VIISTOVALOKUVAUSLAITTEEN KUVAUSVALOJEN OHJAINLAITE

Panu Vuorenmaa

Opinnäytetyö
Huhtikuu 2015
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

VUORENMAA, PANU:
Viistovalokuvauslaitteen kuvausvalojen ohjainlaite

Opinnäytetyö 51 sivua, joista liitteitä 13 sivua
Huhtikuu 2015

Tampereen ammattikorkeakoululle rakennettavan viistovalokuvauslaitteen kuvausvalojen ohjainlaite suunniteltiin yhteensopivaksi jo aiemmin rakennetun viistovalokuvauslaitteiston kanssa. Laitteen ominaisuuksia suunniteltaessa hyödynnettiin Tampereen teknillisen yliopiston vastaavalla periaatteella toimivaa laitteistoa.

Ohjainlaitteelle asetetaan tietokoneen kautta tieto siitä, mitä 12 kuvausvalosta halutaan kuvauksessa käyttää. Ohjainlaite osaa sytyttää, sammuttaa ja vaihtaa seuraavaan kuvausvaloon itsenäisesti kamerasta saadun signaalin avulla.

Valojen ohjainlaite liitetään kameran salamakenkään, jotta saadaan tieto siitä, milloin kamera aloittaa kuvan ottamisen. Tiedon avulla sytytetään ja sammutetaan valittu kuvausvalo, sekä vaihdetaan seuraavaan valoon kuvan ottamisen jälkeen. Salamakengän signaaleista valittiin käyttöön kaksi sopivinta signaalia oskilloskoopilla mittaamalla.

Ohjainlaitteessa käytetyt komponentit valittiin siten, että niiden saatavuus on hyvä ja ovat hinnaltaan edullisia. Valintaan vaikuttivat myös viistovalokuvauslaitteistoon jo valmiiksi asennettujen ledien ominaisuudet. Piirilevy suunniteltiin KiCad-ohjelmistolla siten, että se on mahdollista valmistaa yksipuoleisena TAMKin piirilevyjyrsimellä.

Kuvausvalojen ohjainlaitteen laiteohjelmisto kirjoitettiin Atmel Studio -ympäristössä C-kielellä. Laiteohjelmisto suunniteltiin niin, että sitä on helppo ohjata tietokoneen USB-väylän kautta. Laitteen ohjaukseen käytetyt käskyt tehtiin johdonmukaisiksi, jotta laitetta voidaan ohjata myös terminaalin kautta käsinkirjoitetuilla käskyillä. Käskyjen testaukseen ohjelmoitiin myös työkalu, jolla ohjainlaitteen toimintaa on mahdollista testata ilman varsinaista viistovalokuvauslaitteen käyttöliittymää.

Asiasanat: piirilevysuunnittelu, sulautetut järjestelmät, viistovalokuvaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT
Embedded Systems and Electronics

VUORENMAA, PANU:
Light Controller for Photometric Stereo Equipment

Bachelor's thesis 51 pages, appendices 13 pages
April 2015

This project was to build a light controller for the photometric stereo equipment at Tampere University of Applied Sciences. The controller was designed to use similar principles as used in the similar equipment at Tampere University of Technology.

The equipment has 12 lights that can be controlled with the controller that has been configured with a computer through an USB bus. The controller can switch the lights on/off and change to the next light independently.

The controller is connected to the camera's hotshoe. The hotshoe provides signals that is used to get the information when the exposure starts and ends, so the controller can act accordingly. The used signals were found from the hotshoe by measuring them with an oscilloscope.

Components used in the controller were chosen so that they were available and inexpensive. The PCB was designed with the KiCad software suite so that it was possible to manufacture with the TAMK's CNC router.

Firmware for the controller were written with Atmel Studio in C-language. The firmware were designed so that it was easy to control through the USB bus from the computer. The commands used were made logical, so that the controller is easy to control from the terminal with the handwritten commands.

Key words: embedded systems, PCB design, photometric stereo

SISÄLLYS

LYHENTEET JA TERMIT

EMI	Electromagnetic Interference, sähkömagneettinen häiriö
USART	Universal Asynchronous Receiver Transmitter, synkroninen/asynkroninen vastaanotin/lähetin
E-TTL	Evaluative-Through The Lens, Canon-kameroissa käytetty valotuksen mittausjärjestelmä
USB	Universal Serial Bus, yleisesti käytössä oleva sarjaväyläarkkitehtuuri
TTL	Transistor-transistor logic, bipolaaritransistoripohjainen logiikka
I/O	Input/Output, sisään- ja ulostulo

1 JOHDANTO

Viistovalokuvauslaitteella voidaan tutkia pinnanmuotojen yksityiskohtia eli topografiaa. Kuvauksen avulla voidaan löytää esimerkiksi paperin pinnasta mahdolliset virheet. Kuvaus perustuu materiaalin pinnan kuvaamiseen, kun sitä valaistaan ennalta määrätyistä kulmista, jonka jälkeen kuvista voidaan luoda pinnan korkeuskartta. (Innventia)

Projektin tarkoituksena oli rakentaa kuvausvalojen ohjauslaite TAMKin viistovalokuvauslaitteelle. Käytännössä rakennettava viistovalokuvauslaite on ulkopuoliselta valolta suljettu kaappi, joka sisältää järjestelmäkameran, led-kuvausvalot sekä kelkan, jotta kuvattava näyte on helppo vaihtaa.

Kuvausvalojen ohjauslaite suunniteltiin ohjaamaan TAMKin viistovalokuvauslaitteen kuvausvaloja. Kuvausvaloina käytetään 12 kappaletta led-valoja sekä kameran tarkennuksessa käytettäviä laser-valoja.

2 LAITTEEN SUUNNITTELU

Tämän projektin tarkoituksena oli suunnitella laite, jolla voidaan ohjata viistovalokuvauslaitteeseen jo ennestään asennettuja led-kuvausvaloja tietokoneelta. Suunnittelun aluksi kartoitettiin tarvittavat ominaisuudet tapaamisissa viistovalokuvauslaitteesta vastaavien opettajien kanssa. Mukana oli myös Tampereen teknillisen yliopiston (TTY) henkilökuntaa, joiden kokemus vastaavista laitteista oli hyödyksi. Erillisenä opinnäytetyönä laitteeseen ohjelmoidaan käyttöliittymä kuvien ottamista varten, jonka kautta on mahdollista lähettää käskyjä kuvausvalojen ohjauslaitteelle ja sitä kautta ohjata kuvausvaloja.

2.1 Kuvausvalojen ohjainlaitteen toimintaperiaate

Kuvausvalojen ohjainlaitteen toimintaperiaate on, että tietokoneelta asetetaan tieto kuvauksessa käytettävistä valoista laitteelle. Kun kameran salamakengästä tuleva signaali menee ylätilaan, ohjainlaite sytyttää valituista valoista ensimmäisen, eli kamera aloittaa valotusmittauksen, jonka jälkeen valotuksen. Kuvausvalo on päällä signaalin ylhäälläoloajan, jonka jälkeen laite on valmiina sytyttämään seuraavan vuorossa olevan valon seuraavaa kuvaa varten kameran signaalin mukaisesti.

Ohjainlaite kytketään USB-väylällä tietokoneeseen sekä kameran salamaliittimeen (hotshoe). Tietokone tunnistaa laitteen virtuaalisena sarjaporttina, jonka kautta laitteelle voidaan lähettää käskyjä sen konfigurointia varten. Ohjainlaite saa signaalin kameran salamakengästä, jonka avulla laite saa tiedon siitä, että kuva on otettu.

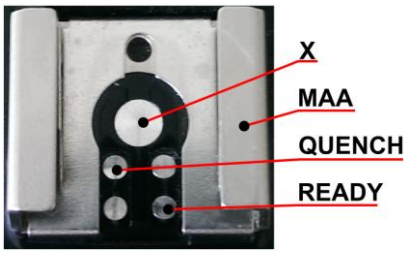
Käytännössä laite toimii kuin yksinkertainen tilakone, joka konfiguroidaan tietokoneelta annettavilla käskyillä. Laitteelle voidaan antaa kolme eri käskyä, joilla asetetaan haluttu toimintatila. Laitteelle voidaan antaa käsky, jolla yksittäinen valo saadaan syttymään hetkeksi, asettaa kuvausvalojen ohjaussekvenssi tai ohjata kameran tarkentamisessa käytettyjä laser-valoja päälle tai pois.

2.2 Kameran signaalit

Koska Canon-kameran salamakengän E-TTL-järjestelmän kommunikaatioprotokollasta ei ole dokumentteja, joista sen toiminta selviäisi, tutkittiin salamakengän signaaleja oskilloskoopilla. Signaali, jota voidaan käyttää ohjainlaitteen kanssa, pitää olla amplitudiltaan riittävä sekä ajoitukseltaan sopiva. Jotta signaalia voidaan lukea suoraan mikrokontrollerin pinnillä, täytyy sen olla ylätilassa noin viisi volttia sekä kestää myös pieni kuormitus jännitteen alenematta liikaa.

Oskilloskoopilla mitattaessa salamakengän signaaleja todettiin, että mikäli salamakenkään ei ole kytketty Canon E-TTL-järjestelmää tukevaa ulkoista salamaa, salamakengän signaalit ovat lähes identtisiä ja eroavat toisistaan vain jännitetasoiltaan. Huomattavaa on myös se, että salamakengän maadoitusraudan alla on piilossa pieni kytkin. Mikäli kytkin ei ole pohjassa, ei salamakenkä aktivoidu.

Jotta signaali saadaan kytketyksi kuvausvalojen ohjainlaitteeseen, tarvitsee se kaapelin, jonka toisessa päässä on salamakenkään (kuva 1) sopiva liitin.

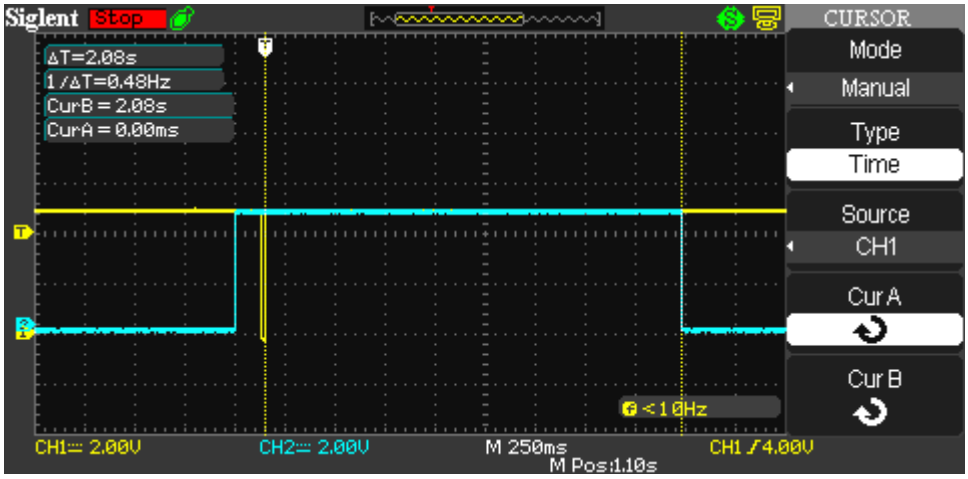


KUVA 1. Canon-kameran salamakengän signaalit

Quench-signaalia mittaamalla todettiin, että 1 kΩ:n vastuksella kuormitettuna jännitetaso ei putoa kuin 0,6 V. Signaalia voidaan täten käyttää TTL-tasoiseen sisääntuloon. Varsinaiseen kytkentään vastusarvoa kasvatettiin 10 kΩ:iin, jolloin jännite putoaa vain 0,1 volttia verrattuna kuormittamattomaan signaaliin.

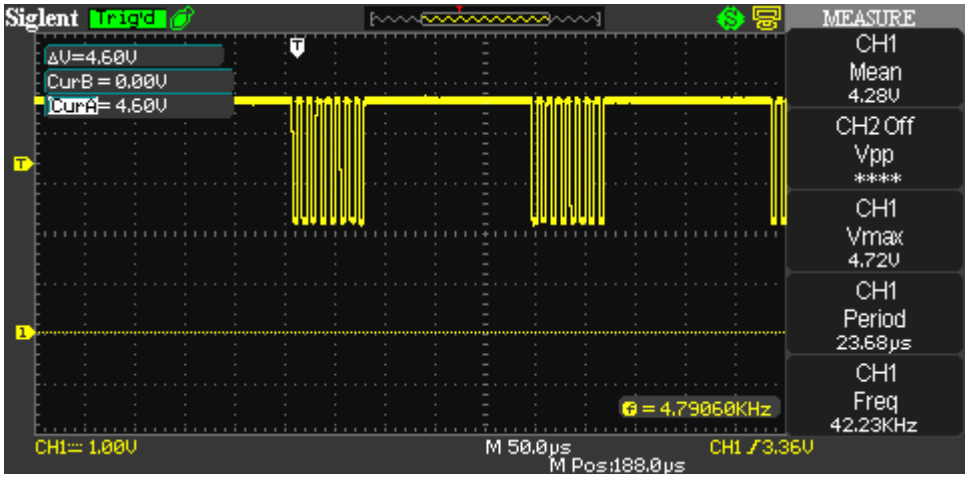
Signaali on alhaalla, kunnes kameran laukaisinta painetaan puoleenväliin. Tämän jälkeen signaali pysyy ylhäällä (5 V), kunnes kuvan ottamisesta on kulunut noin kaksi sekuntia (kuva 2). Oskilloskooppikuvan kanavassa kaksi on quench-signaali ja kanavassa yksi X-signaali. X-signaalista nähdään kuvan valotushetki. Tästä voidaan todeta, että quench-

signaali on ylhäällä 130 ms ennen ja 2 s jälkeen kuvan ottamisen. Yhteensä signaali on siis ylhäällä 2,25 sekuntia, joka on riittävä valojen ohjauksen kannalta.



KUVA 2. Quench-signaalin mittaus.

Mikäli kameran salamakenkään on kytketty E-TTL -järjestelmää käyttävä salama, muuttuu signaalin ominaisuudet täysin, eikä sitä voi enää käyttää tarvittavaan tarkoitukseen (kuva 3).



KUVA 3. Quench-signaali ulkoisen salaman kanssa

Salamakengässä oleva X-signaali on tarkoitettu ulkoisen salaman laukaisemiseen, joten tämä signaali kuulostaa järkevämmältä vaihtoehdolta kuvausvalojen ohjaukseen. Signaalin käytössä tulee kuitenkin ongelma kameran valotusmittauksen kannalta. X-signaalin tila vaihtuu vasta sulkimen ollessa kokonaan auki, ja koska kamera tekee valotusmittauksen ennen varsinaisen valotuksen alkamista, tapahtuisi valotusmittaus täysin pimeässä.

Quench-signaalia käyttämällä tätä ongelmaa ei ole, koska signaali muuttuu heti, kun laukaisinta painetaan.

Mittausten avulla todettiin, että paras signaali kameran ja ohjainlaitteen väliseen synkronointiin on quench-signaali. Mikäli kameran valotus voitaisiin tehdä siten, ettei kameralta tulevaa signaalia tarvita, niin X-signaalin käyttö olisi järkevämpää kuvien valotukseen. Koska X-signaalin päälläoloaika on noin 80 kertaa lyhyempi kuin quench-signaali (kuva 3), tarkoittaa se sitä, että kuvausvalojen ei tarvitse olla päällä kuin vain pelkän valotuksen ajan. Täten ledejä ohjaavan regulaattorin sekä itse ledien lämpeneminen pienentyisi huomattavasti.

Se, kummanko signaalin valitsee käyttöön, riippuu viistovalokuvauslaitteen käyttöliittymän toimintaperiaatteesta. Koska käyttöliittymä ei ole laitetta tehtäessä vielä valmis, laitteeseen tehtiin kytkentä, jolla käytettävä signaali voidaan valita X- tai quench -signaalin kytkimellä.

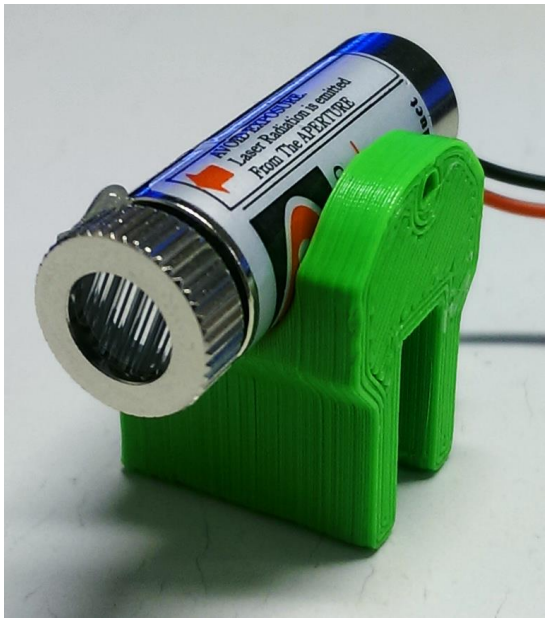
2.3 Laser-tarkennusvalot

Viistovalokuvausta tehtäessä on kameran tarkennuksen onnistuminen välttämätöntä. Mikäli kuvattavana kohteena on valkoinen seinä tai paperi, jossa on vain vähän kontrastin muutoksia, voi automaattinen tarkennus olla mahdotonta (Digital Photography School). TTY:llä tehdyssä viistovalokuvauksessa tämä on ratkaistu piirtämällä lyijykynällä kuvattavaan materiaaliin viivoja, joiden avulla kameran automatiikka pystyy tarkentamaan materiaalin pintaan.

TAMKille rakennettavaan viistovalokuvauslaitteeseen suunniteltiin kokeiltavaksi laserilla kuvattavan materiaalin pintaan heijastetun ristin käyttöä automaattitarkennuksen apuna. Laseria käytetään tarkennuksen apuna esimerkiksi LG:n valmistamassa G3-puhelimessa (TrustedReviews). Lasereilla heijastettu risti helpottaa myös kuvattavan kappaleen kohdistamista alustalle, koska risti vastaa kohtaa, josta kuva otetaan.

Lasereiksi valittiin kaksi kappaletta HLM1230-moduleita ja niihin linssit, joiden avulla laser heijastaa kohteeseen suoran viivan. Laserit on tarkoitus asentaa viistovalokuvaus-

laitteen sisällä oleviin led-valojen telineisiin. Tätä varten tulostettiin TAMKin 3D-tulostimella tarkoitukseen sopivat pidikkeet, joilla laserit on helppo asentaa muokkaamatta alkuperäisiä led-telineitä (kuva 4).



KUVA 4. HLM1230-lasermoduuli telineessä

Alustavissa testeissä kameran tarkentuminen lasereilla heijastettuun kuvioon onnistui. Testi suoritettiin pimeässä huoneessa kamera kohdistettuna valkoiseen A4-paperiin. Kameran tarkennus asetettiin käsin pieleen, jonka jälkeen laserilla heijastettiin viiva paperiin, ja annettiin kameran automaattitarkennuksen tarkentaa kamera. Mikäli kuvio puuttui, ei kameran automaattitarkennus onnistunut.

3 KYTKENTÄ

Kytkennän suunnittelun lähtökohdat olivat yksinkertaisuus, osien saatavuus sekä mahdollinen laajennettavuus. Koska suurin osa komponenteista on yleismallisia, eli saatavilla monelta eri valmistajalta, komponentin valinnassa käytettiin ensisijaisena kriteerinä saatavuutta. Valinnassa käytettiin elektroniikkakomponenttitukkureiden parametrista hakua, joka osoittautui todella hyödylliseksi työkaluksi (http://www.digikey.fi/product-search/). Esimerkiksi MOSFET-transistoria etsittäessä sille annettiin parametreina kotelotyyppi, maksimiresistanssi johtavassa tilassa sekä tehonkesto. Osumat lajiteltiin saatavuuden mukaan ja valittiin hinta huomioiden paras vaihtoehto.

3.1 Kehitysalusta

Kytkentä suunniteltiin Arduino Nano -mikrokontrollerialustan ympärille. Nano sisältää Atmel ATmega 328P -mikrokontrollerin, USB-UART -muuntimen, 5 voltin regulaattorin LM2940 sekä tarvittavat passiivikomponentit. Kaikki tämä on pienellä 45 mm x 18 mm piirilevyllä (kuva 5), josta kaikki I/O-portit on kytketty piikkirimoihin, joten ne on helppo kytkeä esimerkiksi koekytkentäalustaan tai piirilevylle. (Arduino)

KUVA 5. Arduino Nano -kehitysalusta (www.arduino.cc)

Valmiin kehitysalustan käyttäminen kytkennän pohjana yksinkertaistaa piirilevysuunnittelua sekä komponenttihankintaa. Tässä tapauksessa tärkeimmät ominaisuudet kehitysalustalla ovat levylle integroidut mikrokontrollerin oheiskomponentit, kuten kide ja tarvittavat passiivikomponentit sekä USB-UART-muunnin.

Mikrokontrollerialustalla olevan USB-UART-muuntimen avulla mikrokontrolleri voidaan liittää tietokoneeseen USB-väylällä. Tietokone tunnistaa muuntimen virtuaalisena

sarjaporttina, jota voidaan käyttää esimerkiksi terminaaliohjelman avulla. Muunnin keskustelee mikrokontrollerin kanssa USART-väylän kautta, jonka ATmega328P-mikrokontrolleri sisältää. Koska Arduino-kehitysalustojen sisältämän mikrokontrollerin muistiin on jo tehtaalla ladattu alkulatausohjelma (bootloader), voidaan sille ladata ohjelma tietokoneelta USB-väylän kautta. Tämä helpottaa prototyyppivaiheen ohjelmatestausta, sekä myöhemmin tarvittavien päivitysten tekemistä. Alkulatausohjelman ansiosta erillistä mikrokontrollerin ohjelmointilaitetta ei tarvita. Käyttöjännitteet mikrokontrollerille, USB-UART-muuntimelle sekä siirtorekistereille tulevat USB-portin kautta.

Tähän projektiin hankittu kehitysalusta ei ole alkuperäinen Arduino Nano -kehitysalusta, vaan sen pohjalta rakennettu kopio. Suurimpana erona laitteiden välillä on USB-UART-muuntimen tyyppi. Alkuperäisessä laitteessa muunnin on Future Technology Devices Internationalin (FTDI) valmistama FT232RL-piiri. Projektissa käytetyssä kehitysalustassa piiri on WCH:n valmistama CH340-piiri. Tästä erosta johtuen päänvaivaa aiheutti laiteajureiden löytäminen ja asentaminen. Koska FTDI:n valmistamat ajurit ovat saatavilla suoraan Windows-käyttöjärjestelmän automaattisen päivitystyökalun kautta, ei sille tarvitse etsiä tai ladata ajureita erikseen. CH340-piirin ajureiden löytäminen osoittautui hieman hankalammaksi, koska valmistajan sivut ovat vain kiinan kielellä. Ajureiden löytymisen ja asennuksen jälkeen ongelmia ei kuitenkaan näiltä osin ilmennyt.

3.2 Siirtorekisterit

Kytkennässä käytetään kahta NXP:n valmistamaa 8-bittistä siirtorekisteriä tyypiltään 74HC595, jotka ovat kytkettyinä sarjaan. Tämä tarkoittaa sitä, että siirtorekisterien avulla voidaan ohjata yhteensä 16:ta kappaletta ulostuloja. Näistä kuitenkin käytetään tässä kytkennässä vain 14:ää kappaletta.

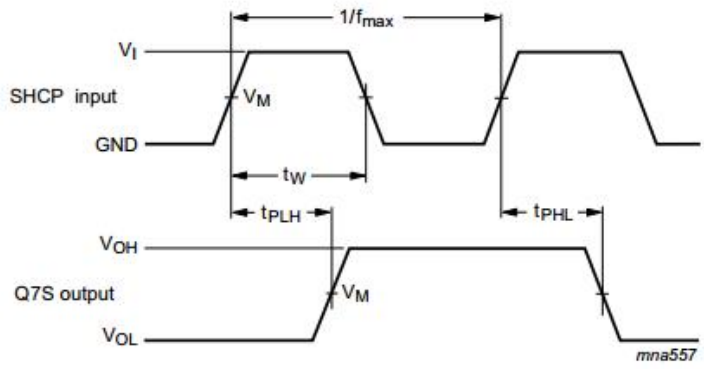
Siirtorekisterit ovat kytkettyinä mikrokontrolleriin kolmella signaalilla: ”DATA”, ”CLOCK” ja ”LATCH”. Mikrokontrollerilla voidaan siis ohjata kolmella pinnillä kaikkia 16:ta kappaletta ulostuloja. Koska käytössä oleva mikrokontrolleri sisältää maksimissaan 8-bittisiä portteja, siirtorekisterien käyttö suoraviivaistaa mikrokontrollerin ohjelmistossa tehtävää ulostulojen ohjausta.

Siirtorekistereitä voidaan kytkeä useampiakin peräkkäin, jolloin samoilla kolmella signaalilla voidaan ohjata vaikka kymmeniä ulostuloja. Ongelmaksi suuremmissa määrissä tosin muodostuu hitaus sekä ohjaussignaalien kuormittuminen.

Siirtorekisterin signaalien ajoitukset on määritelty kyseisen piirin datalehdessä. Koska tässä tapauksessa ulostulosignaalien ajoitukset eivät ole kriittisiä, voidaan ne jättää lähes huomiotta. Tärkeimmät signaalit ovat siirtorekisterin ohjaukseen käytettävät signaalit STCP, SHCP sekä DATA.

Mikrokontrollerille tehdystä siirtorekisterin ohjaukseen käytettävästä funktiosta johtuen nopeimmin muuttuvat signaalit ovat ohjausväylän SHCP- ja STCP-signaalit. SHCP-signaalin nousureunalla luetaan DATA-signaalin arvo (0 tai 1) siirtorekisteriin. Vastaavasti siirtorekisterin sisältämä bitti siirretään sitä vastaavaan ulostuloon STCP-signaalin nousureunalla.

Siirtorekisterin datalehti määrittelee vaadittavat signaalien kestoajat ajoituskaaviolla (kuva 6) ja sitä vastaavalla taulukolla (taulukko 1).



Measurement points are given in Table 8.
V_{OL} and V_{OH} are typical output voltage levels that occur with the output load.

KUVA 6. SHCP-signaalin ajoituskaavio (NXP 2015)

Taulukon 1 mukaan SHCP- sekä STCP-signaalien pulssin pituuden (tw) tulisi olla vähintään 6 ns (25 ºC:n lämpötilassa). Varmuuden vuoksi käytetään kuitenkin ”huonointa” arvoa, joka tässä tapauksessa on minimiarvo 16 ns.

TAULUKKO 1. Siirtorekisterin ajoitustaulukko (NXP 2015)

Symbol	Parameter	Conditions	25 °C			–40 °C to +85 °C		–40 °C to +125 °C		Unit
			Min	Typ[1]	Max	Min	Max	Min	Max	
74HCT595; V_{CC} = 4.5 V to 5.5 V										
t_{pd}	propagation delay	SHCP to Q7S; see Figure 9 [2]	-	25	42	-	53	-	63	ns
		STCP to Qn; see Figure 10 [2]	-	24	40	-	50	-	60	ns
		$\overline{MR}$ to Q7S; see Figure 12 [3]	-	23	40	-	50	-	60	ns
t_{en}	enable time	$\overline{OE}$ to Qn; see Figure 13 [4]	-	21	35	-	44	-	53	ns
t_{dis}	disable time	$\overline{OE}$ to Qn; see Figure 13 [5]	-	18	30	-	38	-	45	ns
t_W	pulse width	SHCP HIGH or LOW; see Figure 9	16	6	-	20	-	24	-	ns
		STCP HIGH or LOW; see Figure 10	16	5	-	20	-	24	-	ns
		$\overline{MR}$ LOW; see Figure 12	20	8	-	25	-	30	-	ns
t_{su}	set-up time	DS to SHCP; see Figure 10	16	5	-	20	-	24	-	ns
		SHCP to STCP; see Figure 11	16	8	-	20	-	24	-	ns
t_h	hold time	DS to SHCP; see Figure 11	3	–2	-	3	-	3	-	ns
t_{rec}	recovery time	$\overline{MR}$ to SHCP; see Figure 12	10	–7	-	13	-	15	-	ns
f_{max}	maximum frequency	SHCP and STCP; see Figure 9 and 10	30	52	-	24	-	20	-	MHz
C_{PD}	power dissipation capacitance	f_i = 1 MHz; V_I = GND to V_{CC} – 1.5 V [6] [7]	-	130	-	-	-	-	-	pF

Koska siirtorekisterin ohjaukseen käytettävässä aliohjelmassa SHCP-signaali ohjataan päälle ja pois peräkkäisillä käskyillä, on signaalin ylhäälläoloaika lyhin mahdollinen käytössä olevalla mikroprosessorilla ja kidetaajuudella. Signaalin ylhäälläoloajan voi päätellä kääntäjän tuottamasta assembly-kielisestä käännöstuloksesta (kuva 7).

```
00000063  RJMP PC+0x0002        Relative jump
                hc595_port &= ~(1 << hc595_dataPin);
00000064  CBI 0x08,0          Clear bit in I/O register
                hc595_port |= (1 << hc595_clockPin);
00000065  SBI 0x08,2          Set bit in I/O register
                hc595_port &= ~( 1 << hc595_clockPin);
00000066  CBI 0x08,2          Clear bit in I/O register
                maski = (maski >> 1);
00000067  LSR R19         Logical shift right
00000068  ROR R18         Rotate right through carry
00000069  SUBI R20,0x01     Subtract immediate
```

KUVA 7. Assembly-kielinen käännöstulos

Käännöstuloksesta näkyy C-kieliset komennot, joilla SHCP-signaali asetetaan ylä- ja heti sen perään alatilaan. Signaali asetetaan ylätilaan komennolla

```
hc595_port &= ~(1 << hc595_clockPin)
```

C-kielisen komennon alla näkyy kääntäjän tuottama assembly-kielinen komento

```
SBI 0x08,2
```

Mikrokontrollerin datalehdestä selviää SBI- ja CBI-komentojen vaikutus, sekä tässä tapauksessa tärkeämpi asia, käskyjen kestoaika eli niiden tarvitsemien kellopulssien lukumäärä (taulukko 2).

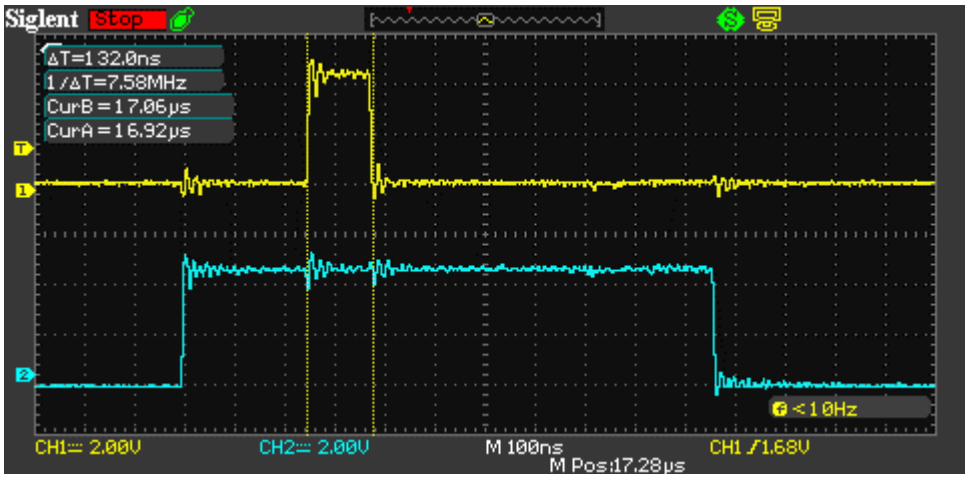
TAULUKKO 2. Prosessorin SBI- ja CBI-käskyn suoritusaika (Atmel 2015)

Mnemonics	Operands	Description	#Clocks
BIT AND BIT-TEST INSTRUCTIONS			
SBI	P,b	Set Bit in I/O Register	2
CBI	P,b	Clear Bit in I/O Register	2
LSL	Rd	Logical Shift Left	1

Taulukosta nähdään, että SBI-komento asettaa I/O-rekisterissä olevan bitin, ja CBI-komento vastaavasti nollaa kyseisen bitin. Molemmat käskyt tarvitsevat suoritukseen kaksi kellopulssia. SHCP-signaali pysyy ylätilassa vain sen aikaa kunnes CBI-komento on suoritettu, joten ylhäälläoloaika on kaksi kellopulssia.

Mikrokontrollerin käyttämän kiteen kellotaajuus on 16 MHz, jolloin yhden kellojakson pituus sekunneissa on sen käänteisluku 1/16 MHz = 62,5 ns. SHCP-signaalin ylhäälläoloaika on siis 125 ns. Pulssin kestoaika on riittävä, koska tarvittava pulssin pituus pitäisi olla minimissään edellä mainittu 16 ns.

SHCP-signaalin ylhäälläoloaika voidaan todeta myös mittaamalla (kuva 8).



KUVA 8. Signaalien SHCP (kanava 1) ja DATA (kanava 2) mittaus

Oskilloskoopilla mitattiin signaalia myös pienemmällä pyyhkäisyajalla tarkkuuden kasvattamiseksi. Tällöin tulokseksi saatiin tasan 125 ns, jonka käänteisluku on 8 MHz. Tämä kerrottuna kahdella, on mikrokontrollerin käyttämän kiteen taajuus 16 MHz, joten mittaamalla saatiin sama tulos kuin päättelemällä.

3.3 MOSFET-transistori

Metallioksidi-puolijohdekanavatransistori (lyh. MOSFET tai FET) on jännitteellä ohjattava transistori. Toisin kuin bipolaaritransistorissa, ei sen ohjaamiseen tarvita jatkuvaa ohjausvirtaa, koska sen tuloresistanssi on hyvin suuri. Suuresta tuloresistanssista johtuen ei FET kuormita ohjausastettaan kytkentä- ja katkaisuhetkeä lukuunottamatta, kuten bipolaaritransistorit.

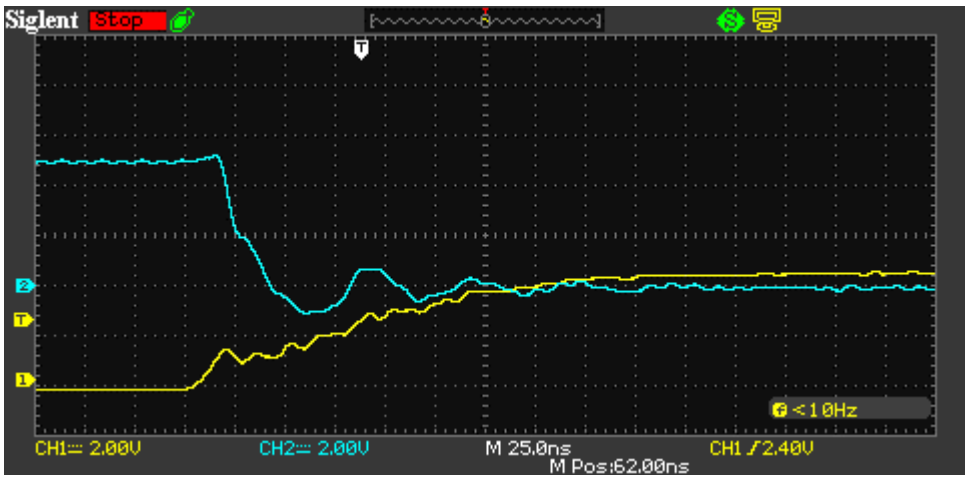
FET:in kanssa on suotavaa käyttää hilavastusta hilan kapasitiivisen kuorman takia. Esimerkiksi jos MOSFET-transistorin hilan tarvitsema varaus on 10 nC ja hilan ohjausjännitteen nousuaika 15 ns, voidaan transistorin hilan piikkivirraksi laskea (1) 670 mA jos syöttävän lähteen lähtöresistanssi olisi 0 Ω.

$$I = Q/t = 10\,nC/15\,ns = 670\,mA \quad (1)$$

Mikäli ohjaava signaali on esimerkiksi mikrokontrollerin tai siirtorekisterin portti, voi portille määritelty suurin sallittu virta-arvo ylittyä. Mikropiirin portin tuhoutuminen erittäin lyhyellä virtapiikillä on kuitenkin epätodennäköistä. Mikropiirin ulostulon tuhoutumisen syynä on usein sen sisäisen resistanssin aiheuttama liiallinen lämpeneminen, ja sitä kautta puolijohteen tuhoutuminen. Tässä tapauksessa puolijohde ei kuitenkaan ehdi juurikaan lämmetä kytkentä- tai katkaisuhetken aikana, joten kytkentä toimii myös ilman hilavastuksia.

Hilavastus estää liiallisen virran ottamisen portista, mutta vastaavasti hidastaa hilan jännitteen nousunopeutta. Mikäli transistoria käytetään kytkimenä, nousunopeuden hidastuminen aiheuttaa tehohäviön kasvamista transistorissa, jos transistori on kauemmin lineaarisessa tilassa. Tässä projektissa tehdyssä kytkennässä hilavastukset kuitenkin puuttuvat. Testien perusteella kytkentä on kuitenkin toimiva, mutta seuraavaan versioon hilavastukset on kuitenkin järkevää lisätä luotettavuuden kannalta.

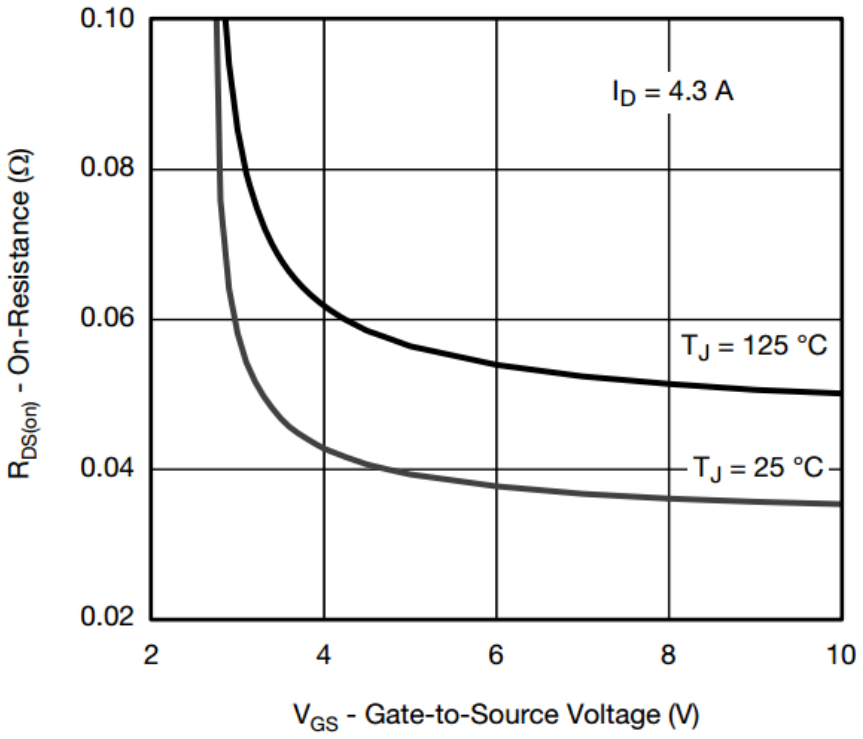
MOSFET-transistoria ohjaavan mikropiirin ulostuloportin sisäresistanssi rajoittaa myös transistorin hilan ottamaa virtapiikkiä kytkentä- ja katkaisuhetkellä. Työssä käytetyn siirtorekisterin ulostuloportin jännitteen nousuajaksi (0 – VCC) mitattiin alle 10 ns, mikäli siihen ei ole kytketty MOSFET-transistoria. Mikäli porttia kuormitetaan FET:in hilalla, portin nousuaika kasvaa (kuva 9).



KUVA 9. FET:n pinnien Gate (kanava 1) ja Drain (kanava 2) jännitteet kytkentähetkellä

Vaatimukset kytkennässä käytetylle N-MOSFET-transistorille on vähintään vakiovirtalähteen syöttämän jännitteen jännitekesto, pieni resistanssi johtavassa tilassa, SOT-23-kotelo sekä saturaatio TTL-tasoisella signaalilla.

Kytkentään valittiin Vishayn valmistama N-kanavainen MOSFET Si2318CDS. Sen ominaisuuksiin kuuluu 40 voltin jännitteenkesto nielun (Drain) ja lähteen (Source) välillä, pieni resistanssi johtavassa tilassa (0,051 Ω, kun Vgs = 4,5 V) sekä ohjaus TTL-tasoisella lähdöllä. FET kestää 2,1 W tehon huoneenlämmössä. (Vishay si2318cd 2012, 1)
Jotta varmistetaan, että FET kestää sen läpi kulkevan virran aiheuttaman lämpötehon, voidaan se laskea käyttämällä sen läpi kulkevaa virtaa, sekä resistanssia johtavassa tilassa (Rds(on)). Resistanssi saadaan datalehden diagrammista (kuva 10).



KUVA 10. Si2318CDS MOSFET-transistorin resistanssi hilajännitteen suhteen (Vishay 2015)

Koska FET:iä ohjaavan siirtorekisterin lähtöjännite on noin 4,5 V, voidaan diagrammista todeta FET:in kanavaresistanssin olevan huoneenlämmössä noin 0,04 Ω. Tässä projektissa ledin läpi kulkeva virta on säädetty noin 350 mA:iin, joten sama virta kulkee myös FET:in läpi. FET:in tehohäviö on vain 5 mW. FET:in johtavassa tilassa oloaika on myös vain muutamia sekunteja kerrallaan, joten lämpeneminen on hyvin pientä.

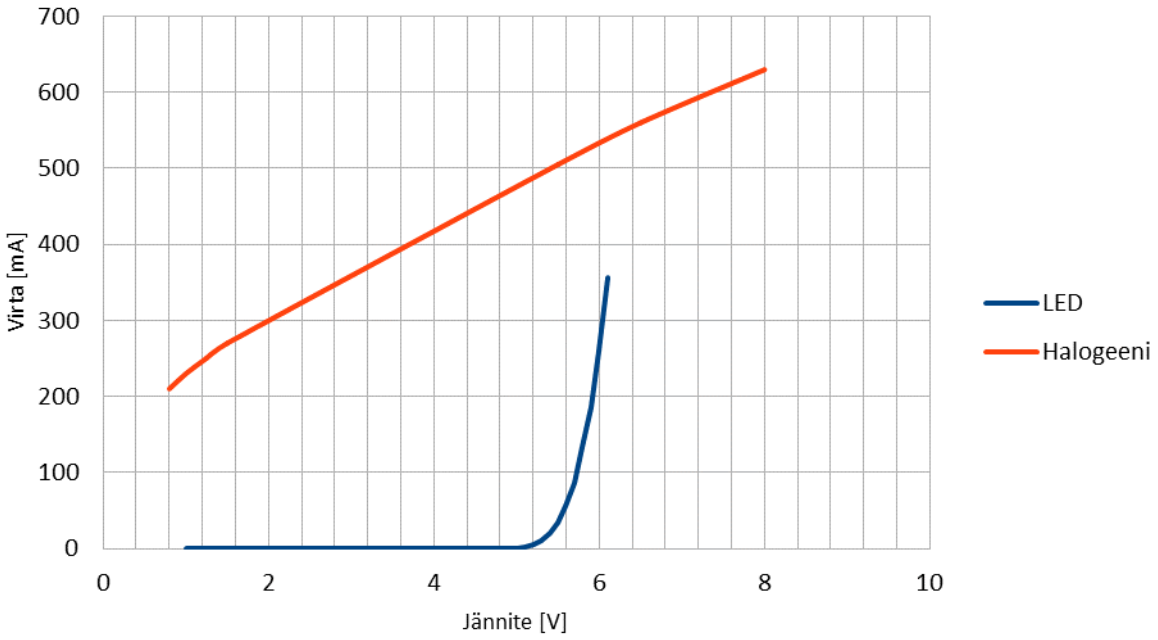
3.4 Led-valojen ohjaus

Led (light-emitting diode) on puolijohdediodi, joka säteilee valoa, kun sen läpi kulkee virta.

Ledit rinnastetaan usein virheellisesti hehkulamppuun. Vaikka molemmat tuottavat valoa, niiden sähköiset ominaisuudet ovat täysin erilaiset. Kytkennässä hehkulamppu vastaa vastusta, jonka resistanssi riippuu hehkulangan lämpötilasta. Vastusarvon muutos lampun loistaessa on kuitenkin suhteellisen pieni, joten jännitearvon pienet muutokset eivät suuresti vaikuta sen läpi kulkevaan virtaan.

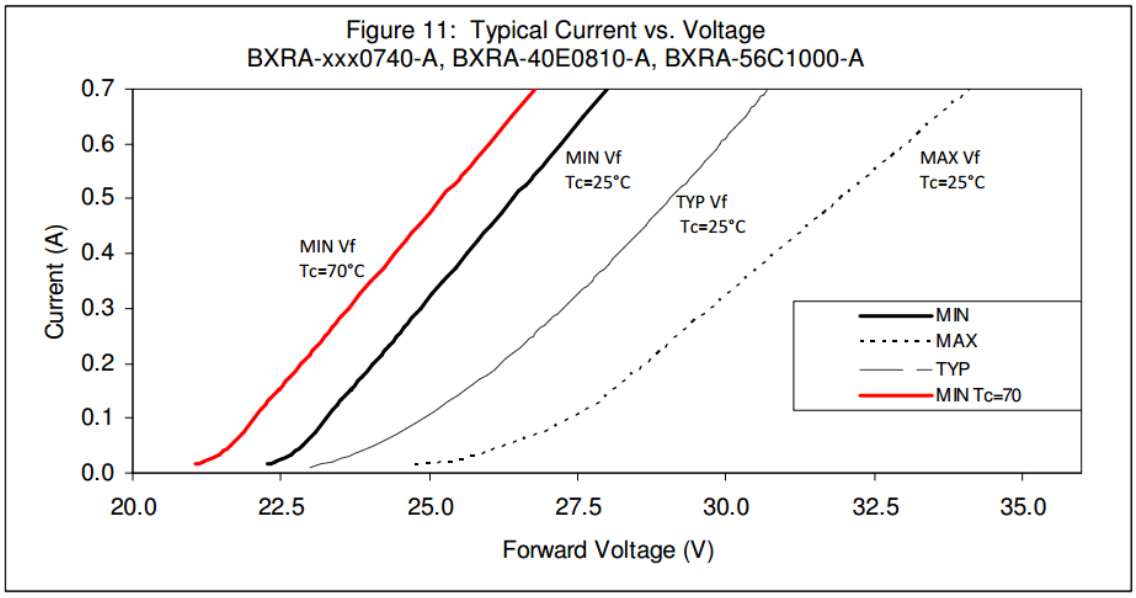
Ledi puolestaan vastaa sähköisiltä ominaisuuksiltaan diodia, jonka kynnysjännite riippuu ledissä käytettävästä puolijohdemateriaalista (Winder 2008, 2). Kun ledin kynnysjännite ylittyy, alkaa sen läpi kulkema virta kasvaa eksponentiaalisesti. Ledien kynnysjännite voi vaihdella huomattavasti eri ledien välillä, vaikka ne olisivatkin samantyyppisiä.

Erot tulevat esille mittaamalla ledin ja hehkulampun, tai tässä tapauksessa ledin ja halogeenipolttimon läpi kulkeva virta suhteessa jännitteeseen. Kuten diagrammista (kuva 11) nähdään, on halogeenipolttimon virrankasvu melko lineaarinen nimellisjännitteen läheisyydessä. Vastaavasti ledin läpi ei kulje virtaa ennen kynnysjännitteen ylittymistä, jonka jälkeen virta kasvaa voimakkaasti.



KUVA 11. Ledin ja halogeenipolttimon läpi kulkeva virta jännitteen suhteen

Kuvausvalojen ohjainlaitteen valoiksi valittujen BXRA-40E0810-A-ledien datalehden diagrammista nähdään sama ilmiö (kuva 12), kuin aiemmin todettiin.



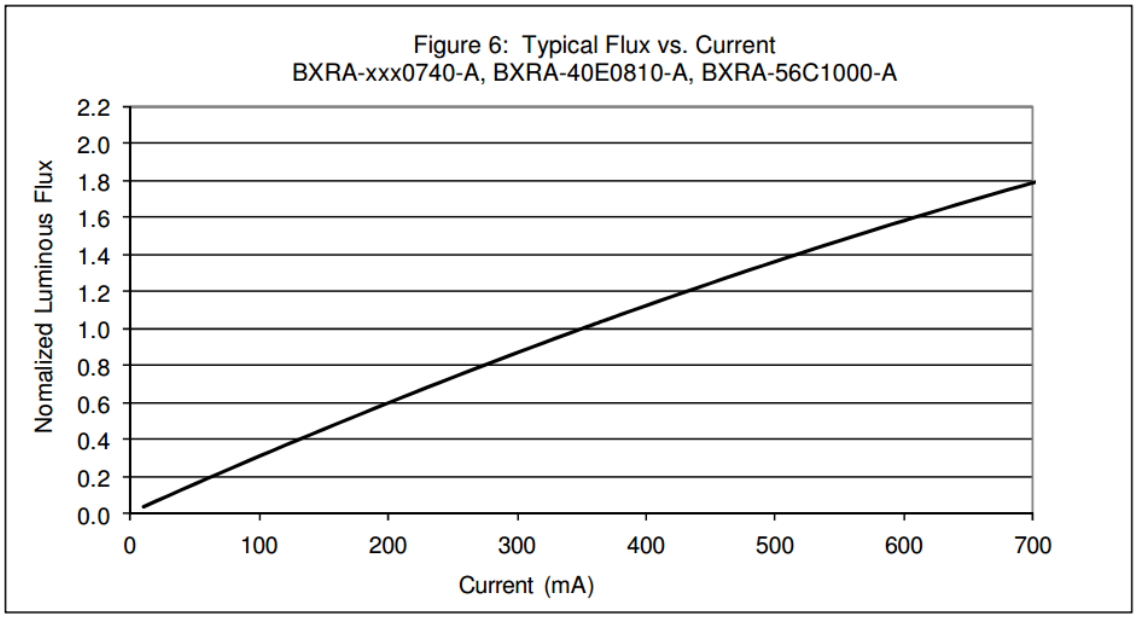
KUVA 12. BXRA-40E0810-A-ledin läpi kulkeva virta jännitteen suhteen (Bridgelux 2015)

Edellä mainitut ledin ominaisuudet tulivat esille ohjainlaitteen suunnittelun alkuvaiheilla. Ledien virta rajoitettiin etuvastuksella, joka on yleisesti käytössä oleva tapa. Tässä tapauksessa kiinteällä vastuksella virran rajoittaminen kuitenkin aiheuttaa ongelmia. Käytössä olevien ledien kynnysjännite datalehden mukaan on minimissään 25,3 V ja maksimissaan 30,9 V tyypillisen arvon ollessa 28,1 V. Kynnysjännitteen mittaukseen käytetty virta on 350 mA.

$$R_L = \frac{U_{in} - U_{vf}}{I_{Led}} \qquad (2)$$

Mikäli ledin etuvastus lasketaan (2) käyttämällä tyypillistä kynnysjännitettä, 350 mA:n virtaa sekä 32 voltin sisääntulojännitettä, saadaan vastusarvoksi 11,14 Ω. Lasketusta arvosta seuraava E12-sarjan vastus on 12 Ω, jolla ledin läpi kulkee 325 mA virtaa. Vastuksen tehohäviöksi tulee 1,27 W.

Vaikka tehohäviö on suuri, ei se ole suurin ongelma etuvastuksen käytössä. Suurin ongelma muodostuu ledin kynnysjännitteen vaihtelusta. Ledin kynnysjännitteen minimiarvolla ledin virta olisi 558 mA samalla 12 Ω:n etuvastuksella. Lähes 60 % suurempi virta aiheuttaa vastuksen tehohäviön kasvamista sekä ledin valontuoton lisääntymistä.



KUVA 13. Projektissa käytetyn ledin valontuotto virran suhteen (Bridgelux 2015)

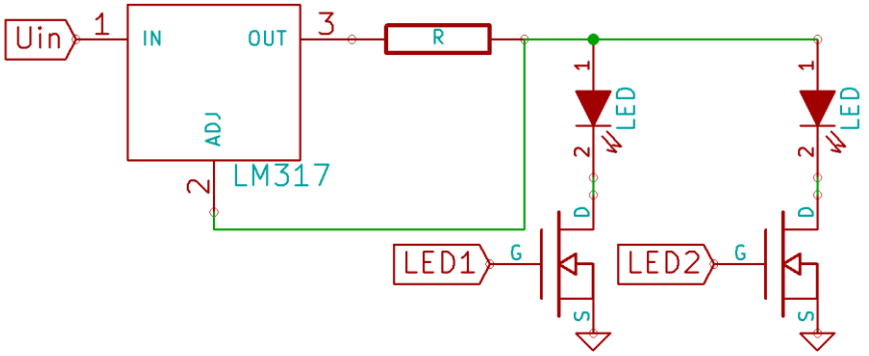
Ledin datalehdessä olevasta diagrammista nähdään (kuva 13), että ledin valontuotto on suoraan verrannollinen sen virtaan. Edellä laskettuja arvoja käyttäen voidaan todeta, että ledin valontuotto kasvaa myös 60 % käytettäessä ledin kynnysjännitteen minimiarvoa verrattuna tyypilliseen arvoon, joten ero on vieläkin suurempi minimi- ja maksimiarvojen välillä. Kynnysjännitteen vaihtelusta johtuen olisi viistovalokuvauksessa käytettyjen kuvausvalojen valontuoton erot toisiinsa nähden liian suuret.

Koska ledien virranrajoitus etuvastuksella ei toimi tässä tapauksessa, päädyttiin käyttämään toisenlaista ratkaisua. Järkevin tapa rajoittaa virtaa on vakiovirtalähteellä. Vakiovirtalähteen toteuttamiseen on erilaisia ratkaisuja kuten buck-, boost-, buck-boost- sekä flyback-topologioihin perustuvia hakkurivirtalähteitä. Tässä tapauksessa päädyttiin kuitenkin lineaariseen vakiovirtalähteeseen, joka on toteutettu regulaattorilla LM327.

Lineaarisen regulaattorin etuna on yksinkertaisuus, hinta sekä se, että regulaattorista ei säteile EMI-häiriöitä. Regulaattorin käytössä on kuitenkin myös huonoja puolia, kuten

että sen yli on jäätävä vähintään 3 V:n jännite. Koska regulaattori muuttaa ylimääräisen tehon lämmöksi, täytyy lämpö johtaa pois regulaattorista, jottei se ylikuumene.

Kytkennässä käytetään vain yhtä vakiovirtalähdettä, joka on kytkettynä jokaisen ledin anodille (kuva 14). Halutun ledin sytyttäminen tapahtuu kytkemällä ledin katodi maihin MOSFET-transistorin avulla. Koska kaikilla ledeillä on yhteinen virtalähde, on niiden läpi kulkeva virta sama, eikä ledien välillä tule eroa esimerkiksi komponenttien toleranssiseista johtuen. Kyseisellä kytkennällä voi kuitenkin olla vain yksi ledi päällä kerrallaan, mutta suunniteltavassa laitteessa ei tule sellaista tilannetta, jossa useamman ledin täytyisi loistaa samaan aikaan.



KUVA 14. LM317-regulaattorilla toteutettu vakiovirtalähde

Koska regulaattori säätää ulostuloa siten, että sen OUT- ja ADJ-pinnien välillä on aina tyypillisesti 1,25 volttia, voidaan regulaattorista tehdä vakiovirtalähde asettamalla näiden pinnien väliin vastus, jonka resistanssiksi saadaan 350 mA:n vakiovirralle 1,25 V / 300 mA = 3,6 Ω. Koska ADJ-pinnistä tuleva virta on vain 50 uA, voidaan se jättää huomiotta. Lähin E12-sarjan arvo on 3,3 ohmia, jolla virta asettuu arvoon 378 mA. Regulaattorin ja vastuksen hukkatehoksi laskettiin 1,4 W (3) tyypillisellä ledin kynnysjännitteellä, kun syöttöjännite on 32 V. Suuresta hukkatehosta johtuen regulaattoriin lisättiin jäähdytyselementti.

$$P = (U_{in} - U_{vf}) \cdot I \tag{3}$$

3.5 Elektroniikan suojaus ja häiriönpoisto

Digitaalisissa piireissä syntyy syöttöjännitteeseen jännitepiikkejä, kun mikropiirien tulo- ja lähtösignaalit muuttavat tilaansa. Tästä syystä mikropiirien syöttöjännitenastoihin on suositeltavaa kytkeä keraamiset 100 nF:n kondensaattorit.

Kytkennässä käytettyjen siirtorekistereiden jännitteensyöttöön kytkettiin suositellut 100 nF:n kondensaattorit mahdollisimman lähelle mikropiirin syöttöjännitenastoja.

Arduino Nano -kehitysalustassa mikrokontrollerin häiriönpoistokondensaattoreina käytetään rinnakkain kytkettynä 10 uF:n ja 100 nF:n kondensaattoreita. Suuremmalla kapasitanssilla vaimennetaan jännitepiikkejä, mikäli kaikki 23 kappaletta I/O-linjaa vaihtavat tilaansa samaan aikaan.

Induktiiviset kuormat, kuten moottorit, tuottavat jännitepiikin, kun niiden syöttöjännite katkaistaan. Jännitepiikki syntyy kelojen romahtavasta magneettikentästä, ja se voi aiheuttaa esimerkiksi releen kontaktien välille syntyvän valokaaren. (Sealevel, 2010.)

Koska viistovalokuvauslaitteessa käytetyt led-valot on kytketty ohjaimeen pitkillä kaapeleilla, aiheuttaa kaapelien induktanssi samanlaisen jännitepiikin. Tästä syystä kaikkien liittimien rinnalle kytkettiin estosuuntainen diodi. Tästä diodista käytetään nimitystä Flyback-diodi, ja sen tarkoitus on estää negatiivisen jännitteen muodostumista. Koska diodin juotostäplät puuttuivat valmistetusta piirilevystä, juotettiin se suoraan liittimen pinneihin.

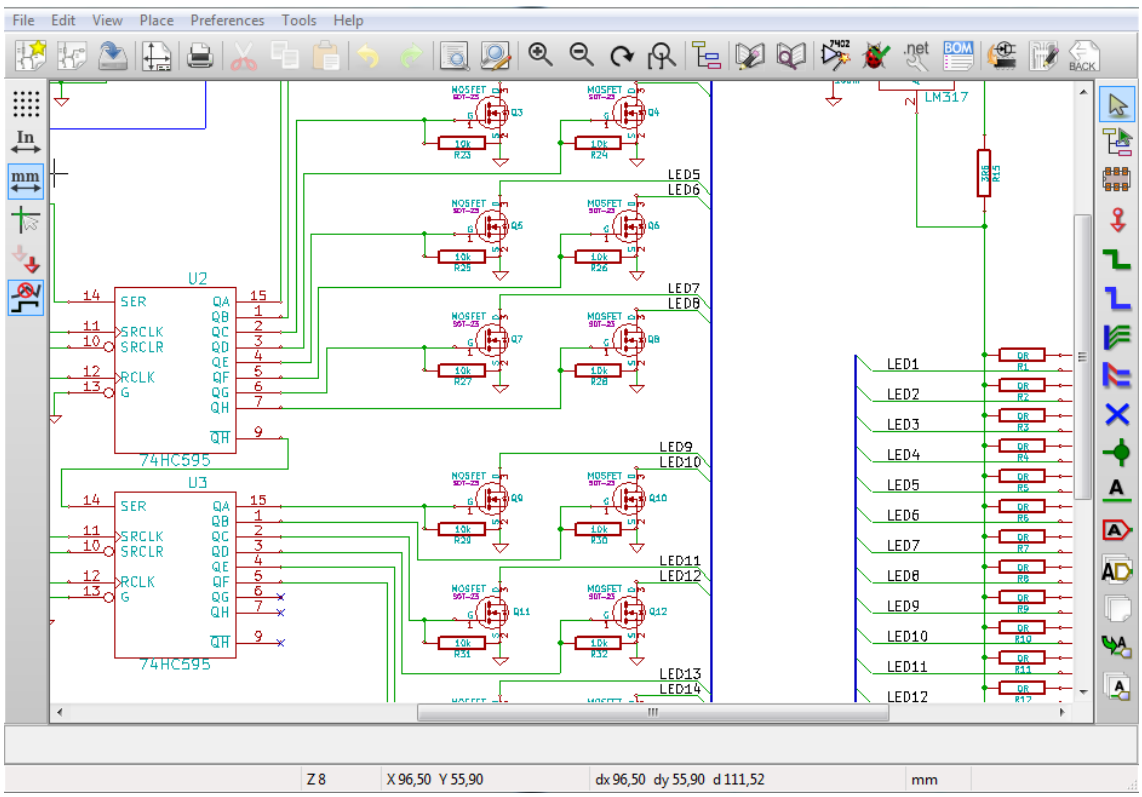
4 KÄYTETYT OHJELMISTOT

Projektia varten tarvittiin työkalut piirilevyn suunnitteluun, mikrokontrollerin ohjelmankehitykseen sekä ohjainlaitteen testaukseen. Aiemmin tutuksi tullutta Altium Designer -ohjelmistoa olisi voitu käyttää piirilevyn suunnittelussa, mutta lisenssin umpeuduttua hyödynnettiin mahdollisuutta tutustua avoimeen lähdekoodiin perustuvaan KiCad-ohjelmistoon. Mikrokontrollerin ohjelmankehitykseen valittiin ennestään tuttu Atmel Studio -ympäristö. Ohjainlaitteen testaukseen ohjelmoitiin työkalu, jotta laitteen perustoiminnot on mahdollista testata ilman varsinaista viistovalokuvauslaitteen käyttöliittymää.

4.1 KiCad

KiCad on avoimeen lähdekoodiin perustuva piirilevyjen suunnitteluohjelmisto. KiCad on saavuttanut suuren suosion harrastelijoiden keskuudessa sen matalan opettelukynnyksen ja erilaisten oppaiden ansiosta. Ohjelmiston työkalujen avulla pystyy suunnittelemaan myös monimutkaisia piirilevyjä, ja esimerkiksi CERN (European Organization for Nuclear Research) on käyttänyt ohjelmaa piirilevyjen suunnitteluun sekä lisännyt ohjelmaan ominaisuuksia, jotka auttavat piirilevyjen suunnittelussa. (KiCad 2015.)

KiCad:llä piirilevyä suunniteltaessa aloitetaan kytkentäkaavion piirtämisellä eeschema-nimisellä työkalulla (kuva 15). Eeschema on CAD-tyyppinen ohjelma (Computer-aided Design), jolla piirretään haluttu kytkentä. Ohjelma sisältää valmiina useita eri komponentteja, ja siihen on helppo lisätä itse tehtyjä tai muiden julkaisemia komponentteja. Tässä projektissa käytetyt komponentit löytyivät valmiiksi ohjelman sisältämästä kirjastosta Arduino Nanoa sekä paria liitintä lukuun ottamatta.



KUVA 15. KiCad-ohjelmiston eeschema-työkalu

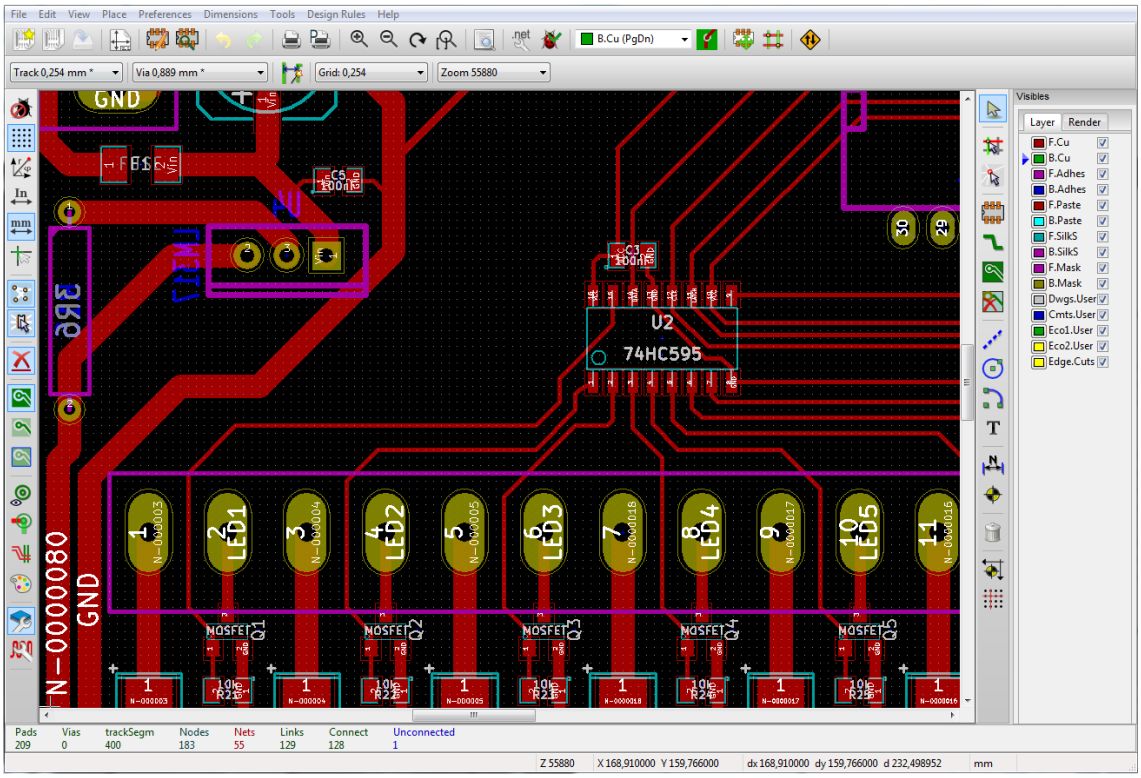
Kun kytkentäkaavio on saatu valmiiksi, täytyy siinä käytettyjen komponenttien kotelotyypit määritellä. Määrittely tehdään CvPcb-työkalulla, jolla annetaan jokaiselle komponentille haluttu kotelotyyppi, jotta piirilevyä suunniteltaessa ohjelma osaa luoda tarvittavat juotospisteet eli juotostäpläkuviot (footprint) piirilevylle.

Tässä projektissa käytetyt komponentit käyttävät yleisesti käytössä olevia kotelotyyppejä, joten ne sisältyvät ohjelmiston kirjastoon. Myöhemmin suunnittelussa liittimien pinnijako kuitenkin muutettiin suuremmaksi, joten sitä varten täytyi luoda vastaavat juotostäpläkuviot.

Piirilevyn suunnittelu tehdään Pcbnew-työkalulla (kuva 16). Suunnittelu aloitetaan siirtämällä kytkentäkaaviossa käytettyjen komponenttien juotostäpläkuviot piirilevylle haluttuihin kohtiin. Tämän jälkeen komponentit yhdistetään kuparivedoilla toisiinsa käyttäen sopivan levyisiä vetoja.

Vedon kuparin leveyden ja paksuuden määrittelyyn KiCad:ssa on Pcb calculator -niminen työkalu. Vedon leveys määritellään mm. sen läpi kulkevan virran perusteella. Tässä

projektissa käytettyjen vetojen leveydet ovat signaaleille 10 mils ja vedoille, joissa kulkee suurempia virtoja, 60 mils.



KUVA 16. KiCad-ohjelmiston Pcbnew-työkalu

Päinvastoin kuin esimerkiksi Altium Designer –piirilevysuunnitteluohjelmassa, KiCad ei sisällä autorouting-ominaisuutta. Tämän ominaisuuden avulla on mahdollista antaa tietokoneen itse laskea optimaaliset vedot halutuille yhteyksille. KiCad on aiemmin sisältänyt tämän ominaisuuden nimeltä FreeRoute, mutta lakiteknisistä syistä se on poistettu.

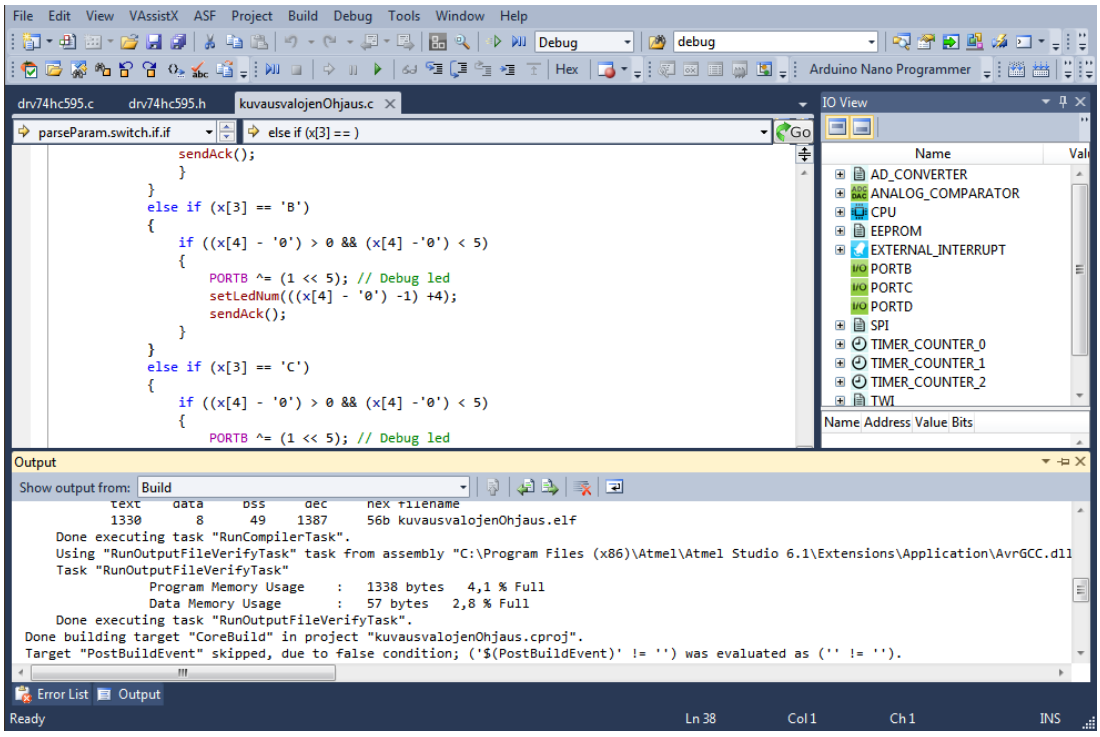
FreeRoute on kuitenkin julkaistu avoimena lähdekoodina erillisenä java-ohjelmana ja on saatavilla osoitteesta https://github.com/nikropht/FreeRouting.

Koska piirilevy suunniteltiin yksipuoleiseksi, on vetojen mahduttaminen yhdelle kerrokselle työlästä. Apuna käytettiin edellä mainittua FreeRoute-työkalua. Ensiksi käsin asetettiin vedot ledien liittimille, jonka jälkeen annettiin FreeRoute-työkalun laskea lopuille vedoille reitit. Laskemisen valmistuttua, tarkistettiin ja muokattiin vedot vielä käsin.

4.2 Atmel Studio

Mikrokontrollerin ohjelman kehitykseen käytettiin jo ennestään tuttua kehitysympäristöä Atmel Studio 6.1 (kuva 17). Kyseinen ohjelmisto sisältää työkalut Atmel ARM, Cortex sekä AVR-mikrokontrollereiden ohjelman kehitykseen ja virheenkorjaukseen (Debugging) C/C++- tai assembly-kielellä. (Atmel 2015.)

Ohjelma kirjoitettiin C-kielellä ja käännettiin käyttämällä Atmel Studion sisältämää GCC-kääntäjää (versio 3.4.2). Atmel Studio sisältää mahdollisuuden käyttää ulkopuolisia työkaluja ja suorittaa niitä työkalurivin painikkeilla. Tätä ominaisuutta hyödynnettiin tekemällä komento ja sitä vastaava painike nimellä ”Arduino Nano Programmer” (kuva 17), jolla käynnistetään mikrokontrollerin ohjelmoinnissa käytettävä AVRDUDE-työkalu. Painikkeen avulla on siis mahdollista ladata käännetty ohjelma suoraan mikrokontrollerille Atmel Studion sisältä, eikä työkalua tarvitse käynnistää erikseen komentoriviltä.



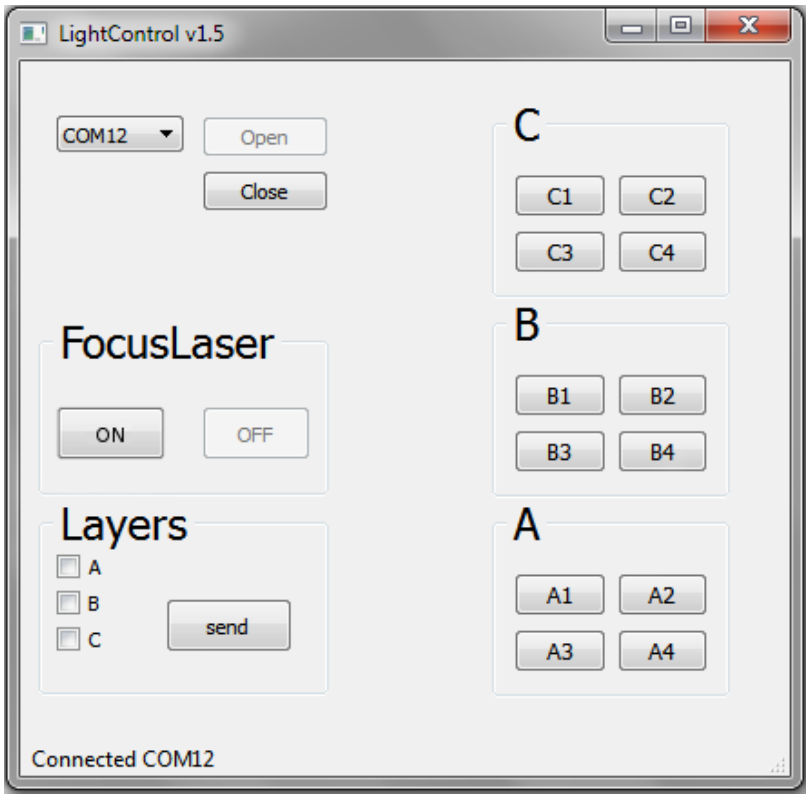
KUVA 17. Atmel Studio -kehitysympäristö

4.3 Kuvausvalojen ohjainlaitteen testaustyökalu

Jotta kuvausvalojen ohjainlaitetta olisi mahdollisuus testata käytössä ilman varsinaista viistovalokuvauslaitteen käyttöliittymää, ohjelmoitiin tähän tarkoitukseen erillinen testaustyökalu (liite 1).

Työkalu ohjelmoitiin C++-kielellä käyttämällä Qt-kehitysympäristöä, joka on alustariippumaton ohjelmistojen ja erilaisten graafisten käyttöliittymien ohjelmointiin tarkoitettu kehitysympäristö (Qt Project).

Testaustyökalun käyttö aloitetaan valitsemalla sarjaportti, johon ohjainlaite on kytketty. Mikäli sarjaporttia ei ole avattu, ovat kaikki painikkeet harmaana, eikä niitä voi käyttää. Kun oikea portti on avattu, voidaan napeilla lähettää ohjainlaitteelle ennalta määrätyt käskyt laitteen ohjaamiseen (kuva 18). Painikkeet A1-C4 sytyttävät kyseisen ledin muutamaksi sekunniksi kerrallaan. ”Layers”-kohdan valintaruuduilla ja ”send”-painikkeella konfiguroidaan ohjainlaite käyttämään valittuja tasoja, ja vaihtamaan seuraavaan lediin kameralta saadun synkronointisignaalin avulla.



KUVA 18. Kuvausvalojen ohjainlaitteen testaustyökalu

5 LAITEOHJELMISTO

Tässä kappaleessa kerrotaan kuvausvalojen ohjainlaitteen mikrokontrollerin laiteohjelmiston (firmware) toimintaperiaate pääpiirteittäin. Ohjelma on kommentoituna kokonaisuudessaan liitteenä. Tästä ohjelman versiosta puuttuu vielä laser-valojen ohjaukseen käytettävä funktio, joka sisältyy lopulliseen versioon.

Ohjelman toiminta perustuu pääosilta keskeytysvektorien käyttöön. Mikrokontrollerin USART-linjalle USB-UART-muuntimelta saapuvat merkit aiheuttavat keskeytyksen, jonka keskeytyspalvelussa linjalta saapunut merkki lisätään rengaspuskuriin (kuva 19). Keskeytyspalvelussa huolehditaan myös rengaspuskurissa käytettävien luku- ja kirjoitusmuuttujien nollauksesta puskurin täytyttyä.

```
105 ISR(USART_RX_vect)                          // USART receive interrupt vector
106 {
107     rxBuffer[rxWritePos++] = UDR0;          // Add received byte to buffer and move position to next
108     if (rxWritePos >= RX_BUFFER_SIZE)       // Reset write index to zero if buffer size is full
109     {
110         rxWritePos = 0;
111     }
112 }
```

KUVA 19. USART-keskeytyspalvelu

Pääohjelmasilmukassa (kuva 20) ajetaan ikuista silmukkaa, jonka sisällä luetaan saapunut merkki temp-muuttujaan rengaspuskurista. Mikäli saapunut merkki on konfigurointikäskyn aloitusmerkki ”$”, luetaan seuraavat kuusi merkkiä command-taulukkoon.

```
80      while(1)
81      {
82      while ((temp = getChar()) != '!')           // Do this while the received character
83          {                                       // is not "command end" character '!'
84
85              if (temp == '$')                    // Reset the command index counter,
86              {                                   //  if "command start" '$' is received
87                  commandIndex = 0;
88              }
89
90              if (temp != '\0')                   // Write character to command array
91              {                                   // only when character is received (not /0)
92                  command[commandIndex++] = temp;
93                  if (commandIndex > 6)           // Commands need to be exactly 6 characters long
94                  {
95                      commandIndex = 0;
96                  }
97              }
98
99          }
100         parseParam(command);
101     }
```

KUVA 20. Pääohjelmasilmukka

Kun kaikki kuusi merkkiä on luettu command-taulukkoon, suoritetaan parseParam-funktio, jossa käydään taulukon sisältö läpi merkki kerrallaan käyttäen switch-case-valintarakennetta (kuva 21). Tässä tarkistetaan myös saapuneen käskyn oikeellisuus. Mikäli command-taulukossa on validi käsky, suoritetaan sen tarvitsemat toimenpiteet.

```
171  void parseParam(char x[])                    // Makes actions according
172 {                                             //  to received command
173     switch (x[1])
174     {
175         case 'L':
176         if(x[2] == 'S')
177         {
178             if (x[3] == 'A')
179             {
180                 if ((x[4] - '0') > 0 && (x[4] -'0') < 5)
181                 {
182                 PORTB ^= (1 << 5); // Debug led
183                 setLedNum((x[4] - '0') -1);
184                 sendAck();
185                 }
186             }
187             else if (x[3] == 'B')
188             {
```

KUVA 21. Osa parseParam-funktion switch-case-valintarakenteesta.

Mikäli vastaanotetulla konfiguraatiokäskyllä asetetaan esimerkiksi led A1 päälle ($LSA1), kutsuu parseParam-funktio setLedNum-funktiota (kuva 22), sekä välittää sille parametrina kyseisen ledin portin numeron.

```
275  void setLedNum(int x)                        // Set led 0-14,
276 {                                             // clear all LEDs with parameter 17
277     TCNT1 = 15625 * LEDTIMER;                 // Set timer for clearing led
278     ledCtrl =  (1 << x);
279     write_hc595(ledCtrl);                     // send ledCtrl array to shift registers
280 }
```

KUVA 22. Ledin ohjaukseen käytetty funktio

Funktiossa setLedNum asetetaan myös ajastin ennen ledin sytyttämistä. Ajastimen tarkoituksena on estää ledin liian pitkä päälläoloaika, josta voisi seurata ledin ja vakiovirtalähteen liiallista lämpeämistä.

Ledin ohjaamiseen käytetään hyväksi write_hc595-funktiota (kuva 23), jolla ohjataan mikrokontrolleriin liitettyjä siirtorekistereitä. Funktiolle annetaan parametrina 16-bittinen

muuttuja, jonka jälkeen jokaisen muuttujassa asetetun bitin indeksinumeroa vastaava siirtorekisterin ulostulo asetetaan päälle. Funktio toimii 74HC595-siirtorekisterin datalehdessä selitetyn ohjauksen mukaan. Funktiolle parametrina annetusta muuttujasta huomioidaan vain haluttu bitti, eli maskataan, tekemällä OR-funktio parametrin ja maski-muuttujan kanssa. Maski-muuttuja sisältää asetetun bitin indeksissä, joka halutaan lukea data-muuttujasta. Mikäli OR-funktion tuloksena on TOSI, asetetaan datapinni ylätilaan, jonka jälkeen annetaan pulssi siirtorekisterin SHCP-sisääntuloon. Vastaavasti OR-funktion tuloksen ollessa EPÄTOSI, data-pinni asetetaan alatilaan ja SHCP-sisääntuloon annetaan pulssi. Tämän jälkeen maski-muuttujan asetettua bittiä siirretään oikealle yhden indeksinumeron verran, ja sama data-muuttujan luku suoritetaan. Kun kaikki 16-bittiä on luettu, annetaan siirtorekisterin STCP-sisääntuloon pulssi, joka siirtää siirtorekisterin rekistereissä olevat bitit ulostuloon.

```
26  void write_hc595(const uint16_t data)
27  {
28      uint16_t maski = 0b1000000000000000;
29         for (int i = 0; i<16; i++)
30         {
31             if(data & maski)
32             {
33                 hc595_port |= (1 << hc595_dataPin);
34             }
35             else
36             {
37                 hc595_port &= ~(1 << hc595_dataPin);
38             }
39             hc595_port |= (1 << hc595_clockPin);
40             hc595_port &= ~( 1 << hc595_clockPin);
41             maski = (maski >> 1);
42
43         }
44         hc595_port |= ( 1 << hc595_latchPin);
45         hc595_port &= ~( 1 << hc595_latchPin);
46
47         hc595_port &= ~(1 << hc595_dataPin);
48
49  }
```

KUVA 23. Siirtorekistereiden ohjaukseen käytetty funktio

Mikäli ohjainlaitteelle annettu käsky on kuvauksessa käytettävien led-valoryhmien eli tasojen määrittely, tunnistetaan se myös parseParam-funktiossa. Käsky, jolla asetetaan käyttöön tasot A, B ja C, on ”$SSAC!”. Käsky tulkitaan parseParam-funktion sisältämässä switch-case-rakenteessa (kuva 24). Rakenteen avulla ledSeq-muuttujasta asetetaan

bitti, mikäli bitin indeksinumero vastaa komennon määrittelemän ledin numeroa. Esimerkiksi A-kerrosta vastaa ledSeq-muuttujan arvo ”0b000000001111”.

```
208         case 'S':
209         if(x[2] == 'S')
210         {
211             ledSeq = 0;
212             ledSeqMask = 1;
213             if (x[3] == 'A')
214             {
215                 ledSeq |= 0b000000001111;
216                 if (x[4] == 'C')
217                 {
218                     ledSeq |= 0b111111111111;
219                 }
220             }
221 
222             else if (x[3] == 'B')
223             {
```

KUVA 24. Tason määrittelyssä käytetyn käskyn tulkitseminen

Kun ledSeq-muuttuja on määritelty, voidaan käskyn määrittelemiä valoja ohjata päälle ja pois mikrokontrollerin pinnillä INT0, johon tässä tapauksessa on kytketty kameran quench-signaali. Keskeytys on määritelty siten, että INT0-pinnissä tapahtuva loogisen tilan muutos aiheuttaa keskeytyksen. Keskeytyspalvelussa tunnistetaan if-else-rakenteella, miten päin signaalin muutos on tapahtunut (kuva 25). Mikäli keskeytys tapahtui signaalin muutoksesta alatilasta ylätilaan, sytytetään sekvenssissä seuraavaksi määritelty ledi. Mikäli muutos tapahtui päinvastoin, sammutetaan ledi.

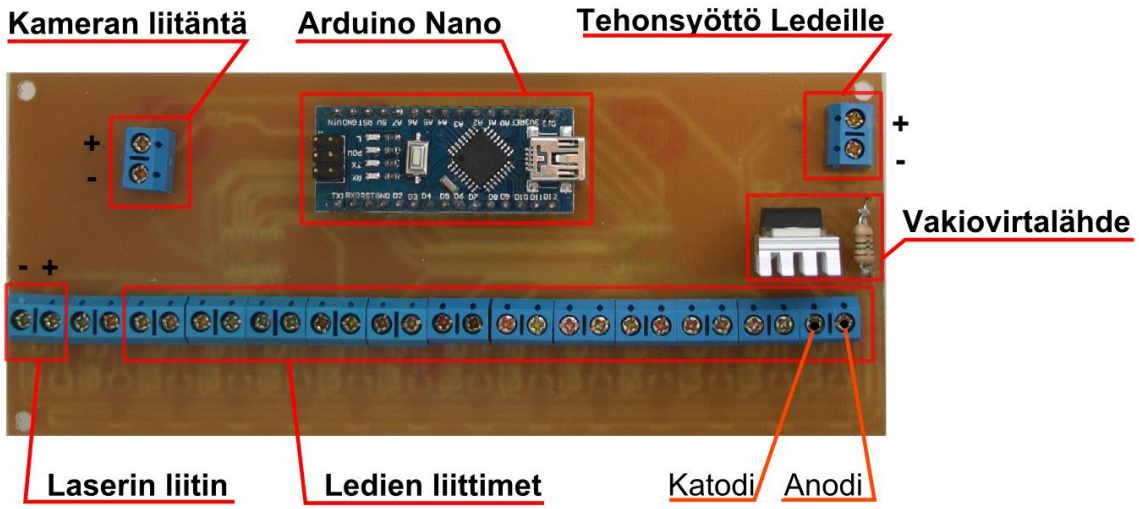
```
123 ISR(INT0_vect)                          // Camera sync pin interrupt
124 {
125     if (PIND & (1 << 2))                // If interrupt came from low to high transition,
126     {                                   // light led next in sequence
127         if ((ledSeq & 0b111111111111))
128         {
129             while(!((ledSeq & 0b111111111111) & ledSeqMask))
130             {
131                 ledSeqMask = ledSeqMask << 1;
132                 if (ledSeqMask >= 0b1000000000000 )
133                 {
134                     ledSeqMask = 1;
135                 }
136             }
137             ledCtrl = ledSeqMask;
138             setLedArray(ledCtrl);
139             ledSeqMask = ledSeqMask << 1;
140         }
141     }
142     else                                // If interrupt came from high to low transition,
143     {                                   // clear LEDs
144         setLedNum(17);
145     }
146 }
```

KUVA 25. Valosekvenssin ohjaamiseen käytetty keskeytyspalvelu

6 LAITTEEN KÄYTTÖ

Kuvausvalojen ohjainlaitteen asennus aloitetaan kytkemällä led-valot niille tarkoitettuihin liittimiin. Laitteessa on 12 kappaletta ledeille tarkoitettuja liittimiä, ledit kytketään kuvasta (kuva 26) katsoen oikealta, järjestyksessä A1, A2, A3, A4, B1, B2, … Tämän jälkeen kytketään laser-valo, huomioiden polariteetti. Laser-liittimen positiivinen pinni on kytketty Arduino Nano -levyn viiden voltin ulostuloon, joten se on suoraan yhteydessä USB-liittimen syöttämään jännitteeseen. Tästä johtuen pinniä ei saa kuormittaa liikaa.

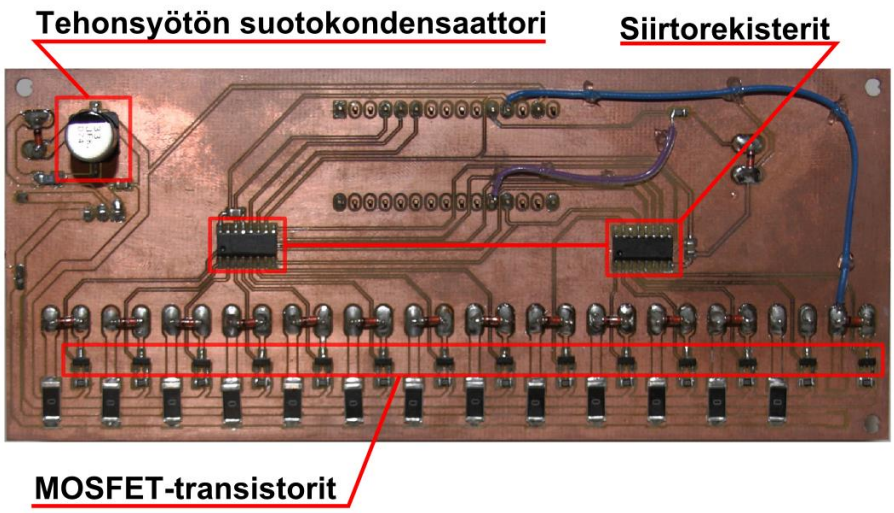
Kameralta tuleva synkronointisignaali kytketään sitä vastaavaan liittimeen, huomioiden polariteetti (kuva 26). Signaalin täytyy olla TTL-tasoista, eikä siinä saa esiintyä kytkinvärähtelyä.



KUVA 26. Kuvausvalojen ohjainlaitteen osat

Seuraavaksi kytketään led-valojen tehonsyöttö, huomioiden kuvassa 26 esitetty polariteetti. Syöttöjännitteen on oltava vähintään kolme volttia ledien kynnysjännitettä suurempi, kuitenkin maksimissaan 35 V. Suuremman jännitteen käyttäminen aiheuttaa turhaa tehohäviötä vakiovirtalähteessä eikä ole suositeltavaa. Viimeiseksi kytketään Arduino Nano USB-johdolla tietokoneeseen.

Piirilevyn alapuolella ovat siirtorekisterit, MOSFET-transistorit sekä muut oheiskomponentit (kuva 27).



KUVA 27. Kuvausvalojen ohjainlaitteen alapuolen osat

Mikäli laitteeseen liitetty tietokone ei tunnista laitetta virtuaalisena sarjaporttina, on siihen asennettava ajurit CH340 USB-UART -muunninta varten. Kirjoitushetkellä ajurit Windows-käyttöjärjestelmälle olivat saatavilla osoitteesta http://www.wch-ic.com.

Ajureiden asennuksen jälkeen laite näkyy käyttöjärjestelmälle virtuaalisena sarjaporttina. Sarjaporttiin voi lähettää käskyjä terminaaliohjelmalla käyttäen asetuksina 9600 baudia, 8 data bittiä, yksi loppubitti sekä nolla pariteettibittiä (9600 8-N-1). Kun terminaaliyhteys on avattu, voidaan yhteys testata lähettämällä ohjainlaitteelle jokin taulukossa 3 esitetyistä komennoista, joihin ohjainlaite vastaa ”ACK”-merkkijonolla.

USB-liityntä on suoraan yhteydessä Arduino Nano -kehitysalustan USB-porttiin, joten sitä kautta on mahdollista ladata uusi laiteohjelmisto mikrokontrollerille. Lataukseen voi käyttää Arduinon omaa kehitysympäristöä tai tässä työssä käytettyä AVRDUDE-työkalua.

Jokainen käsky on aloitettava ”$”-merkillä ja lopetettava ”!”-merkillä, käskyn pituus on tasan 6 merkkiä. Laitteen tunnistamat käskyt on esitelty taulukossa 3. Jokaiseen validiin käskyyn laite vastaa lähettämällä merkkijonon ”ACK”.

TAULUKKO 3. Ohjainlaitteen käskyt

Käsky	Toiminto	Parametrit (xx)
$LSxx!	Asettaa valitun ledin päälle	Ledin kerros sekä numero. Kerros A,B,C ja ledin numero 1-4 (esim. A3)
$SSxx!	Asettaa sekvenssissä käytettävät kerrokset	Aloituskerros sekä lopetuskerros joiden ledejä käytetään A-C. (esim. AB) Voidaan myös asettaa vain yksi kerros laittamalla molemmat parametrit samaksi.
$Fxxx!	Tarkennuksessa käytettävien lasereiden ohjaus	SET Asettaa kanavassa olevan laserin päälle OFF Sammuttaa laserin
$MODx!	Kameralta tulevan signaalin asetus	X Asettaa laitteen toimimaan kameran X-signaalin mukaan. Q Asettaa laitteen toimimaan kameran quench-signaalin mukaan (Fyysinen signaalin vaihto tapahtuu kytkimellä laitteen sisältä).

Kun laiteelle on asetettu käytettävät valaisukerrokset, syttyy ledi kameran liitäntään annetun pulssin nousureunalla ja vastaavasti sammuu pulssin laskureunalla.

Kuvauslaitteeseen asennettujen laservalojen asettaminen päälle tai pois tapahtuu taulukon 3 mukaisilla käskyillä. Päinvastoin kuin kuvausvaloissa, laservalojen päälläoloaikaa ei ole rajoitettu ohjelmallisesti, joten ne on kytkettävä pois päältä sitä vastaavalla komennolla.

Kameralta tuleva signaali, jolla valaisusekvenssissä edetään, voi olla joko Quench- tai X-signaali. Signaalin vaihtaminen tapahtuu asettamalla laitekotelon sisällä oleva kytkin joko X- tai Q-asentoon vastaamaan haluttua signaalia. Tämän jälkeen laitteelle annetaan taulukon 3 mukainen signaalin asetuskäsky, jotta laite osaa toimia signaalin mukaisesti.

7 YHTEENVETO

Kuvausvalojen ohjainlaite suunniteltiin alun perin tehtäväksi kahdessa osassa. Ensiksi prototyyppi, jonka jälkeen laitteen testauksessa mahdollisten vikojen korjausten jälkeen piirilevystä tehtäisiin toinen versio. Koska prototyyppivaiheen valmiiksi saattamisessa kului aikaa muun muassa osien hitaan toimituksen johdosta, ei piirilevystä tehty toista versiota. Ensimmäinen versio on kuitenkin alustavien testien mukaan toimiva sekä täyttää suunnittelun alussa asetetut vaatimukset, joten sitä on mahdollista käyttää varsinaisena ohjainlaitteena.

Ohjainlaitteen toiminta viistovalokuvauslaitteiston kanssa on testattu kameraa manuaalisesti ohjattuna. Koska viistovalokuvauslaitteiston varsinainen käyttöliittymä puuttuu, ei laitetta ole mahdollista testata käytännön kuvaustilanteissa. Ohjainlaite on suunniteltu siten, että sen laiteohjelmistoa on helppo päivittää jälkikäteen, joten mahdollisten käyttöliittymän aiheuttamien ongelmien korjaus on mahdollista.

LÄHTEET

Arduino. Arduino Nano. Luettu 16.3.2015.
http://arduino.cc/en/Main/ArduinoBoardNano

Atmel. Atmel Studio. Luettu 10.3.2015.
http://www.atmel.com/tools/atmelstudio.aspx

Atmel. ATmega328p. Luettu 10.1.2015.
http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

Bridgelux. Bridgelux ES Star Array Series. Luettu 10.1.2015
http://www.bridgelux.com/wp-content/uploads/2013/10/DS23-Bridgelux-ES-Star-Array-Data-Sheet1.pdf

Digital Photography School. 10 Reasons to Turn off Your Autofocus. Luettu 23.3.2015.
http://digital-photography-school.com/10-reasons-to-turn-off-your-autofocus/

Innventia. OptiTopo - Measuring the surface of paper and board. Luettu 23.3.2015.
http://www.innventia.com/Documents/Produktblad/Material%20processes/Pappersyta/OptiTopo%20-%20product%20sheet.pdf

KiCad EDA Software Suite. Luettu 16.3.2015.
http://www.kicad-pcb.org/

NXP. 74HC595. Luettu 10.1.2015.
http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

Sealevel. How To Switch Highly Inductive Loads Using Digital I/O. Luettu 23.3.2015
http://www.sealevel.com/support/article/AA-00470/0/How-to-Switch-Highly-Inductive-Loads-Using-Digital-I-O.html

STMicroelectronics. LM317. Luettu 10.1.2015.
http://www.st.com/web/en/resource/technical/document/datasheet/CD00000455.pdf

Trusted Reviews. How the LG G3 laser AF camera focus works. Luettu 23.3.2015.
http://www.trustedreviews.com/opinions/how-the-lg-g3-laser-af-camera-focus-works

Vishay. si2318cds. Luettu 10.1.2015.
http://www.vishay.com/docs/67030/si2318cd.pdf

Volotinen, V., Lesch, K-B & Haaksikari, J. 1994. Elektroniikka 1: Analoginen elektroniikka. Porvoo: WSOY.

Volotinen, V. 1993. Elektroniikka 2: Digitaalitekniikka. Porvoo: WSOY.

Winder, S. 2008. Power supplies for LED driving. Oxford: Newnes.

LIITTEET

Liite 1. Testausohjelma

Main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Main.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSerialPort/QtSerialPort>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_openBtn_clicked();
    void on_closeBtn_clicked();

    void sendData(QString);
    void readData();

    void on_a1Btn_clicked();
    void on_a2Btn_clicked();
    void on_a3Btn_clicked();
    void on_a4Btn_clicked();

    void on_b1Btn_clicked();
    void on_b2Btn_clicked();
    void on_b3Btn_clicked();
    void on_b4Btn_clicked();

    void on_c1Btn_clicked();
    void on_c2Btn_clicked();
    void on_c3Btn_clicked();
    void on_c4Btn_clicked();

    void on_sendButton_clicked();

    void on_laserOn_clicked();
    void on_laserOff_clicked();

private:
    Ui::MainWindow *ui;
    QSerialPort *serial;
};

#endif // MAINWINDOW_H
```

MainWindow.cpp

2(5)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtSerialPort/QtSerialPort>
#include <QDebug>

const QString commands[30] = {"$LSA1!","$LSA2!","$LSA3!","$LSA4!",
                              "$LSB1!","$LSB2!","$LSB3!","$LSB4!",
                              "$LSC1!","$LSC2!","$LSC3!","$LSC4!",
                              "$SSAA!","$SSAB!","$SSAC!","$SSBB!",
                              "$SSBA!","$SSBC!","$SSCC!","$SSCB!",
                              "$SSCA!","$FSON!","$FOFF!"}; // Array of all of
the available commands

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    serial = new QSerialPort(this); // Make new serialport object
    connect(serial,SIGNAL(readyRead()),this, SLOT(readData())); // If serialport
has data, run readData()
    QList<QString> portit; // List of available ports
    foreach (const QSerialPortInfo &serialPortInfo,
QSerialPortInfo::availablePorts()) {

        qDebug() << "\nPort:" << serialPortInfo.portName();
        portit.append(serialPortInfo.portName()); // Add available ports to the
list
        qDebug() << "Location:" << serialPortInfo.systemLocation();
        qDebug() << "Description:" << serialPortInfo.description();
        qDebug() << "Manufacturer:" << serialPortInfo.manufacturer();
        qDebug() << "Vendor Identifier:" <<
(serialPortInfo.hasVendorIdentifier() ?
QByteArray::number(serialPortInfo.vendorIdentifier(), 16) : QByteArray());
        qDebug() << "Product Identifier:" <<
(serialPortInfo.hasProductIdentifier() ?
QByteArray::number(serialPortInfo.productIdentifier(), 16) : QByteArray());
        qDebug() << "Busy:" << (serialPortInfo.isBusy() ? QObject::tr("Yes") :
QObject::tr("No"));
    }
    ui->portBox->addItems(portit); // ComboBox for selecting the used serialport

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_openBtn_clicked() // Open selected serialport
{
    serial->setPortName(ui->portBox->currentText());
    if (!serial->isOpen())
    {
        bool noError;
        noError = serial->setBaudRate(QSerialPort::Baud9600);
        noError = serial->setDataBits(QSerialPort::Data8);
        noError = serial->setParity(QSerialPort::NoParity);
        noError = serial->setStopBits(QSerialPort::OneStop);
        noError = serial->setFlowControl(QSerialPort::SoftwareControl);
        if(serial->open(QIODevice::ReadWrite) & noError)
        {
            ui->openBtn->setDisabled(1);
            ui->closeBtn->setDisabled(0);    // Set controls available, if serial
port is open
            ui->groupA->setDisabled(0);
            ui->groupB->setDisabled(0);
```

```
            ui->groupC->setDisabled(0);
            ui->layerGroup->setDisabled(0);
            ui->laser->setDisabled(0);
            ui->statusBar->showMessage("Connected " + serial->portName());
        }
        else
        {
            qDebug() << "Can't init or open port"; // Error if can't open
        }

    }
}

void MainWindow::on_closeBtn_clicked() // Close and disable controls
{
    if(serial->isOpen())
    {
        serial->close();
        ui->openBtn->setDisabled(0);
        ui->closeBtn->setDisabled(1);
        ui->groupA->setDisabled(1);
        ui->groupB->setDisabled(1);
        ui->groupC->setDisabled(1);
        ui->layerGroup->setDisabled(1);
        ui->laser->setDisabled(1);
        ui->statusBar->showMessage("Port closed");
    }
}

void MainWindow::readData() // Read data every time there is new data available
{
        QByteArray data;
        while (serial->canReadLine())
        {
            data = serial->readLine();
            ui->statusBar->showMessage("Return: " + data.trimmed()); // Show
received data, mostly
        }                                                                  // "ACK" on
valid command sent
}

void MainWindow::sendData(QString data) // Send data to serialport
{
    if (serial->isOpen())
    {
        QByteArray dataConv = data.toLatin1();
        serial->write(dataConv);
    }
    ui->statusBar->showMessage("Error sending");
}

void MainWindow::on_a1Btn_clicked() // Send command to turn on led A1
{
    sendData(commands[0]);
}

void MainWindow::on_a2Btn_clicked() // Send command to turn on led A2
{
    sendData(commands[1]);
}

void MainWindow::on_a3Btn_clicked()
{
    sendData(commands[2]);
}

void MainWindow::on_a4Btn_clicked()
{
```

4(5)

```
    sendData(commands[3]);
}

void MainWindow::on_b1Btn_clicked()
{
    sendData(commands[4]);
}

void MainWindow::on_b2Btn_clicked()
{
    sendData(commands[5]);
}

void MainWindow::on_b3Btn_clicked()
{
    sendData(commands[6]);
}

void MainWindow::on_b4Btn_clicked()
{
    sendData(commands[7]);
}

void MainWindow::on_c1Btn_clicked()
{
    sendData(commands[8]);
}

void MainWindow::on_c2Btn_clicked()
{
    sendData(commands[9]);
}

void MainWindow::on_c3Btn_clicked()
{
    sendData(commands[10]);
}

void MainWindow::on_c4Btn_clicked()
{
    sendData(commands[11]);
}

void MainWindow::on_sendButton_clicked() // Send command to set used layers
{
    int x = 0;

    if(ui->layerA->isChecked())     // Convert selected checkboxes to
appropriate command
    {
        x = 1;
        if(ui->layerB->isChecked())
        {
            x = 2;
        }
        if(ui->layerC->isChecked())
        {
            x = 3;
        }
    }
    else if (ui->layerB->isChecked())
    {
        x = 4;
        if(ui->layerA->isChecked())
        {
            x = 5;
        }
        if(ui->layerC->isChecked())
        {
```

```
            x = 6;
        }
    }
    else if (ui->layerC->isChecked())
    {
        x = 7;
        if(ui->layerB->isChecked())
        {
            x = 8;
        }
        if(ui->layerA->isChecked())
        {
            x = 9;
        }
    }
    if(x != 0)
    {
     sendData(commands[11+x]);
    }
}

void MainWindow::on_laserOn_clicked()    // Send "laser on" command
{
    sendData(commands[21]);
}

void MainWindow::on_laserOff_clicked() // Send "laser off" command
{
    sendData(commands[22]);
}
```

Liite 2. Laiteohjelmisto

kuvausvalojenOhjaus.c

```
/*
* kuvausvalojenOhjaus.c
*

Software for controlling photography lights in photometric stereo system.
The lightning sequence or individual channels can be configured through the UART.

UART settings: 9600 8-n-1
UART commands:
+-------<Set channel on>----------------+-------<Set used layers>-----------+
|$LSxx!     '$' = Start byte            | $SSxx!        '$' = Start byte    |
|           'L' = "LED"                 |               'S' = "Sequence"    |
|           'S' = "SET"                 |               'S' = "SET"         |
|           'x' = Layer name (A-C)      |               'x' = Start layer   |
|           'x' = Channel number (1-4)  |               'x' = End layer     |
|           '!' = End byte              |               '!' = End byte      |
|                                       |                                   |
+-------<Set used trigger signal>-------+-----------------------------------+
|$MODx!     '$' = Start byte            |
|         "MOD" = Mode                  |
|           'x' = Used signal ( 1 or 0) |
|                 1 = Quench-signal     |
|                 0 = X-signal          |
|           '!' = End Byte              |
|                                       |
+---------------------------------------+

* Created: 18.1.2015 15:36:52
* Author: Panu
* Device: ATmega328p
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "drv74hc595.h"
#include <avr/eeprom.h>

#define EEADDR 0x3FF                        // EEPROM address to save mode bit.
#define F_CPU           16000000            // CPU clockspeed.
#define BAUD            9600                // Baudrate for USART.
#define BRC             ((F_CPU/16/BAUD) -1 )   // Bitrate calculation.
#define RX_BUFFER_SIZE  32                  // Receive buffer size.
#define COMMAND_SIZE    7
#define LEDTIMER        0                   // Safety timer to clear led in seconds .
                                            // (1-2 s, or 0 max (~4 sec)).
volatile char rxBuffer[RX_BUFFER_SIZE];
char command[COMMAND_SIZE];
volatile uint16_t ledCtrl;
volatile int rxWritePos = 0;
volatile int rxReadPos = 0;
volatile uint16_t ledSeqMask = 1;           // Mask used with sequence.

uint16_t ledSeq = 0;                        // Led on sequence.
const char ack[] = {'A','C','K','\n','\r'};     // ACK message reply to valid command.
int commandIndex;
int mode;                                   // Set mode variable for used camera signal
                                            // 0 = X-signal, 1 = Quench-signal.

char getChar();
char peekChar();
void parseParam(char[]);
void setLedNum(int);
void setLedArray(uint16_t);
void sendAck();
void readEEPROM();


int main(void)                              // Main
{
    init_hc595();
    setLedArray(0);                         // Clear leds just to be sure.

    DDRB = 0b00100000;                      // Debug led PB5.
    UBRR0H = (BRC >> 8);                    // Set BAUD rate register upper
    UBRR0L =  BRC;                          // and lower byte.
```

```
76
77      UCSR0B = (1 << TXEN0) | (1 << RXEN0) | (1 << RXCIE0); // Enable transmitter, receiver and
78                                                            //interrupt in control register.
79      UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);     // Set 1 stop bit and 8 data bits.
80
81      TIMSK1 |= (1 << TOIE1);                     // Enable timer overflow interrupt.
82      TCCR1B |= (1 << CS12) | (1 << CS10);        // Set timer clk/1024.
83
84      EICRA = (1 << ISC00);                       // Set interrupt on pin logic change.
85      EIMSK = (1 << INT0);                        // Enable external interrupt.
86
87      DDRD &= ~(1 << 2);                          // Set pin as input for camera sync signal.
88
89      readEEPROM();
90      //setLedNum(17);                            // Clear leds just to be sure.
91
92      sei();
93      char temp;
94      while(1)
95      {
96      while ((temp = getChar()) != '!')           // Do this while the received character is
97          {                                       // not "command end" character '!'.
98              if (temp == '$')                    // Reset the command index counter,
99              {                                   // if "command start" '$' is received.
100                 commandIndex = 0;
101             }
102
103             if (temp != '\0')                   // Write character to command array only
104             {                                   // when character is received (not /0).
105                 command[commandIndex++] = temp;
106                 if (commandIndex > 6)           // Commands is exactly 6 characters long.
107                 {
108                     commandIndex = 0;
109                 }
110             }
111
112         }
113         parseParam(command);
114     }
115 }
116
117 ISR(USART_RX_vect)                              // USART receive interrupt vector.
118 {
119     rxBuffer[rxWritePos++] = UDR0;              // Add received byte to buffer and
120                                                 // move position to next.
121     if (rxWritePos >= RX_BUFFER_SIZE)           // Reset write index to zero if
122     {                                           // buffer size is full.
123         rxWritePos = 0;
124     }
125 }
126
127 ISR(TIMER1_OVF_vect)                            // Interrupt vector for led clearing
128 {
129     if ((ledCtrl & 0b0001111111111111) > 0)
130     {
131         ledCtrl = 0;
132         setLedNum(17);
133     }
134     TCNT1 = 15625 * LEDTIMER;                   // Set next interrupt.
135 }
136
137 ISR(INT0_vect)                                  // Camera sync pin interrupt.
138 {
139     if ((PIND & (1 << 2)) == (mode & (1 << 0))) // Interrupt according to mode variable.
140     {
141         if ((ledSeq & 0b111111111111))
142         {
143             while(!((ledSeq & 0b111111111111) & ledSeqMask))
144             {
145                 ledSeqMask = ledSeqMask << 1;
146                 if (ledSeqMask >= 0b1000000000000 )
147                 {
148                     ledSeqMask = 1;
149                 }
150             }
```

```
151             ledCtrl = ledSeqMask;
152             setLedArray(ledCtrl);
153             ledSeqMask = ledSeqMask << 1;
154         }
155     }
156     else                                        // clear leds.
157     {
158         setLedNum(17);
159     }
160 }
161 char getChar(void)                              // Takes character from receive buffer.
162 {
163     char ret = '\0';
164     if (rxReadPos != rxWritePos)                // If receive buffer has unread characters.
165     {
166         ret = rxBuffer[rxReadPos++];
167         if (rxReadPos >= RX_BUFFER_SIZE)
168         {
169             rxReadPos = 0;
170         }
171     }
172     return ret;
173 }
174
175 char peekChar(void)                             // Returns next character in receive buffer
176 {                                               // without destroying it.
177     char ret = '\0';
178     if (rxReadPos != rxWritePos)
179     {
180         ret = rxBuffer[rxReadPos];
181     }
182     return ret;
183 }
184
185 void parseParam(char x[])                       // Makes actions according to received command.
186 {
187     switch (x[1])
188     {
189         case 'L':
190         if(x[2] == 'S')
191         {
192             if (x[3] == 'A')
193             {
194                 if ((x[4] - '0') > 0 && (x[4] -'0') < 5)
195                 {
196                 PORTB ^= (1 << 5); // Debug led
197                 setLedNum((x[4] - '0') -1);
198                 sendAck();
199                 }
200             }
201             else if (x[3] == 'B')
202             {
203                 if ((x[4] - '0') > 0 && (x[4] -'0') < 5)
204                 {
205                     PORTB ^= (1 << 5); // Debug led
206                     setLedNum(((x[4] - '0') -1) +4);
207                     sendAck();
208                 }
209             }
210             else if (x[3] == 'C')
211             {
212                 if ((x[4] - '0') > 0 && (x[4] -'0') < 5)
213                 {
214                     PORTB ^= (1 << 5); // Debug led
215                     setLedNum(((x[4] - '0') -1) +8);
216                     sendAck();
217                 }
218             }
219         }
220         break;
221
222         case 'S':                               // Set lightning sequence
223         if(x[2] == 'S')
224         {
225             ledSeq = 0;
```

```
            ledSeqMask = 1;
            if (x[3] == 'A')
            {
                ledSeq |= 0b000000001111;
                if (x[4] == 'C')
                {
                    ledSeq |= 0b111111111111;
                }
            }

            else if (x[3] == 'B')
            {
                ledSeq |= 0b000011110000;
            }

            else if (x[3] == 'C')
            {
                ledSeq |= 0b111100000000;
            }

            if (x[4] == 'A')
            {
                ledSeq |= 0b000000001111;
                sendAck();
            }

            else if (x[4] == 'B')
            {
                ledSeq |= 0b000011110000;
                sendAck();
            }

            else if (x[4] == 'C')
            {
                ledSeq |= 0b111100000000;
                sendAck();
            }
        }
        break;

        case 'M':                               // Signal mode command
        if(x[2] == 'O' && x[3] == 'D' && x[4] == 'X')
        {
            // Write mode to EEPROM and update mode variable.
            eeprom_write_byte ((uint8_t*)EEADDR, 0);
            readEEPROM();
            sendAck();
        }
        else if (x[2] == 'O' && x[3] == 'D' && x[4] == 'Q')
        {
            eeprom_write_byte ((uint8_t*)EEADDR, 1);
            readEEPROM();
            sendAck();
        }

        break;

        case 'F':                               // Focus laser command
        if(x[2] == 'S' && x[3] == 'O' && x[4] == 'N')
        {
            setLedNum(13);
            sendAck();
        }

        else if (x[2] == 'O' && x[3] == 'F' && x[4] == 'F')
        {
            setLedNum(17);
            sendAck();
        }
        break;

        default:
        break;
    }

```

```
301     int i;
302     for (i = 0; i<COMMAND_SIZE; i++)
303     {
304         command[i] = '\0';
305     }
306 }
307
308 void sendAck()                                  // Function for sending acknowledged message.
309 {
310     int i;
311     for (i = 0; i<5; i++)
312     {
313         UDR0 = ack[i];
314         while ( !( UCSR0A & (1<<UDRE0)) ){} ;
315     }
316
317 }
318
319 void setLedNum(int x)                           // Set led 0-14, clear all leds with parameter 17.
320 {
321     TCNT1 = 15625 * LEDTIMER;                   // Set timer for clearing led.
322     ledCtrl =  (1 << x);
323     write_hc595(ledCtrl);                       // send ledctrl array to shift registers.
324 }
325
326 void setLedArray(uint16_t x)                    // Set leds on with an 16 bit word,
327 {                                               // where each bit represents an output.
328     TCNT1 = 15625 * LEDTIMER;                   // Set timer for clearing led.
329     write_hc595(x);                             // send ledctrl array to shift registers.
330 }
331
332 void readEEPROM()                               // Read the last signal mode
333 {
334     mode = eeprom_read_byte((uint8_t*)EEADDR);
335     if(mode == 0)
336     {
337         PORTD |= (1 << 2);
338     }
339     else
340     {
341         PORTD |= (1 << 2);
342     }
343 }
```

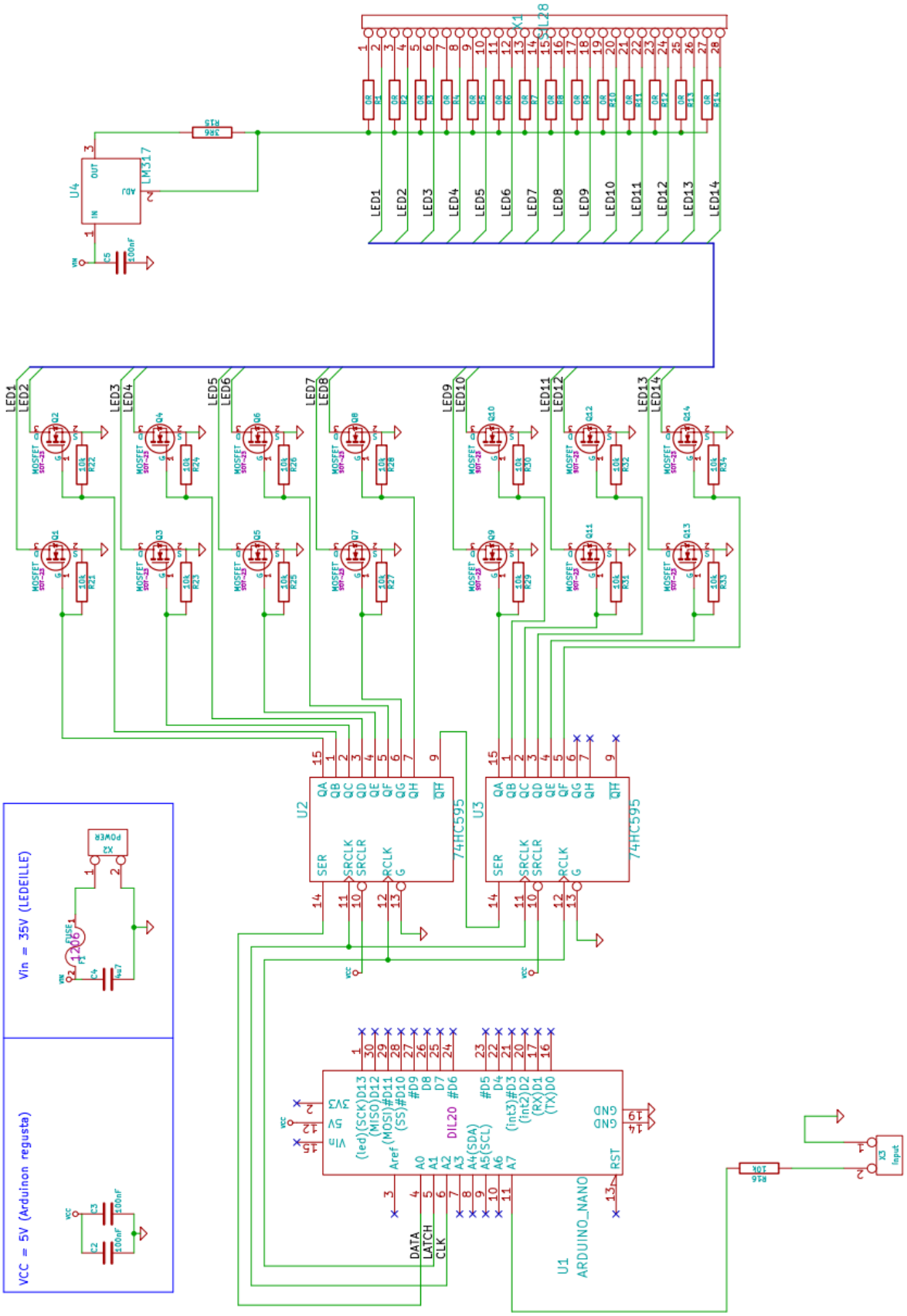
drv74hc595.h

```
1  /*
2   * drv74hc595.h
3   *
4   * Created: 13.11.2014 18:41:24
5   *  Author: Panu
6   */
7
8  #include <avr/io.h>
9
10 #ifndef DRV74HC595_H_
11 #define DRV74HC595_H_
12
13 #endif /* DRV74HC595_H_ */
14
15 #define hc595_dataPin  0 // Set used pins
16 #define hc595_clockPin  2
17 #define hc595_latchPin  1
18 #define hc595_DDR DDRC      // Set used port data register
19 #define hc595_port PORTC    // Set used port
20
21 extern void write_hc595(const uint16_t data);
22 extern void init_hc595();
```

drv74hc595.c

```
/*
 * 74hc595.c
 *
 Driver for the 74hc595 shift register

 Control two daisy chained 74hc595 or similar shift registers.

 init_hc595     Function initializes the port and pins used to control shift registers.
 write_hc595    Function takes unsigned 16 bit word and sends it MSB first to the shift registers.

 * Created: 13.11.2014 18:41:12
 *  Author: Panu
 */

#include "drv74hc595.h"
#include <avr/io.h>



void init_hc595()
{
hc595_DDR ^= (1 << hc595_dataPin) | (1 << hc595_clockPin) | (1 << hc595_latchPin); // set data direction registers
hc595_port &= ~(1 << hc595_dataPin) | (1 << hc595_clockPin) | (1 << hc595_latchPin); // write low
}

void write_hc595(const uint16_t data)
{
    uint16_t maski = 0b1000000000000000;
       for (int i = 0; i<16; i++)
       {
           if(data & maski)
           {
               hc595_port |= (1 << hc595_dataPin);
           }
           else
           {
               hc595_port &= ~(1 << hc595_dataPin);
           }
           hc595_port |= (1 << hc595_clockPin);
           hc595_port &= ~( 1 << hc595_clockPin);
           maski = (maski >> 1);

       }
       hc595_port |= ( 1 << hc595_latchPin);
       hc595_port &= ~( 1 << hc595_latchPin);

       hc595_port &= ~(1 << hc595_dataPin);

}
```

Liite 3. Kytkentäkaavio



Liite 4. Piirilevy