Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka
Mikko Ruuskanen

Opinnäytetyö

Mikrokontrolleriohjattu virtapiirin katkontalaite

Työn valvoja yliopettaja Mauri Inha
Työn ohjaaja insinööri Jani Angervuo, KONE Oyj
Tampere 12/2010

Tampereen ammattikorkeakoulu
Tietotekniikka, sulautetut järjestelmät ja elektroniikka

Tekijä	Mikko Ruuskanen
Työn nimi	Mikrokontrolleriohjattu virtapiirin katkontalaite
Sivumäärä	27
Valmistumisaika	9.12.2010
Työn valvoja	Mauri Inha
Työn ohjaaja	Jani Angervuo, KONE Oyj

TIIVISTELMÄ

Tässä työssä suunniteltiin ja toteutettiin mikrokontrolleriohjattu virtapiirin katkontalaite KONE Oyj:n tutkimus- ja tuotekehitysosaston luotettavuuslaboratorion käyttöön. Aluksi määriteltiin ohjelmiston ja laitteiston vaatimukset. Lähtökohtana oli suunnitella helppokäyttöinen ja itsenäisesti toimiva testilaite, joka on helposti liikuteltavissa ja voidaan näin sijoittaa lähes mihin tahansa järjestelmään. Käyttäjän on myös voitava itse konfiguroida laite sarjaliikenneportin kautta.

Laite toteutettiin kahdessa vaiheessa. Ensimmäisessä vaiheessa suunniteltiin C-kielellä laitteen ohjelmisto. Apuna suunnittelutyössä käytettiin AVR STK500 -kehitysalustaa, AVR Studio 4 -suunnitteluympäristöä ja AVR-GCC-kääntäjää. Pääohjelma ja käyttöliittymä suunniteltiin alusta asti itse ohjelmoimalla. Apuna käytettiin valmiita toimintoja sisältäviä koodikirjastoja.

Toisessa vaiheessa suunniteltiin mikro-ohjaimen ympärille tarvittava elektroniikka. Ensimmäisessä versiossa katkos toteutettiin yksinkertaisella relekytkennällä. Releellä ei kuitenkaan päästä riittävän nopeisiin katkoksiin releen kytkentäviiveen takia. Lopullisessa versiossa katkos tehdään kytkinkäyttöön sopivalla IGBT-transistorilla.

Valmis laite mahdollistaa eripituisten katkosten tekemisen sähköjohtimissa. Näin saadaan testattua erilaisten laitteiden ja järjestelmien luotettavuus ja vikasietoisuus sähkökatkotilanteissa.

Avainsanat AVR, mikrokontrolleri, sähkökatko, testaus

TAMK University of Applied Sciences
Information Technology, Embedded Systems and Electronics

Writer	Mikko Ruuskanen
Thesis	Microcontroller Controlled Electronic Circuit Breaker
Pages	27
Graduation time	9.12.2010
Thesis Supervisor	Mauri Inha
Co-operating Company	Jani Angervuo, KONE Corporation

ABSTRACT

The purpose of this thesis work was to design and build microcontroller controlled electronic circuit breaker for use of KONE Corporation reliability laboratory. The system requirements were defined at first. The basic idea was to make user-friendly and self-operating test device that can be placed into almost any system. User must also be able to configure the device via serial port

The device was implemented in two phases. Software was developed first with C-programming language. Tools used in software development were AVR STK500 development board, AVR Studio 4 -design environment and AVR-GCC-compiler. Main program and user interface were designed for oneself. Basic operations containing libraries were used as help.

Hardware design was done in the second phase. First version was carried out with simple relay circuit. With relay it is not possible to make fast breaks though because of switching delay. In final version break is made with IGBT-transistor.

Complete device allows making of different length circuit breaks in electric wires. The redundancy and reliability of different systems can be tested with help of this circuit breaker device.

Keywords AVR, microcontroller, power cut, testing

Esipuhe

Tämä opinnäytetyö on tehty KONE Oyj:n luotettavuuslaboratorion tilauksesta. Työtä on tarkoitus käyttää lisäksi Tampereen ammattikorkeakoulun tietotekniikan koulutusohjelman opinnäytetyönä.

Haluan kiittää Tampereen ammattikorkeakoulun opettajia laadukkaasta opetuksesta, joka on antanut pohjatiedon tämän työn tekemiseen.

Lisäksi kiitokset Raimo Kaartiselle mielenkiintoisesta työn aiheesta, Jani Angervuolle asiantuntevasta ohjauksesta sekä luotettavuuslaboratorion henkilökunnalle teknisestä tuesta ja työrauhasta opinnäytetyön tekemisen aikana.

Tampereella joulukuussa 2010

Mikko Ruuskanen

Sisällysluettelo

1 Johdanto 7

2 Tutkimus ja tuotekehitys 8

2.1 Elektroniikan luotettavuus 8
2.2 Sähkön laatu 8
2.3 Hissikäyttö 9

3 Työn tavoitteet 10

3.1 Toiminnallinen määrittely 10
3.2 Laitteiston määrittely 10

4 Laitteen suunnittelu- ja kehitysympäristö 12

4.1 Kehitysalusta 12
4.2 Mikrokontrolleri 13
4.3 AVR Studio 4 -ohjelmiston kehitysympäristö 16
4.4 C-kieli, WinAVR ja AVR-GCC-kääntäjä 17
4.5 Konsoliohjelma uCon 18
4.6 Piirisuunnittelueditori OrCAD Capture CIS 18

5 Ohjelmistosuunnittelu 19

5.1 Alustustoimenpiteet 20
5.2 Ajastimet 20
5.3 Tiedonsiirto 21
5.4 Parametrit ja EEPROM 23
5.5 Pääohjelmasilmukka 24
5.6 Käyttöliittymä 26

6 Laitteistosuunnittelu 27

6.1 Mikrokontrolleri 28
6.2 Sarjaliikenne 28
6.3 Käyttöjännite 29
6.4 Led-merkkivalot 29
6.5 Relekytkentä 30
6.6 IGBT-puolijohdekytkentä 31

7 Testaus ja jatkokehitys 32

8 Yhteenveto 33

Lähteet 34

Liiteet 35

Liite 1: Ohjelmalistaus 35
Liite 2: Prosessorikortin piirikaavio 41
Liite 3: IGBT-ohjauksen periaatekuva 42

Lyhenteiden ja termien luettelo

ATmega	AVR-tuoteperheen tuotesarja
AVR	Atmelin valmistama 8-bittinen mikro-ohjainperhe
CPU	Central Processing Unit, suoritin
DIP	Dual In-line Package, elektronisen piirin kotelomalli
EEPROM	Electronically Erasable Programmable ROM, uudelleen ohjelmoitava ja datan säilyttävä muistityyppi
Flash	sähköisesti tyhjennettävä ja uudelleenohjelmoitava muistityyppi
GCC	GNU Compiler Collection, C-kielen käännösohjelmisto
IDE	Integrated Development Environment, integroitu kehitysympäristö
IGBT	Insulated Gate Bipolar Transistor, kytkinkäyttöön sopiva transistori
I/O	Input/Ouput, digitaalisissa piireissä käytetty nimi tulo- ja lähtöporteista
RAM	Random Access Memory, käyttömuisti
ROM	Read-Only Memory, pysyväismuisti
RISC	Reduced Instruction Set Computer, rajoitetun käskykannan filosofia
RS-232	Recommended Standard 232, standardi sarjaliikenneportti
SMD	Surface Mount Device, pintaliitostekniikalla juotettava komponentti
SO	Small-Outline Integrated Circuit (SOIC), elektronisen piirin kotelomalli
TQFP	Thin Quad Flat Pack, elektronisen piirin pintaliitoskotelomalli
USART	Universal Syncronous/Asyncronous Receiver/Transmitter sarjaliikenteen lähetys- ja vastaanottopiiri
UI	User Interface, käyttöliittymä

1 Johdanto

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa mikrokontrolleriohjattu virtapiirin katkontalaite KONE Oyj:n tuotekehityksen käyttöön. Työ tehtiin syksyn 2010 aikana KONE Oyj:n luotettavuuslaboratorion tiloissa.

Kaupallisilta markkinoilta löytyy ohjelmoitavia logiikoita, jotka voidaan ohjelmoida vastaavaan käyttöön. Ne eivät kuitenkaan ole välttämättä kovin helppokäyttöisiä, mikä on yksi työssä suunniteltavan laitteen tavoiteominaisuuksista. Logiikoilla ei myöskään päästä yhtä nopeisiin katkoksiin kuin työn yhtenä osana suunniteltavalla puolijohdekytkennällä.

Valmis laite tulee luotettavuuslaboratorion käyttöön ja mahdollistaa hissin eri osajärjestelmien vikasietoisuuden testaamisen sähkökatkotilanteissa jo tuotekehitysvaiheessa. Näin pystytään aikaisessa vaiheessa korjaamaan järjestelmissä olevat mahdolliset heikot kohdat, jolloin valmis tuote on markkinoille saapuessaan laadukas ja luotettava.

Työn pääpaino oli toimivan ohjelmiston ja käyttöliittymän suunnittelussa. Tässä työssä kuvataan suunnittelutyössä käytetyt työkalut, ohjelmiston rakenne ja laitteistosuunnittelu.

2 Tutkimus ja tuotekehitys

KONE Oyj on yksi maailman suurimmista hissi- ja liukuporrasvalmistajista. Yhtiö tarjoaa edistyksellisiä hissejä, liukuportaita ja automaattiovia sekä monipuolisia ratkaisuja niiden huoltoon ja modernisointiin. Tuotekehityksen osuus on tärkeä osa yhtiön liiketoimintaa.

2.1 Elektroniikan luotettavuus

Pysyäkseen alansa huipulla, on yrityksen pyrittävä jatkuvasti kehittämään nykyisiä ja tulevaisuuden tuotteita.

Elektroniikan ja ohjelmistojen lisääntyminen hissijärjestelmissä on tuonut lisää haasteita tuotekehitysvaiheen testaukseen ja toimintavarmuuden varmistamiseen. Kasvavat markkinat alueilla, joissa sähköverkon laatu ei ole paras mahdollinen, asettavat yhä enemmän vikasietoisuusvaatimuksia elektroniikalle.

Huolellinen testaus ja sitä kautta tuotteiden parantaminen tuotekehityksen aikana voi säästää paljon rahaa myöhemmässä vaiheessa, kun vikaantumismäärät ja huollot saadaan vähentymään.

2.2 Sähkön laatu

Sähkön laatu muodostuu kahdesta tekijästä; sähköä kuljettavan verkoston käyttövarmuudesta ja siirrettävän jännitteen laadusta. Sähkö laatu voidaan myös määritellä seuraavin perustein: sähkön jatkuva saatavuus, jännitetoleranssien ja taajuustoleranssien puitteissa pysyminen sekä puhtaan ja häiriöttömän siniaallon mukaisuus (Fortum 2010).

Sähkön laadun heikkenemisen syynä saattaa olla paikallinen laite tai jakeluverkko. Tyypillisiä vikojen aiheuttajia ovat kuorman äkillinen kytkeminen tai poistaminen, voimalaitoksen äkillinen pysähtyminen, ylikuormitettu linja, myrsky, lumikuorma, salamanisku ja generaattorin tai kuorman aiheuttamat harmoniset säröt (Sähköala 2010).

Kun jokin elektroninen laite kärsii jännitekuopista ja katkoista, seurauksena saattaa olla tuotannon menetyksiä, laitevaurioita tai tietojen katoamista. Vaikutukset ja niiden yhdistelmät voivat aiheuttaa asiakkaiden luottamuksen menetyksiä, jopa markkinaosuuden menetyksiä.

2.3 Hissikäyttö

Hissille aiheutuvat pitkät sähkökatkot johtuvat usein sähköverkon epävarmuudesta. Pidempi sähkökatko hissin ajon aikana merkitsee usein sen välitöntä pysähtymistä ja saattaa jättää matkustajat kerrosten välille. Loukkuun jäämisen välttämiseksi osa hisseistä on varustettu hätäakkukäytöllä, jolloin hissi ajaa lähimpään kerrokseen ja matkustajat pääsevät turvallisesti pois.

Myös hissijärjestelmien sisäisissä johdinrakenteissa saattaa esiintyä lyhyempiä katkoksia, joiden aiheuttajana voi olla esimerkiksi jokin ulkopuolinen tekijä. Tällaiset lyhyet sähkökatkot eivät saa lamaannuttaa koko järjestelmää.

Tässä työssä suunniteltava laite on tarkoitettu lyhyiden katkosten tekemiseen sekä järjestelmän sisäisissä johtimissa että ulkopuolisissa syöttöjännitteissä.

3 Työn tavoitteet

Laitteen kehitystyö aloitettiin määrittelemällä toiminnalliset ja laitteistokohtaiset vaatimukset.

3.1 Toiminnallinen määrittely

Valmiin laitteen pitää olla helppokäyttöinen ja laitteen parametrit täytyy pystyä konfiguroimaan RS-232 sarjaliikenneportin kautta. Asetetut parametrit säilyvät laitteen muistissa, vaikka virta on väliaikaisesti kytketty pois päältä. Laite alkaa suorittaa ohjelmaa annetuilla parametriarvoilla aina itsenäisesti kun kytketään sähköt päälle.

Sähkökatkon pituus on lyhyimmillään yhden millisekunnin mittainen ja suurin väliaika katkojen välillä on yksi vuorokausi. Eli yksi millisekunnin katko vuorokauden aikana. Laitteen parametreinä syötetään kanavan numero, katkon pituus, väliajan pituus ja katkojen lukumäärä. Katkoksia on myös pystyttävä tekemään manuaalisesti ohjauskomennolla sarjaliikenneportin kautta.

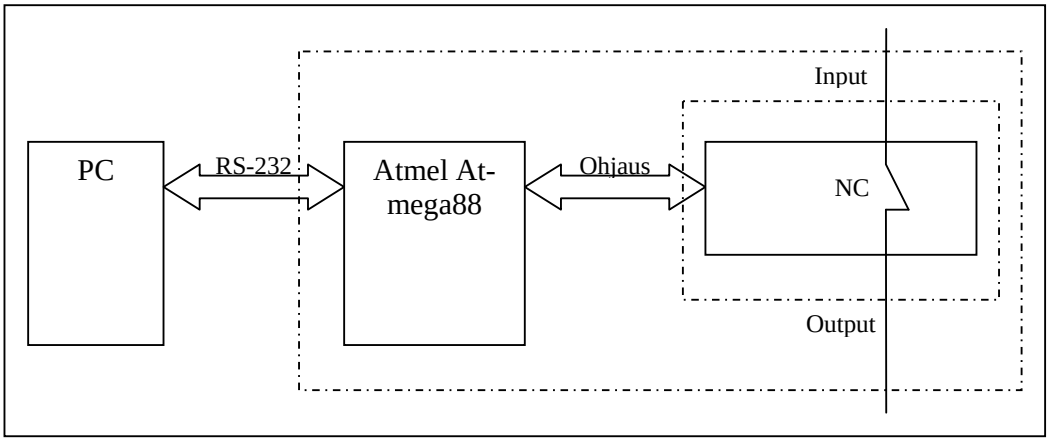
Laite laskee ohjelmallisesti kuinka monta katkoa on tehty ja antaa tuloksen pyydettäessä sarjaliikenneporttiin.

3.2 Laitteiston määrittely

Valmiin laitteen koko on suurimmillaan noin 20 x 20 x 10 senttimetriä. Se vaatii käyttöjännitteeksi +24 voltin tasajännitteen. Laitteen on pystyttävä katkomaan jännitteitä +3,3 voltin DC-jännitteestä 400 voltin AC-jännitteeseen asti, jolloin suurin katkaistava virta voi olla 2 ampeeria.

Käytetyt komponentit tulee olla helposti saatavissa ja yleisesti käytössä olevia. Laitteiston on oltava laajennettavissa jatkokehittelyä varten, kuten lisäkanavien määrän ja statistiikan lisääminen.

Mikrokontrollerilta ulos lähtevään ohjauksen jatkeeksi voidaan tarvittaessa kytkeä erilaisia kytkentäratkaisuja sen mukaan, minkälaisia katkoksia halutaan tehdä.



Kuvio 1: Testilaitteen lohkokaavio.

4 Laitteen suunnittelu- ja kehitysympäristö

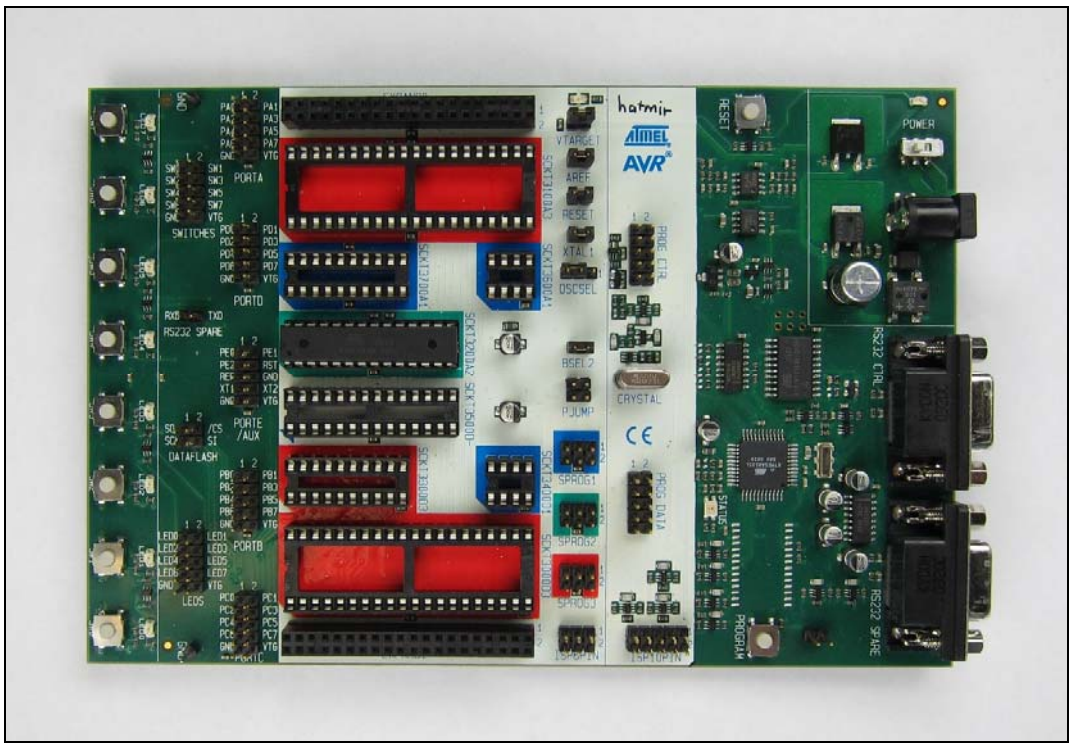
Laitteen suunnittelussa on pyritty hyödyntämään yleisesti käytössä olevia komponentteja ja mahdollisuuksien mukaan ilmaisia ohjelmistoja.

4.1 Kehitysalusta

Mikrokontrollerin kehitys- ja ohjelmointialustaksi valittiin helppokäyttöinen STK500-kehityspaketti, joka on tarkoitettu Atmelin piireille. Kortilla on mikropiirin kannat 8-, 20-, 28- ja 40-nastaisille mikro-ohjaimille. Laajennusliitäntään voi tarvittaessa liittää muita Atmelin STK-lisäosia.

Kortti tarvitsee käyttöjännitteeksi +12 V tasajännitteen, joka otettiin kehitysvaiheessa ohjelmoitavasta jännitelähteestä. Mikrokontrollerin ohjelmointi tapahtuu sarjaliikenneportin kautta. Toinen kehitysalustassa oleva sarjaliikenneportti on asetettu ohjelmoitavan kohdepiirin käyttöön.

Alustalta löytyy lisäksi kahdeksan kappaletta ledejä ja nappeja, jotka ovat hyödyllisiä testauksessa ja vianetsinnässä kehitysvaiheen aikana. Mikrokontrollerin ulostulot ja sisäänmenot ovat helposti kytkettävissä erillisten liitinrimojen kautta.



Kuvio 2: STK500-kehitysalusta, jossa DIP-koteloitu ATmega88 mikro-ohjain.

4.2 Mikrokontrolleri

Mikrokontrolleri, toiselta nimeltään mikro-ohjain, on työssä suunniteltavan laitteen keskeisin osa. Sen valintakriteerit eivät olleet kovin vaativat. Perustoiminnan vaatimuksena kontrollerissa täytyy vähimmillään olla yksi ulostulo, USART-piiri, ajastin sekä sisäistä EEPROM-muistia. Toisaalta jatkokehityksen kannalta varsinkin I/O-portteja täytyy olla enemmän.

Tästä syystä päädyttiin huomattavasti enemmän ominaisuuksia sisältävään Atmelin AVR-tuoteperheen 8-bittiseen ATmega88-mikrokontrolleriin. Yhtenä lisäsyynä hieman ylimitoitetun kontrollerin valintaan oli hyvä saatavuus. Kyseinen kontrolleri löytyi heti DIP-koteloisena materiaalivarastosta ja on KONEen materiaalikirjastossa myös pintaliitosversiona. Noin viiden euron hintakaan ei osoittautunut liian kalliiksi.

Atmel AVR ATmega88

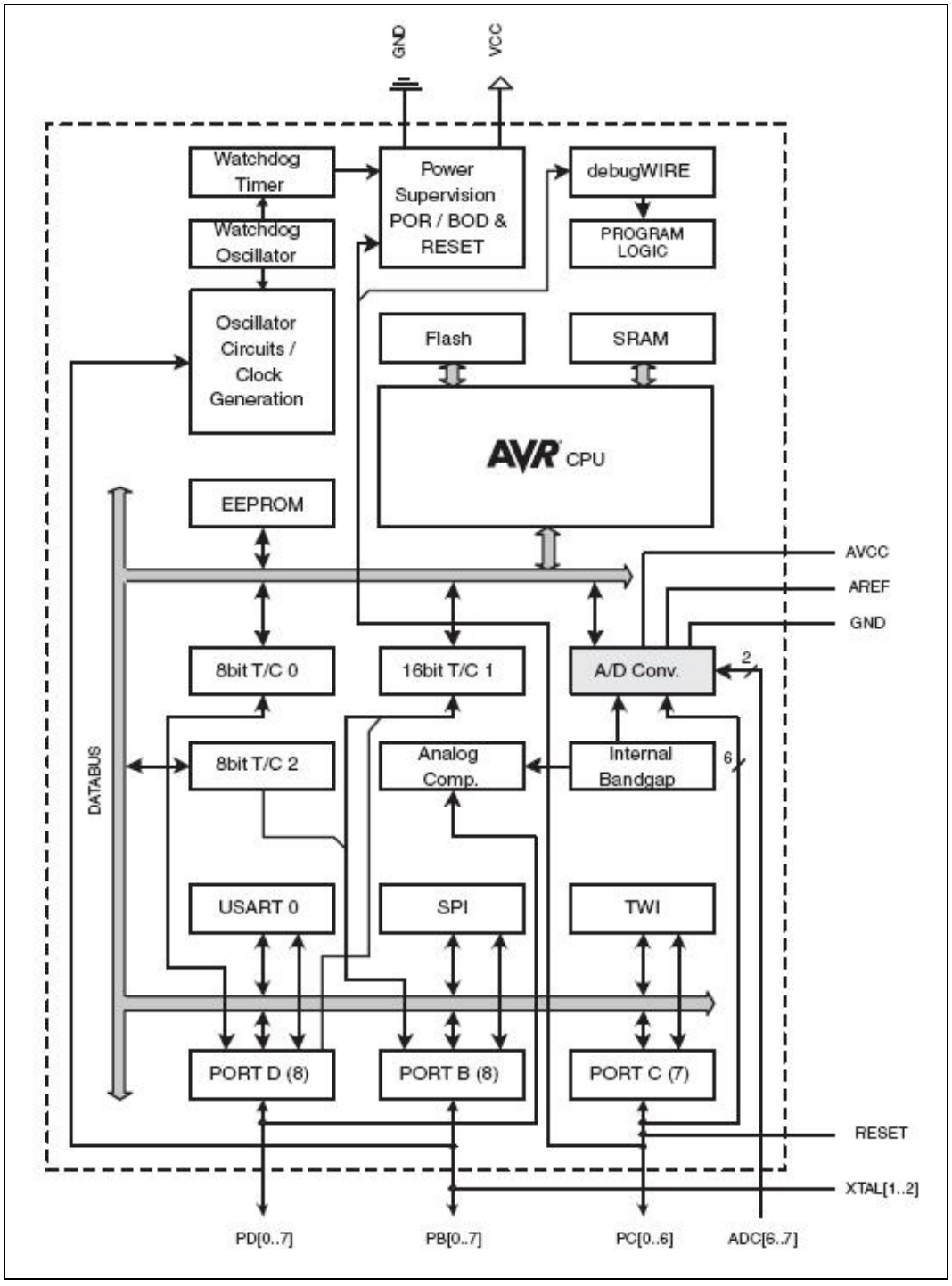
AVR on mukautetun Harvard arkkitehtuurin 8-bittinen RISC-tyyppinen yhden sirun mikrokontrolleri, jonka kehitti Atmel vuonna 1996. AVR-perheen piirejä on käytössä nykyään muun muassa erilaisissa autoteollisuuden sovelluksissa sekä turvallisuus-, voimansiirto- ja viihdejärjestelmissä (Atmel Corporation 2010).

AVR-tuoteperheessä on kolme 8-bittisten ohjainten pääryhmää; tinyAVR, megaAVR ja XMEGA. Eri versioita löytyy noin 190 kappaletta, joista noin 100 on megaAVR:iä Suurimmat erot eri mallien välillä ovat muistien kapasiteeteissa ja I/O-porttien määrissä (Atmel Parametric Table 2010).

AVR ATmega88 sisältää riittävästi ominaisuuksia ja antaa hyvät valmiudet jatkokehittelyä varten. Sen tärkeimpiin ominaisuuksiin kuuluu 23 kpl I/O-liitäntäntöjä, kolme ajastinta, joista kaksi 8-bittistä ja yksi 16-bittinen, A/D-muunnin, SPI ja USART. Tarkempi lohkokaavio on esitetty kuviossa 3.

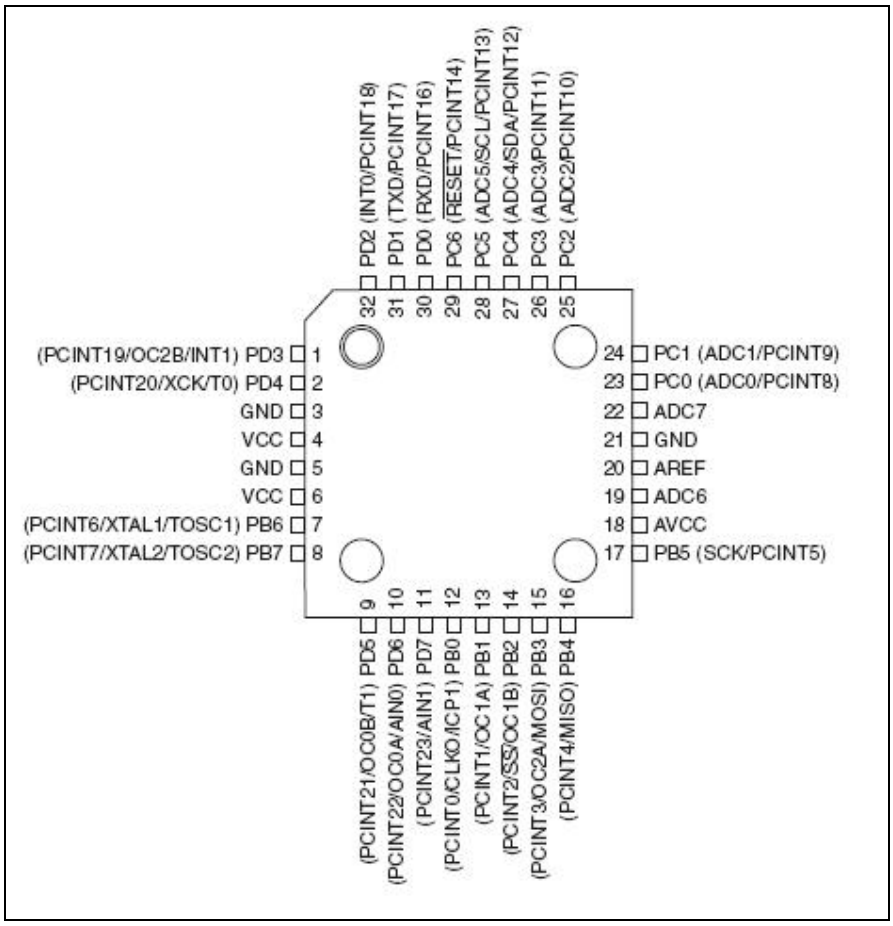
Piirin maksimikellotaajuus on 20 MHz. Työssä suunniteltavan laitteen kellotaajuudeksi valittiin 8 MHz, joka toteutettiin ulkoisella kiteellä. Piiri toimii näin riittävän nopeasti ja luotettavasti, kun valmistaja lupaa luotettavan toiminnan 2,7-5,5 voltin käyttöjännitealueella alle kymmenen megahertsin kellotaajuudella (Atmel Corporation ATmega88 datasheet 2010, 304).

ATmega88-kontrolleri sisältää kolmea erilaista muistityyppiä, 8 kt Flash-muistia, 512 kt EEPROM-muistia ja 1 kt SRAM-käyttömuistia. Ohjelma ladataan Flash-muistiin, joka kestää vähintään 10 000 kirjoitus/tyhjennys sykliä. SRAM-käyttömuisti sisältää rekisterit, I/O-rekisterit ja datamuistin. EEPROM-muistin käyttö on suhteellisen hidasta ja sitä käytetään lähinnä datan pitkäaikaiseen tallennukseen. Kuten Flash-muisti, EEPROM-muisti säilyttää sisältönsä kun sähköenergia poistetaan (Atmel Corporation ATmega88 datasheet, 16-20).



Kuvio 3: ATmega88-mikrokontrollerin lohkokaavio (Atmel Corporation Atmega88 datasheet 2010, 5)

Mikrokontrolleria on saatavissa neljällä erilaisella kotelomallilla. Kehitysvaiheessa oli käytettävissä DIP-mallinen kotelo, joka oli yhteensopiva STK500-kehitysalustan kanssa. Valmis laite toteutetaan TQFP-pintaliitosversiolla. Pintaliitosversion nastajärjestys on esitetty kuviossa 4.



Kuvio 4: Atmel ATmega88-mikrokontrollerin nastajärjestys TQFP-kotelolla (Atmel Corporation Atmega88 datasheet 2010, 2)

4.3 AVR Studio 4 -ohjelmiston kehitysympäristö

AVR Studio on Atmelin vapaasti jakama ohjelma, integroitu kehitysympäristö (IDE), joka on tarkoitettu 8-bittisten AVR-sovellusten kehitykseen Windows-ympäristöissä.

Kehitysympäristön ominaisuuksiin kuuluu muun muassa projektinhallintatyökalu, editori, simulaattori, tuki ulkoiselle C-kääntimelle, ohjelmointiohjelma ja tuki kaikille 8-bittisen AVR-arkkitehtuurin kehitystyökaluille kuten JTAGICE mk II, AVRISP ja STK500.

Projektinhallintatyökalun avulla käyttäjä pystyy hallitsemaan eri tiedostoihin sisältyviä koodeja. Ohjelmakoodi kirjoitetaan editorin avulla ja käännetään C-kielen tapauksessa kutsumalla ulkoista kääntäjää. Kun käännös onnistuu virheittä, voi käyttäjä simuloida ohjelmaa käsky kerrallaan simulaattorilla. Sen avulla voi seurata ohjelman etenemistä ja mikro-ohjaimen toimintaa, kuten I/O-rekistereitä, porttien tilaa, muistien käyttöä ja muuttujien arvoja.

4.4 C-kieli, WinAVR ja AVR-GCC-kääntäjä

Ohjelmointikieli on ohjelmien esittämiseen tarkoitettu, prosessorikohtaiseksi konekieleksi käännettävissä oleva kieli. C-ohjelmointikieli on 1970-luvun alussa kehitetty imperatiivinen ja sittemmin standardisoitu ohjelmointikieli. C-kielelle tyypillisiä ominaisuuksia ovat yksinkertaisuus, laitteistoläheisyys, proseduraalinen ohjelmointi, tehokkuus ja siirrettävyys alustalta toiselle.

Kun ohjelmoidaan C-kielellä, CPU:n sisäinen toiminta ja rekisterien bitit voidaan ainakin osittain unohtaa. Kattava pohjatietämys CPU:n sisäisestä toiminnasta on kuitenkin eduksi.

Kun AVR-Studiolla kirjoitettu C-kielinen ohjelma on saatu valmiiksi, tarvitaan seuraavaksi kääntäjää. Kääntäjä on ohjelma, jonka avulla lähdekoodi käännetään ennen ohjelman ajoa toimivaksi ja ajokelpoiseksi konekieliseksi koodiksi

Tässä työssä käytetty WinAVR-ohjelmisto sisältää useita avoimen lähdekoodin ohjelmistojen kehitystyökaluja, jotka on tarkoitettu Atmelin AVR-sarjan RISC-mikrokontrollereille. Se sisältää myös GCC-kääntäjän. Nappia painamalla AVR Studio kutsuu AVR-GCC kääntäjää, joka kääntää editorissa avoinna olevan projektin C-kielisen lähdekoodin konekoodiksi.

Tämän jälkeen käännöksen tuloksena syntynyt *.hex-tiedosto siirretään PC:n muistista mikro-ohjaimen ohjelmamuistiin erityisen latausohjelman avulla. Latausohjelma siirtää koodin sarjaliikenne- tai USB-portin kautta ohjelmointikaapelia pitkin mikro-ohjaimen ohjelmointiliitäntään.

4.5 Konsoliohjelma uCon

Käyttöliittymäksi valittiin konsoliohjelma uCon. Se on ominaisuuksiltaan kattavampi kuin Windows-ympäristön tarjoama Hyper Terminal ja suunniteltu ensisijaisesti sulautettujen järjestelmien kehittäjille. Ohjelmassa on kahdeksan ohjelmoitavaa toimintonäppäintä ja 16 ohjelmoitavaa painiketta. Erona vain se, että toimintonäppäimet ovat käytettävissä myös näppäimistöltä.

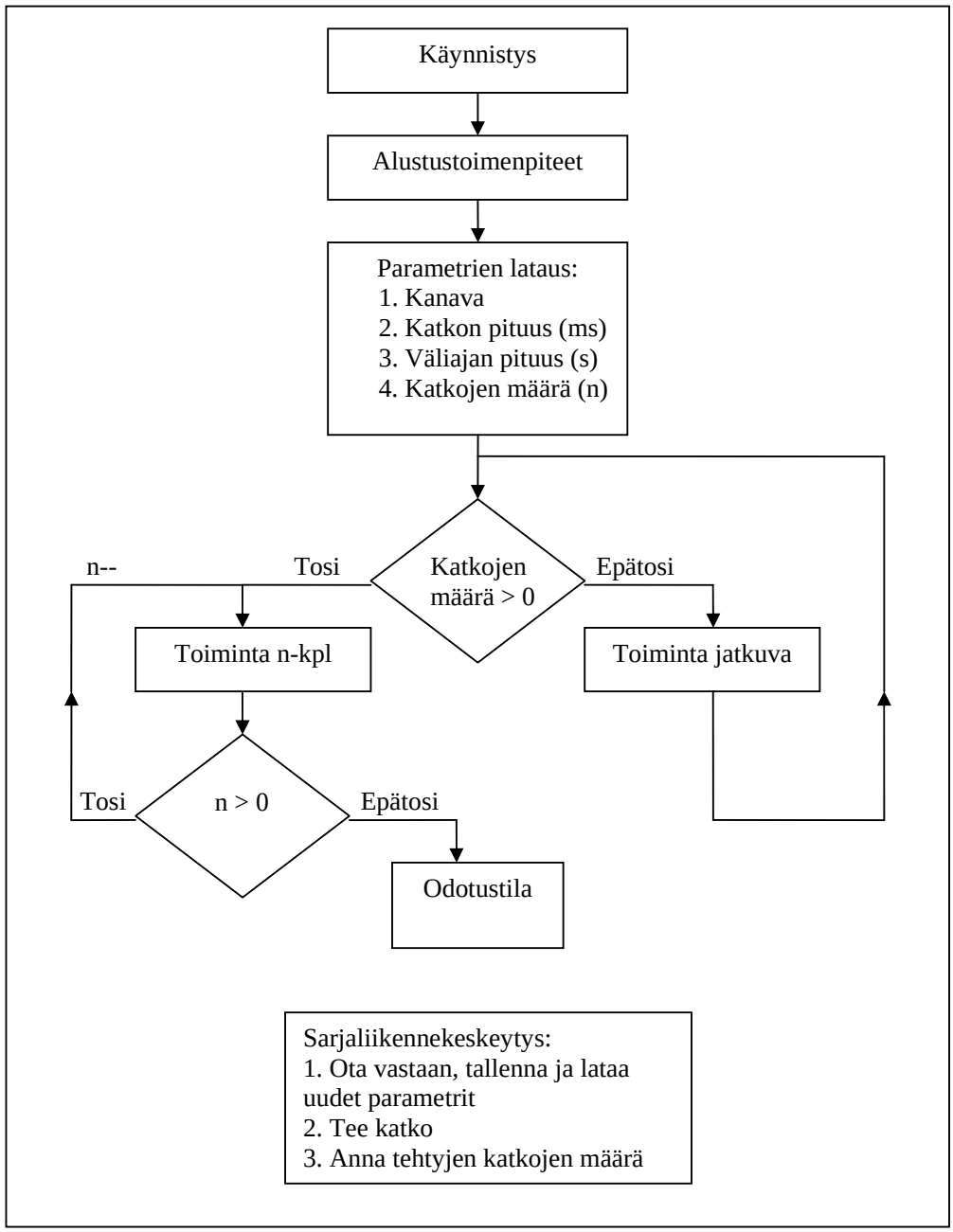
Ohjelma kommunikoi laitteen kanssa sarjaliikenneportin kautta. Itse tehdyt komentosarjat voidaan ohjelmoida käynnistymään nappia painamalla. Tämä helpottaa usein tarvittavien komentojen syöttämistä. Valmiin laitteen käyttöliittymä toteutettiin uCon-konsoliohjelmistolla.

4.6 Piirisuunnitteluedi tori OrCAD Capture CIS

Piirisuunnittelu tehtiin Cadence OrCAD Capture CIS -ohjelmistolla. Se on maailman käytetyin piirikaavioeditori, joka soveltuu hyvin yksittäisen käyttäjän perustarpeisiin kuin myös vaativaan hierarkiseen piirisuunnitteluun (Cadence 2010).

5 Ohjelmistosuunnittelu

Määrittelyjen jälkeen laitteen varsinainen suunnittelutyö aloitettiin ohjelmiston suunnittelulla. Siinä lähdettiin liikkeelle laatimalla yksinkertainen vuokaavio tavoitteen olevasta toiminnasta. Kaavio on esitetty kuviossa 5.



Kuvio: 5 Ohjelman rakenne.

Kytkettäessä virta mikrokontrolleri suorittaa ensin määritetyt alustustoimenpiteet. Sen jälkeen ladataan EEPROM-muistista parametreinä kanavan numero, katkon pituus, väliajan pituus ja katkojen lukumäärä.

Silloin kun katkojen lukumäärä on nolla, katkoksia tehdään parametrien mukaisella aikavälillä loputtomasti. Jos katkojen lukumäärä on annettu, tehdään katkoja annettu määrä, jonka jälkeen siirrytään odotustilaan.

Sarjaliikennekeskeytyksellä käyttäjä voi antaa syötteenä uudet parametrit, jolloin ohjelmasilmukka alkaa alusta uusilla arvoilla. Käyttäjä voi myös pyytää tehtyjen katkoksien lukumäärän tai tehdä välittömän katkon tietyllä komennolla.

Valmiin ohjelman C-kielinen listaus on esitetty liitteessä 1.

5.1 Alustustoimenpiteet

Alustustoimenpiteinä alustetaan ensin käytössä olevat I/O-linjat. Sen jälkeen alustetaan USART0-sarjaliikenneväylä ja parametritietojen vastaanottamiseen käytettävä sarjaliikennepuskuri sekä 8-bittinen Timer0-ajastin. Lopuksi alustetaan pääohjelman muuttujat ja käytettävät aputaulukot, jonka jälkeen sallitaan globaalit keskeytykset ja pääohjelman suoritus voi alkaa.

5.2 Ajastimet

ATmega88 sisältää yhteensä kaksi 8-bittistä ajastinta ja yhden 16-bittisen ajastimen. Tässä sovelluksessa käytetään 8-bittistä Timer0:aa, joka alustetaan toimimaan Compare Match eli CTC-moodissa. Koska lyhyin katkoksen pituus ohjelmassa on yksi millisekunti, asetetaan ajastimen aikaväliksi myös yksi millisekunti.

CTC-moodissa laskuri nollataan aina kun laskurin arvo rekisterissä TCNT0 vastaa asetettua vertailuarvoa OCR0A rekisterissä. Nollaus kääntää lippubitin TIFR0-lippurekisterissä, jota tarkkailemalla saadaan tietoon haluttu yhden millisekunnin aikaväli.

Kahdeksan megahertsin kellotaajuudella käskyjakson pituus on 125 ns. Timer0 on 8-bittinen eli sen arvoalue on 0-255, jolloin yhden millisekunnin keskeytysvälin saamiseksi on järkevää käyttää esijakajaa. Kun esijakajan arvoksi asetetaan 64, ajastimen arvo kasvaa 8 mikrosekunnin välein. Kun jaetaan yksi millisekunti kahdeksalla mikrosekunnilla ja vähennetään siitä yksi, saadaan tavoitearvoksi 124. Alustustoimenpiteet on esitetty alla.

```
TCCR0A |= (1 << WGM01);                 // ajastin 0 CTC-tilaan
TCCR0B |= ((1 << CS10) | (1 << CS11));  // ajastimen alustus esijakajalla 64
OCR0A   = 124;                          // aseta CTC vertailuarvo
```

Ajastinta käytetään ohjelmassa hyödyksi aikavälin ja katkon pituuden laskennassa.

5.3 Tiedonsiirto

Tiedonsiirto PC:n ja piirikortin välillä on toteutettu sarjaliikenneväylän kautta. Mikrokontrolleriin on integroitu USART-sarjaliikennepiiri, joka muuttaa rinnakkaismuotoista dataa sarjamuotoiseen ja päinvastoin. USART ottaa vastaan tavuja ja lähettää ne yksittäisinä bitteinä sarjamuodossa eteenpäin.

Sarjaliikenneväylä alustetaan käyttämään haluttua bittinopeutta ja databittien määrää. Sarjaliikenneväylällä käytettävä nopeus riippuu ensisijaisesti prosessorin kellotaajuudesta. Ohjelmassa käytetään tiedonsiirtoon 9600 bittinopeutta, 8 databitin ja yhden stopbitin tiedonsiirtokehystä, jolloin datalehti antaa virheprosentiksi riittävän alhaisen 0,2 % (Atmel Corporation Atmega88 datasheet 2010, 196).

Sarjaliikennepiirin alustustoimenpiteissä ohjausrekisteri USCR0A rekisteri nollataan ja UCSR0B rekisterin arvoksi asetetaan 0x98, jolla asetetaan keskeytys-, lähetys- ja vastaanottotoiminnot päälle

Alustuksen yhteydessä mikro-ohjaimen nopeusrekisteriin UBRR0 kirjoitetaan arvoksi 0x33, joka on laskettu kaavan 1 mukaisesti. Atmel antaa arvot myös valmiiksi eri nopeuksille ja kellotaajuuksille. Arvon voi tarkistaa datalehden taulukosta (Atmel Corporation Atmega88 datasheet 2010, 196).

$$BAUD = \frac{f}{16(UBRR + 1)} \quad (1)$$

missä: $BAUD$ = sarjaliikennenopeus
f = prosessorin kellotaajuus
$UBRR$ = nopeusrekisterin arvo

Sarjaliikennepiirin alustusfunktio on esitetty alla.

```
void USART_Init(unsigned int ubrr)      // USART alustus, parametrinä laskettu
                                        // tiedonsiirtonopeus 51,0833 eli hexalukuna ilman
                                        // desimaaleja 0x33
{
        UCSR0A=0x00;                    // tyhjentää ohjaus- ja tilarekisterin A
        UCSR0B = 0x98;                  // keskeytys, lähetin ja vastaanotin päälle
        UCSR0C = 0x06;                  // formaatti 8 databittiä, 1 stop bitti
        UBRR0H = (ubrr>>8);             // baudinopeus high-osa
        UBRR0L = ubrr;                  // baudinopeus low-osa
}
```

Sarjaliikenneväylän kautta käyttäjä voi ohjelmoida haluamansa parametrit laitteeseen ja pyytää statistiikkatietoja. Sarjaliikenneportti on varsinkin kannettavissa tietokoneissa nykyään harvinainen, mutta kaupallisilta markkinoilta on saatavissa USB-väylään liitettäviä sarjaliikenneadaptereita. PC:n ja testilaitteen välille tarvitaan lisäksi standardin mukainen ristiinkytketty RS-232-kaapeli.

Sarjaliikennepuskuri

Kun vastaanotetaan parametritietoja sarjaliikenneväylästä, tarvitaan avuksi puskuri vastaanotettujen numeroarvojen siirtämiseksi muuttujiin. Puskurin alustusfunktiolle annetaan parametreinä nimi, puskurin osoite SRAM-muistissa ja puskurin koko.

```
bufferInit(&uartRxBuffer,               // sarjaliikennepuskurin alustusfunktion kutsu
 (u08*) UART_RX_BUFFER_ADDR,
UART_RX_BUFFER_SIZE);
```

Sarjaliikennekeskeytyksen tapahtuessa ohjelma vastaanottaa keskeytysvektorissa numeroita puskuriin niin kauan, kunnes käyttäjä antaa syötteenä ’X’-merkin. Tämän jälkeen puskurissa olevat arvot siirretään taulukkoon ja puskuri tyhjennetään. Puskurin tyhjennys C-koodina on esitetty alla.

```
for(int i=0; i<17; i++)                                  // puskurin tyhjennyssilmukka
{
   table[i] = bufferGetFromFront(&uartRxBuffer);         // haetaan merkki puskurista taulukkoon
   if(table[i]=='0' && table[i-1]==' ')                  // syötteen tarkistus
   {
      table[i]=' ';
   }
}
uartRxBuffer.datalength = 0;                             // puskurin alustus
```

5.4 Parametrit ja EEPROM

Vaadittujen määritysten mukaisten katkosten aikaansaamiseksi ohjelman parametrien minimi- ja maksimiarvot tulee olla seuraavat:

1. kanava	1 - 3
2. katkon kesto	1 ms - 1min
3. väliaika	1 s - 24 h
4. katkojen lukumäärä	1 - 999, tai 0 = päättymättömästi

Nämä parametrit käyttäjä antaa syötteenä muodossa ’1 22222 33333 444’. Ensimmäisenä kanavien määrä, toisena katkon pituus millisekunteina, kolmantena väliajan pituus sekunteina ja viimeisenä katkojen lukumäärä. Näin saadaan vaaditut arvot arvo-alueen sisään:

1. kanava	1 - 9
2. katkon kesto	1 - 99999 ms = 1 ms - 1 min 39 s 999 ms
3. väliaika	1 - 99999 s = 1 s - 27 h 46 min 39 s
4. katkojen lukumäärä	1 - 999 tai päättymättömästi

Kanavan ja katkojen lukumäärän tietotyyppinä käytetään etumerkitöntä 16-bittistä unsigned int -tyyppiä, jonka lukualue on 0-65535. Katkon ja väliajan keston maksimiarvot eivät mahdu kyseiselle lukualueelle, joten niissä käytetään pidempää etumerkitöntä 32-bittistä unsigned long int -tyyppiä.

Sarjaliikennepuskurista aputaulukkoon siirretyt numeroalkiot muutetaan edellä mainituiksi neljäksi eri muuttujaksi. Muuttujien arvot tallennetaan tämän jälkeen EEPROM-muistiin. EEPROM-muisti on pysyväismuistia, jossa parametrit säilyvät sähkökatkon aikana. Tallennus- ja latausfunktiot on suojattu keskeytyskieltoalueella, jolloin ulkoiset keskeytykset eivät ole sallittuja. Näin pyritään välttämään datan korruptoitumista.

Kytkettäessä virta päälle, ohjelman viimeisimmät parametritiedot ladataan EEPROM-muistista muuttujiin ja ohjelman suoritus alkaa viimeisimmillä arvoilla.

Tallennus- ja latausfunktiot saavat parametreinään datan osoitteen SRAM-muistissa, kohdeosoitteen EEPROM-muistissa ja datan pituuden tavuina.

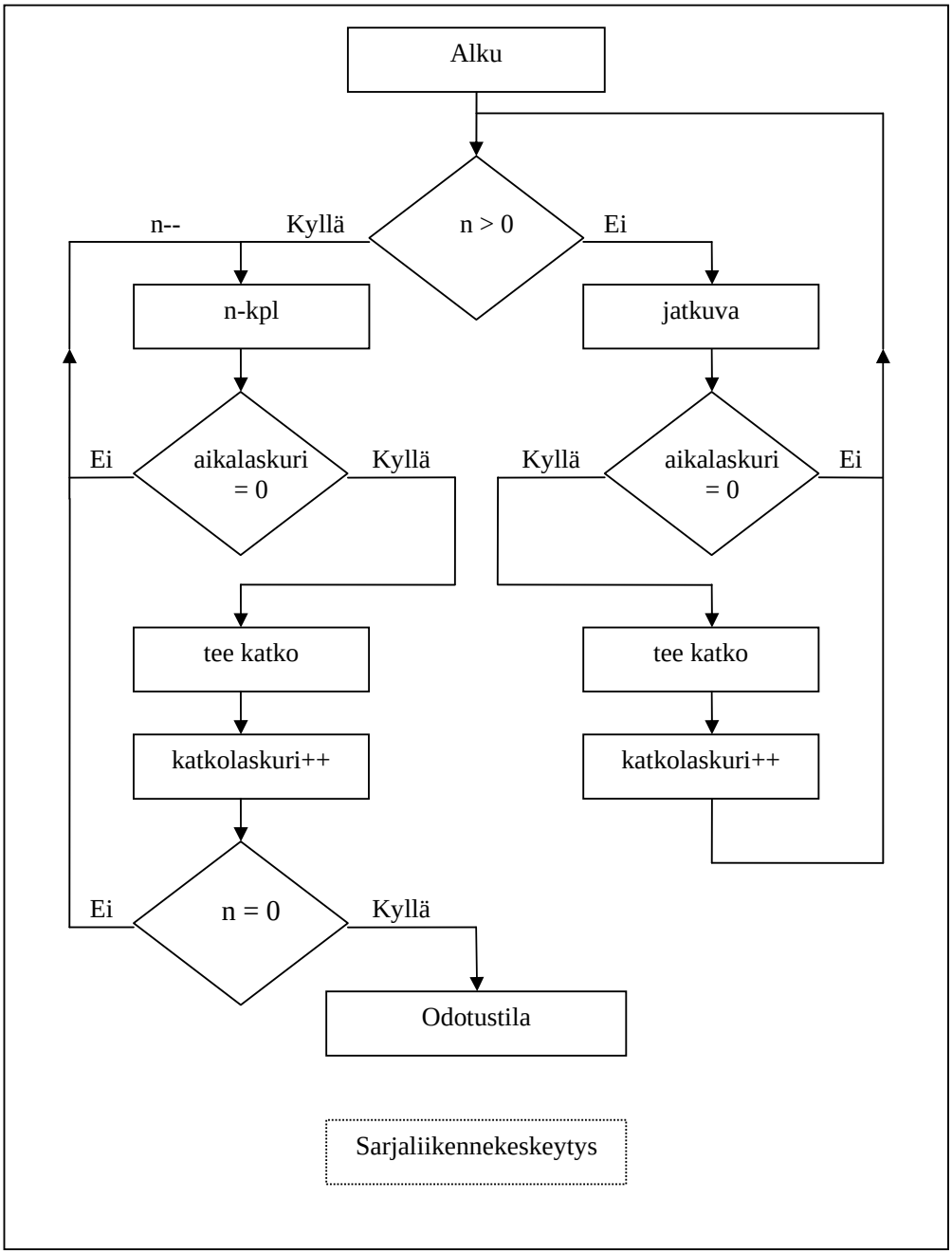
```
// parametrien tallennus
paramStore((void*) &disable, (void*)2,4);                 // katkon kesto EEPROMmiin
paramStore((void*) &enable, (void*)6,4);                  // väliajan kesto EEPROMmiin
paramStore((void*) &numberOfBreaks, (void*)10,2);         // katkojen lukumäärä EEPROMmiin
paramStore((void*) &output, (void*)12,2);                 // kanava EEPROMmiin


// parametrien lataus
paramLoad((void*) &disable, (void*)2,4);                  // EEPROM lataus
paramLoad((void*) &enable, (void*)6,4);                   // EEPROM lataus
paramLoad((void*) &numberOfBreaks, (void*)10,2);          // EEPROM lataus
paramLoad((void*) &output, (void*)12,2);                  // EEPROM lataus
```

5.5 Pääohjelmasilmukka

Pääohjelmasilmukan suoritus alkaa välittömästi parametrien latauksen jälkeen. Ohjelmassa on kaksi toimintatilaa, jatkuva ja n-kpl. Jatkuvassa tilassa ohjelman suorittaa silmukkaa loputtomasti. Jälkimmäisessä tilassa ohjelmasilmukkaa suoritetaan niin monta kierrosta kuin käyttäjä on antanut parametrinä katkojen määräksi. Tämän jälkeen siirrytään odotustilaan.

Jokaisen katkon jälkeen lisätään katkolaskurin arvoa yhdellä ja n-tilassa vähennetään myös n:än arvoa. Katkon aikana on voimassa keskeytyskielto, jolloin ei voida antaa uusia parametrejä eikä kysyä katkojen lukumäärää.

Muuten ohjelmaa suoritetaan niin kauan, kunnes tulee sarjaliikennekeskeytys uusien parametrien syöttämiseksi tai katkoja on tehty parametrin mukainen määrä. Kuvio 6 kuvaa pääohjelman toimintaa.

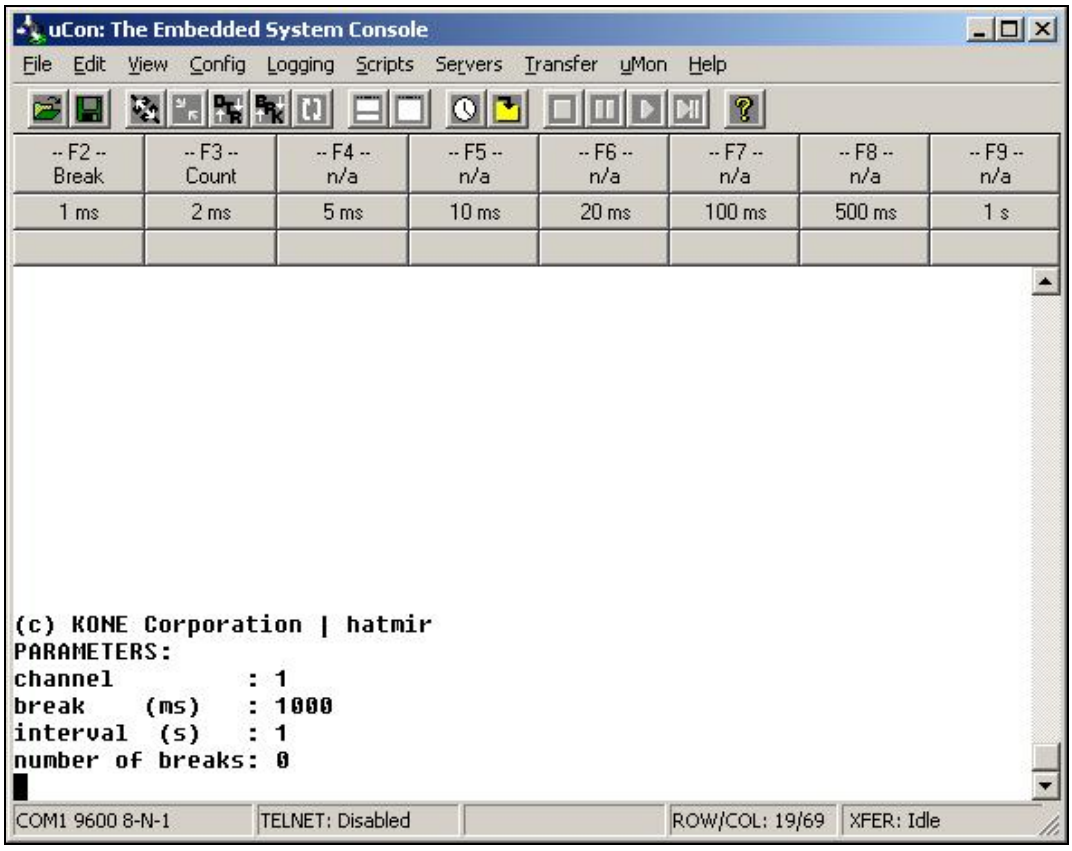


Kuvio 6: Pääohjelman toiminta kaaviona.

5.6 Käyttöliittymä

Piirikortilla ei ole ollenkaan ohjaustoimintoihin käytettäviä painokytkimiä eikä näyttöä. Kortilta löytyy vain reset-kytkin ja kuusi lediä indikoimaan prosessorin tilaa. Kaikki ohjauskomennot suoritetaan konsoliohjelman kautta. Käyttäjä voi antaa parametritiedot syötteenä tai ohjelmoida valmiin komentosarjan painikkeeksi.

Kuviossa 7 on esitetty uCon-konsoliohjelman käyttöliittymä toimintanappeineen. Funktionäppäimiin on ohjelmoitu kaksi komentoa, ensimmäinen katkon tekemiseksi ja toinen tehtyjen katkojen lukumäärän näyttämiseksi. Ensimmäiseen painikeriviin on ohjelmoitu eripituisia katkoksia sekunnin aikavälein. Lopulliseen ohjelmaan valmiit komentosarjat ohjelmoidaan loppukäyttäjien toiveiden ja saadun palautteen mukaisiksi.

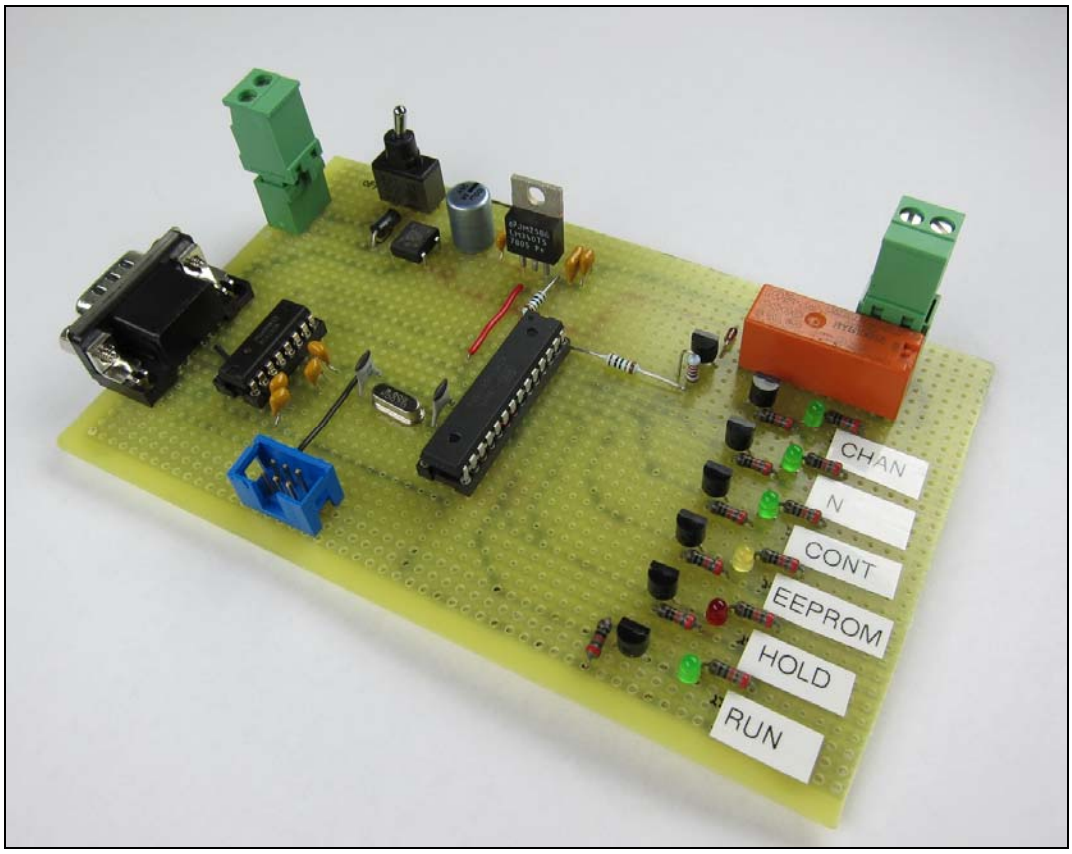


Kuvio 7: uCon terminaaliohjelma.

6 Laitteistosuunnittelu

Työn viimeisenä osana suunniteltiin piirikortti. Tässä kappaleessa on esitetty piirisuunnittelun ydinkohdat. Suunnittelu tehtiin OrCAD Capture CIS -ohjelmistolla. Ensimmäiseen prototyyppiin lisättiin vastaavat kytkennät, jotka löytyvät käytössä olevalta STK-500-kehityskortilta. Lopulliseen versioon kytkentöjä muutettiin paremmin käyttötarkoitukseen sopivaksi.

Ensimmäiseen laitteeseen kytkennät tehtiin läpiladottavilla komponenteilla, jolloin sen muokkaaminen on tarvittaessa helpompaa. Lopullinen versio toteutettiin pääasiassa pintaliitoskomponenteilla, jolloin piirikortin fyysinen koko saatiin pienemmäksi



Kuvio 8: Ensimmäinen prototyyppi johon on jätetty tilaa mahdollisille lisäkomponenteille.

Protokortin yksi ulostulokanava toimii releellä, jolloin releen kytkentäviive rajoittaa lyhimmän katkon pituudeksi noin 20 ms. Lopullisessa laitteessa ulostulokanava toimii sekä releellä että vaihtoehtoisesti IGBT-transistorilla, jonka avulla päästään nopeampiin kytkentätaajuuksiin. Korttiin lisättiin vielä optiot kanavien lisäämiseksi jatkokehittelyä varten. Kokonainen prosessorikortin kytkentäkaavio on esitetty liitteessä 2 ja pääkohdat IGBT-kytkennästä liitteessä 3.

6.1 Mikrokontrolleri

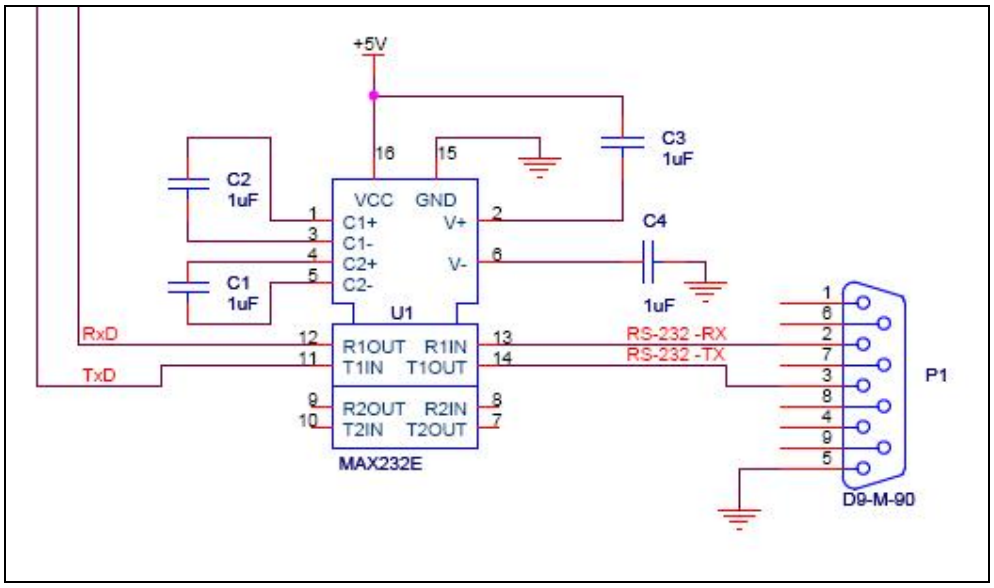
Kehitysvaiheen aikana käytettiin DIP-koteloitua ATmega88-mikrokontrolleria, jolle tehtiin oma protokortti. Valmiiseen korttiin kalustetaan vähemmän tilaa vievä TQFP-pintaliitoskotelo. Mikrokontrolleri vaatii toimiakseen viiden voltin käyttöjännitteen ja ulkoisen kiteen. Kide vaatii lisäksi kaksi kondensaattoria tasavirtojen katkaisemiseksi (ATmega88 datasheet 2010, 30).

RxD- ja TxD -signaalit kytkettiin MAX232-sarjaliikennemuuntimelle. Ohjaukseen ja mikrokontrollerin tilan indikointiin käytettävät I/O:t kytkettiin ledeille. Ylimääräiset mikrokontrollerin jalat kytkettiin datalehden ohjeiden mukaisesti tai maapotentiaaliin (Atmega88 datasheet 2010, 3-4).

Kortin ohjelmointia varten löytyy oma 6-pinninen ohjelmointiliitäntä. Ohjelmointiin voi käyttää joko STK-500-ohjelmointialustaa tai AVRISP-ohjelmointilaitetta. Molemmilla laitteilla voidaan ohjelmoida ATmega88:in Flash- ja EEPROM-muisteja sekä pystytään muokkaamaan sulakebittejä.

6.2 Sarjaliikenne

Parametritiedot välitetään laitteelle sarjaliikenneväylän kautta kuten myös statistiikkatiedot laitteesta takaisin PC:lle. Laitteistoon liitytään terminaaliohjelmalla käyttäen standardin mukaista ristiinkytkettyä RS-232-kaapelia. RS-232-väylä on toteutettu valmiilla MAX232-muunninpiirillä ja tämän vaatimilla oheiskomponenteilla (Texas Instruments 2004, 7). Sarjaliikenneliitynnän toteutus on esitetty kuviossa 9.



Kuvio 9: Sarjaliikenneliitynnän toteutus.

6.3 Käyttöjännite

Ensimmäisessä protokortissa viiden voltin käyttöjännitteen tekemiseen käytetään tasasuuntaussiltaa ja lineaarista LM7805-jänniteregulaattoria, jolla jännite reguloidaan +12 V tasajännitteestä. Käyttöjännitekytkentä on sama kuin STK-500-kehitysalustassa.

Lopullisessa laitteessa jännitteen tekemiseksi käytetään hakkurikytkentää, jolla ulkopuolisesta +24 VDC:n jännitteestä tehdään +5 VDC:n käyttöjännite. Monessa testilaitteen kohdejärjestelmässä löytyy valmiina +24 VDC:n jännite, josta voidaan ottaa jännite myös testilaitteeseen.

Käytettävä LM2675 Simple Switcher on DC/DC- hakkuripiiri yhden ampeerin virrankestolla. Piiri on koteloitu 8-nastaiseen SO-koteloon ja se vaatii toimiakseen ulkoisina komponentteina nollahaaran diodin, energiaa varastoivan kuristimen ja kondensaattoreita (National Semiconductor 2005, 1).

6.4 Led-merkkivalot

Prosessorikortilla on yhteensä yhdeksän Led-merkkivaloa, jotka osoittavat käyttäjälle kortin toimintatilaa. Ensimmäinen ledi näyttää onko kortin viiden voltin käyttöjännite kunnossa. Seuraavat viisi lediä kertovat missä vaiheessa ohjelman suoritus on menossa.

Viimeisenä olevat kanava-ledit aktivoituvat kun katko on päällä. Kanavat 2 ja 3 ovat kalustettu lopulliselle kortille valmiiksi jatkokehitystä varten. Ledejä D2-D9 ohjataan suoraan mikrokontrollerin I/O-porteista.

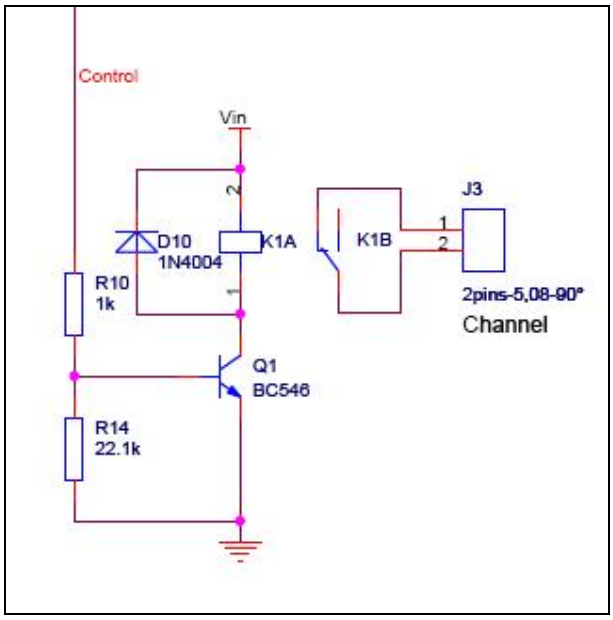
Talukko 1: Led-indikaattorit

Nimi	**Osanumero**	**Väri**	**Toiminta kun led aktiivisena**
+ 5 V OK	D1	Vihreä	+ 5 voltin jännite OK
Run	D2	Vihreä	Toimintatila
Hold	D3	Punainen	Odotustila
EEPROM	D4	Keltainen	EEPROM lataus / tallennus
Continous	D5	Vihreä	Toimintamoodi jatkuva
N	D6	Vihreä	Toimintamoodi n-kpl
Channel 1	D7	Punainen	Kanava 1
(Channel 2)	(D8)	(Punainen)	(Kanava 2)
(Channel 3)	(D9)	(Punainen)	(Kanava 3)

6.5 Relekytkentä

Relettä ohjataan mikrokontrollerin I/O-lähdöistä. Kun npn-transistorin kannalle tulee prosessorikortilta jännite, alkaa transistori johtaa, jolloin kollektorin ja Vin-jännitteen väliin kytketty kela saa täyden jännitteen.

Kun kannalle ei tuoda jännitettä niin ei kelankaan kautta kulje virtaa. Kelan rinnalla on suojadiodi, joka estää kelan virtaa katkaistaessa syntyvää negatiivista jännitepiikkiä hajottamasta ohjaintransistoria.



Kuvio 10: Releen ohjauskytkentä

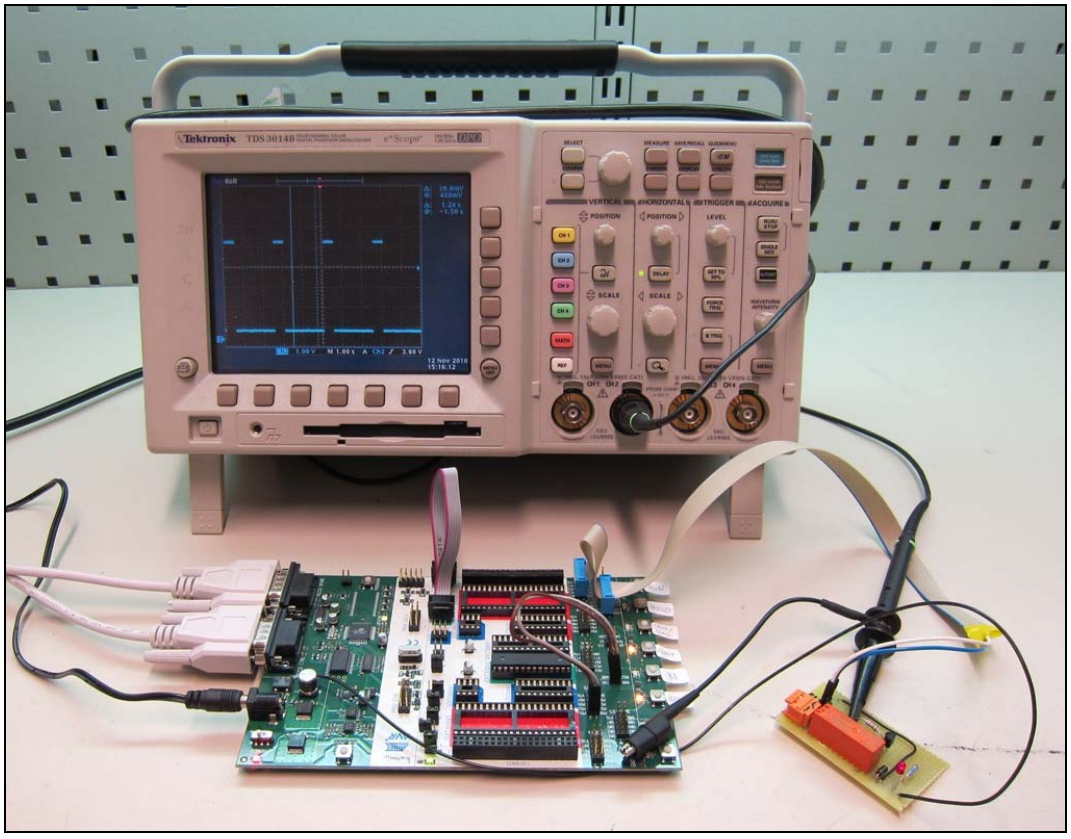
6.6 IGBT-puolijohdekytkentä

IGBT:n ohjaukseen käytettiin KONEen valmista kytkentää, joka on esitetty tässä työssä vain periaatetasolla liitteessä 3. IGBT ei sovellu suoraan AC-jännitteen ohjaukseen. Se voi johtaa virtaa vain yhteen suuntaan.

Käyttämällä sarjakytkentää saadaan katkottua myös AC-jännitettä. Toinen puolijakso kulkee IGBT:n ja toinen diodin kautta ja päinvastoin. Molempia ohjataan samalla ohjaussignaalilla (International Rectifier n.d.).

7 Testaus ja jatkokehitys

Testilaitteen ominaisuudet osoittautuivat pääosin alkuvaatimusten mukaisiksi ja testilaite käyttäytyi toivotulla tavalla. IGBT-puolijohdekatkojan osalta laite vaatii vielä jatkosuunnittelua ja -testausta.



Kuvio 11: Laitteen testausta kehityksen alkuvaiheessa.

Testauksessa on pyritty keskittymään luotettavuuteen, koska kyseessä on testilaite. Laitteiston tulee näin ollen pystyä toistamaan luotettavasti samaa toimintaa silloin, kun sen parametrointi pysyy vakiona.

Ohjelman osalta laitteen toimivuutta on pyritty testaamaan mahdollisimman kattavalla tavalla. Testauksen aikana laitteen toiminnassa ei ole ilmennyt virheitä.

8 Yhteenveto

Tässä opinnäytetyössä tehdyn projektin tavoitteet täyttyivät ja työn tuloksena syntyi kompakti ja helppokäyttöinen katkontalaite.

Työn vaativin ja eniten aikaa vievä osuus oli mikrokontrollerin sovelluksen ohjelmointi. Koska tietämys mikrokontrollereiden ohjelmoinnista oli perustasolla, vaadittiin alkuun pääsemiseksi jonkin verran lisäselvitystyötä. Lopputuloksena saadun valmiin ohjelman rakenne on melko yksinkertainen eikä ohjelman teon aikana ollut suurempia ongelmia.

Työn toisessa osassa fyysisen laitteen suunnittelu ja rakentaminen tehtiin pääosin yhdistelemällä valmiita ja hyväksi havaittuja kytkentöjä. Suunnittelussa pärjäsi hyvin perustason elektroniikkasuunnittelutaidoilla. Ensimmäinen versio laitteesta rakennettiin protokortille. Lopullista versiota varten teetettiin piirilevy.

Laitetta voidaan käyttää monenlaisten järjestelmien vikasietoisuuden testaukseen. Työn tekeminen lisäsi omaa tietämystä mikrokontrollereiden ohjelmoinnista ja laitesuunnittelusta. Valmista laitetta on mahdollista kehittää vielä eteenpäin ohjelmiston avulla.

Lähteet

Atmel Corporation, Industry Leader in the Design and Manufacture of Advanced Semiconductors [online] [viitattu 9.9.2010] http://www.atmel.com/

Atmel Corporation 2010, ATmega88 Datasheet [pdf] [viitattu 1.9.2010] http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf

Atmel Corporation, Parametric Product Table 2010 [online] [viitattu 1.11.2010]. http://www.atmel.com/dyn/products/param_table.asp?family_id=607&OrderBy=part_no&Direction=ASC

Cadence OrCAD Solutions 2010 [online] [viitattu 3.9.2010] http://www.cadence.com/products/orcad/pages/default.aspx

Fortum 2010 [online] [viitattu 11.11.2010] http://www.fortum.fi/fi/document.asp?path=14020;14028;31772;31773;31782;31795;31900;32214

International Rectifier Design Tips n.d. [pdf] [viitattu 1.10.2010] http://www.irf.com/technical-info/designtp/dt94-5.pdf

National Semiconductor 2005, LM2675 datasheet [pdf] [viitattu 27.9.2010] http://www.national.com/ds/LM/LM2675.pdf

Sähköala 2006 [online] [viitattu 15.10.2010] http://www.sahkoala.fi/ajankohtaista/artikkeleita/sahkotekniikka/fi_FI/sahkon_laatu/

Texas Instruments, MAX232 Datasheet 2004 [pdf] [viitattu 3.11.2010] http://focus.ti.com/lit/ds/symlink/max232.pdf

Liiteet

Liite 1: Ohjelmalistaus

```
/*****************************************************************************
Project :       opinnaytetyo.c
Hardware:       STK500 + ATmega88
Software:       AVR Studio 4.18
Date:           1.12.2010
Author:         KONE Corporation | hatmir
Comments:       v. 1.10
*****************************************************************************/


// SYMBOLISTEN NIMIEN MÄÄRITTELY
#define F_CPU 8000000UL                         // prosessorin kellotaajuuden määrittely
#define BAUD 9600                               // sarjaliikennenopeus
#define MYUBRR F_CPU/16/BAUD-1                  // bittinopeuden laskenta 8 MHz
#define UART_RX_BUFFER_ADDR 0x0200              // sarjaliikennepuskurin osoite
#define UART_RX_BUFFER_SIZE 0x0800              // sarjaliikennepuskurin koko


// KIRJASTOJEN SISÄLLYTYS
#include <stdio.h>                              // I/O -palvelut
#include <stdlib.h>                             // perusmakrot ja -funktiot
#include <avr/io.h>                             // I/O-määrittelyt
#include <avr/interrupt.h>                      // keskeytyspalvelut
#include <avr/pgmspace.h>                       // rajapintämäärittelyjä
#include "avrlibdefs.h"                         // hyödyllisiä määrittelyjä
#include "avrlibtypes.h"                        // muuttujien tyyppimäärittelyt
#include "buffer.h"                             // merkkipuskuri bittien käsittelyyn
#include "param.h"                              // parametrien tallennus- ja
                                                // latausfunktiot

// GLOBAALIEN MUUTTUJIEN MÄÄRITTELY
char unsigned rs232_data;                       // RS232 data
cBuffer uartRxBuffer;                           // RS232 puskuri
volatile unsigned char statusFlag = 1;          // ohjelmiston tilalippu, 1=run 2=hold
volatile unsigned char parametersReady = 1;     // apumuuttuja parametrien lataustilaa
                                                // varten
volatile unsigned char start = 1;               // apumuuttuja käynnistystä varten
volatile unsigned char breakStatus = 1;         // katkotilanne
volatile unsigned long int timeCounter = 0;     // aikalaskuri
const char stopParamSet='X';                    // parametrien syöttö loppuu kun
                                                // syötetään 'X'
const char viewCount   ='z';                    // näyttää katkojen lukumäärän kun
                                                // syötetään 'z'
const char makeBreak   ='c';                    // tekee katkon heti kun painetaan 'c'
volatile unsigned int breakCounter = 0;         // katkolaskuri


// FUNKTIOIDEN PROTOTYYPIT
static int uart_putchar(char c, FILE *stream);  // printf
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, // printf
NULL, _FDEV_SETUP_WRITE);
```

(jatkuu)

```
void USART_Init( unsigned int ubrr);                    // USART alustusfunkiton prototyyppi
int write_RS232(char c);                                // sarjaliikennekirjoitus funktion
                                                        // prototyyppi


// SARJALIIKENNEKESKEYTYSVEKTORI                        // sarjaliikennekeskeytysvektori
SIGNAL(SIG_USART_RECV)
{
while(!UCSR0A & (1<< 7));                               // odota että data on vastaanotettu
rs232_data=UDR0;                                        // merkki rs232_data muuttujaan
if(rs232_data == stopParamSet)                          // jos merkki on 'X'
   {
      statusFlag = 1;                                   // tilalippu run-tilaan
      parametersReady = 1;                              // parametrit valmiit
                                                        // tallennukseen/lataukseen
      start = 0;                                        // start muuttuja nollaksi
   }
   else if(rs232_data == viewCount)                     // näyttää katkojen lukumäärän
   {
      printf("count          : %d \n", breakCounter);
   }
   else if(rs232_data == makeBreak)                     // tekee katkon c-nappia painamalla
   {
      breakStatus = 0;                                  // katkostatus nollaksi
      timeCounter = 0;                                  // laskuri nollaksi
      statusFlag = 1;                                   // uudelleenkäynnistys, jos hold-tilassa
   }
   else if(rs232_data == '0'|| rs232_data == '1'||
           rs232_data == '2'|| rs232_data == '3'||
           rs232_data == '4'|| rs232_data == '5'||      // syötteen tarkistusta
           rs232_data == '6'|| rs232_data == '7'||
           rs232_data == '8'|| rs232_data == '9'||
           rs232_data == ' ')
   {
      statusFlag = 2;                                   // status wait-tilaan
      bufferAddToEnd(&uartRxBuffer, rs232_data);        // merkki puskuriin
      write_RS232(rs232_data);                          // kaiutus takaisin sarjaliikenneporttiin
   }
}

// FUNKTIOIDEN TOTEUTUKSET
static int uart_putchar(char c, FILE *stream)           // printf-funktio
{
   if (c == '\n')
      uart_putchar('\r', stream);
      while ( !( UCSR0A & (1<<UDRE0)) );                // odota kunnes datarekisteri tyhjä
   UDR0 = c;
   return 0;
}

void USART_Init( unsigned int ubrr)                     // USART alustus
{
   UCSR0A = 0x00;                                       // tyhjentää kontrolli ja statusrekisterin A
   UCSR0B = 0x98;                                       // lähetin ja vastaanotin päälle
   UCSR0C = 0x06;                                       // formaatti 8 databittiä, 1 stop bitti
   UBRR0H = (ubrr>>8);                                  // baudinopeus high-osa
   UBRR0L = ubrr;                                       // baudinopeus low-osa
}
```

(jatkuu)

```
int write_RS232(char c)                                 // merkin kirjoitus sarjaliikenneporttiin
{
   while ( !( UCSR0A & (1<<UDRE0)) );                   // odota kunnes datarekisteri tyhjä
    UDR0=c;                                             // merkin kirjoitus
   return c;
}


// PÄÄOHJELMA
int main(void)
{
   // ALUSTUSTOIMENPITEET
   DDRC = 0x3F;                                         // portin suunta ulos
   DDRD = 0x0C;                                         // portin suunta sisään/ulos

   USART_Init(MYUBRR);                                  // USART alustusfunkion kutsu

   stdout = &mystdout;                                  // printf

   bufferInit(&uartRxBuffer,                            // sarjaliikennepuskurin alustusfunktion
              (u08*) UART_RX_BUFFER_ADDR,               // kutsu
                   UART_RX_BUFFER_SIZE);

   TCCR0A |= (1 << WGM01);                              // ajastin 0 CTC-tilaan

   TCCR0B |= ((1 << CS10) | (1 << CS11));               // millisekuntiajastin esijakajalla 64,
                                                        // tavoitearvo 124
   OCR0A = 124;                                         // Aseta CTC vertailuarvo 0.001 Hz:iin
                                                        // 8MHz kellotaajuudella, kun esijakaja
                                                        // on 64



   printf("\n(c) KONE Corporation | hatmir");           // alkuteksti

   volatile unsigned long int disable = 0;              // enable aika muuttuja
   volatile unsigned long int enable = 0;               // disable aika muuttuja
   volatile unsigned int channel = 1;                   // ulostulon oletusarvo
   volatile unsigned int numberOfBreaks = 0;            // katkojen lukumäärä muuttuja
   int secondCounter = 1000;                            // apulaskuri sekuntien laskemiseksi

   char table[18];                                      // aputaulukko
   char tableDisable[6];                                // taulukko disable-ajan alkioille
   char tableEnable[6];                                 // taulukko enable-ajan alkioille
   char tableNumberOfBreaks[4];                         // taulukko katkojen lukumäärän
                                                        // alkioille
   char tableChannel[1];                                // taulukko ulostulokanavalle


   // OHJELMASILMUKKA
   while(1)
   {
      if(statusFlag == 1)                               // jos tilalippu on run-tilassa
      {
         // PARAMETRIEN LATAUS/TALLENNUS
         if(parametersReady == 1)                       // jos parametersReady == 1
         {
            cli();

            PORTC=0x04;
            if(start == 0)                              // jos startti == 0, parametrit bufferista
```

(jatkuu)

```
                                                    // muuttujiin ja eeprommiin
{
   for(int i=0; i<17; i++)                          // luvut bufferista taulukkoon
   {
      table[i] = bufferGetFromFront(&uartRxBuffer);

      if(table[i]=='0' && table[i-1]==' ')          // jos merkki 0 ja edellinen merkki väli
      {
         table[i]=' ';                              // merkki välilyönniksi
      }
   }

   uartRxBuffer.datalength = 0;                     // bufferin tyhjennys

   tableChannel[0]  = table[0];                     // ulostulon arvo

   tableDisable[0] = table[2];
   tableDisable[1] = table[3];
   tableDisable[2] = table[4];                      // disabletaulukon täyttö
   tableDisable[3] = table[5];
   tableDisable[4] = table[6];

   tableEnable[0] = table[8];
   tableEnable[1] = table[9];
   tableEnable[2] = table[10];                      // enabletaulukon täyttö
   tableEnable[3] = table[11];
   tableEnable[4] = table[12];

   tableNumberOfBreaks[0] = table[14];
   tableNumberOfBreaks[1] = table[15];              // katkojen määrä -taulukon täyttö
   tableNumberOfBreaks[2] = table[16];

   disable = strtoul (tableDisable,NULL,0);         // disable arvot taulukosta
                                                    // disable muuttujaan
   enable = strtoul (tableEnable,NULL,0);           // enable arvot taulukosta
                                                    // enable muuttujaan
   numberOfBreaks = atoi (tableNumberOfBreaks);     // katkojen lukumäärä taulukosta
                                                    // numberOfBreaks muuttujaan
   channel = atoi (tableChannel);                   // channel muuttuja taulukosta

   breakCounter = 0;                                // katkolaskurin arvo nollaksi

   paramStore((void*) &disable, (void*)2,4);          // disable EEPROMmiin
   paramStore((void*) &enable, (void*)6,4);           // enable EEPROMmiin
   paramStore((void*) &numberOfBreaks, (void*)10,2);  // numberOfBreaks EEPROMmiin
   paramStore((void*) &channel, (void*)12,2);         // channel EEPROMmiin
}
start = 1;                                          // start-tila ykköseksi
parametersReady = 0;                                // parametrien tallennus/lataus valmis

paramLoad((void*) &disable, (void*)2,4);            // EEPROM lataus
paramLoad((void*) &enable, (void*)6,4);             // EEPROM lataus
paramLoad((void*) &numberOfBreaks, (void*)10,2);    // EEPROM lataus
paramLoad((void*) &channel, (void*)12,2);           // EEPROM lataus

printf("\nPARAMETERS:\n");
printf("channel        : %d \n",channel);
printf("break            (ms)   : %ld \n", disable);     // parametrien tulostus
printf("interval  (s)   : %ld \n", enable);              // sarjaliikenneporttiin
printf("number of breaks: %d \n",numberOfBreaks);
```

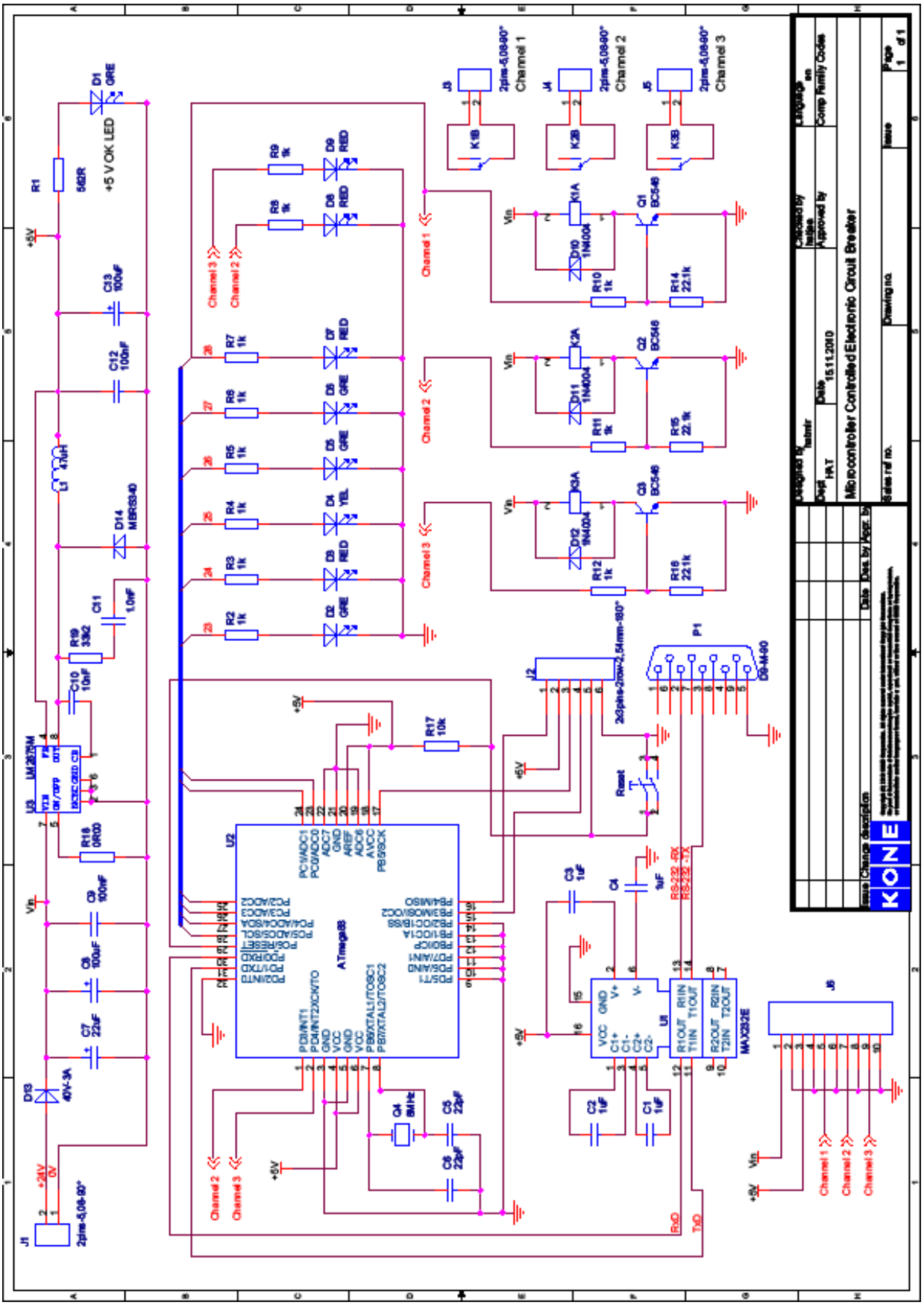
(jatkuu)

```
   timeCounter = enable;                    // aikalaskurin arvoksi enable

   if(channel == 0)                         // kanavan tarkistus
   {
      statusFlag = 2;                       // jos == 0, takaisin odotustilaan
   }
   if(numberOfBreaks > 0 )
   {
      PORTC=0x11;
   }
   else
   {
      PORTC=0x09;
   }

   sei();                                   // keskeytysten sallinta
}

// KATKONTASILMUKKA
else
{
   if(numberOfBreaks > 0)                   // run moodi n-kpl katkoja
   {
      if (TIFR0 & (1 << OCF0A))             // CTC-lipun vertailu
      {
         TIFR0 = (1 << OCF0A);              // CTC-lipun resetointi

         if(timeCounter > 0)                // jos aikalaskuri on > 0
         {
            if(breakStatus == 0)            // jos katkostatus == 0
            {
               secondCounter--;             // sekuntilaskurin vähennys

               if(secondCounter == 0)       // jos sekuntilaskuri == 0
               {
                  timeCounter--;            // pääaikalaskurin vähennys
                  secondCounter = 1000;     // sekuntilaskurin arvo tuhanteen
               }
            }
            else if(breakStatus == 1)       // jos status == 1, vähennetään
            {                               // vain pääaikalaskuria
               timeCounter--;
            }
         }
         else if(timeCounter == 0)          // jos aikalaskuri == 0
         {
            if(breakStatus == 0)            // jos katkotila == 0 (enable)
            {
               PORTC=0x31;                  // kanavan tilanvaihto
               timeCounter = disable;       // muuttujaan disablen arvo
               breakStatus = 1;             // katkotila = 1
               cli();                       // keskeytyskielto katkon ajaksi
            }
            else if(breakStatus == 1)       // jos katkotila == 1 (disable)
            {
               PORTC=0x11;                  // kanavan tilanvaihto
               timeCounter = enable;        // muuttujaan enablen arvo
               breakStatus = 0;             // katkotila = 0
               breakCounter++;              // katkolaskurin kasvatus
               numberOfBreaks--;            // katkojen määrän vähennys
```

(jatkuu)

```
                        sei();                          // keskeytysten sallinta katkon jäkeen

                        if(numberOfBreaks == 0)         // jos n-määrä katkoja tehty
                        {                               // siirrytään odotustilaan
                           statusFlag = 2;
                        }
                     }
                  }
               }
               else                                     // run-moodi jatkuva
               {
                  if (TIFR0 & (1 << OCF0A))             // CTC-lipun vertailu
                  {
                     TIFR0 = (1 << OCF0A);              // CTC-lipun resetointi

                     if(timeCounter > 0)                // jos aikalaskuri on > 0
                     {
                        if(breakStatus == 0)            // jos katkostatus == 0
                        {
                           secondCounter--;             // sekuntilaskurin vähennys

                           if(secondCounter == 0)       // jos sekuntilaskuri == 0
                           {
                              timeCounter--;            // pääaikalaskurin vähennys
                              secondCounter = 1000;     // sekuntilaskurin arvo tuhanteen
                           }
                        }
                        else if(breakStatus == 1)       // jos status == 1, vähennetään
                        {                               // vain pääaikalaskuria
                           timeCounter--;
                        }
                     }
                     else if(timeCounter == 0)          // kun aika == 0
                     {
                        if(breakStatus == 0)            // jos katkotila = 0 (enable)
                        {
                           PORTC=0x29;                  // kanavan tilanvaihto
                           timeCounter = disable;       // muuttujaan disablen arvo
                           breakStatus = 1;             // katkotila = 1
                           cli();                       // keskeytyskielto katkon ajaksi
                        }
                        else if(breakStatus == 1)       // jos katkotila == 1 (disable)
                        {
                           PORTC=0x09;                  // kanavan tilanvaihto
                           timeCounter = enable;        // muuttujaan enable
                           breakStatus = 0;             // katkotila = 0
                           breakCounter++;              // katkolaskurin kasvatus
                           sei();                       // keskeytysten sallinta katkon jäkeen
                        }
                     }
                  }
               }
            }
         }
         else if(statusFlag == 2)
         {
            PORTC=0x02;                                 // odotustila
         }
      }
   }
```

Liite 2: Prosessorikortin piirikaavio



Liite 3: IGBT-ohjauksen periaatekuva