Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut järjestelmät

2010

Ville Haavisto

MIKRO-OHJAINAVUSTEISEN RUUVITUNNISTINJÄRJESTELMÄN SUUNNITTELU JA TOTEUTUS

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut järjestelmät

Kesäkuu 2010 | 29 sivua

Ohjaaja Vesa Torvinen

Ville Haavisto

MIKRO-OHJAINAVUSTEISEN RUUVITUNNISTINJÄRJESTELMÄN SUUNNITTELU JA TOTEUTUS

Matkapuhelimen kokoonpanoprosessissa virhetarkastelu on tärkeää, ettei viallinen tuote joudu kuluttajan käsiin. Tehokkainta on poistaa tai korjata virheellinen tuote mahdollisimman aikaisessa vaiheessa. Yksi kokoonpanoprosessin vaiheista on piirilevyn liittäminen puhelimen mekaniiikkaan.

Tässä opinnäytetyöprojektissa suunnitellaan ja tuotteistetaan mikro-ohjainavusteinen virhetarkastelujärjestelmä tuotannon apuvälineeksi valmiina olevaan ohjausmekaniikkaan. Tavoitteena on nopeuttaa virhetarkastelua, jotta viallinen kappale saataisiin joko korjattua tai poistettua tuotantojonosta mahdollisimman tehokkaasti.

Projekti toteutettiin 19.1.2009 – 7.4.2010 välisenä aikana Nokia Oyj:n tilaamana Sorv-Elektro Oy:lle. Mikro-ohjaimena käytetään AVR Atmega 32-mikro-ohjainta, ohjelmointikielenä C. Ohjelmointiympäristönä on avoimen lähdekoodin AVR-GCC.

Järjestelmä nopeuttaa Nokia Oyj:n virhetarkastelua ja nostaa tuotantotehokkuutta tässä kokoonpanovaiheessa.

ASIASANAT:

mikro-ohjain, matkapuhelimet, tuotanto, apuväline, tunnistin, optoelektroniikka

BACHELOR´S THESIS | ABSTRACT

UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded systems

June 2010 | 29 pages

Instructor Vesa Torvinen

Ville Haavisto

DESIGNING AND MANUFACTURING A MICROCONTROLLER DRIVEN ASSEMBLY SENSOR

Flaw detection is a key issue in the assembly cycle of mobile phones. Ensuring that a faulty device doesn't reach the end user is vital. One part of the assembly cycle is attaching the circuit board to the covers and mechanics of the phone. Helping the assembly operator to detect a flawed assembly so that it can be fixed or discarded accordingly eases the flaw detection at the end tester.

The aim of this thesis project is to design and build a microcontroller driven sensor system that is integrateable to existing assembly mechanics.

This project was built around AVR's Atmega32 microcontroller. The code is written in C using the open source AVR-GCC environment. The project was done for Sorv-Elektro Ltd assigned by Nokia Plc

The system benefits the assembly flaw detection progress and helps build up production efficiency.

KEYWORDS:

microcontrollers, mobile phones, production, sensor, optoelectronics

SISÄLTÖ

KUVAT

TAULUKOT

KAAVAT

LIITTEET

1 Johdanto

Matkapuhelinten valmistuskustannuksien vähentämiseksi on virheellisten kappaleiden poistaminen tuotantojonosta mahdollisimman pian virheen tapahtumisen jälkeen ensiarvoisen tärkeää. Yksi tuotantovaiheista on piirilevyn liittäminen kuoreen ruuvien avulla. Tässä käytetään apuna koontajigiä, joka ohjaa ruuvit oikeisiin kohtiin pitäen ne samalla suorassa. Käyttäjävirheinä tässä tuotantovaiheessa on havaittu ruuvin puutteellinen syöttö sekä inhimillinen erehdys, jolloin yksi tai useampi kiinnityskohta jää ruuvaamatta.

Manuaalinen tarkistus nostaa tuotantoaikoja ja -kustannuksia, joten tarkistus on saatava automaattiseksi. Tarkistusautomaationa on aikaisemmin kokeiltu ns. konesilmää, joka ottaa digitaalikuvan tuotteesta, ja vertaa sitä mallikuvaan. Konesilmän tarkkuus on kuitenkin havaittu riittämättömäksi. Toisenlainen lähtökohta tuotteen oikeellisuuden tarkistamiseen on liittää ruuvitunnistin suoraan koontajigiin.

Tämän opinnäytetyön päätavoitteena on suunnitella ja tuotteistaa modulaarinen ruuvisensorijärjestelmä tuotannon käyttöön. Teoreettisesta opinnäytteestä poiketen tämä opinnäytetyö on kirjoitettu projektityöstä. Tämän takia opinnäytteen rakenne on tietoisesti valittu käytännönläheiseksi. [7]

Luvussa kaksi käsitellään projektin toimivuuden kannalta tärkeimmät toimintaperiaatteet. Luvussa kolme tarkastellaan elektronisia komponentteja ja piirilevyn kytkentöjä. Luku neljä käsittelee ohjelmarakennetta ja mikro-ohjaimen eri osien käyttöönottoa ja käsittelyä rekisterimuutoksilla. Lopetuskappaleessa reflektoidaan projektin toteutusta.

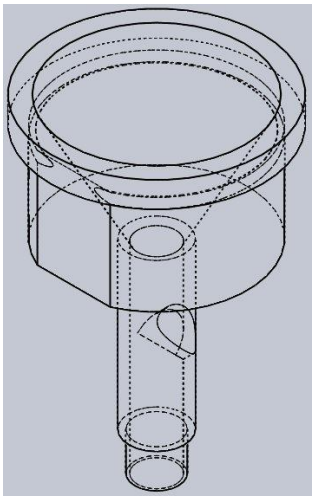
2 Ruuvisensorille asetetut vaatimukset

Ennen projektin aloittamista sille pitää luoda tietyt perusvaatimukset ja toimintaperiaatteet, joiden mukaan tuotetta voidaan alkaa suunnitella. Nämä vaihtelevat tuotanto- ja raaka-ainekustannuksista visuaalisen ilmeen määrittämiseen.

2.1 Perustoimintaperiaatteet

Ensimmäinen sensorille asetettu vaatimus on sen toiminnan perusperiaate. Sensorin pitää pystyä tulkitsemaan, onko kuvan 1 mukaisen ohjuriholkin läpi kulkenut ruuvi. Projektin edetessä tämä vaatimus muuttui niin, että pelkän ruuviväänтimen kärjen havaitseminen riittää. [1][4]

Toinen perustoimintaperiaate on, ettei ruuvitunnistinjärjestelmän elektroniikka tai ohjelmointi aiheuta rikkoutuessaan katkoa koontatyöhön. Mekaaniset viat, kuten ohjuriholkin rikkoutuminen, tai paineilmajärjestelmän viat, aiheuttavat katkon tuotannossa, mutta elektroniikan tai ohjelmoinnin osalta tätä ei sallita. [4]

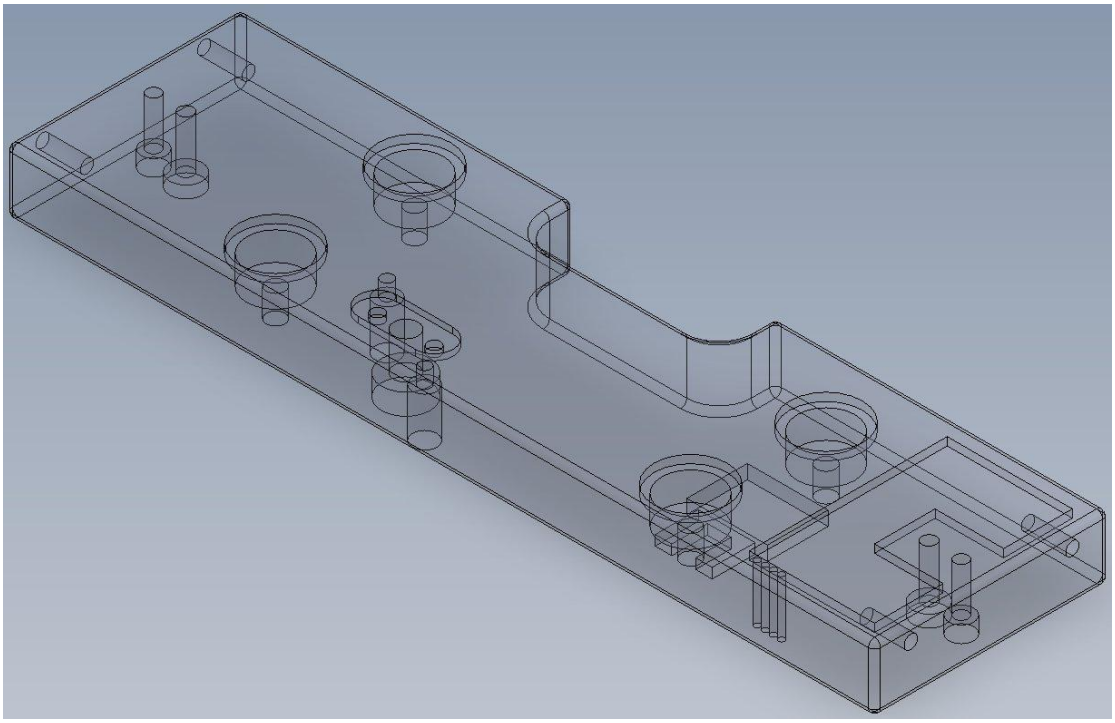


Kuva 1 Ruuvin ohjuriholkin 3D-mallinnus.

2.2 Koon määrittävät tekijät

Ruuvisensorin tunnistinosan pitää mahtua kuvan kaksi mukaisen koontajigin kannen ja koottavan matkapuhelimen väliin, jotta koontajigin toiminta ei muutu. Koontajigin kansi on, mallista riippumatta, materiaaliltaan läpinäkyvää akryyliä. Läpinäkyvyyden on säilyttävä, joten piirilevy jolle sensorin komponentit ladotaan, on pysyttävä pienenä, tai se on asennettava muualle kuin itse kanteen. [4]

Piirilevy voidaan upottaa kanteen, jolloin suurempia komponentteja voidaan käyttää. Piirilevyn upottaminen kannen alapuolelle helpottaa komponenttien sijoittelua ja tuotannossa järjestelmän kokoonpanoa. Kuvan 2 mukaisessa mallissa piirilevy on upotettu kannen yläosaan, jolloin sensorikomponentti upotetaan kannen alaosaan, ja ohjaussähkö tuodaan erillisillä johdoilla kannen läpi sensorilta piirilevylle.



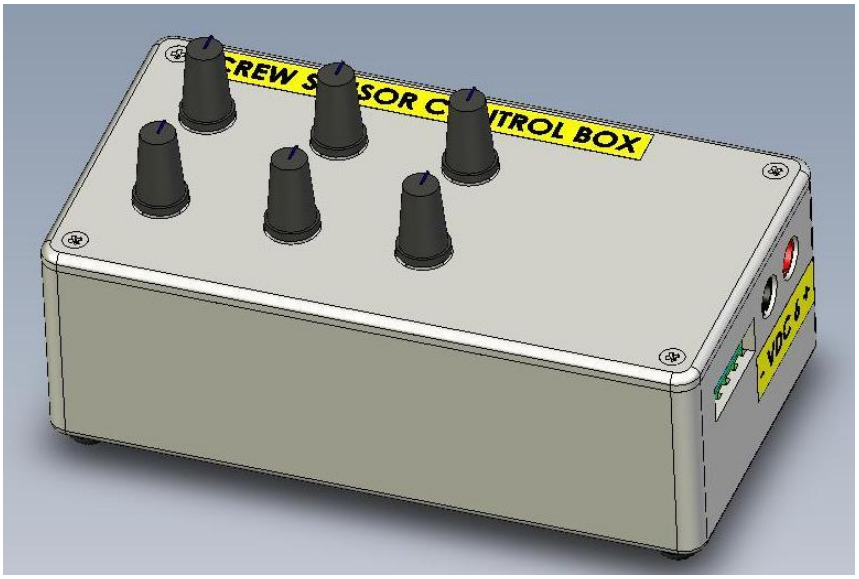
Kuva 2 Koontajigin kannen prototyypin 3D-mallinnus.

2.3 Modulaarisuus

Koska puhelimet ovat erilaisia, tarvitaan eri malleille omanlaisensa koontajigi. Kiinnitysruuvien määrä, paikka ja koko vaihtelevat. Tuotantokustannusten kannalta onkin viisasta tuotteistaa ruuvitunnistin yleismallisena niin pitkälle, kuin se on tuotannollisesti mahdollista.

Ruuvisensorijärjestelmä on suunniteltu alusta alkaen siten, että peruskytkentä pysyy laitteesta toiseen samana. Näin ollen ohjelmamuutoksia ei tarvitse tehdä siirryttäessä laitteesta toiseen. Erillään oleva hallintaosa, jonka 3d-mallinnus on nähtävissä kuvassa kolme, on yhteensopiva samalla pohjalla tehtyjen ohjausjärjestelmien kanssa, ja tukee kahdesta kuuteen sensoria.

Asiakkaan ei tarvitse tilata uutta hallintaosaa siirryttäessä uuteen tuotteeseen, vaan tämä pystyy käyttämään edellisen, jo poistuneen, tuotteen hallintaosaa uuden tuotteen kanssa. Samalla valmistajan ei tarvitse suunnitella ja ohjelmoida uutta laitetta jokaisen tuotteen vaatimusten mukaan. Ainoa muutos tuotteesta toiseen siirryttäessä on kanteen asennettava piirilevy. Komponentit pysyvät samana, kytkentä muuttumattomana ja vain piirilevyn komponenttisijoittelu ja ulkomuoto muuttuvat.



Kuva 3 Hallintaosan 3D-mallinnus.

3 Kytkentä ja sen keskeiset komponentit

Ruuvisensorin toimivuuden kannalta keskeisiä komponentteja on kaksi. Tunnistuksen hoitaa optokatkoja, ja järjestelmää ohjaa mikro-ohjain. Optokatkojat ja loppukäyttäjälle näkyvät komponentit, kuten ledit ja huoltokytkin, ovat koontajigin kannessa. Asentajan käyttöön tarkoitettu hallintaosa on yhdistetty koontajigiin 14-napaisella noin 90 cm pitkällä lattakaapelilla. Ohjauselektroniikka, jännitteensyöttö ja mikro-ohjain sijaitsevat tässä yleismallisessa ohjausosassa (Kuva 3).

Käyttöjänniterajat kytkennälle ovat 5 V – 5.7 V DC, jota tasataan LM1117-regulaattorilla 3.3 V:iin. Tulevissa järjestelmissä käyttöjännitealuetta kasvatetaan 5 V-24 V DC, jotta järjestelmä olisi paremmin yhteensopiva Nokia Oyj:n tuotannossa käyttämän laitteiston kanssa. [4]

3.1 Optokatkoja

Tunnistimena toimii ITR 8402 -optokatkoja, jonka on valmistanut Everlight Electronic Co. LTD. ITR 8402 sisältää NPN fototransistorin, joka on kytketty infrapunalediin. Transistori saa ohjausjännitteensä lediltä, joka heijastaa valoa katkojan koteloon tehdyn aukon kautta. Transistori reagoi vain infrapunaledin heijastamaan valoon ja asettaa transistorin katkojan lähdön positiiviseksi. Ulkoisten valonlähteiden tuottama häiriö on suodatettu upottamalla fototransistori sekä infrapunaledi mustan muovikuoren sisälle. Pieni määrä ulkoista valoa pääsee kuoren sisään, mutta sen aiheuttama häiriö on niin pieni, että transistorin toiminta ei muutu. [5]

3.1.1 Fototransistori

Transistori on puolijohdekomponentti, jota käytetään kytkimenä tai vahvistimena. Transistorit on yleisesti jaettu kanava- ja bipolaaritransistoreihin. Bipolaaritransistori toimii kytkennöissä virran vahvistimena. Sen kaksi alaluokkaa ovat NPN- ja PNP-tyypin vahvistimet. ITR 8402:n sisällä oleva fototransistori on NPN-tyypin bipolaaritransistori. [5][6]

Bipolaaritransistorilla on kolme jalkaa: kollektori (C), emitteri (E) ja kanta (B). NPN-tyypin transistorissa ohjausvirta johdetaan kannalle ja käyttösähkön kulkusuunta on kollektorilta emitterille. Fototransistorissa kantaan johdettava ohjausvirta saadaan valosta – yleensä käytetään infrapuna-aallonpituutta. [5][6]

Transistorin kuormavastuksena käytetään datalehden suosituksen mukaisesti 1 kΩ:n vastusta. Kuormavastus on kytketty kollektorijalkaan. Transistorin käyttöjännite on sama kuin muilla kytkennän komponenteilla (3,3 V). Optokatkojan transistoripuoli on kytketty mikro-ohjaimen sisääntuloon kuormavastuksen ja kollektorin välistä. Ulostulojännite muuttuu ratkaisevasti, kun lukuhaarukan väli peitetään. Optokatkojan ollessa kytkettynä kuorma- ja etuvastuksiin 3,3 V käyttöjännitteellä, ulostulojännitteen (U_{out}) keskiarvoksi saadaan 3,280 V, kun lukuhaarukan kärkiväli on katkaistu. Mikroprosessorille kytkettynä optokatkoja, tyhjällä kärkivälillä U_{out} , on keskiarvoltaan 0,755 V. Tätä arvoa käyttämällä voidaan myöhemmin määrittää raja-arvo tunnistinten valintaan mikro-ohjaimen käynnistysohjelmassa. [2][5]

3.1.2 Infrapunaledi

LED (Light Emitting Diode) on nimensä mukaisesti valoa säteilevä diodi. Ledin väri määräytyy sen valmistuksessa käytetyn seoksen mukaan. ITR 8402:n sisällä oleva ledi on materiaaliltaan alumiinin, galliumin ja arseenin seos (AlGaAs), ja se säteilee infrapuna-aallonpituudella. [5]

Ledin kynnysjännite on 1,5 V, ja se tarvitsee 20 mA virtaa toimiakseen. Koska ledin käyttöjännite on 3.3 V, ja virtaa saattaa liikkua kytkennässä jopa 2 A, tarvitaan ledille etuvastus. Vastusarvo lasketaan Ohmin lain mukaisesti kaavan yksi mukaisesti, jossa R on etuvastuksen arvo, U on käyttöjännitteen ja kynnysjännitteen erotus ja I ledin tarvitsema virta. Laskemalla R tällä kaavan yksi mukaisesti R = (3,3 V–1,5 V)/0,02 A saadaan tulokseksi 90 Ω. Etuvastukseksi valitaan RETMA (Radio Electronics Television Manufacturers Association) –sarjan mukainen seuraava korkeampi vastus 100 Ω. [5][6]

Kaava 1 Vastusarvon laskeminen Ohmin lain mukaan.

$$R = \frac{U}{I}$$

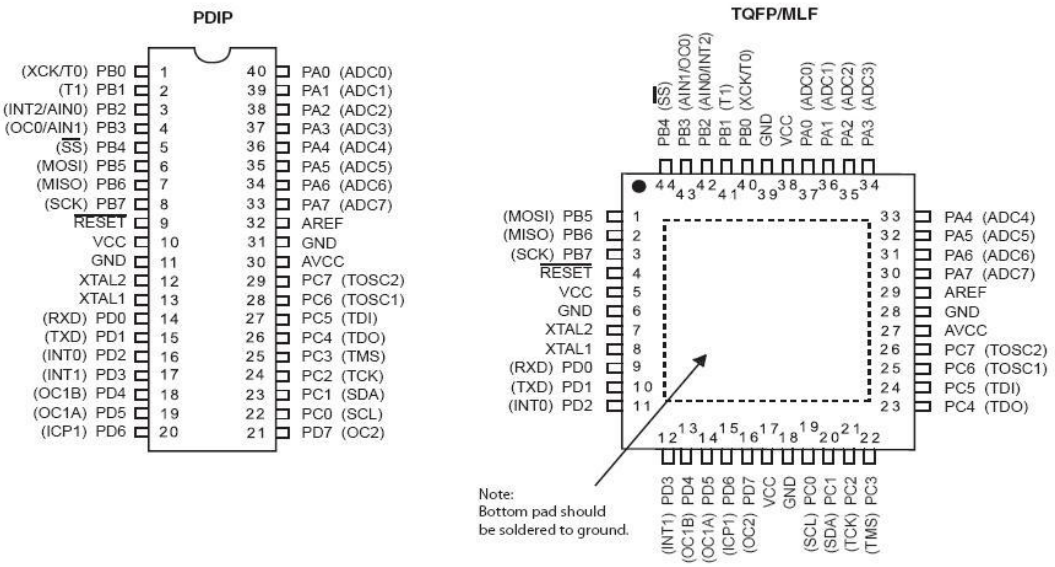
3.2 Mikroprosessori

ITR 8402:n lähettämä ulostulojännite tulkitaan Atmelin ATMEGA32 mikro-ohjaimella. MEGA32 on 8-bittinen RISC (Reduced Instruction Set Computer) –mikro-ohjain, jossa on 32 tuhatta bittiä ohjelmoitavaa Flash-muistia ja 1024 -bittinen EEPROM (Electronically Erasable Programmable Read-Only Memory) -ikimuisti. [2][3]

Mikroprosessorin tehtävä kytkennässä on tarkkailla optokatkojalta tulevaa ulostulojännitettä ja tulkita tästä johtuvia loogisia tilamuutoksia. Kun kytkentään lisätään useampi optokatkoja, on mikro-ohjaimen ohjelmallisesti pystyttävä tulkitsemaan oikea koontajärjestys sekä ilmoitettava käyttäjälle onnistuneesta koonnasta.

ATMega32-mikro-ohjaimella on pakkaustavasta riippuen joko 40- tai 44-jalkaa, kuten on nähtävillä kuvassa neljä. Kaksirivinen PDIP (Plastic Dual-In-Line Package) on 40-jalkainen kotelo. Neliön muotoinen TQFP (Thin Quad Flat Pack) on 44-jalkainen J-jalkainen kotelo. TQFP-kotelon kanssa saman kokoinen MLF (Micro Lead Frame) on kotelo, jossa ei ole konkreettisia jalkoja, vaan kotelon alla on juotospadit. Projektin prototyyppivaiheessa on järkeväää käyttää

PDIP -koteloa, koska sen jalkajako 2,54 mm sopii reikälevylle ja juottaminen helpottuu. Lisäksi on mahdollista käyttää yhteensopivaa kantaa (eng. socket), joka voidaan mieltää mikro-ohjaimen pistorasiaksi. Kantaa käytettäessä ei juotostyötä tarvitse tehdä suoraan mikro-ohjaimen jalkoihin. Näin suojataan itse prosessoria lämpövaurioilta. Kun kytkentä siirretään piirilevylle, otetaan käyttöön TQFP-kotelotyyppi. [2]



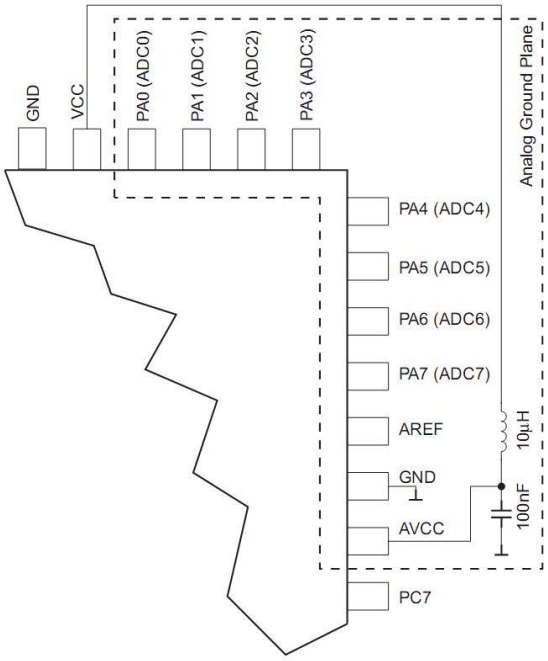
Kuva 4 PDIP- ja TQFP-kotelotyyppien jalkakuvio (AtMega32 Datasheet).

3.2.1 Mikroprosessorin kytkennät

Koska mikro-ohjaimen kaikkia ominaisuuksia ei tarvita, on tarpeellista kytkeä vain tarvittavat jalat komponentteihin. Kytkemättömät jalat juotetaan kiinni juotospadeihin, mutta padit jätetään kellumaan, eikä niitä maadoiteta.

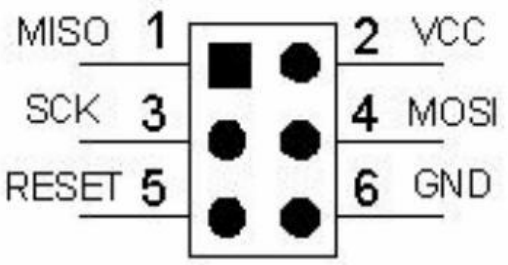
Mikro-ohjaimen portin A kaikki pinnit 0–7 kytketään järjestyksessä optokatkojien 1–6 ulostulolinjoihin U_{out1}–$U_{out6.}$, jotka ovat kytkettyinä kiertokytkimiin 1–6. Käyttöjännite 3,3 V kytketään LM1117 regulaattorin output-jalasta mikro-ohjaimen VCC-väylään. VCC-väylä kytketään AVCC väylään kuvassa kolme näkyvän datalehden suosituksen mukaisen alipäästösuotimen läpi. Alipäästösuodin koostuu AVCC- ja VCC-pintojen välille kytketystä 10 µH kelasta sekä kelaan rinnankytketystä 100 nF kondensaattorista. LC-kytkentä suodattaa

virtapiikkiikien kulkeutumisen AVCC-pinnalle. AVCC-pinnalle tulevat virtapiikit voivat vaarantaa analogi-muunnoksen tarkkuuden, joten ne on pyrittävä minimoimaan. [2][3]



Kuva 5 Alipäästökytkentä AVCC- ja VCC-pintojen yhteenliittämiseksi (ATMega32 Datasheet).

Ohjelmointiliitäntänä käytetään kuusinapaista kaksirivistä piikkirimaa 2,54 mm jaolla. Piikkirimaan kytketään AVRISP (AVR In-System Programmer)-ohjelmointilaite.



Kuva 6 AVRISP MkII liittimen pinout (AVRISP MkII Users Guide).

3.3 Optokatkojan käyttö kytkimenä

Koska optokatkojan ulostulojännite on analoginen, tulisi se tulkita analogisena. Mikroprosessori tulkitsee kuitenkin digitaalisia tilamuutoksia loogisesta 0-tilasta loogiseen 1-tilaan. Sisäisen AD-muuntimen käyttö antaa yhden mahdollisuuden toteuttaa tilamuutosvertailu, mutta sen käyttäminen hidastaa ohjelman ajamista ja aiheuttaa ylimääräisen koodin kirjoittamista.

Toinen lähestymistapa on hyväksikäyttää mikro-ohjaimen loogisen tilamuutoksen raja-arvoa. Karkeasti sanottuna looginen 0-tila on voimassa, kun sisääntulopinni on maassa, eli sille sisääntuleva jännite on 0 V. Looginen 1-tila saavutetaan, kun mikroprosessorin sisäinen kytkentä ylittää sille asetetun raja-arvon loogiselle tilamuutokselle. Tämä arvo annetaan mikroprosessorin datalehdellä sivulta 287 löytyvän taulukon yksi mukaan. [2]

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{IL}	Input Low Voltage except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC}=2.7 - 5.5 V_{CC}=4.5 - 5.5	-0.5		0.2 V_{CC}(1)	V
V_{IH}	Input High Voltage except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC}=2.7 - 5.5 V_{CC}=4.5 - 5.5	0.6 V_{CC}(2)		V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage XTAL1 pin	V_{CC}=2.7 - 5.5	-0.5		0.1 V_{CC}(1)	V
V_{IH1}	Input High Voltage XTAL1 pin	V_{CC}=2.7 - 5.5 V_{CC}=4.5 - 5.5	0.7 V_{CC}(2)		V_{CC} + 0.5	V
V_{IL2}	Input Low Voltage $\overline{\text{RESET}}$ pin	V_{CC}=2.7 - 5.5	-0.5		0.2 V_{CC}	V
V_{IH2}	Input High Voltage $\overline{\text{RESET}}$ pin	V_{CC}=2.7 - 5.5	0.9 V_{CC}(2)		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(3) (Ports A,B,C,D)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	Output High Voltage(4) (Ports A,B,C,D)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.2 2.2			V V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{pu}	I/O Pin Pull-up Resistor		20		50	kΩ

Taulukko 1 Atmega32-mikro-ohjaimen tasajänniteominaisuudet [Atmega32 datasheet, s.287].

Tarkastelemalla taulukkoa 1 havaitaan, että ylä- ja alaraja-arvo loogiselle tilamuutokselle voidaan laskea taulukossa annettujen tietojen avulla. Käytettäessä 3,3 V käyttöjännitettä mikro-ohjain tulkitsee sisääntulojännitteen 1:ksi jännitevälillä 1,98 V DC - 3,8 V DC kaavojen kaksi ja kolme mukaan (sivulla 16). Optokatkojan ulostulojännite sopii tähän toimintansa puolesta, koska katkojan kärkivälin ollessa avoin (infrapunalinja ei katkaistu) ulostulojännitealue on 0,03 V – 0,04 V, jonka mikro-ohjain tulkitsee nollana. Kärkivälin ollessa suljettu (infrapunalinja katkaistu) ulostulojännitealue on 3,18 V – 3,29 V. Toisin sanoen optokatkoja toimii itsessään ON–OFF-kytkimenä.

Kaava 2 Loogisen tilamuutoksen alaraja-arvon laskukaava.

$$V_{IH}^{min} = 0{,}6V * V_{cc}$$

Kaava 3 Loogisen tilamuutoksen yläraja-arvon laskukaava.

$$V_{IH}^{max} = V_{cc} + 0{,}5V$$

4 Mikro-ohjaimen ohjelmointi

Mikro-ohjain tulkitsee optokatkojan ulostulojännitettä ADC-toiminnon avulla (Analog to Digital Converter). ADC muuttaa optokatkojan lähettämän analogisen signaalin mikro-ohjaimen ymmärtämäksi digitaalikseksi signaaliksi. ATMega32 muuttaa analogisignaalin 10-bittiseksi, joko oikealle tai vasemmalle tasaten. Käytössä olevia kanavia on kahdeksan, joiden sisääntulot on sisäisesti kytketty A-portin pinneihin 0–7. [2]

AVR-mikro-ohjaimen ohjelmoinnissa käytetään yleisesti C-kieltä. C-kielen vahvuus mikro-ohjainohjelmoinnissa löytyy sen hierarkkisesta rakenteesta. C-kielinen ohjelma rakentuu yksinkertaisimmillaan peräkkäisistä yhden askeleen käskyistä. Tämä sopii hyvin laitteistoläheiseen ohjelmointiin, sillä konekieli

kostuu samalla tavalla yksinkertaisista komennoista esim. muuta, hae, lisää, vähennä. C-kielisen ohjelman korjaaminen, debuggaus, helpottuu, kun virheet voidaan yleisesti löytää korkeintaan yhden funktion sisältä, eikä tarvitse kahlata koko koodia läpi. [3]

AVR ei kuitenkaan ymmärrä C-kieltä, vaan ohjelma pitää kääntää konekielelle. Oikeasta standardoidusta ANSI-C (American National Standards Institute) –kielestä poiketen mikro-ohjainohjelmoinnissa käytetään vain ANSI-standardissa määriteltyä standardikirjastoa, jossa määritellään ne funktiot joita voidaan yleisesti kutsua useampaan eri ohjelmatilanteeseen. C-kielinen koodi pitää kääntää ennen Flash-muistiin ajoa heksamuotoon. Tätä varten on saatavilla useita kaupallisia ja ilmaisia kääntäjiä. Yleisimmin käytössä oleva kääntäjä on AVR-GCC (AVR GNU Compiler Collection) -kääntäjä, joka voidaan sulauttaa suoraan Atmelin AVR Studio-ohjelmointiympäristöön. AVR-GCC pystyy myös kääntämään C++-, Java- ja Fortran-ohjelmointikielisen koodin heksamuotoon, sekä yhdistämään mikro-ohjainten tarvitsemia funktioita ja rekisterikutsuja standardimuotoiseen ohjelmapuuhun. GCC on avoimen lähdekoodin ja yhteisön ylläpitämä projekti. [3]

4.1 Rekisterien manipulointi

Mikro-ohjaimen toiminta perustuu rekisteriarvoihin, joita muuttamalla saadaan aikaiseksi eri tapahtumia prosessorin sisällä. Rekisteribiteistä osa käsittelee suoraan jonkin toiminnon käynnistämistä tai sammuttamista, osa näiden tilamuutoksia ja osa I/O-porttien käyttäytymistä. Ruuvisensorijärjestelmää ohjelmoitaessa poiketaan suuresti C-kielisen koodin kirjoittamisesta, ja ohjelma toteutetaan lähes kokonaan rekisteriarvoja muuttamalla. [3]

Heti pääohjelman alussa tehdään funktiokutsu, jossa alustetaan A/D-muuntajan rekisteriarvot sellaiseen muotoon, että saadaan oikeanlainen A/D-muunnos ja asetettua mikro-ohjain ottamaan vastaan oikeanlainen vertailujännite ja

tulodata. Kaikki rekisteriarvojen muutokset ja seuraukset löytyvät mikro-ohjaimen datalehdeltä. Rekisterit ovat 8-bittisiä (0b00000000) ja bittien muutokset määrätään käskyllä, joka on muotoa ”REKISTERI TILAMUUTOS (1 << BITTI)”. Käskyllä muutetaan halutun rekisterin bittiä ”BITTI” nollasta ykköseksi. Bitin kääntö takaisin nollaan tehdään invertoimalla kääntö ~-merkillä: ”~(1 << BITTI)”. [3]

4.1.1 ADCSRA-rekisteri

ADCSRA (A/D Control and Status Register A) -rekisteri ohjaa A/D-muunninta. Rekisteristä muutetaan A/D-muuntimen tilaa ja sen toimintaa. Ensimmäinen toimenpide on asettaa datalehden mukaisesti ADPS (ADC Prescaler Select Bits) -rekisterin ADPS2-bitti ykköseksi, jotta saadaan A/D-muunnokseen tarvittava esijakaja (Taulukko 2). Esijakajaa tarvitaan, jotta saadaan aikaiseksi oikeanlainen muunnosnopeus kellotaajuudesta riippuen. Kellotaajuuden olisi sopivinta olla 50 kHz ja 200 kHz välillä hyvän tarkkuuden saavuttamiseksi. Koska ruuvisensori käyttää Mega32:n sisäistä 1 MHz kelloa, esijakajaksi voidaan valita kahdeksan tai 16, jolloin ADC-kellotaajuudeksi tulee: 1 000 000 Hz / 16 = 62,5 kHz tai 1000000 Hz / 8 = 125 kHz. Tämän projektin osalta sopivin valinta esijakajaksi on 16. [2] [3]

```
ADCSRA |= (1 << ADPS2);
```

Taulukko 2 Esijakajan valinta ADPSx-biteillä (jatkuu seuraavalla sivulla).

ADPS2	ADPS1	ADPS0	Esijakaja
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8

1	**0**	**0**	**16**
1	**0**	**1**	**32**
1	**1**	**0**	**64**
1	**1**	**1**	**128**

Seuraava ADCSRA-rekisterin muutos tehdään myös pääohjelman alussa. Kahdella rekisterimuutoksella asetetaan A/D-muunnos vapaasti juoksevaan muotoon (Free Running), jolloin sisääntuloporttia tarkkaillaan jatkuvasti, ja prosessori muuntaa koko ajan sisälle tulevaa analogisignaalia.

```
ADCSRA |= (1 << ADATE);
```

Toinen muutos käynnistää ensimmäisen muunnoksen, jonka jälkeen muunnin pysyy päällä, kunnes virta sammutetaan laitteesta tai muunnos pysäytetään ohjelmallisesti. Vaikka nämä kaksi käskyä ovat sidottuja toisiinsa Free Running -tilan käynnistämiseksi, on ehdottoman tärkeää, että jälkimmäinen käsky on viimeinen ADCSRA-rekisterin tilanmuutos. Jos yrittää käynnistää mittausta, ennen kuin muunnin on päällä, mitään ei tapahdu.

```
ADCSRA |= (1 << ADSC);
```

Ennen mittauksen aloitusta pitää vielä muuttaa A/D-muuntimen tila ”ON”-asentoon ja sallia ADC-riippuvaiset keskeytykset.

```
ADCSRA |= (1 << ADEN);
```

ADEN (ADC Enable) -bitin asettaminen ykköseksi käynnistää A/D-muuntimen sille jo asetetuilla arvoilla.

```
ADCSRA |= (1 << ADIE);
```

ADIE (ADC Interrupt Enable)-bitti hallitsee A/D-muunnoksesta riippuvaisia keskeytyksiä.

Tämän projektin kannalta sopivin tapa on käyttää edellä mainittua vapaasti juoksevaa tilaa. Muita tapoja A/D-muuntimen käyttöön kuitenkin on. A/D-muunto voidaan toteuttaa niin, että jokainen muunnostulos tallennetaan muistirekisteriin ja tarkastetaan sieltä ohjelmallisesti. A/D-muunninta voidaan myös käyttää analogikomparaattorina. A/D-muunnin asetetaan eri tiloihin vaihtamalla ADPSx-bittien tiloja datalehden ohjeiden mukaisesti.

4.1.2 Ajastimet

Mikroprosessorin kellotaajuutta hyväksikäyttämällä voidaan suorittaa ajastettuja toimintoja, mm. pitää yllä kellonaikaa tai tuottaa taajuusohjattua pulssia. Yksi kytkennän komponenteista on pietsosummeri. Summereita on erilaisia, jotkin mallit vaativat analogisignaalin, jotta summeri tuottaisi ääntä. Analogisignaalin tuottamiseksi on ensin asetettava AVR-mikro-ohjaimen sisäinen ajastinrekisteri toimimaan halutulla tavalla. Tämä mahdollisuus on otettu kytkennässä huomioon kytkemällä pietsosummeri PWM-porttiin. [2][3]

Ohjelmassa käytetään yhtä ATMega32-mikro-ohjaimen 8-bittisestä laskurista, joka pitää valita sen mukaan, miten PWM-ohjattava komponentti on kytketty ja mitä toimintoja laskurilta tarvitaan. Koska pietsosummeri on kytketty portin D pinniin 7, otetaan käyttöön Timer/Counter2, jonka ulostulo on kytketty juuri D-portin I/O-pinniin 7. Kytkentä on valittu Timer/Counter2-laskurin ominaisuuksien mukaan. 8-bittinen tarkkuus riittää, ja myöhemmin on mahdollista, ilman suuria ohjelma- ja kytkentämuutoksia, käyttää ulkoista kristallia mikro-ohjaimen kellona. Tämä valinta on tehty siltä varalta, että tulevaisuudessa käytettäisiin syystä tai toisesta PWM-ohjattavaa pietsoelementtiä. [2]

4.2 Keskeytykset

Yksi suurimpia vahvuuksia mikro-ohjainten rakenteessa on mahdollisuus käyttää keskeytyksiä. Keskeytykset sallivat toiminnan manipuloinnin pääohjelman ajaessa omaa rutiiniaan. Mikro-ohjaimen tehtävä on usein tarkkailla jotakin ulkopuolista ärsykettä tai tapahtumaa kuten lämpötilaa, äänenvoimakkuutta tai valotehon muutosta. Näitä tapahtumia varten yleensä annetaan ohjelmalle raja-arvo ja ohjeet siitä, mitä tehdään, kun raja-arvo ylitetään, alitetaan tai se täsmää toisen ennalta annetun arvon kanssa. [3]

4.2.1 ADC_ vect-keskeytysvektorin käyttö tässä projektissa

Ruuvisensoria tarkkaileva ohjelma tulkitsee analogimuuntajalle tulevaa signaalia ja sen digitaalista käännöstä. Käännösohjelma mittaa portin A pinneihin 0–5 liitettyjen kiertokytkimien analogiset jännitteet ja muuntaa ne digitaaliseen, kahdeksanbittiseen muotoon.

ADC-keskeytys tulee aktiiviseksi joka kerta, kun A/D-muunnos on valmis, ellei sitä ole ohjelmallisesti ohitettu. Muunnos jatkuu, kunnes A/D-muuntimen toiminta katkaistaan. Muunnosten välisen ajan voi laskea käyttäen avuksi prosessorin kellotaajuutta sekä datalehdeltä löytyvää tietoa muunnosnopeudesta. Datalehden mukaisesti A/D-muunnos kestää 13 kellopulssia, ja käytössä oleva kellotaajuus on prosessorin sisäinen 1 MHz:n kello. Aikaisemmin ohjelmassa määritettiin A/D-esijakajaksi 16. Näillä tiedoilla voidaan laskea muunnosten välinen aika kaavalla: 13clk/62,5 kHz =208 µs. Ensimmäinen käännös vie aikaa 25 kellopulssia, joten samalla kaavalla laskettuna ensimmäinen käännös kestää 400 µs. Ensimmäinen käännös on epätarkka, ja se pitää jättää huomioimatta. Näin ollen ensimmäisen luotettavan

tuloksen saaminen kestää kaavan neljä mukaan laskettuna 13clk+25clk, eli 608 µs. [3]

Kaava 4 Ensimmäiseen A/D-muunnokseen kuluva aika.

$$\frac{13_{clk}}{ADC_{clk}} + \frac{25_{clk}}{ADC_{clk}} = ADC_{muunnos}$$

A/D-muuntajan ollessa free running -tilassa, muutostuloksia tulee jatkuvasti. Jotta kaikki tulokset olisivat varmasti luotettavia, hidastetaan A/D-muuntajan toimintaa keinotekoisesti lisäämällä muutosväleihin viive. Viive hukkuu indikaattoriledien toimintaan, jolloin käyttäjälle näkyvää ylimääräistä viivettä käynnistykseen ei tule. Indikaattoriledejä hidastamalla niiden toiminta saadaan hyvälle tasolle tulkintaa varten, ja samalla varmistetaan muuntotuloksen oikea tallentuminen. [2]

Yleisesti keskeytysvektorin kutsumassa vektorissa ei suositella käytettäväksi viiveitä tai pitkiä ohjelma-ajoja. Tämän projektin kannalta sillä ei kuitenkaan ole merkitystä. Koska keskeytys ajetaan erillisesti ohjelman käynnistysvaiheessa, muistivuoto- ja ajastusongelmia ei tarvitse miettiä. [3]

Jos kiertokytkimellä valittuun linjaan on kytketty ITR8402-sensori, ja se on kiertokytkimellä valittu aktiiviseksi asettamalla tila loogisesti järjestyksen mukaan, vertailuarvo ylitetään ja kasvatetaan muuttujaa ’b’ määrän mukaisesti. Jos esimerkiksi neljän sensorin järjestelmään on valittu kiertokytkimin kaksi ensimmäistä järjestykseen 2, 1, ja kolmas sensori valitaan järjestyksessä viidenneksi, ohitetaan tämä valinta. Ohjelma tunnistaa vain kaksi ensimmäistä sensoria, tai kolme riippuen neljännen kiertokytkimen valinnasta.

Kun kaikki portit on luettu ja saatu looginen määrä loogisesti järjestettyjä sensoreita, lopetetaan A/D-muunnos ja vapautetaan portti A tunnistamaan ruuveja. Oikea määrä sensoreita ilmoitetaan muuttujassa ’b’, jota käytetään myöhemmin ohjelmassa tarkastamaan, onko kaikki tarvittavat ruuvit ruuvattu kiinni.

4.2.2 Ulkoinen keskeytys, INT0

INT0 tai External Interrupt 0 on nimensä mukaisesti ensimmäinen ulkoinen INT-keskeytysvektori. Keskeytysvektorijärjestyksessä se on tärkeysjärjestyksessä toisena, heti RESET-vektorin jälkeen. Tämä tarkoittaa sitä, että INT0-keskeytysvektorilla määritelty toiminta jatkaa ajamistaan niin kauan, kunnes se ajaa syklinsä loppuun, se katkaistaan RESET-vektorilla tai virta kytketään pois. Tarkastelteassa tilannetta toisin päin INT0-keskeytys kutsutaan ajetaan aina, kun sen keskeytysehdot täyttyvät, vaikka ohjelma ajaisi parhaillaan mitä tahansa muuta keskeytystä. INT0-keskeytykseen ei kuitenkaan mennä, ennen kuin kesken oleva keskeytys on ajettu loppuun. [2]

Ulkoisia keskeytysvektoreita hallitaan ulkoisin keinoin asettamalla mikroprosessorin sisääntulopinnin tilaksi looginen 0. Ulkoisille keskeytyksille on datalehden mukaan määritetty sisääntulopinnit mikroprosessorin jaloilla. INT0-keskeytykselle tämä on PD2 (PORTD, PIN2) TQFP/MLF-kannalla jalka #11. Yleisimmin ulkoiseen keskeytykseen kytketään kytkimenä toimiva komponentti. Tämän järjestelmän tapauksessa ON–OFF-painokytkin, joka yhdistää INT0-vektorille määritellyn jalan DC-järjestelmän maahan.

Kun järjestelmä havaitsee virhetilanteen ja ilmoittaa siitä käyttäjälle, on käyttäjällä oltava mahdollisuus tehokkaasti korjata havaittu virhe. Jotta ruuviohjureita voitaisiin käyttää esimerkiksi puuttuvan ruuvin ruuvaamiseksi ilman, että järjestelmä antaa turhaan virhetilanteen, ajetaan järjestelmä huoltotilaan.

Ruuvitunnistinjärjestelmässä INT0-keskeytys hallitsee huoltotoimenpide-tilaa. Huoltoaliohjelma yksinkertaisesti sulkee tarkastelun kaikista muista haarukka-antureista, paitsi avausta tulkitsevasta SENSOR7-anturista. Kun tunnistusjärjestelmä ilmoittaa käyttäjälle vikatilanteesta, käyttäjä havannoi ja

toteaa virheen, sulkee kannen ja kytkee huoltotilan päälle. Huoltotila pysyy ajossa INT0-keskeytyksessä, kunnes kansi avataan. Tämän jälkeen järjestelmä jatkaa normaalia toimintaa käynnistyksessä määritellyllä tavalla. Käyttäjälle onnistuneesta huoltotilan käynnistymisestä ilmoitetaan vihreän ja punaisen LED-valon loistamisella. Diodit suljetaan huoltotilan päätteeksi. Huoltotila ei anna OK- tai FAIL-ilmoituksia.

4.3 Pääohjelma

Pääohjelman alussa määritetään porttien ja porttirekisterien alkutilat ja ajetaan funktiokutsu ”ADC_init”, joka suorittaa edellä mainitulla tavalla sensorien määrän tarkistuksen. Lisäksi määritetään keskeytyskutsujen hyväksyntä ja toimintatapa. Nämä komennot ajetaan vain kerran järjestelmän käynnistyessä. Itse ohjelmaa ajetaan, kunnes käyttöjännite kytketään pois päältä käyttäen while-silmukkaa. While-silmukan toiminta jatkuu, kunnes sille määrätty toistoehto täyttyy. While (eng. kunnes) on nimensä mukaan silmukka, joka toistaa sisältöään, kunnes ehdosta poiketaan. Mikroprosessoriohjelmat asetetaan hyvin yleisesti ajamaan yhtä silmukkaa, joka sitten kutsuu muuta sisältöä, kunnes virta kytketään pois. Tässä ohjelmassa käytetään silmukkaa ”while(1)”. While(x) pätee aina, kun x <0, joten silmukasta ei poistuta. Muita tapoja ajaa jatkuvaa silmukkaa olisi mm. tyhjä for-argumentti ”for(;;)” tai do-while-silmukka ”do – while(1)”.[3]

While-silmukan sisältö ja koko järjestelmän kannalta tärkein toiminta on tarkastella haarukka-anturien toimintaa ja verrata sitä loogiseen toimintajärjestykseen. Ohjelma on kommentein jaettu kahteen osaan.

- Tapahtumat kannen ollessa suljettuna
- Tapahtumat kannen ollessa avattuna

Nämä kaksi ohjelma-aluetta ovat saman funktion kaksi eri puolta. Kannen auki/kiinni-tila voidaan ajatella loogisena operaationa, jossa auki-kiinni-tilamuutosta tarkkailevan haarukka-anturin ulostulon tila muuttuu. Tässä käytetään hyväksi mikro-ohjaimen tapaa tulkita jännitemuutoksia loogisena operaationa (3.3). Ohjelmallisesti tätä muutosta voidaan käsitellä useallakin eri tavalla. Switch-case rakenteella voidaan yhden lisämuuttujan avulla lisätä portin tilamuutos binaarisena ja tarkastella tätä tilamuutosta. Tällä tavalla rakennettuna ohjelma pysyy yhden rungon varassa

Toinen yksinkertaisempi tapa käsitellä portin sisääntulopinnin tilamuutosta on käyttää GCC:n omaa ”bit_is_set/clear” -funktioparia. Funktion toiminta on yksinkertaisesti bit_is_set/clear(PINx, y)), jossa x on portin tunnus ja y pinnin numero. Tämä funktiopari voidaan lisätä ehtolauseen määrittäväksi ehdoksi esim. if(bit_is_set(PINB, 3)) tai while(bit_is_clear(PINC, 7)). Tällä tavalla kirjoitettuna ohjelma pysyy pienempänä, kun lisämuuttujien tarve pienenee.

4.3.1 ”bit_is_clear”, kansi suljettuna.

Kannen ollessa suljettuna, ohjelma tulkitsee portin ‘A’ pinniin 7 kytketyn haarukka-anturin loogisen tilan 0. Tälloin haarukka-anturin valojuovan välissä ei ole mitään ja output = 0 V. Tässä tilassa ohjelma tutkii kiertokytkimin määrättyjen haarukka-anturien tilamuutoksia ehtolauseilla if(bit_is_set(PINA, y)), jossa y on pinnin numero.

Otetaan esimerkiksi kiertokytkimin valittu sensori numero neljä. Ohjelma tutkii sensorin loogista tilamuutosta käyttämällä ehtolausetta if(bit_is_set(PINA, 3)). Kun ruuvi tai ruuvikärki käy katkaisemassa sensorin valojuovan, muuttuu sen looginen tila. Tällöin ohjelma asettaa muuttujan ”sensor3” arvoksi yksi sekä kulkutarkastuksessa käytettävän muuttujan ”kulku” arvoksi yksi.

Seuraavaksi tarkastetaan, onko järjestyksessä edellisen sensorin valojuova katkaistu tarkastamalla muuttujan ”sensor2” arvo. Arvon ollessa 0, asetetaan virhetarkastelumuuttuja ”huti” arvoon yksi.

```
if(bit_is_set(PINA, 3))                    //kun Sensor3 leikattu
{
sensor3=1;
kulku=1;

                if(sensor2==0)
                {
                huti=1;
                }
}
```

Väärästä koontajärjestyksestä ilmoitetaan käyttäjälle välittömästi virheen tapahduttua sytyttämällä punainen virheilmaisuledi koontajigin kannessa. Virhetarkastelumuuttujan ”huti” tilamuutosta 0 → 1 tarkkaillaan jatkuvasti ehtolauseella if(huti == 1).

Ohjelma laskee sensor-muuttujien arvoja jatkuvasti yhteen, kunnes kansi avataan. Tämä laskutoimitus tallennetaan tarkistusmuuttujaan ”lisatarkistus” yksinkertaisella sijoituslausekkeella ”lisatarkistus = (sensor0+sensor1+...+sensor5)”. Lisatarkistus-muuttujan arvoa verrataan käynnistyksessä määritellyn muuttujan ”b” arvoon, joka sisältää tiedon käytössä olevien ruuvitunnistimien määrästä. Jos molemmat muuttujat täsmäävät, sytytetään koontajigin kannessa oleva vihreä ok-ledi merkiksi käyttäjälle oikeasta koontajärjestyksestä. Samalla asetetaan tarkastusmuuttuja ”ok” arvoon yksi myöhempää käyttöä varten. Jos tämän lisäksi on aikaisemmin sytytetty

punainen virheilmaisuledi, jätetään se myös palamaan. Näin ollen väärässä koontajärjestyksessä koottu puhelin ei aiheuta hälytystä, jos jokainen käynnistyksessä määritelty ruuvi on kuitenkin ruuvattu.

4.3.2 ”bit_is_set”, kansi auki

Kannen ollessa auki, portin ’A’ pinniin 7 kytketty haarukka-anturi lukee loogista tilaa 1, eli haarukka-anturin valojuovan katkaisee sitä varten suunniteltu musta muovinen osa, haarukka-anturin Uout = Vcc. Avaustarkistus pyörii, kunnes kansi jälleen suljetaan käyttäen while-ehtolausetta: ”while(bit_is_set(PINA,7))”.

Avausohjelman aluksi suodatetaan tilanne, jossa kansi on suljettu, mutta avattu tekemättä mitään. Tämä tehdään tarkistamalla kulkutarkastusmuuttujan ”kulku” arvo. Jos muuttujan arvo on yksi, aloitetaan virhe- tai ok-ilmaisu. Muussa tapauksessa ohjelma palaa odottamaan kannen sulkemista.

Seuraavaksi ohjelma tarkastaa lisämuuttujan ”odotus” arvon. Muuttujan arvo on alustettu käynnistyksessä arvoon 0. Jos arvo vastaa alkutilannetta (odotus == 0), suoritetaan kulkutarkastusmuuttujan tarkistus. Kulkutarkastusmuuttujan arvon ollessa yksi, tarkastetaan muuttujan ”ok” arvo ehtolauseella if(ok==1). Jos arvo täsmää, asetetaan vihreän ledin ulostulojalka uudelleen arvoon 1. Tämä on lähinnä varotoimenpide, jolla varmistetaan, että vihreä led todella on päällä. Välttämättä tätä ei tarvitsisi tehdä. Seuraavaksi ohjelma odottaa 200 ms, jonka jälkeen asetetaan odotus-muuttujan arvoksi yksi, jolloin vältetään saman silmukan ajaminen uudelleen. Lopuksi asetetaan vihreän ledin ulostulojalka arvoon 0, jolloin led sammuu.

Jos muuttujan ”ok” arvo on nolla, siirrytään edellisen kaltaiseen silmukkaan if(odotus==0). Tällä kertaa kuitenkin sytytetään punainen led, ja soitetaan hälytysääni piezosummerilla. Pietsosummeri sisältää itsessään ohjauselektroniikan, joten sitä voidaan hallita kytkemällä virta ulostulojalkaan. Tätä virtaa katkotaan 11 kertaa, pitäen virtaa ylhäällä 300ms ja alatilassa

100ms, jolloin saadaan aikaan ”piipittävä” hälytysääni. Tämän silmukan jälkeen kaikki ledit ajetaan alatilaan asettamalla portin C ulostuloksi 0 komennolla ”PORTC=0x00;”, eli annetaan portille C heksa-arvo 00.

Kun kansi jälleen suljetaan, ohjelma nollaa kaikki dynaamiset apumuuttujat ja siirtyy takaisin ajamaan sensoritarkastusta.

5 Omat havainnot

Tämän projektin ja samalla tämän opinnäytteen tavoitteena on ollut tuottaa tuotannon käyttöön apuväline, joka toimiessaan nopeuttaa virhetarkastelua ja -korjausta. Tällaisen projektin koulutusohjelman sisältöä vastaavan projektin tekeminen on ollut mielenkiintoista ja osaltaan syventänyt opiskeltua sisältöä käytännön esimerkillä.

Projektin alussa määritettyjen vaatimusten täyttäminen on osittain ollut mahdollista, osittain niistä on luovuttu. Koska määritykset muuttuivat projektin aikana, aiheutti se konflikteja elektroniikan ja ohjelmiston kannalta. Tämän projektin päätoimintamääritykset muuttuivat matkan varrella siinä määrin, että ohjelmisto kirjoitettiin alusta alkaen uudestaan ennen viimeistä julkaisua. Jälkeenpäin tarkasteltaessa olisi ollut viisasta kirjoittaa ohjelma heti modulaarisesti, jolloin muutosten tekeminen olisi ollut helpompaa. Projektityöskentely onkin yleensä dynaamista, jolloin projektissa pitäisi jo suunnitteluvaiheessa ottaa huomioon mahdollisuus muutoksiin ja se, miten muutosten integrointi olemassa olevaan olisi vaivattominta.

Sulautettujen järjestelmien käyttö tuotannon apuvälineinä on lisääntymässä. Ohjelmoitavan piirin vahvuus esimerkiksi viivepiireillä ja komparaattoreilla tehtyyn samankaltaiseen järjestelmään, on sen helppo muunneltavuus. Tuotantokustannusten pitämiseksi matalana on kannattavaa saada aikaiseiksi järjestelmä, joka on yhteensopiva usean eri laitteen kanssa, ja helposti ja nopeasti muunneltavissa, kun vaatimukset tai käyttötarkoitus muuttuvat.

LÄHTEET

[1] Niemi, Joni. Keskustelut 19.1.2009-5.3.2010.

[2] ATMEGA32 datasheet. [pdf-dokumentti]. Saatavissa:

http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf.

[3] Vahtera Pentti 2003. Mikro-ohjaimen ohjelmointi C-kielellä. Helsinki: WSOY.

[4] Jokinen, Juha. Keskustelut 8.3.2009-7.4.2010.

[5] ITR 8402 datasheet [pdf-dokumentti].

[6] Delton T.Horn, Electronic Components, A Complete Reference For Project Builders, Tab Books 1992. E-book saatavissa:
http://books.google.com/books?id=2BhC92ICFCYC&lpg=PP1&dq=electronic%20components&pg=PA10#v=onepage&q&f=false.

[7] Vilkka, Hanna & Airaksinen, Tiina 2003. Toiminnallinen opinnäytetyö. Jyväskylä: Tammi.

LIITTEET

```
/*****************************************************************************
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
******************************************************************************
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
Project : Portio.c
Date    : 10.02.2010
Hardware: SCSECO + STSCR ATMega32
Author  : VHa
Comments: Ruuvitunnistimen ohjaussofta. Järjestys ja määrä valittavissa
                                    kiertokytkimin. Max 6kpl. Virhetilanteessa piezosummerilla
                                    äänimerkki. Huoltokytkin @ INT0, jolloin sensorit pois päältä.
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
******************************************************************************
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
******************************************************************************/
/*F_CPU määritykset*/
#ifndef F_CPU
#define F_CPU 1000000UL // 1 MHz
#endif
/*Headerit*/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
/*Määritykset*/
#define WAIT(time) for(uint16_t i=0;i<2000;i++)_delay_loop_2(time);
#define aika 5

/*Muuttujien esittely*/
 uint8_t i=0;
 uint8_t a=0;
 uint8_t b=0;
 volatile uint8_t tarkastus=0;
 volatile uint8_t laskuri=0;
 uint8_t kulku=0;
 volatile uint8_t looppi=1;
 uint8_t avaus=0;
 volatile uint8_t f=0;
 volatile uint8_t lopetus=0;
 volatile uint8_t huoltaminen=0;
 uint8_t sensor0=0;
 uint8_t sensor1=0;
 uint8_t sensor2=0;
 uint8_t sensor3=0;
 uint8_t sensor4=0;
 uint8_t sensor5=0;
 uint8_t huti=0;
 uint8_t lisatarkistus = 0;
 uint8_t ok=0;
 uint8_t odotus=0;
```

```
/*Piezoaliohjelma*/
void Summeri(void)
{
 PORTD=0xFF;
 WAIT(300);
 PORTD=0x00;
 WAIT(100);
}



void Huolto(void)
{
looppi=1;
                do
                {
                                lopetus = PINA;
                                switch(lopetus)
                                {
                                case 0b10000000:

                                laskuri         = 0;
                                f=0;
                                lopetus=0;
                                PORTC = 0x00;
                                looppi=0;
                                huoltaminen=1;
                                avaus=0;
                                break;
                                default:
                                PORTC = 0b00000011;
                //              WAIT(aika);
                                }
                }
                while(looppi == 1);
}

/*Huoltokeskeytys*/
ISR(INT0_vect)
{

                Huolto();

}

/*ADC alustus*/
void ADC_init(void)
{               ADCSRA |= (1 << ADPS2) | (1 << ADPS1); //kaikki ADC-bitit asetettava, jotta
saadaan prescaler 128

                ADMUX |= (1 << REFS0);  //AD-referenssijänniteeksi asetetaan AVCC

                ADCSRA |= (1 << ADATE); //free-running mode päälle
```

```
                ADMUX |= (1 << ADLAR); //muutetaan 10-bitin AD-muunnos 8-bittiseksi


                ADCSRA |= (1 << ADEN); //ADC päälle
                ADCSRA |= (1 << ADIE); //ADC keskeytykset päälle

                ADCSRA |= (1 << ADSC); //mittauksen aloitus
}

ISR(ADC_vect) //ADC keskeytys
{



/*ADC lukuohjelma*/
if(ADCH > 30)
                 {

                laskuri++; //laskurin kasvatus



                if(laskuri == 1 )
                {
                //ADMUX &= ~((1 << MUX0) | (1 << MUX1) | (1 << MUX2)); //MUX 0000 ADC0
                ADMUX |= (1 << MUX0); //MUX  0001 ADC1
                PORTC = 0x01;
                WAIT(10);
                PORTC = 0x00;

                WAIT(50); //viive jottei free-running sotke laskuria

                b=1;



                }
                if(laskuri == 2)
                {

                ADMUX &= ~(1 << MUX0); //MUX  0000
                //ADMUX |= (1 << MUX0); //MUX  0001 ADC1
                ADMUX |= (1 << MUX1);  //MUX  0010 ADC2
                PORTC = 0x01;
                WAIT(10);
                PORTC= 0x00;
                WAIT(10);
                PORTC = 0x01;
                WAIT(10);
                PORTC= 0x00;
                WAIT(50);
                b=2;
```

```
}

if(laskuri == 3)
{

ADMUX |= (1 << MUX0) | (1 << MUX1); //MUX  0011 ADC3

PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(50);
b=3;

}

if(laskuri == 4)
{

ADMUX &= ~((1 << MUX0) | (1 << MUX1)); //MUX  0000
ADMUX |= (1 << MUX2); //MUX  0100 ADC4
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(50);
b=4;
}

if(laskuri == 5)
{

ADMUX |= (1 << MUX0) | (1 << MUX2); //MUX  0101 ADC5
WAIT(10);
PORTC = 0x01;
WAIT(10);
PORTC= 0x00;
WAIT(10);
PORTC = 0x01;
WAIT(10);
```

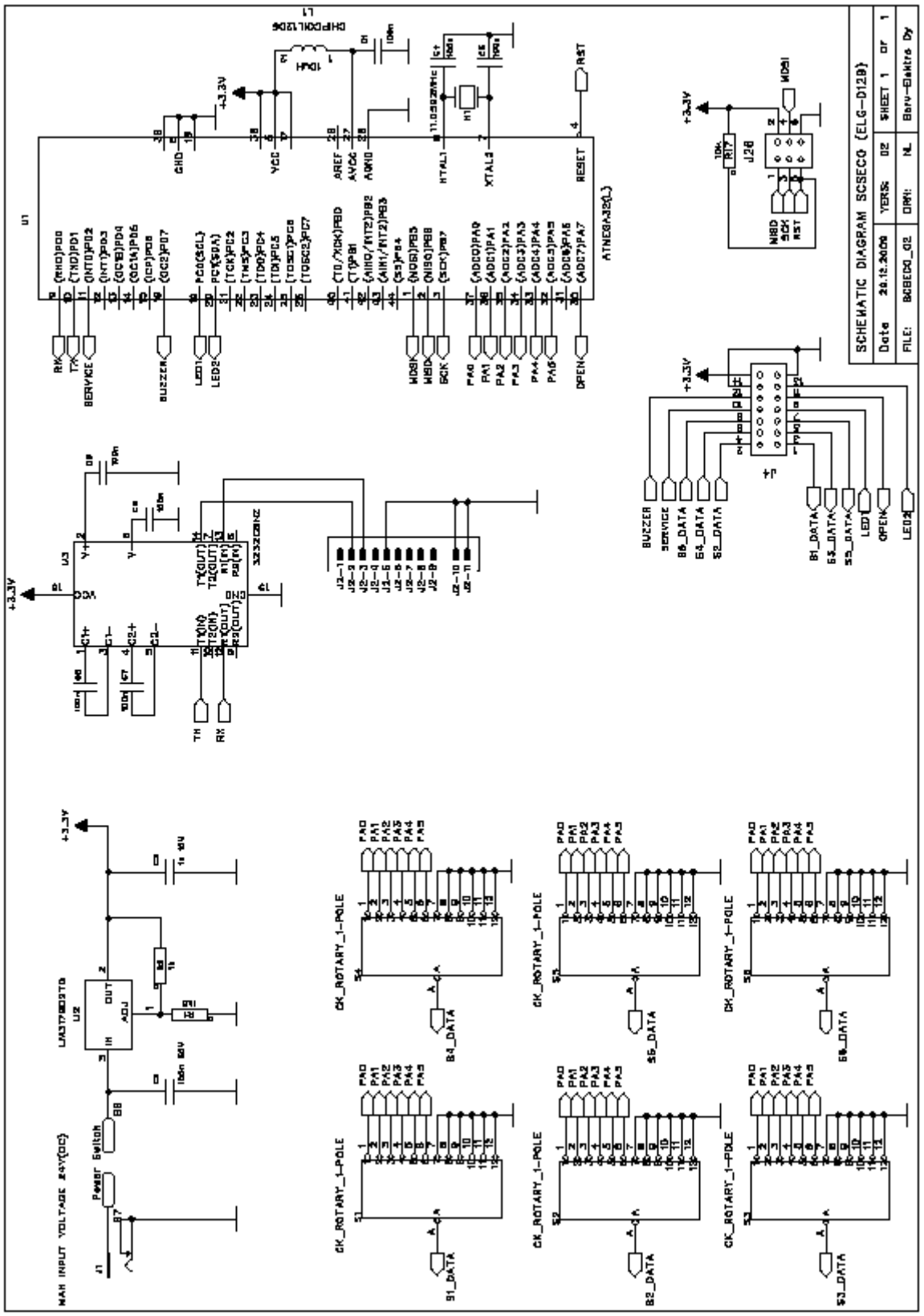
```
                PORTC= 0x00;
                WAIT(10);
                PORTC = 0x01;
                WAIT(10);
                PORTC= 0x00;
                WAIT(10);
                PORTC = 0x01;
                WAIT(10);
                PORTC= 0x00;
                WAIT(10);
                PORTC = 0x01;
                WAIT(10);
                PORTC= 0x00;
                WAIT(50);
                b=5;
                }

                if(laskuri == 6)
                {
/*
                ADMUX &= ~(1 << MUX0); //MUX 0100
                ADMUX |= (1 << MUX1); //MUX 0110*/
                WAIT(50);
                b=6;
                }



                }
                else
                {

                PORTC = 0xFF;
                WAIT(10);
                PORTC = 0x00;
                WAIT(20);
                tarkastus = 1;
                ADCSRA &=  ~(1 << ADEN); //ADC pois

                }
}



/*Pääohjelma*/
int main (void)
{


DDRA = 0x00;                    //Portin A rekisteri input      0000 0000
DDRC = 0xFF;
                                //Portin C rekisteri output     1111 1111
DDRD = 0b10000000;
PORTD = 0b00000100;             //Portin D rekisteri 1<<7       1000 0000     Piezo
PD7
```

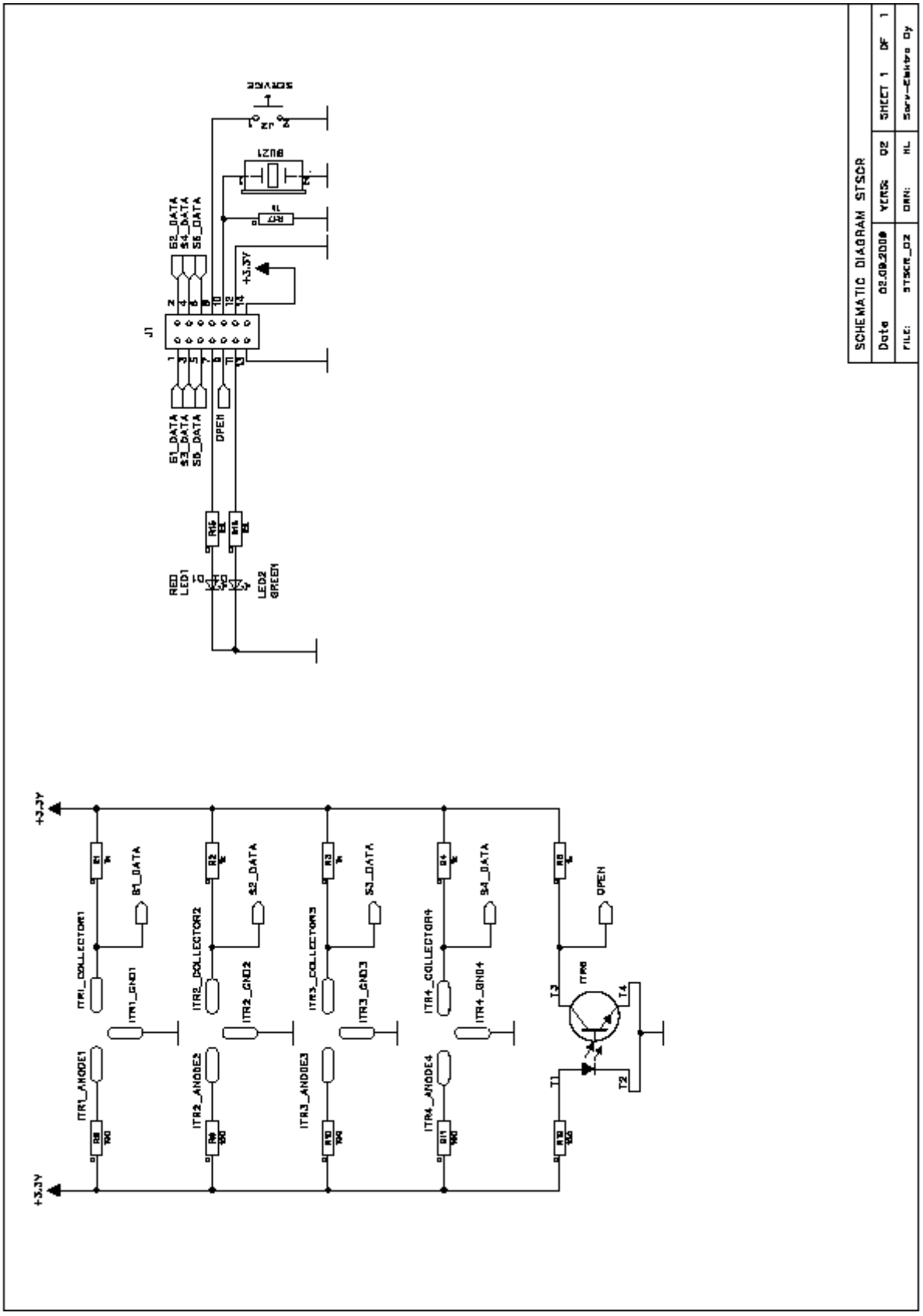
```
sei();                                  // Keskeytykset sallittu
ADC_init();                             // ADC alustusohjelmakutsu

MCUCR = (1<<ISC01) | (1<<ISC00);
GIFR = (1<<INTF0);                      // laskeva reuna generoi keskeytyksen
GICR |= 1 << INT0;                      // INT0 sallittu

while(1)                                // Pyörii kunnes virta kytketään pois
{


//////////////////////kansi suljettuna/////////////////////////////////
                while(bit_is_clear(PINA, 7))                                    //kansi
suljettu
                {




                                if(bit_is_set(PINA, 0))
                                //kun Sensor0 leikattu
                                {

                                                sensor0=1;
                                                kulku=1;        //marker


                                }

                                if(bit_is_set(PINA, 1))
                                //kun Sensor1 leikattu
                                {

                                                sensor1=1;
                                                kulku=1;        //marker

                                                if(sensor0 == 0)
                                                {
                                                huti=1;
                                                }

                                }

                                if(bit_is_set(PINA, 2))
                                //kun Sensor2 leikattu
                                {

                                                sensor2=1;
                                                kulku=1;        //marker

                                                if(sensor1 == 0)
                                                {
                                                huti=1;
                                                }

                                }
```

```
if(bit_is_set(PINA, 3))
//kun Sensor3 leikattu
{

        sensor3=1;
        kulku=1;        //marker

        if(sensor2==0)
        {
        huti=1;
        }


}

if(bit_is_set(PINA, 4))
//kun Sensor4 leikattu
{

        sensor4=1;
        kulku=1;        //marker

        if(sensor3==0)
        {
        huti=1;
        }

}

if(bit_is_set(PINA, 5))
//kun Sensor5 leikattu
{

        sensor5=1;
        kulku=1;        //marker

        if(sensor4==0)
        {
        huti=1;
        }

}

lisatarkistus = (sensor0+sensor1+sensor2+sensor3+sensor4+sensor5);


if(huti==1)                     //väärä järjestys
        {

        PORTC = 0x01;           //PUNANEN

        }

if(lisatarkistus == b)
{
        PORTC = 0x02;
        ok=1;                   //vihreä
```

```
                                        if(huti==1)
                                        {
                                                PORTC = 0x03;
                                                //vihreä & punainen
                                        }
                                }



        }
////////////////////////////Kansi auki/////////////////////////////////////

        while(bit_is_set(PINA, 7))
        {

                if(kulku==1)
                //jos kansi avattu ja jotain tehty
                        {

                        if(ok==1)
                        {
                        if(odotus == 0)
                                {
                                PORTC = 0x02;
                                //vihreä
                                WAIT(200);
                                odotus=1;
                                PORTC = 0x00;
                                        //onetime loop
                                        }
                                }

                        else
                        {
                        if(odotus == 0)
                        {
                        PORTC = 0x01;
                        for(uint8_t x=0; x<10;x++)
                        {
                        Summeri();
                        }
                        PORTC = 0x00;
                        odotus=1;
                        }

                        }




                        }

                        else
                        {

                        //nogo-tila
```

```
                                        }

                }

                odotus=0;
                huti=0;
                kulku=0;
                sensor0=0;
                sensor1=0;
                sensor2=0;
                sensor3=0;
                sensor4=0;
                sensor5=0;
                ok=0;
                lisatarkistus=0;


}



return 0;
}
```


SCHEMATIC DIAGRAM SCSECO (ELG-D128)
Date 28.12.2009
YERS: 02
SHEET 1 OF 1
FILE: SCSECO_02
DRN: NL
Baru-Elektro Oy
ATMEGA32(L)
LM3724BSTB
MAIN INPUT VOLTAGE 24V(DC)
Power Switch
BUZZER
SERVICE
B6_DATA
B4_DATA
B2_DATA
B1_DATA
B3_DATA
B5_DATA
LED1
LED2
OPEN
RST
MOSI
MISO
SCK
CK_ROTARY_1-POLE


SCHEMATIC DIAGRAM STSCR
Date 02.09.2008
FILE: STSCR_02
VERS: 02
DRN: HL
SHEET 1 OF 1
Sarv-Elektro Oy
+3.3V
B1_DATA
B2_DATA
B3_DATA
B4_DATA
OPEN
ITR1_COLLECTOR1
ITR2_COLLECTOR2
ITR3_COLLECTOR3
ITR4_COLLECTOR4
ITR1_GND1
ITR2_GND2
ITR3_GND3
ITR4_GND4
ITR1_ANODE1
ITR2_ANODE2
ITR3_ANODE3
ITR4_ANODE4
ITR5
T1
T2
T3
T4
J1
B1_DATA
B3_DATA
B5_DATA
B2_DATA
B4_DATA
B6_DATA
OPEN
RED LED1
LED2 GREEN
BUZ1
SERVICE