Osoite,3,bitRead(green[x][y+24],1));
      bitWrite(Osoite,4,bitRead(green[x][y],0));
      bitWrite(Osoite,5,bitRead(green[x][y+8],0));
      bitWrite(Osoite,6,bitRead(green[x][y+16],0));
      bitWrite(Osoite,7,bitRead(green[x][y+24],0));
      Osoite++;
   }}
 for(int y=0;y<8;y++){
   for(int x=0;x<64;x++){
      bitWrite(Osoite,0,bitRead(green[x][y],3));
      bitWrite(Osoite,1,bitRead(green[x][y+8],3));
      bitWrite(Osoite,2,bitRead(green[x][y+16],3));
      bitWrite(Osoite,3,bitRead(green[x][y+24],3));
      bitWrite(Osoite,4,bitRead(green[x][y],2));
      bitWrite(Osoite,5,bitRead(green[x][y+8],2));
      bitWrite(Osoite,6,bitRead(green[x][y+16],2));
      bitWrite(Osoite,7,bitRead(green[x][y+24],2));
      Osoite++;
   }}
 for(int y=0;y<8;y++){
   for(int x=0;x<64;x++){
      bitWrite(Osoite,0,bitRead(blue[x][y],1));
      bitWrite(Osoite,1,bitRead(blue[x][y+8],1));
      bitWrite(Osoite,2,bitRead(blue[x][y+16],1));
      bitWrite(Osoite,3,bitRead(blue[x][y+24],1));
      bitWrite(Osoite,4,bitRead(blue[x][y],0));
      bitWrite(Osoite,5,bitRead(blue[x][y+8],0));
      bitWrite(Osoite,6,bitRead(blue[x][y+16],0));
      bitWrite(Osoite,7,bitRead(blue[x][y+24],0));
      Osoite++;
   }}
 for(int y=0;y<8;y++){
   for(int x=0;x<64;x++){
      bitWrite(Osoite,0,bitRead(blue[x][y],3));
      bitWrite(Osoite,1,bitRead(blue[x][y+8],3));
      bitWrite(Osoite,2,bitRead(blue[x][y+16],3));
      bitWrite(Osoite,3,bitRead(blue[x][y+24],3));
      bitWrite(Osoite,4,bitRead(blue[x][y],2));
      bitWrite(Osoite,5,bitRead(blue[x][y+8],2));
      bitWrite(Osoite,6,bitRead(blue[x][y+16],2));
      bitWrite(Osoite,7,bitRead(blue[x][y+24],2));
      Osoite++;
   }}
  Osoite = 0;
}

int bitRead(byte tavu, int bitti){              // bitin luku funktio
   int x = tavu & (1 << bitti);
   return x == 0 ? 0 : 1;
   }

void bitWrite(int tavu, int bit, int arvo){    // bitin kirjoitus funktio
  switch (arvo) {
    case 0:
        kuva[tavu] &= ~(1<<bit);
        break;
    case 1:
        kuva[tavu] |= (1<<bit);
        break;
  }}
```

Liite 4. W5500 moduuli 1(7)

```
1   module W5500(SCK, MOSI, MISO, SSpin, clk, led, led2, led3, trg, redOut, greOut, bluOut,
    ABC, LA, outCLK, OE, tp1, tp2);
2   //
3   //
4      output [3:0] redOut, greOut, bluOut;   // Matriisin lähdöt
5      output [2:0] ABC;
6      output LA, outCLK, OE;
7   // SCK = Serial Clock, MOSI = Master Out, Slave In, SS = Slave Select
8      output SCK, MOSI, SSpin, led, led2, led3, trg, tp1, tp2;
9      input MISO, clk;                        // MISO = Master In, Slave Out
10  //
11  // Luodaan tarvittavat muuttujat
12     wire SCK, MOSI, SSpin, trgD, LA, outCLK, OE, tp1, tp2;
13     wire [3:0] redOut, greOut, bluOut;
14     wire [2:0] ABC;
15     reg alustus, led, led2, led3, start, trg, dataInCLK, vaihto, clk2;
16     wire[7:0] readByte;
17     reg [7:0] writeByte, cntlByte, status, state, dataIn;
18     reg [15:0] offsetAdr, len, osoite, bCounter;
19     reg [31:0] counter;
20     reg [1:0] viive;
21     reg [5:0] vaihe;
22  //
23  ////////////// Lisätään SPImaster moduuli ////////////////////////////////////////
24     SPImaster u1(.SCK(SCK), .MOSI(MOSI), .MISO(MISO), .SSpin(SSpin), .offsetAdr(offsetAdr
    ), .cntlByte(cntlByte), .writeByte(writeByte), .readByte(readByte), .clk(clk), .trg(trg),
     .trgD(trgD));
25     matriisi  u2(.redOut(redOut), .greOut(greOut), .bluOut(bluOut), .ABC(ABC), .LA(LA), .
    outCLK(outCLK), .OE(OE), .dataIn(dataIn), .dataInCLK(dataInCLK), .clk(clk), .vaihto(
    vaihto), .tp1(tp1), .tp2(tp2));
26  //
27  //
28     always @ (posedge clk2) begin
29        if(trgD) trg<=1'd0;  // Jos trgD on ylhäällä niin lasketaan trg alas
30     //
31     // Tästä alkaa ns. "State Machine"
32        case(vaihe)
33     /////////// Ensimmäisenä odotetaan 2 sekuntia että W5500 piiri ehtii startata ///
34           6'd0: begin
35                    if(counter < 32'd50000000) begin
36                       counter<=counter+1'b1;
37                       led<=1'b0;
38                       led2<=1'b0;
39                       led3<=1'b0;
40                    end
41                    else begin
42                       led<=1'b1;
43                       vaihe<=6'd1;
44                       state<=8'd0;
45                    end
46                 end
47     /////////// Nostetaan start ylös siksi että osoitteet vaihtuisi oikeiksi ////////
48           6'd1: begin start<=1'b1; vaihe<=6'd2; end
49     /////////// Lasketaan start alas, lähetettään tiedot SPI väylällä ///////////////
50           6'd2: begin
51     // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
52                    if(SSpin) begin
53                       start<=1'b0;
54                       trg<=1'b1;
55                       state<=state+1'b1;
56                       vaihe<=6'd3;
57                    end
58                 end
59     // Jos dataa on kirjoittamatta alustusta varten niin listään laskuria
60           6'd3: begin
61                    if(state<8'd47) begin
```

```
62                      vaihe<=6'd1;
63                   end
64                   else begin
65                      vaihe<=6'd4;
66                   end
67                end
68    ////////////////////////////////////////////////////////////////////////////////
69    /////////// TCP serveri ////////////////////////////////////////////////////////
70    ////////////////////////////////////////////////////////////////////////////////
71          6'd4: begin
72                   state<=8'd52; vaihe<= 6'd5;
73                   // Aloitetaan kannan 0 alustus TCP serveriksi
74                end
75          6'd5: begin
76                   start<=1'b1; vaihe<= 6'd6;
77                end
78          6'd6: begin
79    // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
80                   if(SSpin) begin
81                      start<=1'b0;
82                      trg<=1'b1;
83                      vaihe<= 6'd7;
84                      state<=state+1'b1;
85                   end
86                end
87    ////////////////////////////////////////////////////////////////////////////////
88          6'd7: begin
89                   if(state<8'd57 || state == 8'd58) begin
90                      vaihe<=6'd5;
91                   end
92                   else if(state == 8'd57) begin // TCP:n tarkistus
93                      if(SSpin) begin
94                         if(readByte == 8'h13) begin
95                         // Siirrytään eteenpäin jos kanta aukesi TCP tilaan
96                            vaihe<=6'd5;
97                         end
98                         else begin  // Jos kanta ei auennu oikein palataan alkuun
99                            vaihe<=6'd4;
100                        end
101                     end
102                  end
103                  else if(state == 8'd59) begin // Serverin tarkistus
104                     if(SSpin) begin
105   // Siirrytään eteenpäin jos serverin luonti onnistu
106                        if(readByte == 8'h14) begin
107                           vaihe<=6'd8;
108                        end
109   // Jos Serverin luonti ei onnistunut niin Aloitetaan koko TCP prosessi alusta
110                        else begin
111                           vaihe<=6'd4;
112                        end
113                     end
114                  end
115   // Tarkistetaan status uudelleen jos mikään muu komento ei pitänyt paikkaansa
116                  else begin  vaihe<=6'd8;  end
117               end
118   ////////////////////////////////////////////////////////////////////////////////
119         6'd8: begin
120                  led2<=1'b0;
121                  vaihe<=6'd9;
122               end
123   ////////////////////////////////////////////////////////////////////////////////
124   /////////// Kannan testausta asiakasta varten //////////////////////////////////
125   ////////////////////////////////////////////////////////////////////////////////
126         6'd9: begin
127                  state<=8'd58; vaihe<= 6'd10;  // Tarkistetaan kannan 0 tila
```

```
128                 end
129         ////////////////////////////////////////////////////////////////////////////////
130               6'd10:begin
131                         start<=1'b1; vaihe<= 6'd11;
132                 end
133         ////////////////////////////////////////////////////////////////////////////////
134               6'd11:begin
135         // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
136                         if(SSpin) begin
137                            start<=1'b0;
138                            trg<=1'b1;
139                            vaihe<= 6'd12;
140                         end
141                 end
142         ////////////////////////////////////////////////////////////////////////////////
143               6'd12: begin
144                         if(SSpin) begin
145                            if(readByte == 8'h17) begin
146                               // Jos asiakas on yhdistetty, jatketaan datan lukemiseen
147                               led2<=1'b1;
148                               state<=8'd60;
149                               vaihe<=6'd13;
150                            end
151                            else if(readByte == 8'h1C) begin
152         // Jos asiakas poistui ilman täydellistä yhteyden katkaisua suoritetaan se loppuun
153                               state<=8'd59;
154                               vaihe<=6'd10;
155                               led2<=1'b0;
156                               led3<=1'b0;
157                               len<=16'd0;
158                            end
159         // Jos kanta on suljettu, palataan TCP serverin luomiseen
160                            else if(readByte == 8'h00) begin
161                               vaihe<=6'd4;
162                               led2<=1'b0;
163                               led3<=1'b0;
164                            end
165         // jollei mikään näistä pidä paikkaansa, luetaan status uudestaan
166                            else begin
167                               vaihe<=6'd9;
168                            end
169                         end
170                 end
171         ////////////////////////////////////////////////////////////////////////////////
172         /////////// Tarkistetaan paljonko dataa on saatavilla //////////////////////////
173         ////////////////////////////////////////////////////////////////////////////////
174               6'd13:begin
175                         start<=1'b1; vaihe<= 6'd14;
176                 end
177         // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
178               6'd14:begin
179                         if(SSpin) begin
180                            start<=1'b0;
181                            trg<=1'b1;
182                            vaihe<= 6'd15;
183                            state<=state+1'b1;
184                         end
185                 end
186         ////////////////////////////////////////////////////////////////////////////////
187               6'd15: begin
188                         if(SSpin) begin
189         // Luetaan ensimmäinen tavu vastaanotetun datan määrästä
190                            if(state == 8'd61) begin
191                               len<=(readByte << 8);
192                               vaihe<=6'd13;
193                            end
```

```
194      // Luetaan toinen tavu vastaanotetun datan määrästä
195                          else if(state == 8'd62) begin
196                             len<=len+readByte;
197                             vaihe<=6'd16;
198                          end
199                       end
200                    end
201      ////////////////////////////////////////////////////////////////////////////////
202            6'd16:begin
203      // Palataan statuksen lukemiseen jos dataa ei ole tarpeeksi saatavilla
204                       if(len<12'd3072) begin
205                          vaihe<=6'd9;
206                       end
207      // Siirrytään eteenpäin jos dataa on tarpeeksi
208                       else begin
209                          led3<=1'b1;
210                          vaihe<=6'd17;
211                       end
212                    end
213      ////////////////////////////////////////////////////////////////////////////////
214      /////////// Luetaan saapuneen datan offset osoite //////////////////////////////
215      ////////////////////////////////////////////////////////////////////////////////
216            6'd17:begin
217                       start<=1'b1; vaihe<= 6'd18;
218                    end
219      ////////////////////////////////////////////////////////////////////////////////
220            6'd18:begin
221      // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
222                       if(SSpin) begin
223                          start<=1'b0;
224                          trg<=1'b1;
225                          vaihe<= 6'd19;
226                          state<=state+1'b1;
227                       end
228                    end
229      ////////////////////////////////////////////////////////////////////////////////
230            6'd19: begin
231                       if(SSpin) begin
232      // Luetaan ensimmäinen tavu osoitteesta
233                          if(state == 8'd63) begin
234                             osoite<=(readByte << 8);
235                             vaihe<=6'd17;
236                          end
237                          else begin     // Luetaan toinen tavu osoitteesta
238                             osoite<=osoite+readByte;
239                             vaihe<=6'd20;
240                          end
241                       end
242                    end
243      ////////////////////////////////////////////////////////////////////////////////
244      /////////// Luetaan saapunut data ja lähetetään se matriisin moduulille /////////
245      ////////////////////////////////////////////////////////////////////////////////
246            6'd20:begin
247      // Vaihdetaan lähetettäväksi osoitteeksi luettavan datan osoite ja nollataan
     laskuri
248                       state<=8'd64;
249                       vaihe<= 6'd21;
250                       bCounter<=16'd1;
251                    end
252      ////////////////////////////////////////////////////////////////////////////////
253            6'd21:begin
254                       start<=1'b1; vaihe<= 6'd22; dataInCLK<=1'b0;
255                    end
256      ////////////////////////////////////////////////////////////////////////////////
257            6'd22:begin
258      // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
```

```
259                     if(SSpin) begin
260                        start<=1'b0;
261                        trg<=1'b1;
262                        vaihe<= 6'd23;
263                     end
264                  end
265      ////////////////////////////////////////////////////////////////////////////////
266            6'd23: begin
267                     if(SSpin) begin
268                        if(bCounter < len) begin// Luetaan ensimmäinen tavu osoitteesta
269                           bCounter<=bCounter+1'b1;
270                           if(osoite == 16'hFFFF) begin osoite<=16'd0; end
271                           else begin osoite<=osoite+1'b1; end
272                           dataIn<=readByte;
273                           vaihe<=6'd24;
274                        end
275      // Kun viimeinen tavu on luettu, lähetetään se eteenpäin ja siirrytään
     viimeistelyyn
276                        else begin
277                           dataIn<=readByte;
278                           osoite<=osoite+1'b1;
279                           vaihe<=6'd25;
280                        end
281                     end
282                  end
283      // Nostetaan matriisille menevän datan kello ylös ja jatketaan seuraava tavun
     lukemiseen
284            6'd24:begin
285                     dataInCLK<=1'b1;
286                     vaihe<=6'd21;
287                  end
288      // Nostetaan matriisille menevän datan kello ylös ja jatketaan eteenpäin
289            6'd25:begin
290                     dataInCLK<=1'b1;
291                     vaihe<=6'd26;
292                  end
293      ////////////////////////////////////////////////////////////////////////////////
294            6'd26:begin
295                     dataInCLK<=1'b0;     // Lasketaan matriisille menevän datan kello alas
296                     vaihto<=1'b1;        // Nostetaan vaihto muuttuja ylös merkiksi siittä
     että kuva on lähetetty
297                     vaihe<=6'd27;        // Siirrytään eteenpäin
298                  end
299      ////////////////////////////////////////////////////////////////////////////////
300      /////////// Kerrotaan w5500 piirille uusi datan offset osoite //////////////////
301      ////////////////////////////////////////////////////////////////////////////////
302            6'd27:begin
303                     vaihto<=1'b0;
304                     state<=8'd65;
305                     vaihe<=6'd28;
306                  end
307      ////////////////////////////////////////////////////////////////////////////////
308            6'd28:begin
309                     start<=1'b1; vaihe<= 6'd29;
310                  end
311      // Jos SS on ylhäällä niin lähetetään seuraava komento SPI väylällä W5500 piirille
312            6'd29:begin
313                     if(SSpin) begin
314                        start<=1'b0;
315                        trg<=1'b1;
316                        vaihe<=6'd30;
317                        state<=state+1'b1;
318                     end
319                  end
320      ////////////////////////////////////////////////////////////////////////////////
321            6'd30:begin
```

```
322                     if(state <= 8'd67) begin
323                        vaihe<=6'd28;
324                     end
325        // Palataan takaisin lukeen kannan statusta jos kaikki on ok!
326                     else begin
327                        vaihe<=6'd9;
328                     end
329                  end
330        ////////////////////////////////////////////////////////////////////////////////
331           endcase
332        end
333     //
334     //
335     //
336        always @ (posedge start) begin
337           case(state)
338     //
339     //       Kantojen 1-7 RX muistin kooksi muutetaan 0 tavua
340              8'd1:  begin   offsetAdr<=16'h001E; cntlByte<= 8'h2C; writeByte<=8'h00; end
341              8'd2:  begin   offsetAdr<=16'h001E; cntlByte<= 8'h4C; writeByte<=8'h00; end
342              8'd3:  begin   offsetAdr<=16'h001E; cntlByte<= 8'h6C; writeByte<=8'h00; end
343              8'd4:  begin   offsetAdr<=16'h001E; cntlByte<= 8'h8C; writeByte<=8'h00; end
344              8'd5:  begin   offsetAdr<=16'h001E; cntlByte<= 8'hAC; writeByte<=8'h00; end
345              8'd6:  begin   offsetAdr<=16'h001E; cntlByte<= 8'hCC; writeByte<=8'h00; end
346              8'd7:  begin   offsetAdr<=16'h001E; cntlByte<= 8'hEC; writeByte<=8'h00; end
347     //       Kantojen 1-7 TX muistin kooksi muutetaan 0 tavua
348              8'd8:  begin   offsetAdr<=16'h001F; cntlByte<= 8'h2C; writeByte<=8'h00; end
349              8'd9:  begin   offsetAdr<=16'h001F; cntlByte<= 8'h4C; writeByte<=8'h00; end
350              8'd10: begin   offsetAdr<=16'h001F; cntlByte<= 8'h6C; writeByte<=8'h00; end
351              8'd11: begin   offsetAdr<=16'h001F; cntlByte<= 8'h8C; writeByte<=8'h00; end
352              8'd12: begin   offsetAdr<=16'h001F; cntlByte<= 8'hAC; writeByte<=8'h00; end
353              8'd13: begin   offsetAdr<=16'h001F; cntlByte<= 8'hCC; writeByte<=8'h00; end
354              8'd14: begin   offsetAdr<=16'h001F; cntlByte<= 8'hEC; writeByte<=8'h00; end
355     //       Kannan 0 RX ja TX muistin kooksi muutetaan 16 kilotavua
356              8'd15: begin   offsetAdr<=16'h001E; cntlByte<= 8'h0C; writeByte<=8'h10; end
357              8'd16: begin   offsetAdr<=16'h001F; cntlByte<= 8'h0C; writeByte<=8'h10; end
358     //       MAC osoite:
359              8'd17: begin   offsetAdr<=16'h0009; cntlByte<= 8'h04; writeByte<=8'hDE; end
360              8'd18: begin   offsetAdr<=16'h000A; cntlByte<= 8'h04; writeByte<=8'hAD; end
361              8'd19: begin   offsetAdr<=16'h000B; cntlByte<= 8'h04; writeByte<=8'hBE; end
362              8'd20: begin   offsetAdr<=16'h000C; cntlByte<= 8'h04; writeByte<=8'hEF; end
363              8'd21: begin   offsetAdr<=16'h000D; cntlByte<= 8'h04; writeByte<=8'hFE; end
364              8'd22: begin   offsetAdr<=16'h000E; cntlByte<= 8'h04; writeByte<=8'hED; end
365     //       IP numero:
366              8'd23: begin   offsetAdr<=16'h000F; cntlByte<= 8'h04; writeByte<=8'hC0; end
367              8'd24: begin   offsetAdr<=16'h0010; cntlByte<= 8'h04; writeByte<=8'hA8; end
368              8'd25: begin   offsetAdr<=16'h0011; cntlByte<= 8'h04; writeByte<=8'h01; end
369              8'd26: begin   offsetAdr<=16'h0012; cntlByte<= 8'h04; writeByte<=8'hB1; end
370     //       Aliverkko:
371              8'd27: begin   offsetAdr<=16'h0001; cntlByte<= 8'h04; writeByte<=8'hC0; end
372              8'd28: begin   offsetAdr<=16'h0002; cntlByte<= 8'h04; writeByte<=8'hA8; end
373              8'd29: begin   offsetAdr<=16'h0003; cntlByte<= 8'h04; writeByte<=8'h01; end
374              8'd30: begin   offsetAdr<=16'h0004; cntlByte<= 8'h04; writeByte<=8'h01; end
375     //       Aliverkon peite:
376              8'd31: begin   offsetAdr<=16'h0005; cntlByte<= 8'h04; writeByte<=8'hFF; end
377              8'd32: begin   offsetAdr<=16'h0006; cntlByte<= 8'h04; writeByte<=8'hFF; end
378              8'd33: begin   offsetAdr<=16'h0007; cntlByte<= 8'h04; writeByte<=8'hFF; end
379              8'd34: begin   offsetAdr<=16'h0008; cntlByte<= 8'h04; writeByte<=8'h00; end
380     //       Suljetaan kannat 1-7 kaikelta liikenteeltä
381              8'd35: begin   offsetAdr<=16'h0001; cntlByte<= 8'h2C; writeByte<=8'h10; end
382              8'd36: begin   offsetAdr<=16'h0001; cntlByte<= 8'h4C; writeByte<=8'h10; end
383              8'd37: begin   offsetAdr<=16'h0001; cntlByte<= 8'h6C; writeByte<=8'h10; end
384              8'd38: begin   offsetAdr<=16'h0001; cntlByte<= 8'h8C; writeByte<=8'h10; end
385              8'd39: begin   offsetAdr<=16'h0001; cntlByte<= 8'hAC; writeByte<=8'h10; end
386              8'd40: begin   offsetAdr<=16'h0001; cntlByte<= 8'hCC; writeByte<=8'h10; end
387              8'd41: begin   offsetAdr<=16'h0001; cntlByte<= 8'hEC; writeByte<=8'h10; end
```

```
388   //    Yleiset asetukset
389            8'd42: begin   offsetAdr<=16'h0000; cntlByte<= 8'h04; writeByte<=8'h00; end
390            8'd43: begin   offsetAdr<=16'h0016; cntlByte<= 8'h04; writeByte<=8'h00; end
391            8'd44: begin   offsetAdr<=16'h0019; cntlByte<= 8'h04; writeByte<=8'h07; end
392            8'd45: begin   offsetAdr<=16'h001A; cntlByte<= 8'h04; writeByte<=8'hD0; end
393            8'd46: begin   offsetAdr<=16'h001B; cntlByte<= 8'h04; writeByte<=8'h08; end
394   //    Asetetaan kannan 0 Moodiksi TCP
395            8'd52: begin   offsetAdr<=16'h0000; cntlByte<= 8'h0C; writeByte<=8'h01; end
396   //    Asetetaan kannan 0 kohdeportti (8888)
397            8'd53: begin   offsetAdr<=16'h0004; cntlByte<= 8'h0C; writeByte<=8'h22; end
398            8'd54: begin   offsetAdr<=16'h0005; cntlByte<= 8'h0C; writeByte<=8'hB8; end
399   //    Avataan kanta 0 TCP tilaan
400            8'd55: begin   offsetAdr<=16'h0001; cntlByte<= 8'h0C; writeByte<=8'h01; end
401   //    Luetaan kannan 0 status
402            8'd56: begin   offsetAdr<=16'h0003; cntlByte<= 8'h08; writeByte<=8'h00; end
403   //    Avataan TCP serveri kannassa 0
404            8'd57: begin   offsetAdr<=16'h0001; cntlByte<= 8'h0C; writeByte<=8'h02; end
405   //    Luetaan kannan 0 status
406            8'd58: begin   offsetAdr<=16'h0003; cntlByte<= 8'h08; writeByte<=8'h00; end

407   //    Suljetaan kannan 0 yhteys
408            8'd59: begin   offsetAdr<=16'h0001; cntlByte<= 8'h0C; writeByte<=8'h08; end
409   //    Luetaan saapuvan datan määrä
410            8'd60: begin   offsetAdr<=16'h0026; cntlByte<= 8'h08; writeByte<=8'h00; end
411            8'd61: begin   offsetAdr<=16'h0027; cntlByte<= 8'h08; writeByte<=8'h00; end
412   //    Luetaan saapuvan datan offset osoite
413            8'd62: begin   offsetAdr<=16'h0028; cntlByte<= 8'h08; writeByte<=8'h00; end
414            8'd63: begin   offsetAdr<=16'h0029; cntlByte<= 8'h08; writeByte<=8'h00; end
415   //    Datan lukemista
416            8'd64: begin   offsetAdr<=osoite;   cntlByte<= 8'h18; writeByte<=8'h00; end
417   //    Ilmoitetaan kannalle uusi osoite
418            8'd65: begin   offsetAdr<=16'h0028; cntlByte<= 8'h0C; writeByte<=(osoite >> 8);
        end
419            8'd66: begin   offsetAdr<=16'h0029; cntlByte<= 8'h0C; writeByte<=(osoite & 8'hFF
    ); end
420   //    Lähetetään kannalle 0 RECV tieto
421            8'd67: begin   offsetAdr<=16'h0001; cntlByte<= 8'h0C; writeByte<=8'h40; end
422
423   //
424          endcase
425      end
426   //
427   always@(posedge clk) begin
428            clk2<=!clk2;
429      end
430   //
431   endmodule
```

Liite 5. SPI moduuli

```
1   module SPImaster(SCK, MOSI, MISO, SSpin, offsetAdr, cntlByte, writeByte, readByte, clk,
    trg, trgD);
2   // Moduuli W5500 piirin kontrolloimiseen SPI protokolaa käyttäen.
3   // SPI moodi on 0 ja SCK taajuus 12.5 Mhz (neljäs pääkellosta)
4   //
5   // Määritellään moduulin lähdöt
6      output SCK, MOSI, SSpin, trgD;
7      // SCK = Serial Clock, MOSI = Master Out, Slave In, SS = Slave Select
8      input MISO, clk, trg;
9      // MISO = Master In, Slave Out
10     input [15:0] offsetAdr;
11     // W5500 piirin offsetAdress, valitsee osoitetavut
12     input [7:0] cntlByte;
13     // W5500 piirin control byte, valitsee rekisterin
14     input [7:0] writeByte;
15     // Rekisteriin tai muistipaikkaan kirjoitettava tavu
16     output[7:0] readByte;
17     // Rekisteristä tai muistipaikasta luettava tavu
18  //
19  // Luodaan tarvittavat muuttujat
20     reg SCK, MOSI, SSpin, init, trgD, clk2;
21     reg [7:0] readByte;
22     reg [5:0] Adr;
23  // Always blokki joka käydään läpi jokaisella kellon nousevalla reunalla
24     always@(posedge clk2) begin
25     /////////// Suoritetaan aina kun trigger ja SS on ylhäällä ////////////////////
26        if(SSpin && trg) begin
27           SSpin<=1'b0;
28           MOSI<=offsetAdr[6'd15];
29           Adr<=6'd32;
30           trgD<=1'b1;
31        end
32     /////////// Suoritetaan jos SCK ja SS on alhaalla ////////////////////////////
33        else if(!SSpin && !SCK) begin
34           if(Adr == 6'd0) begin SSpin<=1'b1; trgD<=1'b0; end
35              // Jos Adr on 0 nostetaan SS takaisin ylös
36           else if(Adr > 6'd0) begin
37              // Jos Adr on enemmän kuin 0, vähennetään siittä 1 ja nostetaan SCK
38              Adr=Adr-1'b1;
39              SCK=1'b1;
40     /////////// Suoritetaan aina kun SCK nostetaan ylös////////////////////////////
41              if(Adr <= 6'd8) begin
42                 readByte[Adr] <= MISO;
43              end
44           end
45        end
46     /////////// Suoritetaan jos SCK on ylhäällä ja SS alhaalla ////////////////////
47        else if(SSpin == 1'b0 && SCK) begin
48           SCK<=1'b0;
49     /////////// Suoritetaan aina kun SCK lasketaan alas ///////////////////////////
50           if(Adr >= 6'd17)                    begin MOSI<=offsetAdr[Adr-6'd17]; end
51           else if(Adr <= 6'd16 && Adr >= 6'd9) begin MOSI<=cntlByte[Adr-6'd9]; end
52           else if(Adr <= 6'd8)                begin MOSI<=writeByte[Adr-1'b1]; end
53        end
54     end
55  //
56     /////////// Kellonjakaja, jakaa pääkellon kahdella ////////////////////////////
57     always@(posedge clk) begin
58           clk2<=!clk2;
59     end
60  //
61  endmodule
```

Liite 6. Matriisin moduuli 1(3)

```
module matriisi(redOut, greOut, bluOut, ABC, LA, outCLK, OE, dataIn, dataInCLK, clk,
vaihto, tp1, tp2);
   //
   output [3:0] redOut, greOut, bluOut;
   output [2:0] ABC;
   output LA, outCLK, OE, tp1, tp2;
   input [7:0] dataIn;
   input dataInCLK, clk, vaihto;
   //
   localparam yKerroin = 7'd64;
   localparam kKerroin = 10'd512;
   //
   wire [9:0] wAdr;
   //
   reg [3:0] redOut,greOut,bluOut;
   wire [2:0] ABC;
   reg LA, outCLK, OE, clk2, tp1, tp2;
   //
   reg [5:0] xIn, xOut;
   reg [2:0] yIn, yOut, vIn, Vaihe;
   reg [1:0] kOut;
   reg [9:0] rAdr, viiveLaskuri;
   reg [15:0] valiViive;
   //
   reg [3:0] red1 [1023:0];
   reg [3:0] red2 [1023:0];
   reg [3:0] red3 [1023:0];
   reg [3:0] red4 [1023:0];
   reg [3:0] gre1 [1023:0];
   reg [3:0] gre2 [1023:0];
   reg [3:0] gre3 [1023:0];
   reg [3:0] gre4 [1023:0];
   reg [3:0] blu1 [1023:0];
   reg [3:0] blu2 [1023:0];
   reg [3:0] blu3 [1023:0];
   reg [3:0] blu4 [1023:0];
   //
   //
   reg kuva,kuvaNro;
   //
   //
   always @(posedge clk2)begin   // Datan kirjoititus siirtorekistereille
      case(Vaihe)
         3'd0: begin    // Vaihe 0: Osoitteen muokkaus.
                  rAdr<=xOut+(yKerroin*yOut)+(kKerroin*kuva);
                  Vaihe<=Vaihe+1'b1;
               end
         3'd1:    begin    // Vaihe 1: datan kirjoitus.
                  case(kOut)
                     3'd0: begin
                           redOut <= red1[rAdr];
                           greOut <= gre1[rAdr];
                           bluOut <= blu1[rAdr];
                           end
                     3'd1: begin
                           redOut <= red2[rAdr];
                           greOut <= gre2[rAdr];
                           bluOut <= blu2[rAdr];
                           end
                     3'd2: begin
                           redOut <= red3[rAdr];
                           greOut <= gre3[rAdr];
                           bluOut <= blu3[rAdr];
                           end
                     3'd3: begin
                           redOut <= red4[rAdr];
```

2(3)

```
66                              greOut <= gre4[rAdr];
67                              bluOut <= blu4[rAdr];
68                              end
69                      endcase
70                      Vaihe<=Vaihe+1'b1;
71                  end
72              3'd2: begin     // Vaihe 2: kellon nosto ylös ja luku kellon lasku alas.
73                      outCLK<=1'b1;
74                      Vaihe<=Vaihe+1'b1;
75                  end
76          // Vaihe 3: Kello alas ja xOut laskurin lisäys jos dataa on jäljellä, muuten
77          // nollataan x, nostetaan Latch ylös ja siirrytään uuteen vaiheeseen.
78              3'd3: begin
79                      outCLK<=1'b0;
80                      if(xOut<6'd63)begin xOut<=xOut+1'b1; Vaihe<=3'd0; end
81                      else begin xOut<=6'd0; Vaihe<=Vaihe+1'b1; LA<=1'b1; end
82                  end
83          // Vaihe 4: lasketaan Latch ja OutputEnable alas ja siirrytään
84          // eteenpäin (OE on aktiivinen alhaalla).
85              3'd4: begin LA<=1'b0; OE<=1'b0; Vaihe<=Vaihe+1'b1; end
86          // Vaihe 5: Viive. Viiveen jälkeen nollataan laskuri, nostetaan
87          // OE ylös ja joko lisätään y laskuria tai siirrytään eteenpäin.
88              3'd5: begin
89                      if(kOut==2'd0&&viiveLaskuri<10'd2)begin
90                          viiveLaskuri<=viiveLaskuri+1'b1;
91                      end
92                      else if(kOut==2'd1&&viiveLaskuri<10'd20)begin
93                          viiveLaskuri<=viiveLaskuri+1'b1;
94                      end
95                      else if(kOut==2'd2&&viiveLaskuri<10'd52)begin
96                          viiveLaskuri<=viiveLaskuri+1'b1;
97                      end
98                      else if(kOut==2'd3&&viiveLaskuri<10'd128)begin
99                          viiveLaskuri<=viiveLaskuri+1'b1;
100                     end
101                     else begin
102                         viiveLaskuri<=10'd0;
103                         OE<=1'b1;
104                         tp1<=1'b0;
105                         if (yOut<3'd7)begin
106                             yOut<=yOut+1'b1;
107                             Vaihe<=3'd0;
108                         end
109                         else begin
110                             if(kOut<2'd3)begin
111                                 kOut<=kOut+1'b1;
112                                 Vaihe<=3'd0;
113                                 yOut<=3'd0;
114                             end
115                             else begin
116                                 kOut<=2'd0;
117                                 yOut<=3'd0;
118                                 tp1<=1'b1;
119                                 Vaihe<=Vaihe+1'b1;
120                             end
121                         end
122                     end
123                 end
124             3'd6: begin     // Vaihe 6: Kirkkauserojen jälkeinen viive
125                     if(valiViive<16'd30000)begin
126                         valiViive<=valiViive+1'b1;
127                     end
128                     else begin
129                         valiViive<=16'd0;
130                         Vaihe<=3'd0;
131                         kuva<=~kuvaNro;
```

```
132                     end
133                 end
134         endcase
135     end
136     //
137     assign wAdr = xIn+(yKerroin*yIn)+(kKerroin*kuvaNro);
138     assign ABC = yOut;
139     //
140     always @(posedge dataInCLK)begin
141         case(vIn)
142             3'd0: begin
143                   red1[wAdr]<=(dataIn >> 4);
144                   red2[wAdr]<=(dataIn & 4'b1111);
145                   end
146             3'd1: begin
147                   red3[wAdr]<=(dataIn >> 4);
148                   red4[wAdr]<=(dataIn & 4'b1111);
149                   end
150             3'd2: begin
151                   gre1[wAdr]<=(dataIn >> 4);
152                   gre2[wAdr]<=(dataIn & 4'b1111);
153                   end
154             3'd3: begin
155                   gre3[wAdr]<=(dataIn >> 4);
156                   gre4[wAdr]<=(dataIn & 4'b1111);
157                   end
158             3'd4: begin
159                   blu1[wAdr]<=(dataIn >> 4);
160                   blu2[wAdr]<=(dataIn & 4'b1111);
161                   end
162             3'd5: begin
163                   blu3[wAdr]<=(dataIn >> 4);
164                   blu4[wAdr]<=(dataIn & 4'b1111);
165                   end
166         endcase
167     end
168     // Sisääntulevan datan laskureiden lisäys kellon negatiivisella reunalla
169     always @ (negedge dataInCLK)begin
170         if(xIn<6'd63)begin
171             xIn<=xIn+1'b1;
172         end
173         else begin
174             if(yIn<3'd7)begin yIn<=yIn+1'b1; xIn<=6'd0; end
175             else begin
176                 if(vIn<3'd5) begin vIn<=vIn+1'b1; xIn<=6'd0; yIn<=3'd0; end
177                 else begin xIn<=6'd0; yIn<=3'd0; vIn<=3'd0; end
178             end
179         end
180     end
181     //
182     always@ (posedge vaihto)begin
183         kuvaNro<=~kuvaNro;
184         tp2<=~tp2;
185     end
186     //
187     always@(posedge clk) begin clk2<=!clk2; end
188 endmodule
```