TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietokonetekniikan suuntautumisvaihtoehto

Jesse Rannikko

DIGITAALISEN HERÄTYSKELLON SUUNNITTELU JA TOTEUTUS

Insinöörityö, joka on jätetty opinnäytteenä tarkastettavaksi insinöörin tutkintoa varten Tampereella 10.3.2008

Työn valvoja: Yliopettaja Kai Poutanen

Tekijä: Jesse Rannikko
Työn nimi: Digitaalisen herätyskellon suunnittelu ja toteutus

Päivämäärä: 10.3.2008
Sivumäärä: 24 sivua ja 39 liitesivua
Hakusanat: AVR, mikrokontrolleri, herätyskello
Koulutusohjelma: Tietotekniikka
Suuntautumisvaihtoehto: Tietokonetekniikka

Työn valvoja: Yliopettaja Kai Poutanen

Herätyskello on useimmille jokapäiväinen apuväline. Keksintönä se on jo vanha, ja erilaisia malleja on tarjolla runsaasti. Useimpien mallien perusrakenne ja toimintatapa ovat kuitenkin samanlaisia, ja kirjoittajan mielestä nämä ovat vaivalloisia käyttää. Työn tavoitteena olikin suunnitella digitaalinen herätyskello, joka olisi helppokäyttöinen ja pysyisi tarkasti ajassa. Lisäksi laitteeseen haluttiin luotettava paristovarmennus.

Kello suunniteltiin Atmelin AVR-mikrokontrollerin ympärille. Kontrolleriksi valittiin ATmega88, koska siinä oli riittävästi I/O-linjoja ja integroituja oheislaitteita, kuten ajastimia ja AD-muunnin. Mikrokontrollerin lisäksi kytkennän tärkeimmät osat ovat tehonsyötön ja näytönohjauksen kytkennät. Nämä toteutettiin siten, että laitteen hukkateho ja virrankulutus ovat mahdollisimman pienet. Laitteen vaivattoman käytön varmistaa syöttölaitteena oleva numeronäppäimistö ja kaksi näyttöä, jotka tulevat kotelon päälle ja sivulle.

Laitteen varsinaisen toiminnallisuuden saa aikaan mikrokontrollerin ohjelma, joka kirjoitettiin C-kielellä ja käännettiin AVR-GCC-kääntäjällä. Ohjelman toiminta perustuu 5 ms:n välein tapahtuvaan keskeytykseen, jonka mukaan ylläpidetään kellonaikaa ja ajoitetaan ohjelman eri toiminnot. Esimerkiksi kukin näytön neljästä numerosta loistaa 5 ms kerrallaan, jolloin koko näyttö tulee päivetetyksi 50 kertaa sekunnissa.

Kytkentäkaaviot ja piirilevykuvat piirrettiin EAGLE Layout Editor -ohjelmalla, minkä jälkeen laite koottiin ja ohjelma siirrettiin mikrokontrollerin muistiin. Laite testattiin huolellisesti, ja kaikki löydetyt virheet saatiin korjattua. Testauksen yhteydessä laitteen virrankulutukseksi mitattiin noin 170 mA verkkosähköllä ja 7 mA paristokäytöllä. Työn lopputuloksena aikaansaatiin helppokäyttöinen ja varmatoiminen herätyskello.

Tämä tutkielma voi olla hyödyllinen suunniteltaessa erilaisia mikrokontrolleripohjaisia laitteita. Kytkentäkaaviossa ja ohjelman lähdekoodissa on paljon muissakin laitteissa hyödynnettävää asiaa. Mallia voi ottaa esimerkiksi paristovarmennuksen tai seitsensegmenttinäyttöjen ohjauksen toteutuksesta. Lisäksi WinAVR-ympäristöä ja STK200-ohjelmointikaapelia voidaan käyttää minkä tahansa AVR-mikrokontrollerin ohjelmointiin.

Author:	Jesse Rannikko
Name of the thesis:	Design and Implementation of a Digital Alarm Clock
Date:	10.3.2008
Number of pages:	24 pages and 39 appendix pages
Keywords:	AVR, microcontroller, alarm clock
Degree program:	Computer Systems Engineering
Specialization:	Embedded Systems
Thesis supervisor:	Senior Lecturer Kai Poutanen

Alarm clock is a daily tool for most of us. As an invention, it is already old and there are plenty of different models available. However, the basic structure and mode of operation are similar in most of them. In the opinion of the writer, they are inconvenient to use. Thus, the aim of the work was to design a digital alarm clock that would be easy to use and would keep time accurately. In addition, a reliable battery backup was wanted to be in the device.

The clock was designed around Atmel's AVR microcontroller. ATmega88 was chosen because it includes enough I/O lines and integrated peripherals, such as timers and AD converter. Besides the microcontroller, the key elements of the circuit are the power supply and the display control circuits. These were implemented in a way that waste power and current consumption are as low as possible. Effortless usage of the clock was ensured by using a number keypad as an input device and by equipping the clock with two displays, one on the top and one on the side of the case.

Device's actual functionality is generated by the program on the microcontroller. The program code was written in C language and compiled with the AVR-GCC compiler. The operation of the program is based on interrupt occurring at 5 ms intervals according to which the clock time is maintained and different activities of the program timed. For instance, each of the four digits of the display is lit for 5 ms at a time, giving the whole display a refresh rate of 50 Hz.

The circuit diagrams and board layouts were drawn with EAGLE Layout Editor. After that, the device was assembled and the program was transferred into the memory of the microcontroller. The device was carefully tested and all the defects found were corrected. Current consumption of the device was measured to be 170 mA on mains supply and 7 mA on battery supply during the testing. The end result of the work was an easy to use and reliably working alarm clock.

This thesis can be useful while designing different kinds of microcontroller-based devices. The circuit diagram and the source code include many things that can be utilized in other devices as well. For example, one can take advantage of the implementation of the battery backup or the seven-segment display control. In addition, the WinAVR environment and the STK200 programming cable can be used to program any AVR microcontroller.

ALKUSANAT

Idea ja tarve helppokäyttöisen herätyskellon rakentamiseen on ollut minulla jo kauan. Asia on hautunut mielessäni ja päätin sen nyt toteuttaa insinöörityökseni. Pääasiassa työ on tehty vuoden 2007 kevään ja syksyn aikana, mutta alustavaa ideointia ja suunnittelua aloitin jo aikaisemmin. Yhteensä työn tekemiseen kului noin 540 tuntia. Kaiken kaikkiaan työ oli erittäin mielenkiintoinen ja opettavainen.

Työni valvojaa Kai Poutasta haluan kiittää arvokkaista kommenteista ja muusta avusta työn eri vaiheissa. Lisäksi kiitos kuuluu kaikille opettajilleni Tampereen ammattikorkeakoulussa laadukkaasta opetuksesta. Erityisesti haluan kiittää vaimoani ja muita läheisiäni oikoluvusta ja kannustuksesta työn aikana.

Jesse Rannikko
Tampereella 10.3.2008

SISÄLLYSLUETTELO

LYHENTEET JA SYMBOLIT

AD-muunnin	*Analog-to-Digital*-muunnin, analogisen jännitteen digitaaliseksi lukuarvoksi muuttava laite
EEPROM	*Electronically Erasable Programmable Read-Only Memory*, haihtumattoman muistin tekniikka
Flash	EEPROMia kehittyneempi ja edullisempi muistitekniikka
I/O-linja	*Input/Output*-linja, sisääntulona ja lähtönä toimiva mikrokontrollerin liitäntäpinni
RISC	*Reduced Instruction Set Computer*, supistetun käskykannan prosessoriarkkitehtuuri
SRAM	*Static Random Access Memory*, haihtuvan muistin tekniikka

1 JOHDANTO

Työn tavoitteena oli suunnitella ja toteuttaa mikrokontrolleripohjainen digitaalinen herätyskello, jossa olisi muutamia parannuksia yleisesti myytäviin malleihin verrattuna. Näistä merkittävimpänä oli helpompi ja nopeampi kellon- ja herätysajan asettamisen tapa verrattuna normaaliin toteutukseen, jossa näppäintä painamalla kasvatetaan tuntien tai minuuttien määrää yhdellä. Tällöin näppäimiä joudutaan usein painamaan harmittavan monta kertaa, ja erityisesti ajan siirtäminen vähän taaksepäin on kovin työlästä.

Toisena tärkeänä parannuskohteena oli paristovarmennus. Laite käyttää siis pääasiallisena energialähteenä verkkosähköä, mutta sähkökatkosten varalle siinä on varmennusparisto, kuten vastaavissa laitteissa yleensä. Varmennuksen mielekkyyttä ja varmatoimisuutta haluttiin parantaa lisäämällä laitteeseen varmennuspariston tyhjentymisen ilmaisu. Pienempänä tavoitteena oli esimerkiksi säädettävä torkkuhälytyksen laukeamisväli. Suunnittelun aikana ominaisuuksiksi haluttiin lisäksi sisäänrakennettu yövalo ja kaksi erillistä herätystä, joista toisen voisi asettaa myös itsestään sammuvaan tilaan.

Toimintoihin liittyvien tavoitteiden lisäksi työn toteutukseen liittyi muitakin tavoitteita, joista tärkeimpänä oli energiatehokkuus eli mahdollisimman pieni virrankulutus ja hukkateho. Luonnollisesti kellon haluttiin myös pysyvän mahdollisimman tarkasti oikeassa ajassa. Lisäksi komponenttien ja muiden tarvikkeiden piti olla mahdollisimman edullisia. Laitteen kotelointi jätettiin tämän työn ulkopuolelle.

Piirilevysuunnitteluun sekä mikrokontrollerin ohjelmistokehitykseen ja ohjelmointiin käytettiin ilmaisia ohjelmia, jotka esitellään kappaleissa 2.2 ja 3.3. Tarvitut komponentit hankittiin Partco Oy:stä [Partco], SP-Elektroniikka Oy:stä [SP-Ele] ja TietoPetri Oy:stä [TietoP]. Kaikki piirilevyt valmistettiin Tampereen Sähköopiskelijat TASO:n piirilevylaboratoriossa. Kytkennän mittaukset suoritettiin Tampereen ammattikorkeakoulun tiloissa Fluke PM3382A -oskilloskoopilla sekä Fluke 75 ja Fluke 79 -yleismittareilla.

Seuraavassa luvussa käsitellään komponenttivalintoja ja piirilevysuunnittelua sekä kerrotaan, kuinka halutut ominaisuudet toteutettiin. Kolmannessa luvussa kuvataan lyhyesti käytettyä mikrokontrolleria ja esitellään siihen liittyvät ohjelmistot. Tämän jälkeen selostetaan mikrokontrollerille kirjoitetun ohjelman tärkeimpiä osia. Viimeisessä luvussa kerrotaan kytkennän kokoamisesta ja testauksesta sekä suoritetuista mittauksista.

2 LAITTEEN SUUNNITTELU

Suunnittelu alkoi kytkennän keskeisten osien valinnalla ja jatkui piirilevyn suunnittelulla. Näiden asioiden lisäksi tässä luvussa kerrotaan, kuinka erilaiset tavoitteena olleet asiat toteutettiin.

2.1 Komponenttien valinnat

Seuraavassa kerrotaan tärkeimpien komponenttien ja kytkentäratkaisuiden valinnoista ja valintaperusteista. Tarkemman kuvan saamiseksi esitellään myös muutamia toteutusvaihtoehtoja, jotka kuitenkin korvattiin paremmilla ratkaisuilla.

2.1.1 Mikrokontrolleri

Mikrokontrolleri päätettiin valita Atmelin AVR-perheestä, koska AVR-mikrokontrollereista oli jonkinlaista aiempaa tuntemusta, ohjelmointiin voidaan käyttää C-kieltä ja näille on saatavilla hyviä, ilmaisia kehitystyökaluja. Lisäksi AVR-mikrokontrollerit ovat harrastajien suosimia, joten tarvittaessa internetistä on saavissa paljon aiheeseen liittyvää materiaalia.

Haluttujen toimintojen toteuttamiseksi mikrokontrollerissa piti olla AD-muunnin ja noin 20 I/O-linjaa. Lisäksi tarvittiin kaksi ajastinta, toinen kaiuttimen soittamista ja toinen kellonajan ylläpitoa ja ohjelman ajoittamista varten. Näiden vaatimusten sekä hinnan ja saatavuuden perusteella kontrolleriksi valittiin ATmega88 [Atmel]. Ohjelma- ja käyttömuistin tarvittavista määristä ei ollut tarkkaa käsitystä suunnittelun alkuvaiheessa, mutta valitun kontrollerin 8 kt:n ohjelmamuistin ja 1 kt:n käyttömuistin oletettiin olevan riittäviä. Kappaleessa 2.1.3 kuvattavan suunnittelun edetessä todettiin, että kytkennässä tarvittiin myös analoginen komparaattori, joka valitussa kontrollerissa olikin sisäänrakennettuna.

2.1.2 Teholähdekytkentä

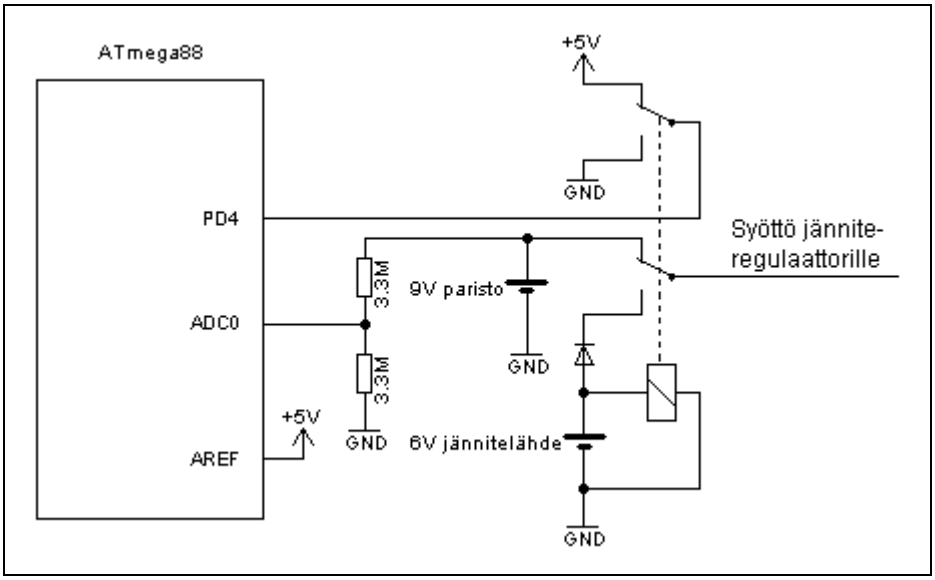
Laitteen syöttöjännite tulee normaalisti ulkoiselta tasajännitelähteeltä, mutta sähkökatkosten varalle laitteessa itsessään on yhdeksän voltin paristo. Käytössä oleva syöttöjännite pienennetään kytkennän käyttöjännitteeksi (5 V) jännite-regulaattorilla. Regulaattoriksi valittiin L4941, koska sen jännitepudotus on pienimmillään vain noin 0,5 V [L4941, s. 4]. Tällöin ulkoisen lähteen jännitteeksi riittää vain vähän kytkennän käyttöjännitettä suurempi arvo, jolloin regulaattorin hukkateho on mahdollisimman pieni.

Koko kytkentä huomioiden sallituksi syöttöjännitealueeksi saadaan 6–12 V, missä maksimin rajoittaa syöttöjännitteen tarkkailukytkentä. Tosin 7,6 volttia suuremmilla syöttöjännitteillä pariston näytetään virheellisesti olevan täysi silloin, kun paristoa ei ole asennettuna lainkaan. Tämä johtuu syöttöjännitteen potentiaalin siirtymisestä mittauskytkentään pariston kytkevän transistorin kautta. Mikäli syöttöjännitteenä käytetään 12 volttia, saattaa regulaattori vaatia jo jäähdytyslevyn.

2.1.3 Paristokäytön kytkentä

Paristokäytön kytkennän tehtävänä on verkkosähkön katketessa vaihtaa syöttöjännite tulemaan paristosta ja verkkosähkön palatessa palauttaa syöttöjännitteeksi ulkoinen lähde. Lisäksi mikrokontrollerille on saatava tieto jännitesyötön vaihtumisesta, koska paristokäytön aikana virrankulutusta pienennetään muun muassa sammuttamalla näytöt.

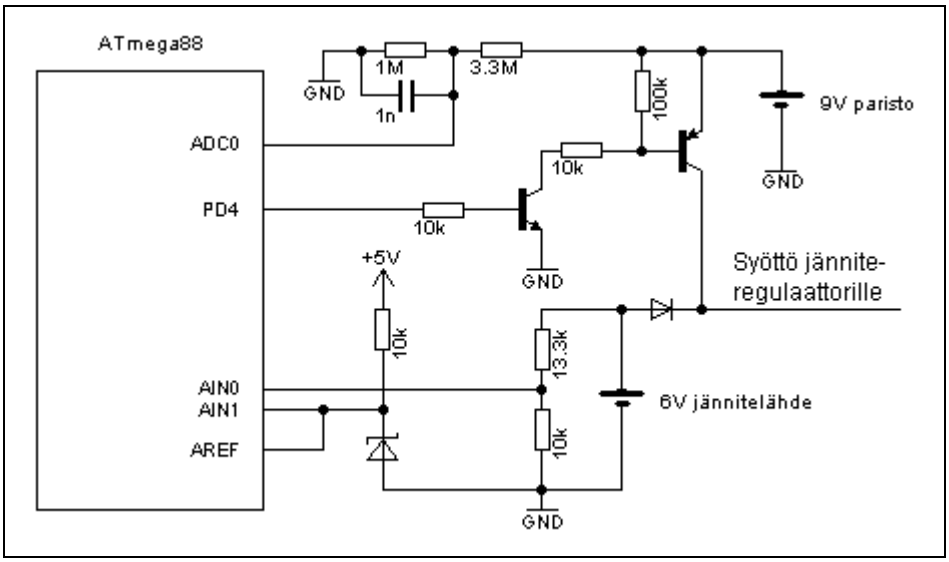
Ensimmäisenä vaihtoehtona suunnitelmissa oli yksinkertainen releeseen perustuva vaihtokytkentä, joka on esitetty kuvassa 1. Releen kela olisi kytkettynä suoraan syöttöjännitteeseen, ja varsinainen syöttö olisi johdettu diodin läpi releen vaihtokärjelle, josta se olisi kytkeytynyt edelleen regulaattorille. Näin syöttö-jännitteen vaihto olisi toiminut täysin itsenäisesti, ja tieto siitä olisi välitetty mikrokontrollerille releen toisella vaihtokärjellä. Releen vaihtoaikaa mittaamalla tämä kytkentä todettiin kuitenkin liian hitaaksi, jotta ulkoisen lähteen jännite olisi voitu pitää mahdollisimman alhaisena ja jänniteregulaattorin tehohäviö pienenä. Syöttöjännitteen pitäisi olla suurempi ja piiriä pitäisi olla vaihtoaikana syöttämässä suuri kondensaattori, jotta tämä kytkentä toimisi. Koska näin ei haluttu tehdä, syöttöjännitteen vaihtokytkentä piti saada nopeammaksi.



Kuva 1. Paristokäytön kytkennän ensimmäinen versio.

Nopeampi vaihtokytkentä saatiin toteutettua kahden transistorin avulla. Tällöin verkkosähkön katkeamisen havaitsemiseen käytetään mikrokontrollerin sisäistä analogista komparaattoria. Komparaattorin toisessa sisääntulossa on vakiona pysyvä referenssijännite ja toiseen on kytkettynä ulkoinen lähde jännitteenjaon kautta. Jännitteenjako muodostettiin siten, että komparaattori vaihtaa tilaansa ulkoisen lähteen jännitteellä 5,8 volttia. Jos syöttöjännite laskisi vielä tästä, alkaisi myös regulaattorin lähtö eli kytkennän käyttöjännite laskea huonoimmassa tapauksessa heti. Tämän rajajännitteen mukaan mikrokontrolleri ohjaa syöttöjännitteeksi transistorikytkimillä joko pariston tai ulkoisen lähteen siten, että käyttöjännite pysyy vakaana koko ajan.

Kuva 2 esittää transistoreilla toteutetun kytkennän ja siihen liittyvät muut komponentit. Paristojännitteeseen kytketty ylösvetovastus pitää varsinaisen kytkintransistorin kannan ja emitterin samassa jännitteessä, jolloin virta ei pääse kulkemaan paristosta. Kun mikrokontrolleri ohjaa aputransistorin johtavaksi, alkaa kytkintransistorikin johtamaan, jolloin paristo syöttää regulaattorikytkentää. Kytkennässä oleva diodi estää paristojännitteen pääsyn ulkoisen lähteen mittauskytkentään eli virheellisen paluun takaisin ulkoisen lähteen käyttöön.



Kuva 2. Paristokäytön toteutunut kytkentä.

Pariston tyhjenemisen havaitsemiseksi paristojännitettä mitataan mikrokontrollerin AD-muuntimella. Pariston jännite (9 V) pienennetään mittaukseen sopivalle tasolle jännitteenjakokytkennällä. Tämä kuormittaa paristoa koko ajan, mutta vastusten yhteenlaskettu resistanssi on 4,3 MΩ, jolloin virta on niin pieni, että paristo tyhjenee itsestään paljon nopeammin kuin tämän kuorman vaikutuksesta. Toisaalta AD-muuntimen sisääntulon impedanssin tulisi olla korkeintaan 10 kΩ [Atmel, s. 252]. Tämä korjattiin lisäämällä mittausvastuksen rinnalle pieni kondensaattori.

Sekä analoginen komparaattori että AD-muunnin käyttävät vertailujännitteenä 2,5 voltin referenssijännitettä, joka on tehty LM385-2.5-referenssillä. Tämä jännite on huomattavasti tarkempi ja vakaampi kuin esimerkiksi kytkennän käyttöjännite, joten referenssin käyttö mahdollistaa tarkemman ja luotettavamman mittauksen.

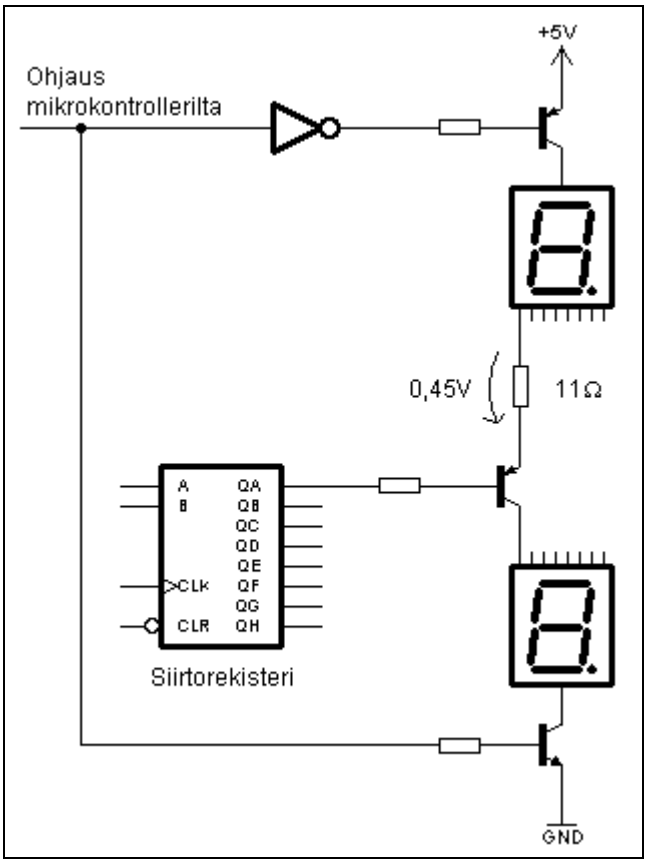
Kytkennän muun suunnittelun edetessä kokonaisvirrankulutusta saatiin vähennettyä ensimmäisistä laskelmista niin paljon, että syöttöjännitelähteen vaihtokytkennän olisi voinut toteuttaa kohtuuden rajoissa myös relekytkennällä. Syöttöjännitettä olisi kuitenkin pitänyt suurentaa esimerkiksi 9 volttiin, jolloin jänniteregulaattorin tehohäviö olisi kasvanut. Tämän vuoksi päätettiin käyttää suunniteltua transistorikytkentää.

2.1.4 Näyttö ja näytönohjauksen kytkentä

Luonnollinen valinta kellon näyttötyypiksi oli seitsensegmenttinäyttö, koska se oli tähän käyttöön riittävä sekä yksinkertainen toiminnaltaan ja ohjaukseltaan. Käytettävyyden vuoksi kelloon haluttiin kaksi näyttöä, toinen kotelon sivulle ja toinen sen päälle. Virrankulutuksen kannalta olisi merkittävää saada näytöt kytkettyä sarjaan, jolloin virta olisi puolet rinnankytkettyjen näyttöjen virrasta.

Näytönohjaus toteutettiin niin, että vuorotellen vain yksi näytön neljästä numerosta on loistamassa lyhyen ajan kerrallaan. Segmenttien ohjausdata siirretään sarjamuotoisena mikrokontrollerilta siirtorekisterille, joka muuntaa sen rinnakkaismuotoiseksi. Siirtorekisteri ohjaa kytkintransistoreita, joilla valitaan kulloinkin halutut segmentit.

Segmenttien ohjaus on kytketty kiinteästi kaikille numeroille, ja näytönohjauksen ensimmäisessä versiossa vuorossa oleva numero olisi valittu yhdistämällä se kahdella transistorilla käyttöjännitteeseen ja maapotentiaaliin. Kuvassa 3 on ohjauskytkennän ensimmäinen versio yhden segmentin osalta.



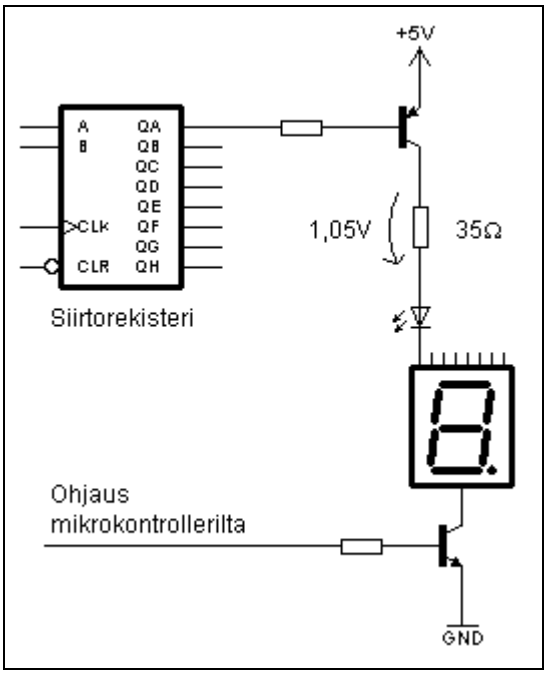
Kuva 3. Näytönohjauksen kytkennän ensimmäinen versio.

Helposti saatavilla olevien, valmiiden näyttöjen datalehtiä tutkimalla ja kahta näyttöä mittaamalla havaittiin, että näyttöjen kynnysjännite on noin 2 V ja halutulla kirkkaudella virran suuruus on 10 mA. Lisäksi tarvittavien kytkintransistorien yli jää pieni jännite, jolloin näytönohjauksen ensimmäisessä versiossa 5 voltin

käyttöjännitteestä olisi jäänyt virranrajoitusvastuksen yli vain noin 0,45 V ja vastuksen suuruus olisi silloin ollut vain noin 11 Ω. Tämän kytkennän käytännön toimivuus olisi ollut epävarmaa, koska komponenttien jännitearvojen pienikin poikkeama olisi aiheuttanut virtaan merkittävän muutoksen.

Tästä syystä toinen näytöistä päätettiin rakentaa irrallisista, suorakaiteen muotoisista ledeistä, koska näiden kynnysjännite on pienempi, noin 1,7 V. Merkittävänä etuna on lisäksi se, että irrallisista ledeistä kootussa näytössä on kaikilla segmenteillä sekä anodi että katodi. Valmiissa näyttökomponenteissa puolestaan jompikumpi näistä on yhteinen kaikille segmenteille. Kun kaikilla segmenteillä on sekä anodi että katodi, pystytään näytöt kytkemään sarjaan siten, että virtatien katkaisemiseen ei vaadita sekä käyttöjännitteen että maapotentiaalin katkaisemista omilla transistoreilla. Ohjaukseen tarvittiin siis yksi transistori vähemmän, ja virranrajoitusvastuksen yli jäi suurempi jännite.

Muutoksena ensimmäiseen versioon parannellussa ohjauskytkennässä (kuva 4) voitiin siis jättää virtatieltä yksi transistori pois, jolloin siirtorekisterin ohjaamat transistorit kytkevät segmentit suoraan käyttöjännitteeseen. Loistamaan halutun numeron valinta hoituu yhdellä transistorilla, joka yhdistää halutun numeron katodin maapotentiaaliin. Lisäksi yhden segmentin keskimääräiseksi virraksi todettiin riittävän 7,5 mA. Koska yksi numero on loistamassa vain neljäsosan ajasta, pitää tänä aikana virran vastaavasti olla nelinkertainen jatkuvaan virtaan verrattuna, eli tässä tapauksessa 30 mA. Näillä muutoksilla virranrajoitusvastuksen yli oleva jännite saatiin nousemaan 1,05 volttiin, jolloin vastuksen suuruus on 35 Ω. Aikaansaatu parannus oli siis merkittävä.



Kuva 4. Näytönohjauksen kytkennän toteutunut versio.

2.2 Piirilevysuunnittelu

Elektroniikka jakaantuu kolmelle piirilevylle: molemmat näytöt on koottu omille piirilevyille ja muu kytkentä omalle levylle. Lisäksi torkkunäppäimelle on pieni apupiirilevy. Kaikki piirilevyt ovat yksikerroksisia, ja kuparivetojen leveydet ja eristevälit mitoitettiin niin, että piirilevyt voitiin valmistaa itse. Kokoamistyötä helpottavana tavoitteena oli myös minimoida hyppylankojen tarve. Häiriöiden välttämiseksi suunnittelussa pyrittiin pitämään mikrokontrollerin käyttöjännite- ja maapotentiaalivedot erillään vedoista, joiden kautta liikkuu näyttöjen katkottu virta.

Kaikki kytkentäkaaviot ja piirilevykuvat piirrettiin EAGLE Layout Editor -ohjelmalla [CadSoft]. Pääpiirilevyn kytkentäkaavio on liitteessä 1. Vaikka kotelointi jätettiinkin tämän työn ulkopuolelle, huomioitiin suunnittelussa piirilevyjen ja muiden komponenttien sijoittaminen koteloon, jonka koko on noin 60x160x140 mm.

EAGLE on piirilevysuunnitteluohjelma, jonka peruskäyttö on helppo oppia, mutta se on samalla myös monipuolinen. Siitä on saatavilla ilmainen versio valmistajan kotisivuilla, jonka käytännön rajoituksena on piirilevyn maksimikoko, 80x100 mm. Suunnittelu etenee suoraviivaisesti piirtämällä ensin kytkentäkaavio, minkä yhteydessä valitaan myös komponenttien kotelot. Tämän jälkeen komponentit sijoitellaan piirilevylle ja kytkennät toteutetaan kuparivetoina. Jos tarvittavaa komponenttia ei ole ohjelman mukana tulevissa kirjastoissa, voi kirjastoja ladata lisää valmistajan internetsivuilta. Uusia komponentteja voi myös piirtää itse joko alusta alkaen tai yhdistelemällä muista, niin kuin tässäkin työssä tehtiin.

2.3 Haluttujen ominaisuuksien toteutukset

Nopea ja vaivaton herätys- ja kellonajan säätö toteutettiin korvaamalla perinteiset painonapit 16-näppäimisellä näppäimistöllä, jossa on numerot 0–9 ja viisi näppäintä asetusten tekoon. Hyvää käytettävyyttä lisäsi myös se, että laitteeseen tuli kaksi näyttöä, toinen kotelon päälle ja toinen sen sivulle. Näin näyttö on hyvin näkyvillä sekä makuu- että istuma-asennossa. Toisena tärkeänä tavoitteena oli ilmaista varmennuspariston tyhjentyminen, joka toteutettiin mittaamalla pariston jännitettä AD-muuntimella. Toiveena ollut torkkuajan pituuden säätömahdollisuus on laitteen asetusvalikossa, ja se oli helposti toteutettavissa ohjelmallisesti.

Koska varmennuspariston kytkeminen toteutettiin mahdollisimman nopeaksi, voi ulkoisen jännitelähteen jännite pienimmillään olla vain vähän laitteen käyttöjännitettä suurempi. Näin laitteen oman teholähdekytkennän hyötysuhde saatiin mahdollisimman hyväksi. Näyttöjen kytkeminen sarjaan pienensi sekä virrankulutusta että tehon hukkaamista lämmöksi virranrajoitusvastuksissa. Lisäksi virrankulutusta vähennettiin valitsemalla käyttöön pienin virta, jolla näytöt ovat vielä hyvin luettavissa valoisassakin. Myös kellotaajuudeksi valittu suhteellisen matala taajuus pienensi omalta osaltaan virrankulutusta.

Kellon käyntivirheen minimoimiseksi mikrokontrollerin kellolähteeksi valittiin ulkoinen kide sisäisen RC-oskillaattorin sijaan. Näppäimistössä oli riittävästi

näppäimiä kahden erillisen hälytyksen kontrollointiin, joten tämä ominaisuus voitiin ohjelmallisesti toteuttaa. Haluttu yövalo toteutettiin valkoisella ledillä, joka syttyy torkkunäppäimestä. Asetusvalikosta tämän toiminnon voi tarvittaessa myös poistaa käytöstä.

3 ATMEGA88-MIKROKONTROLLERI

Tässä luvussa on katsaus laitteen perustaksi valitun ATmega88-mikrokontrollerin tärkeimpiin ominaisuuksiin ja sen ohjelmointiin. Lisäksi esitellään mikrokontrollerin käyttöön liittyvät ohjelmistot.

3.1 Kontrollerin ominaisuudet

Atmelin AVR-tuoteperhe jakautuu mega- ja tiny-kontrollereihin. Tinyt ovat pelkistetyimpiä kontrollereita eli niissä on vähän muistia, I/O-linjoja ja integroituja oheislaitteita. Mega-kontrollereissa puolestaan on yleisesti ottaen näitä kaikkia enemmän, mutta rajanveto ei ole jyrkkä, vaan ominaisuudet lähenevät toisiaan rajakohdassa. Lisäksi mega-kontrollereissa on laajennettu käskykantaa muun muassa kertolaskulla.

Perusrakenteeltaan kaikki AVR-kontrollerit ovat samanlaisia, 8-bittisiä RISC-tyyppisiä mikrokontrollereita. Käskykanta on siis supistettu, ja suurin osa käskyistä suoritetaan yhdessä kellojaksossa. Kaikissa on myös sama prosessoriydin ja 32 kappaletta yleisrekistereitä, joten samaa koodia voidaan suorittaa kaikilla AVR-kontrollereilla [Vahtera, s. 30]. Erot kontrollereiden välillä ovat muistien ja oheislaitteiden määrissä, ja lisäksi joihinkin malleihin saa liitettyä myös ulkoista muistia.

Tässä työssä käytetty ATmega88 on mega-sarjan yksinkertaisimmasta päästä. I/O-linjoja on enimmillään 23, ohjelmamuistia (flash) on 8 kt, käyttömuistia (SRAM) on 1 kt ja haihtumatonta tallennusmuistia (EEPROM) 512 tavua. ATmega48 ja ATmega168 ovat saman kontrollerin variantit, jotka eroavat ainoastaan muistien määrissä.

Kontrollerissa on ajastimia yhteensä kolme, joista kaksi on 8-bittisiä ja yksi 16-bittinen. Jokaisella näistä on oma kellon esijakaja ja useita erilaisia toimintatiloja. Sisäänrakennetulla AD-muuntimella on 6-kanavainen sisääntulo, ja sen muunnostulos on 10-bittinen. Samaa 6-kanavaista sisääntuloa voidaan käyttää myös analogisen komparaattorin negatiivisen sisääntulon korvaajana. Esimerkkejä mikrokontrollerin ominaisuuksista, joita tässä työssä ei tarvittu, ovat muun muassa vahtikoira-ajastin, sarjaliitännät ja virransäästötilat.

Kellolähteenä voi toimia sisäisen RC-oskillaattorin lisäksi ulkoinen oskillaattori tai kellolähde. Kun käyttöjännite on 4,5–5,5 V, voi kellotaajuus olla jopa 20 MHz. Minimikäyttöjännitteellä 2,7 V voi kellotaajuus olla korkeintaan 10 MHz. Piiristä on saatavilla myös matalan jännitteen versio, joka kykenee 4 MHz:n nopeuteen 1,8 voltin käyttöjännitteellä.

3.2 Ohjelmointi

Ohjelmakoodin kirjoittaminen AVR-mikrokontrollereille on mahdollista useilla eri ohjelmointikielillä. Kääntäjä on saatavilla ainakin Basic- ja Pascal-kielille assembly- ja C-kielien lisäksi [Pros, s. 56]. Tässä tapauksessa ohjelma kirjoitettiin C-kielellä.

Ohjelman siirtämiseen mikrokontrollerin muistiin voidaan käyttää joko rinnakkais- tai sarjamuotoista ohjelmointia [Atmel, s. 292, 299]. Tässä työssä käytettiin sarjamuotoista ohjelmointia sen helppouden ja yleisyyden vuoksi. AVR-mikrokontrollerit ovat järjestelmäohjelmoitavia, eli ne voidaan ohjelmoida kontrollerin ollessa piirilevyllä osana kytkentää. Erillistä ohjelmointilaitetta ei siis tarvita, vaan kontrollerin yhdistäminen tietokoneeseen riittää.

Ohjelmointikaapelina käytettiin STK200-tyyppistä itsetehtyä kaapelia, joka liitetään tietokoneen rinnakkaisporttiin [yaap]. Yksinkertaisimmillaan ohjelmointikaapeliksi riittäisi muutama vastus ja johto, mutta kehittyneemmästä kaapelista on etuakin. STK200-kaapelissa olevan puskuripiirin ansiosta signaalitasot pysyvät riittävinä pidemmässäkin kaapelissa, ja useimmissa tapauksissa kaapeli voidaan pitää kiinnitettynä kytkentään ohjelman ajon aikanakin.

Varsinaisen ohjelmamuistin ohjelmoinnin lisäksi pitää mikrokontrollerin erityiset sulakebitit tarkistaa ja tarvittaessa ohjelmoida niille uudet arvot [Atmel, s. 287]. Näiden ohjelmointiin käytetään samaa liityntää kuin muistinkin ohjelmointiin. Sulakebiteillä valitaan muun muassa käytettävä kellolähde sisäisen oskillaattorin ja erilaisten ulkoisten kellolähteiden välillä.

3.3 Käytetyt ohjelmat

Ohjelmakoodi kirjoitettiin tavallisella tekstieditorilla. Koodin kääntämiseen käytettiin AVR-GCC-kääntäjää, joka on osa WinAVR-pakettia [WinAVR]. WinAVR on ilmainen kehitystyökalujen kokoelma AVR-mikrokontrollereille. Se sisältää kääntäjän lisäksi muun muassa C-kirjastotiedostoja ja avrdude-latausohjelman, jolla ohjelmakoodi siirretään mikrokontrollerin muistiin. Avrdudelle on saatavilla myös erillinen graafinen käyttöliittymä [avrd-gui]. Ohjelmat konfiguroitiin toimimaan Microsalon internetsivuilta löytyvien ohjeiden mukaan [Micros].

GCC-kääntäjä, johon AVR-GCC perustuu, on monipuolinen työkalu, jonka käyttö on hieman monimutkaista. Käännöstyötä helpottaa make-apuohjelma, joka sisältyy WinAVR-pakettiin. Tämän toimintaa puolestaan ohjaa makefile-tiedosto, jossa määritellään muun muassa käytetyn kontrollerin malli. Valmis tiedostopohja on saatavilla esimerkiksi Microsalon internetsivuilta [makef].

4 OHJELMA

Laitteen varsinainen toiminnallisuus aikaansaadaan mikrokontrollerille kirjoitettavalla ohjelmalla. Ohjelman suunnittelun periaatteet ja kirjoitetun ohjelman tärkeimmät osat esitellään tässä luvussa. Ohjelman koko lähdekoodi on liitteessä 2.

4.1 Ohjelman suunnittelun periaatteet

Perusperiaatteena ohjelmaa kirjoitettaessa oli selkeys ja koodin helppo lukeminen. Jotta laajojen osien toiminnat olisivat helposti ymmärtävissä, jaettiin koodi pieniä kokonaisuuksia suorittaviksi funktioiksi, vaikka monia niistä kutsutaankin vain kerran yhdestä paikasta. Lisäksi kommentointiin kiinnitettiin erityistä huomiota.

Toisena tärkeänä tekijänä oli tehokkuus, jonka vuoksi ohjelman kaikki muuttujat ovat etumerkittömiä ja muutamaa lukuun ottamatta 8-bittistä char-tyyppiä. Samasta syystä kaikki muuttujat ovat, yleisistä ohjeista poiketen, globaaleja muuttujia. Tällöin koodista tulee samalla yksinkertaisempi, koska samoja muuttujia tarvitaan useissa funktioissa ja lähes kaikkien muuttujien arvojen tulee olla voimassa koko ajan. Toisaalta haittapuolena on se, että muuttujien virheellisen käytön mahdollisuus kasvaa. Globaalien muuttujien käytöstä saatavien hyötyjen arvioitiin kuitenkin olevan niistä aiheutuvia ongelmia suuremmat.

Ohjelmalla suoritettavaa tehtävää hahmoteltiin ensin suurina kokonaisuuksina, minkä jälkeen edettiin kohti pienempiä ja tarkempia osia. Ennen varsinaisen C-kielisen koodin kirjoittamista ohjelma suunniteltiin varsin tarkkoihin yksityiskohtiin asti pseudokielellä. Lopullisen ohjelmakoodin valmistuttua siitä vielä etsittiin virheitä lukemalla ja tarkistamalla. Näin koodin toimivuus laitteessa pyrittiin saamaan mahdollisimman hyväksi ennen sen ensimmäistäkään kokeilua.

4.2 Pääohjelma

Pääohjelman perustana on ikuinen silmukka, jossa tutkitaan onko 5 ms:n kulumisen osoittava lippu_5ms-muuttuja nostettuna. Aina kun 5 ms on kulunut, suoritetaan pääohjelman varsinaiset toiminnot, jotka koostuvat lähes kokonaan funktio-kutsuista. Jos funktiot tarvitsevat pidempiä suoritusvälejä kuin 5 ms, hoidetaan ajoitus funktioissa itsessään.

Ensimmäisenä asiana näyttö palautetaan normaalitilaan eli kellonaika näkyviin, jos mitään näppäintä ei ole painettu kymmeneen sekuntiin. Seuraavaksi tarkistetaan, onko kellonaika muuttunut, ja kopioidaan tarvittaessa uusi aika näyttöpuskuriin. Myös herätyksien laukaisua tutkitaan vain silloin, kun kellonaika on muuttunut. Tämän jälkeen päivitetään näyttö eli vuorossa olevaan näytön numeroon asetetaan loistamaan näyttöpuskurin mukainen sisältö.

Lisäksi pääohjelmassa kutsutaan funktioita, jotka huolehtivat hälytyksen eli kaiuttimen soittamisen ajoituksista, yövalon sammutuksesta ja näppäimistön lukemisesta. Näppäimistön lukua käsitellään kappaleessa 4.7.

4.3 Alustukset

Heti pääohjelman alussa, ennen ikuista silmukkaa kutsutaan funktiota, jossa alustetaan mikrokontrollerin toiminnot. Ensimmäisenä määritellään porttien suuntarekistereihin, mitkä kontrollerin pinnit ovat lähtöjä ja mitkä sisääntuloja. Oletuksena lähdöt ovat 0-tilassa, joten tarvittavat lähdöt asetetaan 1-tilaan. Tämän jälkeen alustetaan ajastimet.

Timer0-ajastin asetetaan vertailutilaan, jossa aina ajastimen arvon muuttuessa sitä verrataan vertailurekisterin arvoon. Jos nämä arvot ovat samat, aiheutuu silloin keskeytys ja ajastimen arvo nollataan automaattisesti. Ajastinkeskeytyksellä aikaansaatava taajuus on $f = \frac{f_{osc}}{2 \cdot N \cdot (1 + OCR0A)}$, missä f on haluttu taajuus, f_{osc} kidetaajuus, N kellon esijakaja ja OCR0A vertailurekisterin arvo [Atmel, s. 97]. Tästä saadaan vertailurekisterin arvolle kaava $OCR0A = \frac{f_{osc}}{f \cdot 2 \cdot N} - 1$. Tässä tapauksessa ei haluttu tiettyä taajuutta, vaan keskeytysohjelman suoritusajaksi tasan 5 ms. Koska taajuuden muodostamisessa tarvitaan kaksi keskeytysohjelman suoritusta yhdellä jaksonajalla, saatiin edellä mainitun kaavan taajuudeksi $f = \frac{1}{2 \cdot 5\,ms} = 100\,Hz$. Kidetaajuus oli 1,8432 MHz, ja esijakajaksi asetettiin 64. Kun arvot sijoitettiin kaavaan, saatiin vertailurekisteriin alustettavaksi arvoksi 143.

Timer1-ajastin asetetaan muuten samaan vertailutilaan, mutta lisäksi vertailu-osumalla vaihdetaan lähtöpinnin PB1 tila. Sen sijaan kellolle ei käytetä esijakajaa, vaan kidetaajuus ohjaa ajastinta suoraan. Vertailurekisterin OCR1A arvo lasketaan samalla kaavalla kuin timer0-ajastimen tapauksessakin (N=1). Ajastinta käytetään kaiuttimen eli pietso-elementin soittamiseen halutulla taajuudella. OCR1A-rekisterin arvo vaihtelee ohjelman suorituksen aikana halutun taajuuden mukaan, ja se asetetaan KaynnistaKaiutin-funktiossa.

Lopuksi alustetaan analoginen komparaattori ja AD-muunnin. Näiden käyttämisestä kerrotaan tarkemmin kappaleessa 4.8. Tässä tapauksessa AD-muunnoksen nopeus ei ole kriittinen, joten AD-muuntimen kelloksi asetetaan esijakajan avulla 57,6 kHz, joka on mahdollisimman lähellä pienintä sallittua taajuutta 50 kHz [Atmel, s. 248]. Ensimmäinen muunnostulos ei ole kelvollinen [Atmel, s. 251], joten alustuksen yhteydessä suoritetaan tämä hylättävä muunnos, jonka lisäksi käynnistetään varsinainen muunnos. Alustusten viimeisenä toimenpiteenä on keskeytysten globaali sallinta.

4.4 Keskeytykset

Keskeytykset ovat ohjelman ja laitteen toiminnan olennainen osa, koska niillä saadaan aikaan tarkka ajoitus ja nopea reagointi ulkoiseen tilanmuutokseen. Yhteensä ohjelmassa on käytössä kolme keskeytystä. Muuttujat, joissa välitetään tietoa keskeytyksien ja muiden funktioiden välillä, määriteltiin käyttäen volatile-lisämäärettä, mikä on välttämätöntä ohjelman toimimisen kannalta [Vahtera s.175].

Timer0-ajastimen vertailuosumakeskeytys tapahtuu tarkasti 5 ms:n välein edellä kuvatun alustuksen mukaisesti. Keskeytysohjelmassa nostetaan pääohjelmalle lippu_5ms-muuttuja merkiksi 5 ms:n kulumisesta, minkä perusteella ajoitetaan pääohjelman ja eri funktioiden suoritukset. Lisäksi keskeytysohjelmassa hoidetaan kellonajan ylläpito, jonka toiminta kuvataan seuraavassa kappaleessa.

Aikakeskeytyksessä käynnistetään myös AD-muunnin kaksi kertaa vuorokaudessa, kello 10:00 ja 20:00. Lisäksi käyttäjä voi halutessaan käynnistää muuntimen asetusvalikon kohdasta ”batt”. Kun muunnos tulee valmiiksi, aiheutuu siitä AD-muuntimen keskeytys, jossa tulos sitten käsitellään ja toimitaan sen mukaisesti. Kolmas ohjelmassa käytetty keskeytys on analogisen komparaattorin keskeytys, joka tulee aina silloin, kun komparaattorin lähdön tila muuttuu. Nämä keskeytysohjelmat käsitellään kappaleessa 4.8.

4.5 Kellonajan ylläpito

Kellonajan perustana on mikrokontrollerin kellosignaalin muodostava kide, joka tahdistaa kontrollerin ajastimet. Timer0-ajastin siis generoi keskeytyksiä 5 ms:n välein, ja ajan kuluminen lasketaan tämän perusteella.

Kellonaika pidetään tallessa muuttujissa siten, että näytöllä näkyvän osuuden jokainen numero on tallennettuna omaan muuttujaan, mikä yksinkertaistaa näytön päivittämistä. Toisin sanoen kymmenet tunnit ja tunnit ovat omissa muuttujissa, ja minuutit samalla tavoin, mutta sekunneille ja millisekunneille on kummallekin yksi muuttuja. Keskeytysohjelmassa lasketaan 5 ms:n ajanjaksojen kulumisia, ja kun yksi sekunti on kulunut, lisätään sekuntien määrää yhdellä. Vastaavalla tavalla päivitetään minuuttien ja tuntien kuluminen. Lisäksi keskeytysohjelmassa nostetaan lippu_kello-muuttuja merkiksi pääohjelmalle aina, kun minuuttien arvo muuttuu.

Keskeytysohjelma siis huolehtii itsenäisesti kellonajan oikeellisuudesta kello-muuttujissa. Näistä muuttujista pääohjelma käy kopioimassa kellonajan näyttöpuskuriin silloin, kun näytöllä kellonaikaa näytetään.

4.6 Näytönpäivitys

Näytön päivittäminen toimii siis virkistysperiaatteella, eli vain yksi näytön neljästä numerosta on kerrallaan loistamassa. Ajoitus hoituu siten, että näytönpäivitys-funktiota kutsutaan pääohjelmasta 5 ms:n välein, joka on siis yhden loistoajan pituus. Koko näyttö tulee siis virkistetyksi 50 kertaa sekunnissa, jolloin silmälle kaikki numerot näyttävät loistavan koko ajan. Kun laite on paristokäytöllä, näyttö pidetään sammuksissa, eikä näytönpäivitysfunktiossa tehdä mitään.

Näytöllä näkymään haluttu sisältö on tallennettuna näyttöpuskurissa eli nelipaikkaisessa puskuri-taulukossa. Esimerkiksi kun käyttäjä painaa asetusvalikon näppäintä, ohjelmassa kirjoitetaan näyttöpuskuriin valikon ensimmäisen kohdan teksti. Muuttunut teksti päivittyy näytölle heti seuraavana vuorossa olevasta numerosta alkaen. Toisin sanoen näytön yhden virkistyskierroksen aikana näytöllä

saattaa osa numeroista olla edellistä ja osa nykyistä puskurin sisältöä. Tällä ei kuitenkaan ole käytännön merkitystä, koska silmä ei sitä ehdi havaitsemaan.

Näytönpäivitysfunktion alussa haetaan puskurista vuorossa olevan numeron sisältö. Sitten määritetään merkkiä vastaava bittijono eli tieto, mistä segmenteistä merkki muodostuu. Seuraavaksi katkaistaan virtasilmukka kaikilta näytön numeroilta eli ohjataan kaikki kytkintransistorit johtamattomiksi. Tämän jälkeen kirjoitetaan ensimmäisen segmentin tila siirtorekisterin datalinjalle ja kellopulssi kellolinjalle. Näin lähetetään kaikkien kahdeksan segmentin ohjaukset. Kahdeksansina segmentteinä ovat herätyksien merkkiledit sekä tuntien ja minuuttien välissä olevan kaksoispisteen ledit. Lopuksi ohjataan vuorossa olevan numeron kytkintransistori johtavaksi ja asetetaan puskurin indeksi seuraavaksi virkistettävän numeron kohdalle.

Lisäksi näytönpäivitysfunktiossa toteutetaan kellonajan vilkkuminen näytöllä laitteen käynnistymisen jälkeen, ennen kuin käyttäjä on asettanut kellonaikaa. Tämä saadaan aikaan kirjoittamalla näytölle vuorotellen puoli sekuntia näyttöpuskurin sisältöä ja puoli sekuntia tyhjä-merkkiä. Näyttö vilkkuu siis hertsin taajuudella, ja samalla tavoin vilkutetaan myös vuorossa olevaa numeroa kaikkien käyttäjän tekemien säätöjen aikana, jotta säädön tekeminen olisi selkeätä.

4.7 Näppäimistön lukeminen

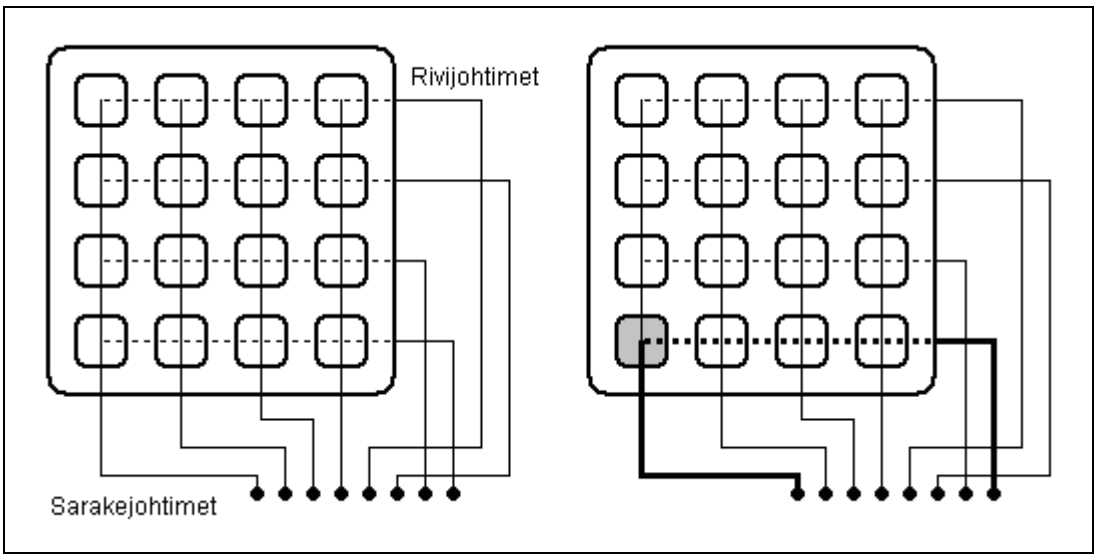
Näppäimistön lukeminen tapahtuu kiertokyselyperiaatteella, eli näppäimistö luetaan aina lyhyen ajan välein. Kun lukeminen tapahtuu kytkinvärähtelyä (noin 1–10 ms) pidemmin välein, vältytään tämän aiheuttamilta virheiltä. Painalluksen lyhimmäksi kestoksi mitattiin (eräällä kytkimellä) kokeilemalla hieman alle 100 ms. Valittuun 50 ms:n lukuväliin päädyttiin, jotta kaikki painallukset tulisivat varmasti havaituksi. Painallukseksi tulkitaan vain painamisen aloittaminen eli tilanne, jossa edellisellä lukukerralla näppäin ei ole painettuna ja seuraavalla kerralla se on painettuna. Näin näppäimeen reagointi tapahtuu välittömästi, ja yhtä painallusta ei tulkita useammiksi painalluksiksi.

Käytetty näppäimistö on 4x4-matriisinäppäimistö, jonka rakenne ja näppäimen painallus on esitetty kuvassa 5. Kun näppäintä painetaan, yhdistyvät sen alla olevat sarakejohdin (kiinteä viiva kuvassa) ja rivijohdin (katkoviiva) toisiinsa. Esimerkiksi kun alimman rivin vasenta näppäintä painetaan, yhdistyvät näppäimistön liittimen reunimmaiset pinnit toisiinsa.

Rivijohtimet on kytketty ylösvetovastuksilla käyttöjännitteeseen, ja sarakejohtimiin on kytkettynä mikrokontrollerin ohjaaman kahdesta neljään -dekooderin alhaalla aktiiviset lähdöt. Näppäimen tilan tutkiminen tapahtuu osoittamalla dekooderilla vuorollaan yksi sarakejohdin ja lukemalla rivijohtimien tilat (jännitteet). Kun näppäin on painettuna, on sen tila 0 ja vastaavasti muulloin 1.

Kun näppäimen havaitaan olevan painettuna, tallennetaan sen numero tai kirjaimen numerovastaavuus merkki-muuttujaan ja lukurutiini keskeytetään. Tämän jälkeen siirrytään painallukseen reagoivaan funktioon, jossa huolehditaan painetun

näppäimen mukaisen toiminnan suorituksesta. Lisäksi tässä funktiossa estetään toisen säädön aloittaminen silloin, kun jokin säätö on jo käynnissä, ja kaikki säätötoimenpiteet herätyksen aikana.



Kuva 5. Näppäimistön rakenne.

Näppäimistön lisäksi laitteeseen tulee isokokoinen näppäin torkkunäppäimeksi. Tämä näppäin on kytketty numeronäppäimen ”3” rinnalle, ja ohjelmassa huolehditaan oikea toiminto eri tilanteissa. Lisäksi torkkunäppäin sytyttää yövalon silloin, kun hälytys tai mikään säätö ei ole käynnissä.

Ohjelmassa on myös näytönpalautustoiminto, joka palauttaa näytön perustilaan, kun mitään näppäintä ei ole painettu kymmeneen sekuntiin tai sekunnin kuluttua säätötoimenpiteen tultua valmiiksi. Jos herätysajan säätö jää kesken, muutettu osa jää voimaan. Minuutteja ei siis tarvitse asettaa uudelleen, jos esimerkiksi aikaa siirretään yksi tunti. Kellonaika puolestaan on aina asetettava kokonaan, eikä kesken jäänyttä säätöä huomioida ollenkaan.

4.8 Paristokäyttö

Verkkosähkön katkeaminen havaitaan mikrokontrollerin sisäisellä analogisella komparaattorilla. Komparaattorin negatiiviseen sisääntuloon on kytkettynä 2,5 V:n jännitereferenssi ja positiiviseen sisääntuloon ulkoinen jännitelähde jännitteenjaon kautta. Komparaattori on alustettu niin, että se aiheuttaa keskeytyksen aina, kun sen lähtö vaihtaa tilaansa eli silloin, kun mittausjännite muuttuu suuremmaksi tai pienemmäksi kuin referenssijännite.

Keskeytysohjelmassa tutkitaan komparaattorin nykyinen tila, jonka mukaan joko asetetaan laite paristokäytölle tai palataan takaisin käyttämään verkkosähköä. Paristokäytölle siirryttäessä ensimmäiseksi sammutetaan näytöt, mikä tehdään katkaisemalla niiden virtasilmukat eli ohjaamalla kytkintransistorit johtamattomiksi. Seuraavaksi vaihdetaan syöttöjännitteeksi paristo ohjaamalla tämän

transistorikytkin johtavaksi (kuva 2). Lopuksi sammutetaan vielä muita virtaa kuluttavia osia ja vaihdetaan paristokaytto-muuttujan tila merkiksi muille funktioille. Näyttöjen virrankulutus on niin suuri, ettei pariston energia riittäisi kuin lyhyeksi ajaksi, jos näyttöjä ei sammutettaisi. Tärkein merkitys paristokäytöllä onkin säilyttää kellonaika, mutta myös hälytys laukaistaan paristokäytön aikana.

Kun paristokäytöltä palataan normaaliin tilaan, ohjataan pariston transistorikytkin johtamattomaksi, jolloin syöttöjännite pääsee taas kulkemaan ulkoiselta jänniteläheeltä regulaattorille. Paristokaytto-muuttuja asetetaan takaisin tilaan ”EI”, jolloin näytöt alkavat taas loistamaan, kun näytönpäivitysfunktio seuraavan kerran suoritetaan. Lopuksi käynnistetään vielä paristojännitteen mittaus.

Pariston jännitettä mitataan AD-muuntimella. Paristojännite pienennetään jännitteenjaolla mikrokontrollerille turvalliselle tasolle siten, että mittausjännite on $V = V_{batt} \frac{1,0}{1,0 + 3,3}$. Tyhjän pariston jännitteeksi valittiin 7,2 V, koska tällöin paristossa on energiaa jäljellä pitämään laite vielä käytössä noin 10 tuntia [Duracell]. AD-muuntimen tulos saadaan kaavalla $ADC = \frac{V \cdot 1024}{V_{ref}}$, missä V_{ref} on referenssijännite, eli tässä tapauksessa 2,5 V [Atmel, s. 256]. Kun jännitteet sijoitettiin kaavaan, saatiin tyhjän pariston 10-bittiseksi muunnostulokseksi $ADC = \frac{7,2V \frac{1,0}{1,0 + 3,3} \cdot 1024}{2,5V} = 685,84 \approx 686 = 1010101110_b$. Tässä tapauksessa kahdeksan bitin tarkkuus riittää, joten ohjelmassa tutkitaan vain ADCH-rekisteriä, johon eniten merkitsevät bitit ovat tallennettuina. AD-muuntimen keskeytys-ohjelmassa yksinkertaisesti sytytetään tai sammutetaan alhaisen paristojännitteen merkkiledi mittaustuloksen perusteella.

4.9 Hälytyksen laukaiseminen

Herätykset käsittelevää funktiota kutsutaan pääohjelmasta silloin, kun kellonaika on muuttunut eli minuutin välein. Ensimmäisenä tutkitaan herätyksen 1 laukeaminen, ja jos se ei laukea, niin tutkitaan myös herätyksen 2 tilanne. Toisin sanoen molemmat herätykset eivät voi laueta yhtä aikaa. Lisäksi samassa funktiossa laukaistaan herätyksen aputoiminto, jonka tapahtuminen on myös sidottu kellonaikaan. Tässä aputoiminnossa laukaistaan kovempi hälytys minuutin päästä herätyksen alusta ja sammutetaan herätys kokonaan 10 minuutin päästä, jos siihen ei ole tänä aikana reagoitu mitenkään.

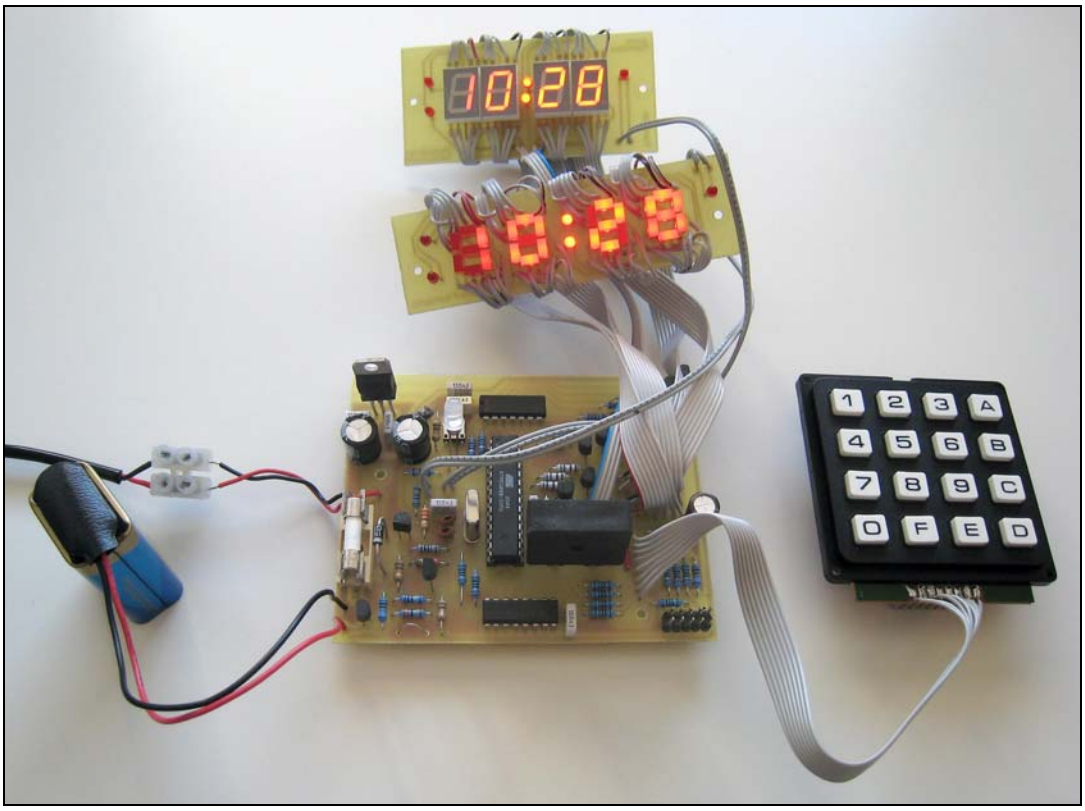
Herätys laukaistaan silloin, jos käyttäjä on mahdollistanut herätyksen sen käyttökytkimellä ja jos nykyinen kellonaika on sama kuin herätyksen tapahtuma-aika, joka on tallennettu heratys_ajat-taulukkoon. Herätyksen laukaiseminen tapahtuu asettelemalla hälytyksen tilaa kuvaavat muuttujat ja ajoituksesta huolehtivat laskurit sekä käynnistämällä kaiuttimen soittaminen.

5 LAITTEEN VALMISTUS JA TESTAUS

Laite valmistettiin suunnitelmien mukaan, minkä jälkeen ohjelman ja laitteiston toiminta testattiin tarkasti. Näiden molempien toiminnasta löytyi testauksen aikana virheitä, joista muutamia otetaan tässä esille. Lisäksi tutkittiin kellonajan tarkkuutta.

5.1 Laitteen kokoaminen

Suunniteltujen piirilevykuvien mukaiset piirilevyt valmistettiin syövytysmenetelmällä. Tämän jälkeen osat juotettiin levyille ja näyttöjen piirilevyt ja näppäimistö johdotettiin pääpiirilevyyn. Lopuksi ohjelma siirrettiin mikrokontrollerin muistiin kappaleessa 3.3 kuvatuilla välineillä. Avrdude-latausohjelma näytti mikrokontrollerin ohjelmamuistista kuluvan noin 40 % ja muuttujien varaavan käyttömuistia 62 tavua eli 6 %. Valitun kontrollerin muistit riittivät siis erittäin hyvin kirjoitetulle ohjelmalle. Kuvassa 6 on valmis laite toiminnassa.



Kuva 6. Koottu laite toiminnassa.

5.2 Ohjelman testaus

Ohjelman toiminta testattiin kokeilemalla kaikki laitteen eri ominaisuudet ja yrittämällä aikaansaada virheellistä toimintaa käyttäjän odotetuilla ja odottamattomilla toimilla.

5.2.1 Herätyksien testaus

Herätyksien laukeamista, ja laukeamattomuutta väärissä tilanteissa, tutkittiin tarkasti. Samoin kokeiltiin herätyksen aputoiminnon kaikkien vaiheiden oikea toiminta. Näissä ei havaittu mitään varsinaisia virheitä, mutta yksi epäjohdonmukaisuus. Molemmat herätykset eivät voi laueta samalla kellonajalla, mutta jos yksi herätys on käynnissä, voi toinen laueta tämän aikana korvaten ensimmäisen. Vaikutus on lähinnä kosmeettinen, koska vain sammutusnäppäin vaihtuu. Tämä voitiin kuitenkin helposti korjata lisäämällä TarkistaHeratykset-funktioon yksi ehtolause.

Lisäksi mitattiin myös itse hälytysäänen taajuudet ja oikea ajoitus. Hälytysääni saadaan aikaan soittamalla vuorotellen kahta eri taajuutta. Kaiuttimen soittamien taajuuksien todettiin olevan ohjelmassa asetettujen taajuuksien mukaiset, ja kumpikin taajuus soi vuorollaan 10 ms. Samoin soiton tauotus 300 ms:n ääneen ja 200 ms:n taukoon toimi oikein. Myös hälytysäänen jaksotus ensimmäisen minuutin aikana 5 sekunnin soittoon ja hiljaisuuteen todettiin oikeaksi.

5.2.2 Näppäimistönluvun toiminta

Näppäimistön lukuväliksi mitattiin oskilloskoopilla odotettu 50 ms. Näppäimistön todettiin reagoivan luotettavasti kaikkiin sallittuihin painalluksiin. Yrittämällä mahdollisimman nopeaa painallusta onnistuttiin kuitenkin saamaan aikaan myös painalluksia, joihin ohjelma ei reagoinut, mutta tällä ei ole käytännön merkitystä. Lisäksi yritettiin aikaansaada virheellistä toimintaa painelemalla vääriä näppäimiä eri tilanteissa. Testauksen perusteella todettiin, ettei vääriä kellonaikoja tai muita virheellisiä tai ohjelmallisesti estettyjä näppäilyjä hyväksytä.

Näppäimistön testauksen yhteydessä tutkittiin myös näytönpalautuksen toimintaa. Tästä löytyi joitain virheitä ja odottamatonta toimintaa. Korjauksien jälkeen näytönpalautus toimi halutulla tavalla, eli eri tilanteiden jälkeen näytölle palautui aina kellonaika ja muutkin ledit asettuivat oikeisiin tiloihin.

5.2.3 Ohjelman muiden toimintojen testaus

Muut toiminnot, jotka ovat käyttäjän säädettävissä, ovat laitteen asetusvalikossa. Myös näiden oikea toimivuus kokeiltiin. Herätyksen 2 soittotavan valinta ja herätyksien yhteinen torkkuajan pituuden valinta vaikuttivat laitteen toimintaan odotetusti. Samoin yövalon toiminta lakkasi, jos se asetusvalikosta kiellettiin. Valon loistoajaksi mitattiin ohjelmassa asetettu 2 minuuttia. Kappaleessa 5.3.1 kuvattavan korjauksen jälkeen myös paristojännitteen mittauksen käynnistäminen valikosta todettiin toimivaksi.

Ohjelman testauksen yhteydessä havaittiin myös joitakin epäjohdonmukaisuuksia ja käytettävyyden puutteita, joita ohjelmaa suunniteltaessa ei osattu ottaa huomioon. Esimerkiksi vasemmassa numerossa oleva nolla, joka ei ole näkyvillä silloin, kun näytöllä on kellonaika, muutettiin näkymään aikoja säädettäessä, jotta asetuksen

tekeminen olisi käyttäjän kannalta selkeätä. Koska aikaa asetettaessa pitää kaikki numerot asettaa joka tapauksessa, on selkeintä pitää kaikki numerot myös näkymässä koko säädön ajan. Lisäparannuksena muutettiin vuorossa oleva numero vilkkumaan asetuksia tehtäessä.

5.3 Laitteiston testaus

Laitteiston testauksessa kiinnitettiin huomiota analogisiin jännitteisiin ja paristokäytön toimivuuteen. Lisäksi mitattiin laitteen virrankulutus verkkosähköllä ja paristokäytöllä.

5.3.1 Analogisten jännitteiden korjaukset

Heti aluksi havaittiin, että alhaisen paristojännitteen merkkiledi ei syty, vaikka paristoa ei ole asennettuna lainkaan. Ongelmaksi osoittautui referenssijännite, joka olikin vain noin 1,7 V eikä 2,5 V. Tämä johtui referenssin liian pienestä virrasta, mikä puolestaan aiheutui referenssin odotettua suuremmasta kuormituksesta.

Alun perin referenssijännitettä oli tarkoitus käyttää vain komparaattorilla, jonka sisääntulojen vuotovirta on korkeintaan 50 nA [Atmel, s. 305]. Referenssin virraksi valittiin 25 μA, joka on hieman pienintä sallittua virtaa, 20 μA, suurempi [LM385, s. 4]. Kun referenssijännite päätettiin kytkeä myös AD-muuntimelle, ei virrankulutusta arvioitu tarkemmin uudelleen. AD-muuntimen kuormituksen referenssiin havaittiin kuitenkin olevan jopa 140 μA [Atmel, s. 340]. Referenssin virta suurennettiin 250 μA:iin, jonka jälkeen sen jännitteeksi mitattiin 2,504 V ja paristojännitteen mittaus toimi oikein.

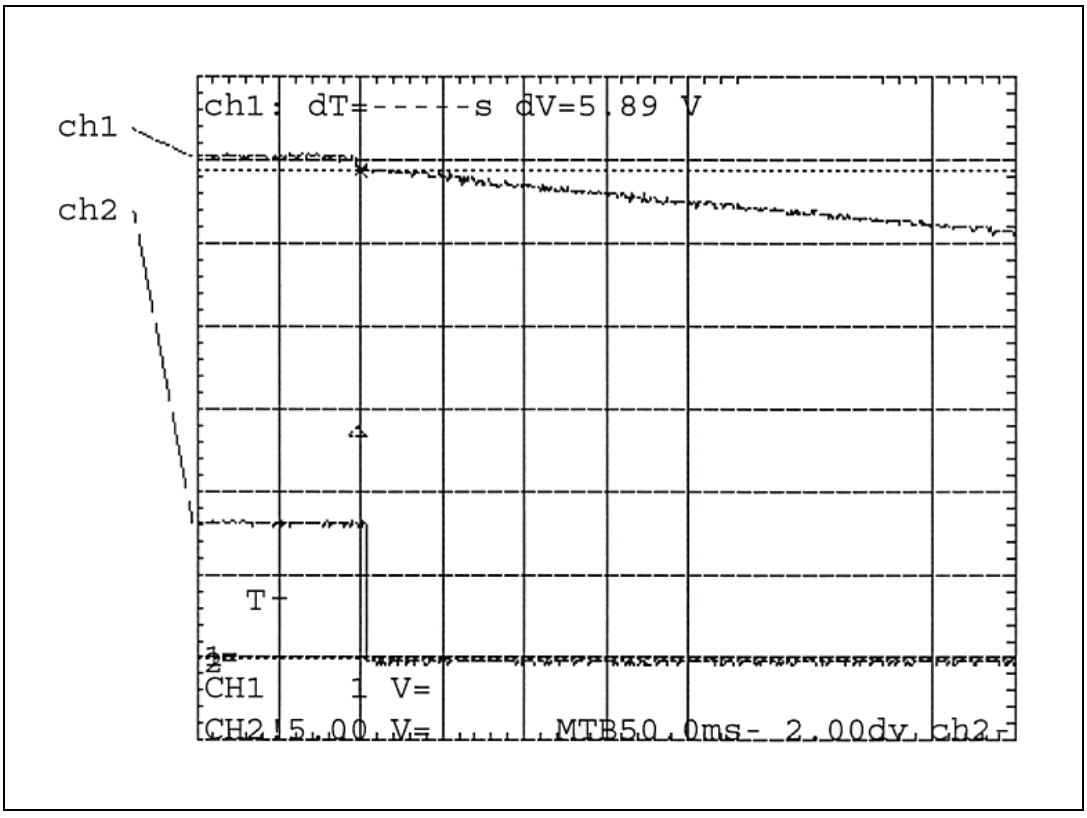
Tämän jälkeen ilmeni, että paristojännitteen mittausta tehdään silmämääräisesti jatkuvasti, vaikka ohjelmassa AD-muunnin käynnistetään vain kaksi kertaa vuorokaudessa. Kokeilemalla ohjelmaa pienillä muutoksilla paljastui, että ohjelman suoritus hyppi koko ajan komparaattorin keskeytysohjelmaan, jossa AD-muunnin käynnistetään, kun paristokäytöltä palataan verkkosähkön käyttöön. Komparaattori siis aiheutti keskeytyksen, mutta ohjelman suorituksen siirryttyä keskeytykseen, jossa ensimmäiseksi luetaan komparaattorin tila, oli tila jo ehtinyt vaihtua takaisin. Tällöin suoritettiin verkkosähkölle paluun toimenpiteet, joihin kuuluu myös AD-muunnoksen käynnistäminen.

Asiaa selvitettiin tarkemmin tutkimalla ja kokeilemalla komparaattoriin liittyvien eri osien toimivuutta. Ongelman aiheuttajaksi osoittautui komparaattorin toisessa sisääntulossa oleva jännitteenjakokytkentä, joka mittaa ulkoista jännitelähdettä. Sen läpi kulkeva virta oli niin pieni, että kytkentä oli altis häiriöille, eikä mittausjännite pysynyt vakaana. Vastuksia pienentämällä virta suurennettiin kymmenkertaiseksi noin 260 μA:iin, minkä jälkeen virheellinen hyppiminen komparaattorin keskeytysohjelmaan loppui.

5.3.2 Paristokäytön testaus

Ulkoisen lähteen jännitetaso, jolla syöttölähteeksi vaihdetaan paristo, mitattiin kahdella eri tavalla. Ensiksi syöttölähteeksi kytkettiin säädettävä laboratorio-jännitelähde, ja jännitettä pienentämällä tutkittiin vaihtotasoa. Siirtyminen paristokäytölle alkoi 5,90 V:n jännitteestä, ja syöttöjännitteen laskettua 5,85 V:iin oli laite siirtynyt kokonaan paristokäytölle. Syöttöjännitteessä havaittiin siis olevan 0,05 V:n alue, jossa näyttö välkkyy eli laite on osan ajasta paristokäytöllä.

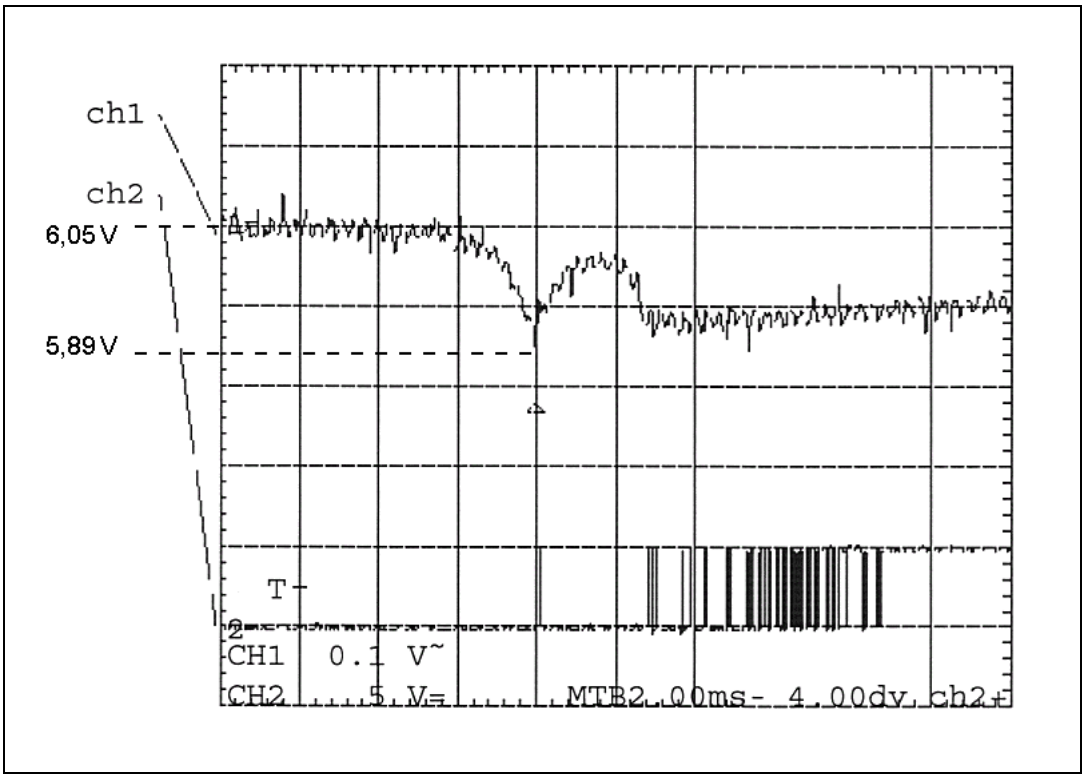
Tämän jälkeen mitattiin oskilloskoopilla jännitetaso, jossa paristokäytölle vaihto tapahtuu silloin, kun ulkoinen (kiinteän jännitteen) jännitelähde irrotetaan verkkosähköstä. Kuva 7 esittää mittaustuloksen: kanavassa 1 on syöttöjännite ja kanavassa 2 pariston kytkevän transistorin kannan jännite. Mittauksen perusteella rajajännitteeksi saatiin 5,89 V. Jotta jännitteen lasku näkyisi selkeämmin, mitattiin jännitettä myös vaihtojännitemittauksella, jonka tulos on kuvassa 8. Kanavassa 1 on syöttöjännite, jonka nollataso on 6,05 V, ja kanavassa 2 on mikrokontrollerin ohjaus, jolla paristo kytketään käyttöön. Tästä nähdään myös se, että paristokäytölle vaihto kestää hieman alle 10 ms.



Kuva 7. Paristokäytön rajajännite.

Eri menetelmillä mitatut jännitteet ovat lähes samoja, mutta noin 0,06 V suurempia kuin komponenttiarvojen perusteella laskettu jännite (5,83 V). Tämä selittyy jännitteen epävakaudella, komparaattorin sisääntulojen erolla ja mittausvastusten toleranssilla. Vaihtotason saisi myös korjattua alemmaksi jännitteenjaon vastusten

resistansseja muuttamalla, mutta tätä ei katsottu tarpeelliseksi, koska kytkentä toimi jo luotettavasti halutulla tavalla.



Kuva 8. Paristokäytön rajajännite, AC-mittaus.

Alhaisen paristojännitteen varoitustasoa tutkittiin kytkemällä pariston tilalle säädettävä laboratoriojännitelähde. Jännitettä vaihdeltiin odotetun varoitustason ympärillä ja paristojännitteen mittaus käynnistettiin asetusvalikosta jokaisen jännitteen muutoksen jälkeen. Näin tyhjän pariston jännitteeksi saatiin 7,21 V, joka on ohjelmassa asetettu jännite.

5.3.3 Virrankulutuksen mittaukset

Suunnittelun tärkeänä osana oli aikaansaada mahdollisimman pieni virrankulutus. Toteutuneet virrankulutukset mitattiin yleismittarilla. Paristokäytöllä laitteen virrankulutus on vain 6,91 mA, joka on hieman vähemmän kuin arvioitu maksimikulutus. Tästäkin yli puolet on jänniteregulaattorin hukkavirtaa, joka on 3,85 mA. Verkkosähköllä virrankulutukseksi mitattiin 167 mA, joka on huomattavasti vähemmän kuin arvioitu virrankulutus.

Selityksenä tähän oli näyttöjen odotettua pienempi virrankulutus. Yhden segmentin virta oli noin 24 mA suunnitellun 30 mA:n sijaan. Tämä selittyi näyttöjen ledien kynnysjännitteellä, joka on hieman suurempi pulssitetulla, suuremmalla virralla kuin jatkuvalla, keskimääräisellä virralla. Toteutuneella virralla aikaansaatu näytön kirkkaus saattaa olla jo hieman liian pieni voimakkaassa päivänvalossa. Toisaalta

näyttöjen havaittiin valaisevan pimeää huonetta niin paljon, ettei kirkkautta haluttu kuitenkaan suurentaa.

5.4 Kellonajan tarkkuus

Kellonajan tarkkuus perustuu kiteen tarkkuuteen, jonka ajastamana aikaansaadaan keskeytys 5 ms:n välein. Keskeytysvälin pituus mitattiin lisäämällä aika-keskeytykseen käsky, jolla mikrokontrollerin yhden (tässä työssä tarvitsemattoman) I/O-linjan tila vaihdetaan jokaisella suorituskerralla. Keskeytysväliksi mitattiin oskilloskoopilla 5,00 ms. Tämän mittauksen tarkkuus ei kuitenkaan ole riittävä, ja kellonaikaa tarkkailemalla havaittiinkin kellon edistävän vuorokaudessa noin 4 sekuntia. Tämä johtuu kiteen taajuuden hienoisesta poikkeamasta ilmoitetusta taajuudesta.

Näin merkittävää edistämistä kellossa ei voida pitää hyväksyttävänä, joten tarkkuutta päätettiin korjata ohjelmallisesti. Tarkemman poikkeaman selvittämiseksi kellon edistäminen mitattiin kahdentoista vuorokauden aikana. Tulokseksi saatiin 53 sekuntia eli noin 4,42 sekuntia vuorokaudessa. Mittaaminen tapahtui vertaamalla kellonaikaa toisen kellon aikaan, jonka aika tahdistettiin radion aikamerkin mukaan. Menetelmään liittyvää inhimillistä virhemarginaalia pyrittiin pienentämään mittausjakson pituudella. Mittausvirheen arvioitiin olevan korkeintaan 1 s, eli 1,9 % mitatun edistyksen suuruudesta.

Kellonajan ohjelmallinen korjaus tapahtuu kahdessa osassa. Kymmenen minuutin välein kelloa siirretään taaksepäin kuusi viiden millisekunnin askelta, joka on siis kellon lyhin askel. Jäljelle jäänyttä virhettä korjataan siirtämällä kelloa vielä 10 askelta kaksi kertaa vuorokaudessa. Näillä muutoksilla mitatusta virheestä tulee korjattua 100,075 %. Korjauksen toteuttamiseksi vaihdettiin kello_5ms_laskuri-muuttujan laskusuunnaksi ylhäältä alaspäin. Tällöin kellon taaksepäin siirto tapahtuu laskuria suurentamalla, eikä negatiivisia lukuja ja suurempaa muuttujaa tarvita.

Tehdyn korjauksen jälkeen kellonajan tarkkuutta tutkittiin uudelleen. Yli kuukauden pituisen mittausjakson kuluttua kello oli edelleen oikeassa ajassa mittaustarkkuuden rajoissa. Tämän perusteella kellonajan tarkkuus oli siis halutulla tasolla.

6 YHTEENVETO

Tavoitteiden määrittelyn jälkeen suunnittelutyö alkoi keskeisten komponenttien ja kytkennän osien valinnoilla. Osa valinnoista oli selkeitä, ja varsinaisia ongelmia ei tässä vaiheessa ollut. Eniten aikaa kului paristokäytön ja näytönohjauksen kytkentöjen suunnitteluun. Toinen työläs vaihe oli kytkentäkaavioiden ja piirilevykuvien piirtäminen, mikä tehtiin EAGLE Layout Editor -ohjelmalla.

Ohjelman suunnittelu alkoi hahmottelemalla ohjelman rakennetta ensin karkeasti, minkä jälkeen edettiin tarkempiin yksityiskohtiin. Työläintä olivat alustukset ja muut mikrokontrollerin toimintaan liittyvät osat, jotka vaativat mikrokontrollerin datalehden tarkkaa tutkimista. Muilta osin ohjelma koostuu yksinkertaisista C-kielen perusrakenteista.

Ohjelmaa kirjoitettaessa havaittiin, että kellon toiminnallisuuden määritykset olivat aika puutteelliset. Iso osa kellon tarkasta käyttäytymisestä eri tilanteissa tuli valituksi vasta näitä ohjelmaan kirjoitettaessa. Kovin tarkkaa toiminnan ja ominaisuuksien määritystä ei tosin ollut tavoitteenakaan tehdä suunnittelun alussa. Tästä kuitenkin aiheutui lisätyötä ohjelmaa kirjoitettaessa. Yksityiskohtainen toiminnan suunnittelu, ennen ohjelman suunnittelua, olisi ollut parempi ratkaisu.

Lopuksi koottu kytkentä ja kirjoitettu ohjelma testattiin huolellisesti. Testauksessa paljastui muutamia virheitä niin kytkennässä kuin ohjelmassakin. Kaikki nämä saatiin kuitenkin ratkottua, ja joitain uusiakin ominaisuuksia toteutettiin vielä. Kun kaikki muutokset oli tehty, suoritettiin vielä uusintatestaus, jolla todettiin laitteen toimivan halutulla tavalla.

Työn lopputulokseksi saatiin alkuperäiset tavoitteet täyttävä ja ylittäväkin käyttövalmis herätyskello. Laitteen käyttäminen on vaivatonta, ja sen toiminta on luotettavaa. Pitkän ajan seurannan perusteella kello myös pysyy tarkasti oikeassa ajassa. Koteloinnin ja mahdollisten jatkokehityksien jälkeen laite tullaan ottamaan omaan käyttöön.

Testauksen yhteydessä havaittiin, että ympäristön valoisuuden vaihdellessa näyttö voi olla joko tarpeettoman kirkas tai liian himmeä. Jotta laite toimisi tältä osin paremmin, voisi näytölle tehdä vaikkapa kolme kirkkaustasoa, jotka vaihtuisivat automaattisesti huoneen valoisuuden mukaan. Tämä on toteutettavissa ohjelmalla ja yksinkertaisella valoon reagoivaan vastukseen perustuvalla kytkennällä, ja näin todennäköisesti tullaan tekemäänkin. Ohjelmallisesti laitteeseen voidaan toteuttaa myös paljon muitakin lisäominaisuuksia, kuten vaihdettava hälytysääni, kunhan tämä ei vain tapahdu käytettävyyden kustannuksella.

LÄHTEET

[Atmel] ATmega88 mikrokontrollerin datalehti (versio M), 19.11.2007
http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf

[avrd-gui] graafinen käyttöliittymä avrdude ohjelmalle, 27.11.2007
http://sourceforge.net/projects/avrdude-gui

[CadSoft] piirilevysuunnitteluohjelma, 19.11.2007
http://www.cadsoftusa.com

[Duracell] 9V:n pariston datalehti, 22.8.2007
http://www.duracell.com/oem/Pdf/new/MN1604_US_CT.pdf

[L4941] L4941 jänniteregulaattorin datalehti, 19.11.2007
http://www.st.com/stonline/products/literature/ds/2142/l4941.pdf

[LM385] LM385-2.5 jännitereferenssin datalehti, 10.12.2007
http://www.national.com/ds.cgi/LM/LM185-2.5.pdf

[Micros] WinAVR-paketin asennusohjeet, 25.10.2007
http://www.microsalo.com/WinAVR_asennus.html

[makef] makefile tiedostopohja AVR-mikrokontrollereille, 25.10.2007
http://www.microsalo.com/avr-code/avr_makefile

[Partco] Partco Oy, 19.11.2007
http://www.partco.fi

[Pros] Pentti Vahtera: AVR sulauttaa bitit
Prosessori-lehti 4/2006, Sanoma Magazines Finland

[SP-Ele] SP-Elektroniikka Oy, 19.11.2007
http://www.spelektroniikka.fi

[TietoP] TietoPetri Oy, 19.11.2007
http://www.tietopetri.fi

[Vahtera] Pentti Vahtera: Mikro-ohjaimen ohjelmointi C-kielellä
WS Bookwell Oy, Porvoo 2003, 338 sivua

[WinAVR] C-kääntäjä AVR-mikrokontrollereille, 27.11.2007
http://winavr.sourceforge.net

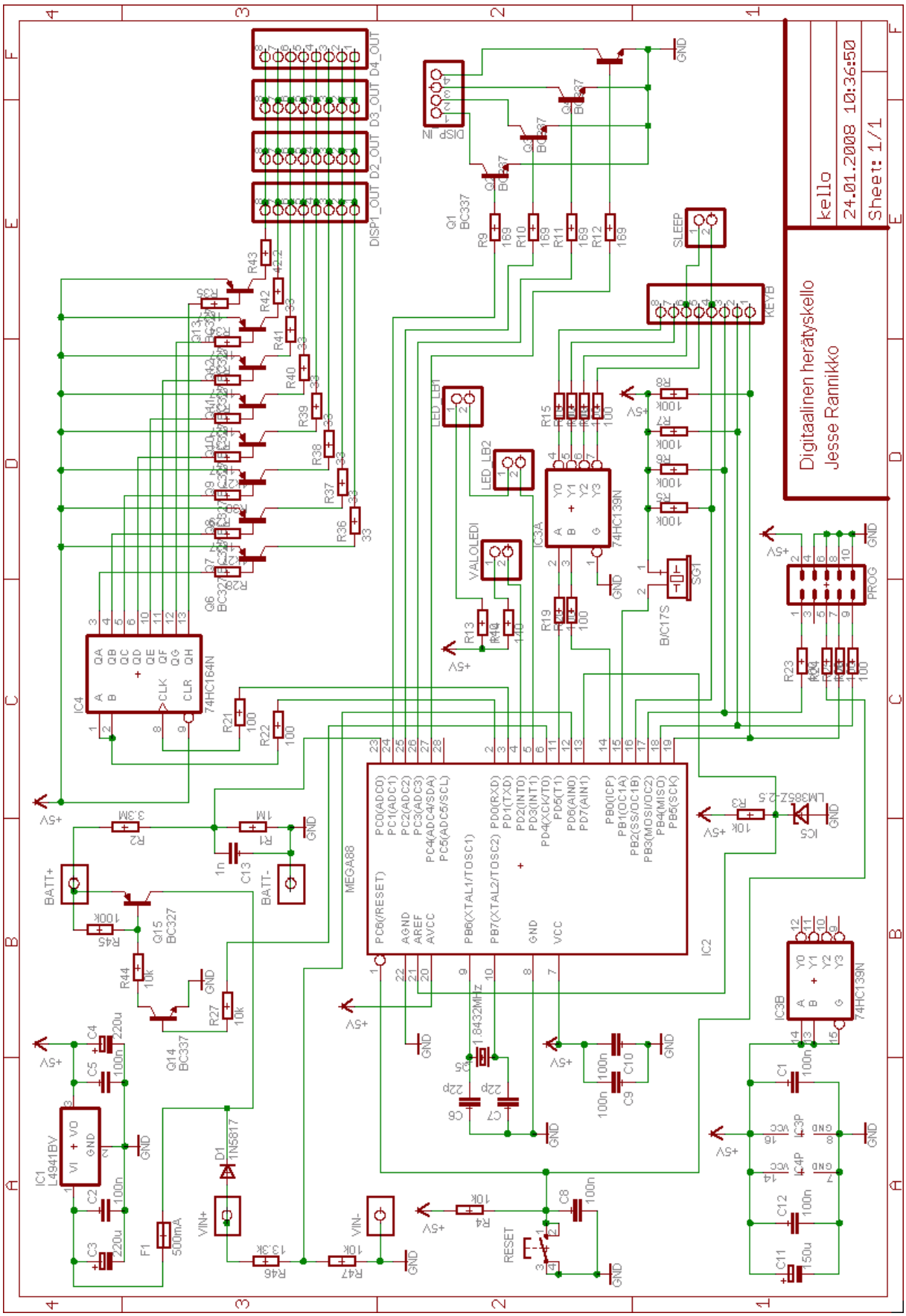
[yaap] STK200-ohjelmointikaapelin kokoamisohje, 26.11.2007
http://www.myplace.nu/avr/yaap/if_dongle.gif

LIITTEET

Liite 1. pääpiirilevyn kytkentäkaavio, 1 sivu

Liite 2. ohjelman lähdekoodi, 28 sivua

Liite 3. mikrokontrollerin datalehden viitatut sivut, 10 sivua


Digitaalinen herätyskello
Jesse Rannikko
kello
24.01.2008 10:36:50
Sheet: 1/1
MEGA88
74HC164N
74HC139N
L4941BV
1N5817
BC337
BC327
1.8432MHz
LM385Z-2.5
DISP1_OUT D2_OUT D3_OUT D4_OUT
KEYB
PROG
SLEEP
RESET
BATT+
BATT-
VIN+
VIN-
+5V
GND

```
/*****************************************************************************/
/* Projekti:   Digitaalinen herätyskello ATmega88-kontrollerilla             */
/* Versio:     1.0                                                           */
/* Päivämäärä: 24.1.2008                                                     */
/* Tekijä:     Jesse Rannikko                                                */
/*****************************************************************************/

/*
Muuttujien ja funktioiden nimissä ja kommentoinneissa käytetyillä termeillä
herätys, hälytys ja säätö tarkoitetaan seuraavaa:
-herätys viittaa koko herätystapahtumaan sen laukeamisesta siihen, kun se
 katkaistaan on/off-kytkimestä
-hälytys viittaa hälytysäänen soimiseen eli erotuksena edelliseen hälytys
 loppuu jo silloin, kun torkkunäppäintä painetaan
-säätö viittaa käyttäjän tekemään herätys- tai kellonajan säätöön tai menussa
 olevien säätöjen tekemiseen
*/


/********************* kirjasto-tiedostojen sisällytykset ********************/
#include <avr/io.h>
#include <avr/interrupt.h>


/************************** vakioiden määritykset ***************************/

#define TYHJA 0xff

//kirjain-näppäimien numerovastaavuudet
#define NAPPI_A 10
#define NAPPI_B 11
#define NAPPI_C 12
#define NAPPI_D 13
#define NAPPI_E 14
#define NAPPI_F 15
//lisäksi käytössä yllä määritelty TYHJA

//näytölle kirjoitettavien kirjaimien numerovastaavuudet
#define KIRJAIN_A 10
#define KIRJAIN_B 11
#define KIRJAIN_D 13
#define KIRJAIN_E 14
#define KIRJAIN_L 21
#define KIRJAIN_P 25
#define KIRJAIN_S 28
#define KIRJAIN_T 29
#define KIRJAIN_Y 34
//lisäksi käytössä yllä määritelty TYHJA

//näytön tilojen numerovastaavuudet
#define NTILA_NORMAALI 0 //näytöllä on kellonaika
#define NTILA_H1_SAATO 1 //herätyksen 1 ajan säätö
#define NTILA_H2_SAATO 2 //herätyksen 2 ajan säätö
#define NTILA_K_SAATO  3 //kellonajan säätö
#define NTILA_PALUU    4 //odotetaan näytönpalautusta, eikä reagoida näppäimiin
#define NTILA_MENU_1   5 //menun 1. kohta
#define NTILA_MENU_TYP 5 //herätyksen 2 tyyppi
#define NTILA_MENU_SLP 6 //torkun pituus
#define NTILA_MENU_LED 7 //valoledi on/off
#define NTILA_MENU_ADC 8 //ADC:n käynnistys
#define NTILA_MENU_V   8 //menun viimeinen kohta

//tilamuuttujien tilojen numerovastaavuudet
#define KYLLA     1
#define EI        0
#define ON        1 //herätyksien on/off-kytkimen tila
#define OFF       0
```

```
#define LOISTAA   1 //ledin tila
#define EI_LOISTA 0
#define KERRAN    1 //heratyksen2_tyyppi
#define JATKUVA   0
#define KORKEA    1 //soitettava taajuus
#define MATALA    0

//kaytto_kytkimet-taulukon kytkimien indeksit
#define HERAT_APU 0
#define HERATYS1  1
#define HERATYS2  2

//heratys_ajat taulukon eri aikojen alkuindeksit
#define AIKA_APU 0
#define AIKA_H1  4
#define AIKA_H2  8

//puskuri8-taulukon ledien indeksit
#define LEDI_HERATYS1 3
#define LEDI_HERATYS2 2
#define LEDI_PISTE1   1
#define LEDI_PISTE2   0


/************************** makrojen määritykset ***************************/

//asettaa portin/muuttujan yksittäisen bitin
#define SET_BIT(port, bit) ( (port) |= (_BV(bit)) )

//nollaa portin/muuttujan yksittäisen bitin
#define CLEAR_BIT(port, bit) ( (port) &= ~(_BV(bit)) )

//irrottaa timer1:n vertailuosuman ulostulon kytkennän PB1-pinniin
#define IRROITA_KAIUTIN() (TCCR1A = 0)


/************************** muuttujien alustukset **************************/

//muuttujan mahdolliset arvot; arvojen tai muuttujan selitys
volatile unsigned char lippu_5ms = EI; //EI(0), KYLLA(1); KYLLA=5ms on kulunut
volatile unsigned char lippu_kello = EI; //EI(0), KYLLA(1); KYLLA=kellonaika
    //(minuutit) on päivittynyt

//kaikkien laskuri-muuttujien yksi askel on 5ms (joka saadaan keskeytyksestä)
unsigned char laskuri_10ms; //0-2(3); laskee (ylhäältä alas) 10ms:n kulumisen,
    //alustetaan käytön yhteydessä
unsigned char laskuri_50ms = 10; //0-10; laskee (ylhäältä alas) 50ms:n
    //kulumisen
unsigned char laskuri_200tai300ms; //0-60(61); laskee (ylhäältä alas) 200ms:n
    //tai 300ms:n kulumisen, alustetaan käytön yhteydessä
unsigned int laskuri_5sek; //0-1000(1001); laskee (ylhäältä alas) 5s:n
    //kulumisen, alustetaan käytön yhteydessä
unsigned int laskuri_10sek = 0; //0-2000(2001); laskee (ylhäältä alas) 10s:n
    //kulumisen, 0=ei vielä käytössä
unsigned int laskuri_2min; //0-24000; laskee (ylhäältä alas) 2 minuutin
    //kulumisen, alustetaan käytön yhteydessä

volatile unsigned char kello_10tunnit = 0; //0-2; kellonajan kymmenet tunnit
volatile unsigned char kello_tunnit = 0; //0-9(10); tunnit
volatile unsigned char kello_10minuutit = 0; //0-5(6); kymmenet minuutit
volatile unsigned char kello_minuutit = 0; //0-9(10); minuutit
volatile unsigned char kello_sekunnit = 0; //0-59(60); sekunnit
volatile unsigned char kello_5ms_laskuri = 0; //0-200(214); laskee (ylhäältä
    //alas) yhden sekunnin kulumisen, yksi askel on 5ms

unsigned char puskuri[] = {0, 0, 0, 0}; //nelipaikkainen näyttöpuskuri, jonka
    //sisältö pidetään näytöllä
```

```
volatile unsigned char puskuri8[] = {0, 0, 0, 0}; //8. bittien puskuri, jolla
    //ohjataan herätyksien merkkiledejä ja kellon kaksoispistettä
unsigned char heratys_ajat[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    //taulukossa on kolme kertaa seuraavat aika-muuttujat: minuutit (paikat 0,
    //4 ja 8), kymmenet minuutit (1, 5 ja 9), tunnit (2, 6 ja 10) ja kymmenet
    //tunnit (3, 7 ja 11), näistä ensimmäiset ovat apu-aika, toiset herätyksen
    //1 aika ja viimeiset herätyksen 2 aika
unsigned char kaytto_kytkimet[] = {OFF, OFF, OFF}; //OFF(0), ON(1); herätyksen
    //aputoiminnon, herätyksen 1 ja herätyksen 2 on/off-kytkin
unsigned char edelliset[] = {0xff, 0xff, 0xff, 0xff}; //taulukon neljän
    //alkion bitteihin 2-5 on tallennettuna 4x4-näppäimistön näppäinten
    //edelliset painallukset, 1=näppäintä ei ole painettu

unsigned char ind_saato; //0-3; käytetään indeksinä säätötoimenpiteissä
    //(heratys_ajat- ja puskuri-taulukoille), alustetaan käytön yhteydessä
unsigned char ind_puskuri = 3; //0-3; puskuri-taulukon indeksi, 3=vasen numero

unsigned char naytontila = NTILA_NORMAALI; //0-8(9); ilmaisee mikä teksti eri
    //vaihtoehdoista (tarkemmin vakioiden määrityksessä) näytöllä on
unsigned char heratyksen2_tyyppi = JATKUVA; //JATKUVA(0), KERRAN(1); ilmaisee
    //herätyksen 2 hälytyksen soittotavan
unsigned char torkun_pituus = 6; //5-9; ilmaisee torkkuajan pituuden (min.)
unsigned char valoledi_mahdollinen = KYLLA; //EI(0), KYLLA(1); KYLLA=valoledi
    //on kytkettävissä päälle
volatile unsigned char paristokaytto = EI; //EI(0), KYLLA(1); EI=käyttöjännite
    //ei tule paristosta (vaan se tulee verkkosähköstä)

unsigned char kellonaika_on_asetettu = EI; //EI(0), KYLLA(1); EI=käyttäjä ei
    //ole vielä asettanut kellonaikaa

unsigned char soi; //EI(0), KYLLA(1); KYLLA=hälytysääni soi
unsigned char tauko; //EI(0), KYLLA(1); KYLLA=hälytysäänessä on tauko (200ms)
unsigned char taajuus; //MATALA(0), KORKEA(1); ilmaisee kumpi taajuuksista on
    //käytössä
unsigned char voimakkaampi_halytys; //EI(0), KYLLA(1); KYLLA=voimakkaampi
    //hälytys on käytössä
unsigned char halyttaa = EI; //EI(0), HERATYS1(1), HERATYS2(2); ilmaisee onko
    //hälytyksen soittaminen käynnissä
unsigned char heratys_kaynnissa = EI; //EI(0), HERATYS1(1), HERATYS2(2);
    //ilmaisee onko herätystapahtuma käynnissä (erotuksena edelliseen osoittaa
    //herätystä myös torkun aikana)

unsigned char apu; //apumuuttuja
unsigned char apuk; //komparaattorikeskeytyksessä käytettävä apumuuttuja
unsigned char merkki; //näppäimistönluvun apumuuttuja
unsigned char i; //silmukkamuuttuja


/************************* funktioiden prototyypit *************************/
void AlustaAvr(void);
void PalautaNaytontila(void);
void KirjoitaKelloPuskuriin(void);
void AsetaMerkkileditKytkintenMukaan(void);
void TarkistaHeratykset(void);
unsigned char VertaaKelloJaHeratysaika(unsigned char indeksi);
void KaynnistaHalytys(unsigned char heratys);
void AsetaApuAika(unsigned char lisays);
void KaynnistaKaiutin(void);
void SuoritaHeratyksenApuToiminto(void);
void SammutaHeratys(void);
void PaivitaNaytto(void);
void MaaritaBittijono(void);
void AjoitaHalytys(void);
void VaihdaTaajuus(void);
void TauotaSoitto(void);
void JaksotaHalytys(void);
void SammutaValoledi(void);
```

```
void KasitteleNappaimisto(void);
void LueNappaimisto(void);
void MaaritaLuettuNappain(void);
void ReagoiPainettuunNappaimeen(void);
void ReagoiOnOff(unsigned char heratys);
void Reagoi3(void);
void ReagoiHeratysajanSaato(unsigned char heratys);
void ReagoiKello(void);
void ReagoiMenu(void);
void ReagoiNumero(void);
unsigned char TarkistaNumeronLaillisuus(void);


/*****************************************************************************/
/* Pääohjelma.                                                               */
/*****************************************************************************/
int main(void)
{
    //alustetaan mikrokontrolleri
    AlustaAvr();

    while (1)
    {
        //jos 5 millisekuntia on kulunut (muuttuja asetetaan keskeytyksessä)
        if (lippu_5ms == KYLLA)
        {
            //"nollataan" 5ms:n merkkilippu
            lippu_5ms = EI;

            //palautetaan näytölle kellonaika, mikäli edellisestä näppäimen
            //painalluksesta on kulunut 10 sekuntia
            PalautaNaytontila();

            //jos kellonaika on muuttunut ja mikään säätö ei ole käynnissä
            if ((lippu_kello == KYLLA) && (naytontila == NTILA_NORMAALI))
            {
                //"nollataan" kellonajan merkkilippu
                lippu_kello = EI;

                //kirjoitetaan päivittynyt kellonaika näyttöpuskuriin
                KirjoitaKelloPuskuriin();

                //laukaistaan herätykset, mikäli niiden on aika laueta
                TarkistaHeratykset();
            }

            //asetetaan näytölle loistamaan (5 millisekunniksi) vuorossa oleva
            //numero
            PaivitaNaytto();

            //suoritetaan hälytykseen liittyvät ajastetut toimenpiteet, mikäli
            //hälytys on käynnissä
            AjoitaHalytys();

            //sammutetaan valoledi, mikäli sen on aika sammua
            SammutaValoledi();

            //luetaan näppäimistö (50ms välein) ja reagoidaan siihen
            KasitteleNappaimisto();
        }
    }
}


/*****************************************************************************/
/* Funktio suorittaa resetin jälkeen tarvittavat AVR:n alustukset.           */
/*****************************************************************************/
```

```
void AlustaAvr(void)
{
    /************************** porttien alustukset **************************/

    //PB0 ja PB1 lähdöiksi
    DDRB = 0b00000011;
    //PC1-PC5 lähdöiksi
    DDRC = 0b00111110;
    //PD0-PD5 lähdöiksi
    DDRD = 0b00111111;
    //PD2 ja PD3 käyttöjännitteeseen eli sammutetaan valo- ja paristoledit
    PORTD = 0b00001100;


    /************************** timer0:n alustukset **************************/
    //timer0:aa käytetään generoimaan keskeytys 5ms välein

    //asetetaan timer tilaan, jossa se nollataan vertailuosumalla (ja jätetään
    //lähtö kytkemättä I/O-pinniin)
    TCCR0A = 0b00000010;
    //asetetaan kellon esijakajaksi 64
    TCCR0B = 0b00000011;
    //asetetaan timerin suurimmaksi arvoksi 143, jolloin keskeytys tapahtuu 144
    //kellojakson välein eli tapahtumaväli on 1/(1.8432MHz/64)*144=5ms
    OCR0A = 143;
    //sallitaan vertailuosuma-keskeytys
    TIMSK0 = 0b00000010;


    /************************** timer1:n alustukset **************************/
    //timer1:tä käytetään lähtöön PB1 kytketyn pietsoelementin soittamiseen
    //halutulla taajuudella

    //asetetaan timer tilaan, jossa se nollataan vertailuosumalla, ja
    //käynnistetään kello ilman jakajaa
    TCCR1B = 0b00001001;
    //vertailuosumalla vaihdetaan lähdön PB1 tilaa
    TCCR1A = 0b01000000;


    /************************ komparaattorin alustukset ************************/
    //komparaattoria käytetään verkkosähkön katkeamisen tunnistamiseen

    //sammutetaan digitaali-puskuri analogisista tuloista
    DIDR1 = 0b00000011;
    //sallitaan komparaattorin keskeytys (ja jätetään keskeytys tulemaan
    //komparaattorin ulostulon jokaisella muutoksella)
    ACSR = 0b00001000;


    /************************ AD-muuntimen alustukset ************************/
    //AD-muunninta käytetään varmennuspariston jännitemittaukseen

    //tasataan muunnostulos vasempaan reunaan (ja jätetään referenssiksi AREF-
    //pinni ja kanavaksi ADC0)
    ADMUX = 0b00100000;
    //sammutetaan digitaali-puskuri analogisesta tulosta
    DIDR0 = 0b00000001;
    //sallitaan ADC, käynnistetään hylättävä muunnos ja asetetaan kellon
    //esijakajaksi 32
    ADCSRA = 0b11000101;
    //odotetaan muunnoksen valmistuminen
    loop_until_bit_is_clear(ADCSRA, ADSC);
    //sallitaan ADC-keskeytys ja käynnistetään uusi muunnos (jonka tulos
    //käsitellään keskeytyspalvelussa)
    ADCSRA = 0b11001101;
```

```
    //globaali keskeytysten sallinta
    sei();
}


/*****************************************************************************/
/* Funktio palauttaa näytöntilan normaaliksi ja kellonajan näyttöpuskuriin,  */
/* jos edellisestä näppäimen painalluksesta on kulunut 10 sekuntia.          */
/*****************************************************************************/
void PalautaNaytontila(void)
{
    //jos näytönpalautuslaskuri on käynnissä
    if (laskuri_10sek != 0)
    {
        //vähennetään laskurista yksi 5ms askel
        laskuri_10sek--;

        //jos 10 sekuntia on kulunut
        if (laskuri_10sek == 0)
        {
            //jätetään laskuri nollaksi, koska palautus suoritetaan nyt

            //palautetaan näytöntilaksi normaalitila
            naytontila = NTILA_NORMAALI;

            //palautetaan kellonaika näyttöpuskuriin
            KirjoitaKelloPuskuriin();

            //ja asetetaan 8. bitit oikein eli herätyksien merkkiledit
            //kytkimen mukaan (kaksoispiste palautuu normaaliksi seuraavassa
            //aikakeskeytyksessä)
            AsetaMerkkileditKytkintenMukaan();
        }
    }
}


/*****************************************************************************/
/* Funktio kopioi nykyisen kellonajan näyttöpuskuriin.                       */
/*****************************************************************************/
void KirjoitaKelloPuskuriin(void)
{
    puskuri[3] = kello_10tunnit;
    puskuri[2] = kello_tunnit;
    puskuri[1] = kello_10minuutit;
    puskuri[0] = kello_minuutit;
}


/*****************************************************************************/
/* Funktio joko sytyttää tai sammuttaa herätyksien merkkiledit niiden        */
/* kytkinten mukaan.                                                         */
/*****************************************************************************/
void AsetaMerkkileditKytkintenMukaan(void)
{
    if (kaytto_kytkimet[HERATYS1] == ON)
    {
        puskuri8[LEDI_HERATYS1] = LOISTAA;
    }
    else
    {
        puskuri8[LEDI_HERATYS1] = EI_LOISTA;
    }

    if (kaytto_kytkimet[HERATYS2] == ON)
    {
```

```
        puskuri8[LEDI_HERATYS2] = LOISTAA;
    }
    else
    {
        puskuri8[LEDI_HERATYS2] = EI_LOISTA;
    }
}


/*****************************************************************************/
/* Funktio laukaisee jommankumman herätyksen, jos sen on aika laueta ja jos  */
/* käyttäjä on ko. herätyksen sallinut on/off-kytkimellä. Lisäksi            */
/* suoritetaan herätyksen aputoiminto, jos sen on aika tapahtua.             */
/*****************************************************************************/
void TarkistaHeratykset(void)
{
    //ei laukaista uutta herätystä, jos herätys on jo käynnissä
    if (heratys_kaynnissa == EI)
    {
        //jos herätyksen 1 kytkin sallii ja nyt on herätyksen 1 aika
        if ((kaytto_kytkimet[HERATYS1] == ON) &&
            VertaaKelloJaHeratysaika(AIKA_H1))
        {
            //käynnistetään herätys 1
            KaynnistaHalytys(HERATYS1);
        }
        //jos herätys 1 ei lauennut, tutkitaan herätys 2
        else if ((kaytto_kytkimet[HERATYS2] == ON) &&
                  VertaaKelloJaHeratysaika(AIKA_H2))
        {
            //ja tarvittaessa käynnistetään herätys 2
            KaynnistaHalytys(HERATYS2);
        }
    }
    //jos herätyksen aputoiminto odottaa suoritustaan ja nyt on sen aika
    if ((kaytto_kytkimet[HERAT_APU] == ON) &&
        VertaaKelloJaHeratysaika(AIKA_APU))
    {
        //suoritetaan herätyksen aputoiminto
        SuoritaHeratyksenApuToiminto();
    }
}


/*****************************************************************************/
/* Funktio vertaa nykyistä kellonaikaa ja herätysaikaa toisiinsa ja          */
/* palauttaa arvon KYLLA(1), jos ajat ovat samat tai arvon EI(0), jos ajat   */
/* ovat erisuuret.                                                           */
/*   Parametrin "indeksi" mahdolliset arvot ovat AIKA_APU(0), AIKA_H1(4) ja  */
/* AIKA_H2(8), ja se ilmaisee mitä heratys_ajat-taulukon aikaa tutkitaan.    */
/*****************************************************************************/
unsigned char VertaaKelloJaHeratysaika(unsigned char indeksi)
{
    //jos kellonaika ja herätysaika ovat samat, palautetaan tosi
    if ((heratys_ajat[indeksi] == kello_minuutit) &&
        (heratys_ajat[indeksi + 1] == kello_10minuutit) &&
        (heratys_ajat[indeksi + 2] == kello_tunnit) &&
        (heratys_ajat[indeksi + 3] == kello_10tunnit))
    {
        return KYLLA;
    }
    //muuten palautetaan epätosi
    else
    {
        return EI;
    }
}
```

```
/*****************************************************************************/
/* Funktio käynnistää hälytyksen eli aloittaa kaiuttimen soittamisen sekä    */
/* asettelee herätykseen liittyvät muuttujat ja aputoiminnon ajastuksen.     */
/*   Parametrin "heratys" mahdolliset arvot ovat HERATYS1(1) ja HERATYS2(2), */
/* ja se ilmaisee mikä herätys pitää käynnistää.                             */
/*****************************************************************************/
void KaynnistaHalytys(unsigned char heratys)
{
    //asetetaan herätys-muuttujiin herätystä vastaava arvo
    halyttaa = heratys;
    heratys_kaynnissa = heratys;

    //aloitetaan soittamalla hälytystä (viiden sekunnin ajan)
    soi = KYLLA;

    //ei aloiteta tauolla
    tauko = EI;

    //ei aloiteta voimakkaalla hälytyksellä
    voimakkaampi_halytys = EI;

    //aloitetaan korkealla taajuudella
    taajuus = KORKEA;

    //alla oleviin laskureihin on asetettava yksi askel liikaa, koska kutakin
    //vähennetään jo kerran (eli AjoitaHalytys-funktio suoritetaan) ennen kuin
    //seuraava 5ms on kulunut

    //asetetaan laskuriin 5s (1000*5ms)
    laskuri_5sek = 1001;

    //asetetaan laskuriin 300ms (60*5ms)
    laskuri_200tai300ms = 61;

    //asetetaan laskuriin 10ms (2*5ms)
    laskuri_10ms = 3;

    //asetetaan kovemman hälytyksen laukeaminen minuutin päähän
    AsetaApuAika(1);

    //käynnistetään kaiutinlähtö
    KaynnistaKaiutin();
}


/*****************************************************************************/
/* Funktio kirjoittaa heratys_ajat-taulukkoon herätyksen aputoiminnon        */
/* tapahtumisajan.                                                           */
/*   Parametrin "lisays" arvo on välillä 1-10 ja se ilmaisee minuuttimäärän, */
/* jonka kuluttua aputoiminnon halutaan tapahtuvan.                          */
/*****************************************************************************/
void AsetaApuAika(unsigned char lisays)
{
    //tehdään lisäys ja kopioidaan kellonaika apu-ajaksi
    heratys_ajat[0] = kello_minuutit + lisays;
    heratys_ajat[1] = kello_10minuutit;
    heratys_ajat[2] = kello_tunnit;
    heratys_ajat[3] = kello_10tunnit;

    //korjataan numerot lailliseksi kellonajaksi
    if (heratys_ajat[0] > 9) //minuutit
    {
        heratys_ajat[0] = heratys_ajat[0] - 10;
        heratys_ajat[1]++;
        if (heratys_ajat[1] == 6) //kymmenet minuutit
```

```
        {
            heratys_ajat[1] = 0;
            heratys_ajat[2]++;
            if (heratys_ajat[2] == 10) //tunnit
            {
                heratys_ajat[2] = 0;
                heratys_ajat[3]++;
            }
            //jos tunneiksi tulee 24, niin korjataan se 00:ksi
            if ((heratys_ajat[3] == 2) && (heratys_ajat[2] == 4))
            {
                heratys_ajat[3] = 0;
                heratys_ajat[2] = 0;
            }
        }
    }
    //asetetaan herätyksen aputoiminto käyttöön
    kaytto_kytkimet[HERAT_APU] = ON;
}


/*****************************************************************************/
/* Funktio käynnistää kaiuttimen soittamaan joko matalaa tai korkeaa         */
/* taajuutta taajuus-muuttujan mukaan.                                       */
/*****************************************************************************/
void KaynnistaKaiutin(void)
{
    //Timerin lähtöpinni vaihtaa tilaansa kun timer saavuttaa arvon OCR1A-
    //rekisterissä. Kaiuttimen taajuus siis määrätään tämän rekisterin avulla
    //ja sen arvo voidaan laskea seuraavalla kaavalla: OCR1A=f(osc)/(2*f)-1,
    //missä f(osc) on kidetaajuus 1,8432MHz ja f on haluttu taajuus.

    //kytketään timerin lähtö I/O-pinniin (PB1)
    TCCR1A = 0b01000000;

    //nollataan timer
    TCNT1 = 0;

    //ja asetetaan timerin maksimiarvo taajuuden mukaan
    if (taajuus == KORKEA)
    {
        //asetetaan taajuudeksi 500Hz
        OCR1A = 1842;
    }
    else
    {
        //asetetaan taajuudeksi 200Hz
        OCR1A = 4607;
    }
}


/*****************************************************************************/
/* Funktio suorittaa jonkin seuraavista toimenpiteistä: kovemman hälytyksen  */
/* laukaisu, torkkuhälytyksen laukaisu tai herätyksen automaattinen          */
/* sammutus.                                                                 */
/*****************************************************************************/
void SuoritaHeratyksenApuToiminto(void)
{
    //jos hälytysääni on soimassa
    if (halyttaa != EI)
    {
        //funktiota käytetään herätyksen automaattiseen sammuttamiseen (jos
        //voimakkaampi hälytys on jo käytössä)
        if (voimakkaampi_halytys == KYLLA)
        {
            //sammutetaan herätys
```

```
            SammutaHeratys();
            //ja poistetaan herätyksen aputoiminto käytöstä
            kaytto_kytkimet[HERAT_APU] = OFF;
        }
        //tai kovemman hälytyksen laukaisuun
        else
        {
            //lopetetaan 5s jaksotus hälytysäänessä
            voimakkaampi_halytys = KYLLA;
            //ja asetetaan hälytysääni soimaan
            soi = KYLLA;
            //asetetaan automaattinen herätyksen sammuttaminen 9 minuutin
            //päähän
            AsetaApuAika(9);
        }
    }
    //muuten (kun hälytysääni ei ole soimassa) funktiota käytetään
    //torkkuhälytyksen laukaisemiseen
    else
    {
        //käynnistetään hälytys
        KaynnistaHalytys(heratys_kaynnissa);
    }
}


/*****************************************************************************/
/* Funktio sammuttaa herätyksen eli sammuttaa kaiuttimen ja "nollaa"         */
/* herätykseen liittyvät muuttujat.                                          */
/*****************************************************************************/
void SammutaHeratys(void)
{
    //sammutetaan kaiutin
    IRROITA_KAIUTIN();

    //merkitään ettei hälytysääni ole soimassa (ja lopetetaan AjoitaHalytys-
    //funktion toiminnot)
    halyttaa = EI;

    //merkitään ettei herätys ole enää käynnissä
    heratys_kaynnissa = EI;

    //poistetaan herätyksen aputoiminto käytöstä
    kaytto_kytkimet[HERAT_APU] = OFF;

    //asetetaan herätyksien merkkiledit oikein
    AsetaMerkkileditKytkintenMukaan();
}


/*****************************************************************************/
/* Funktio asettaa loistamaan (5 millisekunniksi) vuorollaan yhden neljästä  */
/* näytön numerosta, jos laiteen käyttöjännite tulee verkkosähköstä eli      */
/* silloin, kun paristokaytto = EI(0).                                       */
/*****************************************************************************/
void PaivitaNaytto(void)
{
    //jos käyttöjännite tulee verkkosähköstä
    if (paristokaytto == EI)
    {
        //tallennetaan vuorossa oleva numero merkki-muuttujaan
        merkki = puskuri[ind_puskuri];

        //500-1000ms aikana
        if (kello_5ms_laskuri <= 100)
        {
            //jos näytöllä on kellonaika ja sitä ei ole vielä asetettu
```

```
            if ((naytontila == NTILA_NORMAALI) &&
                (kellonaika_on_asetettu == EI))
            {
                //ei näytetä numeroita (eli vilkutetaan kellonaikaa)
                merkki = TYHJA;
            }
            //jos jokin säätö on käynnissä, vilkutetaan vuorossa olevaa numeroa
            if ((ind_puskuri == ind_saato) && !((naytontila == NTILA_NORMAALI)
                || (naytontila == NTILA_PALUU)
                || (naytontila == NTILA_MENU_ADC)))
            {
                merkki = TYHJA;
            }
        }
        //tallennetaan merkki-muuttujaan numeron ja 8. bitin bittijono
        MaaritaBittijono();

        //sammutetaan kaikki ohjaustransistorit
        PORTC = 0;

        //lähetetään bittijono (8 bittiä) siirtorekisteriin
        for (i = 8; i != 0; i--)
        {
            //kirjoitetaan datalähtöön (PD0) vuorossa oleva bitti
            if (merkki & 0b00000001)
            {
                SET_BIT(PORTD, PD0);
            }
            else
            {
                CLEAR_BIT(PORTD, PD0);
            }

            //kirjoitetaan kellolähtöön (PD1) kellopulssi
            SET_BIT(PORTD, PD1);
            CLEAR_BIT(PORTD, PD1);

            //siirretään seuraava bitti käsiteltäväksi
            merkki = merkki >> 1;
        }

        //ohjataan oikea transistori (PC1-PC4) johtavaksi
        SET_BIT(PORTC, (ind_puskuri + 1));

        //asetetaan indeksiksi seuraava numero (seuraavaa kutsukertaa varten)
        if (ind_puskuri == 0)
        {
            ind_puskuri = 3;
        }
        else
        {
            ind_puskuri--;
        }
    }
}


/*****************************************************************************/
/* Funktio määrittää näytönohjaukseen käytetyn bittijonon merkki-muuttujaan  */
/* tallennetun numeron mukaan. Lisäksi kahdeksanneksi bitiksi tallennetaan   */
/* puskuri8-taulukosta oikea arvo ind_puskuri-muuttujan mukaan. Valmis       */
/* bittijono tallennetaan merkki-muuttujaan.                                 */
/*****************************************************************************/
void MaaritaBittijono(void)
{
    //tallennetaan bitit näytettävän merkin mukaan
    switch (merkki)
```

```
{
    case 0:
        if ((ind_puskuri == 3) && (naytontila == NTILA_NORMAALI))
        {
            merkki = ~0b00000000; //kellonajan vasemmassa numerossa ei
                                  //näytetä nollaa
        }
        else
        {
            merkki = ~0b11111100; //segmentit a,b,c,d,e,f
        }
        break;

    case 1:
        merkki = ~0b01100000; //segmentit b,c
        break;

    case 2:
        merkki = ~0b11011010; //segmentit a,b,d,e,g
        break;

    case 3:
        merkki = ~0b11110010; //segmentit a,b,c,d,g
        break;

    case 4:
        merkki = ~0b01100110; //segmentit b,c,f,g
        break;

    case 5:
        merkki = ~0b10110110; //segmentit a,c,d,f,g
        break;

    case 6:
        merkki = ~0b10111110; //segmentit a,c,d,e,f,g
        break;

    case 7:
        merkki = ~0b11100000; //segmentit a,b,c
        break;

    case 8:
        merkki = ~0b11111110; //segmentit a,b,c,d,e,f,g
        break;

    case 9:
        merkki = ~0b11110110; //segmentit a,b,c,d,f,g
        break;

    case KIRJAIN_A:
        merkki = ~0b11101110; //segmentit a,b,c,e,f,g
        break;

    case KIRJAIN_B:
        merkki = ~0b00111110; //segmentit c,d,e,f,g
        break;

    case KIRJAIN_D:
        merkki = ~0b01111010; //segmentit b,c,d,e,g
        break;

    case KIRJAIN_E:
        merkki = ~0b10011110; //segmentit a,d,e,f,g
        break;

    case KIRJAIN_L:
        merkki = ~0b00011100; //segmentit d,e,f
```

```
            break;

        case KIRJAIN_P:
            merkki = ~0b11001110; //segmentit a,b,e,f,g
            break;

        case KIRJAIN_S:
            merkki = ~0b10110110; //segmentit a,c,d,f,g
            break;

        case KIRJAIN_T:
            merkki = ~0b00011110; //segmentit d,e,f,g
            break;

        case KIRJAIN_Y:
            merkki = ~0b01110110; //segmentit b,c,d,f,g
            break;

        case TYHJA:
            merkki = ~0b00000000; //ei mitään
            break;
    }
    //jos kahdeksannen bitin mukaisen ledin halutaan loistavan, kirjoitetaan
    //sen kohdalle 0 (muuten siihen saa jäädä 1)
    if (puskuri8[ind_puskuri] == LOISTAA)
    {
        merkki = merkki & 0b11111110;
    }
}


/*************************************************************************/
/* Funktio huolehtii hälytysäänen soittamisen ajoituksista silloin, kun  */
/* hälytys on käynnissä, eli vaihtaa soitettavaa taajuutta 10ms välein,  */
/* tauottaa hälytysäänen 300ms:n ääneen ja 200m:n hiljaisuuteen ja       */
/* jaksottaa hälytyksen 5 sekunnin soittoon ja 5 sekunnin hiljaisuuteen. */
/*************************************************************************/
void AjoitaHalytys(void)
{
    //jos jompikumpi hälytys on käynnissä
    if (halyttaa != EI)
    {
        //jos soitto on käynnissä
        if (soi == KYLLA)
        {
            //vähennetään laskurista yksi 5ms askel
            laskuri_10ms--;
            //jos 10 millisekuntia on kulunut
            if (laskuri_10ms == 0)
            {
                //asetetaan laskuriin uudelleen 10ms (2*5ms)
                laskuri_10ms = 2;
                //vaihdetaan soitettavaa taajuutta
                VaihdaTaajuus();
            }
            //vähennetään laskurista yksi 5ms askel
            laskuri_200tai300ms--;

            //jos 200ms tai 300ms on kulunut
            if (laskuri_200tai300ms == 0)
            {
                //tauotetaan hälytysääni 300ms:n ääneen ja 200m:n hiljaisuuteen
                //(ja asetetaan laskurille uusi arvo)
                TauotaSoitto();
            }
        }
        //vähennetään laskurista yksi 5ms askel
```

```
        laskuri_5sek--;

        //jos 5 sekuntia on kulunut
        if (laskuri_5sek == 0)
        {
            //asetetaan laskuriin uudelleen 5s (1000*5ms)
            laskuri_5sek = 1000;
            //jaksotetaan hälytys 5 sekunnin soittoon ja 5 sekunnin
            //hiljaisuuteen
            JaksotaHalytys();
        }
    }
}


/*****************************************************************************/
/* Funktio vaihtaa kaiuttimen soittamaksi taajuudeksi korkean tai matalan    */
/* (mikäli kaiuttimen pitää soida).                                          */
/*****************************************************************************/
void VaihdaTaajuus(void)
{
    //jos kaiuttimen pitää tällä hetkellä soida
    if (tauko == EI)
    {
        //vaihdetaan taajuus
        taajuus = taajuus ^ 0b00000001;

        //ja käynnistetään kaiutinlähtö uudella taajuudella
        KaynnistaKaiutin();
    }
}


/*****************************************************************************/
/* Funktio tauottaa hälytysäänen 300ms:n ääneen ja 200m:n hiljaisuuteen.     */
/*****************************************************************************/
void TauotaSoitto(void)
{
    //kun edellinen aika oli 200ms
    if (tauko == KYLLA)
    {
        //asetetaan seuraavaksi ajaksi 300ms (60*5ms)
        laskuri_200tai300ms = 60;

        //lopetetaan tauko kaiuttimen soittamisessa
        tauko = EI;

        //ja käynnistetään kaiutinlähtö
        KaynnistaKaiutin();
    }
    //kun edellinen aika oli 300ms
    else
    {
        //asetetaan seuraavaksi ajaksi 200ms (40*5ms)
        laskuri_200tai300ms = 40;

        //aloitetaan tauko kaiuttimen soittamisessa
        tauko = KYLLA;

        //ja sammutetaan kaiutinlähtö
        IRROITA_KAIUTIN();
    }
}


/*****************************************************************************/
/* Funktio jaksottaa hälytyksen 5 sekunnin soittoon ja 5 sekunnin            */
```

```
/* hiljaisuuteen.                                                           */
/***************************************************************************/
void JaksotaHalytys(void)
{
    //jos käynnissä oleva herätys on herätys 2 ja sen tyyppinä on
    //kertasoitto
    if ((halyttaa == HERATYS2) && (heratyksen2_tyyppi == KERRAN))
    {
        //sammutetaan herätys (ensimmäisen 5 sekunnin jakson lopuksi)
        SammutaHeratys();
    }
    //muuten jaksotetaan hälytysääni 5s soittoihin ja 5s taukoihin, jos
    //voimakkaampi hälytys ei ole käytössä
    else if (voimakkaampi_halytys == EI)
    {
        //vaihdetaan soi-muuttujan tila
        soi = soi ^ 0b00000001;

        //ja käynnistetään tai sammutetaan kaiutin sen mukaan
        if (soi == KYLLA)
        {
            KaynnistaKaiutin();
        }
        else
        {
            IRROITA_KAIUTIN();
        }
    }
}


/***************************************************************************/
/* Funktio sammuttaa valoledin, jos on kulunut 2 minuutin sen syttymisestä.  */
/***************************************************************************/
void SammutaValoledi(void)
{
    //jos valoledi loistaa
    if (bit_is_clear(PIND, PIND2))
    {
        //vähennetään laskurista yksi 5ms askel
        laskuri_2min--;

        //jos 2 minuuttia on kulunut
        if (laskuri_2min == 0)
        {
            //sammutetaan valoledi
            SET_BIT(PORTD, PD2);
        }
    }
}


/***************************************************************************/
/* Funktio kutsuu 50ms välein näppäimistönlukufunktiota ja näppäimen         */
/* painallukseen reagoivaa funktiota.                                        */
/***************************************************************************/
void KasitteleNappaimisto(void)
{
    //vähennetään laskurista yksi 5ms askel
    laskuri_50ms--;

    //jos 50 millisekuntia on kulunut
    if (laskuri_50ms == 0)
    {
        //asetetaan laskuriin uudelleen 50ms (10*5ms)
        laskuri_50ms = 10;
```

```
        //luetaan näppäimistö
        LueNappaimisto();

        //reagoidaan mahdolliseen painettuun näppäimeen
        ReagoiPainettuunNappaimeen();
    }
}


/*****************************************************************************/
/* Funktio tutkii, onko jokin näppäin painettuna (painaminen aloitettu) ja   */
/* tallentaa merkki-muuttujaan näppäimen numeron tai arvon TYHJA, jos        */
/* mitään näppäintä ei ole painettu.                                         */
/*****************************************************************************/
void LueNappaimisto(void)
{
    //merkitään ettei mitään näppäintä ole painettuna
    merkki = TYHJA;

    //käsitellään kaikki 4 saraketta
    for (i = 0; i < 4; i++)
    {
        //asetetaan vuorossa olevan sarakkeen johdin maapotentiaaliin
        //dekooderi-piirillä (muut sarakkeet ovat käyttöjännitteessä)
        if (i == 0)
        {
            CLEAR_BIT(PORTD, PD5); //dekooderin ohjauksena 00
            CLEAR_BIT(PORTB, PB0);
        }
        else if (i == 1)
        {
            SET_BIT(PORTD, PD5); //dekooderin ohjauksena 01
        }
        else if (i == 2)
        {
            CLEAR_BIT(PORTD, PD5); //dekooderin ohjauksena 10
            SET_BIT(PORTB, PB0);
        }
        else
        {
            SET_BIT(PORTD, PD5); //dekooderin ohjauksena 11
        }
        //käsitellään näppäin kaikilta 4 riviltä
        //apu:n arvot ovat 2-5 jotta sillä voidaan viitata suoraan porttiin B
        //kytkettyihin näppäimistön tuloihin, tulossa on 0 jos näppäintä
        //painetaan ja 1 jos sitä ei paineta, vastaavat arvot tallennetaan
        //myös edelliset-taulukkoon
        for (apu = 2; apu < 6; apu++)
        {
            //jos näppäintä painetaan
            if (bit_is_clear(PINB, apu))
            {
                //jos edellisellä lukukerralla ei painettu
                if (bit_is_set(edelliset[i], apu))
                {
                    //tallennetaan näppäimen numero merkki-muuttujaan
                    MaaritaLuettuNappain();

                    //asetetaan edelliseksi tilaksi 0=painettu
                    CLEAR_BIT(edelliset[i], apu);

                    //ja poistutaan sisemmästä silmukasta
                    break;
                }
            }
            //muuten asetetaan edelliseksi tilaksi 1=ei painettu
            else
```

```
            {
                SET_BIT(edelliset[i], apu);
            }
        }
        //poistutaan ulommastakin silmukasta jos löydettiin painettu näppäin
        if (merkki != TYHJA)
        {
            break;
        }
    }
}


/*****************************************************************************/
/* Funktio määrittää LueNappaimisto-funktion muuttujien i ja apu perusteella */
/* painettuna olevan näppäimen hatussa olevan numeron (tai kirjaimen) ja     */
/* tallentaa sen merkki-muuttujaan.                                          */
/*****************************************************************************/
void MaaritaLuettuNappain(void)
{
    //alla olevasta kuvasta näkyy näppäimien järjestys sekä sarakkeiden ja
    //rivien indeksointi i- ja apu-muuttujilla, näiden perusteella tallennetaan
    //merkki-muuttujaan oikea arvo

    //i 0 1 2 3
    //  _______ apu
    // |1 2 3 A| 2
    // |4 5 6 B| 3
    // |7 8 9 C| 4
    // |0 F E D| 5
    //  -------


    if (i == 0) //vasen sarake
    {
        switch (apu)
        {
            case 2: //ylimmän rivin nappi
                merkki = 1;
                break;

            case 3:
                merkki = 4;
                break;

            case 4:
                merkki = 7;
                break;

            case 5: //alimman rivin nappi
                merkki = 0;
                break;
        }
    }
    else if (i == 1) //toinen sarake
    {
        switch (apu)
        {
            case 2:
                merkki = 2;
                break;

            case 3:
                merkki = 5;
                break;

            case 4:
```

```
                merkki = 8;
                break;

            case 5:
                merkki = NAPPI_F;
                break;
        }
    }
    else if (i == 2) //kolmas sarake
    {
        switch (apu)
        {
            case 2:
                merkki = 3;
                break;

            case 3:
                merkki = 6;
                break;

            case 4:
                merkki = 9;
                break;

            case 5:
                merkki = NAPPI_E;
                break;
        }
    }
    else //oikea sarake
    {
        switch (apu)
        {
            case 2:
                merkki = NAPPI_A;
                break;

            case 3:
                merkki = NAPPI_B;
                break;

            case 4:
                merkki = NAPPI_C;
                break;

            case 5:
                merkki = NAPPI_D;
                break;
        }
    }
}


/*****************************************************************************/
/* Funktio suorittaa, merkki-muuttujaan tallennetun, painetun näppäimen      */
/* mukaiset toimenpiteet.                                                    */
/*****************************************************************************/
void ReagoiPainettuunNappaimeen(void)
{
    switch (merkki)
    {
        case TYHJA: //mitään nappia ei ole painettu
            break;

        case NAPPI_A: //herätyksen 1 on/off-kytkin
            ReagoiOnOff(HERATYS1);
            break;
```

```
        case NAPPI_B: //herätyksen 2 on/off-kytkin
            ReagoiOnOff(HERATYS2);
            break;

        case 3: //näppäintä käytetään numerona, torkkuna ja valoledin kytkimenä
            Reagoi3();
            break;

        default: //käsitellään muut näppäimet
        {
            //herätys-tapahtuman aikana ei hyväksytä mitään säätöjä, koska se
            //ei ole tarpeen
            if (heratys_kaynnissa == EI)
            {
                //jos ei olla näytönpalautus-tilassa
                if (naytontila != NTILA_PALUU)
                {
                    //aloitetaan (alusta) 10 sekunnin näytönpalautusajan
                    //laskeminen (2000*5ms)
                    laskuri_10sek = 2000;
                }
                switch (merkki)
                {
                    case NAPPI_C: //herätyksen 1 ajansäätö
                        ReagoiHeratysajanSaato(HERATYS1);
                        break;

                    case NAPPI_D: //herätyksen 2 ajansäätö
                        ReagoiHeratysajanSaato(HERATYS2);
                        break;

                    case NAPPI_E: //kellonajansäätö
                        ReagoiKello();
                        break;

                    case NAPPI_F: //menu
                        ReagoiMenu();
                        break;

                    default: //jokin numeronäppäin
                        ReagoiNumero();
                }
            }
        }
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan, kun herätyksen 1 on/off- */
/* kytkintä (A) tai herätyksen 2 on/off-kytkintä (B) on painettu.            */
/*   Parametrin "heratys" mahdolliset arvot ovat HERATYS1(1) ja HERATYS2(2), */
/* ja se ilmaisee kumman herätyksen kytkintä ollaan painettu.                */
/*****************************************************************************/
void ReagoiOnOff(unsigned char heratys)
{
    //jos mikään säätö ei ole käynnissä
    if (naytontila == NTILA_NORMAALI)
    {
        //vaihdetaan on/off-muuttujan ja merkkiledin tila
        kaytto_kytkimet[heratys] = kaytto_kytkimet[heratys] ^ 0b00000001;
        AsetaMerkkileditKytkintenMukaan();

        //jos näppäintä vastaava herätys on käynnissä (jolloin se halutaan
        //sammuttaa)
        if (heratys_kaynnissa == heratys)
```

```
        {
            //sammutetaan herätys
            SammutaHeratys();
        }
    }
    //jos näppäintä vastaavan herätyksen säätö on käynnissä (jolloin se tuli
    //valmiiksi)
    else if (naytontila == heratys)
    {
        //jätetään asetettu herätysaika näyttöön ja merkkiledi vilkkumaan
        //sekunniksi (200*5ms)
        laskuri_10sek = 200;

        //asetetaan näytöntilaksi NTILA_PALUU, jotta tämän sekunnin aikana ei
        //reagoitaisi minkään näppäimen painallukseen (joka voisi muuttaa
        //asetettua herätysaikaa)
        naytontila = NTILA_PALUU;
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan eri tapauksissa, kun on   */
/* painettu näppäintä "3".                                                   */
/*****************************************************************************/
void Reagoi3(void)
{
    //näppäin torkkunäppäimenä (kun hälytys on käynnissä)
    if (halyttaa != EI)
    {
        //sammutetaan hälytysääni, mutta jätetään heratys_kaynnissa-muuttuja
        //ennalleen
        apu = heratys_kaynnissa;
        SammutaHeratys();
        heratys_kaynnissa = apu;

        //asetetaan torkkuhälytys laukeamaan määritellyn ajan päähän
        AsetaApuAika(torkun_pituus);
    }
    //näppäin numerona (kun jokin säätö on käynnissä)
    else if (naytontila != NTILA_NORMAALI)
    {
        ReagoiNumero();
    }
    //näppäin valoledin kytkimenä (ja jos valo on mahdollistettu tai laite on
    //paristokäytöllä)
    else if ((valoledi_mahdollinen == KYLLA) || (paristokaytto == KYLLA))
    {
        //sytytetään valoledi
        CLEAR_BIT(PORTD, PD2);

        //aloitetaan 2 minuutin sammutusajan laskeminen (24000*5ms)
        laskuri_2min = 24000;
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan, kun herätyksen 1         */
/* ajansäätö-näppäintä (C) tai herätyksen 2 ajansäätö-näppäintä (D) on       */
/* painettu.                                                                 */
/*   Parametrin "heratys" mahdolliset arvot ovat HERATYS1(1) ja HERATYS2(2), */
/* ja se ilmaisee kumman herätyksen kytkintä ollaan painettu.                */
/*****************************************************************************/
void ReagoiHeratysajanSaato(unsigned char heratys)
{
    //hyväksytään vain normaalitilassa tai paluu säädön alkuun
```

```
    if ((naytontila == NTILA_NORMAALI) || (naytontila == heratys))
    {
        //asetetaan näytöntilaksi herätystä vastaava säätö
        naytontila = heratys;

        //muodostetaan oikea indeksi heratys_ajat-taulukolle
        apu = heratys * 4;

        //kirjoitetaan vanha herätysaika näyttöpuskuriin
        for (i = 0; i < 4; i++)
        {
            puskuri[i] = heratys_ajat[apu + i];
        }

        //asetetaan kaksoispiste loistamaan
        puskuri8[LEDI_PISTE1] = LOISTAA;
        puskuri8[LEDI_PISTE2] = LOISTAA;

        //aloitetaan säätö näytön vasemmasta reunasta (kymmenet tunnit)
        ind_saato = 3;
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan, kun kellonajansäätö-     */
/* näppäintä (E) on painettu.                                                */
/*****************************************************************************/
void ReagoiKello(void)
{
    //hyväksytään vain normaalitilassa tai paluu säädön alkuun
    if ((naytontila == NTILA_NORMAALI) || (naytontila == NTILA_K_SAATO))
    {
        //asetetaan näytöntilaksi kellonajansäätö
        naytontila = NTILA_K_SAATO;

        //kirjoitetaan näyttöpuskuriin nykyinen aika (jotta aloitettaessa
        //säätö alusta, olisi näytöllä oikea aika)
        KirjoitaKelloPuskuriin();

        //asetetaan kaksoispiste loistamaan
        puskuri8[LEDI_PISTE1] = LOISTAA;
        puskuri8[LEDI_PISTE2] = LOISTAA;

        //aloitetaan säätö näytön vasemmasta reunasta (kymmenet tunnit)
        ind_saato = 3;
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan ,kun menu-näppäintä on    */
/* painettu.                                                                 */
/*****************************************************************************/
void ReagoiMenu(void)
{
    //hyväksytään vain normaalitilassa tai eteneminen menussa
    if ((naytontila == NTILA_NORMAALI) || (naytontila >= NTILA_MENU_1))
    {
        //ensimmäisellä funktion suorituskerralla
        if (naytontila == NTILA_NORMAALI)
        {
            //asetetaan näytöntilaksi menun ensimmäinen kohta
            naytontila = NTILA_MENU_1;

            //sammutetaan kaksoispiste
            puskuri8[LEDI_PISTE1] = EI_LOISTA;
```

```
            puskuri8[LEDI_PISTE2] = EI_LOISTA;
        }
        //seuraavilla kerroilla
        else
        {
            //edetään menun seuraavaan kohtaan tai palataan ensimmäiseen
            naytontila++;
            if (naytontila > NTILA_MENU_V)
            {
                naytontila = NTILA_MENU_1;
            }
        }

        //kirjoitetaan näytölle vuorossa oleva menu-teksti
        switch (naytontila)
        {
            case NTILA_MENU_TYP:
                //kirjoitetaan näyttöpuskuriin typ ja herätys2:n nykyinen tila
                puskuri[3] = KIRJAIN_T;
                puskuri[2] = KIRJAIN_Y;
                puskuri[1] = KIRJAIN_P;
                puskuri[0] = heratyksen2_tyyppi;
                break;

            case NTILA_MENU_SLP:
                //kirjoitetaan näyttöpuskuriin slp ja torkun nykyinen pituus
                puskuri[3] = KIRJAIN_S;
                puskuri[2] = KIRJAIN_L;
                puskuri[1] = KIRJAIN_P;
                puskuri[0] = torkun_pituus;
                break;

            case NTILA_MENU_LED:
                //kirjoitetaan näyttöpuskuriin led ja valoledin mahdollisuuden
                //nykyinen tila
                puskuri[3] = KIRJAIN_L;
                puskuri[2] = KIRJAIN_E;
                puskuri[1] = KIRJAIN_D;
                puskuri[0] = valoledi_mahdollinen;
                break;

            case NTILA_MENU_ADC:
                //kirjoitetaan näyttöpuskuriin batt
                puskuri[3] = KIRJAIN_B;
                puskuri[2] = KIRJAIN_A;
                puskuri[1] = KIRJAIN_T;
                puskuri[0] = KIRJAIN_T;
                break;
        }
        //asetetaan indeksi säädettävän numeron kohdalle
        ind_saato = 0;
    }
}


/*****************************************************************************/
/* Funktio suorittaa toimenpiteet, jotka tarvitaan, kun jotain               */
/* numeronäppäintä on painettu. Painetun näppäimen numero on tallennettuna   */
/* merkki-muuttujassa.                                                       */
/*****************************************************************************/
void ReagoiNumero(void)
{
    //hyväksytään vain kun jokin säätö on käynnissä ja kun luku on sallitun
    //suuruinen
    if ((naytontila != NTILA_NORMAALI) && (naytontila != NTILA_PALUU) &&
        TarkistaNumeronLaillisuus())
    {
```

```
//aloitetaan (alusta) 10s näytönpalautusaika (2000*5ms)
laskuri_10sek = 2000;

//kirjoitetaan numero näyttöpuskuriin
puskuri[ind_saato] = merkki;

//tallennetaan numero näytöntilan mukaan oikeaan muuttujaan
switch (naytontila)
{
    case NTILA_H1_SAATO:
        //tallennetaan numero herätysajan 1 oikeaan muuttujaan
        heratys_ajat[AIKA_H1 + ind_saato] = merkki;

        //asetetaan on/off-muuttujan tilaksi ON
        kaytto_kytkimet[HERATYS1] = ON;
        break;

    case NTILA_H2_SAATO:
        //tallennetaan numero herätysajan 2 oikeaan muuttujaan
        heratys_ajat[AIKA_H2 + ind_saato] = merkki;

        //asetetaan on/off-muuttujan tilaksi ON
        kaytto_kytkimet[HERATYS2] = ON;
        break;

    case NTILA_K_SAATO:
        //tallennetaan numero apu-ajan oikeaan muuttujaan
        heratys_ajat[AIKA_APU + ind_saato] = merkki;

        //kun viimeinen numero on asetettu
        if (ind_saato == 0)
        {
            //estetään keskeytykset globaalisti, jottei kellonajan
            //kopiointi häiriinny
            cli();

            //aloitetaan timer0:n laskeminen nollasta
            TCNT0 = 0;

            //kirjoitetaan asetettu kellonaika kello-muuttujiin
            kello_10tunnit = heratys_ajat[3];
            kello_tunnit = heratys_ajat[2];
            kello_10minuutit = heratys_ajat[1];
            kello_minuutit = heratys_ajat[0];

            //aloitetaan millisekuntien ja sekuntien laskeminen alusta
            kello_sekunnit = 0;
            kello_5ms_laskuri = 200;

            //sallitaan keskeytykset uudelleen
            sei();

            //nostetaan lippu merkiksi kellonajan päivittymisestä
            lippu_kello = KYLLA;

            //merkitään kellonaika asetetuksi, tällä on merkitystä vain
            //ensimmäisellä kerralla
            kellonaika_on_asetettu = KYLLA;
        }
        break;

    case NTILA_MENU_TYP:
        //jos painettu numero on JATKUVA(0)
        if (merkki == JATKUVA)
        {
            //asetetaan herätykseen 2 normaali jatkuva hälytysääni,
            //joka pitää sammuttaa erikseen
```

```
                heratyksen2_tyyppi = JATKUVA;
            }
            //jos painettiin jotain muuta numeroa
            else
            {
                //asetetaan herätyksen 2 hälytysääneksi vain yksi viiden
                //sekunnin jakso
                heratyksen2_tyyppi = KERRAN;

                //ja korjataan näyttöpuskuriin sitä vastaava luku
                puskuri[0] = KERRAN;
            }
            break;

        case NTILA_MENU_SLP:
            //asetetaan painettu numero torkun pituudeksi
            torkun_pituus = merkki;

            //jos pituus oli alle minimin
            if (merkki < 5)
            {
                //korjataan pituudeksi minimi 5 minuuttia
                torkun_pituus = 5;

                //ja korjataan näyttöpuskuriin luku 5
                puskuri[0] = 5;
            }
            break;

        case NTILA_MENU_LED:
            //jos painettu numero on EI(0)
            if (merkki == EI)
            {
                //poistetaan valoledi käytöstä
                valoledi_mahdollinen = EI;
            }
            //jos painettiin jotain muuta numeroa
            else
            {
                //mahdollistetaan valoledin käyttö
                valoledi_mahdollinen = KYLLA;

                //ja korjataan näyttöpuskuriin sitä vastaava luku
                puskuri[0] = KYLLA;
            }
            break;

        case NTILA_MENU_ADC:
            //käynnistetään pariston jännitemittaus (ADC)
            SET_BIT(ADCSRA, ADSC);

            //ja korjataan näyttöpuskurin teksti oikeaksi
            puskuri[0] = KIRJAIN_T;
            break;
    }

    //jos asetettu numero oli jo viimeinen
    if (ind_saato == 0)
    {
        //jätetään asetusteksti näyttöön sekunniksi (200*5ms)
        laskuri_10sek = 200;

        //asetetaan näytöntilaksi NTILA_PALUU, jotta tämän sekunnin aikana
        //ei reagoitaisi minkään näppäimen painallukseen (joka voisi
        //muuttaa asetettua herätysaikaa)
        naytontila = NTILA_PALUU;
    }
```

```
        else
        {
            //muuten siirrytään seuraavaan numeroon
            ind_saato--;
        }
    }
}


/*****************************************************************************/
/* Funktio tarkistaa, onko merkki-muuttujassa oleva luku kelvollinen         */
/* ind_saato-muuttujan osoittamaksi näytön numeroksi. Jos luku ei ole        */
/* kelvollinen, palautetaan EI, ja muuten palautetaan KYLLA.                 */
/*****************************************************************************/
unsigned char TarkistaNumeronLaillisuus(void)
{
    //jos kymmenet tunnit on enemmän kuin 2
    if ((ind_saato == 3) && (merkki > 2))
    {
        //palautetaan epätosi
        return EI;
    }
    //jos tunnit on enemmän kuin 3
    if ((ind_saato == 2) && (merkki > 3))
    {
        //ja jos kymmenet tunnit on 2
        if (
           ((naytontila == NTILA_H1_SAATO) && (heratys_ajat[AIKA_H1 + 3] == 2))
           ||
           ((naytontila == NTILA_H2_SAATO) && (heratys_ajat[AIKA_H2 + 3] == 2))
           ||
           ((naytontila == NTILA_K_SAATO) && (heratys_ajat[AIKA_APU + 3] == 2))
           )
        {
            //palautetaan epätosi
            return EI;
        }
    }
    //jos kymmenet minuutit on enemmän kuin 5
    if ((ind_saato == 1) && (merkki > 5))
    {
        //palautetaan epätosi
        return EI;
    }
    //muuten palautetaan tosi
    return KYLLA;
}


/*****************************************************************************/
/* Keskeytysfunktio, joka suoritetaan tasan 5ms välein mikrokontrollerin     */
/* timer-keskeytyksen ajastamana.                                            */
/* Funktio ylläpitää oikeata kellonaikaa ja vilkuttaa kaksoispistettä ja     */
/* herätyksien merkkiledejä silloin, kun niiden halutaan vilkkuvan. Lisäksi  */
/* funktio käynnistää varmennuspariston jännitemittauksen kaksi kertaa       */
/* vuorokaudessa (klo. 10:00 ja 20:00).                                      */
/*****************************************************************************/
ISR(TIMER0_COMPA_vect)
{
    //nostetaan lippu merkiksi (pääohjelmalle) 5 millisekunnin kulumisesta
    lippu_5ms = KYLLA;

    //vähennetään laskurista yksi 5 millisekunnin kuluminen
    kello_5ms_laskuri--;

    //jos on kulunut 500ms
    if (kello_5ms_laskuri == 100)
```

```
{
    //kun näytöllä on kellonaika, vilkutetaan kaksoispistettä
    //sekunnin jaksonajalla
    if (naytontila == NTILA_NORMAALI)
    {
        //sammutetaan kaksoispiste puoleksi sekunniksi
        puskuri8[LEDI_PISTE1] = EI_LOISTA;
        puskuri8[LEDI_PISTE2] = EI_LOISTA;
    }
    //herätyksen ja sen säädön aikana vilkutetaan ko. herätyksen
    //merkkilediä samassa tahdissa kaksoispisteen kanssa
    if ((heratys_kaynnissa == HERATYS1) || (naytontila == NTILA_H1_SAATO))
    {
        puskuri8[LEDI_HERATYS1] = EI_LOISTA;
    }
    if ((heratys_kaynnissa == HERATYS2) || (naytontila == NTILA_H2_SAATO))
    {
        puskuri8[LEDI_HERATYS2] = EI_LOISTA;
    }
}
//jos on kulunut 1000ms
if (kello_5ms_laskuri == 0)
{
    //vilkutetaan kaksoispistettä
    if (naytontila == NTILA_NORMAALI)
    {
        //sytytetään kaksoispiste puoleksi sekunniksi
        puskuri8[LEDI_PISTE1] = LOISTAA;
        puskuri8[LEDI_PISTE2] = LOISTAA;
    }

    //vilkutetaan herätyksen merkkiledejä
    if ((heratys_kaynnissa == HERATYS1) || (naytontila == NTILA_H1_SAATO))
    {
        puskuri8[LEDI_HERATYS1] = LOISTAA;
    }
    if ((heratys_kaynnissa == HERATYS2) || (naytontila == NTILA_H2_SAATO))
    {
        puskuri8[LEDI_HERATYS2] = LOISTAA;
    }
    //aloitetaan millisekuntien laskeminen alusta
    kello_5ms_laskuri = 200;

    //ja lisätään sekunteja yhdellä
    kello_sekunnit++;

    //jos minuutti on kulunut, päivitetään muutkin kello-muuttujat
    //tarvittaessa
    if (kello_sekunnit == 60)
    {
        //nostetaan lippu merkiksi kellonajan (minuuttien) päivittymisestä
        lippu_kello = KYLLA;

        kello_sekunnit = 0;
        kello_minuutit++;
        if (kello_minuutit == 10)
        {
            kello_minuutit = 0;
            kello_10minuutit++;
            if (kello_10minuutit == 6)
            {
                kello_10minuutit = 0;
                kello_tunnit++;
                if (kello_tunnit == 10)
                {
                    kello_tunnit = 0;
                    kello_10tunnit++;
```

```
                        //käynnistetään paristojännitteen mittaus (ADC)
                        SET_BIT(ADCSRA, ADSC);

                        //suoritetaan 2 kertaa vuorokaudessa tapahtuva
                        //kellonajan korjaus
                        kello_5ms_laskuri = kello_5ms_laskuri + 10;
                    }
                    //jos tunneiksi tulee 24, niin korjataan se 00:ksi
                    if ((kello_10tunnit == 2) && (kello_tunnit == 4))
                    {
                        kello_10tunnit = 0;
                        kello_tunnit = 0;
                    }
                }
                //suoritetaan 10 minuutin välein tapahtuva kellonajan korjaus
                //(eli lisätään laskuriin kuusi 5ms askelta)
                kello_5ms_laskuri = kello_5ms_laskuri + 6;
            }
        }
    }
}


/*****************************************************************************/
/* Keskeytysfunktio, joka suoritetaan, kun analogisen komparaattorin         */
/* ulostulon arvo vaihtuu.                                                   */
/* Funktio vaihtaa käyttöjännitteen lähteeksi pariston sekä sammuttaa näytöt */
/* ja muut tarpeettomat virtaa kuluttavat osat, kun verkkosähkö katkeaa.     */
/* Verkkosähkön palatessa funktio palauttaa toiminnot käyttöön ja käynnistää */
/* paristojännitteen mittauksen.                                             */
/*****************************************************************************/
ISR(ANALOG_COMP_vect)
{
    //Komparaattorin negatiiviseen sisääntuloon on kytkettynä 2,5V:n
    //jännitereferenssi ja positiiviseen sisääntuloon on kytkettynä
    //jännitteenjaon kautta ulkoiselta lähteeltä tuleva syöttöjännite, 6-12V.
    //Kun syöttöjännite laskee 5,83 volttiin tai alle, komparaattorin ulostulo,
    //ACO bitti, muuttuu nollaksi.


    //jos verkkosähkö katkesi juuri
    if (bit_is_clear(ACSR, ACO))
    {
        //sammutetaan näyttö
        PORTC = 0;

        //ohjataan käyttöjännitteen lähteeksi paristo
        SET_BIT(PORTD, PD4);

        //sammutetaan valoledi ja alhaisen paristojännitteen merkkiledi
        SET_BIT(PORTD, PD2);
        SET_BIT(PORTD, PD3);

        //ohjataan segmenttien ohjaustransistorit johtamattomiksi eli
        //kirjoitetaan siirtorekisterin kaikkiin lähtöihin '1'
        SET_BIT(PORTD, PD0);

        //(lähetetään kahdeksan kellopulssia)
        for (apuk = 8; apuk != 0; apuk--)
        {
            SET_BIT(PORTD, PD1);
            CLEAR_BIT(PORTD, PD1);
        }
        //asetetaan paristokaytto-muuttuja merkiksi PaivitaNaytto-funktiolle
        paristokaytto = KYLLA;
    }
```

```
    //muuten verkkosähkö palasi juuri
    else
    {
        //ohjataan käyttöjännitteen lähteeksi takaisin ulkoinen lähde
        CLEAR_BIT(PORTD, PD4);

        //nollataan paristokaytto-muuttuja
        paristokaytto = EI;

        //käynnistetään pariston jännitemittaus (ADC)
        SET_BIT(ADCSRA, ADSC);
    }
}


/*****************************************************************************/
/* Keskeytysfunktio, joka suoritetaan, kun AD-muunnos on tullut valmiiksi.   */
/* Funktio joko sytyttää tai sammuttaa alhaisen paristojännitteen            */
/* merkkiledin mittaustuloksen perusteella. Rajajännite on 7,2V.             */
/*****************************************************************************/
ISR(ADC_vect)
{
    //Mittauspinnin jännite, V(in), muodostetaan paristojännitteestä, V(batt),
    //jännitteenjaolla seuraavasti: V(in)=V(batt)*1,0/(1,0+3,3).
    //AD-muuntimen tulos ADC=V(in)*1024/V(ref), missä V(ref) on 2,5V.
    //
    //Halutulla rajajännitteellä 7,2V AD-muuntimen tulos on 1010101110, ja
    //näistä 8 eniten merkitsevää bittiä on rekisterissä ADCH, jota alla
    //ehtolauseessa tutkitaan.


    //kun paristojännite on pienempi tai yhtä suuri kuin 7,2V
    if (ADCH <= 0b10101011)
    {
        //sytytetään alhaisen paristojännitteen merkkiledi
        CLEAR_BIT(PORTD, PD3);
    }
    else
    {
        //muuten sammutetaan alhaisen paristojännitteen merkkiledi
        SET_BIT(PORTD, PD3);
    }
}
```

the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk\_I/O}/2$ when OCR0A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

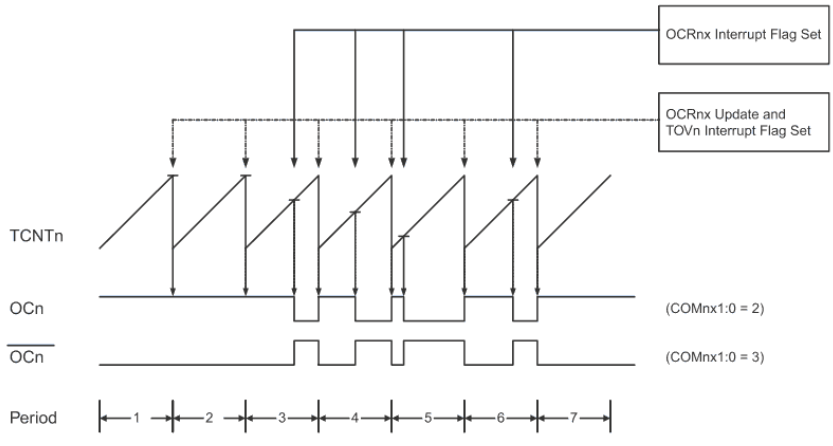
As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

14.7.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM2:0 = 3, and OCR0A when WGM2:0 = 7. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 14-6. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0x and TCNT0.

Figure 14-6. Fast PWM Mode, Timing Diagram

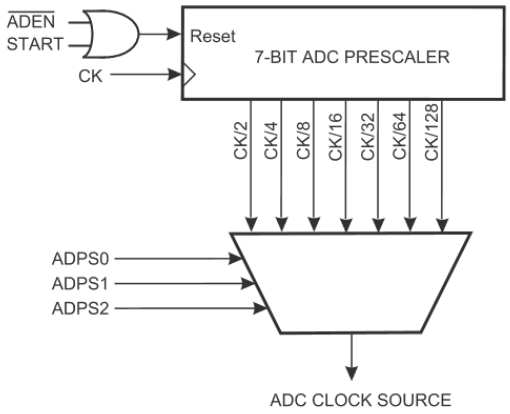


The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches TOP. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

23.4 Prescaling and Conversion Timing

Figure 23-3. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

When the bandgap reference voltage is used as input to the ADC, it will take a certain time for the voltage to stabilize. If not stabilized, the first value read after the first conversion may be wrong.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

23.5.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

23.5.2 ADC Voltage Reference

The reference voltage for the ADC (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either AV_{CC}, internal 1.1V reference, or external AREF pin.

AV_{CC} is connected to the ADC through a passive switch. The internal 1.1V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. V_{REF} can also be measured at the AREF pin with a high impedance voltmeter. Note that V_{REF} is a high impedance source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AV_{CC} and 1.1V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

23.6 ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

a. Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
b. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
c. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

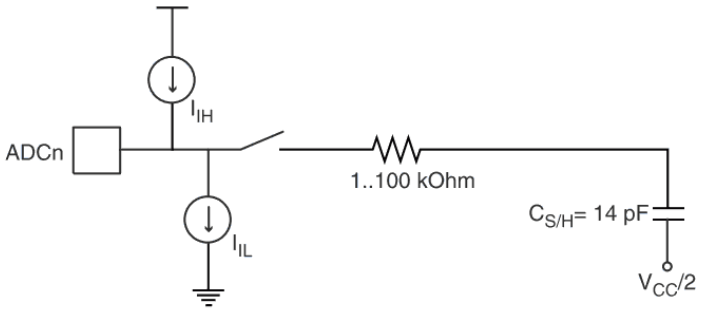
23.6.1 Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated in Figure 23-8. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedance sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 23-8. Analog Input Circuitry



23.6.2 Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

a. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
b. The AV_{CC} pin on the device should be connected to the digital V_{CC} supply voltage via an LC network as shown in Figure 23-9.
c. Use the ADC noise canceler function to reduce induced noise from the CPU.
d. If any ADC [3..0] port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress. However, using the 2-wire Interface (ADC4

23.7 ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 23-2 on page 256 and Table 23-3 on page 257). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

23.8 Register Description

23.8.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 23-2. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 23-2. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see "ADCL and ADCH – The ADC Data Register" on page 259.

- **Bit 4 – Res: Reserved Bit**

This bit is an unused bit in the ATmega48/88/168, and will always read as zero.

- **Bits 3:0 – MUX3:0: Analog Channel Selection Bits**

Table 27-3. Lock Bit Protection Modes[1][2]. Only ATmega88/168.

BLB0 Mode	BLB02	BLB01	
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
BLB1 Mode	**BLB12**	**BLB11**	
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. “1” means unprogrammed, “0” means programmed

27.2 Fuse Bits

The ATmega48/88/168 has three Fuse bytes. Table 27-4 - Table 27-7 describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 27-4. Extended Fuse Byte for mega48

Extended Fuse Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
–	1	–	1
SELFPRGEN	0	Self Programming Enable	1 (unprogrammed)

Table 27-13. XA1 and XA0 Coding

XA1	XA0	Action when XTAL1 is Pulsed
0	0	Load Flash or EEPROM Address (High or low address byte determined by BS1).
0	1	Load Data (High or Low data byte for Flash determined by BS1).
1	0	Load Command
1	1	No Action, Idle

Table 27-14. Command Byte Bit Coding

Command Byte	Command Executed
1000 0000	Chip Erase
0100 0000	Write Fuse bits
0010 0000	Write Lock bits
0001 0000	Write Flash
0001 0001	Write EEPROM
0000 1000	Read Signature Bytes and Calibration byte
0000 0100	Read Fuse and Lock bits
0000 0010	Read Flash
0000 0011	Read EEPROM

27.7 Parallel Programming

27.7.1 Enter Programming Mode

The following algorithm puts the device in Parallel (High-voltage) Programming mode:

1. Set Prog_enable pins listed in Table 27-12 on page 291 to “0000”, RESET pin to 0V and V_{CC} to 0V.
2. Apply 4.5 - 5.5V between V_{CC} and GND.

Ensure that V_{CC} reaches at least 1.8V within the next 20 µs.

3. Wait 20 - 60 µs, and apply 11.5 - 12.5V to RESET.
4. Keep the Prog_enable pins unchanged for at least 10µs after the High-voltage has been applied to ensure the Prog_enable Signature has been latched.
5. Wait at least 300 µs before giving any parallel programming commands.
6. Exit Programming mode by power the device down or by bringing RESET pin to 0V.

If the rise time of the V_{CC} is unable to fulfill the requirements listed above, the following alternative algorithm can be used.

1. Set Prog_enable pins listed in Table 27-12 on page 291 to “0000”, RESET pin to 0V and V_{CC} to 0V.
2. Apply 4.5 - 5.5V between V_{CC} and GND.
3. Monitor V_{CC}, and as soon as V_{CC} reaches 0.9 - 1.1V, apply 11.5 - 12.5V to RESET.

27.7.14 Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 293 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The Calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

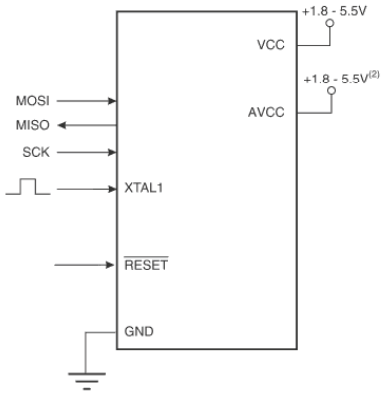
27.7.15 Parallel Programming Characteristics

For characteristics of the parallel programming, see “Parallel Programming Characteristics” on page 313.

27.8 Serial Downloading

Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{RESET}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{RESET}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 27-15 on page 300, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

Figure 27-7. Serial Programming and Verify[1]



Notes: 1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.
2. V_{CC} - 0.3V < AV_{CC} < V_{CC} + 0.3V, however, AV_{CC} should always be within 1.8 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into 0xFF.

Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low: > 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

High: > 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted) (Continued)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{CC}	Power Supply Current(5)	Active 1MHz, V_{CC} = 2V (ATmega48/88/168V)			0.55	mA
		Active 4MHz, V_{CC} = 3V (ATmega48/88/168L)			3.5	mA
		Active 8MHz, V_{CC} = 5V (ATmega48/88/168)			12	mA
		Idle 1MHz, V_{CC} = 2V (ATmega48/88/168V)		0.25	0.5	mA
		Idle 4MHz, V_{CC} = 3V (ATmega48/88/168L)			1.5	mA
		Idle 8MHz, V_{CC} = 5V (ATmega48/88/168)			5.5	mA
	Power-down mode	WDT enabled, V_{CC} = 3V		8	15	µA
		WDT disabled, V_{CC} = 3V		1	2	µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2		10	40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Notes:
1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20 mA at V_{CC} = 5V, 10 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   ATmega48/88/168:
   1] The sum of all I_{OL}, for ports C0 - C5, ADC7, ADC6 should not exceed 100 mA.
   2] The sum of all I_{OL}, for ports B0 - B5, D5 - D7, XTAL1, XTAL2 should not exceed 100 mA.
   3] The sum of all I_{OL}, for ports D0 - D4, $\overline{\text{RESET}}$ should not exceed 100 mA.
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20 mA at V_{CC} = 5V, 10 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   ATmega48/88/168:
   1] The sum of all I_{OH}, for ports C0 - C5, D0- D4, ADC7, $\overline{\text{RESET}}$ should not exceed 150 mA.
   2] The sum of all I_{OH}, for ports B0 - B5, D5 - D7, ADC6, XTAL1, XTAL2 should not exceed 150 mA.
   If II_{OH} exceeds the test condition, V_{OH} may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
5. Values with “Minimizing Power Consumption” enabled (0xFF).

Figure 29-45. AREF External Reference Current vs. V_{CC}

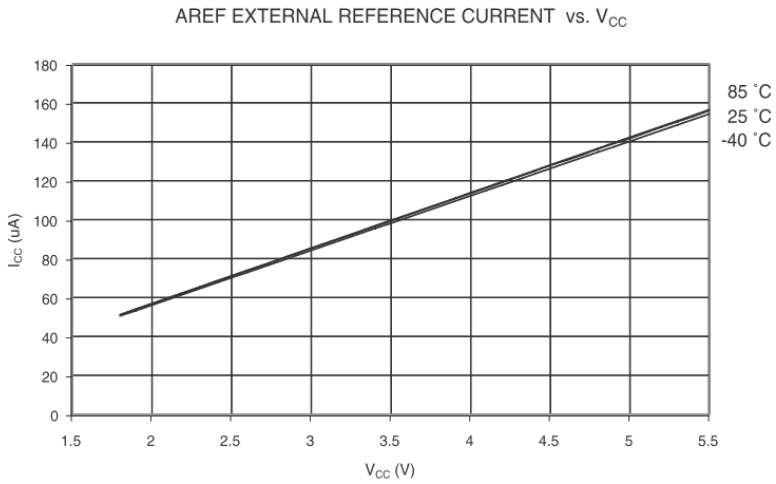


Figure 29-46. Analog Comparator Current vs. V_{CC}