Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut Järjestelmät
Jaakko Isohella

Opinnäytetyö

Mikrokontrolleriohjattu akuston balansointilaite.

Työn ohjaaja Yliopettaja Mauri Inha
Työn tilaaja Tampereen ammattikorkeakoulu
Tampere 3/2012

Tampereen ammattikorkeakoulu
Tietotekniikka, Sulautetut järjestelmät
Tekijä Jaakko Isohella
Työn nimi Mikrokontrolleriohjattu akuston balansointilaite
Sivumäärä 47, josta liitteitä 14 sivua.
Valmistumisaika Maaliskuu 2012
Työn ohjaaja Yliopettaja Mauri Inha
Työn tilaaja Tampereen ammattikorkeakoulu

TIIVISTELMÄ

Tässä työssä suunniteltiin ja toteutettiin mikrokontrolleriohjattu säädettävä vakiovirtalähde sähköskootterin akuston balansointiin. Työssä perehdyttiin flyback-hakkurivirtalähteen peruskytkentään ja sen suunnitteluun sekä vakiovirran säätämiseen mikrokontrollerin avulla. Työssä perehdyttiin myös modulaarisen ohjelmakoodin suunnittelemiseen mikrokontrollerille.

Työssä esitellään flyback-tyyppisen hakkurivirtalähteen suunnitteluprojekti sekä kuinka hakkurista ulostulevan virran suuruutta voidaan säätää akuston latausta varten. Työssä esitellään myös kuinka latausvirta voidaan mitata ja miten mikrokontrolleri erotetaan hakkurivirtalähteestä. Lisäksi työssä käydään läpi millainen ohjelmisto mikrokontrollerilla tulee olla.

Avainsanat Flyback-hakkurivirtalähde, mikrokontrolleri, sulautettu järjestelmä galvaaninen erotus, vakiovirtalähde,

TAMK University of Applied Sciences
Information Technology, Embedded systems
Writer	Jaakko Isohella
Thesis	Microcontroller controlled battery balancer.
Pages	47, appendices 14 pages.
Graduation time	March 2012
Thesis supervisor	Senior lecturer Mauri Inha
Co-operating company	TAMK University of Applied Sciences

ABSTRACT

The objective of this thesis was to design and implement a microcontoller controlled adjustable constant current source for an electric scooter battery management system. This thesis takes a look at the basic flyback converter schematic and how the converter was designed. Also presented is how to adjust the output current of the source with a microcontroller and how to implement a modular program for it.

This thesis sets out to give a clear image of what kind of a project designing a flyback converter is and how the output current of the converter can be controlled to provide a constant current source. This thesis also provides some perspective as to what has to be taken into account on the design and also what kind of a software have to be developed so that the current can be controlled using external device.

Keywords: Flyback converter, microcontroller, embedded system, galvanic isolation, constant current source

Alkusanat

Tämä työ on tehty Tampereen ammattikorkeakoulun sulautettujen järjestelmien linjan opinnäytetyönä.

Tämän työn tarkoituksena oli oppia kuinka hakkurivirtalähde suunnitellaan ja kuinka mikrokontrollerin galvaaninen erotus hakkurista toteutetaan. Työn tarkoituksena oli myös oppia kuinka mikrokontrollerin avulla voidaan tehdä optoerotettu vakiovirtalähde. Työ tehtiin pääasiassa loppuvuoden 2011 ja alkuvuoden 2012 välisenä aikana.

Haluan kiittää kaikkia opettajiani Tampereen ammattikorkeakoulussa, sillä heidän opetuksiensa ja ohjauksensa ansiosta tämä työ pystyttiin tekemään. Kiitän myös erityisesti Mauri Inhaa opinnäytetyön aiheesta sekä Kai Poutasta opastuksesta elektroniikan toteuttamisen suhteen. Kiitän myös kaikkia oikolukuun osallistuneita ja kieliopissa auttaneita henkilöitä. Haluan kiittää myös Nokia Oyj:tä mahdollisuudesta valmistaa ja kalustaa prototyyppipiirilevy heidän tiloissaan.

Tampereella 5. maaliskuuta 2012

Jaakko Isohella

SISÄLLYS

1 JOHDANTO 7
2 SUUNNITTELUN RAJOITUKSET 8
3 LAITESUUNNITTELU 9
3.1 Flyback-hakkurivirtalähde 9
3.2 Mikrokontrollerin liittäminen hakkuriin 13
3.2.1 Mikrokontrollerin virtalähde 14
3.2.2 Mikrokontrollerin ja sen oheislaitteiden kytkentä 15
3.2.3 Optoerotus hakkurista 17
3.3 Piirilevy 20
3.4 Kotelointi 22
4 OHJELMISTOSUUNNITTELU 23
4.1 Pääohjelma 23
4.2 PWM-ohjaussignaali 23
4.3 AD-muunnos 25
4.4 Ajastin 10ms keskeytykselle 26
4.5 Sarjaväylä 27
4.6 Ohjauskomennon käsittely 28
5 TOTEUTUS 30
5.1 Prototyyppipiirilevy 30
5.2 Komponenttien kiinnitys 32
6 YHTEENVETO 33
LÄHTEET 34
LIITTEET 35
Liite 1. Pääohjelman otsikkotiedostot ja C-kielinen ohjelmatiedosto. 36
Liite 2. PWM-moduulin otsikkotiedosto ja C-kielinen ohjelmatiedosto 38
Liite 3. AD-muunnoksen otsikkotiedosto ja C-kielinen ohjelmatiedosto. 39
Liite 4. Ajastimen otsikkotiedosto ja C-kielinen ohjelmatiedosto. 41
Liite 5. Sarjaväylän otsikkotiedosto ja C-kielinen ohjelmatiedosto. 42
Liite 6. Komentoaliohjelman otsikkotiedosto ja C-kielinen ohjelmatiedosto. 45

LYHENTEET JA TERMISTÖ

ISP	In-System Programming, Mikrokontrollerin ohjelmointiin tarkoitettu ominaisuus. Mikrokontrolleri voidaan ohjelmoida ISP-liittimen kautta ilman mikrokontrollerin irroitusta järjestelmästä.
MCU	Microcontroller, Mikrokontrolleri
ADC	Analog to digital converter, muuntaa analogisen jännitteen digitaaliseksi
PWM	Pulse Width Modulation, pulssinleveysmodulaatio. Käytetään mosfetin ohjaamiseen.
UART	Universal Asynchronous Receiver Transmitter, lähettimenä ja vastaanottimena toimiva sarjaliikennepiiri.
BMS	Battery Management System, akustonhallintajärjestelmä.
SoC	State of Charge, akuston jäljellä oleva varaus prosenteina.
SRAM	Static Random Access Memory, mikrokontrollerin keskusmuisti.

1 JOHDANTO

Nykypäivän ajankohtaisena kysymyksenä on ympäristön suojelu ja ihmisten jälkeen jättämä hiilijalanjälki. Tästä syystä sähkökäyttöiset ajoneuvot ovat valtaamassa alaa ja monet alan osaajat pyrkivätkin kehittämään omat ratkaisunsa, jotta sähköllä toimivat ajoneuvot olisivat tasavertaisia polttomoottorikäyttöisten kanssa. Sähköajoneuvo ei ideana ole uusi, mutta tekniikan kehittymisen johdosta sähköajoneuvojen hyötysuhteet ovat parantuneet merkittävästi.

Tätä tutkintotyötä tehdessä perehdyttiin sähkökäyttöisen ajoneuvon akustonhallinta-järjestelmän (BMS) osa-alueisiin ja niiden toimintaperiaatteisiin. Tämän työn aiheeksi valittiin akuston energian tasaamiseen liittyvä akuston balansointilaite, koska sekä BMS-järjestelmä että hakkurivirtalähde koettiin mielenkiintoiseksi ja ne haluttiin toteuttaa. Tämä aihe liittyi sopivasti molempiin osa-alueisiin. Työssä tehtiin BMS-järjestelmään liittyvä akkukennojen balansointia hoitava laite, jonka tarkoituksena on mahdollistaa energian siirtäminen koko akustosta yhdelle akkukennolle, jonka varaus muihin akuston akkukennoihin nähden on heikoin.

Työn tekeminen jaettiin kahteen eri osioon. Ensin suunniteltiin laitteen elektroniikka eli suunniteltiin ja piirrettiin laitteen kytkentäkaavio ja piirilevy sekä kasattiin laite. Toisessa osiossa kirjoitettiin laitteessa olevalle mikrokontrollerille ohjelmisto, joka vahtii kytkennän toimintaa sekä keskustelee sarjaväylän kautta BMS-järjestelmän päälaitteen kanssa.

2 SUUNNITTELUN RAJOITUKSET

Aloitettaessa akustoon liitettävän hakkurivirtalähteen suunnittelua, suunnittelussa täytyy ottaa huomioon monia eri asioita. Ensimmäisiä asioita joita tulee ottaa huomioon on akuston kennojen jännitteet ja tarvittava hakkurivirtalähteen teho. Tässä tapauksessa, kun skootterin akkupaketti koostuu LiFePO4 rautafosfaattiakuista, yksittäisten kennojen jännite vaihtelee 3.3 – 3.65 V välillä ja akkukennoja on 20 kappaletta, niin hakkurin sisääntulojännite voi vaihdella 60 – 75 V välillä. Ulostulojännite pyritään pitämään mahdollisimman tasaisesti 5 voltissa, joka on hieman korkea kennojännitteeksi, mutta BMS-järjestelmässä on ominaisuutena automaattinen balansoinnin katkaisu, kun ladattava kenno saavuttaa maksimijännitteensä. Toinen tärkeä asia, joka tulee ensimmäisenä ottaa huomioon on hakkurivirtalähteen antama ja ottama teho eli paljonko virtaa purettavalta akkukennolta otetaan ja paljonko sitä siirretään vastaanottavalle kennolle. Tälle laitteelle on päätetty, että ulostulevaa virtaa voidaan säätää balansointitarpeen mukaan. Hakkurin ulostulovirran maksimiksi on määritetty kaksi ampeeria. Näin ollen hakkurin ulostuloteho on

$$P = UI = 5\,V \cdot 2\,A = 10\,W. \tag{1}$$

Seuraavaksi tärkeä asia juuri akustoon liitettävälle laitteelle on sen galvaaninen erotus. Koska koko akusto, jolta energiaa puretaan, ja akkukenno, jolle energiaa ladataan, sijaitsevat saman akkupaketin sisällä, mutta ovat eri potentiaalissa, niin niiden välillä ei saa kulkea tasavirta. Eristys täytyy olla esimerkiksi sen takia, että akkupaketin akkukennot ovat eri potentiaaleissa, ja tämän vuoksi suorat kytkennät voisivat muun muassa vaurioittaa kennoja tai aiheuttaa oikosulkuja. Galvaanisen erotuksen toteuttaminen on haastavaa, koska myös vakiovirran säätävä mikrokontrolleri, jota käytetään hakkurivirtalähteen toisiojännitteen ja ulostulovirran mittaamiseen sekä vakiovirtakytkennän mosfet kytkimen ohjaukseen, tulee täysin erottaa sekä hakkurin sisääntulosta että ulostulosta.

Muita rajoituksia tai suunnitteluun vaikuttavia asioita ovat hakkurivirtalähteen haluttu hyötysuhde, kytkentätaajuus, ulostulojännite ja fyysinen koko.

3 LAITESUUNNITTELU

Laitteen elektroniikan suunnittelu aloitettiin tutkimalla eri hakkuritekniikoita sekä erilaisten hakkurivirtalähteiden kytkentäkaavioita. Flyback-tyyppiseen tekniikkaan päädyttiinkin sen yksinkertaisuuden ja melko hyvän hyötysuhteen takia. Myös tehorajoitus oli valintaan vaikuttava tekijä, koska laitteen ulostuloteho on vain kymmenen wattia ja flyback-kytkentää on suositeltu käytettävän 0 – 100 watin teholuokassa.

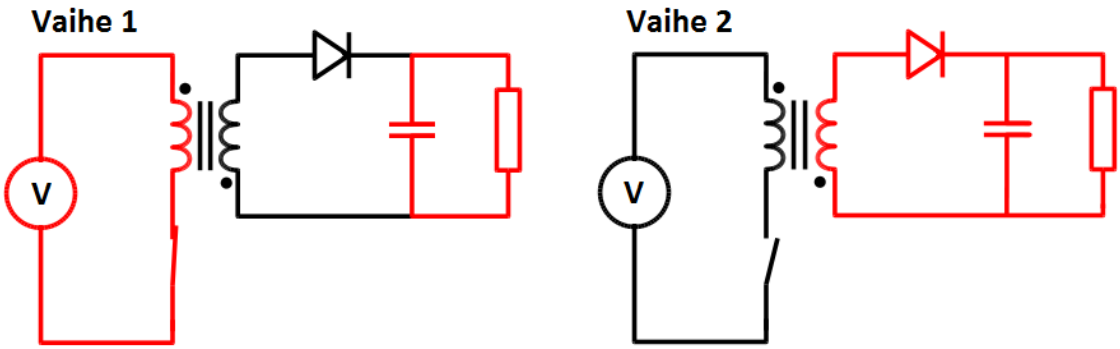
Mikrokontrollerin liittäminen hakkurivirtalähteeseen oli suuntautumisvaihtoehdon takia luontevaa, mutta myös BMS-järjestelmän takia pakollista, koska balansointivirtaa pitäisi pystyä ohjaamaan järjestelmän pääkortilla sarjaväylää pitkin. Mikrokontrolleriohjattu akuston balansointilaite mahdollistaa myös balansointivirtojen tarkkailun ja lähettämisen sarjaväylälle, joten BMS-järjestelmän pääkortti pystyy laskemaan akuston jäljellä olevan varauksen, State Of Charge (SoC), paljon tarkemmin.

3.1 Flyback-hakkurivirtalähde

Yksinkertainen flyback-hakkurivirtalähde koostuu muuntajasta, muuntajan ensiöpuolelle kytketystä mosfet-transistorista, toisiopuolelle kytketystä diodista ja suodatuskondensaattorista. Haluttu ulostulojännite hakkurista saadaan siten, että ensiöpuolella olevaa transistoria ohjataan korkeataajuisella pulssileveysmoduloidulla kanttiaallolla. Kanttiaallon pulssisuhdetta muuttamalla voidaan toisiojännitettä ja toisiovirtaa muuttaa. Ulostulojännitteeseen vaikuttaa myös muuntajan muuntosuhde ja toisiopuolen diodin kynnysjännite.

Flyback-hakkurin toimintaperiaate voidaan kuvata kahdella työvaiheella. Ensimmäisessä työvaiheessa ensiöpuolella oleva transistori johtaa, jolloin ensiökäämille varautuu energiaa ja toisiokäämi ei johda. Toisessa työvaiheessa transistori ei johda, jolloin ensiökäämille varautunut energia siirtyy toisiokäämille ja siitä edelleen diodin läpi suodatuskondensaattoreille. Suodatuskondensaattorin merkitys kytkennässä on suuri, sillä ensimmäisen työvaiheen aikana toisiokäämillä ei kulje virtaa, niin silloin hakkurin ulostuloon kytketty kuorma saa virtansa suodatuskondensaattorilta. Toisen työvaiheen aikana suodatuskondensaattori latautuu ja kuorma saa virtansa toisiokäämin energiasta. Kuvassa 1 on havainnollistettu flyback hakkurin kaksi työvaihetta. Kuvan muuntajassa olevat pisteet kertovat muuntajan kytkentäsuunnasta. Ensimmäisessä vaiheessa kytkimen ollessa kiinni virta menee ensiöpuolelle sisään pisteestä ja pyrkii

tulemaan ulos toisiopuolen pisteeltä. Estosuuntaan kytketty diodi kuitenkin estää toisiopuolen johtavuuden. Kytkimen ollessa auki ensiökäämiin ei enää kulje virtaa, joten ensiökäämiin varautunut energia siirtyy toisiokäämille ja diodin läpi ulostuloon. Muuntajan kytkentäsuunta ja diodin suunta kytkentäsuuntaan nähden ovat erittäin tärkeitä flyback-hakkurin toiminnan kannalta.



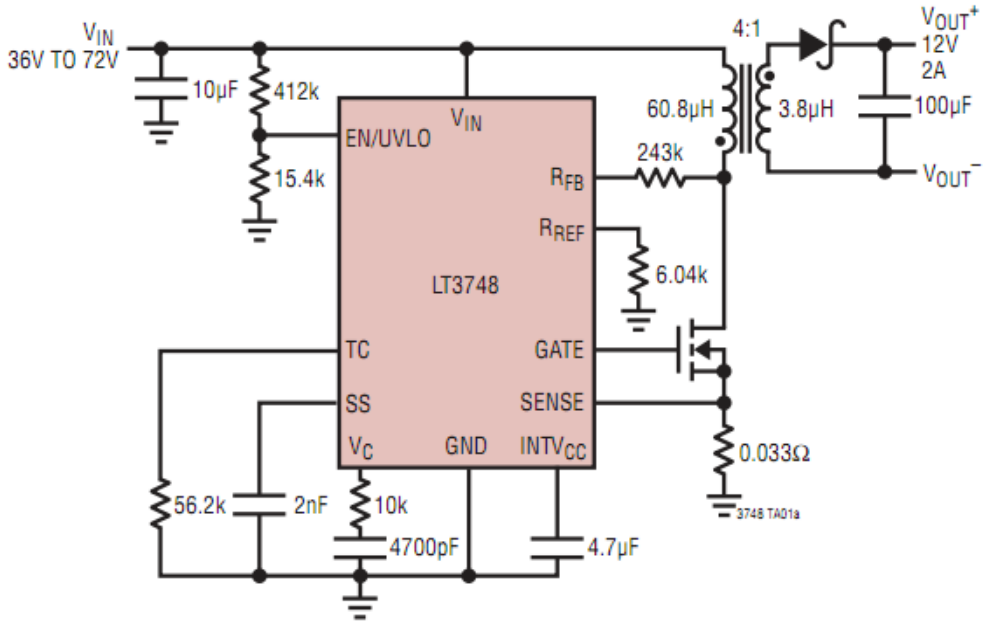
Kuva 1. Flyback-hakkurin työvaiheet. [11]

Flyback-hakkurivirtalähteen suunnittelu aloitettiin simuloimalla yksinkertaisin flyback-kytkentä ilmaisella LTspice IV -ohjelmalla. LTspice IV -ohjelmaa päädyttiin käyttämään, koska se on ilmainen ja melko monipuolinen spicesimulaattori. Ohjelman käytöstä on myös aikaisempaa kokemusta, joten sen käyttöönotossa ei ilmennyt juuri minkäänlaisia ongelmia. [1]

Linear Technologyn valmistamaa LT3748 hakkuripiiriä päädyttiin käyttämään, koska se mahdollisti yksinkertaisen, mutta tehokkaan hakkurikytkennän. LT3748 hakkuripiirin valintaan vaikuttanut ominaisuus oli se, että piiri ei tarvitse takaisinkytkentää tai kolmatta käämiä muuntajalle, vaan se osaa ohjata MOSFET-transistoria ensiökäämin aaltomuodon perusteella. Myös vähäinen oheiskomponenttien määrä ja simulointimallin löytyminen LTspice-ohjelmasta vaikutti piirin valintaan.[2]

Kytkennän suunnittelu aloitettiin tutkimalla LT3748 piirin datalehteä ja valitsemalla datalehden testatuiden muuntajien taulukosta Würthin valmistama 4:1 muuntaja, jonka ulostuloksi on merkitty 5 volttia. Muuntaja olisi voitu itsekin käämiä taulukon parametrien mukaan, mutta valmiiksi käämitty muuntaja päätettiin valita siksi, että se on testattu ja todettu toimivaksi hakkuripiirin kanssa. MOSFET-transistorin valintaan vaikuttivat sen jännitekestoisuus sekä sisäinen resistanssi. MOSFET-transistoriksi valittiinkin Internal Rectifierin IRFR4620PbF, transistorin sisäisen resistanssin ja nielu-lähde läpilyöntijännitteen takia. Datalehden mukaan transistorin resistanssi on

maksimissaan 78 mΩ ja nielu-lähde läpilyöntijännite on 200 V. Toisiopuolen diodin valintaan taas vaikutti sen virtakestoisuus ja nopeus. Diodiksi valittiinkin Vishay Semiconductorsin 2x6A Schottky diodi VS-12CWQ10FNPBF. Kun tärkeimmät komponentit oli valittu, aloitettiin kytkennän suunnittelu kuvan 2 mukaan. [2][3][4]



Kuva 2. LT3748 esimerkkikytkentä.[2]

Simuloinnin suunnittelu aloitettiin piirtämällä kuvan 2 mukainen kytkentä ja laskemalla komponenttiarvot LT3748 piirin datalehden laskukaavojen mukaan. Kuten edellisessä kappaleessa todettiin, niin muuntajaksi valittiin 4:1 muuntaja datalehdestä. Muuntajan ensiökäämin induktanssi on 50uH, jolloin toisiokäämin induktanssiksi tulee

$$L1 = L2 \cdot (4{:}1)^2 \rightarrow L2 = \frac{L1}{(4:1)^2} = \frac{50uH}{16} = 3{,}125\ uH. \quad (2)$$

Komponenttiarvojen laskeminen aloitettiin datalehden sovellusohjeen mukaisesti R_{REF}, R_{FB} ja R_{TC} vastuksista. Datalehdessä sanotaan, että R_{REF} tulisi olla suurinpiirtein 6.04 kΩ. R_{FB} vastuksen arvo pystytään laskemaan kaavalla

$$R_{FB} = \frac{R_{REF} \cdot N_{PS}[(V_{OUT}+V_F)+V_{TC}]}{V_{BG}} \quad (3)$$

josta R_{REF} on 6.04kΩ, N_{PS} on muuntajan muuntosuhde, V_{OUT} on ulostulojännite ja V_F on diodin kynnysjännite. V_{TC} on 0.55 V ja V_{BG} 1.223 V datalehden mukaan.

Sijoitetaan arvot lauseeseen jolloin R_{FB} arvoksi saadaan

$$R_{FB} = \frac{6.04\ k\Omega \cdot 4[(5\ V + 0.65\ V) + 0.55\ V]}{1.223\ V} \approx 100\ k\Omega.$$

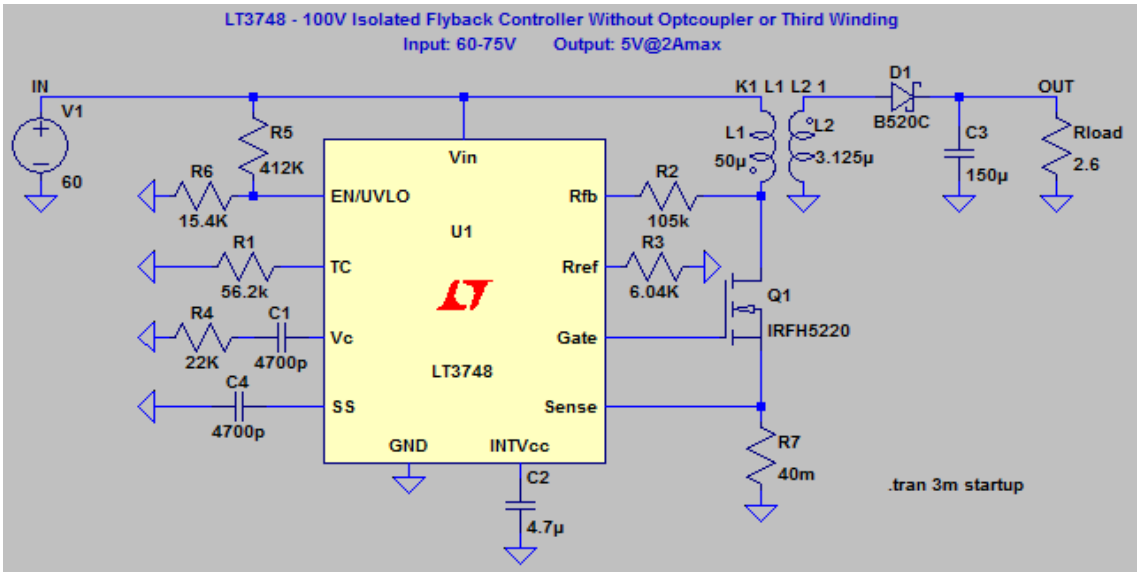
Vastuksen R_{TC} arvoksi valittiin 56.2 kΩ kuvan 2 mukaan ja R_{FB} vastuksen todellinen arvo saatiin mittaamalla 100 kΩ vastusarvolla ulostulojännite ja laskemalla uusi arvo kaavalla 4, kuten datalehdessä on neuvottu.

$$R_{FB(NEW)} = \frac{V_{OUT(DESIRED)}}{V_{OUT(MEASURED)}} \cdot R_{FB(OLD)} = \frac{5\ V}{4.76\ V} \cdot 100\ k\Omega = 105\ k\Omega. \quad (4)$$

Seuraavaksi laskettiin R_{SENSE} vastuksen arvo. MOSFETin virtarajaksi asetettiin mielivaltaisesti 2.5A, jolloin R_{SENSE} vastuksen arvoksi tuli

$$R_{SENSE} = \frac{100\ mV}{I_{LIM}} = \frac{100\ mV}{2.5\ A} = 40\ m\Omega. \quad (5)$$

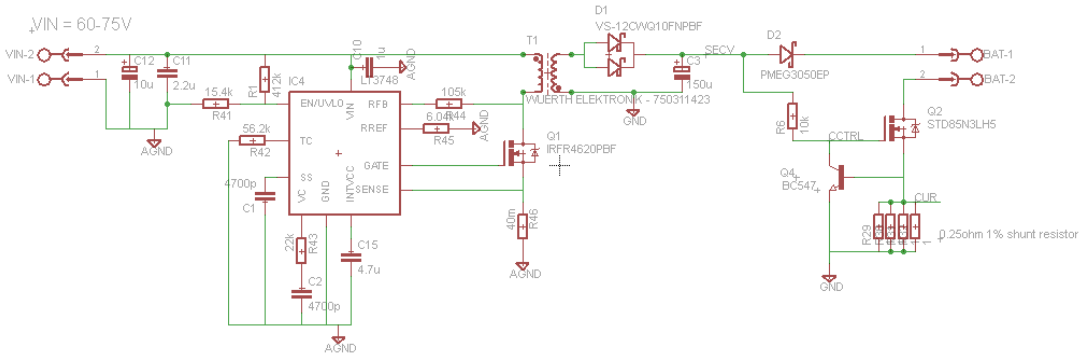
Ulostulokondensaattoriksi valittiin 150 uF matalan sarjaresistanssin, equivalent series resistance (ESR), omaava kondensaattori. Kondensaattorilla tulee olla matala sisäinen resistanssi, jotta sisäresistanssiin hukkuisi mahdollisimman vähän virtaa ja kondensaattori ei lämpenisi.. Loput passiivikomponenttien arvoista valittiin datalehden sovellusesimerkin kytkentäkaavioista ja sijoitettiin simulaatiomalliin. Kuvan 3 mukaisilla arvoilla simuloituna ulostulojännitteeksi, sisääntulojännitteen ollessa 60 V, saatiin 5.15 V rippelijännitteen ollessa noin 45 mV. Ulostulovirta oli rajoitettu 2.6 Ω vastuksella kahteen ampeeriin.



Kuva 3. LT3748 LTSpice simulaatiomalli. [1]

Jännitteen alentamisen lisäksi tarkoituksena oli, että ulostulovirtaa pystyttäisiin säätämään sen mukaan kuinka nopeaa latausta akkukennolle tarvitaan. Balansointivirran säätäminen toteutettiin MOSFET-transistorista, BJT-transistorista ja 0.25 Ω shuntvastuksesta koostuvasta vakiovirtasäätimestä, joka näkyy kuvan 4 oikeassa alakulmassa. Vakiovirtasäädin mahdollisti myös balansointivirran mittaamisen shuntvastuksen yli.

Balansointivirta mitataan shuntvastuksen yli mikrokontrollerin AD-muuntimella. Balansointivirran ollessa kuitenkin pulssimaista täytyy shuntvastuksen rinnalla olla alipäästösuodin ja sen jälkeen vahvistin, joka mahdollistaa jännitteen keskiarvon mittaamisen. Latausvirta säätyy transistori Q2:n pulssisuhteen mukaan. BJT-transistori BC547:n tehtävä kytkennässä on rajoittaa balansointivirran maksimiarvoksi 2.8A, jos MOSFET-transistorille ei tule lainkaan ohjausta.



Kuva 4. Flyback-hakkurin kytkentäkaavio.

3.2 Mikrokontrollerin liittäminen hakkuriin

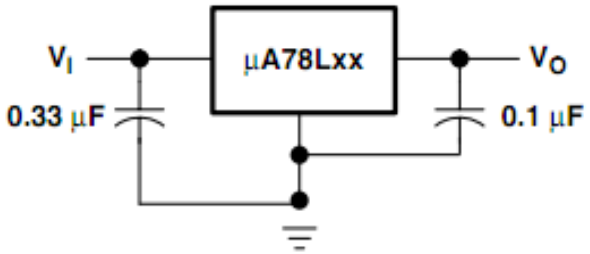
Mikrokontrolleriohjauksen toteuttaminen oli työn haastavin osuus, koska mikrokontrolleri täytyi erottaa täysin hakkurivirtalähteestä ja tietoliikenneväylästä. Erotus toteutettiin optoerottimien avulla. Tietoliikenneväylän optoerotus ei ole aivan välttämätön pakko, sillä BMS järjestelmän pääkortin virta ja balansointilaitteen virta syötetään samasta 12V pisteestä, jolloin laitteet ovat samassa potentiaalissa. Työtä tehdessä päätettiin kuitenkin jättää optio sarjaväylän optoerottamisellekin, mikäli laitetta käytettäisiin jossain eri ympäristössä kuin mihin se on suunniteltu.

Mikrokontrollerikytkennän yksinkertaisuuden takia kontrollerin ja sen oheiskomponenttien kytkennän suunnittelussa ei hirveästi aikaa kulunut. Haastavin osuus mikrokontrollerin liittämisessä hakkuriin oli jännitteen ja virran mittaamisen sekä vakiovirtasäätimen MOSFET-transistorin ohjauksen optoerottaminen. Optoerotuksen

suunnittelussa käytettiin paljon LTspice-ohjelmistoa apuna. Vähäisen tiedon ja kokemuksen takia LTspicella simuloitiin kaikki pienimmätkin ideat optoerotukseen liittyen. [1]

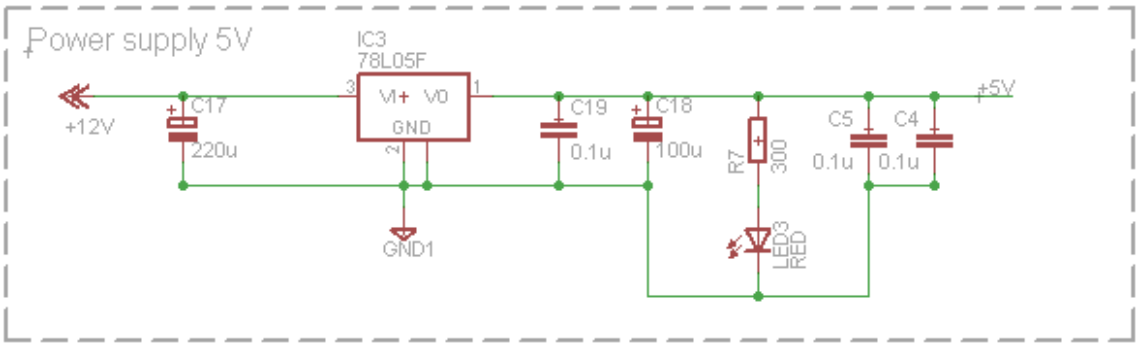
3.2.1 Mikrokontrollerin virtalähde

Koska laitteen mikrokontrolleri oli tarkoitus liittää sähköskootterin 12V jännitteeseen, täytyi mikrokontrollerille suunnitella oma virtalähteensä. Koska mikrokontrolleriosion virrantarve oli alle 100 mA, päädyttiin perinteiseen regulaattorikytkentään virtalähteen osalta. Regulaattoriksi valittiin Texas Instrumentsin 78L05 5V regulaattori. Regulaattorin sisääntulojännite voi datalehden mukaan vaihdella välillä 7-20 V. [6]



Kuva 5. 78L05 regulaattorin esimerkkikytkentä. [6]

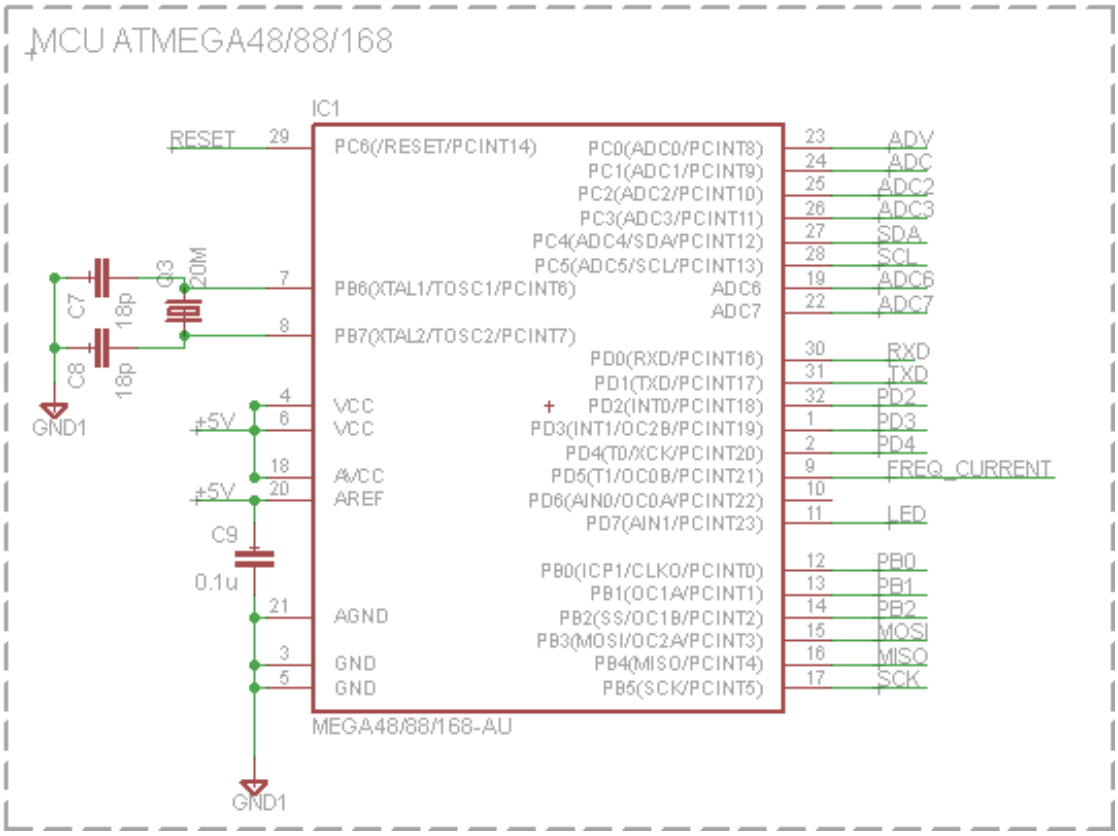
Koska regulaattorin esimerkkikytkentä on yksinkertainen, ja datalehdessäkin sanotaan, että ulkoisia komponentteja ei tarvita, päädyttiin rakentamaan virtalähde datalehden kuvan mukaisesti. Regulaattorin sisääntuloon laitettiin iso kondensaattori suodattamaan jännitevaihteluita ja ulostuloon muutama kondensaattori. ja käyttöjännitteen ilmoittava ledi. Regulaattorin lähtöön laitettiin vielä pari ylimääräistä 0.1 uF kondensaattoria, jotka sijoitetaan mikrokontrollerin välittömään läheisyyteen.



Kuva 6. Mikrokontrollerin virtalähde 78L05 regulaattorilla.

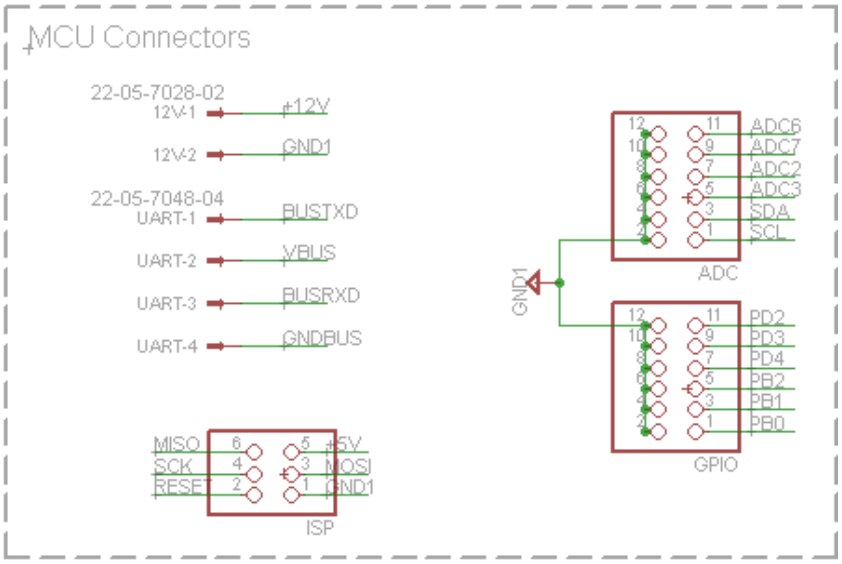
3.2.2 Mikrokontrollerin ja sen oheislaitteiden kytkentä

Mikrokontrolleriksi valittiin Atmelin tuoteperheestä 8-bittinen Atmega48A/88A/168A mikrokontrolleri. Kontrollerin IO-portit nimettiin Eagle PCB –ohjelmassa ja kytkettiin käyttöjännitteet sekä maapisteet kuvan 7 mukaisesti. Kontrollerin porttien nimeäminen on järkevää mikäli kytkentäkaavio halutaan pitää helposti luettavana. AD-muuntimen referenssijännite AREF:in ja AGND:n välille kytkettiin 0.1 uF kondensaattori häiriön vähentämiseksi ja virtapulssien tasaamiseksi, kuten Atmelin datalehdessä oli neuvottu. Lisäominaisuudeksi kontrollerille liitettiin paikka ulkoiselle kiteelle. [5][7]



Kuva 7. Mikrokontrollerin kytkennät.

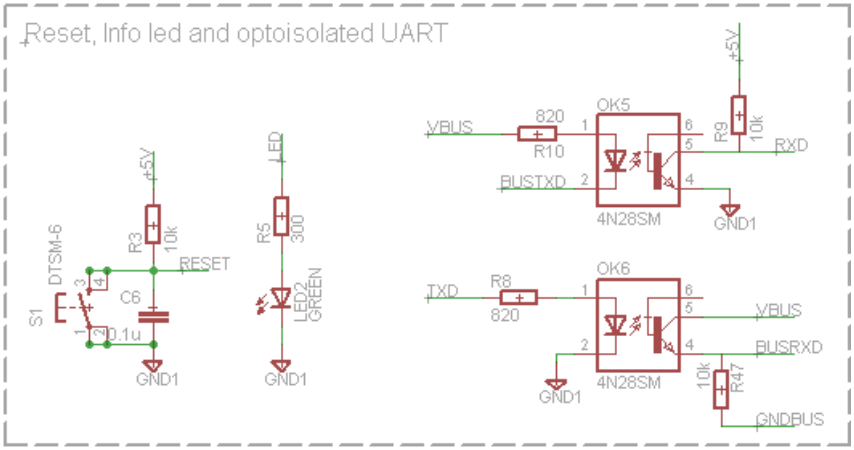
Mikrokontrollerin ylimääräiset portit, jotka eivät tulleet käyttöön heti, ohjattiin erillisiin liittimiin piirilevyn reunalle laajennusmahdollisuuden takia. Tällä tavoin haluttiin tehdä lisäominaisuuksien liittämisestä helpompaa. Kuusi porttia varattiin laitteen ISP-ohjelmointiliittimelle. Käyttöjännitteet ja sarjaportin optoerotettu väyläliitin kytkettiin myös kuvan 8 mukaan.



Kuva 8. Mikrokontrollerin liittimet.

Mikrokontrolleriin liitettiin vielä reset-nappi, ledi ja sarjaväylän optoerotus. Atmega mikrokontrollerin reset-portti on alhaalla aktiivinen eli, kun reset-linja kytketään maahan, niin mikrokontrolleri käynnistyy uudelleen. Reset-linjaan päätettiin napin lisäksi laittaa vastus ja kondensaattori luomaan viiveen jännitteen kasvulle. Tällöin varmistutaan, että käyttöjännitteen kytkemisen jälkeen menee jonkin aikaa, kunnes mikrokontrollerin reset-linjan jännite nousee ja mikrokontrolleri käynnistyy. Tällä tavoin vältytään siltä, että mikrokontrolleri jäisi käynnistyessään virheelliseen tilaan. [7]

Sarjaväylän optoerotus toteutettiin kuvan 9 mukaisesti. Optoerottimia käytetään sarjaväylässä kytkiminä. Esimerkiksi mikrokontrollerin lähettämä data sarjaväylän TX-linjaa pitkin sulkee databitin ollessa yksi kytkimen, jolloin väylän RX-linja kytkeytyy käyttöjännitteeseen. Databitin ollessa 0 kytkin pysyy auki ja RX-linja vedetään maahan 10 kΩ vastuksen kautta.

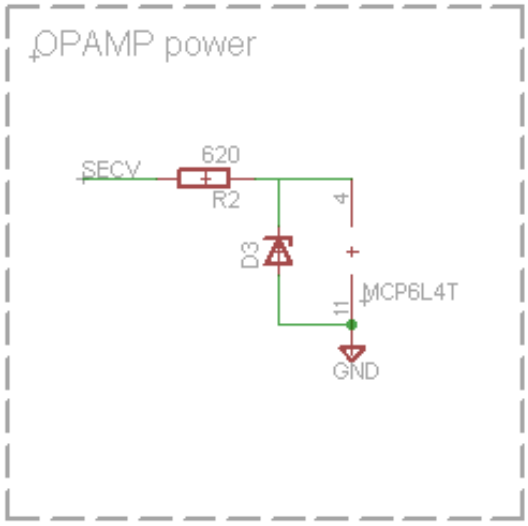


Kuva 9. Sarjaväylän optoerottimet.

3.2.3 Optoerotus hakkurista

Optoerottimiksi valittiin Vishay Semiconductorsin 4N28 optoerottimet, koska suunnitteluhetkellä ne olivat ainoat joita oli heti saatavilla. Alkuperäinen idea oli, että mitattava jännite muutettaisiin taajuudeksi, jolla ajettaisiin optoerotinta ja mikrokontrollerilla sitten mitattaisi optoerottimen ulostulon taajuus. Tämä idea kuitenkin hylättiin melkein heti ja päädyttiin kokeilemaan jännitteen muuttamista virraksi. Toteutustavaksi valittiin jännitteen muuttaminen virraksi, koska jännitteen muuttaminen taajuudeksi osoittautui suunnitteluhetkellä liian haastavaksi. Taajuuden siirtäminen optoerottimen yli olisi kuitenkin ollut tarkkuudeltaan järkevämpi keino. [8]

Optoerottimien yksi tärkeimmistä ominaisuuksista on virransiirtosuhde, current transfer ratio (CTR). Virransiirtosuhde ilmoitetaan yleensä prosentteina ja se on valoa vastaanottavan komponentin ja lähettävän komponentin suhde. Valitun 4N28 optoerottimen virransiirtosuhde huoneenlämmössä on 30%, kun ledin läpi kulkeva virta on 10mA. Virransiirtosuhde kuitenkin vaihtelee virran suuruuden ja lämpötilan mukaan. Tässä käyttötarkoituksessa se ei kuitenkaan ole mikään iso ongelma, koska optoerottimen läpi kulkevan virran vaihtelut ovat 1 mA - 3 mA väliltä. Virransiirto-suhde muuttuu niin vähän tällä välillä, että se ei mittaustuloksen tarkkuuteen hirveästi vaikuta. [8]



Kuva 10. Operaatiovahvistimen virransyöttö.

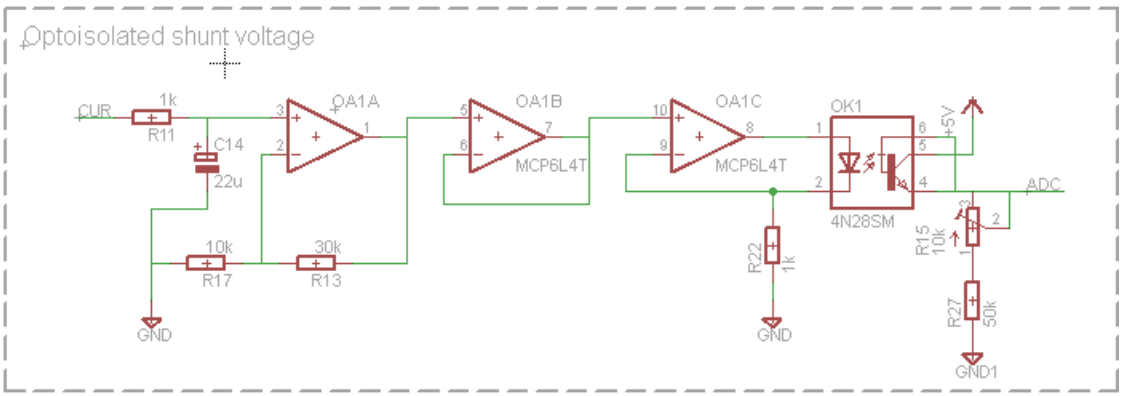
Mitattavan jännitteen muuntaminen virraksi toteutettiin operaatiovahvistimen avulla. Operaatiovahvistimeksi valittiin Microchipin MCP6L4T, joka on Rail-to-Rail tyyppinen neljä operaatiovahvistinta sisältävä piiri. Rail-to-Rail ominaisuus tarkoittaa sitä, että operaatiovahvistimen ulostulot voivat nousta hyvin lähelle käyttöjännitettä. Datalehden

mukaan tämän kyseisen operaatiovahvistimen ulostulojännitteen maksimiarvo on 4.960V. Valittu operaatiovahvistin toimii myös yksipuolisella käyttöjännitteellä, jolloin sen käyttöjännite voidaan ottaa suoraan hakkurin toisiopuolen jännitteestä. Operaatiovahvistimen käyttöjännitelinjaan on kuitenkin laitettu 5.6V zenerdiodi ja etuvastus suojaamaan operaatiovahvistinta ylijännitteeltä kuvan 10 mukaan. [9]

Akkukennon balansointivirran mittaaminen toteutettiin operaatiovahvistinpiirin kolmella operaatiovahvistimella. Balansointivirran mittaaminen olisi onnistunut pelkästään kahdella operaatiovahvistimellakin, mutta operaatiovahvistimen datalehdessä ilmoitettiin selkeästi, että kaikki käyttämättä jääneet operaatiovahvistimet eivät saa jäädä kellumaan vaan ne tulisi kytkeä datalehdessä ilmoitetulla tavalla. Tämän takia kolmas operaatiovahvistin kytkettiin puskuriksi vahvistimen ja jännite-virtamuuntimen väliin.

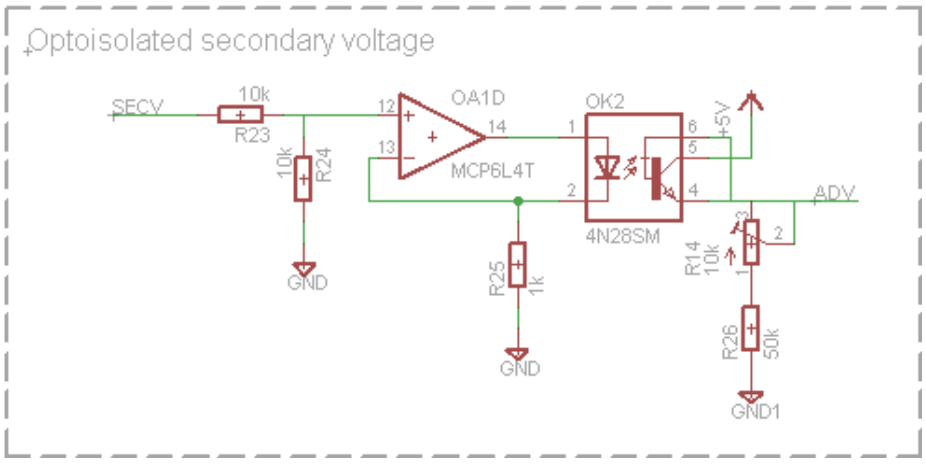
Akkukennon balansointivirta mitataan virtasäätimen 0.25 Ω vastuksen yli. Jotta jännitteen mittaamisesta saataisiin luotettava, jännite alipäästösuodetaan keskiarvojännitteeksi ja vahvistetaan operaatiovahvistimella. Jännittevahvistin vahvistaa jännitteen nelinkertaiseksi, jolloin jännitteen arvo vastaa virtaa ampeereina. Jännitteen vahvistamisen jälkeen se syötetään puskurivahvistimen kautta jännite-virtamuuntimelle. Jännite-virtamuunnin ohjaa optoerottimen lediä virralla, jonka suuruus on tuhannesosa jännitteestä, esimerkiksi 1.1 V jännitteellä ledin virta on 1.1 mA.

Optoerottimen transistorin läpi kulkevan virran suuruus riippuu transistorin virransiirtosuhteesta. Transistorin emitteriltä maihin on mitoitettu vastus, jonka yli jää kymmenesosa mitattavasta jännitteestä. Emitterillä olevan vastuksen yli jää siis 1.1 V esimerkin tapauksessa 110 mV, joka voidaan mitata mikrokontrollerin AD-muuntimella. [8]



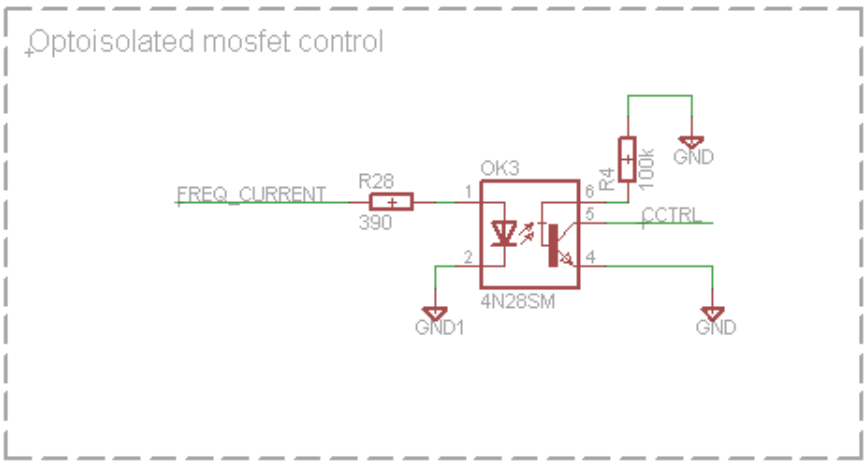
Kuva 11. Optoerotettu balansointivirran mittaus AD-muuntimella.

Flyback-hakkurin toisiojännitteen mittaaminen mikrokontrollerilla tapahtuu samalla tavoin, kuin balansointivirran mittaus. Erona on ainoastaan, että toisiojännitettä ei tarvitse vahvistaa vaan se puolitetaan vastusjaolla ja syötetään jännite-virtamuuntimeen. Jännite-virtamuunnin ohjaa sitten optoerotinta samalla tavoin, kuin virran mittaamisessa ja toisiojännite voidaan mitata AD-muuntimella optoerottimen transistorin emitteriltä.



Kuva 12. Optoerotettu toisiojännitteen mittaus AD-muuntimella.

Vakiovirtasäätimen MOSFET-transistorin ohjaussignaali on myös optoerotettu 4N28 optoerotinta käyttäen. Mikrokontrollerilla luodaan pulssinleveysmoduloitua kanttiaaltoa, jolla optoerottimen lediä ajetaan. Optoerottimen transistoria käytetään MOSFETin hilan kytkemiseen maihin ja optoerottimen transistorin kannalta maihin on kytketty 100 kΩ vastus, jotta transistorin nousu- ja laskuajat pienenevät.

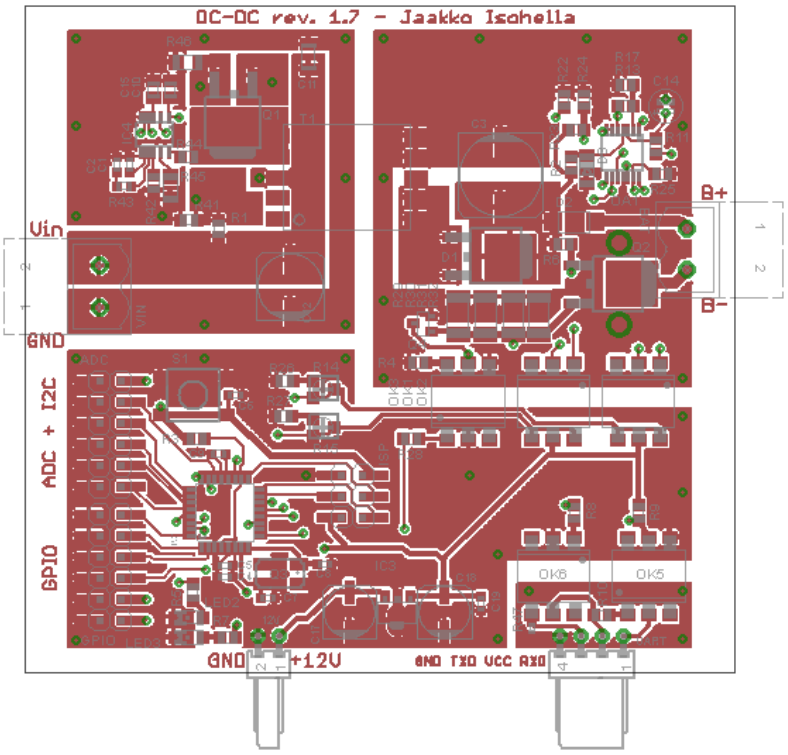


Kuva 13. Optoerotus vakiovirtasäätimen kytkimelle.

3.3 Piirilevy

Piirilevyn piirtämisen käytettiin Eagle PCB ohjelmiston 30-päivän freemium kokeiluversiota, joka mahdollisti halutun kaksikerroslevyn piirtämisen. Vaikka ilmaisia piirilevysuunnitteluohjelmistoja on olemassa suuri määrä, päädyttiin Eagle PCB ohjelmistoon siksi, että se on laajasti harrastelijoiden käytössä ja näin ollen komponenttikirjastoja ja tukea internetistä löytyy runsaasti. Kyseisestä ohjelmistoa on myös käytetty aikaisemmin, minkä takia se valittiin piirilevysuunnittelutyökaluksi tähän projektiin. [5]

Kuten edellisessä kappaleessa tuli ilmi, niin piirilevy piirrettiin kaksikerroksisena ja suunnittelussa käytettiin pääasiassa vain pintaliitoskomponentteja. Käytännön syistä liittimet valittiin läpiladottavina komponentteina. Pintaliitoskomponenttien valintaan vaikutti se, että laitteen piirilevystä haluttiin mahdollisimman pienikokoinen. Piirilevyn kooksi tulikin pintaliitoskomponenttien ansiosta 8.5cm x 8cm. Näin pientä piirilevyä ei läpiladottavilla komponenteilla olisi mitenkään saanut tehtyä. Piirilevyn pintapuolen piirilevykuva on kuvassa 13.

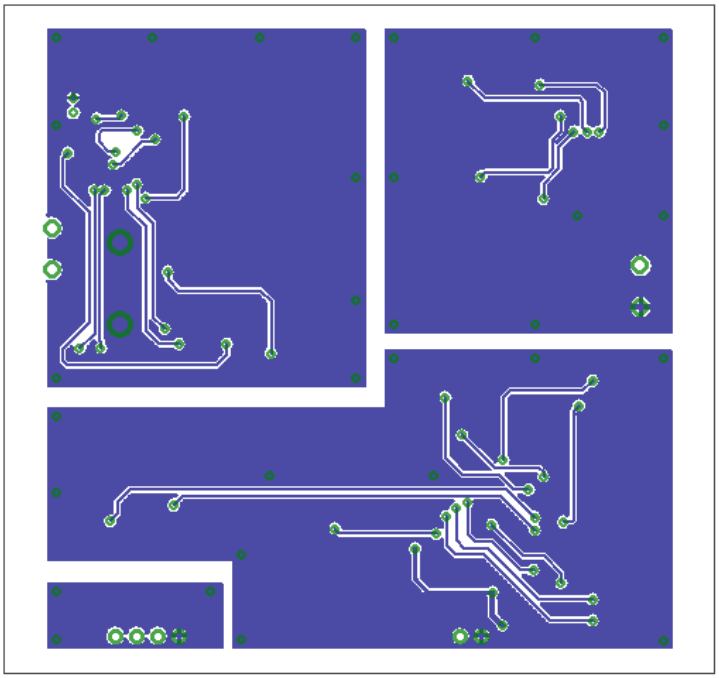


Kuva 14. Piirilevyn komponenttipuolen piirilevykuva.

Piirilevy rakennettiin kolmesta eri lohkosta. Kuvan 13 vasemmassa yläreunassa oleva lohko on flyback-hakkurin ensiöpuoli, missä sijaitsee hakkuripiiri, MOSFET-transistori ja hakkuripiirin passiivikomponentit. Kuvan 13 oikeassa yläreunassa on flyback-

hakkurin toisiopuoli komponentteineen. Toisiopuolella sijaitsee myös operaatiovahvistin, shuntvastus ja virtasäätimen komponentit. Ensiö ja toisiopuolen väliin sijoittuu hakkurin muuntaja. Piirilevyn alaosa on varattu kokonaan mikrokontrollerille ja sen oheiskomponenteille sekä sarjaväylän optoerotukselle.

Idea piirilevyn toteuttamiseen lohkomaisena lähti LT3748 hakkuripiirin datalehdestä, jossa oli esimerkki laitteen piirilevystä, jossa ensiö ja toisio olivat omissa lohkoissaan. Hakkurin piirilevykuvat suunniteltiin LT3748 datalehden innoittamana sillä periaatteella, että vältetään mahdollisimman paljon signaalivetoja ja käytetään ennemmin isoja pintoja. Jos signaalivetoja ei ole mahdollista välttää, niin suotavaa olisi, että signaalit vedettäisiin omassa kerroksessaan piirilevyn sisällä, jotta hakkurin aiheuttamat häiriöt eivät kytkeytyisi herkkiin signaaleihin. Flyback-hakkurin kellotaajuus on useita satoja kilohertsejä, joka on myös yksi syy suunnitella piirilevykuvat isoja pintoja käyttäen. Suuren kellotaajuuden takia hakkuriin liittyvät komponentit, myös hakkuripiirin oheiskomponentit, tulisi sijoittaa mahdollisimman lähelle piiriä, jotta toimintaan vaikuttavat häiriöt pienenevät. Piirilevyn täyttäminen yhtenäisellä pinnalla on muutenkin järkevää sillä se nopeuttaa levyn valmistusaikaa, kun levystä ei tarvitse poistaa kaikkea kuparia. Isot alueet toimivat myös maa-alueina ja häiriösuojina herkille komponenteille.



Kuva 15. Piirilevyn pohjapuolen piirilevykuva.

Kuvassa 14 on piirilevyn pohjapuolen piirilevykuva, mikä on aika tyypillinen piirilevykuva laitteelle, jossa komponentit on sijoitettu piirilevyn pintapuolelle. Pohjapuoli on käytännössä kokonaan maa-tason peitossa, mutta pintapuolen tilapuutteen takia joitakin signaaleja on jouduttu myös pohjapuolen kautta vetämään.

3.4 Kotelointi

Laitteen kotelointiin ei tämän työn osalta kiinnitetä hirveästi huomiota. Muutaman huomionarvoisen seikan kotelointiin kuitenkin esitetään. Koteloksi voidaan valita esimerkiksi muovi- tai alumiinikotelo, kunhan kotelo on tarpeeksi iso. Kotelon mittojen tulisi olla ainakin 120x90x40mm, jotta piirilevy sekä kotelon kylkeen tulevat liittimet ja sulake mahtuisivat hyvin koteloon sisälle. Metallikotelo olisi laitteen sijoituspaikan takia paras vaihtoehto juuri siksi, että se samalla vaimentaisi laitteen mahdollisesti aiheuttamia sähkömagneettisia häiriöitä ja olisi kestävämpi. Vaihtelevien sääolojen takia koteloinnin yhteydessä tulisi kiinnittää myös huomiota, kuinka kosteudenhallinta järjestetään, jotta laitteen toiminta ei vaarantuisi.

4 OHJELMISTOSUUNNITTELU

Laitteen ATmega mikrokontrollerin ohjelmisto päätettiin kirjoittaa C-kieliseksi modulaariseksi ohjelmaksi eli ohjelmiston eri osa-alueet kirjoitettiin omina ohjelmalohkoinaan. Ohjelmisto suunniteltiin Atmelin AVR Studio 4 kehitysympäristöllä, joka osaa kääntämisen ja linkkauksen yhteydessä nitoa eri ohjelmamoduulit yhteen. Ohjelmisto päätettiin tehdä modulaarisena, koska laite jää ammattikorkeakoululle sähköskootterin osaksi ja näin ollen modulaariseksi tehty ohjelmisto on helpompi ymmärtää ja sitä on helpompi muokata kuin täysin yhteen kirjoitettua koodia.

4.1 Pääohjelma

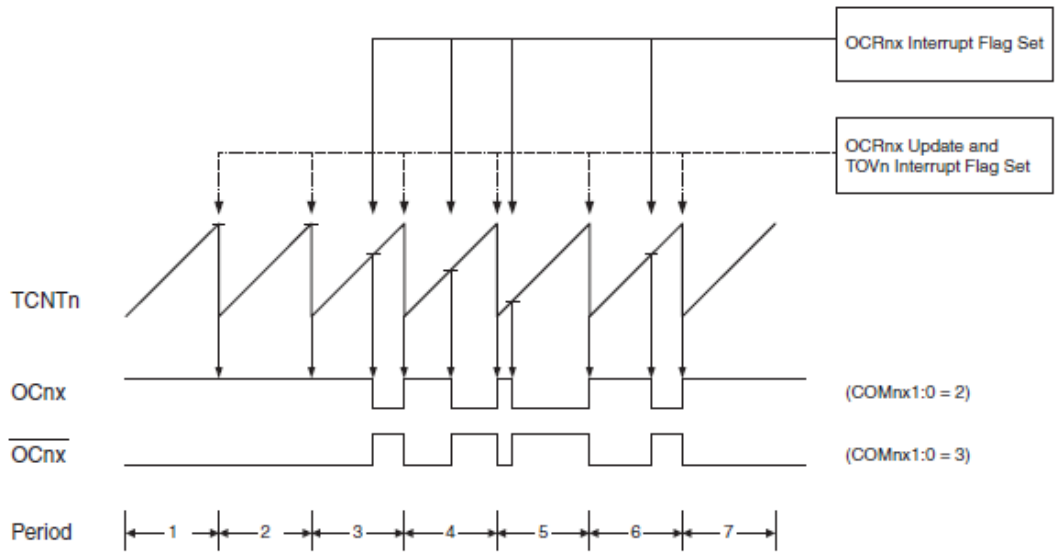
Laitteen pääohjelmassa yhdistetään kaikki erilliset moduulit toimimaan yhteisenä kokonaisuutena. Pääohjelman alussa alustetaan laitteisto ja mikrokontrollerin perusominaisuudet, kuten sisääntulot ja ulostulot. Pääohjelman alussa alustetaan myös lisälaitteet, kuten sarjaportti ja ajastimet, kutsumalla niiden alustusaliohjelmia. Tarvittavien alustusten jälkeen pääohjelma siirtyy loputtomaan silmukkaan, jossa tarkkaillaan onko sarjaväylä vastaanottanut komennon. Jos komento on vastaanotettu, kutsutaan aliohjelmaa, joka purkaa sarjaväylän puskurista komennon ja käsittelee sen. Silmukassa lasketaan myös toisiojännitteen ja balansointivirran keskiarvot, kun kymmenen mittausta on suoritettu. Lasketut keskiarvot sijoitetaan sitten omiin muuttujiinsa lukemista varten.

Pääohjelman suunnittelussa ideana oli, että pääohjelmaa pyritään pitämään mahdollisimman lyhyenä ja ohjelmiston pääasiallinen toiminnallisuus tehdään omina lohkoinaan. Tällä tavoin varmistettiin se, että pääohjelman rakenne pysyy siistinä ja uusien ominaisuuksien kehittämisessä on helppo kirjoittaa tarkistuksia ja toimintoja debuggausta varten pääohjelmaan. Liitteessä 1 on pääohjelman otsikkotiedosto ja C-kielinen ohjelmakoodi.

4.2 PWM-ohjaussignaali

Virtasäätimen ohjaussignaaliksi haluttiin 1 kHz pulssinleveysmoduloitu kanttiaalto. Ohjaussignaalin tuottamiseksi päätettiin käyttää mikrokontrollerin PWM-ominaisuutta. Ohjaussignaali tuotetaan mikrokontrollerin 8-bittisen ajastimen Fast PWM –ominaisuudella. Ajastimeksi valittiin TIMER0. Fast PWM tilassa ajastin laskee alhaalta

ylöspäin TOP-arvoon asti ja aloittaa sen jälkeen laskennan alusta. 8-bittisellä ajastimella TOP-arvon maksimiarvo on 255. Tuotetun signaalin taajuus riippuu ajastinmoduulin kellotaajuudesta ja TOP-arvosta. [7]



Kuva 16. Fast PWM –tilan aikakaavio. [7]

Kuvan 15 aikakaaviosta nähdään Fast PWM -tilan toiminta. Fast PWM -tilassa ajastin laskee alhaalta ylös ja vertaa kokoajan OCRnx rekisterin arvoa TCNTn laskuriin. Kun OCRnx:n arvo on sama kuin laskurin, OCnx portin ulostulo kääntyy. Kuvan 15 mukaan signaalin taajuuden määrää siis laskurin TOP-arvo ja pulssisuhteen OCRnx rekisterin arvo. Rekisterin ja laskurin nimessä n tarkoittaa ajastimen numeroa ja rekisterin nimessä x porttia. Esimerkiksi OCR1B olisi ajastin numero yhden B portin rekisteri.

Jotta saataisiin haluttu 1 kHz pulssileveysmoduloitu signaali tuotettua, täytyy 8-bittisen ajastimen esijakaja olla 64 ja TOP-arvo olla 124. Datalehden laskukaavalla PWM-signaalin taajuudeksi tulee näillä arvoilla

$$F_{PWM} = \frac{F_{CPU}}{N \cdot (1+TOP)} = \frac{8\,000\,000\ Hz}{64*(1+124)} = 1000\ Hz \quad (6)$$

josta F_{CPU} on mikrokontrollerin kellotaajuus ja N on esijakaja.

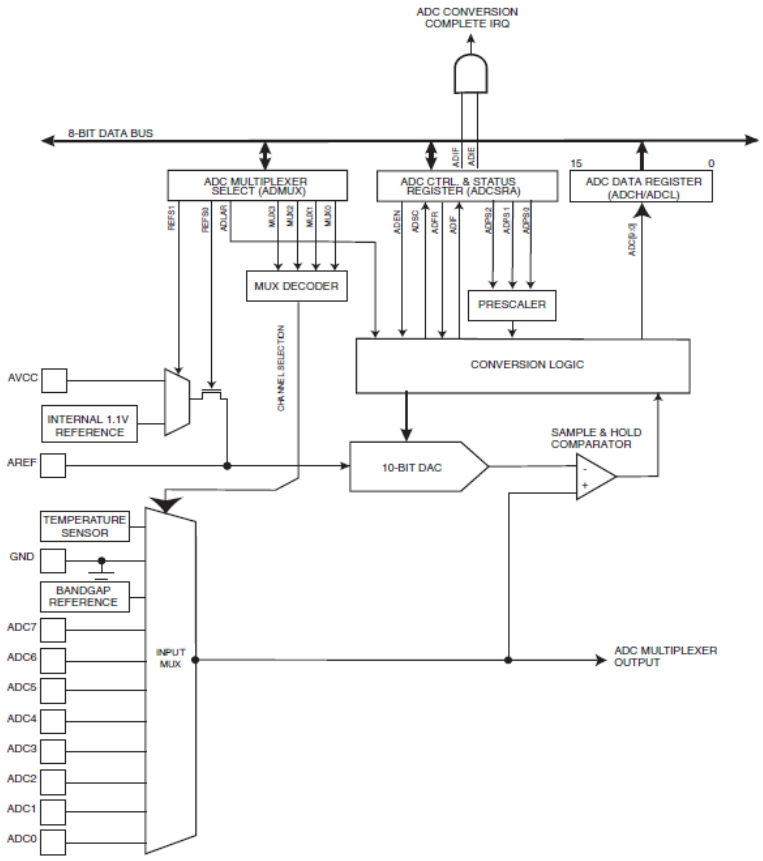
Mikrokontrollerin TIMER0 alustettiin Fast PWM -tilaan asettamalla TIMER0:n ohjausrekisterien WGMxx bittien arvoksi 7. Ajastimen ohjausrekistereillä on mahdollista valita kaksi Fast PWM -tilaa, toisen TOP-arvo on ennalta määritetty arvoon 255 ja toisen tilan TOP-arvo määritetään itse kirjoittamalla se OCR0A rekisteriin. Ajastimen esijakajaksi valittiin 64 asettamalla ajastimen ohjausrekisteri B:n CSxx

bittien arvoksi 3. Lopuksi asetetaan ohjausrekisteri A:n COM0B1 rekisteri ykköseksi, jotta PWM-signaalin muodostus käynnistyy ei-invertoivana. [7]

Mikrokontrolleri tuottaa nyt taustalla 1 kHz PWM-signaalia, jonka pulssisuhteen arvoa voidaan muuttaa kirjoittamalla OCR0B rekisteriin arvoja väliltä 0-124. Liitteessä 2 on PWM moduulin otsikkotiedosto ja C-kielinen ohjelmakoodi.

4.3 AD-muunnos

Flyback-hakkurin toisiojännitteen ja balansointivirran mittaamiseen käytettiin mikrokontrollerin 10-bittistä AD muunninta. Mikrokontrollerin AD-muuntimella on 8 multipleksattua sisääntuloa ja referenssijännitteeksi voidaan valita AD-muuntimen käyttöjännite tai sisäinen 1.1 V referenssijännite. AD-muuntimen piiristö tarvitsee 50-200 kHz kellotaajuuden toimiakseen luotettavasti 10-bittisenä. Muuntimen näytteenottotaajuutta voidaan parantaa kellotaajuutta korottamalla, mutta tällöin muunnostarkkuus putoaa alle 10-bittiin.[7]



Kuva 17. AD-muuntimen lohkokaavio. [7]

Mikrokontrollerin AD-muunnin voidaan käynnistää joko kertamittauksena tai jatkuvana mittauksena. AD-muunnin voidaan myös virittää aiheuttamaan keskeytys muunnoksen valmistuttua.

Virran- ja jännitteenmittaus päätettiin suorittaa 10 mittauksen keskiarvomittauksena. AD-muunnin asetettiin toimimaan kertamittauksena, jolloin se käynnistetään joka 10 ms välein mikrokontrollerin ajastimen avulla. Yhden mittauskierroksen aikana eli 10 ms välein mitataan kahden sisääntulon jännite. AD-muuntimen keskeytyksenpalvelu-aliohjelma hoitaa tuloksen talteenoton ja vaihtaa muuntimen sisääntuloa ja käynnistää uuden mittauksen. Kun 10 mittausta on suoritettu, pääohjelma laskee mittausten keskiarvot ja lataa ne tulosmuuttujiin myöhempää käyttöä varten. Liitteessä 3 on AD-muuntimen otsikkotiedosto ja C-kielinen ohjelmakoodi.

4.4 Ajastin 10ms keskeytykselle

Virran- ja toisiojännitteen keskiarvottamista varten valittiin mikrokontrollerin 8-bittinen Timer/Counter 2 -ajastin. Ajastimen tarkoituksena on aiheuttaa keskeytys noin 10ms välein, jolloin AD-muunnos käynnistetään. Ajastin voitaisiin asettaa toimimaan tavallisena laskurina ja pääohjelmassa sitten verrata sopivaa arvoa ajastimen laskuriin. Tämä tapa vie kuitenkin turhaan mikrokontrollerin resursseja ja tuo epätarkkuutta ajastimen toimintaan. Atmelin mikrokontrollerien ajastimissa on juuri tälläistä tarkoitusta varten CTC ominaisuus. CTC on lyhenne sanoista Clear Timer on Compare Match, mikä tarkoittaa sitä, että ajastin vertaa ennalta määritettyä arvoa laskurin arvoon ja mikäli arvot ovat samat, ajastin nollaa laskurin ja aloittaa alusta. Ajastin voidaan myös asettaa aiheuttamaan keskeytys, kun määritetty arvo on saavutettu.

Ajastin asetettiin CTC toimintatilaan kirjoittamalla TCCR2A rekisterin WGM21 bitti ykköseksi. Seuraavaksi asetettiin Timer 2:n TIMSK2 keskeytysrekisterin OCIE2A bitti ykköseksi, jotta vertailukeskeytys aktivoituu. Ajastimen OCR2A rekisterin vertailuarvo saadaan laskettua kaavalla 6. [7]

$$OCR_{val} = \left(\frac{F_{CPU}}{N} \cdot t_{int}\right) - 1 = \left(\frac{8\,000\,000\,Hz}{1024} \cdot 0.01\,s\right) - 1 = 77.125 \approx 77 \qquad (7)$$

josta F_{CPU} on mikrokontrollerin kellotaajuus, N esijakan arvo ja t_{int} keskeytyksen aikaväli.

Vertailuarvo jouduttiin pyöristämään lähimpään kokonaislukuun, koska mikrokontrollerin rekisterit ymmärtävät vain kokonaislukuja. Pyöristyksen takia keskeytys tapahtuu 9.856 ms välein. Tämä ei kuitenkaan tässä tarkoituksessa haittaa, sillä keskeytyksen tarkoitus on ainoastaan käynnistää AD-muunnos tasaisin väliajoin. Jos keskeytysajaksi haluttaisiin tarkasti 10ms aika, valittaisiin esijakajan arvoksi semmoinen arvo millä laskukaava antaisi kokonaisluvun vastaukseksi. Tässä tapauksessa esijakajaksi päätettiin kuitenkin valita 1024, jotta keskeytys voitaisiin luoda 8-bittistä ajastinta käyttäen. Jos esijakajaksi olisi valittu pienempi kuin 1024, ei vertailuarvo olisi mahtunut enää 8-bittisen ajastimen rekisteriin ja olisi jouduttu käyttämään 16-bittistä ajastinta sen sijaan. Keskeytyksen luominen 16-bittisellä ajastimella olisi toteutettu samalla tavalla kuin 8-bittisellä ajastimella. Keskeytyksen luontiin päätettiin valita kuitenkin 8-bittinen ajastin.

Kun ajastimen toimintatila ja keskeytykset on valittu sekä vertailuarvo asetettu, asetetaan vielä ajastimen TCCR2B rekisterin CS22:0 bitit ykköseksi, jotta esijakajan arvo 1024 on valittu ja ajastin käynnistyy. Jotta ajastimen keskeytykset toimisivat, täytyy pääohjelmassa vielä sallia keskeytykset komennolla **sei()**. Liitteessä 4 on ajastimen otsikkotiedosto ja C-kielinen ohjelmakoodi.

4.5 Sarjaväylä

Sarjaväylän muodostukseen käytettiin mikrokontrollerin USART-lisälaitetta. Mikrokontrollerin USART-lisälaite voidaan asettaa toimimaan joko synkronisena tai asynkronisena. Näiden kahden tilan erona on se, että synkronisessa tilassa datasiirto on synkronoitu ulkoisen kellosignaalin mukaan, kun asynkronisessa tilassa datasiirto on synkronoitu aloitus- ja lopetusbittien mukaan.

Sarjaväylä toteutettiin asynkronisena ja kehysmuodoksi valittiin 8:1:1 eli 8 databittiä, 1 aloitus- ja 1 lopetusbitti. Sarjaväylän nopeudeksi valittiin 9600 Baud ja sarjaväylä asetettiin aiheuttamaan keskeytys, kun yksi tavu on vastaanotettu. Sarjaväylää varten luotiin myös FIFO-puskuri **uartRxBuffer**, johon vastaanotettu tavu tallennetaan. Puskuri tarvitaan, jotta laitetta voidaan ohjata sarjaväylää pitkin käyttämällä selkokielisiä komentoja. [7]

Sarjaväylä alustetaan mikrokontrollerissa kutsumalla aliohjelmaa **uartInit(BAUD)**, jossa BAUD on haluttu sarjaväylän nopeus. Aliohjelman alussa lasketaan

mikrokontrollerin UBRR rekisteriin ladattava sarjaväylän nopeutta vastaava arvo BaudRate, joka lasketaan kaavalla

$$BaudRate = \frac{F_{CPU}}{(16 \cdot BAUD)} - 1 = \frac{8\,000\,000\,Hz}{(16 \cdot 9600)} - 1 = 51 \quad (8)$$

Laskettu BaudRaten arvo ladataan USART-moduulin UBRR rekisteriin, alustetaan FIFO-puskurin alku ja loppupisteet nolliksi. Myös USART-moduulin lähetys ja vastaanotto sekä keskeytys, kun vastaanotto on valmis, aktivoidaan. Kehysmuotoa ei tarvitse erikseen asettaa, koska mikrokontrollerin sarjamoduuli on vakiona asetettu 8:1:1 muotoon. Vastaanoton keskeytyspalvelualiohjelmassa tallennetaan vastaanotettu tavu FIFO-puskurin etuosaan ja kasvatetaan puskurin osoitinta yhdellä.

Sarjaväylän toteutukseen kirjoitettiin alustuksen ja keskeytyspalvelualiohjelman lisäksi muutama muu aliohjelma. Nämä aliohjelmat ovat datan luku puskurista, datan tyhjäys puskurista tiettyyn merkkiin asti (tämä merkki poistetaan myös), datan lähetys ja datataulukon lähetys. Sarjaväylän toteutukseen kirjoitettiin myös aliohjelma, joka lukee ohjelmamuistista vakiolauseita ja lähettää ne sarjaväylää pitkin. Vakiolauseet päätettiin tallentaa ohjelmamuistiin, jotta ne eivät veisi turhaa tilaa SRAM-muistista. Liitteessä 5 on sarjaväylän otsikkotiedosto ja C-kielinen ohjelmakoodi.

4.6 Ohjauskomennon käsittely

Laitteen ohjauskomentojen erottelu ja käsittely rakennettiin omaksi ohjelmamoduulikseen. Kuten sarjaväylän esittelyssä tuli ilmi, komennot vastaanotetaan sarjaväylältä merkki kerrallaan ja merkit tallennetaan erilliseen sarjapuskuriin. Ohjauskomennonkäsittelyaliohjelma lukee sarjapuskurista merkkejä taulukkoon kunnes merkki on erotusmerkki. Erotusmerkeiksi on valittu piste, yhtä kuin ja tähti. Kun erotusmerkki saavutetaan, taulukkoon luettua merkkijonoa verrataan ennalta määritettyyn merkkijonoon, joka määrää mitä toimintoja ohjelmassa suoritetaan.

Ohjauskomentojenkäsittelyaliohjelma **CmdParse()** hakee aluksi sarjapuskurista merkkijonon erotusmerkkiin asti ja tarkastaa oliko merkkijono ’BMS’, joka kertoo mihin laitteen toimintoon komento halutaan kohdistaa. Seuraavana haetaan sarjapuskurista merkkijono, joka määrää käytettävän toiminnon. Jos syötettyä komentoa ei löydy tai komennon joku osa on väärä, ohjelma lähettää sarjaväylälle lauseen ’FALSE COMMAND’. Laitteen ohjauskomennot ovat luotu GSM AT komentosarjan kaltaisiksi. Taulukko 1 sisältää laitteen ohjauskomennot. [10]

Taulukko 1. Laitteen komentotaulukko. [10]

Command sentence	Description
BMS.ON*	Turn DC-DC converter on.
BMS.OFF*	Turn DC-DC converter off.
BMS.SETI=XXXX*	Set charging current. XXXX is current in mA.
BMS.READ*	Read secondary voltage and charging current.

Merkkijonon lukemista varten luotiin ohjelmaan tietue, joka sisältää muuttujat **c[8]**, **length** ja **is.last**. Tietue luotiin, koska merkkijono luetaan monta kertaa ja tietuetta käyttämällä ei tarvitse muuttujia esitellä erikseen joka kerta. Tietueen käyttö mahdollistaa myös merkkijonon pituuden tallentamisen sekä tiedon siitä onko vastaanotettu merkkijono komentolauseen viimeinen. Tietuetta käytetään merkkijonon vastaanoton lisäksi aliohjelman tyyppinä, jotta aliohjelma palauttaa tietueen päättyessään.

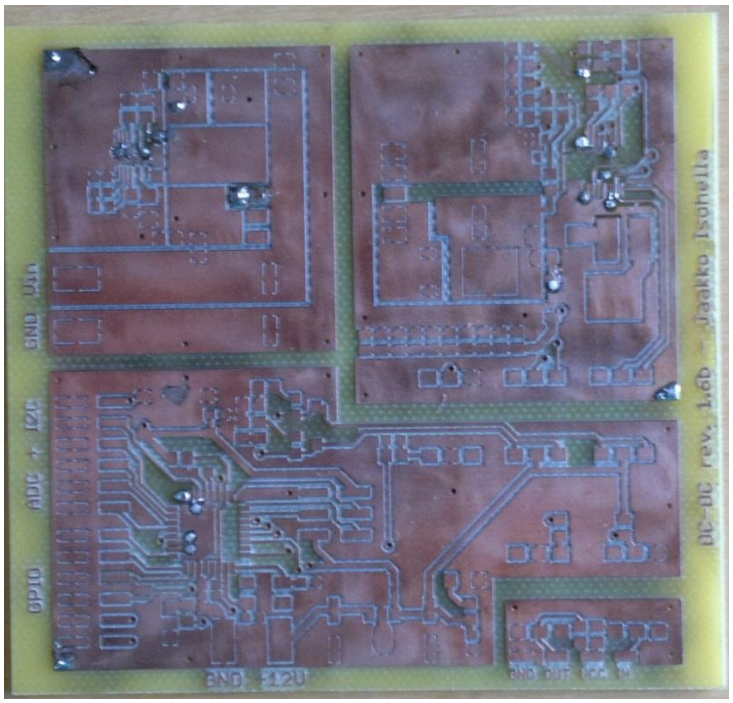
Ohjauskomennonkäsittelyaliohjelmaa voidaan helposti laajentaa uusille komennoille ja toiminnoille lisäämällä IF-lauseita. Komennonkäsittely päätettiin luoda lausetyyppiseksi yksittäisten merkkien sijaan, koska haluttiin, että komennot olisivat selkokielisiä. Selkokieliset komennot helpottavat myös laitteen ohjaamista esimerkiksi esittelytarkoituksissa. Liitteessä 6 on ohjauskomennonkäsittelyaliohjelman otsikkotiedosto ja C-kielinen ohjelmakoodi.

5 TOTEUTUS

Laitteesta tehtiin kaksipuoleinen piirilevy, johon laitteen komponentit juotettiin. Alkuperäisenä ideana oli, että tehtävä piirilevy tulee olemaan lopullinen virheetön versio. Piirilevyä kalustaessa ja testatessa huomattiin suunnittelusta kuitenkin muutamia virheitä ja puutteita, joita suunnitteluvaiheessa ei oltu osattu ottaa huomioon. Virheet ja puutteet olivat kuitenkin sen verran pieniä, että täysin kalustettua piirilevyä ei tarvinnut tehdä uudestaan, vaan pienillä muutoksilla piirilevyn viat saatiin korjattua ja puutteet lisättyä. Piirilevy päätettiinkin pitää prototyyppipiirilevynä, johon voitaisiin tehdä erilaisia muutoksia ja kokeiluja.

5.1 Prototyyppipiirilevy

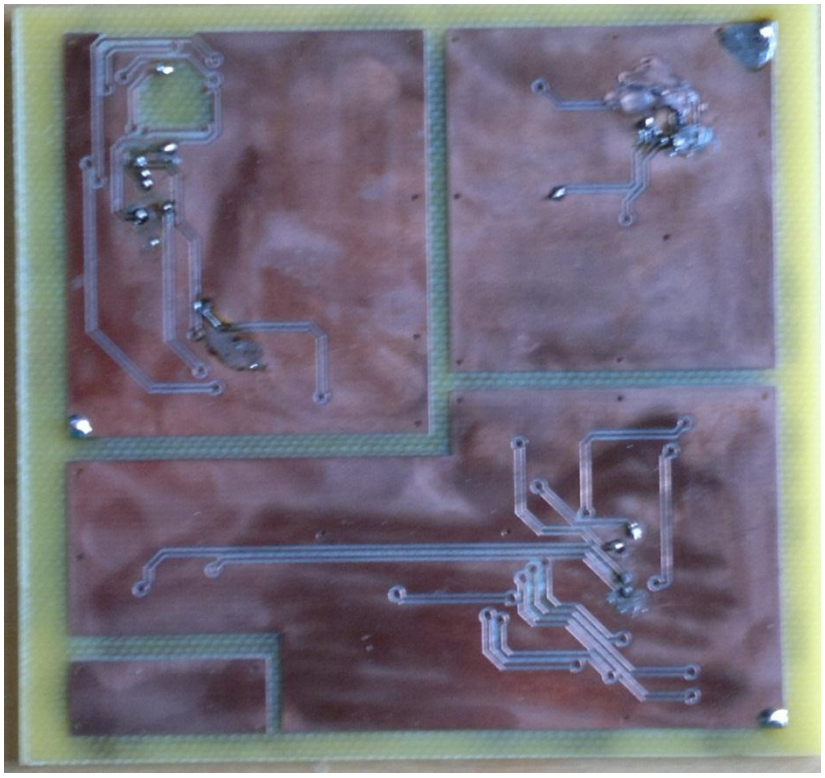
Prototyyppipiirilevy oli aluksi tarkoitus valmistaa valotus-syövytys menetelmällä ammattikorkeakoulun tiloissa. Projektin aikana tuli kuitenkin mahdollisuus valmistaa piirilevy jyrsimällä. Tämä oli paljon parempi ratkaisu sillä muutaman pintaliitoskomponentin jalkojen väli on vain 6 milsiä eli 0.15 mm. Noin tarkkaa jälkeä perinteisellä valotus-syövytys menetelmällä ei olisi saatu aikaan. Jyrsiminen oli siis ainut järkevä vaihtoehto. Piirilevyn olisi voinut myös teettää piirilevyvalmistajalla, jolloin yhden levyn hinta olisi ollut lähemmäs sata euroa juuri tuon tarkkuuden takia.



Kuva 18. Prototyyppipiirilevyn pintapuoli.

Prototyyppipiirilevy valmistettiin Tampereen Hervannassa Nokia Oyj:n tiloissa. Piirilevyn valmistukseen käytettiin piirilevyjyrsintä, jolle piirilevykuvat vietiin gerber tiedostoina. Jyrsin aloitti operaation jyrsimällä kapeimmat juotostäplien välit ja siirtyi siitä aina leveämpään. Viimeisimpänä jyrsin jyrsi levyn ulkoalueet ja porasi levyn reiät. Koska piirilevy toteutettiin kaksipuoleisena täytyi puolenvaihto suorittaa manuaalisesti. Jyrsinohjelma keskeytti jyrsimen ja ilmoitti selkeästi, milloin jyrsittävä piirilevy tuli kääntää ympäri.

Levyn tekeminen onnistui muuten ihan hyvin, mutta reikiä poratessa pohjapuolen reiät eivät sattuneet kohdalleen, koska pohjapuolen asemointi ei ollut ihan tarkalleen oikein. Vaikka jyrsinohjelma mittaa kameralla työstettävästä levystä kohdistuspisteet ja jyrsii ja poraa niiden mukaan, on ohjelmassa jokin toleranssi tai vika, minkä takia kaksipuoleisia levyjä tehdessä pinta- ja pohjatasojen kohdistus saattaa olla hiukan pielessä. Pieleen menneestä kohdistuksesta huolimatta prototyyppipiirilevy oli todella onnistunut. Jyrsinnän jälkeen oli edessä hapettumien puhdistus piirilevyn kuparipinnasta pienoisporakoneen terästharjalla ja komponenttien kiinnitys piirilevylle.



Kuva 19. Prototyyppipiirilevyn pohjapuoli.

5.2 Komponenttien kiinnitys

Komponentit kiinnitettiin prototyyppipiirilevyyn manuaalisesti mikroskooppia ja pintaliitostyökaluja käyttäen. Ensimmäisenä piirilevyyn juotettiin hakkurin osat ja mikrokontrollerin regulaattorivirtalähde, jotta niiden toiminta voitaisiin varmistaa ennen kriittisempien komponenttien kiinnitystä. Komponenttien kiinnitys oli mikroskoopin ansiosta melko helppoa. Myös hyvät työkalut ja pintaliitoskomponenttien juottamiseen tarkoitetut kolvin kärjet helpottivat työskentelyä huomattavasti. Myös laadukas juotosvesi helpotti työskentelyä, koska juotosveden avulla tina tarttui piirilevyn kupariin paljon helpommin.

Komponenttien kiinnitys ei ollut piirilevystä johtuen siistiä työtä, koska piirilevyllä oli niin paljon paljasta kuparia, että pienikin käsien tärinä aiheutti tinan tarttumisen piirilevyllä väärään kohtaan. Tähän ongelmaan olisi ollut ratkaisuna soldermask, joka on piirilevyn pintaan maalattu kerros, jossa vain juotostäplät ovat paljasta kuparia. Tällöin juotostina tarttuu ainoastaan haluttuihin kohtiin ja ylimääräisiltä oikosuluilta vältytään. Piirilevyltä jouduttiinkin juottamisen jälkeen poistamaan ylimääräinen juotostina tinaimusukkaa käyttäen ja mittaamaan piirilevy yleismittarilla mahdollisten oikosulkujen takia. Viimeisenä piirilevyn kaikkien signaalivetojen resistanssi mitattiin maatasoa vasten, jotta varmistuttiin, että piirilevyllä ei ole tinasiltoja aiheuttamassa oikosulkuja. Myös signaaleiden väliset resistanssi mitattiin, jotta signaalit eivät olisi keskenään oikosulussa.

6 YHTEENVETO

Työn tavoitteena oli suunnitella isomman laitekokonaisuuden yksi pienempi osa. Laitteelle ei ollut projektin kannalta määritelty muita ominaisuuksia, kuin ohjattavuus sarjaväylän kautta. Laitteeseen pyrittiin kuitenkin sisällyttämään sellaisia ominaisuuksia, jotka tukevat monipuolisen BMS-järjestelmän toimintaa. Laitteen toimintaperiaatteen hahmottelu oli melko helppoa, mutta jännitteen- ja virranmittaus optoerotettuina toivat huomattavasti lisää haastetta.

Laitteen ominaisuuksien suunnittelu ei ollut mitenkään yksiselitteistä ja laitteen ominaisuudet muuttuivatkin muutamaan kertaan suunnittelun aikana. Laitteessa on kuitenkin vielä paljon kapasiteettia jäljellä, ja potentiaalia uusien ominaisuuksien kehittämiselle on runsaasti. Alkuperäisenä ideana oli, että laite mittaisi myös ympäristön lämpötilaa ja kosteutta, mutta tämä ominaisuus päätettiin jättää toistaiseksi toteuttamatta.

Laitteen suunnittelu vei paljon aikaa, koska pelkästään laitteen elektroniikan suunnittelu oli erilaisten hakkurikytkentöjen takia aikaa vievää. Myös suunnittelun aikana tulleiden virheiden korjaaminen ja testaaminen on vienyt monta miestyötuntia. Monet suunnitteluvirheet olisi voitu välttää, mikäli elektroniikan suunnittelusta, erityisesti optoerottimista, olisi ollut enemmän kokemusta.

Työ on tehtyjen virheiden ja laajuutensa takia kuitenkin opettanut paljon sekä elektroniikan että ohjelmiston suunnittelusta. Kaiken kaikkiaan opinnäytetyö oli hyvä kokemus monimutkaisemman laitteen suunnittelusta. Opinnäytetyön vuoksi osataan myös tulevaisuudessa välttää yleisimpiä virheitä, joita tätä työtä tehdessä on tehty.

LÄHTEET

1 Linear Technology – LTspice IV. [online][viitattu 5.3.2012]
http://www.linear.com/designtools/software/
2 LT3748 datalehti. [online][viitattu 5.3.2012]
http://cds.linear.com/docs/Datasheet/3748fa.pdf
3 IRFR4620PbF datalehti. [online][viitattu 5.3.2012]
http://www.irf.com/product-info/datasheets/data/irfr4620pbf.pdf
4 VS-12CWQ10FNPbF datalehti. [online][viitattu 5.3.2012]
http://www.vishay.com/docs/94135/12cwq10f.pdf
5 Cadsoft Eagle PCB. [online][viitattu 5.3.2012]
http://www.cadsoftusa.com/
6 TI 78L05 datalehti. [online][viitattu 5.3.2012]
http://www.ti.com/lit/ds/slvs010s/slvs010s.pdf
7 ATmega48A/88A/168A/328A datalehti. [online][viitattu 5.3.2012]
http://www.atmel.com/Images/doc8271.pdf
8 Vishay 4N28 optoerottimen datalehti. [online][viitattu 5.3.2012]
http://www.vishay.com/docs/83725/4n25.pdf
9 Microchip MCP6L4 datalehti. [online][viitattu 5.3.2012]
http://ww1.microchip.com/downloads/en/DeviceDoc/22135a.pdf
10 GSM AT Komentosarja. [online][viitattu 5.3.2012]
http://www.zeeman.de/wp-content/uploads/2007/09/ubinetics-at-command-set.pdf
11 Flyback-hakkurin työvaiheet. [online][viitattu 5.3.2012]
http://en.wikipedia.org/wiki/File:Flyback_operating.svg

LIITTEET

Liite 1. Pääohjelman otsikkotiedostot ja C-kielinen ohjelmatiedosto, 2 sivua.

Liite 2. PWM-moduulin otsikkotiedosto ja C-kielinen ohjelmatiedosto, 1 sivu.

Liite 3. AD-muunnoksen otsikkotiedosto ja C-kielinen ohjelmatiedosto, 2 sivua.

Liite 4. Ajastimen otsikkotiedosto ja C-kielinen ohjelmatiedosto, 1 sivu.

Liite 5. Sarjaväylän otsikkotiedosto ja C-kielinen ohjelmatiedosto, 3 sivua.

Liite 6. Komennonkäsittelyaliohjelmann otsikkotiedosto ja C-kielinen ohjelmatiedosto, 3 sivua.

Liite 1. Pääohjelman otsikkotiedostot ja C-kielinen ohjelmatiedosto.

main.h

```
#ifndef MAIN_H
#define MAIN_H

// includes

#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/interrupt.h>

#include "uart.h"
#include "command.h"
#include "adc.h"
#include "pwm.h"
#include "timer.h"
#include "global.h"

#endif
```

global.h

```
#ifndef GLOBAL_H
#define GLOBAL_H

#define F_CPU 8000000UL

// variables

extern volatile unsigned char CmdRdy, on_off, avg_count, avg_on;
extern volatile unsigned int adc_c[15], adc_v[15], setc, halfs_counter;
extern volatile unsigned int voltage, current;

#endif
```

main.c

```
/*************************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: Main program. Initialize peripherals
//        and calculate adc averages. Control PWM
//        duty cycle.
//
*************************************************/

#include "main.h"

volatile unsigned int voltage=0, current=1000;

int main()
{
    CmdRdy = 0;
```

```
    on_off = 0;
    avg_on = 0;
    uartInit(9600);    // initialize uart @ 9600baud
    adc_init();      // initialize ADC
    pwm_init();        // initialize PWM
    interval_interrupt_init();  //initialize timer interrupt

    OCR0B = 62;// 62/125 = 50% Duty Cycle. Fet gate duty inverted.

    sei();

    while (1)
    {
       /* check if command received. */
       if(CmdRdy == 1)
       {
          CmdParse();  // parse command
          CmdRdy = 0;
       }

       /* calculate ADC average. 10 measurement average. */
       if(avg_count >= 11 && avg_on == 1)
       {
          int count;
          unsigned int v=0,c=0;
          for(count=1; count<11; count++)
          {
             v += adc_v[count];
             c += adc_c[count];
          }
          v = v / 10;
          c = c / 10;

          voltage = (long)v*1100/1024*10;
          current = (long)c*1100/1024*10;

          avg_count = 0;
       }

       /* change PWM duty cycle if current is lower
          or higher than desired current. Simple way. Should
          be written to be more intelligent.*/
       if(halfs_counter >= 50 && current < (setc-100))
       {
          OCR0B -= 1;
          halfs_counter = 0;
       }
       else if(halfs_counter >= 50 && current > (setc+100))
       {
          OCR0B += 1;
          halfs_counter = 0;
       }
    }
    return 0;
}
```

Liite 2. PWM-moduulin otsikkotiedosto ja C-kielinen ohjelmatiedosto.

pwm.h

```
#ifndef PWM_H
#define PWM_H

// includes

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

// functions

void pwm_init();

#endif
```

pwm.c

```
/*********************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: Initialize 1kHz PWM signal generation.
//
*********************************************/

#include "pwm.h"

/* initialize PWM module. */
void pwm_init()
{
    /* Fast PWM mode. OCR0A sets counter TOP.
    Prescaler set to 64. non-inverting*/
    TCCR0A |= (1<<WGM01)|(1<<WGM00)|(1<<COM0B1);
    TCCR0B |= (1<<WGM02)|(1<<CS01)|(1<<CS00);

    OCR0A = 125;

    DDRD|=(1<<PD5); // set PD5(OC0B) as output.
}
```

Liite 3. AD-muunnoksen otsikkotiedosto ja C-kielinen ohjelmatiedosto.

adc.h

```
#ifndef ADC_H
#define ADC_H

// includes

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "global.h"

// variables

#define  ADC_CHANNELS      2

// functions

void adc_init();
void adc_string(unsigned int, char *);

#endif
```

adc.c

```
/*********************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: ADC initialize and interrupt.
//
*********************************************/

#include "adc.h"

volatile unsigned int adc_c[15], adc_v[15];
volatile unsigned char avg_count=0;
unsigned char adc_index = 0;

/* initialize ADC module. */
void adc_init()
{
    /* set adc prescaler to 64. Sample rate 125kHz @ 8MHz. */
    ADCSRA |= (1<<ADPS2)|(1<<ADPS1);

    ADMUX |= (1<<REFS0)|(1<<REFS1); // internal 1.1V as reference

    ADCSRA |= (1<<ADEN);  // enable adc

    ADCSRA |= (1<<ADIE);  // enable interrupt on conversion ready
}

/* convert int to string */
void adc_string(unsigned int adcvalue, char *string)
{
    itoa(adcvalue,string,10);
}
```

```
/* adc interrupt routine */
ISR(ADC_vect)
{
    /* read ad conversion result */
    if(adc_index == 0) // secondary voltage measurement
    {
        adc_v[avg_count] = ADCL;
        adc_v[avg_count] += (ADCH<<8);
        ADMUX = (1<<REFS0)|(1<<REFS1)|adc_index; // change ADC input
        ADCSRA |= (1<<ADSC);

    }
    else if (adc_index == 1) // charge current measurement
    {
        adc_c[avg_count] = ADCL;
        adc_c[avg_count] += (ADCH<<8);
        ADMUX = (1<<REFS0)|(1<<REFS1)|adc_index; // change ADC input
        avg_count++;
    }

    /* select next adc input. */
    if (++adc_index >= ADC_CHANNELS)
    {
        adc_index=0;
    }
}
```

Liite 4. Ajastimen otsikkotiedosto ja C-kielinen ohjelmatiedosto.

timer.h

```
#ifndef TIMER_H
#define TIMER_H

// includes

#include <avr/io.h>
#include <avr/interrupt.h>
#include "global.h"

//functions

void interval_interrupt_init();

#endif
```

timer.c

```
/*********************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: Initialize ~10ms timer interrupt.
//
*********************************************/

#include "timer.h"

volatile unsigned int halfs_counter=0;

/* 10ms interval interrupt. Must be initialized at last */
void interval_interrupt_init()
{
   TCCR2A |= (1<<WGM21)|(1<<COM2B0);   // Timer on CTC mode, toggle OC2B on compare match
   TIMSK2 |= (1<<OCIE2B);              // enable interrupt.

   OCR2A = 78;// interrupt 9.98ms @ 8MHz.

   DDRD|=(1<<PD3); // set PD3(OC2B) as output.

   /* start timer. Prescaler 1024 */
   TCCR2B |= (1<<CS20)|(1<<CS21)|(1<<CS22);
}

/* timer interrupt routine */
ISR(TIMER2_COMPA_vect)
{
   halfs_counter++;
   if(avg_count < 11 && avg_on == 1)
   {
      ADCSRA |= (1<<ADSC); //start AD conversion
   }
}
```

Liite 5. Sarjaväylän otsikkotiedosto ja C-kielinen ohjelmatiedosto.

uart.h

```
#ifndef UART_H
#define UART_H

// includes

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#include "global.h"

// variables

#define uartRxBufferSize 150

// functions

void uartInit(unsigned long);
unsigned char uartRx(unsigned char *);
void uartSkipTo(unsigned char);
void uartTx(unsigned char);
void uartTxString(const unsigned char *);
void Print_Msg_P(const char *);

#endif
```

uart.c

```
/*********************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: Initialize UART module and uartbuffer.
//              Functions for TX, RX and buffer operations.
//
*********************************************/

#include "uart.h"

volatile unsigned char CmdRdy;
volatile unsigned char uartRxBuffer[uartRxBufferSize];
volatile unsigned char uartRxBufferHead;
volatile unsigned char uartRxBufferTail;

/* initialize serial port */
void uartInit(unsigned long Baud)
{
    /* calculate baudrate */
    unsigned int BaudRate = F_CPU / (16 * Baud) - 1;

    /* set BaudRate to registers UBRR0H and UBRR0L */
    UBRR0H = (unsigned char) (BaudRate>>8);
    UBRR0L = (unsigned char) BaudRate;
```

```
    UCSR0B = 0; // reset UART

    uartRxBufferHead = 0; // reset buffer
    uartRxBufferTail = 0; // reset buffer

    DDRD|=(1<<PD1); // set PD1(TX) as output.

    /* enable RX, TX and interrupt */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
}

/* RX interrupt routine. read data to buffer and increase buffer pointer */
ISR(USART_RX_vect)
{
    unsigned char data = UDR0; // read data
    /* check buffer location */
    unsigned char temp = (uartRxBufferHead + 1) % uartRxBufferSize;

    /* command fully received if data is * */
    if(data == '*') CmdRdy = 1;
    if(temp != uartRxBufferTail) // if not overflow
    {
        uartRxBuffer[uartRxBufferHead] = data; // store in buffer
        uartRxBufferHead = temp; // increase buffer pointer
    }
}

/* read data from buffer, return buffer size before reading */
unsigned char uartRx(unsigned char *bufSize)
{
    /* if buffer empty, return 0 */
    if(uartRxBufferHead == uartRxBufferTail)
    {
        *bufSize = 0;
        return 0;
    }
    else
    {
        *bufSize = (uartRxBufferSize + uartRxBufferHead - uartRxBufferTail) % uartRxBufferSize;
        unsigned char data = uartRxBuffer[uartRxBufferTail]; // read from buffer
        uartRxBufferTail = (uartRxBufferTail + 1) % uartRxBufferSize; // advance buffer tail pointer
        return data; // return data
    }
}

/* clear buffer until c (c will be cleared from buffer too) */
void uartSkipTo(unsigned char c)
{
    unsigned char d;
    unsigned char b;
    do { d = uartRx ( &b );} while ( b == 0 || d != c);
}

/* waits for transmitter to not be busy then tx a byte */
void uartTx(unsigned char data)
{
    if(data == '\n') uartTx('\r');
    /* wait while TX is not busy */
    loop_until_bit_is_set(UCSR0A, UDRE0);
    UDR0 = data; // transmit
}
```

```
/* transmit string through uart */
void uartTxString(const unsigned char *str)
{
    while(*str)
    {
        uartTx(*str);
        str++;
    }
}

/* print message from flash to uart */
void Print_Msg_P(const char *msg)
{
    unsigned char c;

    while ((c = pgm_read_byte(msg++))) // while c != NULL
    uartTx(c);
}
```

Liite 6. Komentoaliohjelman otsikkotiedosto ja C-kielinen ohjelmatiedosto.

command.h

```
#ifndef COMMAND_H
#define COMMAND_H

// includes

#include "uart.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "global.h"
#include "adc.h"

// variables

typedef struct _CmdWord
{
	unsigned char c[15];
	unsigned char length;
	unsigned char isLast;
} CmdWord;

// functions

void CmdParse();

#endif
```

command.c

```
/*********************************************
//
// HARDWARE: ATmega128
// COMPILER: WinAVR + AVRStudio4
// DATE: 2.3.2012
// AUTHOR: Jaakko Isohella
// DESCRIPTION: UART Command reading and parsing
//              from buffer.
//
*********************************************/

#include "command.h"

volatile unsigned char on_off, avg_on;
volatile unsigned int setc=1000;

/* stores a string until a +, = or * is found */
CmdWord ReadCmd()
{
   CmdWord result;

   result.length = 0;

   unsigned char bytesAvail;
   unsigned char data = 0;
```

```
    while (data != '=' && data != '*' && data != '.')
    {
        /* read a byte from uart buffer and store */
        do { data = uartRx(&bytesAvail); } while (bytesAvail == 0);

        if (data != '='  && data != '*' && data != '.')
        {
            /* add to c if not delimiter */
            result.c [result.length] = data;
            result.length += 1;
        }
    }
    result.c [result.length] = 0; // null terminate

    /* if * found, mark string as last in sentence */
    if (data == '*') result.isLast = 1; else result.isLast = 0;

    return result;
}

void CmdParse()
{
    CmdWord CmdWord_;

    char string[5];

    CmdWord_ = ReadCmd();

    if(strcmp((char *)&CmdWord_.c, "BMS") == 0)
    {

        CmdWord_ = ReadCmd(); // read command header

        /* see which command it is and process it */
        if (strcmp((char *)&CmdWord_.c, "ON") == 0) // turn device on
        {
            if(on_off == 0)
            {
                ADCSRA |= (1<<ADSC);  // Start AD Conversion

                /* const progmem string. Uses Flash only. Literal string
                   would use both SRAM and Flash. Would waste SRAM. */
                Print_Msg_P(PSTR("ON.OK\n"));
                on_off = 1;
                avg_on = 1;

            }
            else
            {
            Print_Msg_P(PSTR("Already.ON\n"));
            }
        }
        else if(strcmp((char *)&CmdWord_.c, "OFF") == 0) // turn device off
        {
            if(on_off == 1)
            {
                Print_Msg_P(PSTR("OFF.OK\n"));
                on_off = 0;
                avg_on = 0;
            }
            else
            {
                Print_Msg_P(PSTR("Already.OFF\n"));
```

```
            }
        }
        else if(strcmp((char *)&CmdWord_.c, "SETI") == 0) // set balancing current
        {
            CmdWord_ = ReadCmd();

            if(on_off == 1)
            {
                /* convert string to int */
                setc = (CmdWord_.c[0] - '0')*1000
                     + (CmdWord_.c[1] - '0')*100
                     + (CmdWord_.c[2] - '0')*10
                     + (CmdWord_.c[3] - '0');

                Print_Msg_P(PSTR("SETI="));
                uartTxString(CmdWord_.c);
                Print_Msg_P(PSTR(".OK\n"));
            }
            else
            {
                Print_Msg_P(PSTR("Device.OFF\n"));
            }
        }
        else if(strcmp((char *)&CmdWord_.c, "READ") == 0) // read voltage and current
        {
            if(on_off == 1)
            {
                /* print voltage to uart */
                adc_string(voltage,string);
                Print_Msg_P(PSTR("Voltage:"));
                uartTxString((unsigned char *)&string);
                Print_Msg_P(PSTR(" mV"));

                /* print current to uart */
                adc_string(current,string);
                Print_Msg_P(PSTR("\nCurrent:"));
                uartTxString((unsigned char *)&string);
                Print_Msg_P(PSTR(" mA\n"));
            }
            else
            {
                Print_Msg_P(PSTR("Device.OFF\n"));
            }
        }
        else
        {
            Print_Msg_P(PSTR("FALSE COMMAND\n"));
        }

        /* clear the rest of the command if not last*/
        if(CmdWord_.isLast != 1)
        {
            uartSkipTo('*');
        }
    }
    else
    {
        Print_Msg_P(PSTR("FALSE COMMAND\n"));
    }

}
```