Pasi Vuorinen

MIKROPROSESSORIOHJATTU 3D-RISTINOLLA

Tietotekniikan koulutusohjelma
2016

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

MIKROPROSESSORIOHJATTU 3D-RISTINOLLA

Vuorinen, Pasi
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Kesäkuu 2016
Ohjaaja: Ekholm, Ari
Sivumäärä: 72
Liitteitä: Ratkaisualgoritmit, kytkentäkaaviot.

Asiasanat: sulautetut järjestelmät, elektroniikka, ohjelmointi, digitaalitekniikka, mikroprosessori

---

Opinnäytetyössä käsiteltiin mikroprosessoriohjattua 3D-ristinolla-peliä. Peli suunniteltiin ja toteutettiin alusta alkaen itse sekä laitteiston että ohjelmiston osalta. Työ saatiin toimeksiantona Satakunnan ammattikorkeakoululta ja se tehtiin parityönä. Työ jaettiin kahteen osaan: isäntälaitteeseen ja orjalaitteeseen. Tämä työ käsittelee isäntälaitetta ja sen toinen osa liittyen orjalaitteisiin on Janne Äijälän kirjoittama samanniminen opinnäytetyö vuodelta 2008.

Työn teoriaosuudessa käsiteltiin laitteen rakennetta ja toimintaperiaatetta sekä käytetyn mikroprosessorin teknologioita, joita olivat väylät, keskeytykset, siihen liitettävä LCD-näyttö, piirin ohjelmointiliitännät ja muistit. Työssä käytetty mikroprosessori oli RISC periaatteeseen perustuva Atmelin valmistama AVR sarjan kahdeksan bittinen ATmega32 -ohjain.

Työn käytännön osuudessa käsiteltiin laitteen rakentamista ja suunnittelua. Laitteen suunnittelussa käytiin läpi käytettyjä ohjelmia kuten CadSoft Eagle piirilevyn suunnitteluohjelmaa ja Bungard piirilevyjyrsimen ohjaukseen käytettyä ohjelmaa. Samalla käytiin läpi myös laitteiston kasaamista ja tähän vaiheeseen liittyneitä haasteita.

Viimeisenä työssä käytiin läpi mikroprosessorin ohjelmointia C-ohjelmointikielellä. Käytiin läpi pelin yleistä rakennetta, oheislaitteiden käsittelyä, tilakone-ideologiaa, LCD-näytön valikkorakenteen toteutusta, laitteiden välistä kommunikointia ja lopuksi ristinolla-pelin ratkaisualgoritmeja.

MICROPROCESSOR CONTROLLED 3D TIC-TAC-TOE

Vuorinen, Pasi
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
June 2016
Supervisor: Ekholm, Ari
Number of pages: 72
Appendices: Solution algorithms, circuit diagrams.

Keywords: embedded systems, electronics, programming, digital technology, microprocessors

---

The purpose of this thesis was to design, build and program a microprocessor controlled game of 3D-tic-tac-toe. The game was designed and implemented from the very beginning, by the authors Janne Äijälä and Pasi Vuorinen. The work was divided into two parts: the master device and the slave devices. This thesis covers the former, the master device. Janne Äijälä's thesis from 2008 covers the slave devices and is named Microprocessor controlled 3D tictactoe. The thesis was commissioned by Satakunta University of Applied Sciences.

The theory section of the thesis covered the structure and functionality of the work and technologies used in the microprocessor. This included buses, interrupts, peripheral LCD-displays, programming headers and memories. The microprocessor used was based on RISC strategy and manufactured by Atmel, an eight bit ATmega32 controller.

The practical part of the thesis covered the planning and building of the device. It included the used programs like CadSoft Eagle circuit designer and Bungard CCD router software.

The last section covered the programming of the microprocessors in C-language. The general structure, peripheral handling, state machine ideology, LCD-display's menu structure, device to device communication and 3D-solving algorithms were included in the last section.

SISÄLLYS

1 LYHENNELUETTELO 5
2 JOHDANTO 7
3 YLEISTÄ TIETOA 8
3.1 Kuvaus laitteesta 9
3.1.1 Isäntälaite 10
3.1.2 Orjalaite 11
3.2 AVR ja oheislaitteet 12
3.2.1 I²C 13
3.2.2 Keskeytykset 17
3.2.3 HD44780 18
3.2.4 SPI & In-System Programming 20
3.2.5 EEPROM 23
4 LAITTEISTO 26
4.1 Suunnittelu 26
4.1.1 CadSoft Eagle 27
4.1.2 HID-laite 29
4.1.3 ISP 30
4.1.4 Virtalähde 31
4.1.5 I²C 32
4.1.6 Oheiskomponentit 33
4.2 Bungard CCD 34
4.3 Laitteiston kasaaminen 36
5 OHJELMISTO 38
5.1 Ohjelman kulku ja rakenne 38
5.2 Laitteen tilat 40
5.2.1 HID-laite 42
5.2.2 Tilakone 45
5.2.3 Funktio-osoitin 46
5.2.4 LCD puskuri 48
5.2.5 Tasojen tarkastelu 51
5.2.6 Valikon muut valinnat 53
5.3 I²C kommunikointi 61
5.4 3D algoritmi 66
6 JOHTOPÄÄTÖKSET 71
LÄHTEET 73
LIITTEET

1 LYHENNELUETTELO

AVR	Atmelin valmistama mikro-ohjainperhe. Muodostuu suurimmilta osin kahdeksan bittisistä RISC mikro-ohjaimista.
RISC	Reduced Instruction Set Computer, pienen ja yksinkertaisen käskykannan omaava mikro-ohjain. Vertaa esimerkiksi CISC, x86 arkkitehtuuriin.
LCD	Liquid Crystal Display, nestekidenäyttö.
LED	Light Emitting Diode, valodiodi tai ledi.
I/O	Input/Output. Rajapinta, jolla voi ohjata jotain laitetta, esimerkiksi painonapit (input) ja näyttö (output).
UART	Universal Asynchronous Receiver Transmitter. Yleinen oheislaitteiden tiedonsiirrossa käytetty piiri. Tunnetaan arkikielessä nimellä ”Sarjaportti/-väylä”.
TTL	Transistor-Transistor Logic. Käytännössä kuvaa piiriperhettä, jonka logiikan taso vastaa viiden voltin positiivista jännitettä.
I²C (tai TWI)	Inter-Integrated Circuit (2-wire interface) Kahden johtimen sarjaliitäntä-väylä.
SDA	Serial Data Line, I²C-väylän data-johdin.
SCL	Serial Clock Line, I²C-väylän tahdistus-johdin.
SPI	Serial Peripheral Interface, synkroninen sarjaliikenneväylä.
MOSI	Master Out, Slave In. SPI protokollassa lähettävä väylä.
MISO	Master In, Slave Out. SPI protokollassa vastaanottava väylä.
SCK	SPI Clock line. SPI protokollassa tahdistusväylä.
ISP	In-System Programming. Atmel AVR-laitteiden ohjelmointirajapinta.
EEPROM	Puolijohdemuisti, joka voidaan sähköisesti tyhjentää ja uudelleen ohjelmoida.
Flash	Puolijohdemuisti, joka voidaan sähköisesti tyhjentää ja uudelleen ohjelmoida.
HEX	Heksadesimaalijärjestelmä. 16 kantaluvun lukujärjestelmä.

ASCII	American Standard Code for Information Interchange. Seitsemän bittinen, tietokoneen merkistö.
IRQ	Interrupt request. Keskeytyspyyntö joka voidaan luoda jos tarvitaan välitöntä palvelua suorittimelta, esim. napin painallus.
INT1	Interrupt 1. Katso IRQ.
PS/2	PS/2 liitäntä (IBM Personal System/2). Vanhoissa tietokoneissa ja laitteissa yleisesti käytössä ollut DIN-liitin oheislaitteille. Usein hiirelle ja näppäimistölle.
USB	Universal Serial Bus. Nykyisin tietokoneissa hyvin yleisessä käytössä oleva oheislaiteliitäntä.
ALU	Arithmetic logic unit. Mikro-ohjaimen yksikkö jolla suoritetaan kokonaislukujen aritmeettiset ja bitti –operaatiot.
HID	Human Interface Device. Laite, jolla ihminen voi olla vuorovaikutuksessa jonkin laitteen kanssa.
LCD	Liquid-Crystal Display. Nykyään yleisin käytössä oleva näyttötyyppi jossa yksittäiset pikselit ovat nimensä mukaisesti nestemäisiä kiteitä.
HID	Human Interface Device. Laite jolla henkilö voi olla vuorovaikutuksessa koneen kanssa. Kuten esimerkiksi näppäimistö tai näyttö.
EXCELLON	Yleinen CNC-jyrsimien ohjaukseen käytetty formaatti.
CAM	Computer Aided Manufacturing. Arkistotiedosto HPGL ja EXCELLON formaateille.
HPGL	Hewlett-Packard Graphics Language. Tulostimien ohjauskieli.
PWM	Pulse-Width Modulation. Tekniikka jolla voidaan säätää tietyn jännitteen tehollisarvoa. Esimerkiksi sähkömoottorin nopeuden säätämiseen.
ADC	Analog to Digital Conversion. Muunnin joka pystyy muuntamaan analogisen signaalin digitaaliseksi.

2 JOHDANTO

Tässä opinnäytetyössä käydään läpi kolmiulotteisen ristinolla-pelilaitteen suunnittelua, toteutusta ja rakentamista. Työ on jaettu kahteen osaan, joista ensimmäisen kirjoitti Janne Äijälä, 2008, samannimisessä opinnäytetyössään.

Toimeksianto työlle saatiin Satakunnan Ammattikorkeakoululta ja alusta alkaen huomattiin tehtävän olevan sen verran suuri, että sen voisi toteuttaa parityönä. Joten työ päätettiin jakaa kahteen osaan: pelitason ohjaimiin ja pääohjaimeen, joista tämä työ käsittelee jälkimmäistä eli isäntälaitetta.

Työssä käsitellään ensin laitteessa käytettyjen teknologioiden toiminnallisuutta teoriatasolla. Sitten käsitellään laitteen suunnittelua ja rakentamista käytännössä ja käydään samalla läpi mahdollisia vastaan tulleita ongelmia. Lopuksi käydään läpi tämän työn tärkeimpänä osuutena, laitteen ohjelmistoa ja sen rakennetta sekä toteutusta.

Työssä on tarkoitus oppia digitaalielektroniikan käytännön sovelluksia ja sen suunnittelua sekä mikro-ohjaimien ohjelmointia. Työn tekijöiden opiskeluaikainen suuntautuminen sulautettuihin järjestelmiin ja elektroniikkaan vaikuttivat osaltaan hyvin paljon toimeksiannon vastaanottamiseen. Tämän vuoksi työltä toivottiin riittävää haasteellisuutta oman osaamisen kehittämiseksi.

3 YLEISTÄ TIETOA

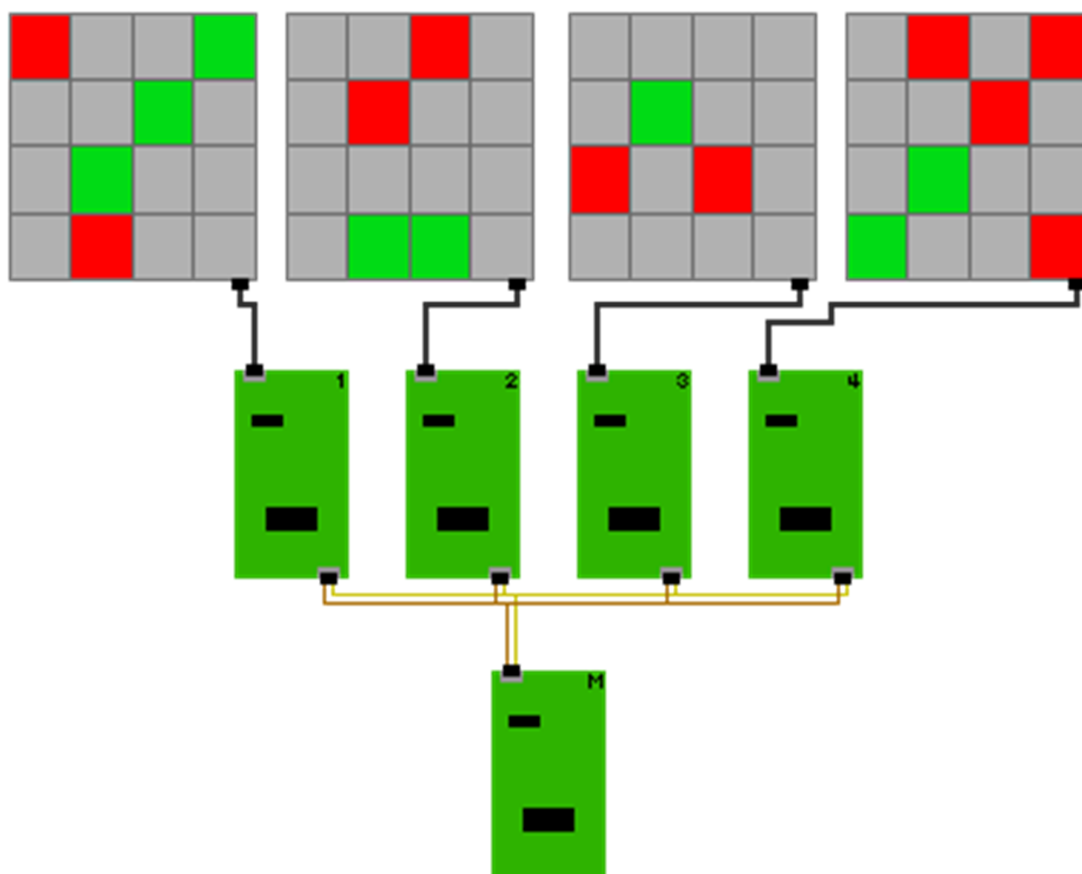
Työn aiheena oli rakentaa ja ohjelmoida laite, jolla voidaan pelata kolmiulotteista ristinollaa. Mitään ennakkosuunnitelmia työstä ei annettu, vaan sen suunnittelu ja toteutus hoidettiin itse alusta loppuun.

Ristinolla on kaikille tuttu peli, jossa yritetään saada yleensä kolme ristiä tai nollaa peräkkäin joko diagonaalisesti, vaaka- tai pystysuoraan. Pelialustana käytetään usein yhdeksää ruudukkoa, jotka ovat jaettu kaksiulotteiseen, kolme kertaa kolmen, matriisiin. Itse työn aiheena oli suunnitella ristinolla, joka jaettaisiin neljä kertaa neljän ryhmiin, mutta kolmeen ulottuvuuteen, eli pelitasoja on neljä. Tämä tuo diagonaalisen, pysty- ja vaakasuoran voittolinjan lisäksi tasojen väliset voittorivit.

Työ päätettiin jo aikaisessa vaiheessa kirjoittaa kahdessa osassa ja ensimmäinen vaihe oli miettiä, miten työ jaettaisiin, jotta siitä saataisiin kaksi osa-aluetta kummallekin tekijälle. Työ jaettiin orja- ja isäntälaitteeseen, koska huomattiin, että isäntälaitteeseen tulisi vähemmän laitteistosuunnittelua verrattuna orjalaitteeseen ja päinvastoin orjalaitteeseen tulisi vähemmän ohjelmistosuunnittelua verrattuna isäntälaitteeseen. Näin ollen työmäärä jakautuisi tasaisesti kahteen työhön. Tässä työssä kerrotaan isäntälaitteesta eli pelilogiikan ohjaimesta. Orjalaitteista kertoo Janne Äijälä samannimisessä opinnäytetyössään.

Aluksi määriteltiin yksinkertainen kaavio siitä, miten laitteen tulisi toimia, miltä sen tulisi näyttää ja miten minimoida kustannukset, mutta samalla tehdä laitteesta helposti käytettävän. Tähän sisältyi suunnitelma siitä, miten käyttäjä on vuorovaikutteessa laitteen kanssa ja miten laite indikoi käyttäjälle visuaalisesti pelivuoron, painallukset ja voittorivin, miten ohjata pelitasoja ja miten hallita pelin etenemisen logiikkaa.

Laitetta päätettiin ohjata kosketuskalvoilla, jotka jaettiin kuuteentoista ruudukkoon ja joiden alapuolelle sijoitettiin kaksiväriset ledit indikoimaan pelivuoroa. Kosketuskalvot olivat läpinäkyviä, jotta alempi pelitaso olisi helpommin näkyvillä. Kalvoja ja ledejä ohjaa orjalaite, joka on yhteydessä isäntälaitteeseen, jolla ohjataan pelin kulkua ja lasketaan voittorivi.



Kuvio 1. Haivannekuva laitteen rakenteesta.

3.1 Kuvaus laitteesta

Edellisessä kappaleessa käytiin läpi laitteen yleinen kuvaus. Tässä kappaleessa tutustutaan tarkemmin laitteen eri osien toimintaan. Kuten jo aiemmin todettiin, ohjataan laitetta neljällä kosketuskalvolla. Kosketuskalvoiksi valittiin resistiivisellä tekniikalla toimivat kalvot, koska työtä suunniteltaessa resistiiviset kalvot olivat edullisempia ja saatavuus oli parempi. Nykypäivänä tämä voitaisiin ennemmin toteuttaa kapasitiivisilla kosketuskalvoilla, koska matkapuhelintekniikka on kehittänyt kapasitiivista teknologiaa ja tuonut niiden hinnat alas.

Kosketuskalvot jaettiin kuuteentoista osaan, joiden alle sijoitettiin kaksiväriset ledit indikoimaan kumpi pelaaja, punainen tai vihreä, on painanut ruutua. Kosketuskalvojen mukana toimitettiin omat ohjaimet, jotka liitettiin orjalaitteen sarjaporttiin. Sama orjalaite ohjaa myös ledejä ja muistaa niiden tilat.

Kun pelaaja painaa kosketuskalvon ruutua, varmistaa orjalaite, onko ruutu tyhjä ja asettaa siihen liitetyn ledin päälle sen mukaan kumman pelaajan vuoro on kyseessä. Tämä tilatieto jää orjalaitteen puskuriin muistiin ja isäntälaite kyselee orjalaitteilta tiedot painetuista ruuduista. Näitä orjalaitteita on yhteensä neljä kappaletta. Laitteet ovat keskenään identtisiä ja ainoastaan I²C-väylälaitteen osoite muutetaan, jonka perusteella laite voidaan yksilöidä.

Isäntä- ja orjalaitteet ovat kytketty keskenään I²C-väylään. Isäntälaite toimii syöttö- ja tulostuslaitteena sekä pelialgoritmin laskennallisena laitteena. Aina, kun isäntälaite vastaanottaa orjalaitteelta sanoman tilan muutoksesta, tarkistaa se mahdolliset voittorivit. Jos voittorivejä ei ole, vaihdetaan pelaajan vuoro. Jos löytyy neljän suora vaaka-, pysty-, diagonaalisessa tai kolmiulotteisessa tasossa, peli lopetetaan ja pelaajaa informoidaan isäntälaitteeseen kytketyn LCD-näytön avulla.

Isäntälaitetta voidaan ohjata siihen liitetyllä kahdella painonapilla. Napeilla voidaan selata valikkorakennetta.

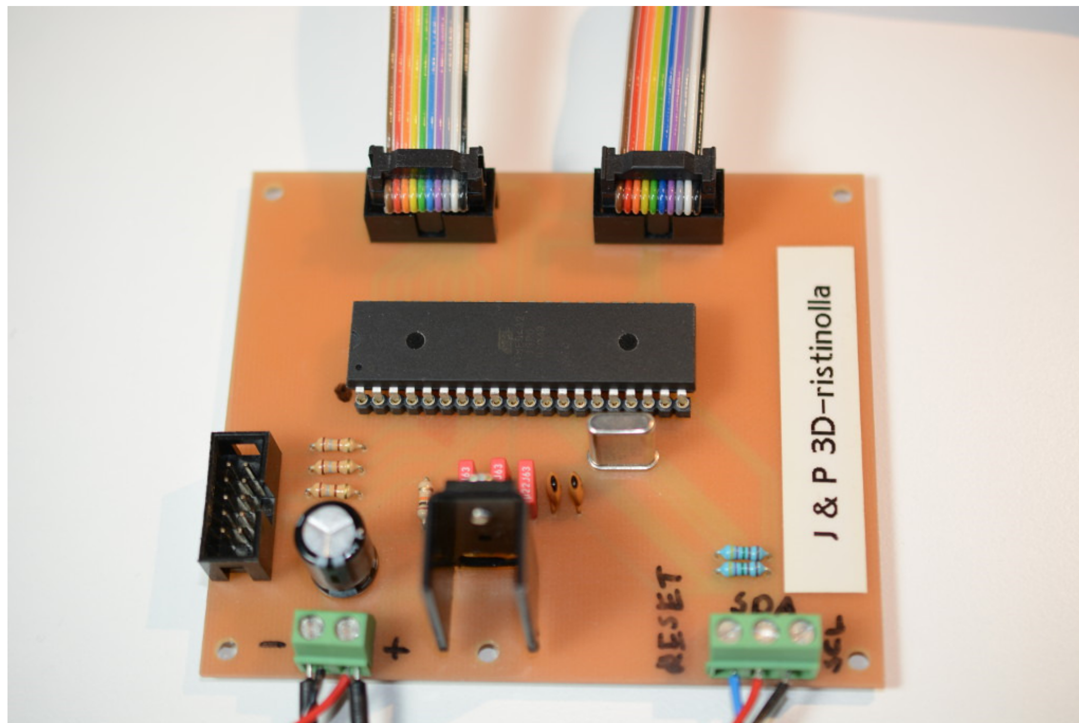
3.1.1 Isäntälaite

Alusta alkaen laitteesta päätettiin rakentaa vain yksi prototyyppiversio. Tästä syystä jokaiseen orja- ja isäntälaitteeseen sisällytettiin myös ohjelmointiliitännät, jotta käytetyt piirit voitaisiin ohjelmoida irrottamatta niitä itse laitteesta.

Jos ohjelmointiliitäntää ei huomioida, on isäntälaite hyvin yksinkertainen. Suorittimena käytetään AVR ATmega32 mikro-ohjainta. Ohjain valittiin, koska se on hyvin monipuolinen, sisältää kaikki tarvittavat liitännät ja on hyvin tehokas käskyjen suorituksessa. Myös mikro-ohjaimen suosio harrastelijapiireissä vaikutti hyvin paljon valintaan, koska siihen on saatavilla hyvät avoimet ja vapaat työkalut sekä kattava dokumentaatio. Tämän johdosta mikro-ohjaimella on myös hyvä saatavuus ja sen hinnoittelu on edullinen.

Ulkoisina komponentteina käytettiin jänniteregulaattoria laitteen virtalähteenä, ulkoista kellokidettä tahdistukseen ja liitäntää LCD-näytölle, painonapeille ja I²C-

väylälle sekä muita passiivisia komponentteja suodatukseen. Käytännössä isäntälaite on kehitysalusta eli arkikielessä tavallinen ”devaus” –kortti, ilman kaikkia ATmega32 mikro-ohjaimen liitäntöjä kytkettynä.

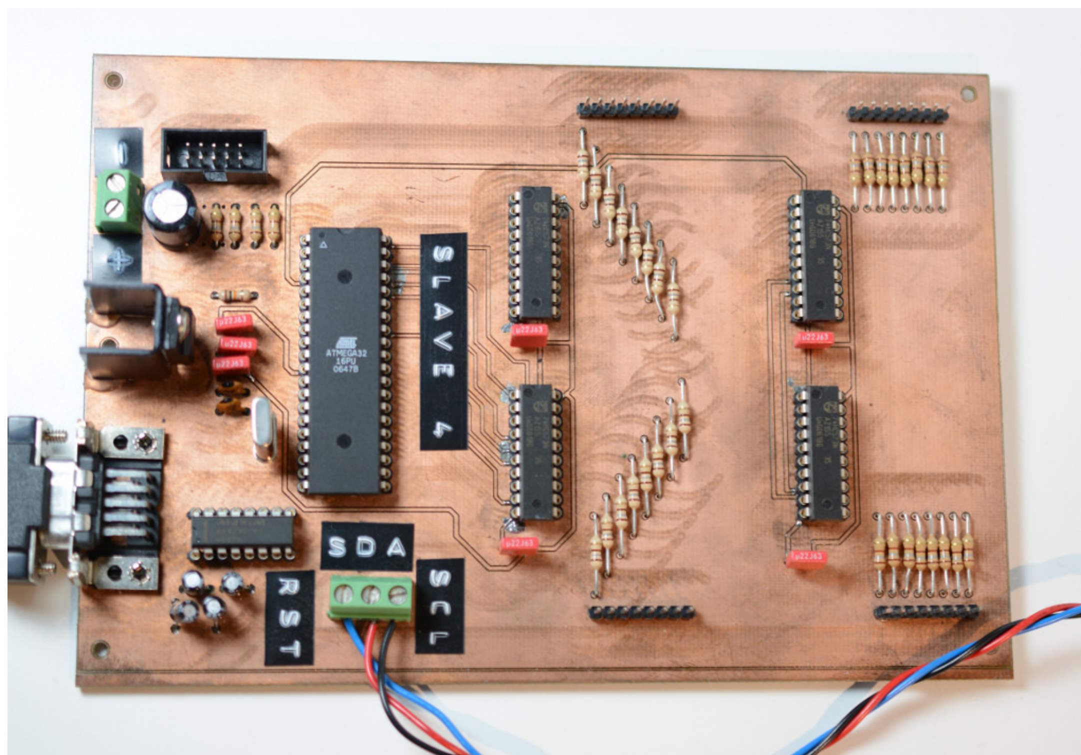


Kuvio 2. Isäntälaite.

3.1.2 Orjalaite

Orjalaitteessa päädyttiin käyttämään samaa mikro-ohjainta, jota käytettiin isäntälaitteessa eli ATmega32:ta. Jännitteen regulointi, ulkoinen kellokide, ohjelmointiliitäntä ja I²C-väylän liitäntä suunniteltiin samalla tavalla kuin isäntälaitteessa. Näiden lisäksi laitteeseen lisättiin UART-portti DB-9 liittimellä, jossa jännitesovitus hoidettiin Maxim MAX232 piirillä. MAX232 on piiri, joka muuntaa RS-232 sarjaväylän signaalit TTL-tasoisiksi. Tällä liitännällä laite keskustelee kosketuskalvojen kanssa sarjaväylän kautta. Lisäksi ledien ohjaamiseen on orjalaitteessa mikroprosessorin I/O-porttien vähyyden ja virrantarpeen vuoksi siirtorekisterit puskuroimassa niiden tilat ja ohjaukset.

Orjalaitteesta voi lukea lisää Janne Äijälän opinnäytetyössä ”Mikroprosessoriohjattu 3D-ristinolla, 2008”.



Kuvio 3. Orjalaite.

3.2 AVR ja oheislaitteet

AVR on kahdeksanbittinen mikro-ohjain, jonka kehittivät kaksi opiskelijaa Trondheimin yliopistossa Norjassa (Norwegian Institute of Technology), Alf-Egil Bogen ja Vegard Wollan. AVR perustuu mikro-ohjaimissa yleisesti käytettyyn muokattuun Harvard arkkitehtuuriin. Mikro-ohjaimen suunnittelufilosofiana on käytetty RISC käskykantaa, jossa pyritään pitämään suoritinpohjaiset käskyt mahdollisimman vähäisinä ja tehokkaina. AVR Mikro-ohjaimet ovat hyvin monipuolisia, vähävirtaisia, nopeita ja ne sisältävät yleensä ADC-muuntimia, TWI- ja SPI-väylän, laskureita, joilla voidaan toteuttaa PWM-signaaleja, TTL tasoisia I/O-portteja ulkoisille laitteille, hyvät rajapinnat laitteen testaukselle, paljon ohjelmointi- ja käyttömuistia ja monia muita ominaisuuksia. Tämän seurauksena mikro-ohjaimet ovat kohdistettu monille eri markkinoille, joista mainittavimmat ovat autoteollisuus, taloautomaatio, kodintekniikka, kodin viihdelaitteet, teollisuusautomaatio, valaistus ja niitä käytetään myös monissa muissa laitteissa. Käyttömahdollisuudet ovat lähes rajattomat. (The Story of AVR 2008.)

Bogen ja Wollan suunnittelivat AVR mikro-ohjaimen yliopistossa ja kirjoittivat siitä opinnäytetyönsä. He valmistuivat vuonna 1992 ja kehittivät mikro-ohjainta vielä parin vuoden ajan valmistumisen jälkeen. Ollessaan omasta mielestään valmiita, he etsivät mikro-ohjaimelle rahoittajan ja sopivat tapaamisen silloisen Atmel yhtiön johdon kanssa. Onnistuneen tapaamisen myötä perustettiin toimisto Trondheimiin. Kolme vuotta myöhemmin, vuonna 1997, lanseerattiin ensimmäinen AVR tuote markkinoille. Ilmaisten työkalujen ja halpojen emulaattoreiden ansiosta AVR sai suurta suosiota ja tämä näkyy nykypäivänä myös suosiona harrastajapiireissä. (The Story of AVR 2008.)

Piireille on olemassa hyvät ”korkean tason” ohjelmointirajapinnat C-kielellä, hyvä dokumentointi, ilmaiset työkalut, debuggerit ja emulaattorit. AVR mikro-ohjaimet ovat myös edullisia, vähävirtaisia ja tehokkaita; ne kykenevät lähes yhden käskyn suorittamiseen kellojaksolla. Kaikki edellä mainitut syyt vaikuttivat paljon siihen valintaan, miksi tässä opinnäytetyössä mikro-ohjaimeksi valittiin AVR ATmega ohjain.

Seuraavaksi käymme teoreettisesti läpi työssä käytettyjä pää-teknologioita, joita on sisäänrakennettu AVR mikro-ohjaimeen. Näiden lisäksi käsittelemme myös muita käytettyjä oheislaitteita.

3.2.1 I²C

Inter-Integrated Circuit, tunnetaan myös nimellä TWI (Two-wire interface) on alun perin Philipsin suunnittelema sarjamuotoinen väylä piirien väliseen kommunikointiin. Koska I²C oli historiallisesti rekisteröity tavaramerkki, käyttää Atmel väylästä nimeä TWI, joka on jatkettu versio I²C-väylästä. Väylää voidaan käyttää oheispiirien liittämisessä itse mikro-ohjaimeen. Oheispiiriä ovat esimerkiksi muuntimet, muistit, ohjaimet, toiset mikro-ohjaimet, anturit ja monia muita laitteita. Kyseisen väylän protokolla on sisäänrakennettu melkein jokaiseen markkinoilla myytävään AVR mikro-ohjaimeen ja muihinkin markkinoilla oleviin laitteisiin, joten se on hyvin yleinen. Väylä on myös suhteellisen helppo ottaa käyttöön ja SPI väylä ei ollut varsin soveltuva kyseiseen tarkoitukseen, joten I²C-väylän käyttäminen orja- ja isäntälaitteen väliseen kommunikointiin oli helppo valinta.

I²C on kaksijohtiminen väylä, joista toista johdinta käytetään datan välittämiseen (SDA) ja toista tahdistukseen (SCL). Väylä on vuorosuuntainen ja toimii isäntä-orja periaatteella, jossa on yksi lähettävä laite (isäntä) ja yksi vastaanottava laite (orja). Tiedonsiirto voi tapahtua vain yhteen suuntaan kerralla ja jokainen väylässä oleva laite voi toimia isäntälaitteena. Muut laitteet ovat kiinni väylässä avoin-kollektori lähdöllä korkeaimpedanssitilassa, jotta ne eivät vedä väylää alas, koska väylä on vedetty ylösvetovastuksien kautta käyttöjännitteeseen. Väylä on avoin-kollektori tyyppinen, mutta koska teknologia on toteutettu eristehilatransistoreilla, niin yleensä puhutaan avoimesta nielusta. On myös huomattava, että väylälle ei saa liittää eri jännitteillä toimivia laitteita, esimerkiksi 3.3 ja 5 voltin toteutuksia, vaan tasot tarvitsee sovittaa vaikka helpolla eristehilatransistorikytkennällä. (Horowitz & Hill 2015, 1034; I2C-Bus www-sivut 2016; NXP Semiconductors I2C spesifikaation datalehti 2016.)

Väylään voidaan liittää maksimissaan niin monta laitetta, mitä sen osoiteavaruus sallii. Osoitteet ovat seitsemän tai kymmenen bittisiä. Seitsemän bittisessä väylässä maksimi laitemäärä on teoreettisesti 128 kappaletta, mutta 16 osoitetta on varattu erikoiskäyttöön, joten todellinen osoitemäärä on 112 kappaletta seitsemän bittisillä osoitteilla. Samoin kymmenen bittinen osoiteavaruus mahdollistaa 1024 laitetta, joista samat 16 osoitetta ovat varattuja, joten todellinen laitemäärä on 1008 kappaletta. Atmelin käyttämä TWI väylä ei tue kymmenen bittisiä osoitteita ja työssä käytetään seitsemän bittistä osoiteavaruutta. Osoitteiden valintaan ei vaikuttanut valmistajien esimääritellyt osoitteet, joten ne valittiin itse. (Horowitz & Hill 2015, 1034; I2C-Bus www-sivut 2016; NXP Semiconductors I2C spesifikaation datalehti 2016.)

Toinen rajoittava tekijä on väylän maksimi kapasitanssi, joka on noin 400 pikofaradia. Kapasitanssirajoitus aiheuttaa myös puolestaan pituusrajoituksen väylälle, joka on teoreettisesti noin kaksi metriä. Kapasitanssi on myös yleisin syy, joka rajoittaa monissa tapauksissa laitteiden määrää ja nopeutta väylällä. Tähän voidaan vaikuttaa mitoittamalla linjan ylösvetovastukset oikein. I²C-väylä on suunniteltu lyhyille etäisyyksille, mutta aina on hyvä varautua ulkoisiin häiriöihin, varsinkin pienillä väylän jännitteillä. Pidemmillä johtimilla olisi hyvä käyttää johtimina kierrettyä paria. Suojatuilla kaapeleilla, parikierrolla, oikeilla impedanssisovituksilla, matalilla nopeuksil-

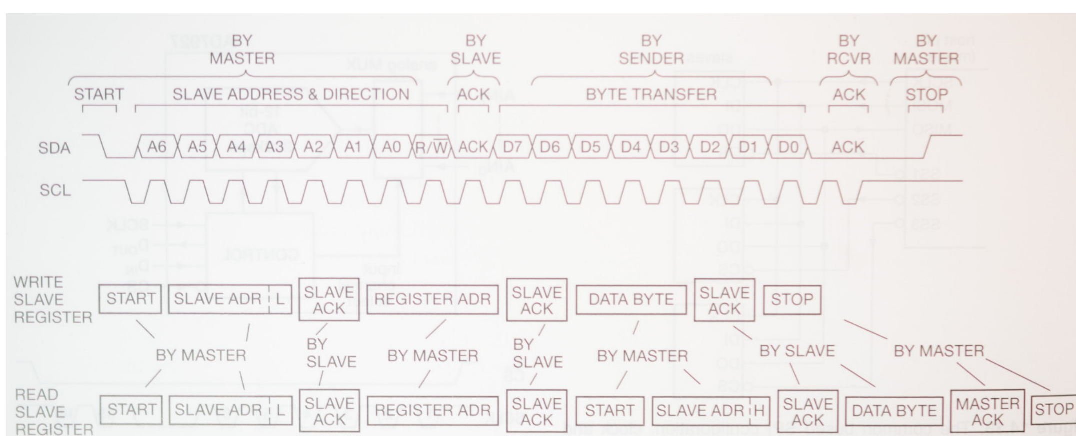
la ja erillisillä väylävahvistimilla voidaan muodostaa pidempi väylä. (Horowitz & Hill 2015, 1034; I2C-Bus www-sivut 2016; NXP Semiconductors I2C spesifikaation datalehti 2016.)

Kuten aiemmin jo kävi ilmi, on väylä isäntä-orja tyyppinen. Jokainen laite väylällä voi toimia isäntälaitteena ja se laite, joka aloittaa sanoman, toimii isäntänä. Väylässä voi olla useampi isäntälaite, mutta ne eivät voi toimia samaan aikaan, vaan ne toimivat vuorottain. Isäntälaitteet eivät myöskään voi keskustella keskenään. Väylän laitteiden ei tarvitse etukäteen sopia millä nopeudella toimitaan, vaan isäntälaite lähettää aina tahdistussignaalin, kun se aloittaa sanoman. I²C on siis synkroninen väylä. Orjalaitteilla pystyy myös tarvittaessa hidastamaan isännän tahdistusta vetämällä SCL linja alas eli nollapotentiaaliin. Tämä on tarpeellista esimerkiksi silloin, jos vastaanottava orjalaite on juuri tekemässä jotain pitkää laskelmaa eikä voi heti vastata isännän pyyntöön. Tätä kutsutaan ”clock stretching” –ominaisuudeksi eli kellon venytykseksi. (Atmel ATmega32 datalehti 2016.)

Yleisimmät käytetyt nopeudet väylällä ovat 100 kbit/s tai 400 kbit/s, mutta standardi sallii nopeudet aina 5 Mbit/s asti. Atmelin käyttämä TWI väylä ei toistaiseksi tue RISC mikro-ohjaimissa korkeita nopeuksia, vaan yleisin on 400 kbit/s. Kyseiset nopeudet eivät tarkoita, että väylää tarvitsee ajaa sillä nopeudella, vaan yleensä jokin laite tukee maksimissaan korkeinta ilmoitettua nopeutta. Väylä voi kyllä toimia hitaammin itse valitulla nopeudella ja vakionopeutta ei edes voi taata yllä mainitun kellon venytyksen, väylän kapasitanssin, johtimien pituuden, käytetyn jännitteen ja käytettyjen vastuksien takia. Nämä kaikki vaikuttavat osaltaan signaalin nousu- ja laskuaikaan eli nopeus on yleensä hyvä mitoittaa tilanteen mukaan. Koska väylä tahdistetaan erillisellä signaalilla, ei tarvitse myöskään välittää väylällä toimivien laitteiden sisäisistä kellotaajuuksista ja täten ongelmista, että laite ajautuisi pois tahdista, jos isännän ja orjan sisäiset kellotaajuudet eroaisivat paljon. On silti pidettävä huoli, että laitteen sisäinen kello on suurempi kuin väylän tahdistussignaali. ATmega32 mikro-ohjainta käytettäessä täytyy orjalaitteen sisäisen kellon olla vähintään 16-kertainen käytettyyn SCL väylän taajuuteen verrattuna. (Atmel ATmega32 datalehti 2016.)

I²C-väylän sanoman rakenne on yksinkertainen. Se koostuu yhdeksän bitin lohkoista aloitus (START) ja lopetus (STOP) ehtojen välillä. Lohko sisältää kahdeksan bitin

ryhmän, johon vastataan yhdellä ”hyväksytty” bitillä (ACK). Aina ensimmäisenä isäntälaite aloittaa keskustelun, jos väylä on vapaa eli SDA ja SCL –linjat ovat ylhäällä. Isäntä lähettää START –ehdon, jolloin väylään liitetyt laitteet alkavat kuunnella tulevaa osoitetta. Sitten lähetetään kohdelaitteen osoite. Osoite koostuu seitsemästä bitistä (0 - 127) ja yhdestä ”suunta” bitistä, joka kertoo onko tulevat sanomat kirjoitus- vai lukuoperaatioita. Toisin sanoen vastaanottaako vai lähettääkö isäntälaite paketin. Tämän jälkeen, jos väylältä löytyy orjalaite oikealla osoitteella, vastaa se siihen ACK bitillä. Seuraavaksi lähetetään itse data samalla periaatteella, mutta ilman ”suunta” bittiä eli datapaketin koko on aina kahdeksan bittiä. Näitä datapaketteja voidaan lähettää niin monta kunnes isäntä lähettää STOP –ehdon. (Horowitz & Hill 2015, 1034; I2C-Bus www-sivut 2016; NXP Semiconductors I2C spesifikaation datalehti 2016.)



Kuvio 4. I^{2}C-väylän sanoman rakenne. (Horowitz & Hill 2015, 1034.)

Kuten kuvasta huomataan, vaihtaa data tilaansa vain kellon ollessa alhaalla, jotta vältyttäisiin START ja STOP –ehtojen täyttymiseltä kesken tiedonsiirron. Väylällä onkin olemassa yleinen sääntö *”data voi vaihtua vain, kun kello on alhaalla”!* Erityistilanteita väylällä voi olla aiemmin mainittu tilanne, jossa orjalaite ei heti pysty vastaamaan isäntälaitteen pyyntöön, vaan joutuu ”venyttämään” (Clock stretching) tahdistusväylää kunnes on valmis vastaamaan. Toinen tilanne on toistettu-aloitus ehto (Repeated START), jossa esimerkiksi isäntä haluaa lähettää aluksi datapaketin ja sitten vaihtaa lähetyksen tilan vastaanottavaksi. Isäntälaite lähettää heti ACK sanoman jälkeen uuden START –ehdon ja lähettää uudestaan osoitteen ja vaihtaa kirjoitusbitin tilaa, jolloin isäntä odottaa lähetyksen sijaan vastausta orjalta. Muita tilanteita ovat

usean isännän tila (Multimaster) ja joitain laitteistospesifisiä ominaisuuksia, joihin tässä ei oteta kantaa. (Atmel ATmega32 datalehti 2016.)

3.2.2 Keskeytykset

Keskeytyspyyntö eli IRQ (Interrupt request), on melkein jokaisessa mikro-ohjaimessa sisäänrakennettu rautatason ominaisuus, jolla voidaan luoda välitön palvelupyyntö prosessorille ja keskeytyspyyntöä palvellaan sitten mahdollisimman nopeasti. Hyvin yleinen ja hyvä esimerkki on PS/2-väyläiset vanhat näppäimistöt tietokoneessa, joille luodaan välittömästi keskeytyspyyntö, kun nappia painetaan ja tietokone reagoi siihen mahdollisimman nopeasti, kun taas esimerkiksi USB-väyläiset näppäimistöt eivät aiheuta keskeytystä, vaan niiltä kysellään satoja kertoja sekunnissa ”onko raportoitavaa”. Keskeytyksiä voi olla mikro-ohjaimen ulkoisia tai sisäisiä. Ulkoiset keskeytykset liitetään ohjaimen pinneihin, kun taas sisäiset voivat olla esimerkiksi ajastin tai laskuri, jotka laukaistaan, kun tietyt ehdot täyttyvät, esimerkiksi ajastimen ylivuoto. Mikro-ohjaimissa olisi suositeltavaa käyttää keskeytyksiä, koska jatkuva laitteiden ”pollaus” aiheuttaa vain ylimääräistä työtä joka kellojaksolla eikä se ole aina kaikissa tilanteissa tarpeeksi reaaliaikaista. Sen sijaan, että kyseltäisiin kokoajan laitteilta tiloja, voitaisiin hyödyntää kyselyaika laittamalla laite lepotilaan ja säästäen näin energiaa.

Kun mikro-ohjaimen jokin tekniikka tai ulkoinen laite luo keskeytyksen, suorittaa mikro-ohjain kesken olevan konekielisen käskyn loppuun, estää oletuksena kaikki muut keskeytykset ja tallentaa tarvittavat liput ja sen hetkisen ohjelmalaskurin arvon pinomuistiin. Ohjelma-laskurin seuraavaksi muistiosoitteeksi asetetaan niin sanotun keskeytyspalvelurutiinin eli Interrupt Service Routine (ISR) osoite. Samanniminen funktio löytyy myös AVR:n ohjelmointirajapinnasta, jota käytetään keskeytyksien käsittelyssä. C-kielessä kääntäjä yleensä pitää huolen, että ISR:n sijaintiosoite sijoitetaan oikeaan paikkaan muistissa. Tätä muistipaikkaa kutsutaan keskeytysvektoriksi. Sitten keskeytyspalvelurutiini suorittaa tarvittavan ohjelmakoodin, jonka jälkeen palautetaan pinomuistista tallennettu ohjelmalaskurin arvo ja muut talletetut rekisterien arvot sekä sallitaan globaalit keskeytykset. Tämä voidaan hoitaa konekielisellä käskyllä RETI (Return from Interrupt), joka löytyy myös AVR:n C-kielen ohjelmointi-

rajapinnasta. Kääntäjä on kuitenkin hoitanut tämän ohjelmoijan puolesta, joten siitä ei tarvitse välittää. Tämän jälkeen jatketaan pääohjelman suoritusta. (Atmel ATmega32 datalehti 2016.)

Tätä samaa tekniikkaa käytetään ristinolla-pelin isäntälaitteessa nappien keskeytyksien generoinnissa. Kun nappia painetaan, luo mikro-ohjain keskeytyspyynnön ja suorittaa sille annetun tehtävän. Aiheesta enemmän Ohjelmisto ja Laitteisto -kappaleiden osiossa HID-laite.

3.2.3 HD44780

HD44780 on Hitachin valmistama pistematriisinäytön –ohjain, joka on hyvin yleisessä käytössä oleva ohjaintyyppi maailmalla. Melkein joka paikassa, jossa kyseistä pistematriisinäyttötyyppiä käytetään, on sitä ohjaamassa Hitachi HD44780 yhteensopiva tai siitä johdettu ohjain ja protokolla, joten siitä on lähestulkoon muodostunut alan standardi. Näytöistä löytyy useita kokoja, joista mainittavimmat ovat 2x16, 2x20 ja 4x20 -merkkiset näytöt. Yksi HD44780 ohjain pystyy näyttämään 80 merkkiä kerralla, joten siitä isommat näytöt vaativat useamman ohjaimen. (Wikipedia Hitachi HD44780 LCD Controller 2016.)

HD44780 liitäntä on aina 16-pinninen. Niiden järjestys on seuraavanlainen:

1. GND (nollapotentiaali)
2. Ucc (käyttöjännite +3 tai +5 volttia)
3. kontrasti (säädettävä käyttöjännitteestä potentiometrillä)
4. RS (Register Select, data- tai komentorekisterin valitsin)
5. R/W (luku tai kirjoitus)
6. Enable (aktivoidaan näyttö suorittamaan annetut käskyt)
7. D0 (data bitti LSB)
8. D1
9. D2
10. D3
11. D4
12. D5

13. D6
14. D7 (data bitti MSB)
15. A (taustavalo anodi +)
16. K (taustavalo katodi -)

Ilmeiset pinnit ovat 1, 2 ja 3 eli nollapotentiaali, käyttöjännite ja kontrasti. Pinnit 4, 5 ja 6 ovat ohjauspinnejä. Nämä pinnit määräävät mitä dataväylälle (pinnit 7 – 14) tehdään. RS-pinnillä kerrotaan, lähetetäänkö näytölle dataa tai komentoja. Data on merkkejä, kuten vaikka merkkiliteraali ”A” ja komennot ovat esimerkiksi ”siirrä osoitinta yhden vasemmalle”. Myös pinni 5 on aika selkeä, sillä määrätään, luetaanko näytöltä tietoa vai kirjoitetaanko sinne. Enable-pinni kertoo näytölle, että voidaan aloittaa datan tai komennon luku- tai kirjoitussuoritus. Se vedetään maihin hetkellisesti ja pulssin laskevalla reunalla näyttö alkaa suorittaa sille annettuja komentoja riippuen pinnien 4 ja 5 tilasta, esimerkiksi lukee data-rekisteristä sen hetkisen tilan. Pinnit 7 – 14 ovat datapinnejä (puhutaan myös portista), joista voidaan lukea tai joihin voidaan kirjoittaa valitun rekisterin tila. Kaksi viimeistä pinniä ovat taustavalon sähkönsyöttö. (HD44780 LCD Started guide 2016; Wikipedia Hitachi HD44780 LCD Controller 2016.)

Hyvin harvoin näytöltä luetaan mitään tietoa, vaan ennemmin kirjoitetaan, tämän takia R/W-linja usein kytketään nollapotentiaaliin. Esimerkki kirjoitustapahtumasta:

1. asetetaan RS-linjalle jännite tai nollapotentiaali riippuen kumpaan rekisteriin halutaan kirjoittaa, tässä tapauksessa data-rekisteriin
2. asetetaan R/W-linja nollapotentiaaliin, joka ilmaisee kirjoitustilaa
3. asetetaan mikro-ohjaimesta portti, johon näyttö on kytketty, ”ulostulo” tilaan
4. kirjoitetaan mikro-ohjaimen portin rekisteriin haluttu tieto, esimerkiksi ”A” kirjain (0x41h)
5. sitten liipaistaan Enable-linja, joka aktivoi näytön lukemaan data-rekisterin arvon
6. odotetaan vähän aikaa (joitain mikrosekunteja), koska yleensä mikro-ohjain on liian nopea verrattuna näyttöön, joten luodaan keinotekoinen viive ennen seuraavaa operaatiota.

Näyttö voi myös toimia neljän tai kahdeksan bitin tilassa. Tarkoittaen, että kahdeksan bitin tilassa käytetään kaikkia porttipinnejä väliltä 7 – 14. Tässä tilassa näyttö lukee esimerkiksi data-rekisterin tilan kaikista näistä pinneistä. Kun käytetään neljän bitin tilaa, niin porttipinnejä 7 – 10 ei käytetä lainkaan, vaan ne vedetään nollapotentiaaliin ja data kirjoitetaan pinneihin 11 – 14 kahteen kertaan puolitavu (englanniksi ”nibble”) kerralla. Ensiksi luetaan ylätavun arvo, minkä jälkeen Enable-pinniä liipaistaan, että näyttö lukisi bittien tilan ja sama operaatio toistetaan alatavun kohdalla. Neljän bitin tila on usein käytössä oleva tila, koska se ei vaadi niin montaa pinniä mikro-ohjaimelta ja täten näyttö voidaan kytkeä yhteen porttiin. (HD44780 LCD Started guide 2016.)

Työssä käytetty pistematriisinäyttö on Winstarin valmistava nelirivinen, 20-sarakkeinen ja HD44780 yhteensopivan ohjaimen omaava näyttö. Näyttötyypin yleinen saatavuus ja valmiit kirjastot vaikuttivat työssä sen valintaan. Kirjastojen saatavuus on hyvä myös Atmelin piireille ja ehkä yleisin käytössä oleva kirjasto on tunnetusti Peter Fleuryn kirjoittama. Myös AVR:n C-kirjasto tarjoaa kyseiselle näytölle matalan tason ajurin, mutta koska näytön ohjaaminen on työlästä, niin valmiin kirjaston käyttö helpottaa käyttöönottoa ja koodi on jo hyväksi todettua.

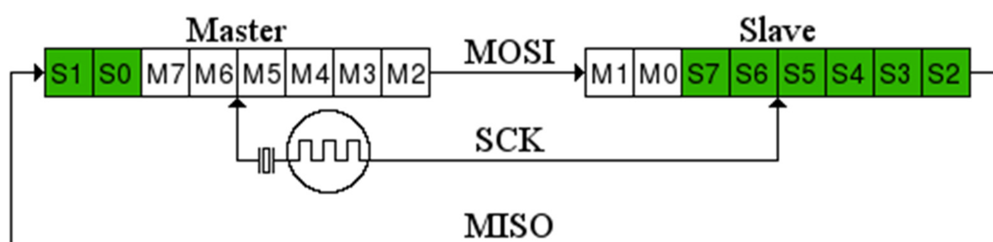
3.2.4 SPI & In-System Programming

Serial Peripheral Interface, eli SPI, on nykyään hyvin yleinen ja yksinkertainen laitteissa käytetty synkroninen kaksisuuntainen sarjamuotoinen protokolla. Se on alun perin Motorolan kehittämä ja on hyvin yleisessä käytössä. Esimerkiksi tunnettu muistikorttiformaatti Secure Digital käyttää kyseistä väylää kommunikointiin. Käyttökohteita on myös monia muita, mutta Atmel mikro-ohjaimet käyttävät väylää myös piirin ohjelmointiin. Väylää voi tietenkin käyttää mikro-ohjaimissa oheislaitteidenkin kanssa, mutta tässä artikkelissa keskitytään itse piirin ohjelmointiin. Kyseinen ominaisuus tunnetaan nimellä In-System Programming. (Wikipedia Serial Peripheral Interface Bus 2016.)

SPI on protokollaltaan hyvin yksinkertainen. Kuten I²C, SPI toimii isäntä-orja periaatteella, jossa on yksi lähettävä laite (isäntä) ja yksi vastaanottava laite (orja). Siinä

missä I²C-väylä vaatii aloitusbitin, kohdeosoitteen, ACK-bitin, datan ja niin edelleen, perustuu SPI väylä kahden laitteen kahdeksanbittisiin siirtorekistereihin, jotka synkronoidaan samalla kellolla, jonka isäntä luo. Koska siirtorekisterit ovat vain kahdeksanbittisiä, niin siirtotapahtuma vaatii vain kahdeksan kellojaksoa tahdistusväylältä, jonka jälkeen mikro-ohjaimen rekisteristä voidaan lukea saatu tavu talteen. Periaatteessa SPI vaatii vain kolme johdinta:

- lähettämiseen (MOSI – Master Out Slave In)
- vastaanottamiseen (MISO – Master In Slave Out)
- tahdistukseen (SCK – SPI Clock line).



Kuvio 5. SPI-väylän tiedonsiirto. (Maxembedded.com www-sivut 2016.)

Kuten kuvasta huomataan, on protokolla hyvin yksinkertainen. Siirtotapahtumassa siirretään dataa aina tavu kerrallaan: jokaisella kellopulssilla luetaan yksi bitti kunnes kahdeksannen bitin jälkeen mikro-ohjaimen sisäinen laskuri aiheuttaa keskeytyksen ja lukee saadun tiedon rekisteriin talteen. Kuvassa tiedon siirron suunta on siis isännältä orjalaitteelle. Kuvasta huomataan myös väylän kaksisuuntainen (tunnetaan arkikielessä nimellä full-duplex) luonne. Samalla, kun isäntä lähettää orjalle bitin, voi orja lähettää isännälle bitin. Siirtorekisteri toimii siis kuin FIFO-tyyppinen jono. (Maxembedded.com www-sivut 2016.)

Väylällä on kuitenkin olemassa myös neljäs johdin SS (Slave Select), jolla voidaan valita käytettävä orjapiiri, jos niitä on useampi. Hyvin usein käytössä on vain yksi isäntä- ja orjalaite, joten normaalisti sitä käytetään piirin aktivointiin eli SS-pinni asetetaan piirilevyn nollapotentiaaliin vain silloin, kun dataa siirretään. Tämä tila vastaa siirtorekisterin salpaamista. Väylällä voi myös olla useita isäntiä, mutta kyseinen tilanne on hyvin harvinainen ja monimutkainen tahdistuksen osalta, joten niihin ei tässä opinnäytetyössä puututa. (Maxembedded.com www-sivut 2016.)

In-System Programming eli ISP, on Atmel laitteissa SPI-väylän yhteyteen liitetty piirin ohjelmointirajapinta. Sen kautta voidaan ohjelmoida kaikki mikro-ohjaimen haihtumattomat muistit. Väylä helpottaa piirin ohjelmointia, koska itse piiriä ei tarvitse irrottaa ohjelmoinnin ajaksi, vaan riittää, että kortilta löytyy jonkin tyyppinen liitin ohjelmointilaitetta varten. Atmel tarjoaa kattavan dokumentoinnin ohjelmointiliitännän toiminnasta ja sen lisäämisestä projektille tai ohjelmointilaitteen rakentamista varten. (Atmel In System Programming datalehti 2016.)

SPI-väylään verrattuna ISP-liitäntä vaatii MISO, MOSI ja SCK –johtimien lisäksi käyttöjännitteen (U_{CC}), nollaus ($\overline{RESET}$) ja nollapotentiaali (GND) –liitännät. Muuten ISP toimii kuten SPI-väylä. Tietoa siirretään bitti kerralla joka kellopulssilla. ISP –tilassa ohjelmointilaite on aina isäntätilassa ja ohjelmoitava laite aina orjatilassa sekä isäntälaite tarjoaa aina tahdistussignaalin. On myös tärkeää vakaan toiminnan vuoksi pitää ohjelmointilaite ja ohjelmoitava piiri samassa nollapotentiaalissa, jottei laitteisiin tulisi potentiaalieroja tai muita häiriöitä, jotka voisivat vaikuttaa ohjelmointiin. Käyttöjänniteliitännän kautta voidaan ottaa sähkönsyöttö ohjelmointilaitteelle tai vaihtoehtoisesti koko piiriin voidaan syöttää sähköt ohjelmointilaitteen kautta. (Atmel In System Programming datalehti 2016.)

Tavallisesti nollausliittimestä käynnistetään mikro-ohjain uudelleen vetämällä se hetkellisesti nollapotentiaaliin, mutta ISP tilassa nollausliitin pidetään nollapotentiaalissa, jolloin mikro-ohjain siirtyy lähes heti ohjelmointitilaan. Tässä vaiheessa on tärkeää pitää tahdistuslinja vakaana, koska pienikin pulssin nouseva reuna voi aiheuttaa sen, että mikro-ohjain menettää tahdistuksen ohjelmointilaitteen kanssa. Tässä tilassa on odotettava vähintään 20 millisekuntia ennen kuin annetaan ensimmäinen komento. (Atmel In System Programming datalehti 2016.)

Ainoa komento, jonka mikro-ohjain ensimmäiseksi hyväksyy, on ”ohjelmoinnin aktivointi”. Ilman tätä komentoa ei voida tehdä mitään ja mikro-ohjain hylkää kaikki sille lähetetyt komennot. Kun aktivointikomento on lähetetty onnistuneesti, voidaan kaikki haihtumattomat muistit ohjelmoida, lukuun ottamatta tilannetta, jossa mikro-ohjaimesta on asetettu muistinsuojaus bitit päälle. Ohjelmoitavia muisteja ovat yleensä Flash ja EEPROM –muistit. Mikro-ohjain ei vastaa onnistuneeseen aktivointikomentoon millään lailla, mutta mahdollistaa muiden komentojen lähettämisen.

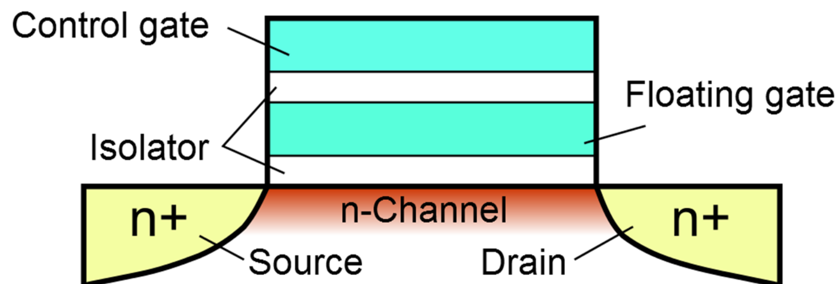
Yksi usein käytetty komento aktivoinnin onnistumisen tarkistamiseen, on lukea laitteen tunniste. Tunniste sisältää valmistajan, tuoteperheen (esimerkiksi AVR-laitteet), Flash-muistin koon ja tuoteperheen jäsenen (esimerkiksi työssä käytetty ATmega32) –tiedot. (Atmel In System Programming datalehti 2016.)

Itse komennot koostuvat neljästä tavusta, joista ensimmäinen tavu sisältää komennon koodin, onko luku- vai kirjoitusoperaatio ja kohdemuistin (Flash vai EEPROM). Toinen ja kolmas tavu sisältää kohdemuistin osoitteen ja neljäs tavu sisältää datan.

Kun mikro-ohjain on onnistuneesti laitettu ohjelmointitilaan, voidaan se tarvittaessa (tämä on suositeltavaa) tyhjentää ennen ohjelmointia ja sitten ohjelmoida käännetty ohjelma valittuun muistiin. Ohjelma on yleensä Intelin HEX formaatissa oleva tiedosto, joka on pino –tyylinen formaatti, jossa binäärinen data on kuvattu ASCII-muodossa rivi riviltä, jossa rivi sisältää vaihtelevan määrän tavuja. Itse ISP-protokolla on hyvin monipuolinen ja komentoja on niin paljon, että niihin ei tässä opinnäytetyössä puututa. (Atmel In System Programming datalehti 2016.)

3.2.5 EEPROM

EEPROM eli Electrically Erasable Programmable Read-Only Memory on nimensä mukaisesti lukumuisti, joka voidaan sähköisesti tyhjentää eli se on haihtumaton puolijohdemuisti. Siihen voidaan siis kirjoittaa tietoa ilman, että se katoaa, kun laitteesta poistetaan käyttösähkö. EEPROM kehitettiin vuonna 1977 ja sen toiminta perustuu kelluvan hilan omaavaan kanavatransistoriin. Päällä olevan ohjaushilan varausta kontrolloidaan transistorin johtavuudella ja ohjaushila on eristetty muusta piiristä niin hyvin, ettei sen varaus pääse vuotamaan pois. Koska transistorin rakenne eristekerroksen takia muistuttaa kondensaattoria, varautuu se myös samalla ilmiöllä ja se varataankin kapasitiivisesti. Tätä ilmiötä kutsutaan Fowler-Nordheim tunneloinniksi. Varauksen tilaa käytetään muistipaikan bitin tilan indikoinnissa eli onko se päällä (1) vai pois (0). Tähän transistorityyppiin perustuu suuri osa nykyisistäkin käytössä olevista haihtumattomista muisteista. (Wikipedia EEPROM 2016.)



Kuvio 6. Eristehilatransistori. (Wikipedia Floating-gate MOSFET 2016.)

EEPROM on hyvin yleinen käytössä oleva muistityyppi tiedon säilömistä varten ja mikro-ohjaimissa se on yleensä sisäänrakennettu itse ohjaimeen. Sitä voidaan ohjata omilla C-kielen funktioilla, jotka osoittavat mikro-ohjaimen oikeaan muistialueeseen eli EEPROM muistiin. Esimerkiksi tässä työssä sitä käytetään valikkojen kielen tilan tallentamiseen. Usein kaikki sisäänrakennetut EEPROM muistit mikro-ohjaimissa omaavat oman ohjelmointirajapinnan EEPROM muistialueelle kirjoittamiseen, mutta samaa muistia saa myös ulkoisena, yleensä kahdeksan pinnisissä paketeissa. Näitä voidaan ohjata jollain sarjamuotoisella protokollalla esimerkiksi työssä aiemmin puhutuilla SPI- ja I²C-väylillä, myös Microwire, UNI/O ja 1-wire ovat käytettyjä sarjamuotoisia protokollia, mutta niihin ei tässä työssä puututa. (Wikipedia EEPROM 2016.)

EEPROM-piiri voidaan kytkeä myös rinnakkain, joka on yleisin käytetty kytkentämuoto mikro-ohjaimen sisäisissä EEPROM muisteissa. Tällöin EEPROM-piiri sisältää yleensä niin leveän dataväylän kuin on piirin pienin osoitettava muistin koko. Yleensä tämä on tavun verran eli kahdeksan bittiä leveä. Tämän lisäksi on muistin osoiteväylä, jolla voidaan osoittaa koko muistialueeseen. Muita kytkentöjä ovat kohdasta SPI tutuksi tulleet SS eli ”chip-select” ja lisäksi kirjoituksen esto. Rinnakkaiskytkennän takia piirin koko ja pinnimäärä on niin iso, että sitä käytetäänkin vain sisäänrakennetuissa muisteissa ja ulkoisissa muisteissa käytetään hyvin usein yksinkertaisen kytkennän vuoksi sarjamuotoista väylää. (Wikipedia EEPROM 2016.)

EEPROM rakenteesta nouseekin kysymys, miksi ei käytettäisi mikro-ohjaimen sisäistä ohjelmamuistia (Flash) tiedon tallentamiseen jos se voidaan uudelleenohjelmoida, kuten EEPROM. Flash-muisti on EEPROM-muistista jatkettu muistityyppi ja

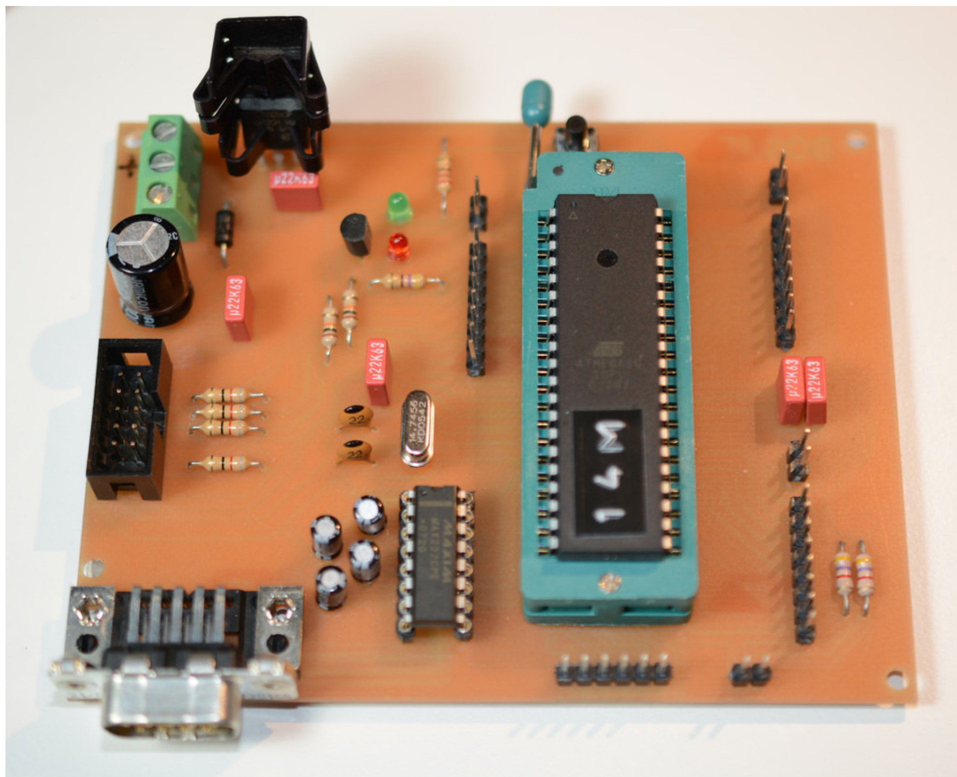
se on niin sanottu lohko (block) –tason muisti eli kun sinne kirjoitetaan tietoa, täytyy koko lohko uudelleen kirjoittaa. Tämä lohko voi sisältää useamman tavun kerralla, yleensä 512 tavua, kun taas EEPROM-muistissa voidaan aina osoittaa pienimpään kirjoitettavaan muistinalueen kokoon, joka on yleensä yksi tavu. Tämä puolestaan aiheuttaa sen, että EEPROM-muisti voidaan tyhjentää nopeammin kuin Flash-muisti, koska koko lohkoa ja sen osia ei tarvitse uudelleen kirjoittaa. Isommat muistit ovat yleensä näin lohkottuja, koska silloin tarvitaan vähemmän osoitteita osoittamaan pelkkiin lohkoihin. Yhden tavun kerralla kirjoittaminen myös vähentää riskiä siitä, että muistipaikka korruptoituisi, kun ollaan kirjoittamassa sinne tietoa. Tämän vuoksi EEPROM-muistia käytetään usein mikro-ohjaimissa erilaisten asetusten tallentamiseen.

4 LAITTEISTO

Tässä kappaleessa käsitellään isäntälaitteen fyysistä rakennetta, kasaamista ja siihen käytettyjä tekniikoita eli käydään läpi laitteen ja oheislaitteiden suunnittelu, työstäminen, kasaaminen ja kohdatut ongelmat. Orjalaitteita ja niiden suunnittelua käsiteltiin Janne Äijälän samannimisessä opinnäytetyössä.

4.1 Suunnittelu

Jo hyvin aikaisessa vaiheessa, melkein ensimmäisellä suunnittelukerralla, osattiin hyvin hahmottaa millainen isäntälaitteesta tulisi tehdä. Koska työn alkumetreillä suunniteltiin enemmän laitteen käyttäjän ja itse laitteen vuorovaikutusta, niin suuri osa suunnitellusta painottui orjalaitteisiin, kuten kosketuskalvoihin ja LED-indikaattoreihin. Tiedettiin, että tarvittiin jokin näyttö, josta käyttäjä voisi seurata pelin tilaa ja joitain nappeja, joista pystyisi ohjaamaan laitetta. Tiedettiin myös, että laitteita tulisi olemaan useita, joita ohjataan yhdellä keskeisellä laitteella ja ne pitäisi saada jotenkin keskustelemaan keskenään väylän kautta, joten oli hyvin selvää, että isäntälaite olisi yksinkertainen ja ettei siihen tulisi muita ulkoisia komponentteja kuin näyttö, napit, virtalähde, liitin käytettävälle väylälle ja työn prototyyppiluonteen takia ohjelmointirajapinta.



Kuvio 7. Geneerinen kehitysalusta.

Kuten kuvasta huomataan, on isäntälaite hyvin samantapainen kuin esimerkiksi pelkistetty AVR kehitysalusta, mutta ilman kaikkia mikro-ohjaimen pinnejä kytkettynä, joten sen suunnittelu ja rakentaminen eivät koituneet varsinaiseksi haasteeksi työssä. Isäntälaitteessa käytettiin jo tutuiksi tulleita ja ohjekirjan mukaisia tekniikoita, joten ongelmiakaan ei sen suhteen syntynyt.

4.1.1 CadSoft Eagle

Laitteet suunniteltiin CadSoft Eagle piirilevynsuunnitteluohjelmalla. Ohjelmaan sisältyy kaksi päänäkymää, joissa työskennellään. Ne ovat kaavio- ja piirilevynäkymä (Schematic- ja Board editor), joista ensimmäisessä suunnitellaan projektiin kuuluvat komponentit, miten ne ovat yhteydessä toisiinsa ja niiden nimeämiset sekä niihin liitettävät jännitepotentiaalit eli suunnitellaan kaaviokuva. Ohjelmassa on myös iso määrä valmiita elektroniikan komponentteja, joita voidaan käyttää suunnitteluun ja tarvittaessa komponenttikirjastoja voidaan myös itse lisätä. Työssä käytetyt komponentit löytyivät valmiina Eaglen kirjastosta. Piirilevynäkymässä suunnitellaan ja muotoillaan itse piirilevy (yksi vai useampi –kerroksinen), sijoitetaan komponentit

haluttuihin paikkoihin levyllä ja kytketään ne toisiinsa komponenttien välisillä vedoilla eli suunnitellaan kytkentäkaavio. Tässä näkymässä on olemassa myös automatisoitu ominaisuus, jolla ohjelma pyrkii itse suunnittelemaan komponenttien välisille vedoille järkevän reitin. Toiminta tunnetaan yleisesti nimellä ”autoroute” ja tämä ominaisuus voi löytyä myös muista piirilevynsuunnitteluohjelmista. Nykyään kyseinen ominaisuus osaa aika hyvin suunnitella sopivat reitit vedoille ja sitä kannattaakin käyttää alustavasti hyvin sekavassa piirilevyssä komponenttien sijoittelun jälkeen järjestystä helpottamaan. Sen jälkeen voidaan käsin korjata tarvittavat kohdat itselle mieluisemmaksi. Pienillä piirilevyillä on helpompaa tehdä johdotukset manuaalisesti.

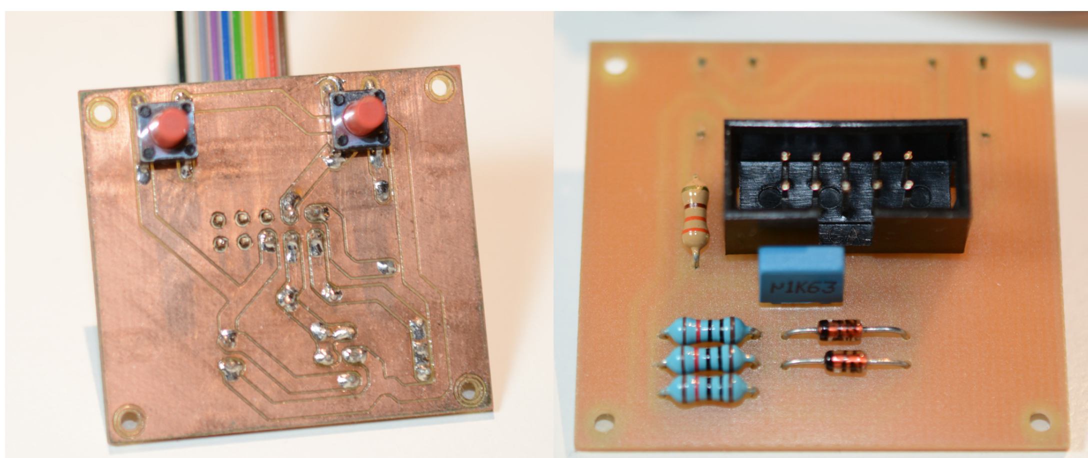
Tässä vaiheessa olisi hyvä huomioida piirilevyn valmistuksessa käytettävät tekniikat, esimerkiksi valotetaanko ja syövytetäänkö piirilevy vai jyrsitäänkö se, koska valottamalla voidaan tehdä hyvinkin ohuita vetoja, mutta jyrsimällä täytyy ottaa huomioon jyrsinterän leveys ja jyrsittävän uran syvyys. Vierekkäisille nastoille ja vedoille täytyisi jättää tarpeeksi tilaa, jottei jyrsimen terä rikkoisi jo jyrsittyä vetoa tai liitäntänastaa. Nykyään hyvin yleinen ja melkein pelkästään käytössä oleva tekniikka on valotus ja syövytys, koska se on myös huomattavasti nopeampi tapa. Muita huomioonotettavia asioita ovat tietenkin komponenttien sijoittelussa esimerkiksi liittimet, päästäänkö niihin helposti käsiksi ja muiden komponenttien kohdalla, onko niitä helppo työstää. Uudemmilla pintaliitoskomponenteilla tämä ei varsinaisesti koidu ongelmaksi, koska niitä harvoin käsin ladotaan ja piirilevyt juotetaan uunissa. Prototyypeissä useimmiten käytetään jalallisia, reiän läpi vedettäviä komponentteja, joten niiden sijoittelua voidaan pohtia tarkemmin.

Kun piirilevy on piirilevynäkymässä suunniteltu valmiiksi, voidaan siitä luoda erilaisia koordinaatiotiedostoja, esimerkiksi työssä käytetylle piirilevyjyrsimelle (Bungard CCD) CAM-tiedosto. Tämä kertoo jyrsimelle mistä mikäkin läpivienti, johdinveto ja leikkaussauma löytyvät ja näiden perusteella piirilevy voidaan jyrsiä. Eagle sisältää CAM Processor nimisen ohjelman, jolla kyseinen tiedosto voidaan luoda. Aiheesta enemmän kohdassa 4.2 Bungard CCD.

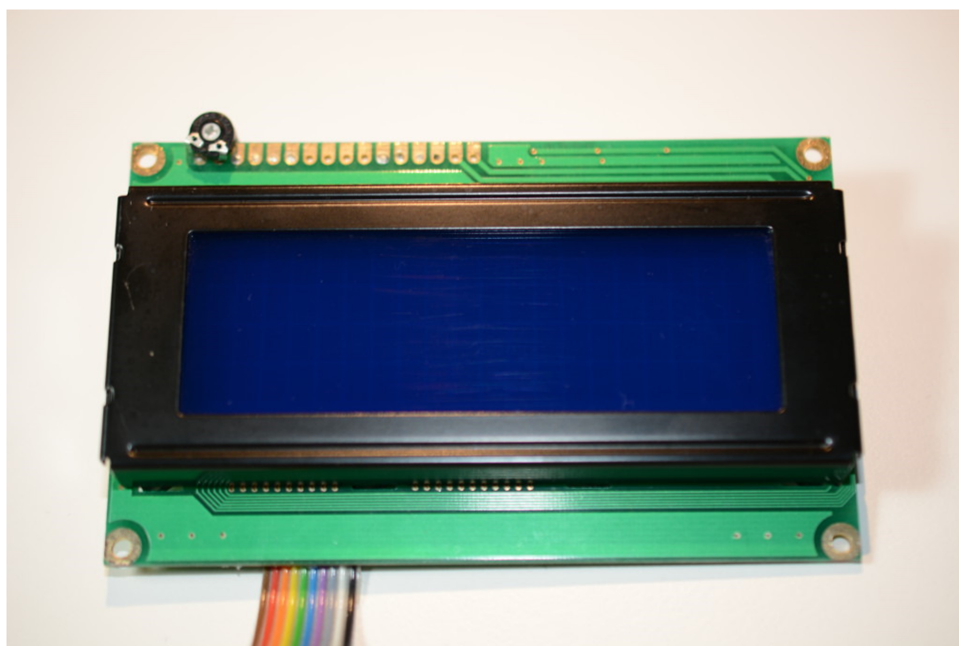
Isäntälaitteen yksinkertaisuuden vuoksi varsinaisia ongelmia ei suunnittelussa kohdattu. Oman elektroniikkaharrastuksen kautta tiedettiin jo suurin piirtein mitä komponentteja piirilevylle tarvittaisiin, joten suunnittelutyö oli hyvin suoraviivaista.

4.1.2 HID-laite

HID eli Human Interface Device on laite, jolla ihminen voi olla vuorovaikutuksessa koneen kanssa. Työn tapauksessa ne ovat isäntälaitteen napit ja näyttö. Tätä varten isäntälaitteelle suunniteltiin erillinen piirikortti napeille, joilla sitä pystyy ohjaamaan. Näyttö oli jo valmiiksi ostettu, joten sitä ei erikseen suunniteltu ja rakennettu, vaan sille lisättiin liitin mikro-ohjaimeen kiinnittämistä varten. Nämä pinnit kytkettiin suoraan mikro-ohjaimen A-porttiin. Työssä käytetyn näytön liitännät käytiin aikaisemmin läpi kohdassa 3.2.3 HD44780.



Kuvio 8. HID-laite, kytkimet.



Kuvio 9. HID-laite, pistematriisinäyttö.

Nappien piirikortti on pieni ja yksinkertainen. Se on kytketty isäntälaitteeseen kymmenen pinnisellä lattakaapelilla. Piirikortilla on kaksi painokytkintä, jotka ovat kummatkin liitetty mikro-ohjaimen INT1 keskeytysnastaan. Lisäksi kummatkin kytkimet on liitetty mikro-ohjaimen omiin D-portin pinneihin eli toinen kytkimistä on liitetty D-portin pinniin 6 ja toinen pinniin 7. Tällä mahdollistetaan se, että kytkimille ei tarvitse käyttää omia keskeytyksiä, koska mikro-ohjaimessa on käytettävissä vain kolme ulkoista keskeytysvektoria.

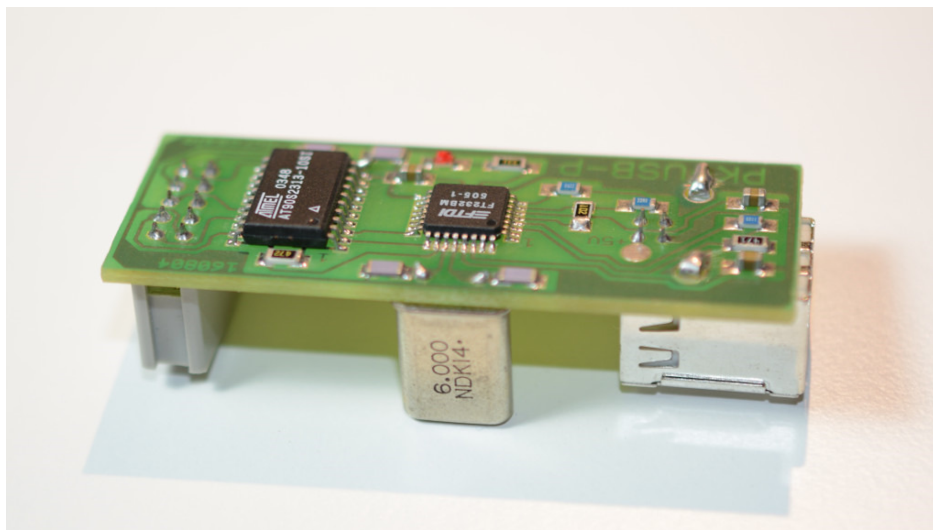
Kun jompaakumpaa kytkintä painetaan, niin painetun kytkimen porttipinni vedetään maihin eli nollapotentiaaliin ja luodaan ulkoinen keskeytys INT1, jonka jälkeen ohjelma siirtyy keskeytyspalvelurutiiniin. Palvelurutiini siirtyy keskeytysvektoriin INT1_vect ja suorittaa siellä olevan ohjelmakoodin. Ohjelmakoodissa tarkistetaan D-portin pinnien 6 ja 7 tilat eli onko jompaakumpaa kytkintä painettu. D-portin muut pinnit kuin 6 ja 7 ovat oletuksena vedetty mikro-ohjaimen sisäisten ylösvetovastuksien kautta käyttöjännitteeseen eli portti PORTD on tilassa 0x37 (0011 0111, ensimmäiset nollat vastaavat kytkimiä ja kolmas nolla on INT1 keskeytys). Tällä tekniikalla saadaan hyvinkin monta kytkintä piilotettua yhden keskeytyksen taakse, koska mikro-ohjain käsittelee aliohjelmassa kytkinten tilat niin nopeasti, että käyttäjä ei ehdi päästää kytkimestä irti ennen keskeytyspalvelurutiiniin loppumista.

Muita piirilevyn komponentteja ovat muutamat ylösvetovastukset, joilla pidetään D-portin pinnit 6, 7 ja INT1 viidessä voltissa. Muovikondensaattori poistamassa häiriöitä linjalta, ettei synny vääriä keskeytyksiä. Lisäksi linjavastus on maa-linjassa rajoittamassa kytkentävirtaa. Sekä kaksi diodia kytkinten yhteydessä keskeytyslinjassa, jotta kummatkin kytkimet voivat käyttää samaa keskeytyslinjaa ilman, että molemmat maadoittuvat toisen kytkimen painamisesta.

4.1.3 ISP

Laitteisiin lisättiin kymmenen pinniset liittimet Microsalon PK-USB-P (avr910 yhteensopivan) ohjelmointilaitetta varten, koska laitteet haluttiin ohjelmoida ilman, että mikro-ohjainta tarvitsee irrottaa. MOSI, MISO, SCK ja $\overline{\text{RESET}}$ –väylille lisättiin linjavastukset, jotta käytettäessä ISP-väylää oheislaitteiden kanssa, se ei häiritsisi SPI-

ohjelmointilaitetta. Työssä ei kuitenkaan käytetty ulkoisia SPI-laitteita, joten vastukset olivat periaatteessa ylimääräisiä komponentteja. Suunnitteluvaiheessa ei huomattu lisätä ohjelmointiliitännälle käyttöjännitettä ja koska käytössä ollut ohjelmointilaite tarvitsi virransyötön mikro-ohjaimelta toimiakseen, jouduttiin hyppylanka lisäämään jälkeenpäin liittimelle.



Kuvio 10. AVR910 yhteensopiva ohjelmointilaite.

4.1.4 Virtalähde

Työssä käytettyjen laitteiden virtalähteet ovat hyvin yksinkertaisia ja ne koostuvat ehkä yleisimmästä käytössä olevasta 78xx regulaattori perheestä. Isäntälaitteessa käytetty regulaattori on L7805CP eli viiden voltin jännitettä tarjoava lineaarinen regulaattori, jonka syöttöjännite on väliltä 7 - 35 volttia ja komponentin paketointi on tyyppiä TO-220FP. Regulaattori vaatii kaverikseen vain hyvin vähän ulkoisia komponentteja ja isäntälaitteessa onkin jännitettä tasoittamassa elektrolyyttikondensaattori, joka hoitaa myös äkillisiä virranvaatimuksia.

Isäntä- ja orjalaitteissa käytetään jokaisessa omaa regulaattoria, mutta periaatteessa kaikki laitteet olisivat voineet käyttää vain yhtä regulaattoria virtalähteenä. Koska orjalaitteet ja isäntälaite pinotaan, ovat ne hyvin lähellä toisiaan, joten johdinvedot eivät olisi olleet pitkiä ja näin ollen ei merkittäviä häviöitä olisi syntynyt pitkien johtimien takia. Näin lyhyet etäisyydet eivät olisi vaikuttaneet myöskään jännitteen vakauteen eli käytetty elektrolyyttikondensaattori olisi kokonsa (1000 µF) puolesta riit-

tänyt, koska laitteiden kokonaisvirrantarve on aika pieni, keskimäärin noin 60 mA per laite, isäntälaite mukaan lukien. Toisin sanoen häviöitä olisi ehkä voinut olla vähemmän pienemmällä määrällä komponentteja ja ylimääräisen kuorman takia olisi voitu mitoittaa jäähdytyssiili hieman isommaksi, jonka seurauksena lämpötila ei olisi vaikuttanut liian negatiivisesti regulaattorin elinikään. Tosin, työn prototyyppiluonteen vuoksi suunniteltiin jokaiselle laitteelle oma regulaattori, jotta niitä olisi helppo erikseen testata. (Keeping, S. 2016.)

Lineaariregulaattorit eivät ole hyötysuhteiltaan kovin hyviä, mutta mitä pienempi tulon ja lähdön jännite-ero on, sitä isompi on hyötysuhde. Tämä olisi voitu myös toteuttaa hakkurivirtalähteellä, mutta se vaatii monimutkaisen piirin, jotta toimisi hyvin ja hakkuri voi myös aiheuttaa huonosti suunniteltuna häiriöitä linjalle. Näistä syistä päädyttiin halpaan ja toimivaan lineaariregulaattoriin. Laitteen virtalähteessä on suositeltavaa käyttää noin 12 voltin jännitettä, koska käytetyn regulaattorin datalehden mukaan, regulaattorista saadaan eniten virtaa noin seitsemän voltin tulo- ja lähtöjännitteen erotuksella eli sen hyötysuhde on silloin parhaimmillaan. (Keeping, S. 2016.)

4.1.5 I²C

I²C-väylä on helppo ottaa käyttöön, koska sen protokolla on sisäänrakennettu itse mikro-ohjaimeen. Väylää varten ei ollut tarve suunnitella ylimääräisiä impedanssisovituksia, vaan piirilevylle suunniteltiin kolminastainen liitin, josta SDA ja SCL -väylät johdettiin suoraan mikro-ohjaimen vastaaviin pinneihin. Molemmat väylät vedettiin oman vastuksen kautta viiden voltin potentiaaliin ATmega32 datalehden mukaisesti. (Atmel ATmega32 datalehti 2016.)

Ylösvetovastukset lasketaan seuraavalla kaavalla, jos SCL-väylällä käytetty taajuus on alle tai yhtä suuri kuin 100 kHz.
Silloin pienin suositeltava vastus on:

$$\frac{U_{CC} - 0{,}4\,V}{3\,mA}$$

ja suurin suositeltava vastus on:

$$\frac{1000\,ns}{C_b}$$

missä C_b on väylän johtimen kapasitanssi pikofaradeissa. Nämä kaavat ovat suoraan mikro-ohjaimen datalehdestä, joten on aina muistettava ja huomioitava kerrannaisyksiköiden sovitus.

Kun SCL-väylän taajuus on yli 100 kHz, käytetään pienimmälle suositeltavalle vastukselle samaa laskentakaavaa kuin yllä, mutta suurimmalle suositeltavalle vastukselle seuraavaa kaavaa:

$$\frac{300\ ns}{C_b}$$

missä C_b on väylän johtimen kapasitanssi pikofaradeissa. (Atmel ATmega32 datalehti 2016.)

Koska isäntälaitteessa ei väylälle kohdistunut mitään erikoisvaatimuksia, niin siinä käytettiin Atmelin suosittelemaa 4,7 kilo ohmin vastusta. I²C-väylän liittimen kolmanteen nastaan tuotiin lisäksi yksi mikro-ohjaimen porttipinni, jolla voidaan resetoida orjalaitteet vetämällä portin pinni nollapotentiaaliin, tämä kytkettiin suoraan orjalaitteiden $\overline{\text{RESET}}$-pinneihin. (Atmel ATmega32 datalehti 2016.)

4.1.6 Oheiskomponentit

Yllämainittujen komponenttien lisäksi isäntälaitteessa ei ollut kuin ulkoinen 14,7456 MHz kellokide, kellokiteelle sovitetut keraamiset kondensaattorit ja pari muuta muovikondensaattoria häiriönpoistoa varten. Kellokiteen kuormakapasitanssin tarvittava arvo tietyllä kiteen taajuudella saadaan selville kiteen datalehdestä, mistä voidaan laskea tarvittavat kondensaattorin arvot. Tarvittavat kondensaattorin arvot voidaan myös katsoa mikro-ohjaimen datalehdestä, jossa on annettu suositeltavat arvot tietyille kiteen taajuuksille ja sen perusteella kiteelle valittiin 22 pF kondensaattorit. Kellokiteen 14,7456 MHz nopeus valittiin, koska orjalaitteiden kosketuskalvoissa käytetty USART-lähetin on vähiten virhealtis kyseisellä nopeudella mikro-ohjaimen datalehden mukaan.

f_{osc} = 14.7456MHz			
U2X = 0		U2X = 1	
UBRR	Error	UBRR	Error
383	0.0%	767	0.0%
191	0.0%	383	0.0%
95	0.0%	191	0.0%
63	0.0%	127	0.0%
47	0.0%	95	0.0%
31	0.0%	63	0.0%
23	0.0%	47	0.0%
15	0.0%	31	0.0%
11	0.0%	23	0.0%
7	0.0%	15	0.0%
3	0.0%	7	0.0%
3	-7.8%	6	5.3%
1	-7.8%	3	-7.8%
0	-7.8%	1	-7.8%
921.6Kbps		1.8432Mbps	

Kuvio 11. UBRR rekisterin kerroin ja virhemarginaali 14,7456 MHz taajuudella. (Atmel ATmega32 datalehti 2016.)

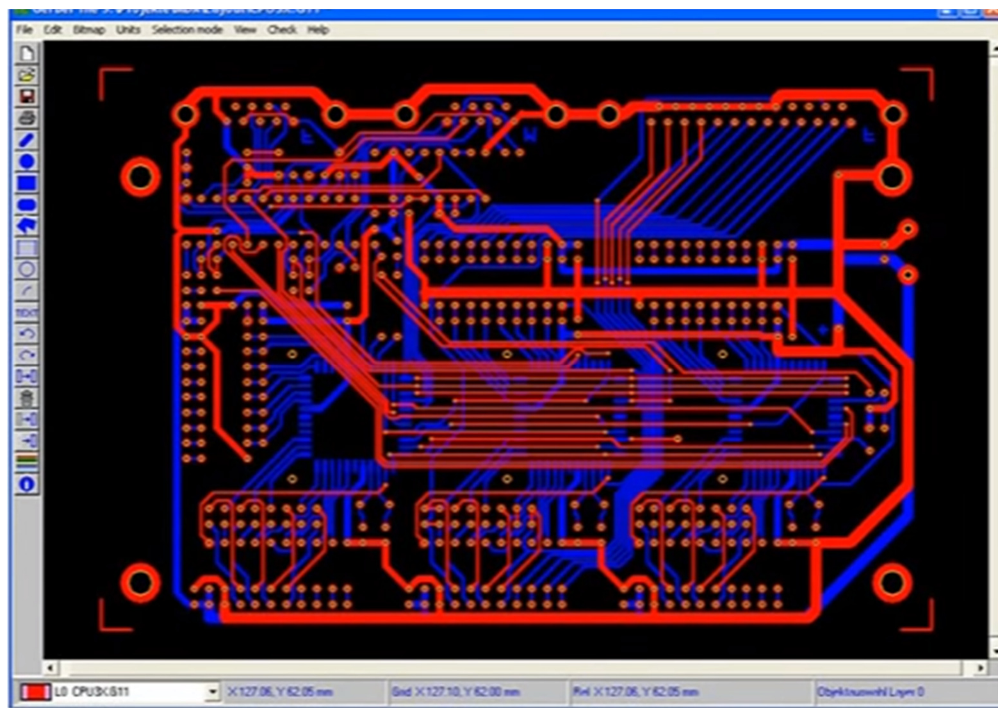
4.2 Bungard CCD

Bungard CCD (Computer Controlled Drilling) on tietokoneohjattu jyrsin, jolla voidaan jyrsiä kaiken tyyppisiä materiaaleja. Sitä voidaan käyttää muun muassa alumiinin ja muovin kaiverrukseen, pintaliitoskomponenttiliiman annosteluun, juotospastan annosteluun ja teknisten piirustusten piirtämiseen. Useimmiten jyrsintä käytetään kuitenkin piirilevyjen jyrsimiseen (vetojen eristämiseen), reikien poraamiseen ja leikkaamiseen. Jyrsimellä voidaan vaihtaa jyrsinterä automaattisesti, mutta koulussa käytössä olleeseen jyrsimeen vaihdettiin terät käsin.

Kuten aikaisemmin kohdassa 4.1.1 CadSoft Eagle mainittiin, käytetään jyrsimen kassa CAM-tiedostoja jyrsintään. CAM-tiedostot esiintyvät eri formaateissa ja työssä käytetyt formaatit ovat EXCELLON ja HPGL, joista ensimmäinen sisältää koordinaatit porausta varten ja jälkimmäinen koordinaatit jyrsintää varten.

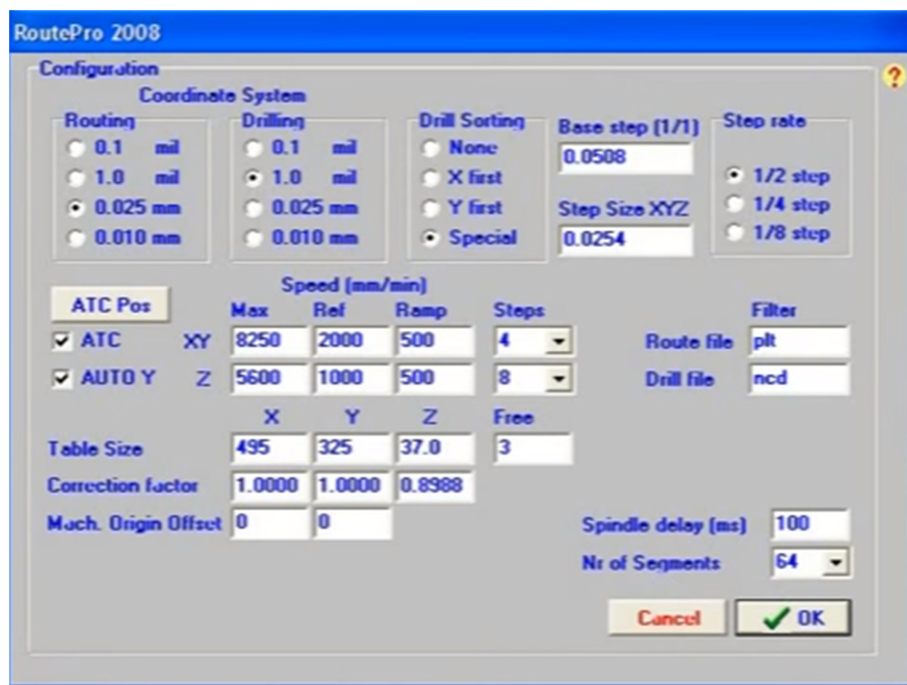
Bungard jyrsintä käytettäessä tarjoaa sen valmistaja kaksi ohjelmaa jyrsimen käyttöä varten. Nämä ovat IsoCAM ja RoutePro -ohjelmat. IsoCAM-ohjelmaa käytetään

Eaglesta saadun CAM-tiedoston konvertointiin muihin formaatteihin, kuten yllämainitut EXCELLON ja HPGL. IsoCAM-ohjelmalla voidaan myös alustavasti sovitella ja kohdistaa porattavat reiät ja jyrsittävät piirilevyjohtimet ennen varsinaista jyrsimistä. Kun lopulliset EXCELLON ja HPGL tiedostot ovat saatavilla, niin voidaan aloittaa itse jyrsiminen, johon käytetään RoutePro-ohjelmaa.



Kuvio 12. Esimerkkikuva IsoCAM-ohjelmasta. (Bungard – CCD 2011.)

RoutePro-ohjelmaa käytetään varsinaiseen jyrsimen ohjaamiseen joko manuaalisesti tai aikaisemmin mainittujen tiedostojen perusteella. RoutePro-ohjelmalle kerrotaan myös mitä teriä käytetään ja sillä myös asetetaan muita arvoja ennen jyrsintää, kuten nopeus, hidastukset ja kiihtyvyydet, korjauskertoimet x-, y- ja z-akseleille, jyrsinpään askeleen pituus (kuinka paljon liikutaan kerralla) ja monia muita arvoja.



Kuvio 13. Esimerkkikuva RoutePro-ohjelmasta. (Bungard – CCD 2011.)

Kun aloitetaan jyrsiminen, on piirilevy huolellisesti kiinnitettävä jyrsintasolle ja on varmistettava, että jyrsinterä pääsee vapaasti liikkumaan piirilevyllä ilman, että se osuisi kiinnikkeisiin. Jyrsittäessä syntyy paljon purua ja lastuja, joten ne olisi hyvä imuroida tai puhaltaa pois. Bungard jyrsimeen saa imurin lisätarvikkeena, mutta koululla se ei ollut käytettävissä. Jos piirilevy on jostain syystä päässyt nousemaan irti jyrsintasolta, niin jyrsimen joustavan istukan ansiosta jousikuorma pitää suulakkeen aina piirilevyn pinnassa kiinni (jos suulake on käytössä). Jyrsittäessä on aina pidettävä huolta, että piirilevy pysyy hyvin paikallaan ja samassa asennossa jos esimerkiksi jyrsitään kaksipuoleisia piirilevyjä. Lopussa todettiin, että piirilevyn jyrsiminen on työvaiheena aika työläs esimerkiksi verrattuna piirilevyn valottamiseen ja syövyttämiseen, joten jatkossa piirilevyn tuottaminen hoidettaisiin eri tekniikoilla.

4.3 Laitteiston kasaaminen

Kun komponentit olivat saatavilla, kasattiin isäntä- ja orjalaitteet kaikki samalla kertaa. Heti huomattiin, että jyrsitty piirilevy oli hankala ottaa käyttöön. Piirilevyn kaikki vedot ja nastat täytyi erikseen mitata maata vasten yleismittarin ”summeri” -testerillä, jotta jyrsimisessä syntyneitä kuparilastuja ei olisi jäänyt levylle, mikä aiheuttaisi oikosulkuja. Myös johtimien ohuet vedot olivat jyrsimisen seurauksena

herkkiä katkeamaan, joten juotettaessa täytyi olla varovainen. Samoin ympärille ja joidenkin johtimien väliin jääneet kuparikaistaleet voivat aiheuttaa piirissä häiriöitä tai kapasitanssia, jos niihin pääsee indusoitumaan jännitteitä. Tämän seurauksena, piirilevyn jyrsimistä ei todennäköisesti tulla tulevaisuudessa käyttämään, vaan piirilevyjen tekeminen toteutetaan valottamalla.

Muuten laitteiston kasaaminen oli aika suoraviivaista ja isäntälaite vähine komponentteineen kasattiin nopeasti. Suunnittelu oli siinä mielessä onnistunut hyvin, että juotettaessa komponenteilla oli tarpeeksi tilaa ja ne eivät päässet lämpenemään liikaa pienten juotosvälien seurauksena. Joitain ongelmia esiintyi tilattaessa komponentteja, lähinnä orjalaitteiden siirtorekistereiden ja kosketuskalvojen osalta, muuten osat tulivat nopeasti ELFA Elektroniikka tukkurilta.

5 OHJELMISTO

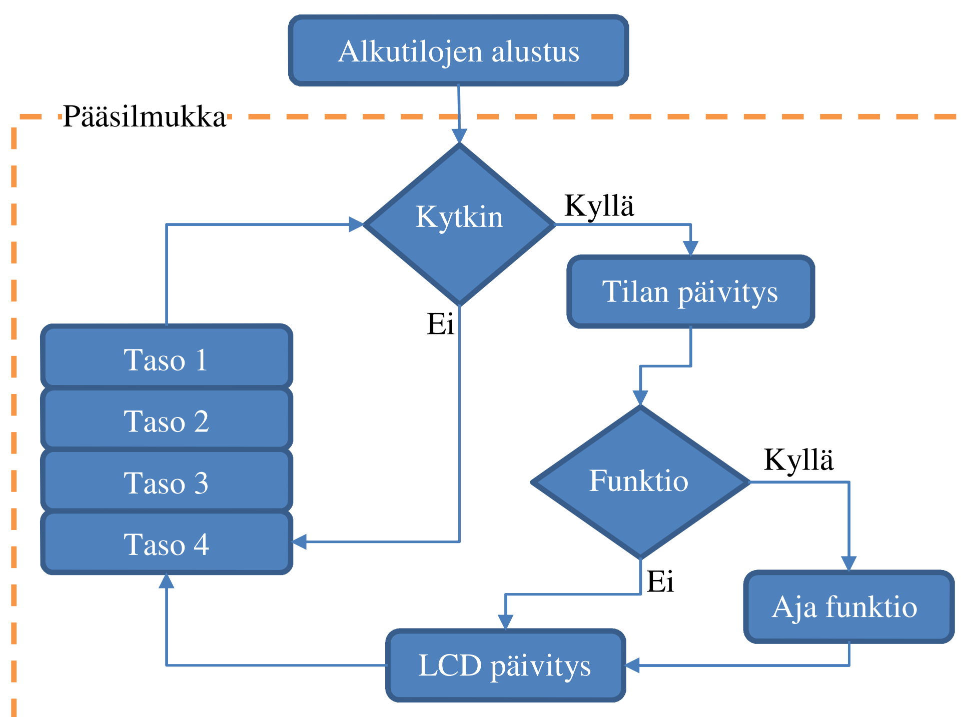
Seuraavaksi käydään läpi mikro-ohjaimen ohjelmiston käyttöliittymän rakenne, C-ohjelmointikielen sallimat valikon erityisrakenteet, näytölle suunniteltu puskuri, käytettyjen tekniikoiden vaatimat alustukset, tallennusmuisti ja käytetyt algoritmit. Tutustutaan myös yleisesti väylän sanomiin ja ohjelman kulkuun.

Ohjelmoinnissa käytettiin Debian käyttöjärjestelmässä pyörivää Eclipsen CDT –ohjelmointiympäristöä ja AVR Eclipse –liitännäistä. Eclipse mahdollisti helposti ison ohjelman ylläpidon ja lähdetiedostojen välillä siirtymisen ja AVR –liitännäinen toi kehitysympäristöön työkalut, joilla pystyi valitsemaan tarvittavat FUSE-moodit, käytetyn piirin, taajuuden ja ohjelmointilaitteen konfiguraation. Tämä olisi voitu hoitaa myös käsin, mutta AVR –liitännäinen hoiti nämä vaiheet automaattisesti valittujen konfiguraatioiden pohjalta ja nopeutti ohjelmointityötä huomattavasti. Kyseinen liitännäinen käyttää samoja työkaluja, joita normaalisti käytettäisiin ohjelman kehittämisessä AVR mikro-ohjaimiin. Käännetyn ohjelman lataamiseen mikro-ohjaimeen käytettiin AVRDUDE (AVR Downloader/UploaDEr) –nimistä ohjelmaa. Se on hyvin monipuolinen ja melkein pelkästään käytössä oleva lataaja-ohjelma harraste- sekä osittain ammattipiireissä, niin Windows- kuin Linux-alustalla. Kääntämiseen käytettiin avr-gcc –kääntäjää, avr-binutils –työkaluja ja avr-libc –ohjelmointikirjastoa. Ohjelmointilaitteena käytettiin geneeristä AVR910 yhteensopivaa SPI –ohjelmointilaitetta ja myöhemmässä vaiheessa huomattavasti nopeampaa Atmel-ICE –ohjelmointilaitetta.

5.1 Ohjelman kulku ja rakenne

Ohjelman rakenne perustuu tilakone –tyyppiseen ratkaisuun, jossa jonkin ohjelmallisen tai ulkoisen interaktion perusteella valitaan tila, jonka pohjalta suoritetaan asiat: esimerkiksi siirrytäänkö napin painalluksesta valikossa ylös, alas, eteen vai taakse. Tällainen tilakone –ajattelu on hyvin yleistä mikro-ohjaimissa, kun rakennetaan valikkoja ja muitakin ratkaisuja. Se helpottaa suunnittelu- sekä ohjelmointityötä ja ohjelman loogista rakennetta on helppo seurata.

Tilakone –mallia sovellettiin koko työhön lähinnä valikkojen takia ja kyseisen valikkorakenteen mahdollisti osaltaan tehokas C-kielen ominaisuus luoda osoittimia funktioihin, joista kerrotaan lisää kohdassa 5.2.3 Funktio-osoitin. Ohjelmoidessa pohdittiin myös mahdollisesta vuorottajan ohjelmoimisesta, jolla olisi saatu tilakone rinnastettua muiden ohjelman komponenttien kanssa, kuten esimerkiksi näyttöpuskurin päivityksen. Tämä kuitenkin todettiin heti alussa hyödyttömäksi, koska noin 14 MHz taajuudella laitteen käyttö oli hyvin sujuvaa ja vuorottaja olisi osaltaan tuonut työhön muita ongelmia. Käydään seuraavaksi läpi ohjelman yleinen perusrakenne vuokaaviona.



Kuvio 14. Ohjelman perusrakenne vuokaaviona.

Kuten vuokaaviosta huomataan, on ohjelman rakenne hyvin suoraviivainen. Ensimmäiseksi alustetaan mikro-ohjaimen tarvittavat rekisterit oikeisiin tiloihin, jonka jälkeen siirrytään itse pääsilmukkaan, jossa pyöritään koko ohjelman suorituksen ajan. Pääsilmukassa tutkitaan tilakoneen muuttujien tiloja, joita on useita. Ensimmäiseksi tutkitaan onko HID-laitteen kytkintä painettu, jonka pohjalta siirrytään joko suorittamaan tilan päivitystä tai tasojen tarkasteluja. Tilan päivityksessä alustetaan tilakone kytkimen painalluksen osoittamaan tilaan. Jos seuraava tila on funktio, ei nykyistä tilaa alusteta siihen, vaan se suoritetaan erikseen tilakoneen jälkeen. Aina tilan muutoksen yhteydessä päivitetään näytön puskuri, joka käy piirtämässä nykyiseen tilaan

perustuen valikon rakenteen tekstit näytölle. Lopuksi käydään läpi tasojen tilat ja niissä tutkitaan onko muodostunut mahdollisia voittorivejä. Seuraavissa kappaleissa käydään läpi eri tilojen ohjelmallista toteutusta.

5.2 Laitteen tilat

Laitteen tilakoneen rakenne perustuu hyvin yleisessä käytössä olevaan valikkorakenne ideaan, joka perustuu alun perin Atmelin kehittämään AVR Butterfly kehitysalustan valikkorakenteeseen. Rakenne on hyvin suoraviivainen ja siihen on helppo lisätä uusia valikoita ja alivalikoita. Siinä on kuvattu valikon rakenne perustiloina, joilla on kytkimien eri tilat esimerkiksi ”painallus ylös” tai ”painallus alas” ja näillä kummallakin kytkimen tilalla on niitä vastaavat seuraavat tilat. Näin voidaan luoda useita eri tiloja, joilla on aina sama rakenne ja ne osoittavat aina seuraavaan ja edelliseen tilaan.

```
MENU_NEXTSTATE menu_nextstate[] = {
            // Tila                    Painettu nappi          Seuraava tila
            {ST_NEW_GAME,              KEY_UP,                 NULL},
            {ST_NEW_GAME,              KEY_ENTER,              ST_NEW_GAME_FUNC},
            {ST_NEW_GAME,              KEY_DOWN,               ST_HIGH_SCORES},

            {ST_HIGH_SCORES,           KEY_UP,                 ST_NEW_GAME},
            {ST_HIGH_SCORES,           KEY_ENTER,              ST_HIGH_SCORES_FUNC},
            {ST_HIGH_SCORES,           KEY_DOWN,               ST_SETTINGS},

            {ST_SETTINGS,              KEY_UP,                 ST_HIGH_SCORES},
            {ST_SETTINGS,              KEY_ENTER,              ST_SETTINGS_LANG},
            {ST_SETTINGS,              KEY_DOWN,               NULL},

            // NULL rivi indikoimassa viimeistä alkiota.
            {NULL,                     NULL,                   NULL}
};
```

Kuvio 15. Esimerkki valikoiden tiloista ja perusrakenteesta.

Kuvasta huomataan valikoiden perustilojen ja kytkimien suhde. Jos uusi peli (ST_NEW_GAME) –tilassa painetaan alaspäin kytkintä (KEY_DOWN), niin tilakone hakee taulukosta seuraavan tilan, joka on esimerkin tapauksessa ennätykset (ST_HIGH_SCORES) –tila. Kuvasta huomataan myös sen hetkisen tilan funktiokutsu, joka on osoitin valikon tilan tekstin nimiseen funktioon. On myös huomattava, että valikon ensimmäinen ja viimeinen tila päättyy aina NULL –tyyppiseen tilaan. Tätä tilaa ei käsitellä ja kyseistä valintaa ei piirretä ruudulle. Valikon rakenteeseen

liittyy myös toinen tilataulukko, joka lähinnä ilmoittaa nykyisen tilan valikkotekstin ja jos tila on funktio, niin funktion nimen.

```
MENU_STATE_C menu_state[] = {
            // Tila                    Tilan teksti             Tilan funktio
            {ST_NEW_GAME,              MT_NEW_GAME,             NULL},
            {ST_NEW_GAME_FUNC,         NULL,                    new_game},
            {ST_HIGH_SCORES,           MT_HIGH_SCORES,          NULL},
            {ST_HIGH_SCORES_FUNC,      NULL,                    high_score},
            {ST_SETTINGS,              MT_SETTINGS,             NULL},
            {ST_SETTINGS_LANG,         MT_SETTINGS_LANG,        NULL},
            {ST_SETTINGS_LANG_FUNC,    NULL,                    language},

            // NULL rivi indikoimassa viimeistä alkiota.
            {NULL,                     NULL,                    NULL},
};
```

Kuvio 16. Esimerkki valikoiden teksteistä ja funktioista.

Kuvasta huomaamme aiempana esitellyn esimerkkivalikon tilojen suhteet valikon teksteihin ja niiden funktioihin. Jokaiselle valikon tilalle on määritelty oma rivi, jolla ilmoitetaan tilan teksti tai funktio. Kuten kuvasta voidaan huomata, aina jos tilalla ei ole tekstiä, niin voidaan olettaa, että sillä on funktio ja päinvastoin. Tätä ominaisuutta voidaan myös myöhemmin käyttää hyväksi tilakoneen suunnittelussa. Kummatkin yllämainitut tilataulukot kuvataan ohjelmakoodissa struct –tietotyyppeinä.

```
typedef struct {
    unsigned char state_name;
    unsigned char state_button;
    unsigned char next_state;
} MENU_NEXTSTATE;
```

```
typedef struct {
    unsigned char state;
    char *p_state_text;
    char (*p_state_func)(char input);
} MENU_STATE_C;
```

Kuvio 17. Esimerkkitaulukoiden tietotyyppirakenteet.

Kuten kuvasta huomataan, on rakenteet määritelty omina tietotyyppeinään (typedef avainsana). Valikon rakennetta kuvaava tietotyyppi MENU_NEXTSTATE on yksinkertainen ja sisältää vain kolmen eri tilan, yhden tavun kokoista muuttujaa. Kun taas tietotyyppi MENU_STATE_C rakentuu sen hetkisestä tilasta, merkkijono- ja funktio-osoittimesta, joista merkkijono-osoitin osoittaa ohjelmakoodissa ennalta määriteltyyn merkkijonoon. Nämä ovat valikon tekstejä. Funktio-osoitin on C-kielessä ominaisuus, jolla voidaan esitellä muuttuja, johon voidaan alustaa esitellyn funktion osoitin. Tästä lisää kappaleessa 5.2.3 Funktio-osoitin. Seuraavaksi käydään läpi valikkorakenteen toteutukseen vaadittavat ohjelmakomponentit eli aikaisemmin kuvatun vuokaavion komponentit.

5.2.1 HID-laite

Kuten aiemmin on työssä käynyt ilmi, ohjataan HID-laitteella isäntälaitetta kahden painokytkimen avulla. Kaksi kytkintä asettaa omalta osalta haasteen ohjelmakoodille: miten ohjata valikkoa vain kahdella kytkimellä. Ohjain olisi voitu suunnitella ristiohjain tyylisesti neljällä kytkimellä, mutta päädyttiin selkeyden vuoksi vain kahteen. Vasemmalla kytkimellä voidaan valikkoa selata ylöspäin ja oikealla alaspäin. Kytkimille luotiin ohjelmakoodissa kaksi eri tilaa kytkintä kohden. Kun oikean puoleista kytkintä painetaan ja painallus kestää alle puolen sekuntia, niin asetetaan kytkimelle tila ”kytkin alas” (KEY_DOWN). Jos painallus kestää yli puolen sekuntia, niin asetetaan sille toinen tila, joka oikeanpuoleisen kytkimen tapauksessa on ”kytkin suorita” (KEY_ENTER). Vasemman puoleisella kytkimellä on sama perustila ja toisiotila, perustilassa liikutaan valikkoa ylös ja pitkällä painaluksella taaksepäin (KEY_BACK).

Jotta kytkimet voitaisiin ottaa käyttöön, täytyy portti, johon ne ovat liitetty, alustaa. Kyseinen portti on portti-D. Tämä tapahtuu asettamalla portin suuntarekisteri (DDRx) osoittamaan sisäänpäin. Seuraavaksi alustetaan portin datarekisteri ja tähän asetettu tila riippuu tilarekisterin arvosta. Koska tässä tapauksessa tila on sisäänpäin, kirjoitetaan datarekisteriin arvo, jolla asetetaan kaikki muut pinnit paitsi ne, johon kytkimet ovat kytketty, ylösvetovastuksilla käyttöjännitteeseen. Tämän lisäksi, koska HID-laite toimii keskeytysperiaatteella, täytyy oikea keskeytys (INT1) ja globaalit keskeytykset ottaa käyttöön. Täytyy muistaa keskeytyksien kohdalla huomioida myös se suoritetaanko keskeytys pulssin nousevalla vai laskevalla reunalla.

```
void initialize_interrupt()
{
    GICR = (1<<INT1);                   // Enabloidaan INT1
    MCUCR = (1<<ISC11) | (1<<ISC10);    // Enabloidaan keskeytykset nousevalla reunalla
    sei();                              // Enabloidaan keskeytykset globaalisti
}

void initialize_port_states(void)
{
    DDRA = 0xFF;          // A-portin tila ulospäin, LCD-näyttö.
    PORTA = 0x00;

    DDRD = 0x00;          // D-portti sisäänpäin. PD6 & PD7 ovat nappeja.
    PORTD = 0x37;         // Sisäiset ylösvetovastukset muille kuin pinneille 7, 6 ja 4.
}
```

Kuvio 18. Porttien ja keskeytyksien alustukset.

Yllä olevasta kuvasta huomataan miten porttien alustukset tapahtuvat. Ylempänä myös alustetaan keskeytykset käyttöön kahdesta eri rekisteristä ja ”sei” -makro asettaa päälle globaalit keskeytykset yleisestä tilarekisteristä (SREG). Tilan muuttaminen voidaan alustaa kyseisiin rekistereihin suoraan päälle, koska rekisterien perustilat ovat ”kaikki bitit pois päältä”.

Kuten jo aiemmin kävi ilmi, perustuu ohjelmarakenne eri tiloihin ja kytkimille luotiin oma tilamuuttuja BUTTON_STATE, jota tilakone tarkkailee. Tätä tilamuuttujaa päivitetään keskeytysvektorilla INT1_vect. Kun kytkintä painetaan, siirtyy ohjelman suoritus kyseiseen keskeytysvektoriin ja suorittaa siellä olevan ohjelmapätkän.

```
void check_buttons(void)
{
    // Tutkitaan Input-porttia D.
    switch (PIND) {
        case BUTTON_7_STATE:
            // Katsotaan painettiinko nappia pitkään
            check_button_secondary_action(NULL); // Argumentilla NULL nollataan laskuri
            if (!BUTTON_STATE) {
                BUTTON_STATE=KEY_DOWN; // Oletusarvo jos ei pitkää painallusta.
            }
            break;

    // Samat kuin yllä, mutta toiselle napille.
        case BUTTON_6_STATE:
            check_button_secondary_action(NULL);
            if (!BUTTON_STATE) {
                BUTTON_STATE=KEY_UP;
            }
            break;
    }
}
```

Kuvio 19. Keskeytysvektorista kutsuttava kytkimien funktio.

Yllä olevasta ohjelmanpätkästä huomataan kytkinten käsittely. Koska portit ovat input –tyyppisiä, tarkkaillaan niiden sen hetkistä tilaa PINx rekisteristä, tässä tapauksessa PIND. Switch-lauseella käydään läpi portin eri tilat, joissa toinen on vasemman puoleinen kytkin (BUTTON_6) ja toinen oikeanpuoleinen (BUTTON_7). Toissijainen painallus tarkistetaan check_button_secondary_action funktiolla ja, jos pitkää painallusta ei tapahdu, alustetaan napille perustila. Tarkastellaan seuraavaksi vain yhden napin käsittelyä toissijaisen painalluksen funktiossa.

```
void check_button_secondary_action(int time)
{
    // Tutkitaan onko nappi painettuna pitkään
    // ja aktivoidaan toinen toiminto tarvittaessa.
    switch (PIND) {
        case BUTTON_7_STATE:
            _delay_ms(BUTTON_DELAY);
            time++;
            if (time < BUTTON_DELAY_FACTOR) {
                // Jos nappi on vielä painettua ja ei olla odotettu 0,5 sekuntia.
                // Kutsutaan tätä funktiota uudelleen inkrementoidulla muuttujan time arvolla.
                check_button_secondary_action(time);
            } else {
                // Jatketaan normaalisti. Perustila BUTTON_7 alaspäin.
                BUTTON_STATE=KEY_ENTER;
            }
            break;
    }
}
```

Kuvio 20. Keskeytysvektorista kutsuttava kytkimien toissijainen funktio.

Jotta toissijaiselle painallukselle voitaisiin tehdä tarkistus, täytyy sen olla huomaamaton käyttäjälle, minkä vuoksi sille ei voitu tehdä aina puolen sekunnin viivettä. Muuten valikon käyttö olisi hidasta. Sen sijaan, päädyttiin tekemään tarkistukselle perusviive, joka on noin yhden millisekunnin verran. Joka kerta viiveen jälkeen kasvatetaan aikamuuttujaa yhdellä, jota verrataan viivekertoimeen, joka on tässä tapauksessa 500. Viivekerroin kerrotaan yhden millisekunnin viiveellä, josta saadaan napin painalluksen pituus, joka tässä tapauksessa on 500 millisekuntia eli puolen sekuntia. Tämä tarkistus on rekursiivinen eli, jos inkrementoitu muuttuja on alle viivekertoimen, kutsuu funktio itseään uudelleen inkrementoidulla muuttujalla niin monta kertaa kuin se on alle 500 ja kun mennään yli, alustetaan BUTTON_STATE muuttujalle arvo KEY_ENTER. Jos portin tila muuttuu kesken kaiken eli kytkimestä päästetään irti, poistutaan funktiosta. Näin saadaan yhdelle kytkimen painallukselle minimissään yhden millisekunnin viive, jota ihminen ei käytössä huomaa. Keskeytysvektori määritellään pääohjelmassa, josta kutsutaan check_buttons funktiota. Alla on esitelty pääohjelmassa määritelty keskeytysvektorin funktio.

```
ISR(INT1_vect)
{
    check_buttons();
}
```

Kuvio 21. Keskeytysvektorista kutsuttava kytkimien funktio.

Kun määritellään muuttuja, jota käsitellään keskeytysvektorissa ja itse ohjelmassa, täytyy muistaa ottaa huomioon muuttujan volatile –määre. Muuten C-kielen kääntäjä

voi pahimmassa tapauksessa optimoida muuttujan arvon siten, että sitä ei koskaan päivitetä ja keskeytyksessä käytetty tilamuuttuja ei toimi. Volatile –määre kertoo kääntäjälle, että muuttujan arvo voi muuttua muistakin lähteistä kuin niistä, joita koodissa ollaan määritelty ja täten estetään sen liiallinen optimointi esimerkiksi silmukoissa.

5.2.2 Tilakone

Aikaisemmissa kappaleissa käytiin läpi valikon rakenne ja kytkinten päivittämä tilamuuttuja, jota tilakone käyttää. Kytkimen tilamuuttujan perusteella päivittää tilakone valikon tilamuuttujan arvoa (CURRENT_STATE) ja tilakone suoritetaan aina, kun kytkintä painetaan eli BUTTON_STATE muuttuja muuttaa arvoa. Tilakone itsessään on hyvin yksinkertainen, se hakee määritellystä valikon tilarakenteesta tilan arvon, joka vastaa sen hetkistä tilaa ja samalla se vertailee onko haetulla tilalla samoja kytkimien tiloja, mitä sen hetkinen kytkimen tilamuuttuja pitää sisällään.

```
void refresh_state(void)
{
    unsigned int i=0;
    char next_state_is_func=FALSE;

    // Jos globaali tila on alustettu.
    for (i=0; menu_nextstate[i].state_name;i++) {
        // Jos listalta haettu tila ja kytkimen arvo ovat samat kuin nykyiset.
        if (menu_nextstate[i].state_name == CURRENT_STATE &&
            menu_nextstate[i].state_button == BUTTON_STATE) {

            // Haetaan seuraavan tilan tieto onko se funktiokutsu.
            next_state_is_func=state_has_function(menu_nextstate[i].next_state);

            // Tutkitaan onko seuraava tila tyhjä (NULL) tai, että se ei ole funktio kutsu.
            if (menu_nextstate[i].next_state && !next_state_is_func) {
                // Jos ei ole, niin vaihdetaan tilaa.
                CURRENT_STATE = menu_nextstate[i].next_state;
            } else {
                // Jos oli, niin tarkastetaan oliko seuraava tila funktio.
                if (next_state_is_func) {
                    // Otetaan funktio-osoitin talteen.
                    STATE_FUNCTION = menu_nextstate[i].next_state;
                }
            }
            // Poistutaan tilakoneesta.
            break;
        }
    }
    /*
     * Täytyy aina muistaa asettaa nappien tila nollaksi.
     */
    BUTTON_STATE = NULL;
}
```

Kuvio 22. Tilakone.

Kuten yllä mainittiin, käy kuvan tilakone menu_nextstate listaa läpi niin kauan, kunnes se löytää nykyisen tilan, jossa ollaan. Tässä käytetään hyväksi taulukon viimeistä NULL –riviä ja silmukan ehtona voi olla ”tee niin kauan, kun tila ei ole NULL”. Tämä mahdollistaa mielivaltaisen pitkän valikon tekemisen ilman, että silmukoiden arvoja tarvitsee muutella jälkikäteen. Sitten vertaillaan kyseisestä tilasta sen hetkisiä kytkimen arvoja. Jos löytyy sopiva arvo, menee se tilan käsittelyyn. Ensimmäiseksi tilakone hakee tiedon talteen, onko seuraava tila funktiokutsu vai ei ja sitten tutkitaan onko seuraava tila tyhjä eli NULL ja että se ei ole funktio. Jos ehdot täyttyvät vaihtaa tilakone tilaa asettamalla menu_nextstate listan next_state muuttujan arvon nykyiseksi tilaksi. Jos ehdot eivät täyty, poistuu tilakone muuttamatta tilaa. Vain jos seuraava tila on funktio, otetaan funktio-osoitin talteen ja se suoritetaan tilakoneen jälkeen. Kohdassa, jossa tilakone tarkistelee onko seuraava suoritettava tila funktio, perustuu aikaisemmin todettuun rakenteeseen, että funktiolla ei voi olla tilatekstiä ja päinvastoin.

```
char state_has_function(unsigned char cstate)
{
    unsigned int i=0;

    for(i=0; menu_state[i].state; i++) {
        // Katsotaan onko tilalla valikkotekstiä.
        // Funktioilla ei ole tekstiä, niin voidaan olettaa sen olevan NULL.
        if (menu_state[i].state == cstate && menu_state[i].p_state_text == NULL) {
            return TRUE;
        }
    }
    return FALSE;
}
```

Kuvio 23. Tilakoneen funktiotilan tarkastelu.

Kuten kuvasta käy ilmi, käydään menu_state muuttujan tilat läpi verraten niitä annettuun tilaan ja jos sen tilan teksti on tyhjä eli NULL voidaan olettaa, että tila on funktio. Koska aikaisemmin esitellyn menu_state muuttujan rakenne on määritelty siten, että funktiolla ei voi olla tekstiä.

5.2.3 Funktio-osoitin

Aiemmassa kappaleessa puhuttiin tilakoneesta ja siitä, että sen jälkeen suoritetaan mahdollisesti löydetyt tilan funktiot, jotka asetetaan talteen STATE_FUNCTION -

muuttujaan. Tämä muuttuja on oikeastaan osoitin funktioon ja tilakoneen menu_state taulun kohdassa määritellään vain ajettavan funktion nimi. Koska C-kielessä funktio voidaan esitellä ilman sen vaatimia argumentteja, osoittaa tämä kyseinen alkio samannimiseen funktioon. Tämä ominaisuus osaltaan mahdollistaa valikkotyylisen dynaamisen rakenteen, jossa yksi samanniminen muuttuja voi kutsua erinimisiä funktioita, koska muuten pitäisi aina tietää missä tilassa ollaan ja mikä funktio suoritetaan sen perusteella. Tästä vain syntyisi iso switch-lauseke. Nyt voidaan vain suorittaa se arvo, joka sillä hetkellä on funktiotilamuuttujassa. (Kernighan & Richie 1988)

```
void run_function(unsigned char nstate)
{
    int i=0;

    for (i=0; menu_state[i].state; i++) {
        if (menu_state[i].state == nstate) {
            // Käytetään tässä C-kielen funktio-osoitinta!
            menu_state[i].p_state_func(TRUE);
            break;
        }
    }
    // Täytyy aina muistaa nollata tilat!
    STATE_FUNCTION = NULL;
}
```

Kuvio 24. Funktio-osoittimen suorittaminen.

Kuten kuvasta huomataan, kutsutaan sen hetkistä tilaa kuin funktiota eli annetaan sille argumentti. Tässä tapauksessa TRUE, jota käytetään valikkofunktioissa indikaattorina siitä, että ollaan tultu tilakoneesta. Jotta tällainen kutsu toimisi, täytyy muuttuja määritellä funktio-osoitin tyyppiseksi. Kertauksen vuoksi, tarkastellaan MENU_STATE_C tietotyypin rakennetta.

```
typedef struct {
    unsigned char state;
    char *p_state_text;
    char (*p_state_func)(char input);
} MENU_STATE_C;
```

Kuvio 25. MENU_STATE_C tietotyypin määrittely.

Kuten kuvasta huomataan, joudutaan funktio-osoittimen argumentit ja palautusarvot ennalta määrittelemään rakenteeseen. Tämä puolestaan aiheuttaa käytetyille funktioille yhden rajoituksen, joka on se, että niiden esittelyt täytyy olla identtiset. Työssä tästä ei aiheutunut ongelmia ja yhden tavun argumentit sekä palautusarvot olivat riittäviä funktioiden kanssa kommunikointiin.

5.2.4 LCD puskuri

Tilakoneen ja funktio-osoittimien suorittamisen lisäksi päätettiin näytölle piirrettävät tekstit suorittaa omassa aliohjelmassa eli LCD-puskurissa. Puskuri suoritetaan aina vain tilan muutoksen jälkeen, jottei näytölle tehtäisi turhia kirjoituksia. Puskurin suunnittelu oli helppoa, kun tiedettiin valikon rakenne ja tilakone –tyylinen ajattelu. Näytölle kirjoittaminen perustuu nykyiseen tilan arvoon. Kaikessa yksinkertaisuudessaan voitaisiin näytölle päivittää sen hetkisen tilan teksti, joka riittäisi indikoimaan, missä tilassa ollaan, mutta päätettiin, että valikkomaisen rakenteen saavuttamiseksi piirrettäisiin myös edellinen ja seuraava tila. Tähän ajattelumalliin perustuu koko puskurin rakenne. Työssä käytetty LCD-näytön ohjainkirjasto on Peter Fleyrun kirjoittama.

```
void refresh_lcd_buffer(void)
{
    char* prev_text=get_previous_state_text(CURRENT_STATE);
    char* curr_text=get_menu_state_text(CURRENT_STATE);
    char* next_text=get_next_state_text(CURRENT_STATE);

    // Otsikko
    lcd_clrscr();
    lcd_gotoxy(3,0);
    lcd_puts(MT_TITLE);

    // Ylempi teksti
    if (prev_text) {
        lcd_gotoxy(2,1);
        lcd_puts(prev_text);
    } else {
        lcd_gotoxy(2,1);
        lcd_puts(MT_EMPTY);
    }

    // Nykyinen valinta.
    if (curr_text) {
        lcd_gotoxy(0,2);
        lcd_putc('*');
        lcd_gotoxy(2,2);
        lcd_puts(curr_text);
    }

    // Alempi teksti
    if (next_text) {
        lcd_gotoxy(2,3);
        lcd_puts(next_text);
    } else {
        lcd_gotoxy(2,3);
        lcd_puts(MT_EMPTY);
    }
}
```

Kuvio 26. LCD-puskurin rakenne.

Kuten kuvasta saattaa huomata, on rakenne hyvin suoraviivainen. Aluksi haetaan vain nykyiseen tilaan perustuvat valikkotekstit, jotka sitten piirretään järjestyksessä näytölle. Ensiksi otsikko, sitten edellinen tila ja sen jälkeen nykyinen tila, jonka eteen piirretään valintaindikaattori (* -merkki) ja viimeiseksi seuraava tila. Koko piirtäminen voidaan perustaa tähän yksinkertaiseen rakenteeseen, koska piirtäminen on ehdollinen eli jos tila löytyy, niin piirretään se ruudulle. Aikaisemmin kävimme läpi valikkorakenteen muuttujan, jossa indikoitiin aina seuraava tila NULL –tyyppiseksi, jos se oli valikon viimeinen, joten sen tapauksessa se jätettäisiin piirtämättä. Tällä saadaan aikaiseksi illuusio, että liikutaan valikossa. Otetaan esimerkkinä vielä yhden tilan tekstin hakeminen ohjelmallisesti.

```
char* get_previous_state_text(unsigned char cstate)
{
    unsigned int i=0;
    unsigned char tmp_state=NULL;

    // Haetaan nykyisen tilan perusteella edellinen tila.
    // Jos edellinen tila oli NULL, niin poistutaan.
    for (i=0; menu_nextstate[i].state_name; i++) {
        if (menu_nextstate[i].state_name == cstate &&
            menu_nextstate[i].state_button == KEY_UP) {
            if (menu_nextstate[i].next_state) {
                tmp_state=menu_nextstate[i].next_state;
            } else {
                return NULL;
            }
            break;
        }
    }
    // Sitten haetaan edelliselle tilalle teksti.
    return get_menu_state_text(tmp_state);
}
```

Kuvio 27. Edellisen tilan tekstin hakeminen.

Kuten kuvasta huomataan, käydään edellisen tilatekstin kohdalla hakemassa nykyinen tila, jonka kytkimen arvo on KEY_UP eli ylöspäin ja otetaan talteen siitä tilasta seuraava tila next_state. Jos tällaista tilaa ei löydy eli se on valikossa NULL, niin palautetaan NULL. Muuten haetaan funktion lopussa sille tilalle teksti. Tarkastellaan vielä tekstin hakeminen.

```
char* get_menu_state_text(unsigned char cstate)
{
    unsigned int i=0;

    for(i=0; menu_state[i].state; i++) {
        if (menu_state[i].state == cstate) {
            return menu_state[i].p_state_text;
        }
    }
    return NULL;
}
```

Kuvio 28. Tilan tekstin hakeminen.

Kuten huomataan käy funktio valikkotilamuuttujaa läpi etsien annettua tilaa. Jos sellainen löytyy, palautetaan sen tilan tekstin arvo. Koska aiemmin jo todettiin, että tilakoneen tila ei voi olla funktio, niin voidaan olettaa, että löydetty merkkijono on aina valikkoteksti.

Kuvio 29. 3D-ristinollapelin valikko.

5.2.5 Tasojen tarkastelu

Kytkimen painalluksen rekisteröinnin, tilakoneen kellottamisen, funktio-osoittimen suorittamisen ja LCD-puskurin kirjoittamisen lisäksi isäntälaitteen viimeinen pääsilmukan osa-alue on tasojen tarkastelu. Tätä tarkastelua (pollaamista) suoritetaan pääsilmukassa kokoajan ehdolla ”onko orjalaite rekisteröinyt painalluksia”. Kysely suoritetaan TWI-väylän kautta, josta puhutaan lisää kohdassa 5.3 I²C kommunikointi.

Kun orjalaitteen tila on muuttunut, luetaan siltä tasolta kaikki sen hetkiset rivit GAME_ARRAY tilataulukkoon. Tämä kaksiulotteinen taulukko sisältää sen hetkisen tason eli z-ulottuvuuden ja x-ulottuvuuden sekä bittivektorin tiedon. Bittivektoria käsitellään y-ulottuvuutena ja tästä kerrotaan lisää kohdassa 5.4 3D algoritmi. Kun tason tilan tiedot on luettu talteen, tyhjennetään orjalaitteen ”uusia painalluksia” tilan tieto lähettämällä sille TWI-väylän kautta sanoma. Sitten vaihdetaan pelaajan vuoro ja tehdään sama myös orjalaitteelle lähettämällä sille sanoma. Kun vuoronvaihto on onnistunut, suoritetaan 3D-ratkaisualgoritmi. Jos algoritmi löytää mahdollisen voittorivin, palauttaa se voittajan arvon, jota indikoidaan arvoilla 1 tai 2 eli punainen tai vihreä. Tämän jälkeen ilmoitetaan näytöllä voittaja ja suoritetaan LCD-puskurin refresh –funktio. Lopuksi vielä nollataan kaikki tarvittavat tilamuuttujien tiedot ja orjalaitteet. Alla on vielä listaus selitetystä ohjelman kulusta.

```
char check_for_layer_state(char layer_address)
{
    int layer_number=0, winner=0;

    // GAME_ARRAY taulukon tasojen alkio-osoitteet.
    switch (layer_address) {
        case LAYER_1:
            layer_number=0;
            break;
        case LAYER_2:
            layer_number=1;
            break;
        case LAYER_3:
            layer_number=2;
            break;
        case LAYER_4:
            layer_number=3;
            break;
        default:
            // Väärä osoite.
            return NULL;
    }

    // Tutkitaan onko tasolla tapahtunut painalluksia.
    if (poll_layer(layer_address)) {
        // Jos on, niin luetaan kaikki tason rivit.
        GAME_ARRAY[layer_number][0]=read_line_vector(0,layer_address);
        GAME_ARRAY[layer_number][1]=read_line_vector(1,layer_address);
        GAME_ARRAY[layer_number][2]=read_line_vector(2,layer_address);
        GAME_ARRAY[layer_number][3]=read_line_vector(3,layer_address);

        reset_slave_state(layer_address); // Resetoidaan "uusia painalluksia" -lipun arvo.
        PLAYER_TURN ^= 0x03;              // Vaihdetaan vuoro. XOR 11 ^ 01 == 10
        change_slave_turn(layer_address,PLAYER_TURN);// Vaihdetaan orjalaitteen pelaajan vuoro.

        // Tutkitaan löytyykö voittorivejä.
        winner = check_for_winning_rows();
        if (winner) {
            switch (winner) {
                case RED:
                    announce_winner(RED);
                    break;
                case GREEN:
                    announce_winner(GREEN);
                    break;
            }
            // Päivitetään näyttö perustilaan ilmoituksen jälkeen.
            refresh_lcd_buffer();
            PLAYER_TURN=GREEN;      // Oletus aloitus pelaaja
            reset_all_layers();     // Sitten resetoidaan tasot
            return TRUE;
        }
    }
    return NULL;
}
```

Kuvio 30. Orjalaitteen tilan ja mahdollisten voittorivien tarkastelu tasokohtaisesti.

5.2.6 Valikon muut valinnat

Aikaisemmin käytiin läpi esimerkkinä valikon rakennetta kohdassa 5.2 Laitteen tilat, jossa kävi ilmi kolme tilaa: Uusi peli, Nopeimmat pelit ja Asetukset. Näiden lisäksi valikon Asetukset alla on alivalikko, josta löytyy tilat Kielivalinta, Tallenna asetukset ja Resetoi kaikki.

Uusi peli –valinta on yksinkertainen, se vain vaihtaa oletusvuoron vihreälle pelaajalle, tyhjentää GAME_ARRAY muuttujan alkion tiedot ja lopuksi lähettää kaikille tasoille resetointikäskyn. Tämän jälkeen LCD-puskurissa sijaitsevaa apufunktiota kutsutaan indikoimaan näytölle suoritetun valikkofunktion ”OK!” teksti. Alla on listaus ohjelman kulusta.

```
char new_game(char game)
{
    // Resetoidaan vain jos tullaan tilakoneesta.
    if (game==TRUE) {
        int i=0,j=0;
        // Oletus aloitus pelaaja
        PLAYER_TURN=GREEN;
        // Nollataan kaikki tasot
        reset_all_layers();
        // Tyhjennetään pelitaulukko
        for (i=0; i<4; i++) {
            for (j=0; j<4; j++) {
                GAME_ARRAY[i][j]=0;
            }
        }
        // Indikoidaan valinta
        choice_ok();
    }
    return NULL;
}
```

Kuvio 31. Valikon Uusi peli -funktio.

Nopeimmat pelit –valinta tulostaa näytölle kolme nopeinta pelattua peliä nousevassa järjestyksessä. Nämä tiedot alustetaan FASTEST_GAMES taulukon kolmeen ensimmäiseen alkioon. Tässä taulukossa on neljä alkiota, joista kolmea alkiota käytetään tulostamiseen ja viimeistä edellisen pelin tiedon tallentamiseen. Täten saadaan käytettyä helposti vain yhtä taulukkoa pelien seuraamiseen. Koska taulukko on pieni, vain neljä alkiota, järjestetään se kuplalajittelualgoritmilla nousevaan järjestykseen, joten se voidaan sellaisenaan aina tulostaa näytölle olettaen, että sen sisältö on aina oikeassa järjestyksessä. Alla on listaus taulukon tulostamisesta.

```c
void draw_high_scores(void)
{
    char s[6] = { 0 };
    lcd_clrscr();

    // Otsikko
    lcd_clrscr();
    lcd_gotoxy(3,0);
    lcd_puts(MT_HIGH_SCORES);

    // 1. sija
    lcd_gotoxy(0,1);
    lcd_puts("1. ");
    lcd_gotoxy(3,1);
    if (FASTEST_GAMES[0] == MAX) {
        strcpy(s, MT_EMPTY);
    } else {
        itoa((FASTEST_GAMES[0]-TRUE),s,10);
    }
    lcd_puts(s);

    // 2. sija
    lcd_gotoxy(0,2);
    lcd_puts("2. ");
    lcd_gotoxy(3,2);
    if (FASTEST_GAMES[1] == MAX) {
        strcpy(s, MT_EMPTY);
    } else {
        itoa((FASTEST_GAMES[1]-TRUE),s,10);
    }
    lcd_puts(s);

    // 3. sija
    lcd_gotoxy(0,3);
    lcd_puts("3. ");
    lcd_gotoxy(3,3);
    if (FASTEST_GAMES[2] == MAX) {
        strcpy(s, MT_EMPTY);
    } else {
        itoa((FASTEST_GAMES[2]-TRUE),s,10);
    }
    lcd_puts(s);

    // Ajan yksikkö oikeaan alakulmaan.
    lcd_gotoxy(17,3);
    lcd_puts("[t]");

    // Jäädään valikkoon kunnes painetaan nappia
    BUTTON_STATE=NULL;
    while (!BUTTON_STATE);
}
```

Kuvio 32. Nopein peli valinnan puskurin piirtämisfunktio.

Kuten kuvasta huomataan, tulostetaan FASTEST_GAMES taulukon alkiot järjestyksessä nollasta kahteen sillä ehdolla, että taulukon alkion koko ei ole alkualustusarvo. Alkualustusarvon tilalle tulostetaan tyhjä merkkijono. Jotta taulukon alkion kokonaisluku saataisiin näytölle tulostettavaan muotoon, muunnetaan se C-kielen itoa –standardifunktiolla merkkijonoksi. Oikeassa alakulmassa näytetään ajan yksikkö eli arvot ovat sekunteja.

Koska käytetty peliaika voi olla pidempi kuin neljä minuuttia ja 15 sekuntia eli yli yhden tavun kokoisen muuttujan, niin päätettiin FASTEST_GAMES taulukko alustaa 16 bittiseksi kokonaisluku tietotyypiksi eli se voisi sisältää numerot 0 - 65535. Jotta kuplalajittelualgoritmi olisi toimiva, jouduttiin FASTEST_GAMES taulukko alustamaan oletusarvoilla 0xFFFF eli jokainen solu sisältäisi arvon 65535, näitä arvoja ei tulosteta näytölle. Täten saataisiin aina pienin peliaika lajiteltua ensimmäiseen alkioon, koska C-kielen kääntäjä alustaa taulukon arvoilla -1, jos sitä ei erikseen määritellä. Alla on listaus kuplalajittelualgoritmista.

```
void sort_save_high_scores(uint16_t gametime)
{
    uint8_t i=0, j=0;
    uint16_t swap=0;

    // Järjestä taulukko. Bubble sort.
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            if (FASTEST_GAMES[j] > FASTEST_GAMES[j+1]) {
                // Blockataan alustus atomisesti, koska 16bit int alustukset ovat
                // 2 tavuisia ja ne alustetaan kahdessa osassa.
                ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
                    swap = FASTEST_GAMES[j];
                    FASTEST_GAMES[j] = FASTEST_GAMES[j+1];
                    FASTEST_GAMES[j+1] = swap;
                }
            }
        }
    }
}
```

Kuvio 33. Kuplalajittelualgoritmi.

Nopeimmat pelit –ajastin käynnistetään aina, kun rekisteröidään painallus ja pysäytetään, kun valitaan uusi peli tai jompikumpi pelaaja voittaa. Tähän käytetään mikro-ohjaimen sisäänrakennettua 16-bittistä ajastinta Timer1, joka on itsenäinen yksikkö mikro-ohjaimessa. Koska käytetty kellotaajuus oli 14,7456 MHz, jouduttiin ajastimen esijakaja alustamaan arvolla 256. Näin käytetty taajuus saatiin jaettua tarpeeksi pieneksi, jotta sitä voitiin käyttää yhden sekunnin laskemisessa. Ajastinta ajettiin CTC (Clear Timer on Compare match) –tilassa eli ajastin aina nollaa itsensä (TCNT1 Timer/Counter rekisteri) ja alkaa alusta, kun se saavuttaa OCR1A (Output Compare Register) rekisteriin asetetun arvon. OCR1A rekisterin arvo lasketaan mikro-ohjaimessa käytetystä kellolähteen nopeudesta. Aluksi halutaan tietää kuinka kauan yksi kellosykli kestää. Koska taajuuden yksikkö on yksi per sekunti, voidaan kellosyklin pituus laskea yksinkertaisesti kaavasta:

$$Timer1_{res.} = \frac{1}{F_CPU_{freq.}}$$

Tämän perusteella saadaan yhden syklin pituudeksi 0,0000000678 sekuntia. Tämän jälkeen voidaan tarvittava laskurin arvo, jotta se saavuttaisi halutun viiveen, laskea kaavasta:

$$Timer_{count} = \left(\frac{Vaadittu\ viive}{Timer1_{res.}}\right) - 1$$

Kaavasta huomataan heti, että taajuutta on pakko jakaa pienemmäksi, jotta laskurin arvo mahtuisi 16-bittiseen rekisterimuuttujaan. Kuten aikaisemmin mainittiin, jouduttiin käyttämään taajuuden esijakajaa, jotta ajastimen kellotaajuutta saataisiin pienemmäksi. Tämä jakajan alustus tehdään rekisterin TCCR1B (Timer/Counter1 Control Register B) kolmeen alimpaan bittiin. Nämä bitit täytyy myös alustaa, jotta ajastin lähtisi päälle. Mahdolliset esijakajan arvot ovat: 1 (ei jaeta), 8, 64, 256 ja 1024. Näistä voidaankin muodostaa taulukko, josta voidaan tutkia mikä olisi hyvä ajastimen arvo.

Taulukko 1. Esijakajan taajuudet ja niiden laskurin arvot.

Esijakaja	Taajuus	Laskurin arvo
1	14,7456 MHz	14745599
8	1,8432 MHz	1843199
64	230,4 kHz	230399
256	57,6 kHz	57599
1024	14,4 kHz	14399

Koska haluttu viive oli yhden sekunnin verran, tulee laskurin arvoksi aikaisemman kaavan perusteella: käytetty taajuus miinus yksi. Kuten taulukosta huomataan, jakautuu 14,7456 MHz aina kokonaisluvuksi, tämä ei aina pidä paikkaansa eripituisilla halutuilla viiveillä tai kellolähteen nopeuksilla. Tämä oli myös yksi syy kyseisen kellokiteen nopeuden valintaan. Koska taulukon kaksi viimeistä esijakajan arvoa mah-

tuvat hyvin 16-bittiseen kokonaislukumuuttujaan, voidaan käyttää jompaakumpaa arvoa. (Atmel ATmega32 datalehti 2016.)

```
void initialize_timer(void)
{
    TCCR1B |= (1 << WGM12); // Asetetaan ajastin CTC tilaan.
    TIMSK |= (1 << OCIE1A); // Asetetaan ajastimen keskeytys.
    OCR1A = 57600; // Ajastimen laskurin maksimi arvo, jonka jälkeen se nollaa itsensä.
    TCCR1B |= (1 << CS12); // Asetetaan esijakajan arvoksi 256.
}
```

Kuvio 34. CTC ajastimen alustus.

Kuvassa alustetaan TIMSK (Timer/Counter Interrupt Mask) rekisteristä keskeytys päälle ja alustuksen jälkeen ajastin käy aina sekunnin välein TIMER1_COMPA_vect keskeytysvektorissa ja nolla itsensä eli TCNT1 rekisterin arvon. Nopeimmat pelit – laskennassa keskeytysvektoriin asetettiin PLAYTIME_TIMER muuttuja, jota aina inkrementoidaan, kun peli alkaa ja nollataan sen päättyessä.

Asetukset –valinnan alta löytyy kolme tilaa. Ensimmäisestä tilasta voidaan vaihtaa käyttöliittymän kieli, toisesta voidaan tallentaa nykyiset asetukset EEPROM-muistiin ja viimeinen valikon valinta on Resetoi kaikki. Asetusten tallennus kirjoittaa EEPROM-muistiin kielivalinnan ja nopeimmat pelit. Resetoi kaikki -valinnalla voidaan tyhjentää kaikki muistiasetukset (kieli ja nopeimmat pelit) sekä lähettää orjalaitteille fyysinen nollaus vetämällä niiden $\overline{\text{RESET}}$-linja nollapotentiaaliin, joka on kytketty isäntälaitteen C-portin pinniin. Tämä ominaisuus lisättiin suunnitteluvaiheessa, jotta saataisiin helposti nollattua orjalaitteet isäntälaitteesta käsin ongelmatilanteissa.

Käyttöliittymän teksti tulostetaan tilakoneen tilataulukon muuttujan arvoista ja sen hetkiselle kielen tilalle on myös alustettu tilamuuttuja MENU_LANGUAGE. Esimerkiksi otsikon tekstissä on käytetty merkkijonomuuttujaa MT_TITLE. Kielen vaihto perustuu ideaan, jossa kaikkiin merkkijonomuuttujiin kirjoitetaan uusi teksti eri kielellä. Kun valitaan valikosta kieli, vaihtuu se järjestyksessä suomen kielestä ruotsin kieleen ja viimeiseksi englannin kieleen. Oletuskielenä on englanti ja isäntälaitteen käynnistyksen yhteydessä haetaan EEPROM muistista tallennettu kieli, jos sellainen löytyy. Ohessa on esimerkkinä suomen kielen tilatekstimuuttujien alustus.

```
void set_language(char lang)
{
    switch (lang) {
        case FINNISH:
            // Yleiset valikkotekstit
            strcpy(MT_TITLE,          "3D-ristinolla   ");
            strcpy(MT_NEW_GAME,       "Uusi peli       ");
            strcpy(MT_HIGH_SCORES,    "Nopeimmat pelit ");
            strcpy(MT_SETTINGS,       "Asetukset       ");
            strcpy(MT_SETTINGS_LANG,  "Kielivalinta    ");
            strcpy(MT_SETTINGS_RESET, "Resetoi kaikki  ");
            strcpy(MT_SETTINGS_SAVE,  "Tallenna asetu. ");
            // Valikon valinnan tilan muutostekstit.
            strcpy(MT_CHOICE_FAIL,    "Jo tallennettu  ");
        break;
    }
}
```

Kuvio 35. Esimerkkinä kielen vaihto suomeksi.

Kuten kuvasta huomataan, tekstit ovat kaikki yhtä pitkiä ja merkkijonojen loppuun on täytteeksi laitettu välilyöntejä. Tämän avulla vältetään LCD-näytön vilkkuminen, kun sitä ei joka kerta tarvitse tyhjentää lcd_clrscr –funktiolla, vaan vanha teksti yksinkertaisesti ylikirjoitetaan näytön oikeaan reunaan asti. Koska merkkijonot vievät muistista paljon tilaa ja sitä on mikro-ohjaimissa rajattu määrä, niin olisi hyvä käytäntö kirjoittaa kaikki merkkijonot Flash-muistiin ja lukea ne suoraan sieltä. Tässä työssä ei kuitenkaan nähty sitä tarpeelliseksi, koska ajoaikaisesta käyttömuistista (SRAM) ei käytetty kuin vähän yli puolet.

Tallenna –valinnasta voidaan tallentaa sen hetkisen kielen tilan ja nopeimmat pelit tilan tieto EEPROM-muistiin, josta ne ladataan takaisin muistiin mikro-ohjaimen käynnistyksen yhteydessä. Koska AVR mikro-ohjaimissa EEPROM-muisti sijaitsee täysin eri muistialueella kuin datamuisti, täytyy sinne kirjoittaessa käyttää AVR-libc –kirjaston omia funktioita. Funktiot ovat myös täysin asynkronisia, koska EEPROM kirjoitusoperaatio hoidetaan erillisellä piirillä mikro-ohjaimen sisällä. Tästä syystä melkein kaikki funktiot, joita voidaan käyttää EEPROM-muistin käsittelyssä, eivät palauta mitään arvoja. Lukuun ottamatta tietenkin funktioita, joilla luetaan EEPROM-muistista tietoa. Lukutapahtuma on hyvin yksinkertainen ja se voidaan hoitaa yhdellä funktiokutsulla, kuten myös kirjoittaminen. Funktiolle annetaan vain se tavun kokoinen arvo, joka EEPROM-muistiin halutaan kirjoittaa ja muistipaikan kohdeosoite. AVR mikro-ohjaimissa EEPROM-muistia on usein kilotavun verran, joten muistiosoitealue on silloin 0 – 1024. EEPROM-kirjastosta löytyy myös funktio, joka päivittää tiedon muistiin eli se ei kirjoita mitään, jos nykyinen arvo on sama,

jota yritetään kirjoittaa. Tätä funktiota kannattaa käyttää usein, koska yleensä EEPROM-muisteilla on rajattu kirjoitusmäärä ja ne eivät sisällä minkäänlaisia kulumisen tasoitus algoritmeja. AVR mikro-ohjaimissa EEPROM-muistin maksimi kirjoitus- tai tyhjennyssyklit ovat yleensä noin 100 tuhatta. Alla on esimerkki kielenasetuksen lukemisesta EEPROM-muistista käynnistyksen yhteydessä.

```
// Luetaan viimeksi käytetty kieli EEPROM:sta
uint8_t eeplang=eeprom_read_byte((uint8_t*)EEP_LANG_ADDR);
// Jos luetaan roskaa, niin ei käytetä sitä.
if (eeplang==SWEDISH || eeplang==ENGLISH || eeplang==FINNISH) {
    MENU_LANGUAGE=eeplang;
} else {
    // Oletuskieli
    MENU_LANGUAGE=ENGLISH;
}
```

Kuvio 36. Esimerkki kielen lukemisesta EEPROM-muistista.

Kuten kuvasta huomataan, haetaan kielen tilan tieto muistista yhdellä funktiolla, joka palauttaa tavun kokoisen arvon. Funktiolle annettu muistipaikan osoite määriteltiin itse. Työssä käytetyt osoitteet ovat kaikki tavun kokoisia, joten muita EEPROM-funktioita ei käytetä, kuin yhden tavun luku-, kirjoitus- ja päivitysfunktioita. Muita funktion kokoja ovat sana (16 bit), tuplasana (32 bit) ja lohko (itse määritelty koko). Kirjoitusoperaatio on hyvin samantyyppinen kuin lukuoperaatio, mutta osoitteen lisäksi kirjoitusfunktiolle annetaan kirjoitettava arvo. Alla on esimerkki kielen tilan tallennuksesta.

```c
char save_settings_lang(void)
{
    // Nykyinen talteen
    uint8_t eeplang=eeprom_read_byte((uint8_t*)EEP_LANG_ADDR);

    // Päivitetään kieli EEPROM:n vain tarvittaessa
    // Eli jos nykyinen on jo sama, niin ei kirjoiteta turhaan.
    switch (MENU_LANGUAGE) {
        case FINNISH:
            eeprom_update_byte((uint8_t*)EEP_LANG_ADDR, FINNISH);
            break;
        case SWEDISH:
            eeprom_update_byte((uint8_t*)EEP_LANG_ADDR, SWEDISH);
            break;
        case ENGLISH:
            eeprom_update_byte((uint8_t*)EEP_LANG_ADDR, ENGLISH);
            break;
    }

    // Luetaan arvo takaisin ja verrataan ennen aikaisempaan,
    // jonka perusteella indikoidaan käyttäjälle.
    if (eeprom_read_byte((uint8_t*)EEP_LANG_ADDR) != eeplang) {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

Kuvio 37. Esimerkki kielen tallentamisesta EEPROM-muistiin.

Kuvan funktiossa aluksi luetaan nykyisen tallennetun kielen tieto sen vuoksi, että voidaan sen perusteella indikoida funktion lopussa onnistuiko päivitys, koska eeprom_update_byte funktio ei palauta mitään arvoja. Sen hetkisen kielen tila tallennetaan muistiin eeprom_update_byte funktiolla, joka kirjoittaa tiedon muistiin vain, jos kirjoitettava arvo on eri kuin muistissa oleva. Nopeimmat pelit taulukon sisältö kirjoitetaan EEPROM-muistiin täysin samalla tavalla miten kielivalinta, mutta se kirjoitetaan silmukassa. Joka kerta tarkastellaan eeprom_is_ready makrolla, onko edellinen kirjoitusoperaatio vielä kesken, koska EEPROM-muistiin kirjoittaminen voi kestää jopa kahdeksan millisekuntia.

Resetoi kaikki –valinnalla voidaan tyhjentää yllä mainitut tiedot EEPROM-muistista. Tämän lisäksi valinta asettaa C-portin kolmannen pinnin tilan nollapotentiaaliin, mikä vetää orjalaitteiden $\overline{\text{RESET}}$-linjan alas ja pakottaa näiden uudelleenkäynnistymisen. Alla on esimerkki kielen tilan poistamisesta EEPROM-muistista.

```c
char reset_all_lang(void)
{
    // Kirjoitetaan kielen muistialue tyhjäksi jos se ei jo ole tyhjä.
    eeprom_update_byte((uint8_t*)EEP_LANG_ADDR, EEP_EMPTY);

    // Varmistetaan että se tyhjentyi ja indikoidaan asiasta.
    if (eeprom_read_byte((uint8_t*)EEP_LANG_ADDR) == EEP_EMPTY) {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

Kuvio 38. Esimerkki kielen poistamisesta EEPROM-muistista.

Kuvan funktiossa hoidetaan tiedon tyhjennys kirjoittamalla kielen muistiosoitteeseen (EEP_LANG_ADDR) arvon 0xFF. Tämä vastaa EEPROM-muistin tyhjää perustilaa. Samankaltaista tyhjennysfunktiota käytetään myös Nopeimmat pelit –tietojen poistamiseen. Samalla uudelleenkäynnistetään orjalaitteet alla olevalla ohjelman listauksessa määritellyllä layers_hard_reset funktiolla.

```c
void layers_hard_reset(void)
{
    // Ajetaan reset linja alas
    layer_reset_enable();
    // Reset linjaa täytyy pitää alhalla vähintään 1,5 mikrosekuntia
    _delay_ms(1);
    // Ja takaisin ylös.
    layer_reset_disable();
}

void layer_reset_enable(void)
{
    // Nollataan C-portin kolmas pinni
    PORTC &= ~(1<<PC2);
}
void layer_reset_disable(void)
{
    // Asetetaan C-portin kolmas pinni
    PORTC |= (1<<PC2);
}
```

Kuvio 39. Orjalaitteiden resetointifunktiot.

5.3 I²C kommunikointi

Työssä käytettiin isäntä- ja orjalaitteiden välillä I²C-väylää kommunikointiin. Väylän funktiot olivat ensimmäisiä ohjelmanpätkiä, jotka valmistuivat testejä tehdessä ja näiden toiminnallisuus ja rakenne sovittiin etukäteen. Näitä samoja funktioita ja kirjastoja käytettiin sekä orja- että isäntälaitteessa sellaisenaan. Testivaiheessa huomattiin myös, että orjalaitteella ei ole missään vaiheessa tarvetta toimia väylällä master –

tilassa, joten isäntälaite on ainoa, joka käskyttää väylän laitteita. Tämän pohjalta sovittiin myös protokolla eli mitä sanomia orjalaitteen ohjauksessa käytettäisiin. Alla on kuvailtu protokollan sanomat taulukossa.

Taulukko 2. Protokollan sanomat.

Sanoma	Sanoman kuvaus
0x01	Vaihdetaan pelivuoro punaiselle pelaajalle.
0x02	Vaihdetaan pelivuoro vihreälle pelaajalle.
0x03	Pelitaulukko alustetaan ja pelivuoro asetetaan nollaksi.
0x7F	Palautetaan ”uusia painalluksia” –tilan arvo isäntälaitteelle.
0x80	Palautetaan pelitaulukon 1. rivi.
0x81	Palautetaan pelitaulukon 2. rivi.
0x82	Palautetaan pelitaulukon 3. rivi.
0x83	Palautetaan pelitaulukon 4. rivi.
0x84	Resetoidaan “uusia painalluksia” –tilan tieto.

Kappaleissa 3. ja 4. käytiin jo tarkemmin läpi I²C-väylän protokollan rakenne, joten nyt keskitymme väylän ohjelmalliseen toiminnallisuuteen. Väylän alustus tehdään jo heti ohjelman alussa muiden alustuksien lomassa ja työssä siihen ei käytetä kuin kahta rekisteriä, jotka kummatkin osaltaan vaikuttavat väylän nopeuteen. Nämä ovat TWSR ja TWBR eli TWI Status rekisteri ja TWI Bit Rate rekisteri. TWSR rekisterin kahteen alimpaan bittiin asetetaan tarvittaessa esijakajan arvo, joka vaikuttaa SCL-linjan nopeuteen, jos esimerkiksi mikro-ohjaimen kiteen taajuus on liian korkea. Huomaa myös, että mikro-ohjaimen kellokiteen taajuuden on oltava vähintään 16 kertainen SCL-linjan taajuuteen verrattuna. TWBR rekisterissä alustetaan taajuuden jakajan kerroin. Väylän nopeuden laskukaava on seuraavanlainen:

$$SCL_{freq} = \frac{Kellokiteen\ taajuus}{16 + 2(TWBR) \cdot 4^{TWPS}}$$

Kaavassa TWPS rekisteri on esijakajan bittien arvo TWSR rekisterissä ja nämä voidaan jättää laskusta pois, joten kaava voidaan laskea muotoon:

$$TWBR = \frac{\left(\frac{Kellokiteen\ taajuus}{SCL_{freq}} - 16\right)}{2}$$

Tästä saadaan TWBR rekisteriin oikea jakajan kerroin, josta mikro-ohjain osaa laskea oikean taajuuden mikä työssä on noin 50 kHz. Alla on listaus väylän nopeuden alustuksesta. (Atmel ATmega32 datalehti 2016.)

```
void initialize_twi(void)
{
    TWSR = 0x00; // Ei käytetä prescaleria. Ei ole tarvetta jakaa taajuutta pienemmäksi.
    TWBR = ((F_CPU/SCL_CLOCK)-16)/2; // TWI SCL taajuudeen jakajan kerroin.
}
```

Kuvio 40. TWI-väylän nopeuden alustus.

Koska TWI-väylän protokolla on sisäänrakennettu mikro-ohjaimeen, tapahtuu sen toiminnallisuuden ohjaus rekisterien kautta. Yksi tärkeä rekisteri, jonka viimeistä bittiä käsitellään aina, kun TWI-väylällä tapahtuu muutos ja se odottaa uutta syötettä, on TWCR eli TWI Control rekisteri. TWINT eli TWI interrupt flag –lippu asetetaan aina päälle kyseisessä rekisterissä, kun väylä havaitsee aloitusehtoja tai on lopettanut suorittamansa toiminnon. Jos globaalit keskeytykset ja ohjausrekisterin bitti TWIE on asetettu, luo väylän muutokset aina keskeytyksen ja ohjelma siirtyy suorittamaan TWI-väylän keskeytysvektoria. Tätä toiminnallisuutta ei isäntälaitteessa tarvittu, koska se toimi aina väylän isäntälaitteena. Tässä samassa ohjausrekisterissä on TWI-väylän START ja STOP ehtojen bitit, joista puhuttiin kappaleessa 3.2.1. Toinen tärkeä rekisteri, josta voidaan tarkistaa väylälle lähetettyjen ehtojen jälkeen sen tila, on TWSR. Tämä rekisteri mainittiin jo aikaisemmin ja sen kaksi alinta bittiä asettaa esijakajan arvon, mutta tässä tapauksessa rekisteristä käytetään viittä ylintä bittiä, jotka kertovat väylällä tapahtuneen ehdon jälkeisen tilan. Nämä tilat löytyvät mikro-ohjaimen datalehdestä. (Atmel ATmega32 datalehti 2016.)

Isäntälaitteen funktiot, joilla lähetetään käskyt orjalaitteelle, koostuivat ennalta määritellyistä vaadittavan tapahtuman nimisistä funktioista. Esimerkiksi vuoron vaihto funktiossa (change_slave_turn) kutsutaan ensiksi i2c_start funktiota, jonka jälkeen kirjoitetaan arvo i2c_write funktiolla ja lopuksi i2c_stop funktiolla lopetetaan sanoman lähetys. Kaikki työn orjalaitteen kommunikoinnissa käytetyt funktiot ovat yhdistelmiä näistä perusfunktioista. Funktio i2c_start yksinkertaistettuna lähettää väy-

lälle START ehdon asettamalla TWCR rekisterin bitit TWI interrupt, TWI Start ja TWI Enable päälle. Sitten odotetaan kunnes lähetys onnistuu eli TWI interrupt vaihtaa tilaa, jonka jälkeen väylälle tehdään tilan tarkistus makrolla, joka maskataan arvolla 0xF8. Maskin tulosta verrataan TW_START vakioon eli saatiinko orjalaitteelta kuittaus. Se tarkoittaa, että kaikki on kunnossa ja jos ei ole, niin poistutaan. Sitten lähetetään laitteen osoite ja asetetaan samat TWI interrupt ja TWI Enable bitit. Tämän jälkeen odotetaan, että lähetys onnistui ja sen jälkeen tarkistetaan taas maskilla 0xF8, saatiinko kuittaus. Jos kaikki on kunnossa, palautetaan onnistunut arvo. Alla on i2c_start ohjelman listaus. (Atmel ATmega32 datalehti 2016.)

```
char i2c_start(unsigned char address, unsigned char dir)
{
    uint8_t   twst;

    // Lähetetään START ehto.
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // Odotettaan kunnes lähetys onnistui.
    while (!(TWCR & (1<<TWINT)));

    // Tarkistetaan TWI Status rekisteri, oliko START onnistunut.
    // Eli tarkistetaan ACK bitti.
    twst = TW_STATUS & 0xF8;
    if ((twst != TW_START) && (twst != TW_REP_START))
        return 1;

    // Lähetetään laitteen osoite.
    TWDR = address*2 + dir;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // Odotellaan kunnes saadaan kuittaus. ACK bitti.
    while (!(TWCR & (1<<TWINT)));

    // Verrataan taas Status rekisteriä, oliko ACK onnistunut.
    twst = TW_STATUS & 0xF8;
    if ((twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK))
        return 1;

    return 0;
}
```

Kuvio 41. TWI-väylän kirjoitustapahtuman aloitus.

Funktio i2c_write on samantyylinen, mutta yksinkertaisempi kuin i2c_start. Siinä aloitetaan laittamalla lähetettävän datan arvo TWDR eli TWI Data rekisteriin ja sitten tarkkaillaan ohjausrekisterin TWI interrupt ja Enable bittejä sekä odotellaan, että lähetys onnistui. Viimeiseksi tarkistetaan maskilla 0xF8 tila, että vastaanotettiinko ACK bitti, jonka jälkeen poistutaan funktiosta. Alla on listaus i2c_write funktiosta.

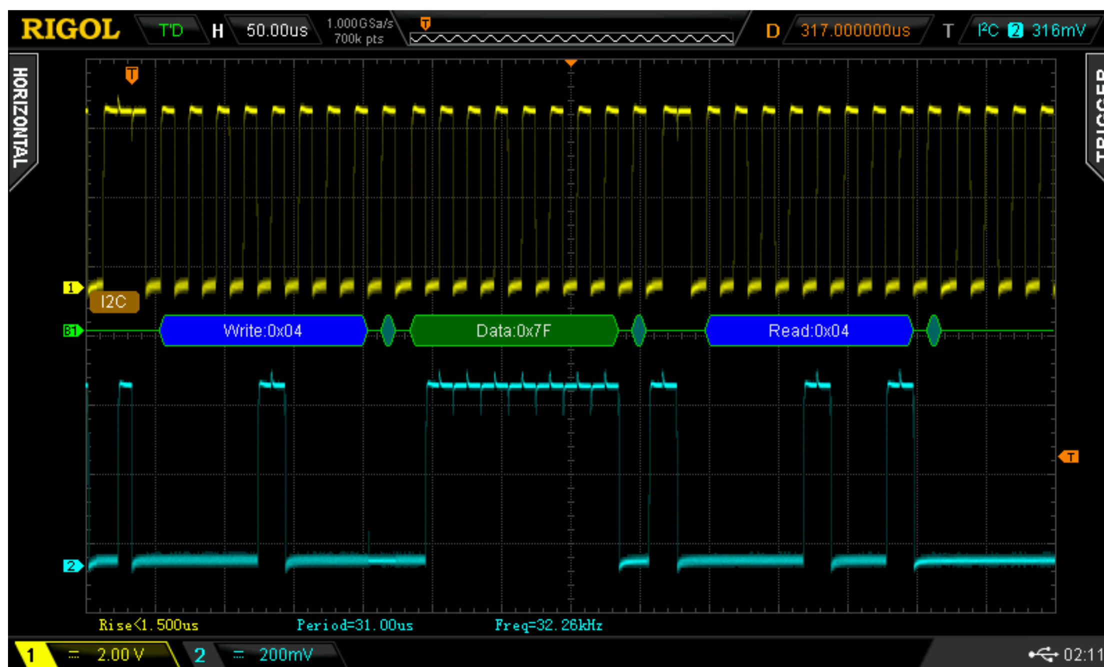
```
unsigned char i2c_write(unsigned char data)
{
    uint8_t   twst;

    // Asetetaan lähetettävä data
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);

    // Odotellaan, että lähetys onnistui.
    while (!(TWCR & (1<<TWINT)));

    // Ja lopuksi katsotaan tila, että lähetys oli onnistunut.
    twst = TW_STATUS & 0xF8;
    if (twst != TW_MT_DATA_ACK)
        return 1;

    return 0;
}
```

Kuvio 42. TWI-väylän kirjoitustapahtuman aloitus.

Viimeinen kutsuttava funktio on i2c_stop, jolla lopetetaan sanoma lähettämällä STOP ehto väylälle. Tässä funktiossa ei tehdä muuta kuin kirjoitetaan ohjausrekisteriin tarvittavat TWI interrupt ja Enable bitit sekä STOP bitti. Sitten odotetaan, että lähetys onnistuu. Alla on listaus funktiosta.

```
void i2c_stop(void)
{
    // Alustetaan STOP ehto.
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    // Odotetaan, kunnes väylä vapautuu.
    while (TWCR & (1<<TWSTO));
}
```

Kuvio 43. TWI-väylän lopetustapahtuma.



Kuvio 44. TWI-väylän 0x7F sanoman haku (poll_layer –funktiolla).

5.4 3D algoritmi

Tässä kappaleessa käydään läpi käytetyt ratkaisualgoritmit tasojen voittoriveille. Perusperiaate voittorivin löytämiselle oli jo alusta alkaen, että käydään läpi kaikki mahdolliset voittotilat, eikä yritetä etsiä tai soveltaa matemaattisia ratkaisuja voittotilojen etsimiselle.

Aikaisemmin työssä käytiin orjalaitteen rakenne läpi, jossa todettiin, että ledejä ohjataan siirtorekistereillä. Näiden siirtorekisterien tiloja kuvataan ohjelmakoodissa yhden tavun tiloina eli esimerkiksi bittijono 01 01 01 01 (jossa bittien tila 01 on punainen ja tila 10 vihreä) vastaa heksadesimaalilukuna arvoa 0x55, joka on punainen voittorivi. Kun taas vastaavasti bittijono 10 10 10 10 vastaa arvoa 0xAA ja on vihreä voittorivi. Tämä yhden tavun arvo vastaa siis yhtä riviä. Näitä siirtorekistereitä on orjalaitteessa neljä kappaletta eli yksi per rivi. Joten tämä rakenne pakotti ajattelemaan kolmiulotteisen taulukon yhtä ulottuvuutta bittivektorina, joka tässä tapauksessa voidaan ajatella kuution y-ulottuvuutena ja sen hetkisen taulukon alkioiden arvona. Bittivektorin takia luotiin voittorivitaulukosta siis kaksiulotteinen, joten neljä siirtorekisteriä voidaan ajatella x-ulottuvuutena eli toisena taulukon alkioista. Toiseen

taulukon alkioon alustetaan jäljelle jäänyt z-ulottuvuus eli pelitasot. Tarkastellaan seuraavaksi ratkaisutaulukon ohjelmallista rakennetta.

```
/*
 * 3D-taulun rakenne!
 *
 * GAME_ARRAY [tasot] [x_akseli] = y-suunnan bittivektori
 *
 * Tutkitaan ja ratkaistaan kuutio yhdestä suunnasta!
 * Eli katsotaan sitä aina edestäpäin.
 *     _________
 *   /        /|
 *  /        y |
 * /________/  |
 * |        |  |
 * |        z  |
 * |        | /
 * |___x____|/
 */

unsigned char GAME_ARRAY[4][4]={{ NULL }};
```

Kuvio 45. 3D ratkaisutaulukon rakenne ja sen alustus.

Kuten kuvasta huomataan, alustetaan ratkaisutaulukko moniulotteiseksi taulukoksi, jossa kolmas neljän alkion ulottuvuus on bittivektori. Kuvan tapauksessa kaikki bittivektorit asetetaan alkuarvoon NULL. Huomaa kommenteissa mainittu oletustila, jota voittorivin hakemisessa käytetään, on että katsotaan kuutiota aina edestäpäin.

Voittorivit kategorioitiin seuraaviin ratkaisutiloihin ilman z-ulottuvuutta:

- Tasokohtaisesti yksi pystyrivi kerrallaan y-suunnassa.
- Tasokohtaisesti yksi vaakarivi kerrallaan x-suunnassa.
- Tasokohtaisesti kulmasta kulmaan eli ristisuunnassa.

Seuraavaksi voittorivit ratkaistiin z-ulottuvuudessa (katsotaan kuutiota edestäpäin):

- Ristisuuntaan vasemmalta oikealle.
- Ristisuuntaan edestä taakse.
- Ristisuunta kulmasta kulmaan kuution läpi kahdesta suunnasta.
- Tasojen väliset pystyrivit.

Bittivektorin eri ledeille luotiin bittimaskit. Vektorit maskattiin näillä tiloilla JA-operaation kanssa, jonka seurauksena saatiin aina sen vektorin yhden ruudukon ledin tila (punainen tai vihreä) selville. Alla on bittimaskien määrittelyt.

```
/*
 * Yhden rivin ledien bittimaskit
 *
 * Maskit_|__Ruudukot___
 * 0xC0   |  11 00 00 00
 * 0x30   |  00 11 00 00
 * 0x0C   |  00 00 11 00
 * 0x03   |  00 00 00 11
 *
 */
#define LED_1  0xC0
#define LED_2  0X30
#define LED_3  0x0C
#define LED_4  0x03
```

Kuvio 46. Bittivektorien maskit.

Kuvasta huomataankin heti bittivektorien välinen ratkaisutapa. Jos halutaan x-suunnassa ratkaista voittorivi, niin inkrementoidaan ratkaisutaulukon x-muuttujaa ja maskataan aina sen ledin ruutu, jonka vaakarivi halutaan ratkaista. Jotta ehtolauseista ei tulisi liian isoja verrattaessa jokaista lediä erikseen punaiseen tai vihreään lediin (0x01 ja 0x02), voidaan luoda väliaikainen bittivektorimuuttuja. Tähän muuttujaan summataan jokainen maskattu arvo omille sijoilleen. Kuten kymmenkantajärjestelmässä voidaan pilkun paikkaa muuttaa jakamalla tai kertomalla kantaluku kymmenen potensseilla, voidaan sama tehdä binäärille jakamalla tai kertomalla se kahden potensseilla. Eli aina bittivektorin maskauksen jälkeen jaetaan tai kerrotaan tulos joko luvulla: 4, 16 tai 64. Näin siirretään maskattua bittiriviä aina kaksi sijaa vasemmalle tai oikealle (yhden ruudun verran). Tätä muuttujaa sitten verrataan lopuksi punaiseen tai vihreään voittoriviin (0x55 tai 0xAA) ja näin saadaan ehtolauseesta huomattavasti pienempi. Otetaan koodin esimerkki.

```
/*************************** vasen vaakarivi ***************************/
// Käydään jokainen taso yksi kerrallaan läpi riveittäin x-suunnassa.
// Eli ledin 1. rivi. Poistutaan ohjelmasta heti, jos löytyy voittoja.
for (layer=0; layer<4; layer++) {
    win_row_0=0;

    win_row_0+= (GAME_ARRAY[layer][0]&LED_1);
    win_row_0+=((GAME_ARRAY[layer][1]&LED_1)/4);
    win_row_0+=((GAME_ARRAY[layer][2]&LED_1)/16);
    win_row_0+=((GAME_ARRAY[layer][3]&LED_1)/64);

    if (win_row_0==GREEN_WIN_ROW) {
        return GREEN;
    }
    if (win_row_0==RED_WIN_ROW) {
        return RED;
    }
}
```

Kuvio 47. Vaakarivin ratkaisu x-suunnassa.

Kuten kuvasta huomataan, summataan win_row_0 muuttujaan bittimaskin läpi LED_1 rivi, joka jaetaan aina jokaisella eri x-ulottuvuuden kohdassa olevalla luvulla: 4, 16 tai 64. Näin saadaan kasattua mahdollinen voittorivi, jota sitten verrataan joko vihreään tai punaiseen voittoriviin. Y-suunnassa ratkaistava voittorivi on helpompi selvittää. Verrataan vain sen hetkistä bittivektoria joko punaisen tai vihreän voittorivin arvoon. Alla on listaus y-suunnan ratkaisusta tasoittain.

```
/***************************** pystyrivit ******************************/
// Käydään jokainen taso yksi kerrallaan läpi riveittäin y-suunnassa.
//  - joka riviltä tutkitaan bittivektorin y-suunnasta onko voittorivejä
for (layer=0; layer<4; layer++) {
    for (x_axis=0; x_axis<4; x_axis++) {
        if (GAME_ARRAY[layer][x_axis]==GREEN_WIN_ROW) {
            return GREEN;
        }
        if (GAME_ARRAY[layer][x_axis]==RED_WIN_ROW) {
            return RED;
        }
    }
}
```

Kuvio 48. Pystyrivin ratkaisu y-suunnassa.

Tutkitaan vielä yhtä ristisuunnassa olevan voittorivin ratkaisumallia.

```
/****** ristisuunta (vasemmalta oikealle) eri tasoilla *******/
for (x_axis=0; x_axis<4; x_axis++) {
    win_row_0=0,win_row_1=0;

    win_row_0+=(GAME_ARRAY[0][x_axis]&LED_1);
    win_row_0+=(GAME_ARRAY[1][x_axis]&LED_2);
    win_row_0+=(GAME_ARRAY[2][x_axis]&LED_3);
    win_row_0+=(GAME_ARRAY[3][x_axis]&LED_4);

    if (win_row_0==GREEN_WIN_ROW) {
        return GREEN;
    }
    if (win_row_0==RED_WIN_ROW) {
        return RED;
    }
}
```

Kuvio 49. Ristisuunnan ratkaisu z-ulottuvuudessa.

Kuten kuvasta huomataan, käydään jokaisella tasolla x-ulottuvuuden ristisuunta eli eri ledien maskit läpi. Samaa aikaisempaa periaatetta sovelletaan, jossa win_row_0 muuttujaan summataan eri ruudukoiden kohtaan vihreän tai punaisen ledin arvo. Tässä tapauksessa ei bittivektoria tarvitse jakaa tai kertoa, koska se tehdään jo maskaamalla eri ledien arvoilla. Tätä samaa voidaan soveltaa myös toiseen ristisuuntaan kääntämällä ledit tai tasot päinvastaiseen suuntaan.

Aikaisempaa ratkaisumallia, jossa jaettiin tai kerrottiin bittivektori maskin jälkeen tai edellisessä kohdassa käytyä ratkaisumallia jossa vain vaihdettiin eri ledien maskeja, voidaan myös soveltaa kaikissa muissa ratkaisusuuntien eri kohdissa. Kaikki käytetyt ratkaisualgoritmit löytyvät työn lopusta liitteenä.

6 JOHTOPÄÄTÖKSET

Työn alkuvaiheessa, kun sitä ensimmäisiä kertoja suunniteltiin ja sen jälkeen lähdettiin rakentamaan, oli työn vaiheet hyvin mielessä ja työ oli aika suoraviivaista. Alkuperäisestä suunnitelmasta ei poikettu juuri lainkaan niin raudan kuin ohjelmistonkaan osalta. Kuitenkin työn lopussa huomattiin aika montakin asiaa, joihin olisi voitu tehdä muutoksia.

Aluksi ehkä isoimpana ongelmana oli piirilevyjen jyrsiminen. Niiden kohdalla syntyi eniten ongelmia koko työn aikana ja kasausvaihe oli ongelmien takia myös todella työläs. Kaikki johteet piti mitata, kun ei voitu olla varmoja jyrsimisessä syntyneistä lastuista ja niiden oikosuluista. Tämä vaihe olisi pitänyt toteuttaa valottamalla ja syövyttämällä. Työn rakenne olisi voitu myös toteuttaa yksinkertaisemmin. Monen ledin juottaminen ja liittäminen osoittautui todella työlääksi ja rumaksi tavaksi indikoida käyttäjälle pelaajan tila. Esimerkiksi jokin läpinäkyvä segmentti-, pistematriisi tai tft-näyttö olisi ollut omiaan indikoimaan käyttäjälle jonkin muodon. Olisi myös päästy eroon siirtorekisteripuskureista ja tilalle olisi tullut näytön valmistajalta pieni ohjain, jotka yleensä ovat integroituja näyttöön. Tosin tällä ratkaisulla ei olisi saatu pidettyä kustannuksia kurissa ja lähdekoodin koko olisi kasvanut entisestään. Isäntälaitteen piirilevyn koko on sopiva prototyypiksi, mutta orjalaitteet olisi voitu ehkä suunnitella vähän pienemmiksi. Myös virtalähde olisi voitu toteuttaa hakkuriperiaatteella ja rakentaa se omalle piirilevylleen. Täten olisi saatu piirilevyjä vielä pienemmäksi.

Ohjelmiston puolella ehkä isoimpana puutteena pelissä oli tietokonevastustajan puuttuminen. Nyt peliä pystyy pelaamaan vain kaksinpelinä. Äly tälle tietokonevastustajalle olisi ehkä voitu saada statistiikan ja minimax-algoritmin kautta. Tämä kuitenkin olisi osoittautunut liian työlääksi tähän opinnäytetyöhön vaikka mikro-ohjaimessa olisi laskentateho siihen riittänyt. Orjalaitteiden ohjelmallinen resetointi isäntälaitteesta käsin osoittautui turhaksi, koska itse isäntälaitteeseen ei lisätty mitään watchdog tyylistä resetointia, jolloin jumiutumistilanteessa isäntälaite olisi resetoinut itsensä. Nyt pelistä joudutaan poistamaan virrat, mikä joka tapauksessa resetoi samalla myös orjalaitteet. Lopuksi ehkä olisi voitu suorittaa pieniä optimointeja lähdekoodissa, esimerkiksi voittorivien ratkaisualgoritmissa olisi voitu jakamisen sijaan käyttää

bittien ”shiftausta”, koska se yleensä on ALU laskentayksikössä paljon nopeampi operaatio kuten esimerkiksi jakaminen. Tämä olisi silti niin pieni muutos, että sillä ei loppupeleissä ole käytännön merkitystä. Viimeiseksi olisi ollut laitteen virrankulutuksen kannalta iso asia lisätä isäntä- sekä orjalaitteille virransäästötilat, nämä olisi ollut helppo toteuttaa ja Atmel suosittelee datalehdessä aina käyttämään kyseisiä tiloja jos mahdollista.

Lopuksi kuitenkin työn prototyyppiluonteen takia, todettiin, että siinä onnistuttiin oikein hyvin. Laite toimi hienosti työn joka vaiheessa. Tekeminen oli äärimmäisen opettavaa niin elektroniikan osaamisen ja ohjelmoinnin kannalta. Esimerkiksi tilakonemaisen ajattelun ansiosta on laitteen ohjelmistoa helppo laajentaa tarvittaessa ja niistä saatiin hyvät valmiit ohjelmakirjastot omaan jatkokäyttöön. Onnistumisen tunne, joka työstä saatiin, kannusti opettelemaan lisää sulautetuista järjestelmistä, joka työn tekijöillä oli opintojen ensisijainen valinta.

LÄHTEET

The Story of AVR 2008. Viitattu 17.1.2016.
https://www.youtube.com/watch?v=HrydNwAxbcY

I2C-Bus www-sivut. Viitattu 12.2.2016.
http://www.i2c-bus.org/

NXP Semiconductors I2C spesifikaation datalehti. Viitattu 12.2.2016.
http://www.nxp.com/documents/user_manual/UM10204.pdf

HD44780 LCD Starter guide. Viitattu 28.2.2016.
https://web.stanford.edu/class/ee281/handouts/lcd_tutorial.pdf

Wikipedia Hitachi HD44780 LCD Controller. Viitattu 28.2.2016.
https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

Wikipedia Serial Peripheral Interface Bus. Viitattu 23.3.2016.
https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Atmel In System Programming datalehti. Viitattu 23.3.2016.
http://www.atmel.com/images/doc0943.pdf

Maxembedded.com www-sivut. Viitattu 24.3.2016
http://maxembedded.com/2013/11/serial-peripheral-interface-spi-basics/

Steven Keeping. Understanding the Advantages and Disadvantages of Linear Regulators. Viitattu 28.4.2016.
https://www.digikey.com/en/articles/techzone/2012/may/understanding-the-advantages-and-disadvantages-of-linear-regulators

Bungard – CCD 2011. Viitattu 3.5.2016.
https://www.youtube.com/watch?v=C8QyN9Po0fI

Wikipedia EEPROM. Viitattu 24.5.2016.
https://en.wikipedia.org/wiki/EEPROM

Wikipedia Floating-gate MOSFET. Viitattu 24.5.2016.
https://en.wikipedia.org/wiki/Floating-gate_MOSFET

Atmel ATmega32 datalehti. Viitattu 30.5.2016.
http://www.atmel.com/images/doc2503.pdf atmega32

Kernighan, B. & Richie, D. 1988. The C Programming language Second Edition. Massachusetts: Prentice Hall P T R

Horowitz, P. & Hill, W. 2015. The Art of Electronics Third Edition. Glasgow: Cambridge University Press.

RATKAISUALGORITMIT

```c
/*
 * solve_3d.c
 *
 *  Created on: May, 2016
 *      Author: Pasi Vuorinen
 */

#include <util/delay.h>
#include "main.h"
#include "lcd.h"
#include "solve_3d.h"
#include "menu_structure.h"


/*
 * Tutkitaan mahdolliset voittorivit ja poistutaan heti palauttamalla voittajan väri.
 *
 * Punainen        0x01
 * Vihreä        0x02
 */
unsigned char check_for_winning_rows(void) {
    /*
     * 3D-taulun rakenne!
     *
     * GAME_ARRAY [tasot] [x_akseli] = y-suunnan bittivektori
     *
     * Tutkitaan ja ratkaistaan kuutio yhdestä suunnasta!
     * Eli katsotaan sitä aina edestäpäin.
     *    _________
     *   /        /|
     *  /        y |
     * /________/  |
     * |        |  |
     * |        z  |
     * |        | /
     * |___x____|/
     */

    unsigned char GAME_ARRAY[4][4]={{ NULL }};

    unsigned char win_row_0=0, win_row_1=0, win_row_2=0, win_row_3=0;
    unsigned char x_axis=0, layer=0;


    /****************************** pystyrivit ********************************/
    // Käydään jokainen taso yksi kerrallaan läpi riveittäin y-suunnassa.
    //  - joka riviltä tutkitaan bittivektorin y-suunnasta onko voittorivejä
    for (layer=0; layer<4; layer++) {
        for (x_axis=0; x_axis<4; x_axis++) {
            if (GAME_ARRAY[layer][x_axis]==GREEN_WIN_ROW) {
                return GREEN;
            }
            if (GAME_ARRAY[layer][x_axis]==RED_WIN_ROW) {
                return RED;
            }
        }
    }
```

```
/*************************** vasen vaakarivi ****************************/
    // Käydään jokainen taso yksi kerrallaan läpi riveittäin x-suunnassa 1. rivi.
    for (layer=0; layer<4; layer++) {
        win_row_0=0;

        win_row_0+= (GAME_ARRAY[layer][0]&LED_1);
        win_row_0+=((GAME_ARRAY[layer][1]&LED_1)/4);
        win_row_0+=((GAME_ARRAY[layer][2]&LED_1)/16);
        win_row_0+=((GAME_ARRAY[layer][3]&LED_1)/64);

        if (win_row_0==GREEN_WIN_ROW) {
            return GREEN;
        }
        if (win_row_0==RED_WIN_ROW) {
            return RED;
        }
    }
    /*************************** 2. vaakarivi ******************************/
    // Käydään jokainen taso yksi kerrallaan läpi riveittäin y-suunnassa 2. rivi.
    for (layer=0; layer<4; layer++) {
        win_row_0=0;

        win_row_0+=((GAME_ARRAY[layer][0]&LED_2)*4);
        win_row_0+=((GAME_ARRAY[layer][1]&LED_2));
        win_row_0+=((GAME_ARRAY[layer][2]&LED_2)/4);
        win_row_0+=((GAME_ARRAY[layer][3]&LED_2)/16);

        if (win_row_0==GREEN_WIN_ROW){
            return GREEN;
        }
        if (win_row_0==RED_WIN_ROW){
            return RED;
        }
    }
    /*************************** 3. vaakarivi ******************************/
    // Käydään jokainen taso yksi kerrallaan läpi riveittäin y-suunnassa 3. rivi.
    for (layer=0; layer<4; layer++) {
        win_row_0=0;

        win_row_0+=((GAME_ARRAY[layer][0]&LED_3)*16);
        win_row_0+=((GAME_ARRAY[layer][1]&LED_3)*4);
        win_row_0+= (GAME_ARRAY[layer][2]&LED_3);
        win_row_0+=((GAME_ARRAY[layer][3]&LED_3)/4);

        if (win_row_0==GREEN_WIN_ROW){
            return GREEN;
        }
        if (win_row_0==RED_WIN_ROW){
            return RED;
        }
    }
    /*************************** oikea vaakarivi ****************************/
    // Käydään jokainen taso yksi kerrallaan läpi riveittäin y-suunnassa 4. rivi.
    for (layer=0; layer<4; layer++) {
        win_row_0=0;

        win_row_0+=((GAME_ARRAY[layer][0]&LED_4)*64);
```

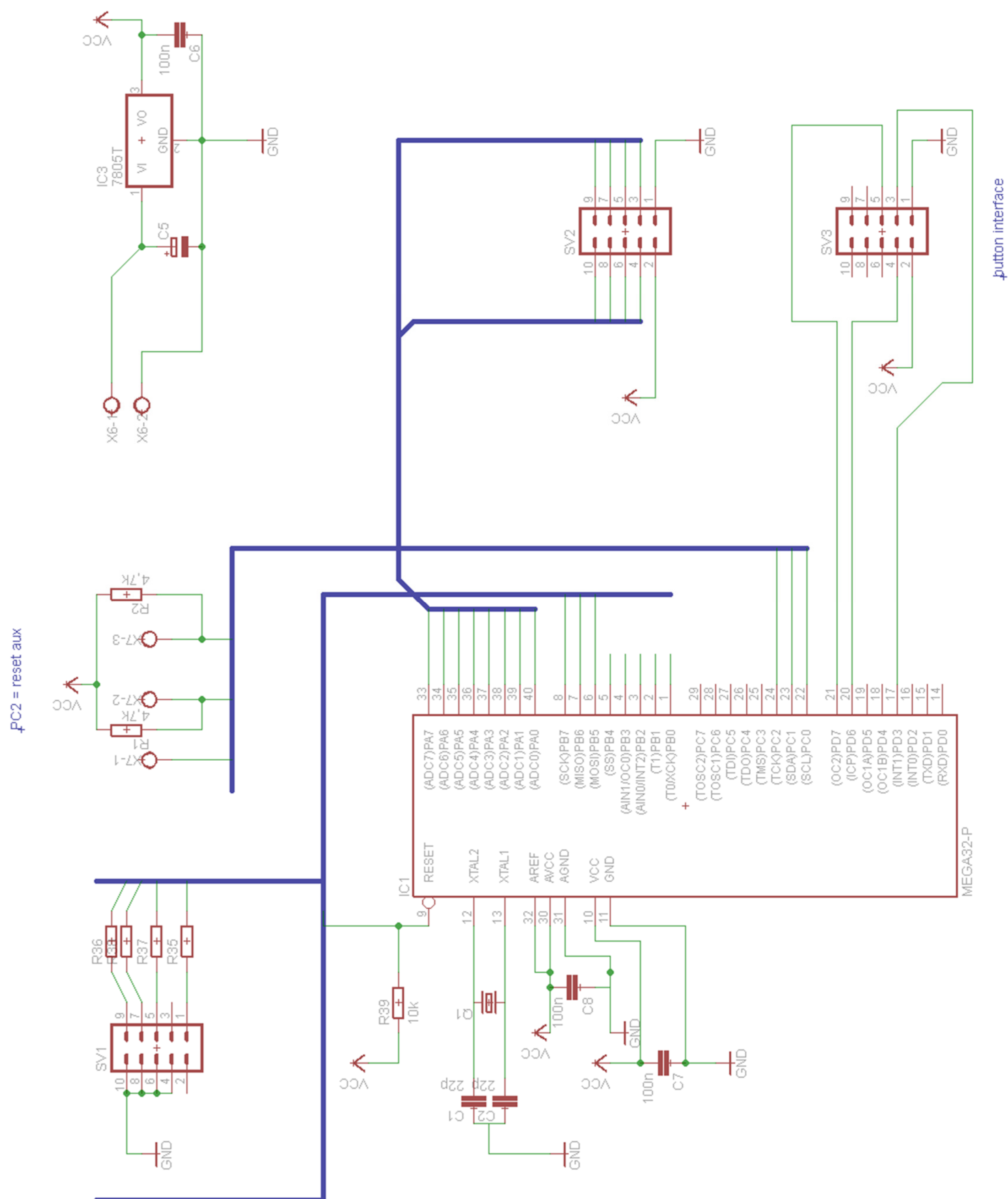
```
        win_row_0+=((GAME_ARRAY[layer][1]&LED_4)*16);
        win_row_0+=((GAME_ARRAY[layer][2]&LED_4)*4);
        win_row_0+= (GAME_ARRAY[layer][3]&LED_4);

        if (win_row_0==GREEN_WIN_ROW) {
            return GREEN;
        }
        if (win_row_0==RED_WIN_ROW) {
            return RED;
        }
    }


    /************** ristisuunta (ylhäältä alas) yhdellä tasolla **************/
    for (layer=0; layer<4; layer++) {
        win_row_0=0,win_row_1=0;

        win_row_0+=(GAME_ARRAY[layer][0]&LED_1); // toinen suunta
        win_row_0+=(GAME_ARRAY[layer][1]&LED_2);
        win_row_0+=(GAME_ARRAY[layer][2]&LED_3);
        win_row_0+=(GAME_ARRAY[layer][3]&LED_4);

        win_row_1+=(GAME_ARRAY[layer][3]&LED_1); // ja toinen
        win_row_1+=(GAME_ARRAY[layer][2]&LED_2);
        win_row_1+=(GAME_ARRAY[layer][1]&LED_3);
        win_row_1+=(GAME_ARRAY[layer][0]&LED_4);

        if ((win_row_0==GREEN_WIN_ROW)||(win_row_1==GREEN_WIN_ROW)) {
            return GREEN;
        }
        if ((win_row_0==RED_WIN_ROW)||(win_row_1==RED_WIN_ROW)) {
            return RED;
        }
    }
    /****** ristisuunta (X-tyylisesti vasemmalta oikealle) eri tasoilla *******/
    for (x_axis=0; x_axis<4; x_axis++) {
        win_row_0=0,win_row_1=0;

        win_row_0+=(GAME_ARRAY[0][x_axis]&LED_1);
        win_row_0+=(GAME_ARRAY[1][x_axis]&LED_2);
        win_row_0+=(GAME_ARRAY[2][x_axis]&LED_3);
        win_row_0+=(GAME_ARRAY[3][x_axis]&LED_4);

        win_row_1+=(GAME_ARRAY[3][x_axis]&LED_1);
        win_row_1+=(GAME_ARRAY[2][x_axis]&LED_2);
        win_row_1+=(GAME_ARRAY[1][x_axis]&LED_3);
        win_row_1+=(GAME_ARRAY[0][x_axis]&LED_4);

        if ((win_row_0==GREEN_WIN_ROW)||(win_row_1==GREEN_WIN_ROW)) {
            return GREEN;
        }
        if ((win_row_0==RED_WIN_ROW)||(win_row_1==RED_WIN_ROW)) {
            return RED;
        }
    }
```

```
/******** ristisuunta (X-tyylisesti edestä taakse) eri tasoilla ***********/
// Alustetaan seuraavaa kierrosta varten.
win_row_0=0,win_row_1=0,win_row_2=0,win_row_3=0;


win_row_0+=((GAME_ARRAY[0][0]&LED_1)/64);
win_row_0+=((GAME_ARRAY[1][1]&LED_1)/16);
win_row_0+=((GAME_ARRAY[2][2]&LED_1)/4);
win_row_0+=((GAME_ARRAY[3][3]&LED_1));

win_row_1+=((GAME_ARRAY[0][0]&LED_2)/16);
win_row_1+=((GAME_ARRAY[1][1]&LED_2)/4);
win_row_1+=((GAME_ARRAY[2][2]&LED_2));
win_row_1+=((GAME_ARRAY[3][3]&LED_2)*4);

win_row_2+=((GAME_ARRAY[0][0]&LED_3)/4);
win_row_2+=((GAME_ARRAY[1][1]&LED_3));
win_row_2+=((GAME_ARRAY[2][2]&LED_3)*4);
win_row_2+=((GAME_ARRAY[3][3]&LED_3)*16);

win_row_3+=((GAME_ARRAY[0][0]&LED_4));
win_row_3+=((GAME_ARRAY[1][1]&LED_4)*4);
win_row_3+=((GAME_ARRAY[2][2]&LED_4)*16);
win_row_3+=((GAME_ARRAY[3][3]&LED_4)*64);

// Voittotarkistus
if ((win_row_0==GREEN_WIN_ROW)||
    (win_row_1==GREEN_WIN_ROW)||
    (win_row_2==GREEN_WIN_ROW)||
    (win_row_3==GREEN_WIN_ROW)) {
    return GREEN;
}
if ((win_row_0==RED_WIN_ROW)||
    (win_row_1==RED_WIN_ROW)||
    (win_row_2==RED_WIN_ROW)||
    (win_row_3==RED_WIN_ROW)) {
    return RED;
}

// Alustetaan seuraavaa kierrosta varten.
win_row_0=0,win_row_1=0,win_row_2=0,win_row_3=0;


win_row_0+=((GAME_ARRAY[0][3]&LED_1)/64);
win_row_0+=((GAME_ARRAY[1][2]&LED_1)/16);
win_row_0+=((GAME_ARRAY[2][1]&LED_1)/4);
win_row_0+=((GAME_ARRAY[3][0]&LED_1));

win_row_1+=((GAME_ARRAY[0][3]&LED_2)/16);
win_row_1+=((GAME_ARRAY[1][2]&LED_2)/4);
win_row_1+=((GAME_ARRAY[2][1]&LED_2));
win_row_1+=((GAME_ARRAY[3][0]&LED_2)*4);

win_row_2+=((GAME_ARRAY[0][3]&LED_3)/4);
win_row_2+=((GAME_ARRAY[1][2]&LED_3));
win_row_2+=((GAME_ARRAY[2][1]&LED_3)*4);
win_row_2+=((GAME_ARRAY[3][0]&LED_3)*16);

win_row_3+=((GAME_ARRAY[0][3]&LED_4));
win_row_3+=((GAME_ARRAY[1][2]&LED_4)*4);
win_row_3+=((GAME_ARRAY[2][1]&LED_4)*16);
win_row_3+=((GAME_ARRAY[3][0]&LED_4)*64);
```

```
// Voittotarkistus
if ((win_row_0==GREEN_WIN_ROW)||
    (win_row_1==GREEN_WIN_ROW)||
    (win_row_2==GREEN_WIN_ROW)||
    (win_row_3==GREEN_WIN_ROW)) {
    return GREEN;
}
if ((win_row_0==RED_WIN_ROW)||
    (win_row_1==RED_WIN_ROW)||
    (win_row_2==RED_WIN_ROW)||
    (win_row_3==RED_WIN_ROW)) {
    return RED;
}


/*********** ristisuunta kolmannessa ulottuvuudessa eri tasoilla **********/
/********************* (vinoittan reunasta reunaan) **********************/
win_row_0=0,win_row_1=0;

win_row_0+=(GAME_ARRAY[0][0]&LED_1);
win_row_0+=(GAME_ARRAY[1][1]&LED_2);
win_row_0+=(GAME_ARRAY[2][2]&LED_3);
win_row_0+=(GAME_ARRAY[3][3]&LED_4);

win_row_1+=(GAME_ARRAY[3][0]&LED_1);
win_row_1+=(GAME_ARRAY[2][1]&LED_2);
win_row_1+=(GAME_ARRAY[1][2]&LED_3);
win_row_1+=(GAME_ARRAY[0][3]&LED_4);

if ((win_row_0==GREEN_WIN_ROW)||(win_row_1==GREEN_WIN_ROW)) {
    return GREEN;
}
if ((win_row_0==RED_WIN_ROW)||(win_row_1==RED_WIN_ROW)) {
    return RED;
}

// Toiseen suuntaan.
win_row_0=0,win_row_1=0;

win_row_0+=(GAME_ARRAY[0][3]&LED_1);
win_row_0+=(GAME_ARRAY[1][2]&LED_2);
win_row_0+=(GAME_ARRAY[2][1]&LED_3);
win_row_0+=(GAME_ARRAY[3][0]&LED_4);

win_row_1+=(GAME_ARRAY[0][0]&LED_4);
win_row_1+=(GAME_ARRAY[1][1]&LED_3);
win_row_1+=(GAME_ARRAY[2][2]&LED_2);
win_row_1+=(GAME_ARRAY[3][3]&LED_1);

if ((win_row_0==GREEN_WIN_ROW)||(win_row_1==GREEN_WIN_ROW)) {
    return GREEN;
}
if ((win_row_0==RED_WIN_ROW)||(win_row_1==RED_WIN_ROW)) {
    return RED;
}
```

```
    /*********** pystysuunta eritasoilla (y-johteen mukaisesti) ***************/
    // Alustetaan seuraavaa kierrosta varten.
    win_row_0=0,win_row_1=0,win_row_2=0,win_row_3=0;

    for (x_axis=0; x_axis<4; x_axis++) {
        win_row_0=0,win_row_1=0,win_row_3=0,win_row_3=0;
        win_row_0+=((GAME_ARRAY[0][x_axis])&LED_1)/64;
        win_row_0+=((GAME_ARRAY[1][x_axis])&LED_1)/16;
        win_row_0+=((GAME_ARRAY[2][x_axis])&LED_1)/4;
        win_row_0+=((GAME_ARRAY[3][x_axis])&LED_1);

        win_row_1+=((GAME_ARRAY[0][x_axis])&LED_2)/16;
        win_row_1+=((GAME_ARRAY[1][x_axis])&LED_2)/4;
        win_row_1+=((GAME_ARRAY[2][x_axis])&LED_2);
        win_row_1+=((GAME_ARRAY[3][x_axis])&LED_2)*4;

        win_row_2+=((GAME_ARRAY[0][x_axis])&LED_3)/4;
        win_row_2+=((GAME_ARRAY[1][x_axis])&LED_3);
        win_row_2+=((GAME_ARRAY[2][x_axis])&LED_3)*4;
        win_row_2+=((GAME_ARRAY[3][x_axis])&LED_3)*16;

        win_row_3+=((GAME_ARRAY[0][x_axis])&LED_4);
        win_row_3+=((GAME_ARRAY[1][x_axis])&LED_4)*4;
        win_row_3+=((GAME_ARRAY[2][x_axis])&LED_4)*16;
        win_row_3+=((GAME_ARRAY[3][x_axis])&LED_4)*64;

        // Voittotarkistus
        if ((win_row_0==GREEN_WIN_ROW)||
            (win_row_1==GREEN_WIN_ROW)||
            (win_row_2==GREEN_WIN_ROW)||
            (win_row_3==GREEN_WIN_ROW)) {
            return GREEN;
        }
        if ((win_row_0==RED_WIN_ROW)||
            (win_row_1==RED_WIN_ROW)||
            (win_row_2==RED_WIN_ROW)||
            (win_row_3==RED_WIN_ROW)) {
            return RED;
        }
    }

    return NULL;
}
```

KYTKENTÄKAAVIOT



SV1

R36

R37

R35

X7-1

X7-2

X7-3

GND

VCC

R39

10k

IC1

MEGA32-P

RESET

XTAL2

XTAL1

AREF

AVCC

AGND

VCC

GND

(ADC7)PA7

(ADC6)PA6

(ADC5)PA5

(ADC4)PA4

(ADC3)PA3

(ADC2)PA2

(ADC1)PA1

(ADC0)PA0

(SCK)PB7

(MISO)PB6

(MOSI)PB5

(SS)PB4

(AIN1/OC0)PB3

(AIN0/INT2)PB2

(T1)PB1

(T0/XCK)PB0

(TOSC2)PC7

(TOSC1)PC6

(TDI)PC5

(TDO)PC4

(TMS)PC3

(TCK)PC2

(SDA)PC1

(SCL)PC0

(OC2)PD7

(ICP)PD6

(OC1A)PD5

(OC1B)PD4

(INT1)PD3

(INT0)PD2

(TXD)PD1

(RXD)PD0

C1

C2

22p

Q1

C8

C7

100n

X6-1

X6-2

C5

IC3

7805T

VI

VO

C6

C11

C12

C13

C14

IC4

IC5

IC6

IC7

74AC574N

1D 2D 3D 4D 5D 6D 7D 8D

1Q 2Q 3Q 4Q 5Q 6Q 7Q 8Q

CLK

OC

R2

R10

R18

R26

JP4

JP5

JP6

JP7

X1

MAX232

IC2

R2IN

R1IN

T2OUT

T1OUT

R2OUT

R1OUT

T2IN

T1IN

C2-

C2+

C1-

C1+

V-

V+

C3

C4

C9

C10

1u