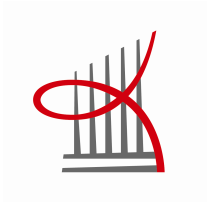


# SÄHKÖMITTARIN LUKIJALAITE

Jukka-Tapio Järvenpää

Opinnäytetyö
Kesäkuu 2013
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja
elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU

Tampere University of Applied Sciences

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

JUKKA-TAPIO JÄRVENPÄÄ:
Sähkömittarin lukijalaite

Opinnäytetyö 51 sivua, josta liitteitä 11 sivua
Kesäkuu 2013

---

Tämän opinnäytetyön tarkoituksena oli kehittää laite, joka lukee sähkökaapin mittarista sähkön kulutusta ja näyttää LCD-näytöllä sähköenergian kulutusta sekä tehoa.

Useimpien sähkökaappien kWh-mittareissa on LED-valo, joka vilkahtaa aina kun 1 Wh sähköenergiaa on kulutettu. Lukijalaitteen toiminta perustuu siihen, että se lukee fototransistorin avulla mittarin LED-valoa, josta saatu tieto käsitellään MSP430-sarjan mikrokontrolleriin ohjelmoidulla ohjelmalla. Ohjelma laskee vilkahdusten määrän ja niiden välissä kuluvan ajan perusteella sähköenergian kulutusta ja tehoa, ja näyttää lukemat LCD-näytöllä. Laitteella on myös mahdollista lähettää tiedot eteenpäin langattomalla yhteydellä. Mikrokontrollerin ohjelmointiin on käytetty eZ430-F2013 kehitystyökalua, ja ohjelma on toteutettu C-kielellä.

Testaustilanteessa laite toimi suunnitellulla tavalla. Laitteen toiminnan kannalta todettiin tärkeäksi, että fototransistori on asennettu LED-valon lähelle siten, että se on riittävästi suojattu ulkopuoliselta valon vaihtelulta. MSP430-mikrokontrollerin soveltuvuus paristokäyttöiseen lukijalaitteeseen todettiin hyväksi.

---

Asiasanat: sähkömittarit, sähkönkulutus, msp430, mikro-ohjaimet, fototransistorit

**ABSTRACT**

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded Systems and Electronics

JUKKA-TAPIO JÄRVENPÄÄ:
Electricity Meter Reading Device

Bachelor's thesis 51 pages, appendices 11 pages
June 2013

---

The purpose of this thesis was to develop a device that can read electric energy consumption and power from an electricity meter, and display this information on an LCD-screen.

Most kWh-meters have an LED-light that blinks when 1 Wh of electric energy has been consumed. The operation of the reading device is based on monitoring the LED-light with a phototransistor, and the collected information is processed with a program made for MSP430-series microcontroller. The program uses the number of the blinks and the delay between the blinks to calculate electric energy consumption and power, and displays this information on an LCD-screen. The device can also send the information forward through a wireless connection. The microcontroller was programmed using eZ430-F2013 development tool, and the program was made in C-language.

The device was working correctly in a testing situation. It was noticed that it is important to install the phototransistor close to the LED-light in a way that keeps it covered from the interference of outside light sources. It was concluded that the suitability of the MSP430 microcontroller for the battery-operated reading device is good.

---

Key words: electricity meters, electric energy consumption, msp430, microcontrollers, phototransistors

**SISÄLLYS**

**LYHENTEET**

| | |
|---|---|
| DCO | Digitally Controlled Oscillator – Digitaalisesti ohjattu oskillaattori |
| I/O | Input/Output – Tulo/lähtö |
| LCD | Liquid Crystal Display – Nestekidenäyttö |
| LED | Light Emitting Diode – Valoa hohtava diodi |
| P2IFG | Port P2 interrupt flag – Portin P2 keskeytyslippu |
| PWM | Pulse-Width Modulation – Pulssinleveysmodulaatio |
| RF | Radio Frequency – Radiotaajuus |
| RISC | Reduced Instruction Set Computer – Supistetun käskykannan suoritin |
| RTC | Real-Time Clock – Reaaliaikakello |
| SMCLK | Sub-Main Clock |
| UART | Universal Asynchronous Receiver/Transmitter – Universaali asynkroninen vastaanotin/lähetin |

# 1 JOHDANTO

Työn aiheena on sähkömittarin lukijalaitteen kehittäminen käyttäen MSP430-sarjan mikrokontrolleria, sekä valmiin laitteen osien ja toiminnan analysointi. Laite lukee sähkökaapin kWh-mittarin vilkkuvaa LED-valoa fototransistorin avulla, laskee tämän perusteella kulutetun sähköenergian määrää ja tehoa sekä näyttää nämä tiedot LCD-näytöllä. Laite voi myös lähettää tiedot eteenpäin UART-muodossa langattomalle lähettimelle. Useissa sähkömittareissa on LED-valo joka vilkahtaa aina kun 1 Wh sähköenergiaa on kulunut, kaikissa mittareissa ei kuitenkaan ole omaa LCD-näyttöä tai mahdollisuutta lähettää lukematietoa automaattisesti muihin laitteisiin.

Työssä keskeisimpänä osana käytetään Texas Instrumentsin MSP430-sarjan mikrokontrolleria, jolle ladattu ohjelmakoodi käsittelee fototransistorilta tulevan tiedon, laskee sähkön kulutuksen, ja lähettää tiedon halutussa muodossa edelleen LCD-näytölle tai UART-lähtöön. Työssä käsitellään mikrokontrollerin ominaisuuksia ja ohjelmointia työn aiheena olevaan laitteeseen liittyen. Tämä sisältää mm. ohjelmaympäristön käytön, tulojen ja lähtöjen käsittelyn, keskeytyspalvelut, ajastimet, sisäiset kellotaajuudet, virransäästöominaisuudet, UART-lähetyksen, sähkönkulutuksen laskemisen ja LCD-näytölle lähetyksen. Mikrokontrollerin ohjelmakoodi toteutetaan C-kielellä. Lisäksi työssä käsitellään myös muut laitteessa tarvittavat osat ja huomioitavat seikat, kuten piirilevy ja virransyöttö, fototransistorin toiminta, LCD-näyttö, langaton tiedonlähetys (UART/Amber), mahdolliset käytännön sovelluksessa vastaantulevat virhetilanteet ja niiden huomioiminen.

Laitteen kehittämiseen on käytetty eZ430-F2013 -kehitystyökalua ja sen mukana olevaa IAR Embedded Workbench -ohjelmointiympäristöä. Laite on koottu reikäpiirilevylle ja se saa virtansa kahdesta AA-paristosta. Laitteen kytkentäkaavio ja piirilevyn layout suunniteltiin alunperin kynällä ja paperilla, ja jälkeenpäin näistä tehtiin myös siistimmät versiot ilmaisella KiCad-ohjelmalla.

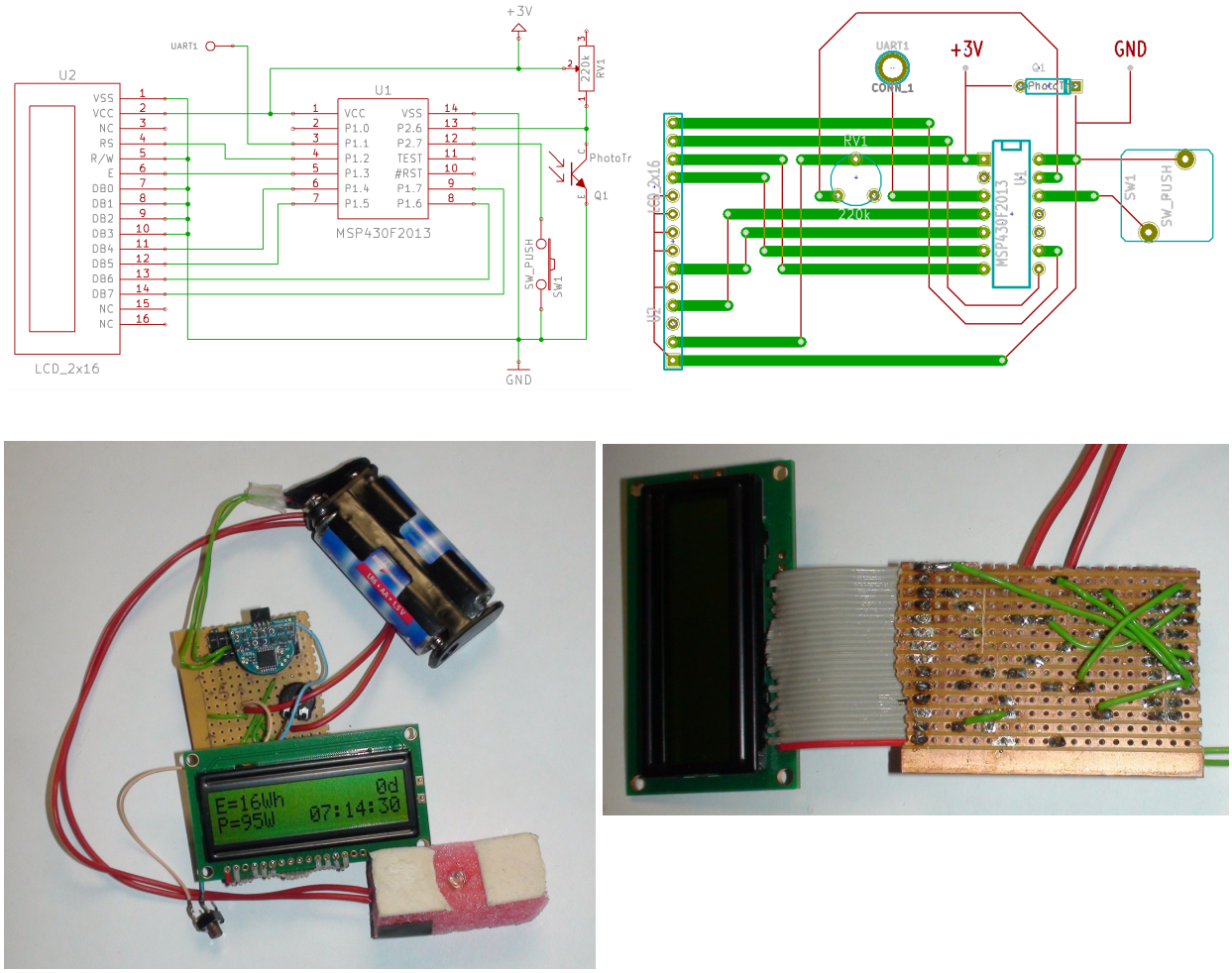
# 2 LAITTEEN KOKOAMINEN JA OSIEN TOIMINTA

Tässä luvussa käsitellään lukijalaitteen kokoamista ja sen eri osien toimintaa. Laitteen tekemiseen tarvitaan seuraavat osat:

- eZ430-F2013 -kehitystyökalu (sis. MSP430F2013-mikrokontrollerin)
- Noin 55 x 35 mm kokoinen reikäpiirilevy
- 14-pinninen (2 x 7) piikkirima
- Kanta piikkirimalle
- Fototransistori BPV11
- 220 kΩ:n säätövastus
- 2 x 16 merkkinen 3 V:n LCD-moduuli (FDCC1602L-RNNYBW-16LE)
- RF-moduuli AMB8420 (valinnainen)
- Kytkin
- Paristokotelo kahdelle AA-paristolle
- Johtimia
- Ohjelmointiympäristö (IAR Embedded Workbench)
- C-kielinen ohjelma

## 2.1 Piirilevy ja virransyöttö

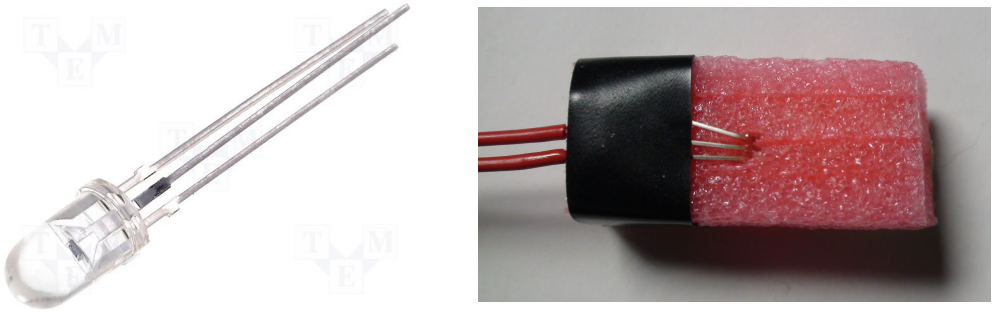
Työssä tehty lukijalaite on koottu yksipuoleiselle reikäpiirilevylle. Se saa virtansa kahdesta sarjaan kytketystä 1,5 voltin AA-paristosta, joita varten on oma irrotettava paristokotelo. Työssä käytetyn eZ430-F2013 -kehitystyökalun kohdekortille on juotettu 14-pinninen piikkirima, jonka avulla mikrokontrolleri saadaan helposti liitettyä varsinaisessa laitteessa olevaan kantaan, sekä irrottaa siitä takaisin USB-emulointiliitäntäkorttiin uudelleenohjelmointia ja testausta varten. LCD-näyttö on yhdistetty piirilevylle juottamalla lattakaapeli näiden välille, mutta erilliset irtojohtimetkin kävisivät tähän. Levyllä on johdinten kautta yhdistettynä yksi kytkin laitteen asetusten muuttamista varten. Näiden lisäksi piirilevylle on liitetty fototransistorin valoherkkyyden säätämistä varten oleva 220 kΩ:n säätövastus sekä n. 35 cm johdinten kautta itse fototransistori BPV11. Varsinaista kotelointia laitteella ei vielä ole, mutta laite mahtuu 70 x 57 x 37 mm kokoisen laatikon sisälle.




Kuva 1. Kytkentäkaavio ja kuvia laitteesta [liitteet 1 ja 2]

## 2.2 Fototransistori

NPN-tyyppinen fototransistori Vishay BPV11 on herkkä näkyvälle valolle, sen vasteaika on lyhyt ja katselukulma kapea. Piirilevyllä sen emitteri on kytketty maihin ja kollektori MSP430:n tuloon P2.6 sekä 220 kΩ:n säätövastukseen, kantaa ei ole kytketty mihinkään. Fototransistori johtaa sitä paremmin sähköä, mitä suurempi valoteho siihen kohdistuu. Säätövastuksen avulla säädetään fototransistorin läpi kulkevan virran määrää sopivaksi siten, että MSP430:n tulo muuttuu loogiseksi nollaksi silloin, kun mitattavana oleva LED-valo syttyy ja takaisin ykköseksi kun se sammuu. Säätövastusta tarvitaan, koska ympäristöstä tuleva valo vaikuttaa myös fototransistorin johtavuuteen ja erilaisissa olosuhteissa voidaan joutua säätämään kytkennän valoherkkyyttä erilaiseksi. Ympäristön valoa pyritään suodattamaan pois myös siten, että fototransistori on asetettu vaahtomuovinpalasen sisälle, joka on tarkoitus kiinnittää mahdollisimman lähelle mitattavaa LED-valoa.



Kuva 2. Fototransistori BPV11

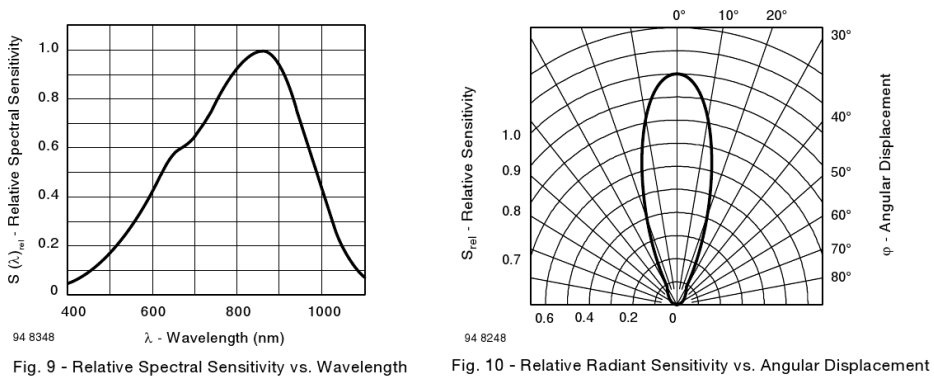


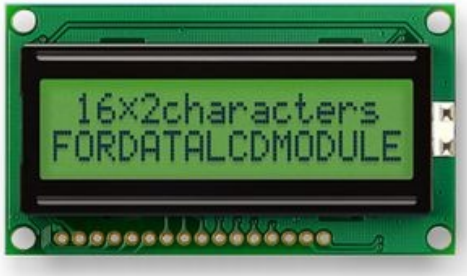

Fig. 9 - Relative Spectral Sensitivity vs. Wavelength

Fig. 10 - Relative Radiant Sensitivity vs. Angular Displacement

Kuva 3. BPV11:n taajuusalue ja katselukulma [5]

## 2.3 LCD-moduuli

Laitteen LCD-näyttö näyttää kulutetun sähkön määrää ja muita tietoja mikrokontrollerin ohjaamana. Näytöksi on valittu Fordatan valmistama LCD-moduuli FDCC1602L-RNNYBW-16LE, koska sitä voidaan käyttää 3 voltin käyttöjännitteellä ja se oli saatavilla olevista vaihtoehdoista edullisimmasta päästä. Moduuli sisältää 2 x 16 merkkisen LCD-pistematriisinäytön lisäksi LCD:n ohjauspiirin, jonka avulla näytölle on yksinkertaista lähettää merkkejä ja ohjauskäskyjä. LCD:n ohjauspiiri on Hitachin HD44780 -standardin mukainen, joka löytyy lähes kaikista vastaavantyyppisistä LCD-moduuleista joten samalla ohjelmakoodilla voidaan käyttää useimpien valmistajien LCD-moduuleja. Moduulille voidaan lähettää dataa joko 8- tai 4-bittisessä muodossa. Näiden databittien lisäksi näytön ohjaukseen tarvitaan signaalit RS, E ja R/W, joista tosin R/W voi olla koko ajan kytkettynä maihin, jos näytölle halutaan ainoastaan kirjoittaa, eikä lukea dataa [7]. Tässä työssä käytetään 4-bittistä datansiirtoa, joten LCD:n ohjaukseen tarvitaan yhteensä 6 lähtöä MSP430F2013:n 10:stä I/O-pinnistä.



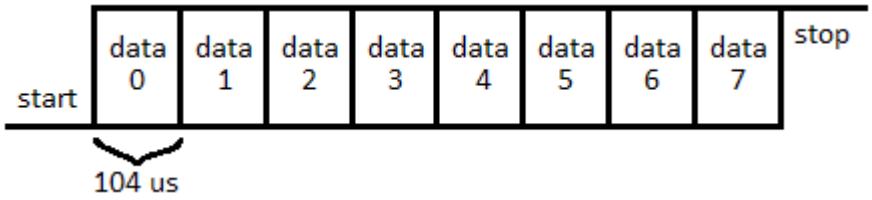

Kuva 4. LCD-moduuli FDCC1602L-RNNYBW-16LE [8]

## 2.4 Kytkin

Kytkintä käytetään näytön kytkemiseen päälle ja pois, Wh-laskurin nollaamiseen sekä kellonajan asettamiseen. Kytkin on liitetty MSP430:n tulon P2.7 ja maan välille, jolloin kytkintä painettaessa tulon tila muuttuu nollaksi, ja kytkimen ollessa vapautettuna, P2.7:n sisäinen ylösvetovastus pitää sen tilan ykkösenä. Kytkintä painettaessa ja vapautettaessa syntyy kytkinvärähtelyä, joka suodatetaan pois ohjelmallisesti. Tulojen rajoitetun määrän vuoksi laitteessa on vain yksi kytkin, mutta ohjelmallisesti sillä saadaan toteutettua useita eri toimintoja, tunnistamalla erikseen lyhyet ja pitkät painallukset sekä näiden yhdistelmät.

## 2.5 UART ja langaton yhteys

Laitteen UART-lähtöön (P1.1) on mahdollista kytkeä esimerkiksi Amber Wirelessin valmistama RF-moduuli AMB8420, jossa on 868 MHz:n taajuudella toimiva radiolähetin/-vastaanotin. Sen kanssa voi kommunikoida UART-sarjaliikenneprotokollan avulla, jossa jokaisen tavun lähetys koostuu yleensä 10 bitistä eli start-bitistä, 8 databitistä sekä stop-bitistä. Moduuli on oletusarvoisesti *transparent*-toimintatilassa ja käyttää 9600 bit/s tiedonsiirtonopeutta. Tässä toimintatilassa moduuli lähettää automaattisesti sille annetun datan eteenpäin, joten sen ohjaamiseen ei tarvita kuin yksi linja MSP430:lta eli pinni P1.1 kytketään AMB8420:n pinniin URXD [6]. Tällöin lähetys on yksisuuntainen eikä datan perillemenoa varmisteta mitenkään. Langattomassa lähetyksessä voi ilmetä häiriöitä, joten ohjelmakoodi lähettää jokaisessa paketissa kaksi tavua, ensin kirjaimen ”W” ASCII-koodin ja perään Wh-laskurin 8 viimeistä bittiä. Näiden avulla pitäisi vastaanottavan ohjelman pystyä seuraamaan oikeaa sähkönkulutuksen määrää, vaikka useita viestipaketteja jäisikin vastaanottamatta. Haluttaessa voitaisiin lähettää joka paketissa vaikka koko Wh-laskurin arvo sekä tarkistusbittejä, mutta tällöin tarvitsisi lähettää useampia tavuja.




Kuva 5. Yhden tavun lähetys UART-muodossa 9600 bit/s nopeudella

Kuva 6. RF-moduuli AMB8420

## 2.6 MSP430-mikrokontrolleri ja ohjelma

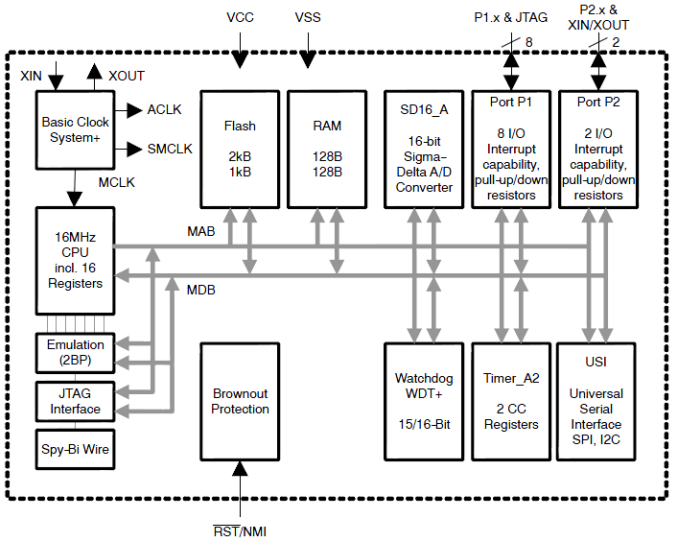
Laitteen mikrokontrollerina on kehitystyökalun irrotettavalla kohdekortilla oleva MSP430F2013. Mikrokontrolleri vastaanottaa valoinformaation, ja sille ladattu ohjelma laskee havaittujen valoimpulssien ja reaaliaikakellon (RTC) avulla kulutetun sähköenergian määrän sekä tehon, ja lähettää tämän tiedon LCD-näytölle. Tieto voidaan lähettää ulos myös UART-muodossa (Universal asynchronous receiver/transmitter) pinnin 3 kautta, joka voidaan suoraan liittää esim. langattomaan lähetinmoduuliin (Amber wireless). Ohjelma käyttää keskeytyksiä tulosignaalien (fototransistori ja kytkin) havaitsemiseen sekä UART-lähetyksen ja RTC:n toteutukseen. Tällä tavoin mikrokontrolleria voidaan pitää virransäästötilassa suurimman osan aikaa, eli silloin, kun tulojen tiloissa ei tapahdu muutoksia tai ajastinlaskuria ei tarvitse kasvattaa.

## 2.7 Kehitystyökalu ja ohjelmointiympäristö

Laitteen ohjelmointiin on käytetty tietokoneen USB-porttiin kytkettävää Texas Instrumentsin eZ430-F2013 -kehitystyökalua, jonka avulla voidaan helposti ladata käännetty ohjelmakoodi mikrokontrollerille sekä myös testata sitä. Ohjelma on tehty C-kielellä ja käännetty sekä ladattu mikrokontrollerille käyttämällä IAR Embedded Workbench -ohjelmointiympäristöä (IDE).

# 3 MSP430F2013-MIKROKONTROLLERI

MSP430 on Texas Instrumentsin valmistama mikrokontrolleriperhe, jonka suoritin perustuu 16-bittiseen RISC-arkkitehtuuriin. Erittäin alhaisen tehonkulutuksen ansiosta se soveltuu hyvin paristokäyttöisiin laitteisiin. Työssä käytettävä malli MSP430F2013 sisältää itse suorittimen lisäksi mm. 10 I/O-pinniä, 2 kilotavua Flash-muistia, 128 tavua RAM-muistia, 16-bittisen A/D-muuntimen, sisäisen oskillaattorin ja 16-bittisen ajastimen. Siinä on myös sisäiset sensorit lämpötilan ja käyttöjännitteen mittaamiseen. Sen käyttöjännitealue on 1.8 V – 3.6 V ja virrankulutus aktiivitilassa 1 MHz taajuudella ja 3.0 V käyttöjännitteellä on n. 300 µA. Mikrokontrollerin kellomoduulilla on monipuoliset ohjelmallisesti säädeltävät käyttöominaisuudet, ja suoritinta voidaan käyttää eri kellotaajuuksilla, joko sisäisen oskillaattorin tai ulkoisen kiteen tahdistamana. Asettamalla kellotaajuus esimerkiksi 4 kHz:iin, saadaan aktiivitilan virrankulutus laskettua alle 2 mikroampeeriin, kun taas maksimikellotaajuudella eli 16 MHz:llä virrankulutus nousee muutamaan milliampeeriin. Suoritin voidaan asettaa myös viiteen erilaiseen virransäästötilaan, jolloin virrankulutus voi laskea jopa alle yhden mikroampeerin [1]. Työssä käsiteltävä laite on asetettu käyttämään 1 MHz kellotaajuutta sisäisen digitaalisesti ohjatun oskillaattorin (DCO) tahdistamana.




Kuva 7. MSP430F2013:n lohkokaavio [1]

## 3.1 Keskeytykset ja virransäästötilat

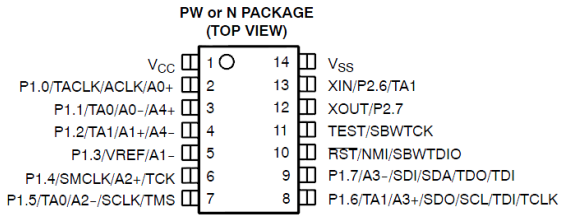
Keskeytykset ovat oleellinen osa sulautettuja järjestelmiä ohjelmoitaessa. MSP430F2013 voidaan ohjelmoida tuottamaan keskeytyksen esim. jos jonkin tulon tila muuttuu, A/D-muunnos on tehty tai ajastinlaskuri on edennyt haluttuun lukemaan. Keskeytysten käyttäminen näihin toimintoihin on yleensä toimivampi ja luotettavampi ratkaisu, kuin että pääohjelmassa jatkuvasti tarkistettaisiin eri muuttujien tiloja. Tämän lisäksi on mahdollista asettaa mikrokontrolleri virransäästötilaan silloin, kun mitään ohjelmakäskyjä ei tarvitse suorittaa, ja keskeytysten avulla voidaan palata takaisin aktiivitilaan silloin, kun halutaan jatkaa ohjelman suorittamista. Kaikissa viidessä virransäästötilassa itse suoritin (CPU) on pysäytetty, joten ohjelmakoodia ei suoriteta, lisäksi eri virransäästötiloissa on mahdollista pysäyttää myös erilliset kellosignaalit MCLK, SMCLK, ACLK, DCO ja kideoskillaattori [1].

## 3.2 Ajastimet

MSP430F2013 sisältää monipuolisilla toiminnoilla varustetun 16-bittisen ajastimen *Timer_A*, jota voidaan käyttää mm. intervalliajastimena tai PWM-generaattorina, se voidaan tahdistaa joko sisäisen tai ulkoisen kellosignaalin mukaan, ja se voidaan asettaa suorittamaan keskeytyksen halutuin aikavälein kahden eri capture/compare -rekisterin avulla tai TAR-rekisterin ylivuodon tapahtuessa. Timer_A:n lisäksi mikrokontrollerissa on myös *Watchdog timer*, jota voidaan käyttää vahtiajastimena tai yleisenä ajastimena [1]. Asettamalla ajastin tuottamaan keskeytyksen tasaisin väliajoin, voidaan toteuttaa esimerkiksi reaaliaikakello, laskea erilaisiin tapahtumiin kuluvaa aikaa tai tuottaa halutun taajuisia signaaleja.

### 3.3 Portit ja I/O-pinnit

MSP430F2013 sisältää 10 I/O-pinniä, jotka muodostavat 8-bittisen portin P1 sekä 2-bittisen portin P2. Jokainen kymmenestä I/O-pinnistä voidaan asettaa erikseen joko tuloksi tai lähdöksi, sekä eri pinnit voidaan asettaa toimimaan myös jossain erityistehtävässä, kuten 16-bittisen A/D-muuntimen tuloina, ulkoisen kellosignaalin tulona, PWM-lähtönä, SPI tai I2C tiedonsiirrossa (käytetään mm. laitteen ohjelmointia varten) tai Reset toimintona. Tulot voidaan myös ohjelmoida aiheuttamaan keskeytyksen joko nousevalla tai laskevalla reunalla. Pinnit voidaan asettaa käyttämään haluttaessa sisäisiä n. 35 kΩ:n ylös-/alasvetovastuksia, jolloin niihin on mahdollista liittää kytkimiä ilman ulkoisia ylösvetovastuksia. Lähdöistä saadaan tarpeeksi virtaa esim. 1-2 LEDin päälle pitämiseen ilman ulkoista vahvistusta, mutta jos lähdöistä tarvittava kokonaisvirta nousee yli 20 mA:n, niin niiden jännite alkaa jo selvästi laskea, jolloin olisi hyvä vahvistaa virtaa ulkoisella transistorilla [1].

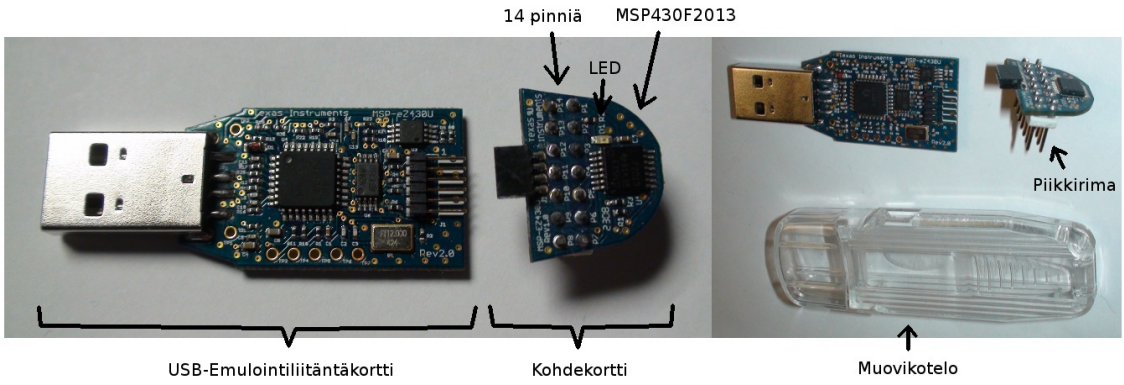



Kuva 8. MSP430F20x3:n pinnijärjestys [1]

# 4 EZ430-F2013 -KEHITYSTYÖKALU JA OHJELMOINTIYMPÄRISTÖ

Laitteen toteuttamisessa on käytetty Texas Instrumentsin eZ430-F2013 -kehitystyökalua, joka mahdollistaa helpon ja edullisen tavan MSP430-pohjaisten projektien kehittämiseen ja testaukseen. eZ430-F2013 on tietokoneen USB-porttiin liitettävä muistitikun kokoinen moduuli, joka koostuu kahdesta osasta, USB-emulointiliitäntäkortista sekä kohdekortista. Kohdekortti sisältää itse MSP430F2013-mikrokontrollerin ja siihen liitetyn LED-valon. Emulointiliitäntäkortti sisältää tarvittavan logiikan mikrokontrollerin ohjelmointia, emulointia ja debuggausta varten, sekä antaa laitteelle tarvittavan 3 voltin käyttöjännitteen hyödyntäen USB-portista saatavaa 5 voltin jännitettä [2]. Moduulia voidaan käyttää sellaisenaan ohjelmien testaukseen, esimerkiksi kohdekortilla olevan LED-valon vilkuttamiseen, tai emulaattorin ja mukana tulevan ohjelmiston avulla debuggaukseen, jolloin voidaan suorittaa koodia MSP430:lla vaikkapa yksi käsky kerrallaan ja lukea sen muistin sisältöä samalla.

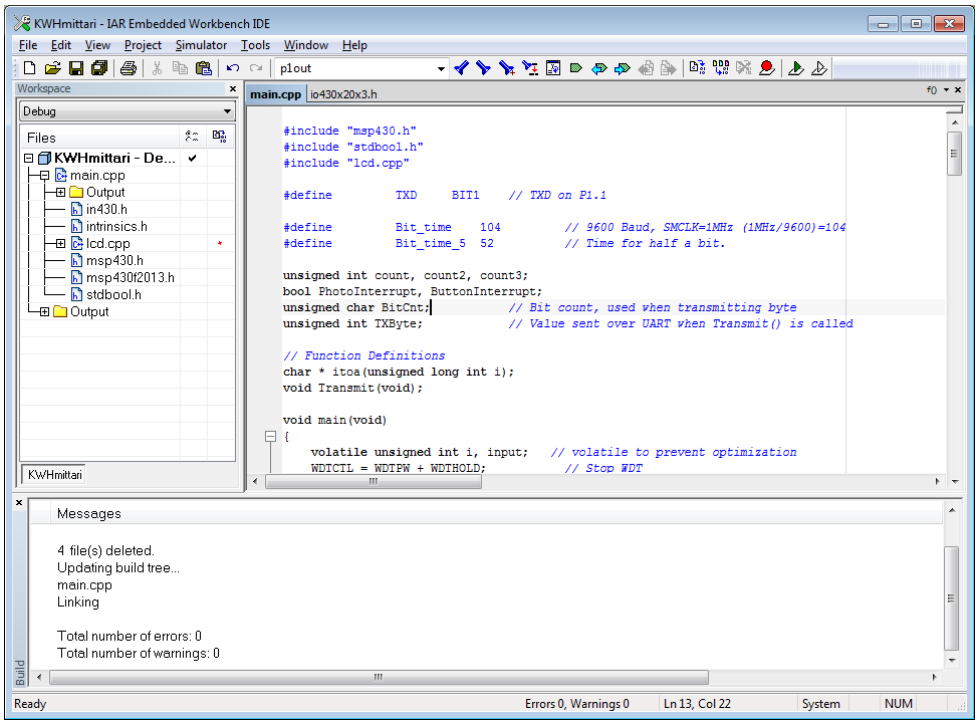
Kortit ovat läpinäkyvän muovikotelon sisällä, joka on mahdollista avata ja sulkea, jolloin päästään käsiksi mikrokontrollerin kaikkiin 14 pinniin, ja näin esimerkiksi ulkoisten komponenttien tai johdinten juottaminen suoraan piirilevylle on mahdollista. Pelkän MSP430:n ja LED-valon sisältävä kohdekortti voidaan myös irrottaa USB-tikun emulointiosasta ja käyttää sitä sellaisenaan itsenäisen laitteen osana. Työssä kohdekortille on juotettu 14-pinninen piikkirima, jonka avulla se on helppo liittää lukijalaitteen kantaan testausta varten ja takaisin USB-kehitystyökaluun uudelleenohjelmointia varten.




Kuva 9. eZ430-F2013 -kehitystyökalu

## 4.1 Ohjelmointiympäristö

Laitteen ohjelmointiin on käytetty kehitystyökalun mukana tulevaa IAR Embedded Workbench -nimistä integroitua kehitysympäristöä (IDE). Sitä voidaan käyttää mm. ohjelmakoodin kirjoittamiseen, kääntämiseen C-, tai assembly-kielestä, ohjelman lataamiseen MSP430-mikrokontrollerille, debuggaukseen sekä simulointiin. Mukana tuleva ilmaisversio ohjelmasta on rajoitettu 4 kilotavun ohjelmakoodiin, joka riittää kuitenkin yksinkertaisille sovelluksille ja MSP430F2013 sisältääkin vain 2 kilotavua ohjelmamuistia.




Kuva 10. IAR Embedded Workbench for MSP430 (ver 5.51.3)

Uusi projekti voidaan aloittaa seuraavilla valinnoilla:

- Avataan uusi workspace: *File → New → Workspace*
- Uusi projekti: *Project → Create New Project (C++ -> Main)*
- Asetukset: *Project → Options*
  *→ General Options → Target → Device → MSP430F2013*
  *→ Debugger → Setup → Driver → FET Debugger*

Kun koko projektin ohjelmakoodi halutaan kääntää/linkittää ja tarkistaa virheiden varalta, valitaan käsky *”Project → Rebuild All”*.

## 4.2 Tiedonsiirto ja testaus

Kun käännetty ohjelmakoodi halutaan ladata MSP430-mikrokontrollerille, tarvitsee eZ430-F2013 -kehitystyökalu olla ensin asetettuna tietokoneen USB-porttiin, ja sitten ajetaan ohjelman valikosta käsky *”Project → Download → Download active application”,* tai *”Project → Download and Debug”* mikäli halutaan siirtyä suoraan Debug-tilaan. Tämän jälkeen ohjelmaa voidaan suorittaa ja debuggata kehitysympäristössä, taikka kohdekortti voidaan irrottaa USB-tikusta ja asettaa se varsinaiseen laitteeseen, jolloin voidaan testata siihen liitettyjä oheislaitteita kuten LCD-näytön ohjausta ja fototransistorin tilan lukemista.

Työssä tehdyn ohjelman testaus on tehty lähinnä ilman debuggerin käyttöä, kun MSP430 on kytketty suoraan itse laitteeseen ja testattu fototransistorin lukemista, sähkön kulutuksen laskuria, LCD-moduulin ohjausta ja kytkimen käyttöä. Ohjelmaa testattaessa on käytetty myös apuna kohdekortilla olevaa LEDiä, jonka tilaa voidaan muuttaa käskyllä *”P1OUT ^= BIT0;”,* kun ohjelmakoodin suorittamisessa saavutetaan tietty kohta.

Jos halutaan nähdä kuinka paljon käännetty ohjelma vie tilaa mikrokontrollerin muistista, voidaan valita *Project → Options → Linker → List → Generate linker listing* asetus päälle, ja kun koodi käännetään nyt, tuottaa ohjelma .map tiedoston jonka lopusta voidaan lukea seuraavanlaiset tiedot:

```
 1 640 bytes of CODE  memory
    91 bytes of DATA  memory (+ 24 absolute )
    27 bytes of CONST memory
```

Tästä nähdään, että Flash-muistin 2048 tavusta on 1640 tavua käytetty ohjelmakoodille (sis. keskeytysvektorit) ja 27 tavua vakioille (const), kuten esim. *PrintStr("Wh");* käskyn tekstiosalle. 128 tavun RAM-muistista on 91 tavua muuttujien ja pinon (stack) käytössä.

# 5 OHJELMAN TOIMINTA

MSP430F2013-mikrokontrollerille tehdyn ohjelman keskeisenä osana on keskeytysten käyttö, jolloin laite voi olla virransäästötilassa suurimman osan aikaa. Keskeytys tapahtuu silloin, kun fototransistori vastaanottaa valosignaalin, kytkin vapautetaan tai ajastinlaskuri saavuttaa vertailurekisterin arvon. Keskeytyksen tapahtuessa mikrokontrolleri herää virransäästötilasta ja alkaa suorittaa kyseiselle keskeytykselle määriteltyä ohjelmaa. Fototransistori-tulon muuttuessa nollaksi tai kytkin-tulon muuttuessa ykköseksi, suoritetaan keskeytysohjelma Port_2, joka tarkistaa, kumpi tulo aiheutti keskeytyksen (BIT6 tai BIT7), jonka jälkeen keskeytyslippu P2IFG nollataan ja asetetaan muuttuja PhotoInterrupt tai ButtonInterrupt TRUE-tilaan. Tämän jälkeen asetetaan virransäästötila pois päältä ja palataan suorittamaan pääohjelmaa, jossa suoritetaan kyseiseen keskeytykseen liittyvä ohjelma. Ajastinkeskeytyksen kohdalla puolestaan suoritetaan keskeytysohjelma Timer_A, joka päivittää reaaliaikakelloa, sekä tarvittaessa lähettää bitin UART-lähtöön, mikäli Transmit-aliohjelmaa on kutsuttu. Ajastinkeskeytysohjelmassa tarkistetaan myös, onko kytkin ollut painettuna yli puoli sekuntia, ja mikäli näin on tai reaaliaikakellolle tulee sekunti kuluneeksi, asetetaan virransäästötila pois päältä ja palataan suorittamaan pääohjelmaa. Muussa tapauksessa keskeytysohjelmasta poistuttaessa palataan takaisin virransäästötilaan. Pääohjelmassa tarkistetaan, mitkä keskeytykset ovat tapahtuneet, ja suoritetaan siihen liittyvät toiminnot, jonka jälkeen asetetaan mikrokontrolleri takaisin virransäästötilaan odottamaan seuraavaa keskeytystä. Kuvassa 11 on esitetty ohjelman vuokaavio, joka kuvaa mitä toimintoja ohjelma suorittaa kunkin keskeytyksen kohdalla.

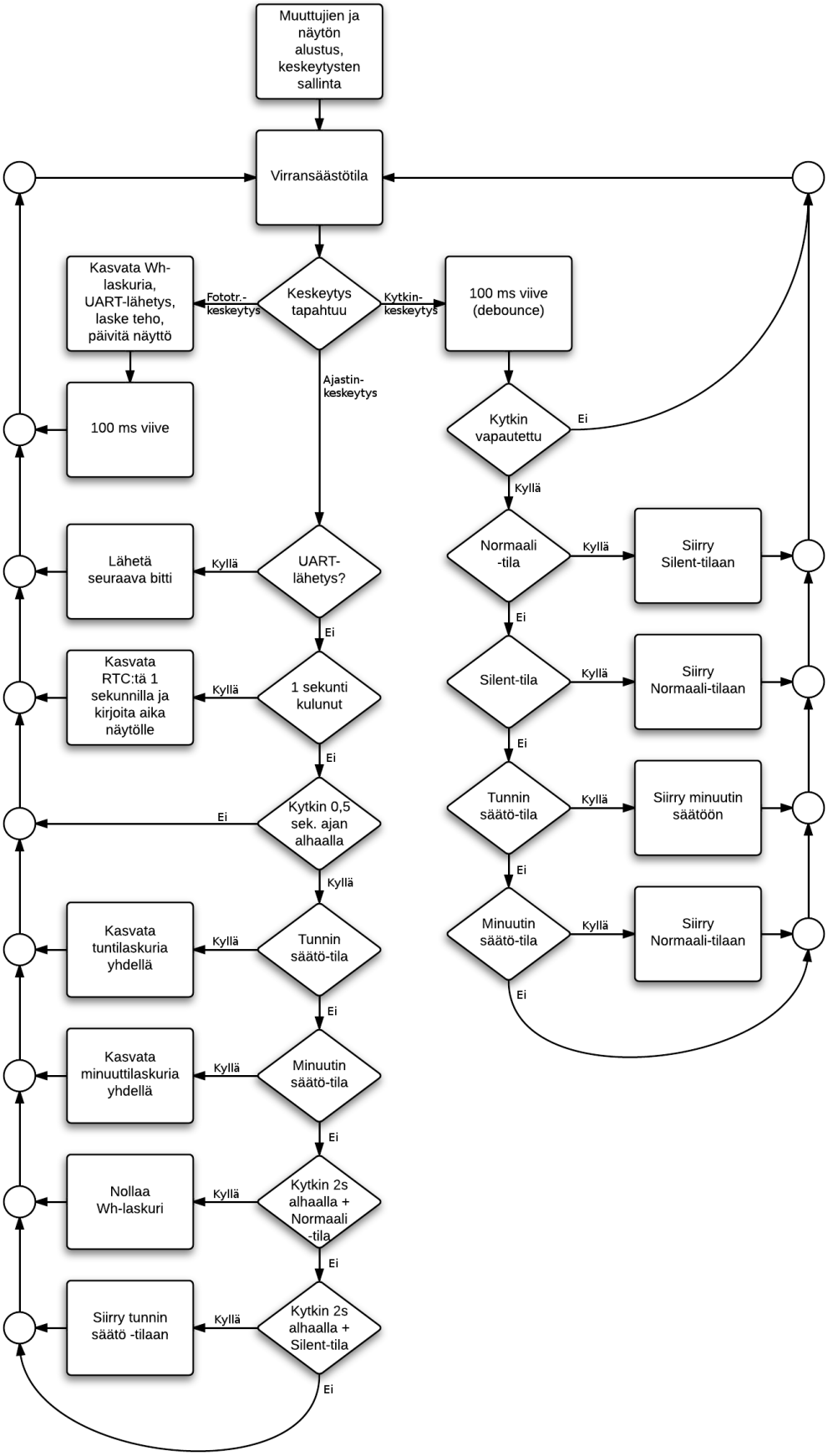
Ajastin Timer_A on asetettu ns. *vertailutilaan,* jossa se aiheuttaa keskeytyksen joka 104:s mikrosekunti. Tämä siksi, että saataisiin UART-lähetykselle soveltuva 9615 bittiä sekunnissa lähettävä tiedonsiirtonopeus. Tässä laitteessa samaa keskeytystä käytetään myös reaaliaikakellon toteutukseen, kytkinviiveiden ja hetkellisen tehon laskemiseen.

Reaaliaikakello on toteutettu siten, että ajastinkeskeytys kasvattaa laskuria kunnes 9615 kertaa 104 mikrosekuntia eli 1 sekunti on kulunut, jonka jälkeen ajetaan pääohjelman silmukka jossa kasvatetaan sekunti-muuttujan arvoa ja tarvittaessa myös minuutti, tunti ja päivä -muuttujia, sekä kirjoitetaan kellonaika LCD-näytölle.

Mikäli fototransistorikeskeytys on asettanut muuttujan PhotoInterrupt TRUE-tilaan, pääohjelmassa kasvatetaan Wh-laskurin lukemaa yhdellä ja kirjoitetaan LCD-näytölle tämä lukema. Näytölle kirjoitetaan myös tehon arvo, joka on laskettu kahden edellisen fototransistorikeskeytyksen välissä kuluneen ajan perusteella, jota mittaa ajastinkeskeytyksen avulla laskuri count_phototime. PhotoInterrupt-silmukassa lähetetään myös kaksi tavua, kirjain ”W” sekä Wh-laskurin 8 viimeistä bittiä UART-lähtöön. Tämä tehdään asettamalla lähetettävän tavun 8 bittiä muuttujaan *TXByte* ja kutsumalla Trasmit()-aliohjelmaa, jossa *TXByte*:en lisätään alku- ja loppubitit sekä asetetaan seuraavat 10 ajastinkeskeytystä lähettämään nämä bitit yksitellen UART-lähtöön 104 us välein.

Kytkinkeskeytyksen tapahtuessa, tai jos kytkintä on pidetty alas painettuna 0,5 sekuntia, siirrytään pääohjelmassa kytkimen tilaa käsittelevään osaan, ja tapahtumasta riippuen muutetaan näytön tila joko päälle tai pois, siirrytään kellon asetustilaan, muutetaan kellonaikaa tai nollataan Wh-laskuri. Kytkinkeskeytyksiä käsiteltäessä on tärkeää suodattaa pois kytkinvärähtelyt, joka on tässä ohjelmassa toteutettu yksinkertaisesti pitämällä 100 millisekunnin tauko kytkinkeskeytyksen tapahtuessa.

LCD-näytön toimintaa ohjataan tiedostossa lcd.cpp olevien aliohjelmien avulla. Ohjelman alussa suoritettava InitializeLcm()-aliohjelma asettaa ensin tiedonsiirron 4-bittiseen tilaan sekä suorittaa muut alustustoimenpiteet. Tämän jälkeen kaikki varsinaiset käskyt näytölle tehdään kutsumalla SendByte()-aliohjelmaa. Se asettaa ensin LCD-moduulin neljälle datalinjalle ohjauskäskyn tai lähetettävän merkin 4 eniten merkitsevää bittiä, sitten asettamalla RS-linja (Register Select Signal) nollaksi jos kyseessä on ohjauskäsky, tai ykköseksi jos kyseessä on merkin lähetys, ja sen jälkeen se antaa 200 mikrosekunnin pulssin E-linjalle (Enable Signal), jonka laskevalla reunalla LCD-moduuli vastaanottaa lähetetyt 4 bittiä. Samat toiminnot suoritetaan 4:n vähiten merkitsevän bitin lähettämiseksi.




Kuva 11. Ohjelman vuokaavio

# 6 C-KIELINEN OHJELMA

Tässä kappaleessa esitellään **main.cpp** tiedoston ohjelmakoodi. Se käyttää apuna lcd.cpp tiedostossa olevaa LCD-moduulin ohjauskoodia, joka on otettu osoitteesta [http://cacheattack.blogspot.fi/2011/06/quick-overview-on-interfacing-msp430.html], ja molemmat lähdetiedostot ovat myös liitteessä 3. UART-lähetyksen koodi on muokattu osoitteessa [http://www.msp430launchpad.com/2010/08/half-duplex-software-uart-on-launchpad.html] olevaa esimerkkiohjelmaa apuna käyttäen.

Ohjelman alussa sisällytetään siihen kääntäjän intrinsic-funktiot sisältävä in430.h, MSP430-sarjan perusmäärittelyt sisältävä otsikkotiedosto msp430.h, *bool*-muuttujatyypin käyttöön tarvittava otsikkotiedosto stdbool.h sekä LCD-moduulin ohjauskoodin sisältävä tiedosto lcd.cpp. Tämän jälkeen määritetään TXD eli UART-lähetys käyttämään pinniä P1.1 ja asetetaan Bit_time:n arvoksi 104, josta seuraa että ajastin Timer_A tuottaa keskeytyksen 104:n kellopulssin välein eli 1 MHz / 104 = 9615,38 kertaa sekunnissa, joka on tarpeeksi lähellä UART-lähetykselle tarvittavaa 9600 bit/s siirtonopeutta. RTC_freq:n arvoa käytetään laskemaan yhden sekunnin kestoa ajastinkeskeytysohjelmassa. Mikäli reaaliaikakello edistää tai jätättää, voidaan tätä arvoa muuttamalla säätää sekunnin pituutta.

```
#include "in430.h"                // intrinsic functions
#include "msp430.h"               // standard MSP430 definitions
#include "stdbool.h"              // bool data type
#include "lcd.cpp"                // functions to control the LCD-module

#define TXD       BIT1            // TXD on P1.1
#define Bit_time  104             // 9600 Baud, SMCLK=1MHz (1MHz/9600)=104
#define RTC_freq  9615            // modify to adjust RTC speed, default 9615 (= 1s / 104us)
```

Seuraavaksi määritellään ohjelman muuttujat ja esitellään aliohjelmat.

```
// Global variable definitions

unsigned long int count_wh = 0;     // Wh count
unsigned long int PhotoTime = 0;    // 104us ticks since last PhotoInterrupt, to calc Watts
unsigned long int count_phototime = 40000000; // counter for PhotoTime, gives 0 Watts at start

unsigned int count_sec = 0;         // Timer_A interrupt count, counts 1s = 9615 * 104us
unsigned int count_buttonhold = 0;  // Timer_A interrupt count, counts 0.5s = 4808 * 104us
unsigned int days = 0;              // days count

unsigned int TXByte;                // Value sent over UART when Transmit() is called
unsigned char BitCnt;               // Bit count, used when transmitting byte
```

```
unsigned char hours = 0;            // hours count
unsigned char minutes = 0;          // minutes count
unsigned char seconds = 0;          // seconds count, controlled by count_sec
unsigned char Mode = 0;             // 0=normal, 1=LCD off, 2=to 3, 3=set hours, 4=set minutes
unsigned char ButtonHoldTime = 0;   // Time the button has been down in 0.5sec units

bool SecondElapsed = FALSE;         // flag when count_sec in Timer_A reached 1 second
bool ButtonHoldCall = FALSE;        // flag when count_buttonhold in Timer_A reached 0.5s
bool PhotoInterrupt = FALSE;        // Phototransistor interrupt flag (from dark to light)
bool ButtonInterrupt = FALSE;       // Button interrupt flag (on release)

// Function Definitions

void PrintTime(void);               // print time on LCD
void PrintEnergy(void);             // print energy and power on LCD
char * itoa(unsigned long int i);   // convert unsigned long int(32bit) to ascii string
void Transmit(void);                // transmits character from TXByte
```

Pääohjelman alussa asetetaan watchdog-ajastin pois päältä ja asetetaan mikrokontrolleri toimimaan sisäisen DCO-oskillaattorin tahdistamana 1 MHz taajuudella. Sitten asetetaan portin 1 pinnit lähdöiksi ja portin 2 pinnit tuloiksi sekä asetetaan P2.7 eli kytkintulon ylösvetovastus toimimaan. P2.6 ja P2.7 -pinnit määritellään aiheuttamaan keskeytyksen, P2.6 laskevalla reunalla eli kun fototransistori havaitsee valoa ja P2.7 nousevalla reunalla eli kun kytkin vapautetaan. Kun keskeytykset on sallittu, kutsutaan LCD-moduulin alustuskomentoa ja tyhjennetään näyttö. UART-lähtöön lähetetään kirjain ”A” aliohjelmalla Transmit(), joka samalla myös alustaa ajastimen ja mahdollistaa ajastinkeskeytykset. Ennen pääsilmukkaan menoa kirjoitetaan vielä näyttöön kellonaika (00:00:00).

```
void main(void)
{
      WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
      BCSCTL1 = CALBC1_1MHZ;        // Set range
      DCOCTL = CALDCO_1MHZ;         // SMCLK = DCO = 1MHz

      P1SEL = TXD;                  // P1.1 connected to timer(UART)
      P1DIR = 0x0FF;                // P1 all outputs
        P2SEL = 0;                  // digital i/o
        P2DIR = 0;                  // P2 all inputs
        P2REN |= BIT7;              // P2.7 pull-up resistor enabled
      P2IES = BIT6;                 // Int edges: light On(P2.6=0) or button Released(P2.7=1)
      P2IFG &= ~(BIT6 + BIT7);      // P2.6 + P2.7 IFG cleared
      P2IE |= BIT6 + BIT7;          // P2.6 + P2.7 interrupt enabled

      __bis_SR_register(GIE);       // interrupts enabled

      InitializeLcm();              // initialize LCD-module
      ClearLcmScreen();
      TXByte = 65;                  // Transmit byte "A" to TXD (UART) and activate timer
      Transmit();
      PrintTime();                  // print 00:00:00 on LCD

      while(1)
      {
```

Pääsilmukassa ensimmäiseksi tarkistetaan onko ajastinkeskeytysohjelmassa oleva laskuri saavuttanut 1 sekunnin rajan, jolloin SecondElapsed-lippu menee TRUE-tilaan. Mikäli laite ei ole kellon asetustilassa, kasvatetaan reaaliaikakellon arvoa yhdellä sekunnilla, ja kirjoitetaan uusi aika LCD-näytölle ellei se ole silent-tilassa. Lopuksi nollataan SecondElapsed-lippu ettei tätä if-silmukkaa suoriteta uudestaan ennen kuin seuraava sekunti on kulunut.

```
if((SecondElapsed) && (Mode < 2)) { // update time when 1 second elapsed
      seconds++;              // increment seconds counter
      if(seconds == 60) {
        minutes++;            // minutes++
        seconds = 0;
        if(minutes == 60) {
          hours++;            // hours++
          minutes = 0;
          if(hours == 24) {
            days++;           // days++
            hours = 0;
          }
        }
      }
      if(Mode == 0) {         // print time only in normal mode
        PrintTime();
      }
      SecondElapsed = FALSE;  // clear flag
}
```

Seuraava if-silmukka suoritetaan 0,5 sekunnin välein mikäli ajastinkeskeytysohjelma on havainnut että kytkin on ollut alas painettuna vähintään 0,5 sekunnin ajan (ButtonHoldCall-lippu ylhäällä). Tämä tapahtuma voi käynnistää erilaisia toimintoja riippuen laitteen sen hetkisestä tilasta (Mode):

1. Tunninasetus-tilassa ohjelma kasvattaa kellon tuntimäärää kerran puolessa sekunnissa.
2. Minuutinasetus-tilassa ohjelma kasvattaa kellon minuuttimäärää kerran puolessa sekunnissa.
3. Normaalitilassa ohjelma nollaa Wh-laskurin mikäli kytkin on ollut painettuna 2 sekuntia.
4. Silent-tilasta laite siirtyy tilaan 3, jos kytkin on ollut painettuna 2 sekuntia. Tila 3 on ns. välitila, jossa näyttö laitetaan takaisin päälle, ja siitä siirrytään tunninasetus-tilaan kun kytkin vapautetaan.

Silmukan lopuksi nollataan ButtonHoldCall-lippu.

```
if(ButtonHoldCall) {          // called once every 0.5sec if button is held down
            ButtonHoldTime++; // increase ButtonHoldTime
            if(Mode == 3) {         // setup hours -mode, button down for >0.5s
              hours++;
              if(hours == 24) {
                hours = 0;
              }
              PrintTime();
            }
            else if(Mode == 4) {    // setup minutes -mode, button down for >0.5s
              minutes++;
              if(minutes == 60) {
                minutes = 0;
              }
              PrintTime();
            }
            else if(ButtonHoldTime == 4) {      // if button held down for 2 sec
              if(Mode == 0) {       // from normal mode
                count_wh = 0;       // reset Wh counter
                ClearLcmScreen();
                PrintEnergy();
                PrintTime();
              }
              else if(Mode == 1) {        // from lcd off mode
                Mode = 2;                 // to "wait for button release"-mode
                SendByte(0x0E, FALSE);    // LCD on, set cursor on, no blinking
                ClearLcmScreen();
                PrintEnergy();
                PrintTime();
              }
            }
            ButtonHoldCall = FALSE;       // clear flag
      }
```

ButtonInterrupt-silmukka suoritetaan, mikäli Port_2-keskeytys on havainnut, että kytkin vapautettiin. Seuraavaksi ohjelma odottaa 100 ms, jotta kytkinvärähtelyt saadaan suodatettua pois. Tämän jälkeen tarkistetaan vielä, että kytkin on todellakin vapautettu, ja jos näin on, niin vaihdetaan tilaa mikäli jokin seuraavista on tosi:

1. Mikäli laite on normaalitilassa ja kytkin oli painettua alle 2 sekuntia, vaihdetaan tilaksi silent-tila ja sammutetaan LCD-näyttö.
2. Mikäli laite on silent-tilassa ja kytkin oli painettua alle 2 sekuntia, siirrytään normaalitilaan ja laitetaan näyttö takaisin päälle.
3. Mikäli laite on ns. välitilassa (kytkin painettuna yli 2 s normaalitilassa), siirrytään tunninasetus-tilaan.
4. Mikäli laite on tunninasetus-tilassa ja kytkin oli painettuna alle 0,5 sekuntia, siirrytään minuutinasetus-tilaan.
5. Mikäli laite on minuutinasetus-tilassa ja kytkin oli painettuna alle 0,5 sekuntia, siirrytään takaisin normaalitilaan ja nollataan sekuntilaskuri.

Lopuksi nollataan kytkimen alhaallaolo-aikaa mittaavat laskurit ja ButtonInterrupt-lippu.

```
if(ButtonInterrupt) {
            __delay_cycles(100000);       // SW Delay to filter bouncing
            if (P2IN & BIT7) {            // Button released (not just bounce)

              // change display mode unless button was held down for long
              if((Mode == 0) && (ButtonHoldTime < 4)) {
                Mode = 1;                 // to silent mode
                SendByte(0x08, FALSE);    // LCD OFF
              }
              else if((Mode == 1) && (ButtonHoldTime < 4)) {
                Mode = 0;                 // to normal mode
                SendByte(0x0C, FALSE);    // LCD ON
                PrintEnergy();
                PrintTime();
              }
              else if(Mode == 2) {
                Mode = 3;                 // to setup hours -mode
                seconds=0;                // seconds to 00
                PrintTime();
              }
              else if((Mode == 3) && (!ButtonHoldTime)) {
                Mode = 4;                 // to setup minutes -mode
              }
              else if((Mode == 4) && (!ButtonHoldTime)) {
                Mode = 0;                 // to normal mode
                count_sec = 0;            // reset count_sec and flag
                SecondElapsed = FALSE;
                SendByte(0x0C, FALSE);    // cursor off
              }
            }
            // P1OUT ^= BIT0;             // Toggle P1.0 = LED for testing
            ButtonHoldTime = 0;           // reset hold timer
            count_buttonhold = 0;
            ButtonInterrupt = FALSE;      // clear interrupt flag
      }
```

PhotoInterrupt-silmukka suoritetaan, mikäli Port_2-keskeytys on havainnut, että fototransistori on havainnut valoa sen verran, että tulon tila on vaihtunut ykkösestä nollaksi. Tällöin kasvatetaan Wh-laskuria ja jos ei olla silent-tilassa, niin kirjoitetaan Wh-laskurin ja tehon arvot LCD-näytölle. Sitten lähetetään kirjainta ”W” vastaava ASCII-koodi sekä Wh-laskurin viimeisistä 8 bitistä muodostettu tavu UART-lähtöön. Tämän jälkeen pidetään 100 ms viive, jotta vältyttäisiin fototransistorin tilan liian nopeilta muutoksilta. Lopuksi PhotoInterrupt-lippu nollataan.

```
      if(PhotoInterrupt) {
            count_wh++;                   // increase Wh counter
            if(Mode != 1) {               // if not in silent mode then
              PrintEnergy();              // show Energy and Power on LCD
            }
            TXByte = 87;                  // transmit byte "W" to TXD (UART)
            Transmit();
            TXByte = count_wh & 0x0FF;    // transmit last 8 bits of Wh count
            Transmit();

            // SW Delay to limit sample rate below 10Hz = 36kW
            // to ignore too fast and unwanted light flickering
            __delay_cycles(100000);
            PhotoInterrupt = FALSE;       // Clear interrupt flag
      }
```

Pääsilmukassa asetetaan kursori päälle tunti- tai minuuttiluvun kohdalle, mikäli laite on tunnin- tai minuutinasetus-tilassa. Jos tässä vaiheessa jokin aiempien if-silmukoiden lipuista on noussut takaisin ylös, jatketaan ohjelman suorittamista pääsilmukan alusta, muussa tapauksessa asetetaan mikrokontrolleri virransäästötilaan käskyllä *__bis_SR_register(CPUOFF + GIE);* ja odottamaan uutta keskeytystä.

```
            if((Mode == 2) || (Mode == 3)) {    // setup hours -mode
                  LcmSetCursorPosition(1,9);    // move cursor to location of hours
            }
            if(Mode == 4) {                     // setup minutes -mode
                  LcmSetCursorPosition(1,12);   // move cursor to location of minutes
            }

            // if no other interrupts pending, then
            // disable CPU (LPM0) and continue when an interrupt happens

            if(!(SecondElapsed || ButtonHoldCall || ButtonInterrupt)) {
                  __bis_SR_register(CPUOFF + GIE);
            }

      }
}
```

PrintTime()-aliohjelma kirjoittaa näytölle reaaliaikakellon ajan muodossa hh:mm:ss näytön oikeaan alareunaan sekä kuluneiden päivien lukumäärän näytön oikeaan yläkulmaan.

```
// print time on LCD
void PrintTime()
{
      if(days < 10)
            LcmSetCursorPosition(0,14);
      else if(days < 100)
            LcmSetCursorPosition(0,13);
      else
            LcmSetCursorPosition(0,12);
      PrintStr(itoa(days));               // print days
      PrintStr("d");

      LcmSetCursorPosition(1,8);
      if((hours)<10) {                    // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(hours));              // hours

      PrintStr(":");
      if((minutes)<10) {                  // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(minutes));            // minutes

      PrintStr(":");
      if((seconds)<10) {                  // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(seconds));            // seconds
}
```

PrintEnergy()-aliohjelma kirjoittaa LCD-näytölle lasketun sähkönkulutuksen wattitunteina, sekä sen hetkisen tehon arvon. Tehon arvo saadaan laskettua, kun tiedetään, kuinka pitkä aika on kulunut kahden viimeisen valosignaalin välillä, joka vastaa siis 1 Wh:n kulutusta. Fototransistorikeskeytyksen ja ajastinkeskeytyksen avulla on talletettu tämä aika muuttujaan PhotoTime. Jokainen PhotoTime:n yksikkö vastaa 104 mikrosekuntia, josta voidaan laskea että teho watteina on 34615385 / PhotoTime.

```
// print Energy and Power on LCD
void PrintEnergy()
{
      LcmSetCursorPosition(0,0);
      PrintStr("E=");                    // print energy on first line
      PrintStr(itoa(count_wh));
      PrintStr("Wh");
      LcmSetCursorPosition(1,3);
      PrintStr("     ");                 // clear area before rewrite
      LcmSetCursorPosition(1,0);
      PrintStr("P=");                    // print power on second line

      // using the number of 104us ticks elapsed from previous PhotoInterrupt to calculate power
      // E = P * t
      // P = E / t
      // P = 1Wh / (X * 104us)
      // P = (1Wh / 104us) / X
      // P = (3600Ws / 0.000104s) / X
      // P = 34615385W / X
      PrintStr(itoa(34615385 / PhotoTime));
      PrintStr("W");
}
```

Aliohjelmaa itoa() käytetään kokonaisluku-tyyppisen muuttujan muuntamiseen ASCII-tekstimuotoon, jotta LCD-näytölle kirjoittava aliohjelma PrintStr() osaisi käsitellä sen. IAR:n peruskirjastoista ei löydy suoraan tätä funktiota ja sprintf:llä toteutettuna se veisi liian paljon tilaa ohjelmamuistista. Kyseiselle funktiolle on kuitenkin löydettävissä lukuisia eri toteutustapoja, joista tässä käytössä yksinkertainen mutta toimiva versio.

```
// convert unsigned long int(32bit) to ascii string
char * itoa(unsigned long int i)
{
      static char buf[11];
      char* bp = buf+sizeof(buf);
      *--bp = '\0';
      do
        *--bp = i%10+'0';
      while (i /= 10);
      return bp;
}
```

Transmit()-aliohjelmaa kutsutaan kun halutaan lähettää tavu UART-lähdöstä (P1.1). Se myös asettaa Timer_A ajastimen tuottamaan keskeytyksiä 104 us välein, jonka avulla se voi lähettää bittejä 9615 Hz taajuudella. Aliohjelman alussa asetetaan TACCTL0 eli capture/compare-ohjausrekisterin OUT-bitti ykköseksi ja asetetaan ajastimen ohjausrekisteri TACTL käyttämään SMCLK:n 1 MHz kellotaajuutta jatkuvalla ylöspäin laskennalla. Lähetettävien bittien määräksi asetetaan 10 eli 8 databittiä sekä start ja stop -bitit. Vertailurekisteri TACCR0 alustetaan samalla arvolla, mikä on ajastimen 16-bittisessä laskurirekisterissä TAR, ja sen jälkeen siihen lisätään Bit_time:n arvo eli 104. Sitten lähetettävään tavuun TXByte lisätään start ja stop -bitit. Lopuksi TACCTL0 -rekisterillä asetetaan TACCR0:n tuloksi signaali CCIxB (ACLK), valitaan *output mode 1* eli lähtö ykköseksi ja sallitaan ajastinkeskeytykset. Sitten odotetaan, että ajastinkeskeytys on lähettänyt kaikki bitit.

```
// transmits character from TXByte
void Transmit()
{
      CCTL0 = OUT;                        // TXD Idle as Mark
      TACTL = TASSEL_2 + MC_2;            // SMCLK, continuous mode

      BitCnt = 0xA;                       // Load Bit counter, 8 bits + ST/SP
      CCR0 = TAR;                         // Initialize compare register

      CCR0 += Bit_time;                   // Set time till first bit
      TXByte |= 0x100;                    // Add stop bit to TXByte (which is logical 1)
      TXByte = TXByte << 1;               // Add start bit (which is logical 0)

      CCTL0 =  CCIS0 + OUTMOD0 + CCIE;    // Set signal, intial value, enable interrupts
      while ( BitCnt != 0);               // Wait for previous TX completion
      // Because we're using Timer_A interrupt also for the RTC, we continue here immediately
      // after the stop bit is set, to allow next Timer_A int to be ready for possible start bit
}
```

Port_2-keskeytysohjelma suoritetaan, kun fototransistori alkaa johtaa eli havaitsee valoa tai kun kytkin lakkaa johtamasta (kytkinvärähtelyn takia tämä tapahtuu yleensä useita kertoja sekä kytkintä painettaessa että vapautettaessa). Fototransistorikeskeytyksen ollessa kyseessä tallennetaan valoimpulssien välistä aikaa mittaavan laskurin arvo ja nollataan tämä laskuri, jolloin se alkaa mitata aikaa seuraavaa impulssiin asti. Sitten P2.6 -pinnin keskeytyslippu nollataan ja asetetaan PhotoInterrupt-lippu ylös, jotta pääohjelmassa oleva silmukka suorittaisi halutut toimenpiteet. Kytkinkeskeytyksen kohdalla ainoastaan nollataan P2.7 -lippu ja asetetaan ButtonInterrupt-lippu ylös sen käsittelysilmukkaa varten. Lopuksi asetetaan virransäästötila pois päältä, jotta pääohjelman suoritus voi jatkua ja käsitellä keskeytyksen aiheuttaneen tapahtuman.

```
// Port 2 interrupt service routine
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
      if(P2IFG & BIT6)                    // Phototransistor interrupt (from dark to light)
      {
            PhotoTime = count_phototime;  // save the time of interrupt
            count_phototime = 0;          // reset the time counter
            P2IFG &= ~BIT6;               // P2.6 IFG cleared
            PhotoInterrupt = TRUE;
      }
      if(P2IFG & BIT7)                    // Button interrupt (on release)
      {
            P2IFG &= ~BIT7;               // P2.7 IFG cleared
            ButtonInterrupt = TRUE;
      }
      __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to continue main loop
}
```

Timer_A aiheuttaa ajastinkeskeytyksen, kun sen laskuri TAR saavuttaa vertailurekisteri TACCR0:n arvon. Sitten TACCR0 -rekisteriin lisätään jälleen 104 ja tarkistetaan onko UART-lähetystä odottamassa yhtään bittejä. Jos on, niin lähetetään TXByte:n alin bitti lähtöön TACCTL0 -rekisterin kautta, ja mikäli bittejä on vielä lähetettävänä, siirretään seuraava bitti valmiiksi seuraavaa keskeytystä odottamaan.

```
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
{
      CCR0 += Bit_time;                   // Add Offset to CCR0
      if ( BitCnt != 0)                   // If there's bits to TX
      {
            CCTL0 |= OUTMOD2;             // Set TX bit to 0
            if (TXByte & 0x01)
                  CCTL0 &= ~ OUTMOD2;     // If it should be 1, set it to 1
            TXByte = TXByte >> 1;
            BitCnt--;
      }
```

Tämän lisäksi ajastinohjelmassa kasvatetaan laskuria *count_phototime,* joka mittaa aikaa kahden valosignaalin välillä, sekä laskuria *count_sec,* joka mittaa yhteen sekuntiin kuluvaa aikaa. Mikäli havaitaan, että sekunti on kulunut (ts. 9615 keskeytystä joista jokainen kestää 104 mikrosekuntia), nollataan sekunnin laskuri, asetetaan SecondElapsed-lippu ylös ja poistutaan virransäästötilasta, jotta pääohjelma voi suorittaa halutut toiminnot. Ajastinkeskeytyksessä tarkistetaan myös, mikäli kytkin on alas painettuna, jolloin kasvatetaan *count_buttonhold* -laskuria ja tarkistetaan onko tämä ollut painettuna puoli sekuntia. Jos on, niin nollataan laskuri, asetetaan ButtonHoldCall-lippu ylös, poistutaan virransäästötilasta ja palataan pääohjelmaa suorittamaan.

```
      count_phototime++;                 // counts 104us ticks since last PhotoInterrupt
      count_sec++;                       // RTC implementation using 104us counter
      if (count_sec == RTC_freq) {       // 9615 * 104us = 0.999960s
            count_sec = 0;
            SecondElapsed = TRUE;
            __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to update time in main loop
      }

      if (!(P2IN & BIT7)) {                          // if button down
            count_buttonhold++;
            if(count_buttonhold == 4808) {           // once in 0.5sec
                  count_buttonhold = 0;
                  ButtonHoldCall = TRUE;             // activate ButtonHoldCall
                  __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to main loop
            }
      }
}
```

# 7 TESTAUS JA TULOSTEN ANALYSOINTI

Laitteen testauksessa käytettiin apuna mm. signaaligeneraattoria, oskilloskooppia ja mikrokontrollerikortilla olevaa LEDiä. Laitteen toimintaa testattiin myös käytännössä Hager EC350/352 -energiamittarin kanssa. Vaahtomuovipalan sisällä oleva fototransistori asetettiin energiamittarin merkkiLEDiä vasten, ja valoherkkyyden säätövastuksen sopivalla asennolla saatiin todettua, että näissä testausolosuhteissa laite pystyy lukemaan luotettavasti energiamittarin antamien valoimpulssien määrää ja näistä laskettua sähköenergiankulutusta sekä näyttämään tämän tiedon LCD-näytöllä.

## 7.1 Fototransistorin herkkyys

Lukijalaitteen mittaustulosten oikeellisuus riippuu pitkälti siitä, että fototransistori pystyy havaitsemaan jokaisen valonvilkahduksen LED-valosta jota se tarkkailee, mutta tämä ei saisi tulkita mitään ulkopuolelta tulevan valon vaihteluita valosignaaliksi. Tämän vuoksi fototransistori pitäisi asettaa kohteeseen siten, että se on tarpeeksi suojassa ulkopuoliselta valolta, ja sen säätövastus pitäisi asettaa sopivalle herkkyydelle. Jos säätövastus on väärässä asennossa, tai ylimääräistä valoa on liikaa, saattaa mikrokontrollerin tulon looginen tila vaihdella satunnaisesti ykkösen ja nollan välillä tai pysyä koko ajan joko ykkösessä tai nollassa. Haitallista tulon tilan värähtelyä saattaisi pystyä vähentämään myös lisäämällä tuloon jonkinlaisen smith-trigger piirin, jossa olisi isompi hystereesi kuin mikrokontrollerin omassa tulossa. Eri fototransistorit ovat myös ominaisuuksiltaan ja häiriöherkkyydeltään erilaisia.

## 7.2 RF-lähetyksen testaus

Myös lukijalaitteen liittämistä Amberin RF-lähettimeen ja lähetetyn tiedon vastaanottamista pöytätietokoneella olevalla ohjelmalla testattiin. UART-muotoisen tiedon lähetys ja vastaanotto ei alussa heti sujunut ongelmitta, esim. oskilloskoopilla testatessa havaittiin, että yksittäinen bitti saattoi kadota siirtovälillä tai kellotaajuus ei ollut ihan oikealla taajuudella, mutta muutaman pienen ohjelmamuutoksen jälkeen ja

sopivilla asetuksilla saatiin yksittäisiä tavuja lähetettyä langattomalla yhteydellä melko hyvin. Havaittiin myös, että jos tietoa yritetään lähettää samaan aikaan kun vastaanottava Amber vielä käsittelee, mahdollisesti jostain toisesta laitteesta lähetettyä tietoa, jäävät muut samaan aikaan lähetetyt tietopaketit vastaanottamatta. Koska tiedonsiirto on tässä tapauksessa vain yksisuuntainen, ei lähettävä laite voi tarkistaa että menikö lähetys perille, joten jonkinlainen virheentarkistus olisi tehtävä vastaanottavan laitteen tai tietokoneen ohjelmistossa. Tämän vuoksi lukijalaite lähettääkin jokaisen valoimpulssin havaittuaan ensin merkin "W" ja perään viimeiset 8 bittiä Wh-laskurin arvosta, jotta vastaanottava sovellus voisi tarkistaa Wh-laskurin oikean arvon. RF-lähetystä ja -vastaanottoa ei kuitenkaan testattu kovin paljoa, mutta suurin osa kohdatuista ongelmista näyttäisi olleen peräisin ohjelmassa olevista puutteista.

### 7.3 RTC:n tarkkuus

Laitteen reaaliaikakellon tarkkuutta testattiin jonkin verran pitämällä laitetta päällä useita tunteja ja vertaamalla sen kelloa oikeaan aikaan. Perusasetuksilla reaaliaikakello edisti noin 1,5 sekuntia tunnissa. Säätämällä sekunnin laskemiseen käytettyä laskuria voidaan kuitenkin saada kello pysymään jonkin verran tarkemmin ajassa. Lisää tarkkuutta saataisiin käyttämällä tarkoitukseen sopivaa ulkoista kellokidettä.

Laite käyttää kellosignaalina sisäiseltä digitaalisesti ohjatulta oskillaattorilta (DCO) saatua 1 MHz:n signaalia. Tämän kellotaajuus ei välttämättä ole kuitenkaan erityisen tarkka, jos halutaan reaaliaikakellon (RTC) pysyvän oikeassa ajassa pitkällä aikavälillä. Myös lämpötila ja laitteen jännite voivat vaikuttaa taajuuteen. Ulkoinen kide voisi antaa paremman tarkkuuden.

Lisäksi ohjelman reaaliaikakello on toteutettu hieman poikkeuksellisella tavalla, käyttäen sen tahdistamiseen samaa Timer_A-keskeytystä, jota käytetään ensisijassa UART-lähetyksen tahdistamiseen 9615 Hz:n taajuudella. Näin ei tarvitse ohjelmoida toista ajastinkeskeytystä erikseen RTC:lle, mutta sekunnit pitää laskea käyttäen 104 mikrosekunnin välein kasvavaa laskuria. Tästä seuraa, että yhden sekunnin teoreettiseksi kestoksi saadaan 0,999960s tai 1,000064s, mutta kuten yllä mainittu,

muutkin ominaisuudet vaikuttavat kellon tarkkuuteen, ja todellinen sekunnin kesto saadaan vain testaamalla. Reaaliaikakellon nopeutta voidaan säätää 104us/1s hyppäyksin ohjelman alussa määritellyn RTC_freq:n arvolla. Jos kellon tarkkuutta haluttaisiin hienosäätää vielä tarkemmin, käyttämättä kuitenkaan toista ajastinkeskeytystä, voitaisiin esim. säätää Bit_time muuttujan arvoa välillä 103-105 sopivin väliajoin, jolloin eri sekuntien pituus vaihtelisi yhdellä mikrosekunnilla. Oikea luku riippuu siitä, kuinka paljon kello jättää/edistää, ja se pitäisi hakea pitkän aikavälin testauksella ja vertailulla oikeaan aikaan, ottaen huomioon myös mm. käytetyn kiteen, lämpötilan ja paristojännitteen vaikutukset kellon taajuuteen.

Ohjelmaan voisi myös lisätä automaattisen RTC:n kalibrointiominaisuuden, esim. antamalla kytkimellä kaksi signaalia tasan 24 tunnin välein, joiden avulla ohjelma voisi laskea montako sekuntia RTC:n virhe on ja tästä laskemalla se säätäisi itsensä lyhentämään tai pidentämään sekunnin kestoa sopivin väliajoin.

Kun sekunnin laskemiseen käytetään 9615 * 104us = 0,999960s laskuria, pitäisi reaaliaikakellon teoriassa edistää 3,5 sekuntia vuorokaudessa. Käytännön kokeilulla kuitenkin nähdään, että sisäisen oskillaattorin epätarkkuus aiheuttaa selvästi suuremman virheen reaaliaikakelloon. Säätämällä ohjelmassa käytettävää *RTC_freq*-muuttujaa, voidaan saada kello pysymään hieman paremmin ajassa. Sisäisen oskillaattorin antama tarkkuus on joka tapauksessa riittävä laitteen ensisijaisiin toimintoihin, kuten sähkönkulutuksen mittaamiseen, tehon laskemiseen ja UART-lähetykseen, mutta jos kellonaikaa halutaan pitää pitkällä aikavälillä tarkemmin ajassa, olisi ulkoisen 32768 Hz:n kiteen käyttäminen reaaliaikakellon signaalinlähteenä suositeltavaa.

Reaaliaikakellon ja UART-lähetyksen yhdistäminen käyttämään samaa ajastinkeskeytystä on teoriassa ihan toimiva idea, mutta Transmit()- ja ajastinkeskeytysohjelmien toimintaa tarkemmin tutkiessa tulee mieleen että ne eivät välttämättä toimi täysin tarkasti ohjelmassa olevalla toteutuksella. Transmit()-aliohjelmaa kutsuttaessa nimittäin alustetaan TACCR0:n arvo uudelleen, vaikka ajastin olisi jo laskemassa seuraavaa keskeytysajankohtaa, joten joka kerta kun Transmit()-aliohjelmaa kutsutaan, saattaa aiheutua enintään 104 mikrosekunnin virhe kellolle, joka tosin on melko huomaamaton. Toinen mieleen tuleva kysymys on että aiheuttaako

TAR-rekisterin ylivuoto ylimääräisen keskeytyksen aiheuttaen pienen virheen UART lähetyksessä, jos se sattuu samaan aikaan (todennäköisyys n. 1%?). Tämän vaikutuksesta pitäisi hankkia lisää tietoa testaamalla UART-lähetystä paremmin, mutta näyttäisi että toteutuksessa olisi vielä hieman parantamisen varaa, vaikka kyseiset virhetilanteet olisivatkin käytännön sovelluksessa huomaamattomia. Voidaankin todeta, että kun halutaan muokata valmista koodia uuteen käyttötarkoitukseen, täytyy olla tarpeeksi hyvin perillä kokonaisuuden toiminnasta, ja joskus voi olla parempi suunnitella koko funktio uudestaan rivi riviltä.

## 7.4 Virrankulutus

Mikrokontrollerin tarkan virrankulutuksen mittaaminen on vaikeaa. Laite on suurimman osan ajasta virransäästötilassa ja käyttää enemmän virtaa vain hetkittäin. Valmistajan antamien arvojen mukaan ja laitteen ohjelmakoodia tarkasteltaessa voidaan kuitenkin arvioida, että mikrokontrollerin keskikulutus olisi korkeintaan muutaman kymmenen mikroampeerin luokkaa, jolloin tavalliset (1500-3000mAh) AA-paristot saattaisivat pystyä pitämään sen käynnissä useita vuosia. Huomattavasti enemmän virtaa vievät kuitenkin LCD-näyttö ja RF-lähetin. Myös fototransistori ja kytkin vievät jonkin verran virtaa johtaessaan. Valmistajan tietojen mukaan LCD-moduuli kuluttaa 2 mA, joten jatkuvasti päällä ollessaan se kuluttaisi tavalliset AA-paristot loppuun noin kuukaudessa. Jos näyttöä kuitenkin pidettäisiin sammutettuna suurimman osan ajasta ja RF-lähetystä ei tarvita, tai se ohjelmoitaisiin lähettämään tietoja vain harvoin, saattaisi olla mahdollista, että yhdet AA-paristot kestäisivät yli vuodenkin.

Käytettäessä laitetta langattoman lähettimen kanssa voitaisiin säästää virtaa lähettämällä UART-lähtöön laskurin arvo ainoastaan esimerkiksi kerran 5 minuutissa, eikä jokaisen impulssin kohdalla.

Paristonvaihdosten tai muiden sähkökatkosten varalta voisi olla myös hyödyllistä ajoittain tallettaa energialaskurin ja kellon lukemat mikrokontrollerin Flash-muistiin, josta ne saadaan luettua laitteen käynnistyessä uudelleen. Täysin katkeamattoman virransyötön mahdollistaminen paristonvaihdon ajaksi voitaisiin toteuttaa myös esim.

isolla kondensaattorilla tai lisäämällä laitteeseen toinen paristokotelo ensimmäisen kanssa rinnakkain kytkettynä.

## 7.5 Jatkokehitys

Laitteeseen voisi tarvittaessa lisätä uusia ominaisuuksia:

- Vaihtoehtoiset tiedonlähetystilat, kuten esim. yksinkertainen pulssilähtö.
- Kulutuksen näyttö kuluvan tunnin, päivän tai itse valitun aikavälin sisällä.
- Max. ja min. hetkellinen teho tietyn aikavälin sisällä.
- RTC:n tarkkuuden parantaminen
- Kulutuksen tallennus esim. jokaisen tunnin aikana myöhempää tarkastelua varten ja USB-tiedonsiirtomahdollisuus (→ Excel).
- RF-lähetyksen kehitys: Tiedon lähetys paketeissa sopivin väliajoin, virheentarkistus ja kaksisuuntainen tiedonsiirto.
- GSM-moduuli.

MSP430F2013 sisältää 2 kilotavua ohjelmamuistia ja 10 I/O-pinniä. Nämä ovat riittävät laitteen nykyiselle versiolle, mutta haluttaessa lisää ominaisuuksia, saattaa tulla tarvetta isommalle muistille tai useammalle I/O-pinnille, jolloin esimerkiksi MSP430G2553 tai MSP430F2252 ovat hyviä vaihtoehtoja.

Huomioitavaa on, että työssä tehdyssä laitteessa piikkirima on juotettu eZ430-F2013:n kohdekortin alapuolelle, jolloin mikrokontrolleri asettuu piirilevylle peilikuvana verrattuna normaaliin asetelmaan, eikä layout kuva (liite 2) ole ihan kaikilta osin identtinen kuvissa (liite 1) näkyvän prototyypin kanssa.

Ohjelmassa käytetyt viiveet, kuten kytkinvärähtelyn ja valonvärähtelyn suodatuksessa käytetty 100 ms viive voitaisiin toteuttaa myös ajastinkeskeytyksen avulla, jolloin ei tarvitsisi jäädä yhteen kohtaa koodia paikoilleen niin pitkäksi aikaa ja voitaisiin olla enemmän aikaa virransäästötilassa.

Kytkimen tilan havainnoinnissa saattaa tulla häiriöitä mikäli sen värähtelyä ei suodateta kunnolla. Testaamalla havaittiin että 100 ms viive kytkinkeskeytyksen jälkeen suodattaa hyvin käytetyn kytkimen värähtelyt, mutta esim. 50 ms olisi liian lyhyt aika tähän. Jollakin toisella kytkimellä voi olla tarpeellista käyttää vieläkin pidempää viivettä.

2 kilotavun Flash-ohjelmamuistin lisäksi MSP430F2013:ssa on myös 256 tavun Flash-muistialue nimeltään *information memory* osoitteissa 01000h – 010FFh. Tämän alueen käyttämiseen tarvitsee hieman tutkia millä kääntäjän komennoilla sinne voidaan kirjoittaa tietoja (http://processors.wiki.ti.com/index.php/Placing_Variables_in_Specific_Memory_Location_-_MSP430), mutta sopivilla käskyillä sinne pitäisi olla mahdollista tallentaa ohjelmakoodiakin (http://supp.iar.com/Support/?note=39271).

# 8 YHTEENVETO

Sähkömittarin lukijalaitteen kehittäminen ja testaaminen antoi hyvän perehdytyksen sulautettujen järjestelmien keskeisiin aiheisiin ja MSP430:n ohjelmointiin. Vaikka laitteen toiminta on yksinkertainen, tuli toteutusvaiheessa kuitenkin eteen monia ongelmatilanteita, joiden tutkiminen ja ratkaiseminen antoi paljon lisätietoa aihepiiristä.

Testattaessa valmis laite toimi hyvin suunnitellussa tarkoituksessaan, tosin pitkäaikaista käytännön testausta ei vielä tehty, ja joitakin ominaisuuksia laitteessa voisi vielä parantaa. Lukijalaitteen mittausarvojen luotettavuuden kannalta on tärkeää, että fototransistori on asennettu sähkömittarin LED-valon lähelle siten, että se on riittävästi suojattu ulkopuoliselta valon vaihtelulta, ja että sen valoherkkyys on asetettu sopivalle alueelle säätövastuksen avulla.

Työtä tehdessä on laitevalmistajien omien sivujen ja manuaalien lisäksi apuna ollut myös erilaisten netistä löytyvien harrastelijaprojektien tutkiminen. Myös Theseus.fi -sivustolta löytyvien opinnäytetöiden lukeminen on antanut ideoita ja ollut hyödyksi oman opinnäytetyön viimeistelyssä.

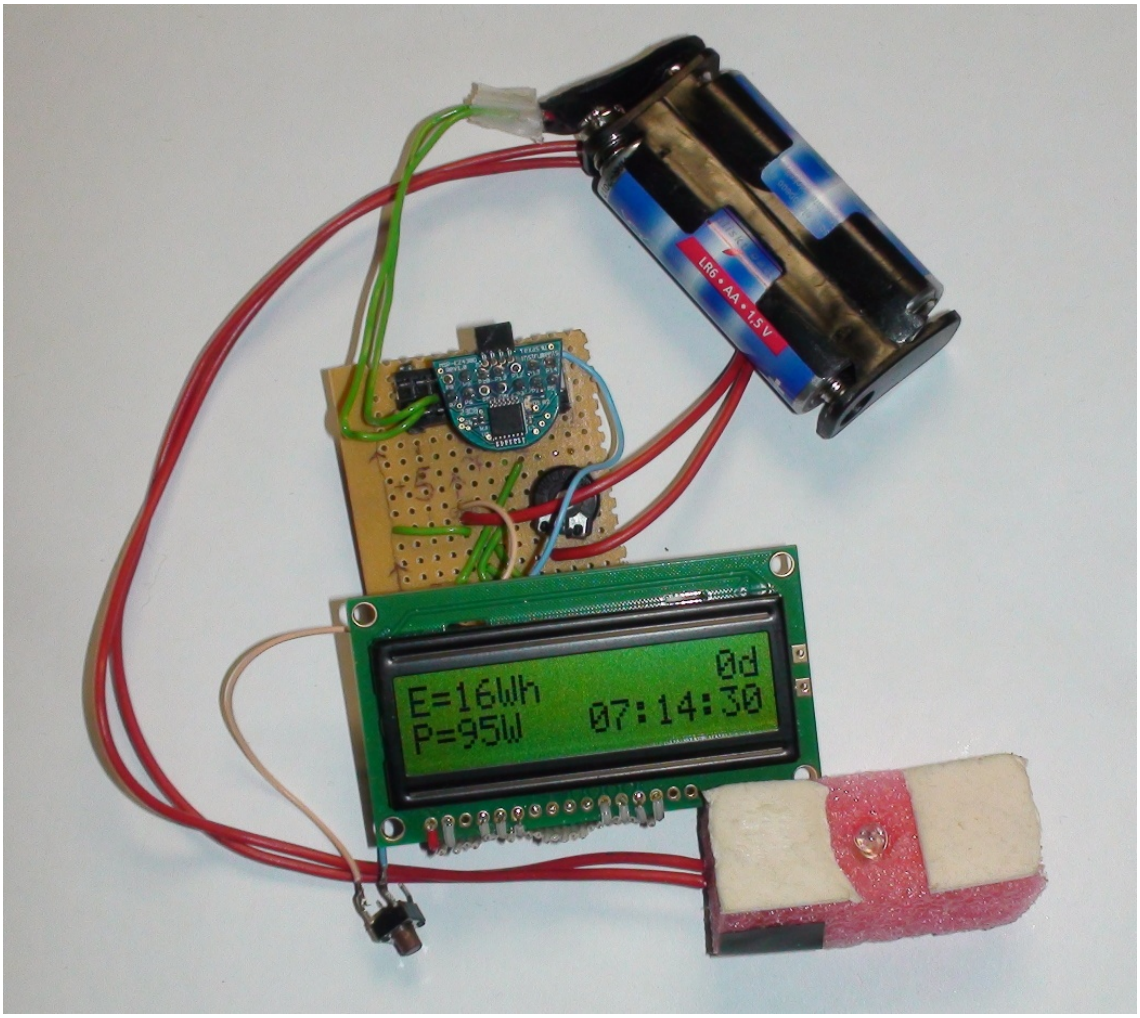
MSP430F2013 soveltuu erittäin hyvin yksinkertaisiin paristokäyttöisiin projekteihin sen monipuolisten ominaisuuksien, helppokäyttöisten kehitystyökalujen sekä alhaisen virrankulutuksen, käyttöjännitteen ja hinnan ansiosta. Rajoittavina tekijöinä voi kuitenkin olla sen vain 2 kilotavun kokoinen ohjelmamuisti tai I/O-pinnien lukumäärä, mutta suurempia projekteja varten löytyy MSP430-sarjasta monia muita vastaavanlaisia mikrokontrollereita isommalla Flash-muistilla tai useammalla I/O-pinnillä varustettuna.

**LÄHTEET**

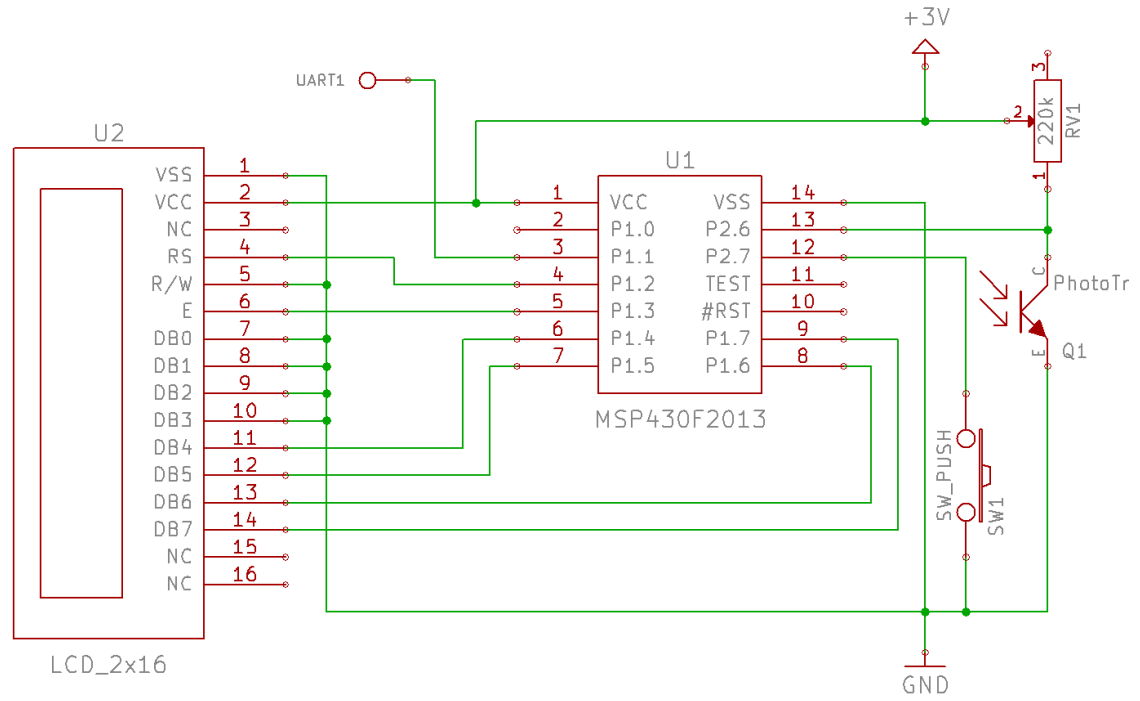

1. MSP430F2013 manuaalit. Luettu 2013. http://www.ti.com/product/msp430f2013
2. eZ430-F2013 kehitystyökalu. Luettu 2013. http://www.ti.com/tool/ez430-f2013
3. LCD-moduulin ohjauskoodi. Luettu 2012. http://cacheattack.blogspot.fi/2011/06/quick-overview-on-interfacing-msp430.html
4. UART esimerkkiohjelma. Luettu 2012. http://www.msp430launchpad.com/2010/08/half-duplex-software-uart-on-launchpad.html
5. Vishay BPV11 datasheet. Luettu 2013. http://www.vishay.com/docs/81504/bpv11.pdf
6. Amber AMB8420. Luettu 2012. http://amber-wireless.de/58-1-AMB8420.html
7. Fordata LCD datasheet. Luettu 2013. http://www.farnell.com/datasheets/653660.pdf
8. Fordata LCD kuva. Luettu 2013. http://fi.farnell.com/fordata/fdcc1602l-rnnybw-16le/disp-lcd-16x2-3v-reflect/dp/1847939?Ntt=FDCC1602L-RNNYBW-16LE

**LIITTEET**

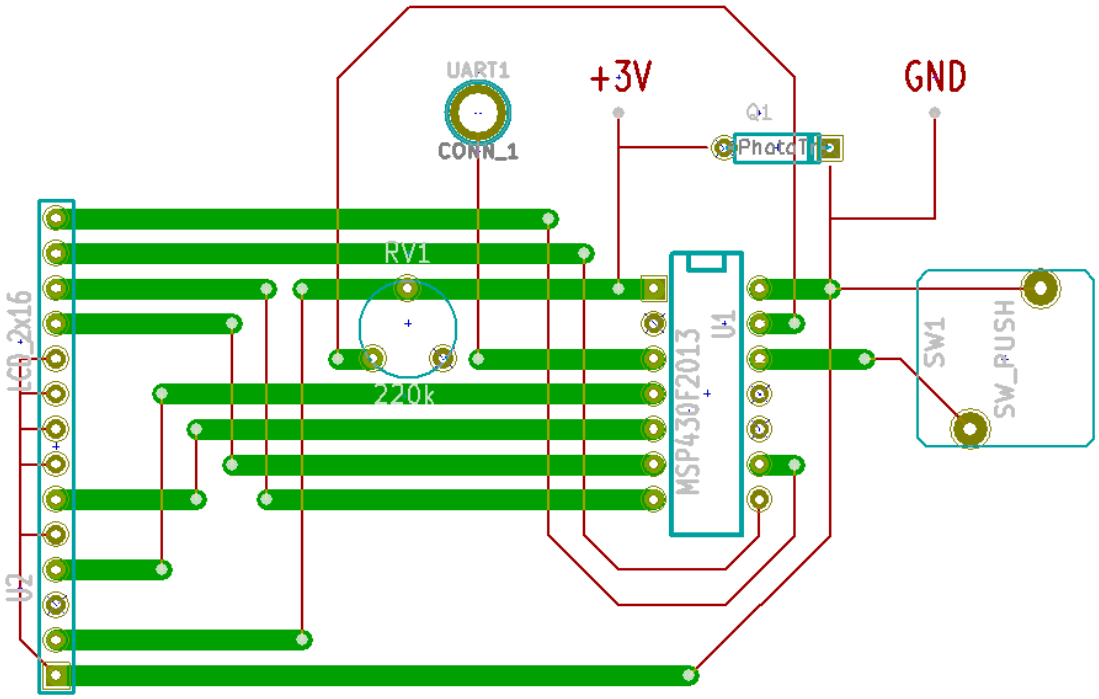
Liite 1. Kuva laitteesta



Liite 2. Kytkentäkaavio ja piirilevyn layout




Kytkentäkaavio




Reikäpiirilevyn layout, punaiset viivat ovat hyppylankoja.

(huom. MSP430:n kannan malli on tässä kuvassa mitoitettu DIP-kotelolle. Käytettäessä kehitystyökalun kohdekorttia piikkiriman kanssa, on kannan oltava kapeampi.)

Liite 3. Ohjelmakoodi

**main.cpp:** 1 (9)

```
// Code for: Electricity Meter Reader Device (Sähkömittarin Lukijalaite)
//
// By: Jukka-Tapio Järvenpää
// Tampereen Ammattikorkeakoulu
// 2012
//
// Compiled with IAR Embedded Workbench
// For MSP430F2013 microcontroller
//

#include "in430.h"                  // intrinsic functions
#include "msp430.h"                 // standard MSP430 definitions
#include "stdbool.h"                // bool data type
#include "lcd.cpp"                  // functions to control the LCD-module

#define TXD       BIT1        // TXD on P1.1
#define Bit_time  104         // 9600 Baud, SMCLK=1MHz (1MHz/9600)=104
#define RTC_freq  9615        // modify to adjust RTC speed, default 9615 (= 1s / 104us)

// Global variable definitions

unsigned long int count_wh = 0;           // Wh count
unsigned long int PhotoTime = 0;    // 104us ticks since last PhotoInterrupt, to calc Watts
unsigned long int count_phototime = 40000000; // counter for PhotoTime, gives 0 Watts at start

unsigned int count_sec = 0;         // Timer_A interrupt count, counts 1s = 9615 * 104us
unsigned int count_buttonhold = 0;  // Timer_A interrupt count, counts 0.5s = 4808 * 104us
unsigned int days = 0;              // days count

unsigned int TXByte;                // Value sent over UART when Transmit() is called
unsigned char BitCnt;               // Bit count, used when transmitting byte

unsigned char hours = 0;            // hours count
unsigned char minutes = 0;          // minutes count
unsigned char seconds = 0;          // seconds count, controlled by count_sec
unsigned char Mode = 0;             // 0=normal, 1=LCD off, 2=to 3, 3=set hours, 4=set minutes
unsigned char ButtonHoldTime = 0;   // Time the button has been down in 0.5sec units

bool SecondElapsed = FALSE;         // flag when count_sec in Timer_A reached 1 second
bool ButtonHoldCall = FALSE;        // flag when count_buttonhold in Timer_A reached 0.5s
bool PhotoInterrupt = FALSE;        // Phototransistor interrupt flag (from dark to light)
bool ButtonInterrupt = FALSE;       // Button interrupt flag (on release)

// Function Definitions

void PrintTime(void);               // print time on LCD
void PrintEnergy(void);             // print energy and power on LCD
char * itoa(unsigned long int i);   // convert unsigned long int(32bit) to ascii string
void Transmit(void);                // transmits character from TXByte


void main(void)
{
      WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
      BCSCTL1 = CALBC1_1MHZ;        // Set range
      DCOCTL = CALDCO_1MHZ;         // SMCLK = DCO = 1MHz

      P1SEL = TXD;                  // P1.1 connected to timer(UART)
      P1DIR = 0x0FF;                // P1 all outputs
        P2SEL = 0;                  // digital i/o
        P2DIR = 0;                  // P2 all inputs
        P2REN |= BIT7;              // P2.7 pull-up resistor enabled
      P2IES = BIT6;                 // Int edges: light On(P2.6=0) or button Released(P2.7=1)
      P2IFG &= ~(BIT6 + BIT7);      // P2.6 + P2.7 IFG cleared
      P2IE |= BIT6 + BIT7;          // P2.6 + P2.7 interrupt enabled
```

```
__bis_SR_register(GIE);       // interrupts enabled

InitializeLcm();        // initialize LCD-module
ClearLcmScreen();
TXByte = 65;                  // Transmit byte "A" to TXD (UART) and activate timer
Transmit();
PrintTime();                  // print 00:00:00 on LCD

while(1)
{

      if((SecondElapsed) && (Mode < 2)) { // update time when 1 second elapsed
            seconds++;        // increment seconds counter
            if(seconds == 60) {
              minutes++;            // minutes++
              seconds = 0;
              if(minutes == 60) {
                hours++;            // hours++
                minutes = 0;
                if(hours == 24) {
                  days++;           // days++
                  hours = 0;
                }
              }
            }
            if(Mode == 0) {         // print time only in normal mode
              PrintTime();
            }
            SecondElapsed = FALSE;  // clear flag
      }

      if(ButtonHoldCall) {          // called once every 0.5sec if button is held down
            ButtonHoldTime++; // increase ButtonHoldTime
            if(Mode == 3) {         // setup hours -mode, button down for >0.5s
              hours++;
              if(hours == 24) {
                hours = 0;
              }
              PrintTime();
            }
            else if(Mode == 4) {    // setup minutes -mode, button down for >0.5s
              minutes++;
              if(minutes == 60) {
                minutes = 0;
              }
              PrintTime();
            }
            else if(ButtonHoldTime == 4) {      // if button held down for 2 sec
              if(Mode == 0) {       // from normal mode
                count_wh = 0;       // reset Wh counter
                ClearLcmScreen();
                PrintEnergy();
                PrintTime();
              }
              else if(Mode == 1) {        // from lcd off mode
                Mode = 2;                 // to "wait for button release"-mode
                SendByte(0x0E, FALSE);    // LCD on, set cursor on, no blinking
                ClearLcmScreen();
                PrintEnergy();
                PrintTime();
              }
            }
            ButtonHoldCall = FALSE;       // clear flag
      }

      if(ButtonInterrupt) {
            __delay_cycles(100000);       // SW Delay to filter bouncing
            if (P2IN & BIT7) {            // Button released (not just bounce)

              // change display mode unless button was held down for long
              if((Mode == 0) && (ButtonHoldTime < 4)) {
                Mode = 1;                 // to silent mode
                SendByte(0x08, FALSE);    // LCD OFF
```

```
                }
                else if((Mode == 1) && (ButtonHoldTime < 4)) {
                  Mode = 0;                 // to normal mode
                  SendByte(0x0C, FALSE);    // LCD ON
                  PrintEnergy();
                  PrintTime();
                }
                else if(Mode == 2) {
                  Mode = 3;                 // to setup hours -mode
                  seconds=0;                // seconds to 00
                  PrintTime();
                }
                else if((Mode == 3) && (!ButtonHoldTime)) {
                  Mode = 4;                 // to setup minutes -mode
                }
                else if((Mode == 4) && (!ButtonHoldTime)) {
                  Mode = 0;                 // to normal mode
                  count_sec = 0;            // reset count_sec and flag
                  SecondElapsed = FALSE;
                  SendByte(0x0C, FALSE);    // cursor off
                }
              }
              // P1OUT ^= BIT0;       // Toggle P1.0 = LED for testing
              ButtonHoldTime = 0;           // reset hold timer
              count_buttonhold = 0;
              ButtonInterrupt = FALSE;      // clear interrupt flag
        }

        if(PhotoInterrupt) {
              count_wh++;             // increase Wh counter
              if(Mode != 1) {               // if not in silent mode then
                PrintEnergy();        // show Energy and Power on LCD
              }
              TXByte = 87;                  // transmit byte "W" to TXD (UART)
              Transmit();
              TXByte = count_wh & 0x0FF;    // transmit last 8 bits of Wh count
              Transmit();

              // SW Delay to limit sample rate below 10Hz = 36kW
              // to ignore too fast and unwanted light flickering
              __delay_cycles(100000);
              PhotoInterrupt = FALSE;       // Clear interrupt flag
        }

        if((Mode == 2) || (Mode == 3)) {    // setup hours -mode
              LcmSetCursorPosition(1,9);    // move cursor to location of hours
        }
        if(Mode == 4) {                     // setup minutes -mode
              LcmSetCursorPosition(1,12);   // move cursor to location of minutes
        }

        // if no other interrupts pending, then
        // disable CPU (LPM0) and continue when an interrupt happens

        if(!(SecondElapsed || ButtonHoldCall || ButtonInterrupt)) {
              __bis_SR_register(CPUOFF + GIE);
        }

  }
}


// print time on LCD
void PrintTime()
{
  if(days < 10)
        LcmSetCursorPosition(0,14);
  else if(days < 100)
        LcmSetCursorPosition(0,13);
  else
        LcmSetCursorPosition(0,12);
  PrintStr(itoa(days));               // print days
  PrintStr("d");
```

```
      LcmSetCursorPosition(1,8);
      if((hours)<10) {              // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(hours));              // hours

      PrintStr(":");
      if((minutes)<10) {                  // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(minutes));            // minutes

      PrintStr(":");
      if((seconds)<10) {                  // print zero if needed
            PrintStr("0");
      }
      PrintStr(itoa(seconds));            // seconds
}

// print Energy and Power on LCD
void PrintEnergy()
{
      LcmSetCursorPosition(0,0);
      PrintStr("E=");                     // print energy on first line
      PrintStr(itoa(count_wh));
      PrintStr("Wh");
      LcmSetCursorPosition(1,3);
      PrintStr("     ");                  // clear area before rewrite
      LcmSetCursorPosition(1,0);
      PrintStr("P=");                     // print power on second line

      // using the number of 104us ticks elapsed from previous PhotoInterrupt to calculate power
      // E = P * t
      // P = E / t
      // P = 1Wh / (X * 104us)
      // P = (1Wh / 104us) / X
      // P = (3600Ws / 0.000104s) / X
      // P = 34615385W / X
      PrintStr(itoa(34615385 / PhotoTime));
      PrintStr("W");
}

// convert unsigned long int(32bit) to ascii string
char * itoa(unsigned long int i)
{
      static char buf[11];
      char* bp = buf+sizeof(buf);
      *--bp = '\0';
      do
        *--bp = i%10+'0';
      while (i /= 10);
      return bp;
}

// transmits character from TXByte
void Transmit()
{
      CCTL0 = OUT;                        // TXD Idle as Mark
      TACTL = TASSEL_2 + MC_2;            // SMCLK, continuous mode

      BitCnt = 0xA;                       // Load Bit counter, 8 bits + ST/SP
      CCR0 = TAR;                   // Initialize compare register

      CCR0 += Bit_time;             // Set time till first bit
      TXByte |= 0x100;              // Add stop bit to TXByte (which is logical 1)
      TXByte = TXByte << 1;               // Add start bit (which is logical 0)

      CCTL0 =  CCIS0 + OUTMOD0 + CCIE;    // Set signal, intial value, enable interrupts
      while ( BitCnt != 0);               // Wait for previous TX completion
      // Because we're using Timer_A interrupt also for the RTC, we continue here immediately
      // after the stop bit is set, to allow next Timer_A int to be ready for possible start bit
}
```

```
// Port 2 interrupt service routine
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
      if(P2IFG & BIT6)             // Phototransistor interrupt (from dark to light)
      {
            PhotoTime = count_phototime; // save the time of interrupt
            count_phototime = 0;         // reset the time counter
            P2IFG &= ~BIT6;              // P2.6 IFG cleared
            PhotoInterrupt = TRUE;
      }
      if(P2IFG & BIT7)             // Button interrupt (on release)
      {
            P2IFG &= ~BIT7;              // P2.7 IFG cleared
            ButtonInterrupt = TRUE;
      }
      __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to continue main loop
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A(void)
{
      CCR0 += Bit_time;            // Add Offset to CCR0
      if ( BitCnt != 0)            // If there's bits to TX
      {
            CCTL0 |= OUTMOD2;      // Set TX bit to 0
            if (TXByte & 0x01)
                  CCTL0 &= ~ OUTMOD2;    // If it should be 1, set it to 1
            TXByte = TXByte >> 1;
            BitCnt--;
      }

      count_phototime++;                 // counts 104us ticks since last PhotoInterrupt
      count_sec++;                       // RTC implementation using 104us counter
      if (count_sec == RTC_freq) {       // 9615 * 104us = 0.999960s
            count_sec = 0;
            SecondElapsed = TRUE;
            __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to update time in main loop
      }

      if (!(P2IN & BIT7)) {              // if button down
            count_buttonhold++;
            if(count_buttonhold == 4808) {     // once in 0.5sec
                  count_buttonhold = 0;
                  ButtonHoldCall = TRUE; // activate ButtonHoldCall
                  __bic_SR_register_on_exit(CPUOFF); // exit from LPM0 to main loop
            }
      }
}
```

**lcd.cpp:** 6 (9)

```
//
// MSP430 LCD Code
//
// FROM: http://cacheattack.blogspot.fi/2011/06/quick-overview-on-interfacing-msp430.html
//

#define     LCM_DIR               P1DIR
#define     LCM_OUT               P1OUT

//
// Define symbolic LCM - MCU pin mappings
// We've set DATA PIN TO 4,5,6,7 for easy translation
//
#define     LCM_PIN_RS            BIT2          // P1.2
#define     LCM_PIN_EN            BIT3          // P1.3
#define     LCM_PIN_D7            BIT7          // P1.7
#define     LCM_PIN_D6            BIT6          // P1.6
#define     LCM_PIN_D5            BIT5          // P1.5
#define     LCM_PIN_D4            BIT4          // P1.4


#define        LCM_PIN_MASK   ((LCM_PIN_RS  |  LCM_PIN_EN  |  LCM_PIN_D7  |  LCM_PIN_D6  |  LCM_PIN_D5  |
LCM_PIN_D4))

#define     FALSE                 0
#define     TRUE                  1

//
// Routine Desc:
//
// This is the function that must be called
// whenever the LCM needs to be told to
// scan it's data bus.
//
// Parameters:
//
//     void.
//
// Return
//
//     void.
//
void PulseLcm()
{
    //
    // pull EN bit low
    //
    LCM_OUT &= ~LCM_PIN_EN;
    __delay_cycles(200);

    //
    // pull EN bit high
    //
    LCM_OUT |= LCM_PIN_EN;
    __delay_cycles(200);

    //
    // pull EN bit low again
    //
    LCM_OUT &= (~LCM_PIN_EN);
    __delay_cycles(200);
}



//
// Routine Desc:
//
// Send a byte on the data bus in the 4 bit mode
// This requires sending the data in two chunks.
```

```
// The high nibble first and then the low nible
//
// Parameters:
//
//    ByteToSend - the single byte to send
//
//    IsData - set to TRUE if the byte is character data
//                 FALSE if its a command
//
// Return
//
//     void.
//
void SendByte(char ByteToSend, int IsData)
{
    //
    // clear out all pins
    //
    LCM_OUT &= (~LCM_PIN_MASK);
    //
    // set High Nibble (HN) -
    // usefulness of the identity mapping
    // apparent here. We can set the
    // DB7 - DB4 just by setting P1.7 - P1.4
    // using a simple assignment
    //
    LCM_OUT |= (ByteToSend & 0xF0);

    if (IsData == TRUE)
    {
        LCM_OUT |= LCM_PIN_RS;
    }
    else
    {
        LCM_OUT &= ~LCM_PIN_RS;
    }

    //
    // we've set up the input voltages to the LCM.
    // Now tell it to read them.
    //
    PulseLcm();
     //
    // set Low Nibble (LN) -
    // usefulness of the identity mapping
    // apparent here. We can set the
    // DB7 - DB4 just by setting P1.7 - P1.4
    // using a simple assignment
    //
    LCM_OUT &= (~LCM_PIN_MASK);
    LCM_OUT |= ((ByteToSend & 0x0F) << 4);

    if (IsData == TRUE)
    {
        LCM_OUT |= LCM_PIN_RS;
    }
    else
    {
        LCM_OUT &= ~LCM_PIN_RS;
    }

    //
    // we've set up the input voltages to the LCM.
    // Now tell it to read them.
    //
    PulseLcm();
}


//
// Routine Desc:
//
// Set the position of the cursor on the screen
```

```
//
// Parameters:
//
//     Row - zero based row number
//
//     Col - zero based col number
//
// Return
//
//     void.
//
void LcmSetCursorPosition(char Row, char Col)
{
    char address;

    //
    // construct address from (Row, Col) pair
    //
    if (Row == 0)
    {
        address = 0;
    }
    else
    {
        address = 0x40;
    }

    address |= Col;

    SendByte(0x80 | address, FALSE);
}


//
// Routine Desc:
//
// Clear the screen data and return the
// cursor to home position
//
// Parameters:
//
//    void.
//
// Return
//
//     void.
//
void ClearLcmScreen()
{
    //
    // Clear display, return home
    //
    SendByte(0x01, FALSE);
    SendByte(0x02, FALSE);
}


//
// Routine Desc:
//
// Initialize the LCM after power-up.
//
// Note: This routine must not be called twice on the
//           LCM. This is not so uncommon when the power
//           for the MCU and LCM are separate.
//
// Parameters:
//
//    void.
//
// Return
//
//     void.
```

```
//
void InitializeLcm(void)
{
    //
    // set the MSP pin configurations
    // and bring them to low
    //
    LCM_DIR |= LCM_PIN_MASK;
    LCM_OUT &= ~(LCM_PIN_MASK);


    //
    // wait for the LCM to warm up and reach
    // active regions. Remember MSPs can power
    // up much faster than the LCM.
    //
    __delay_cycles(100000);


    //
    // initialize the LCM module
    //
    // 1. Set 4-bit input
    //
    LCM_OUT &= ~LCM_PIN_RS;
    LCM_OUT &= ~LCM_PIN_EN;

    LCM_OUT = 0x20;
    PulseLcm();

    //
    // set 4-bit input - second time.
    // (as reqd by the spec.)
    //
    SendByte(0x28, FALSE);

    //
    // 2. Display on, cursor off, blink cursor off
    //
    SendByte(0x0C, FALSE);

    //
    // 3. Cursor move auto-increment
    //
    SendByte(0x06, FALSE);
}


//
// Routine Desc
//
// Print a string of characters to the screen
//
// Parameters:
//
//    Text - null terminated string of chars
//
// Returns
//
//     void.
//
void PrintStr(char *Text)
{
    char *c;

    c = Text;

    while ((c != 0) && (*c != 0))
    {
        SendByte(*c, TRUE);
        c++;
    }
}
```