MSP430 JA KIIHTYVYYSANTURI

Niko Heikkinen

Opinnäytetyö
Joulukuu 2012
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

HEIKKINEN, NIKO:
MSP430 ja kiihtyvyysanturi

Opinnäytetyö 43 sivua, josta liitteitä 6 sivua
Joulukuu 2012

Tämän työn tarkoituksena on tutkia kiihtyvyysanturin lukemista Texas Instrumentsin MSP430G2553-prosessorilla ja tehdä ohjelma, jolla voidaan lähettää kiihtyvyysanturin lukemia tietokoneen USB-porttiin helpostiluettavassa muodossa.

Kiihtyvyysanturina käytetään Freescale Semiconductorin valmistamaa MMA2301-kiihtyvyysanturia. Työssä käydään läpi kiihtyvyysantureita yleisesti, esitellään MMA2301:n ominaisuuksia sekä TI:n MSP430 Launchpad -kehitysalustaa ja MSP430G2553-prosessorin tässä työssä tarvittavia ominaisuuksia. Varsinaista prototyyppiä ei ole tarkoitus saada valmiiksi, mutta sitä varten suunnitellaan kytkentä ja ohjelma. Ohjelman ja kiihtyvyysanturin toimintaa testataan ja esitellään testitulokset.

Tavoitteena on kokeilla kappaleen paksuuden mittausta kohdistamalla kappaleeseen isku tietyllä voimalla ja mittaamalla syntynyttä värähtelyä. Testausvaiheessa käytetään saatavilla olevia materiaaleja, mutta laitteen on tarkoitus mitata järven tai lammen jään paksuutta.

Asiasanat: msp430, kiihtyvyysanturi

ABSTRACT

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

HEIKKINEN, NIKO:
MSP430 and accelerometer

Bachelor's thesis 43 pages, appendices 6 pages
December 2012

The purpose of this thesis is to examine reading an accelerometer with the MSP430G2553-microprocessor made by Texas Instruments, and make a program which sends readings from the accelerometer to a PC through USB port.

The accelerometer used is the MMA2301 made by Freescale Semiconductor. General properties of accelerometers, properties of the MMA2301, and properties of the MSP430G2553, which are needed in this thesis, are described. Also the TI MSP430 Launchpad development board is introduced. An actual prototype will not be built, but circuit diagram and program will be designed for it. The functioning of the program and the accelerometer are tested, and test results are presented.

The goal is to measure thickness of an object by striking it with a certain force, and measure the vibration caused by the strike. In the testing phase, available materials are used, but the purpose of the device is to measure the thickness of ice on a lake.

Key words: msp430, accelerometer, acceleration sensor

SISÄLLYS

LYHENTEET JA TERMIT

DCO	Digitally Controlled Oscillator. Digitaalisesti ohjattu oskillaattori
DTC	Data Transfer Controller. Siirtää tietoa automaattisesti suoraan muistiin
GPIO	General Purpose Input/Output. Yleiskäyttöinen portti, joka voidaan ohjelmoida joko signaalin vastaanottajaksi tai lähettäjäksi
LPM	Low Power Mode. Virransäästötila
UART	Universal Asynchronous Receiver Transmitter. Muuntaa rinnakkaismuotoista tietoa sarjamuotoiseksi ja päinvastoin

1 JOHDANTO

Pienikokoisia ja edullisia jäänpaksuutta mittaavia laitteita ei juurikaan ole markkinoilla. Useimmiten mittalaitteet on kiinnitetty johonkin ajoneuvoon, kuten autoon sisälle tai moottorikelkalla vedettävän kelkan päälle. Nämä laitteet ovat myös hankintahinnaltaan tavalliselle luonnossa liikkujalle usein liian kalliita, eikä niitä ole mahdollista kuljettaa mukanaan esimerkiksi suksilla liikuttaessa. Tämän vuoksi helposti mukana kulkevalle jäänpaksuusmittarille voisi olla kysyntää, varsinkin jos hinta saadaan pysymään kohtuullisena.

Jäänpaksuusmittari pyritään toteuttamaan käyttämällä hyväksi mekaanista iskuria sekä kiihtyvyysanturia, jonka lähettämän signaalin perusteella voidaan laskea jään paksuus. Kun iskuri lyö jäähän, jää värähtelee iskun seurauksena. Kiihtyvyysanturilla voidaan mitata tätä värähtelyä ja sen perusteella pyritään määrittämään jään paksuus. Tämä työ keskittyy kiihtyvyysanturin lukemiseen ja tuloksen esittämiseen käyttäen Texas Instrumentsin MSP430-perheen mikrokontrolleria. Laitteen mekaaniseen toteutukseen ei siis paneuduta kovin tarkasti, koska se ei ole työn kannalta olennaista.

Valmista prototyyppiä ei tämän työn lopputuloksena synny, joten tavoitteena onkin saada mittauksen periaatteet toimimaan aluksi laboratorio-oloissa, jonka jälkeen voidaan tulevaisuudessa rakentaa ensimmäinen prototyyppi.

2 KIIHTYVYYSANTURIT

2.1 Toimintaperiaate

Kiihtyvyyden mittaus perustuu Newtonin II lakiin, eli dynamiikan peruslakiin:

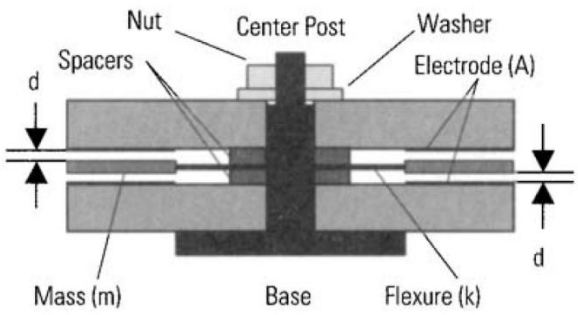
$$F = ma \tag{1}$$

Anturin aktiiviseen komponenttiin kohdistuva voima *F* on suoraan verrannollinen siihen kiinnitettyyn seismisen massan painoon *m* ja anturin kiihtyvyyteen *a*. Mitä suurempi kiihtyvyys *a* on, sitä suurempi voima *F* kohdistuu aktiiviseen komponenttiin ja sitä suurempi muutos näkyy anturin ulostulossa. Komponentit on kiihtyvyysantureissa yleensä kytketty siltaan, koska siltakytkennästä voidaan havaita pienetkin komponenttien arvojen muutokset.

Erilaiset anturityypit on jaettu käytetyn teknologian mukaan pietsosähköisiin, pietsoresistiivisiin ja kapasitiivisiin antureihin. Tässä keskitytään lähinnä kapasitiiviseen tekniikkaan, koska työssä käytetty kiihtyvyysanturi on kapasitiivinen.

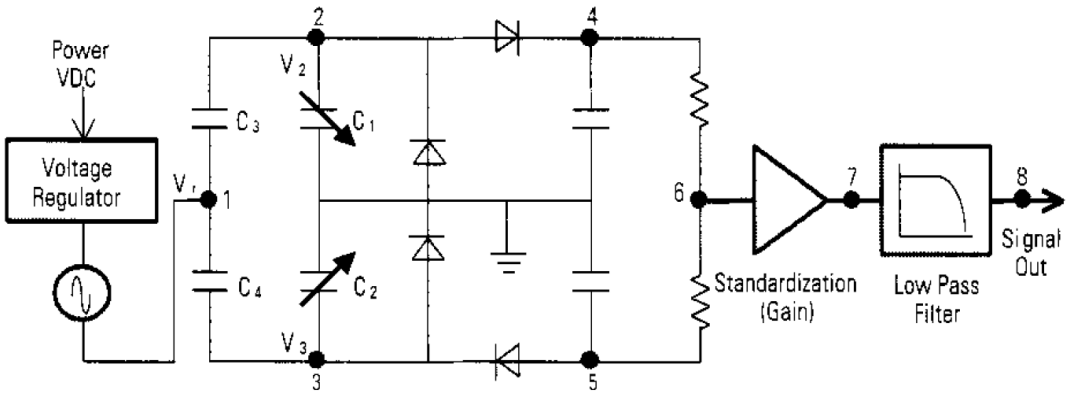
2.2 Kapasitiivinen kiihtyvyysanturi

Kapasitiivinen kiihtyvyysanturi havaitsee kiihtyvyyden kapasitanssin muutoksena. Kaksi elektrodia ja jousen päässä oleva massa muodostavat kaksi kondensaattoria. Nämä kondensaattorit on kytketty siltaan. Kun anturilla on kiihtyvyyttä, liikkuu jousen päässä oleva massa lähemmäksi toista elektrodia ja kondensaattorien kapasitanssit muuttuvat. Kapasitiivisen kiihtyvyysanturin rakenteen periaate on esitetty kuvassa 1. [1, s. 146]

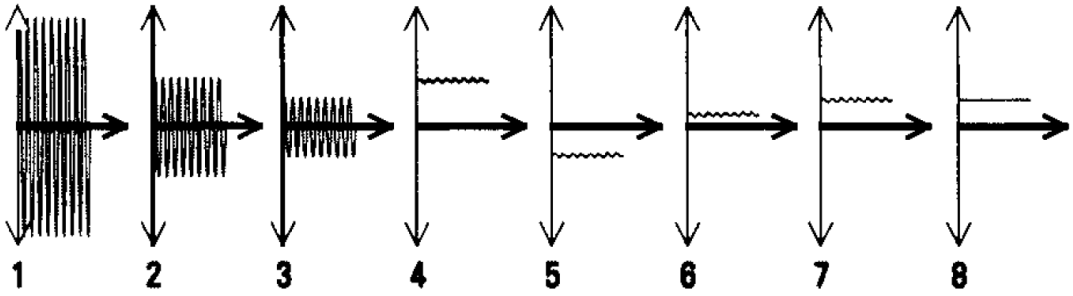


KUVA 1. Kapasitiivisen kiihtyvyysanturin rakenteen periaate [1, s. 146]

Kapasitiiviset kiihtyvyysanturit tarvitsevat toimiakseen oheiskomponentteja, jotta kapasitanssin muutoksilla voidaan mitata staattista sekä muuttuvaa kiihtyvyyttä. Oheiskomponenteilla luodaan myös jännitesignaali, jota voidaan mitata. Kuvassa 2 on esitetty kytkennän rakenne, ja kuvassa 3 signaalin muoto piirin eri kohdissa. [1, s.147]



KUVA 2. Kapasitiivisen kiihtyvyysanturin kytkentä [1, s. 147]

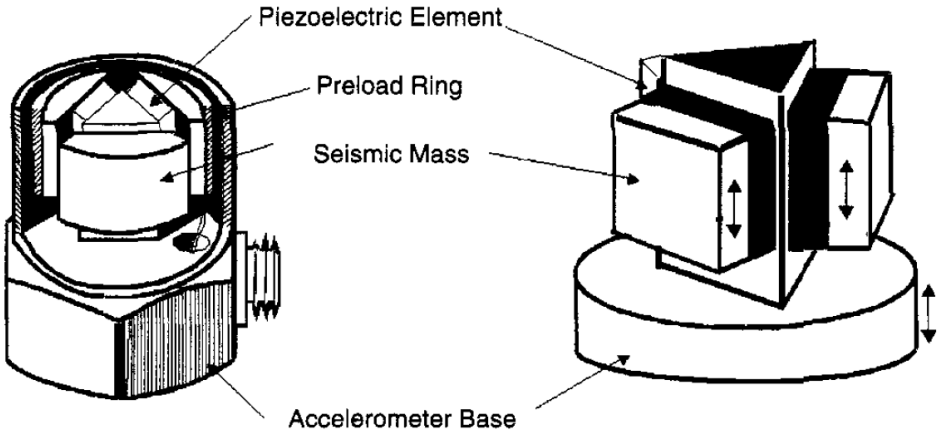


KUVA 3. Signaalin muoto kytkennän eri pisteissä, jotka on merkitty kuvaan 2 [1, s. 147]

Anturin sisäänrakennettu oskillaattori syöttää sillalle korkeataajuista signaalia pisteeseen 1. Signaali jakautuu pisteisiin 2 ja 3, ja kunkin jakautuneen signaalin amplitudi on suoraan verrannollinen kapasitanssin suuruuteen. Jakautuneiden signaalien huippuarvot pisteissä 4 ja 5 summataan pisteessä 6. Lopuksi signaalia vahvistetaan (piste 7) ja alipäästösuodin tekee siitä tasaisempaa (piste 8). [1, s. 147]

2.3 Pietsosähköinen ja pietsoresistiivinen kiihtyvyysanturi

Pietsosähköisen kiihtyvyysanturin aktiivinen komponentti on pietsosähköistä materiaalia, joka tuottaa sähköisen jännitteen, kun siihen kohdistuu mekaanista rasitusta. Pietsosähköisen kiihtyvyysanturin rakenteen periaate on esitetty kuvassa 4. [1, s. 138]

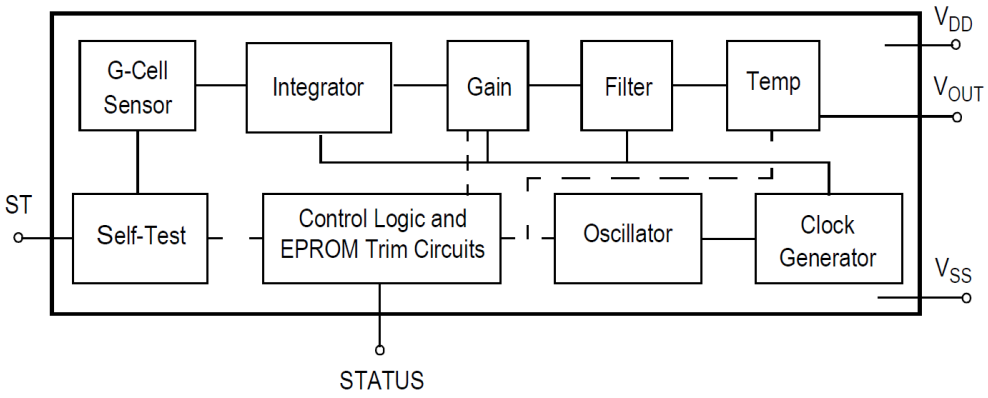


KUVA 4. Pietsosähköisen kiihtyvyysanturin rakenteen periaate [1, s. 138]

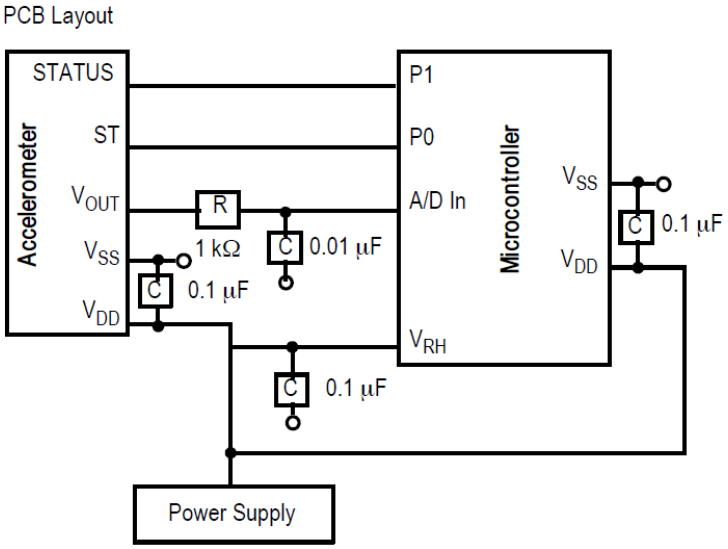
Pietsoresistiivisen anturin toimintaperiaate on samanlainen, mutta aktiivisen komponentin resistanssi muuttuu, kun siihen kohdistuu mekaanista rasitusta.

2.4 MMA2301EG-kiihtyvyysanturi

MMA2301EG on Freescale Semiconductorin valmistama SOIC-16 koteloitu kapasitiivinen kiihtyvyysanturi. Anturilla voidaan mitata ±200 g:n kiihtyvyyksiä yhden akselin suuntaisesti. Tarvittavien ulkoisten komponenttien määrä on vähäinen, koska piirille on sisäänrakennettu mittauksessa tarvittava kytkentä. Tämä tekee piirin liittämisestä mikrokontrollerin AD-muuntimen tuloon yksinkertaista. Tyypillisiä käyttökohteita piirille ovat tärinän ja iskujen seuranta, ja se toimii laajalla lämpötila-alueella: −40 - +125 °C. Kiihtyvyysanturin yksinkertaistettu lohkokaavio on esitetty kuvassa 5 ja kiihtyvyysanturin kytkeminen mikrokontrolleriin ja komponenttien sijoittelu on esitetty kuvassa 6. MMA2310EG:n nastojen selitykset on esitetty taulukossa 1.
[2, s. 1]



KUVA 5. MMA2301EG-kiihtyvyysanturin yksinkertaistettu toiminnallinen lohkokaavio [2, s. 1]



KUVA 6. MMA2301EG:n liittäminen mikrokontrolleriin ja komponenttien sijoittelu piirilevyllä [2, s. 5]

TAULUKKO 1. MMA2301EG:n nastojen selitykset [2, s. 5]

Nastan nro	Nimi	Kuvaus
1-3, 9-16	N/C	Ei kytketty, nastat 9-13 ovat tehtaalla tehtyä hienosäästöä varten
4	ST	Self-Test, looginen sisäänmeno, jolla voidaan aloittaa piirin toimivuuden testaus
5	V_{OUT}	Kiihtyvyysarvo
6	STATUS	Looginen ulostulo piirin sisäisen virheen havainnointia varten
7	V_{SS}	Maa
8	V_{DD}	Käyttöjännite

Self-Test

Piirin Self-Test -toiminnon avulla voidaan varmistaa piirin mekaaninen ja elektroninen toimivuus. Varsinkin sellaisissa sovelluksissa, joissa piirin toimimattomuus voi johtaa henkilövahinkoihin kuten autojen airbag-järjestelmissä, piirin tulee toimia aina. Kun Self-Test -nastaan syötetään ylhäällä oleva signaali, muodostuu anturissa testausta varten olevan elektrodin ja liikkuvan elektrodin välille elektrostaattinen voima, joka liikuttaa liikkuvaa elektrodia. Näin voidaan siis simuloida kiihtyvyyttä, joka näkyy piirin ulostulossa normaalisti. [2, s. 4]

Status

Anturissa on sisäänrakennettu häiriöntunnistus ja häiriösalpa. Status-nasta on häiriösalvan ulostulo, joka nousee ylös kun käyttöjännite laskee liian alas, kello-oskillaattorin taajuus laskee liian pieneksi tai EPROM-bittien pariteetti on pariton. Häiriösalvan voi nollata syöttämällä nousevan reunan Self-Test -nastaan, jos häiriön aiheuttanut tila on poistunut. [2, s. 4]

3 MSP430

3.1 MSP430 LaunchPad

Kehitysalustaksi valittiin Texas Instrumentsin MSP430 LaunchPad MSP-EXP430G2, jonka mukana tulee MSP430G2553-mikrokontrolleri. Mukana tulee myös MSP430G2452-mikrokontrolleri, jossa on vähemmän ominaisuuksia ja pienemmät flash- ja SRAM-muistit. MSP430G2553:n ominaisuuksista kerrotaan tarkemmin myöhemmin tässä luvussa.

MSP430 LaunchPad sisältää USB-liitännäisen ohjelmointi- ja debug-rajapinnan, joten piirin ohjelmointi ja testaus on helppoa. Ohjelmointiin käytettävä kehitysympäristö voi olla joko TI:n Code Composer Studio tai IAR Systemsin IAR Embedded Workbench. Molemmat ovat opiskelukäytössä ilmaisia, joskin molemmissa ohjelman koko on rajoitettu. Code Composer Studion rajoitus on 16 kB ja IAR Embedded workshopin rajoitus on 4 kB. Tämän työn tekemiseen valittiin Code Composer Studio, koska MSP430G2553-mikrokontrollerissa on 16 kB flash-muistia, joten muistin määrä ei ole rajoittavana tekijänä. Suurin osa muistista tullaan todennäköisesti käyttämään mittaustulosten tallentamiseen, joten 4 kB voi hyvinkin olla liian vähän. [3, s. 5]

Muita LaunchPad-kehitysalustan ominaisuuksia ovat:

- 9600 baudin UART sarjaliikenteiseen tiedonsiirtoon
- DIP-kanta, johon voidaan liittää kaikki LaunchPadin tukemat PDIP14- tai PDIP20-koteloidut MSP430G2xx- tai MSP430F20xx-mikrokontrollerit
- Kaksi lediä, vihreä ja punainen, jotka on kytketty GPIO-nastoihin ja voidaan tarvittaessa poistaa käytöstä jumppereilla
- Kaksi mikrokytkintä, joista toinen on kytketty reset-nastaan ja toinen P1.3-nastaan
- Kaksi 10-nastaista piikkirimaliitintä, jotka on kytketty kontrollerin nastoihin
- Liitin ulkoiselle jännitelähteelle
- Juotostäpät 32,768 kHz:n kiteelle, joka tulee kehitysalustan mukana

[3, s. 5]

3.2 MSP430G2553

MSP430G2553-mikrokontrollerin tärkeimmät ominaisuudet tämän työn kannalta ovat:

- Kolme mahdollista lähdettä kolmelle sisäiselle kellosignaalille, joista yksi on ulkonen lähde ja kaksi sisäistä lähdettä
- Pieni virrankulutus, aktiivisessa tilassa alle 500 µA 1 MHz:n kellotaajuudella ja 3,3 V:n käyttöjännitteellä, virransäästötilassa alle 1 µA
- 8-kanavainen 10-bittinen AD-muunnin 200 kHz:n näytteenottotaajuudella
- Kaksi 16-bittistä Timer_A-ajastinta kolmella Capture/Compare-rekisterillä

[5, s. 1]

3.2.1 Basic Clock Module+ -kellomoduuli

Basic Clock Module+ -kellomoduuli mahdollistaa pienen virrankulutuksen myös ilman ulkoisia kellolähteitä tai komponentteja. Kolme sisäistä kellosignaalia on mahdollista valita tilanteen mukaan niin, että tarvittaessa suorituskykyä on saatavilla matalasta virrankulutuksesta huolimatta. [4, s. 278]

Eri kellolähteitä ovat:

DCOCLK (DCO, Digitally Controlled Oscillator):
DCO on sisäinen, ilman ulkoista kidettä toimiva, digitaalisesti ohjattu oskillaattori. DCO:n taajuus voidaan määritellä DCOCTL-rekisterissä olevilla DCOx ja MODx -biteillä ja BCSCTL1-rekisterissä olevilla RSELx-biteillä. Piirikohtaiset esiasetetut taajuudet löytyvät piirin datalehdestä. [4, s. 283]

VLOCLK (VLO, Very-Low-Power Low-Frequency Oscillator):
VLO on sisäinen vähävirtainen matalalla taajuudella toimiva oskillaattori. Tyypillisesti tämä taajuus on 12 KHz, eikä oskillaattori tarvitse ulkoista kidettä. Oskillaattori otetaan käyttöön antamalla BCSCTL3-rekisterissä oleville LFCT1Sx-biteille arvo 10 ja BCSCTL1-rekisterissä olevalle XTS-bitille arvo 0. Tämä poistaa samalla LFXT1-oskillaattorin käytöstä. [4, s. 281]

LFXT1CLK:
LFXT1 on oskillaattori, jota käytetään ulkoisen kiteen kanssa, joko 32,768 kHz:n kellokiteen kanssa tai joissain piireissä korkeammalla taajuudella 400 kHz:stä 16 MHz:iin asti. MSP430G2553 ei tue LFXT1-oskillaattorin korkeataajuustilaa (XTS = 1). Käytettäessä 32,768 kHz:n kidettä ulkoisia komponentteja ei tarvita, vaan voidaan valita sisäänrakennetuista kondensaattoreista sopivat BCSCTL3-rekisterissä olevilla XCAPx-biteillä. Kapasitanssi voi olla 1 pF, 6 pF, 10 pF tai 12,5 pF. [4, s. 281]

Kaikkien sisäisten kellosignaalien lähteeksi voidaan valita lähteeksi mikä vain kolmesta kellolähteestä sillä rajoituksella, että VLO ja LFXT1 eivät voi olla käytössä samaan aikaan. Kaikille kelloille voidaan asettaa toisistaan riippumaton esijakaja. Mahdollisia esijakajan arvoja ovat 1, 2, 4, ja 8. [4, s. 279]

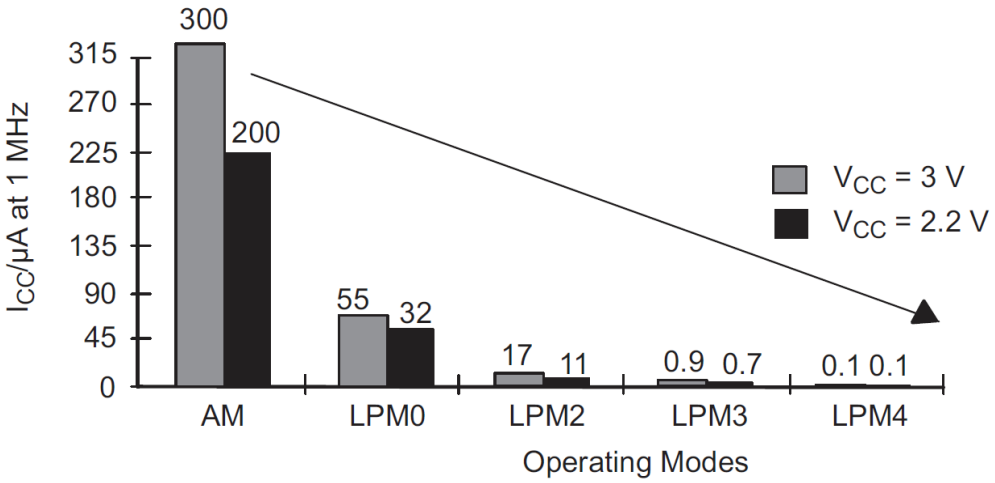
Sisäisiä kellosignaaleja ovat MCLK (Master Clock), ACLK (Auxiliary Clock) ja SMCLK (Sub-main Clock). MCLK on CPU:n käyttämä kello ja myös muita moduuleita voidaan käyttää tällä kellolla. Eri moduulien, esimerkiksi AD-muuntimen ja ajastimien, kelloksi voidaan määritellä joko ACLK tai SMCLK. Vaikka ACLK ja SMCLK saattavatkin saada kellosignaalin samasta lähteestä, voidaan esijakajien arvoilla vaikuttaa todelliseen kellosignaaliin. [4, s. 278]

3.2.2 Virransäästöominaisuudet

MSP430-perheen mikrokontrollerit on suunniteltu vähävirtaisiksi, ja tämä saavutetaan sammuttamalla CPU, käytössä olevat moduulit ja oskillaattorit, kun niitä ei tarvita. Kaikkea ei tarvitse sammuttaa kerralla, vaan sammutettavat osat voi määritellä tilarekisterin (SR, Status Register) CPUOFF-, OSCOFF-, SCG0- ja SCG1-biteillä sekä kunkin moduulin ohjausrekisteristä. MSP430G2553-mikrokontrollerin viisi eritasoista toimintatilaa (LPM, Low Power Mode) on esitetty taulukossa 2 ja virrankulutus eri toimintatiloissa on esitetty kuvassa 7. [4, s. 41]

TAULUKKO 2. MSP430G2553-mikrokontrollereiden toimintatilat [4, s. 42]

Tila	SCG1	SCG0	OSCOFF	CPUOFF	CPU:n ja kellojen tilat
Aktiivinen	0	0	0	0	CPU ja käytössä olevat kellot ovat aktiivisia.
LPM0	0	0	0	1	CPU ja MCLK on sammutettu, SMCLK ja ACLK ovat aktiivisia.
LPM1	0	1	0	1	CPU ja MCLK on sammutettu, DCO ja sen DC-generaattori on sammutettu, jos DCO ei ole SMCLK:n lähteenä. ACLK on aktiivinen.
LPM2	1	0	0	1	CPU, MCLK, SMCLK ja DCO on sammutettu. DC-generaattori on päällä ja ACLK on aktiivinen.
LPM3	1	1	0	1	CPU, MCLK, SMCLK, DCO ja DC-generaattori on sammutettu. ACLK on aktiivinen.
LPM4	1	1	1	1	CPU ja kaikki kellot on sammutettu.



KUVA 7. Virrankulutus eri toimintatiloissa [4, s. 41]

Kuvasta 7 nähdään, että jo pelkästään CPU:n ja MCLK:n sammuttaminen laskee virrankulutusta paljon. Aktiivisen tilan virrankulutukseen vaikuttavat DCO:n taajuus sekä piirin käyttöjännite. Siksi onkin virrankulutuksen kannalta parempi käyttää SMCLK:n ja ACLK:n lähteenä pienemmällä taajuudella toimivia VLO:ta tai LFXT1:tä.

Virransäästötiloista päästään nopeasti aktiiviseen tilaan keskeytysten avulla. Keskeytyksen tullessa ohjelmalaskuri (PC, Program Counter) ja tilarekisteri (SR) tallennetaan pinoon, jonka jälkeen CPUOFF-, SCG1- ja OSCOFF-bitit nollautuvat. Keskeytysohjelman lopussa tilarekisterin arvot palautetaan pinosta ja piiri menee keskeytystä edeltävään toimintatilaan, mutta on myös mahdollista muuttaa tilarekisterin pinossa olevia arvoja, jos halutaan keskeytyksen jälkeen asettaa piiri toiseen toimintatilaan. [4, s. 43]

3.2.3 AD-muunnin

AD-muunnin muuntaa analogisen signaalin vastaavaksi 10-bittiseksi digitaaliseksi signaaliksi. Muunnin käyttää kahta vertailutasoa (V_{R+} ja V_{R-}), joiden perusteella määritellään muunnoksen jännitealue. Digitaalinen ulostulo (N_{ADC}) saa suurimman arvon (03FFh), kun muunnettavan signaalin amplitudi on yhtä suuri tai suurempi kuin V_{R+} ja vastaavasti ulostulo on nolla (0000h), kun amplitudi on yhtä suuri tai pienempi kuin V_{R-}. Käännöstulos voidaan esittää suoraan binäärimuodossa tai 2:n komplementtina. [4, s. 550]

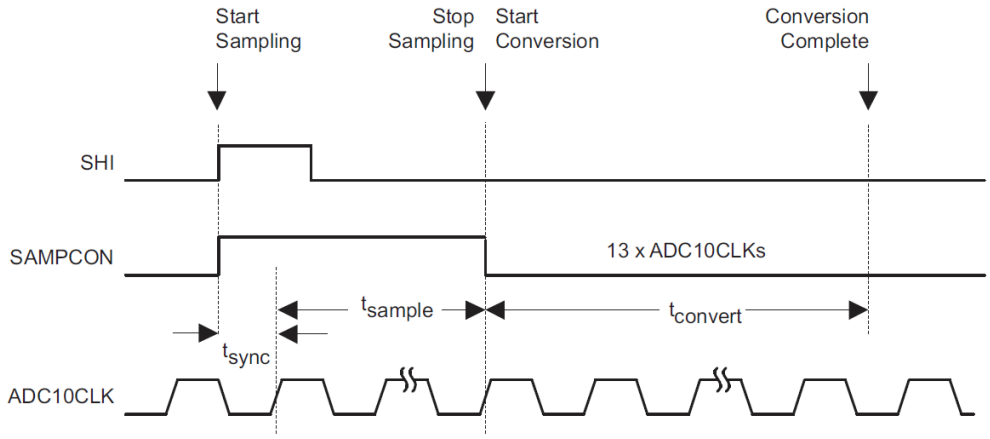
AD-muuntimen kellon ADC10CLK lähteenä voi olla MCLK, SMCLK, ACLK tai muuntimen sisäinen oskillaattori ADC10OSC. Käyttäjän täytyy huomioida, ettei kellolähdettä sammuteta ennen kuin muunnos on valmis. Sisäisen oskillaattorin taajuus on tyypillisesti 5 MHz, mutta se riippuu käyttöjännitteen ja lämpötilan suuruuksista ja vaihteluista. Myös eri piirien välillä on eroja, ja esimerkiksi MSP430G2553:n datalehti ilmoittaa taajuuden minimi- ja maksimiarvoiksi 3,7 MHz ja 6,3 MHz. [4, s. 550]

AD-muuntimella on kahdeksan ulkoista ja neljä sisäistä kanavaa eli sisääntuloa. Kanavan valinta ja vaihto tapahtuu analogisella multiplekserillä. Multiplekseri vähentää kanavan vaihdosta johtuvaa sisääntulojen välistä kohinaa sekä kanavien yhteenkytkeytymistä. Käyttämättömät kanavat kytketään maahan (V_{SS}), jotta hajakapasitanssi ei aiheuttaisi kanavien välistä ylikuulumista. Kanavien vaihdosta voi myös syntyä jännitepiikkejä, joten muunnos aloitetaan vasta signaalin asetuttua. [4, s. 550]

Vertailutasoiksi voidaan valita ulkoinen jännite tai muuntimen sisäisen generaattorin tuottama 1,5 V:n tai 2,5 V:n vertailutaso. Sisäiset vertailutasot on myös mahdollista tuoda ulos V_{REF+}- ja V_{REF-}-nastoihin, jotka vastaavat AD-muuntimen kanavia A3 ja A4. Samojen nastojen kautta voidaan myös syöttää ulkoiset vertailutasot V_{EREF+} ja V_{EREF-}. Ulkoinen positiivinen vertailutaso V_{EREF+} on myös mahdollista puskuroida pienentämään virrankulutusta. [4, s. 551]

AD-muunnin on muiltakin osin suunniteltu virtaa säästäväksi. Kun muunnos ei ole käynnissä, sammutetaan sisäinen generaattori, oskillaattori ja ydin, eivätkä ne tuolloin kuluta virtaa. Generaattori ja oskillaattori voidaan myös sammuttaa erikseen, jos niitä ei tarvita muunnoksen yhteydessä. [4, s. 551]

AD-muunnos alkaa SHI-signaalin nousevalla reunalla. SHI-signaalin lähteeksi voidaan valita ADC10OSC-bitti tai jokin TIMER_A:n kolmesta ulostulosta. Näytteenottojakson t_{sample} pituudeksi voidaan määrittää 4, 8, 16 ja 64 ADC10CLK:n kellojaksoa. Tämä määrittää SAMPCON-signaalin ylhäälläoloajan seuraavasta ADC10CLK:n nousevasta reunasta lähtien. SAMPCON-signaalin laskevasta reunasta alkaa varsinainen muunnos, joka kestää 13 ADC10CLK:n kellojaksoa. Koko muunnoksen kesto saadaan siis, kun lasketaan yhteen t_{sync}, t_{sample} ja $t_{convert}$. AD-muunnoksen ajoitus on esitetty kuvassa 8. [4, s. 552]



KUVA 8. AD-muuntimen muunnoksen ajoitus [4, s. 552]

Kun SAMPCON on alhaalla, kaikki sisääntulot ovat korkeaimpedanssisessa tilassa. Kun SAMPCON on ylhäällä, valittuna olevaan sisääntuloon liitetään sisäisesti RC-alipäästösuodin näytteenoton ajaksi. MSP430G2553-mikrokontrollerissa RC-piirin vastuksen R_I arvo on noin 1kΩ ja kondensaattorin C_I arvo on maksimissaan 27 pF. Tämä täytyy ottaa huomioon näytteenottoajan t_{sample} määrittelyssä, koska kondensaattorin C_I täytyy latautua tarpeeksi, jotta käännös on luotettava. Tämä voidaan laskea 10-bittiselle muunnokselle kaavan 2 mukaan: [4, s. 552]

$$t_{sample} > (R_S + R_I) \cdot \ln(2^{11}) \cdot C_I \qquad (2)$$

jossa R_S = signaalilähteen lähtöresistanssi (MMA2301EG lähtöimpedanssi Z_O = 300 Ω)

R_I = sisäinen vastus (1 kΩ)

C_I = sisäinen kondensaattori (27 pF)

Näytteenoton lyhimmäiskestoksi saadaan siis

$$\begin{aligned} t_{sample} &> (R_S + R_I) \cdot \ln(2^{11}) \cdot C_I \\ &= (0{,}3\ \text{k}\Omega + 1\ \text{k}\Omega) \cdot 7{,}624 \ldots \cdot 27\ \text{pF} \\ &= 0{,}267 \ldots \cdot 10^{-6}\ \text{s} \approx 0{,}27\ \mu\text{s} \end{aligned}$$

Näytteenottoaika t_{sample} täytyy olla siis vähintään 0,27 µs.

AD-muuntimella on neljä erilaista toimintatilaa, joilla voidaan valita mitä kanavia käytetään ja kuinka useasti muunnos tehdään. Toimintatila valitaan CONSEQx-biteillä, jotka ovat ADC10CTL1-rekisterissä. Eri toimintatilat on esitetty taulukossa 3. [4, s. 553]

TAULUKKO 3. AD-muuntimen eri toimintatilat [4, s. 553]

CONSEQx	Toimintatila
00	Yksi kanava muunnetaan kerran.
01	Monta kanavaa muunnetaan sarjana.
10	Jatkuva yhden kanavan muunnos.
11	Jatkuva monen kanavan sarjan muunnos.

Ensimmäisen toimintatilan ollessa aktiivinen, INCHx-biteillä valittu kanava muunnetaan kerran ja tulos tallennetaan ADC10MEM-rekisteriin. Tämä on toimintatiloista yksinkertaisin. [4, s. 554]

Kun AD-muunnin on tilassa, jossa muunnetaan monen kanavan sarja, valitaan ensimmäinen muunnettava kanava INCHx-biteillä. Kun ensimmäinen kanava on muunnettu, siirrytään siitä järjestysnumeroltaan edelliseen kanavaan. Näin jatketaan kunnes kanava A0 on muunnettu, jonka jälkeen muunnos loppuu. Jokaisen muunnoksen tulos tallennetaan muunnoksen jälkeen ADC10MEM-rekisteriin ennen kanavan vaihtoa. [4, s. 555]

Muunnoksen voi myös määrittää jatkuvaksi, jolloin yhden kanavan tai kanavien sarjan muunnoksen jälkeen aloitetaan uusi muunnos tai muunnosten sarja ilman, että muunnosta tarvitsee liipaista uudelleen. Tällöin käytetään MSC-bittiä, jolloin ensimmäinen SHI-signaalin nouseva reuna riittää liipaisemaan ensimmäisen muunnoksen ja seuraava muunnos alkaa heti edellisen loputtua. SHI-signaalin nousevia reunoja ei huomioida ennen kuin muunnosten sarja on valmis, tai kun ENC-bitti nollataan jatkuvan muunnoksen ollessa käynnissä. [4, s. 558]

Nollaamalla ENC-bitti voidaan pysäyttää käynnissä oleva muunnos, mutta eri toimintatilojen välillä on pieniä eroja. Kanavien sarjan, jatkuvan kanavien sarjan tai jatkuvan yhden kanavan muunnoksen ENC-bitin nollaus lopettaa seuraavan muunnoksen tai sarjan muunnoksen lopussa, ja muunnostulokset ovat luotettavia. Jos kyseessä on yhden kanavan ei-jatkuva muunnos, täytyy odottaa ADC10BUSY-bitin nollautumista, ettei käännöstä pysäytetä kesken ja ettei muunnos olisi epäluotettava. Kaikki muunnokset voidaan lopettaa kesken suorituksen nollaamalla ENC-bitti ja asettamalla CONSEQx-biteiksi 00, mutta tällöin muunnostulos on epäluotettava. [4, s. 558]

AD-muuntimen DTC (Data Transfer Controller) siirtää käännöstulokset automaattisesti ADC10MEM-rekisteristä muualle piirin sisäiseen muistiin, kun rekisteriin ladataan uusi käännöstulos. DTC otetaan käyttöön määrittämällä ADC10DTC1-rekisterin arvoksi jokin muu kuin nolla. ADC10DTC1-rekisteri määrittää yhteen lohkoon mahtuvien DTC:n tekemien siirtojen määrän. Yksi siirto vie kaksi muistipaikkaa, ja DTC voi käyttää yhtä tai kahta samankokoista lohkoa. Ensimmäisen lohkon aloittavan muistipaikan osoite määritetään ADC10SA-rekisterissä. DTC:n sisäinen laskuri ja osoitin huolehtivat itsenäisesti oikean muistipaikan valinnasta ja siitä, ettei lohkon ulkopuoliseen muistipaikkaan kirjoiteta. Jos käytössä on yksi lohko, DTC lopettaa kirjoittamisen lohkoon kun laskuri on nolla. Jos käytössä on kaksi lohkoa, DTC siirtyy ensimmäisen lohkon lopusta toisen lohkon alkuun ja lopettaa kirjoittamisen toisen lohkon lopussa. On myös mahdollista käyttää jatkuvaa siirtoa, jolloin DTC jatkaa kirjoittamista niin kauan, kun ADC10CT-bitti on ylhäällä. Kun se nollataan, DTC lopettaa kirjoittamisen viimeisen lohkon lopussa, eikä siirto katkea kesken lohkon. [4, s. 558−563]

3.2.4 Komparaattori

Comparator_A+ on analogisia jännitteitä vertaileva komparaattori. Komparaattori vertailee + ja − -tulojen jännitteitä keskenään, ja kun + -tulon jännite on suurempi kuin – -tulon jännite, komparaattorin ulostulo CAOUT asettuu. Komparaattori kytketään päälle ja pois CAON-bitillä, ja CAOUT on aina alhaalla kun komparaattori on sammutettu. Komparaattorin tulot kytketään P2CAx-biteillä. Molemmat tulot voidaan määritellä erikseen, ja niihin voidaan kytkeä joko ulkoiset tai sisäiset signaalit. [4, s. 539]

Sisäinen vertailujännite V_{CAREF+} tuotetaan komparaattorin vertailujännitegeneraattorilla, ja se voidaan kytkeä kumpaan komparaattorin tuloon tahansa. Vertailujännitegeneraattorin tulo määritellään CAREFx-biteillä ja CARSEL-biteillä valitaan, kumpaan tuloon vertailujännite kytketään. [4, s. 541]

Comparator_A+-keskeytys mahdollistaa tässä sovelluksessa AD-muunnoksen nopean liipaisun, joka ei ohjelmallisesti toteutettuna olisi mahdollista. CAIFG-keskeytyslippu asettuu joko CAOUT-lähdön nousevalla tai laskevalla reunalla, riippuen CAIES-bitistä. Kun CAIE- ja GIE -bitit on asetettu, CAIFG-keskeytyslippu aiheuttaa keskeytyspyynnön. [4, s. 542]

3.2.5 Ajastin

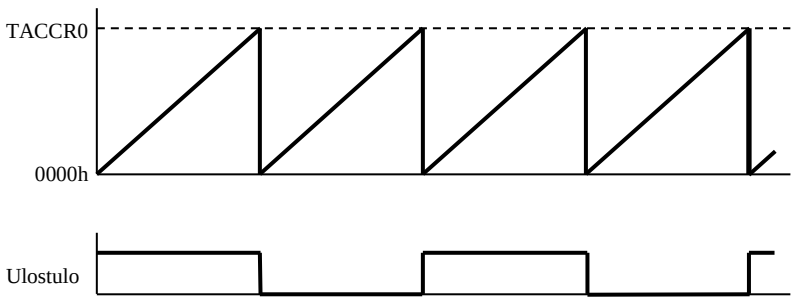
Timer_A on 16-bittinen ajastin kolmella capture/compare-rekisterillä. Timer_A:n kellolähteeksi voidaan valita yksi joko sisäisistä kellosignaaleista tai ulkoisista kellosignaaleista. Sisäisiä kellosignaaleja ovat ACLK ja SMCLK ja ulkoisia kellosignaaleja ovat TACLK tai INCLK. Kellolähde valitaan TASSELx-biteillä. Laskuri inkrementoi tai dekrementoi rekistereitään aina kellosignaalin nousevalla reunalla. Timer_A-ajastimella on neljä toimintatilaa: pysäytetty, ylöslaskeva, jatkuva ja ylös-alaslaskeva. Toimintatila valitaan MCx-biteillä, ja ajastin voidaan pysäyttää milloin vain asettamalla MCx-bittien arvoksi 00. [4, s. 364–366]

Ylöslaskevassa toimintatilassa laskuri laskee nollasta ylöspäin kunnes se saavuttaa jonkin TACCRx-rekistereissä olevan arvon. Tämä tuottaa keskeytyspyynnön ja muuttaa vastaavan ulostulon arvoa ulostulon toimintatilan mukaan. Ulostuloilla on eri toimintatiloja kahdeksan, jotka on esitelty taulukossa 4. [4, s. 366]

TAULUKKO 4. Timer_A-ajastimen ulostulojen toimintatilat [4, s. 372]

OUTMODx	Toimintatila	Kuvaus
000	Output	Ulostulo määritetään OUTx-bitillä ja se päivittyy välittömästi
001	Set	Ulostulo asetetaan kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo pysyy asetettuna kunnes ajastin nollataan tai kun valitaan toinen toimintatila, joka vaikuttaa ulostuloon.
010	Toggle/Reset	Ulostulo vaihtaa tilaa kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo nollataan kun ajastin laskee TACCR0:n arvoon saakka.
011	Set/Reset	Ulostulo asetetaan kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo nollataan kun ajastin laskee TACCR0:n arvoon saakka.
100	Toggle	Ulostulo vaihtaa tilaa kun ajastin laskee TACCRx:n arvoon saakka. Ulostulon jakso on kaksi kertaa pidempi kuin ajastimen jakso.
101	Reset	Ulostulo nollataan kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo pysyy nollattuna kunnes valitaan toinen toimintatila, joka vaikuttaa ulostuloon.
110	Toggle/Set	Ulostulo vaihtaa tilaa kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo asetetaan kun ajastin laskee TACCR0:n arvoon saakka.
111	Reset/Set	Ulostulo nollataan kun ajastin laskee TACCRx:n arvoon saakka. Ulostulo asetetaan kun ajastin laskee TACCR0:n arvoon saakka.

AD-muuntimelle luodaan tahdistussignaali käyttäen Timer_A-ajastinta ylöslaskevassa tilassa ja valitsemalla ulostulon toimintatilaksi tilaa vaihtava eli toggle. Tällöin pitää kuitenkin huomioida, että tahdistussignaalin jaksonaika on kaksinkertainen TACCRx-rekisterin arvoon verrattuna. Timer_A:n ulostulo tässä tilassa on esitetty kuvassa 8.



KUVA 8. Timer_A:n ulostulo ylöslaskevassa tilassa ja ulostulon ollessa tilaa vaihtava

4 PROTOTYYPIN SUUNNITTELU

4.1 Code Composer Studio 5

Code Composer Studio 5 on kehitysympäristö TI:n sulautetuille prosessoreille. Ohjelma sisältää kääntäjän, koodieditorin sekä debuggerin. Muistien ja rekistereiden lukeminen debuggauksen aikana on helppoa, ja tämä on todella hyvä ominaisuus ohjelman ja prosessorin toimintaa tarkasteltaessa. CCStudio on TI:n itse räätälöimä versio Eclipsestä, joten siksi se on looginen valinta kehitettäessä ohjelmia TI:n prosessoreille, varsinkin jos kyseessä on edullinen Value Line -sarjan prosessori, joille ilmaisen version 16 kB:n ohjelman kokorajoituksella ei käytännössä ole vaikutusta.

4.2 Ohjelma

Ohjelman lähdekoodi on esitetty kokonaisuudessaan liitteessä 2.

4.2.1 Muuttujien ja taulukoiden alustus

Ohjelmassa käytettävät muuttujat ja taulukot alustetaan globaaleiksi, eli ennen pääohjelmasilmukkaa.

ARRAY_SIZE määritetään taulukon *samples[]* kooksi ja DTC:n näytteiden määräksi.

```
#define ARRAY_SIZE 200
```

Kokonaislukua *i* käytetään pääohjelman for-silmukan kierrosten määrää varten. Char-tyyppiset muuttujat *start_transmit* ja *dtc_finished* ovat lippuja, joiden tilaa tarkastellaan pääsilmukassa. Lippujen asetus tapahtuu ADC10- ja USCIAB0RX-keskeytysaliohjelmissa.

```
static volatile unsigned int    i = 0,

static volatile unsigned char   start_transmit = 0,
                                dtc_finished = 0,
```

Taulukoita *digits[]* ja *decade[]* käytetään desimaaliluvun muunnossa ASCII-merkeiksi. Taulukko *digits[]* sisältää ASCII-merkit luvuille 0-9, ja alkion indeksi vastaa kutakin lukua. Taulukkoa *decade[]* käytetään kolminumeroisen desimaaliluvun kunkin numeron arvon määrityksessä.

```
volatile const char  digits[10] =    { 0x30, 0x31, 0x32, 0x33, 0x34,
                                       0x35,0x36, 0x37, 0x38, 0x39 };
volatile const int   decade[3] =     { 1, 10, 100 };
```

AD-muuntimen muunnostulokset tallennetaan taulukkoon *samples[]*. Taulukon koko on sama kuin DTC:n siirtojen lukumäärä. DTC kirjoittaa AD-muuntimen muunnostulokset suoraan taulukon alkioiden muistipaikkoihin, ja taulukkoa *samples[]* käytetään vain arvojen lukemiseen.

```
volatile unsigned int                samples[ARRAY_SIZE] = { 0 };
```

4.2.2 Moduulien alustukset

Watchdog Timer pysäytetään, jotta se ei häiritse ohjelman toimintaa testatessa.

```
WDTCTL = WDTPW + WDTHOLD;            // Stop WDT+
```

DCO:n taajuudeksi asetetaan 1 MHz.

```
BCSCTL1 = CALBC1_1MHZ;               // Set DCO
DCOCTL = CALDCO_1MHZ;
```

AD-muunnin asetetaan muuntamaan yhtä kanavaa jatkuvasti, kunnes muunnos pysäytetään ja kanavaksi valitaan A0 eli nasta P1.0. Vertailujännitteeksi valitaan sisäisen 2,5 V:n jännite. Ideaalitilanteessa vertailujännitteenä toimisi kiihtyvyysanturin käyttöjännite, jolloin kiihtyvyysanturin ulostulo olisi aina oikein verrannollinen vertailujännitteeseen käyttöjännitteen muutoksista huolimatta. ADC10AE0-rekisteristä nasta P1.0 analogiseksi, jolloin nastan puskuri poistetaan käytöstä ja siitä syntyvä loisvirta häviää. Ulkoisen V_{EREF+}-vertailujännitteen sallittu maksimiarvo MSP430G2553-kontrollerilla on kontrollerin käyttöjännite V_{CC}, joka on 3,6 V, ja kiihtyvyysanturin käyttöjännite on 5 V. Tämän vuoksi kiihtyvyysanturilta AD-muuntimelle tulevan signaalin DC-taso on korkeampi kuin vertailualueen puoliväli, mutta arvot ovat ainakin testausvaiheessa tarpeeksi tarkkoja. Sample-and-hold aika eli t_{SAMPLE} määritetään kahdeksan AD-muuntimen kellojakson pituiseksi. AD-muuntimen keskeytys sallitaan asettamalla ADC10CTL-kontrollirekisterin ADC10IE-bitti. ADC10DTC1-kontrollirekisteriin kirjoitetaan DTC:n siirtämien muunnosten lukumäärä, joka on sama kuin taulukon *samples[]* koko.

```
ADC10CTL1 = CONSEQ_2 + INCH_0 +      // Repeat single channel, input A0,
            SHS_2;                   // SHI source Timer_A0 OUT0
ADC10AE0 |= 0x01;                    // P1.0 ADC option select
ADC10CTL0 =
     SREF_1 + REFON + REF2_5V +      // Reference 2,5 V, sample-and-hold
     ADC10SHT_1 + ADC10ON + ADC10IE; // 8 × ADC10CLKs, ADC10 on,
                                     // ADC interrupt enable.
ADC10DTC1 = ARRAY_SIZE;              // Number of conversions stored by DTC.
```

Timer_A-ajastinta käytetään 1 ms:n viiveen luomiseen, jotta AD-muuntimen vertailujännite ehtii asettua alustuksen jälkeen. Viive voisi olla paljon lyhyempikin, mutta käytännössä 1 ms:n viivettäkään ei huomaa. TACCR0-rekisteriin kirjoitetaan arvo, jonka saavuttamisen jälkeen sallitaan keskeytys ja käynnistetään Timer_A. Siirrytään virransäästötilaan, josta poistutaan kun Timer_A:n laskuri saavuttaa TACCR0-rekisterissä olevan arvon. Keskeytysaliohjelmassa nollataan TACCR0-rekisteri ja sen jälkeen estetään Timer_A:n keskeytys. Timer_A-ajastimen ulostuloa käytetään myös AD-muuntimen tahdistukseen, jotta näytteiden välinen aika voidaan asettaa halutuksi. Siksi TACCR0-rekisteriin kirjoitetaan vielä arvo lopuksi ja ulostulon tyypiksi valitaan tilaa vaihtava. Käytännössä näytteiden välinen aika on siis kaksinkertainen, koska AD-muunnos käynnistyy aina Timer_A:n ulostulon nousevalla reunalla.

```
__enable_interrupt();                 // Enable interrupts.
TACCR0 = 0x03E8;                      // 1 ms delay to allow Ref to settle
TACCTL0 |= CCIE + OUTMOD_4;           // Compare interrupt, output mode toggle.
TACTL = TASSEL_2 + MC_1;              // TACLK = SMCLK, Up mode.
__bis_SR_register(CPUOFF + GIE);      // Wait for delay.
TACCTL0 &= ~CCIE;                     // Disable timer Interrupt
__disable_interrupt();                // Disable interrupts.
TACCR0 = 0x0032;                      // Time between ADC samples / 2
```

Portti 1:n pinneistä P1.0 ja P1.7 määritetään sisääntuloiksi AD-muunninta ja kiihtyvyysanturilta tulevaa STATUS-signaalia varten. P1.1 ja P1.2 määritellään P1SEL- ja P1SEL2 -rekistereiden avulla USCIUART:n lähetystä ja vastaanottoa varten. Näiden suunnalla ei ole väliä, koska UART ohjaa niitä suoraan. UART:n kellolähteeksi tulee SMCLK joka on sama kuin DCO, koska esijakaja on SMCLK:lla oletuksena 1. Sarjaliikenteen nopeudeksi valitaan 9600 baudia UCABR0- ja UCABR1 -rekistereiden avulla. Tämä on MSP430 Launchpadin sarjaliikenteen maksiminopeus. Lopuksi käynnistetään UART.

```
P1DIR = 0x7E;                         // P1.0 and P1.7 as inputs
P1OUT = 0;                            // All P1.x reset
P1SEL = BIT1 + BIT2 ;                 // P1.1 = RXD, P1.2=TXD
P1SEL2 = BIT1 + BIT2 ;                // P1.1 = RXD, P1.2=TXD
UCA0CTL1 |= UCSSEL_2;                 // SMCLK
UCA0BR0 = 104;                        // 1MHz 9600
UCA0BR1 = 0;                          // 1MHz 9600
UCA0MCTL = UCBRS0;                    // Modulation UCBRSx = 1
UCA0CTL1 &= ~UCSWRST;                 // Initialize USCI state machine
```

Sallitaan USCI0RX-keskeytys.

```
IE2 |= UCA0RXIE;                      // Enable USCI_A0 RX interrupt
```

4.2.3 Aliohjelmat

Ohjelmassa usein käytettävät komennot on koottu omiin aliohjelmiinsa, joka selkeyttää koodia ja helpottaa sen muokkaamista.

AD-muuntimen pysäytys tapahtuu *stop_adc()*-aliohjelmassa. ADC10CTL0-kontrollirekisterin ENC-bitti nollataan, jolloin jatkuva muunnos pysähtyy käynnissä olevan muunnoksen ollessa valmis. Seuraavassa while-silmukassa odotetaan, kunnes AD-muunnin ei ole aktiivinen, jonka jälkeen käännetään AD-muuntimen keskeytys pois päältä nollaamalla ADC10CTL10-kontrollirekisterin ADC10IE-bitti.

```
void stop_adc(void)
{
      ADC10CTL0 &= ~ENC;              // Stop conversion
      while (ADC10CTL1 & BUSY);       // Wait if ADC10 core is active
      ADC10CTL0 &= ~ADC10IE;
}
```

AD-muunnoksen aloitus tapahtuu *start_conversion*-aliohjelmassa. AD-muuntimen keskeytys sallitaan asettamalla ADC10CTL-kontrollirekisterin ADC10IE-bitti ja käännös aloitetaan asettamalla ENC- ja ADC10SC-bitit. Bittien asetus tehdään erillisinä käskyinä, jotta keskeytys on varmasti asetettu ennen AD-muunnoksen käynnistämistä.

```
void start_conversion(void)
{
      ADC10CTL0 |= ADC10IE;           // Enable ADC10 interrupt
      ADC10CTL0 |= ENC + ADC10SC;     // Start conversion
}
```

DTC:n käynnistys tapahtuu *start_dtc()*-aliohjelmassa. Parametrina tuodaan taulukon *samples[]* osoite, joka on sen muistipaikan osoite, josta DTC aloittaa AD-muunnosten siirron. ADC10DTC1-kontrollirekisteriin kirjoitetaan siirrettävien muunnosten lukumäärä ja ADC10SA-rekisteriin kirjoitetaan aloitusosoite.

```
void start_dtc(unsigned int table_address)
{
      ADC10DTC1 = ARRAY_SIZE;         // Number of conversions stored by DTC.
      ADC10SA = table_address;        // DTC starting address set as table
                                      // samples address.
}
```

Merkin lähetys tapahtuu *transmit()*-aliohjelmassa, jolle tuodaan parametrina lähetettävän merkin ASCII-koodi. While-silmukassa odotetaan, että USCIAB0TX-keskeytyslippu on asetettu, jonka jälkeen kirjoitetaan lähetettävä merkki UCA0TXBUF-lähetyspuskuriin.

```
void transmit(unsigned char digit)
{
      while (!(IFG2&UCA0TXIFG));      // USCI_A0 TX buffer ready?
      UCA0TXBUF = digit;              // TX character
}
```

Desimaaliluvun muunnos ASCII-merkeiksi tapahtuu *dec_to_ascii()*-aliohjelmassa, jolle tuodaan parametrina muunnettava desimaaliluku. Apumuuttujiksi alustetaan *digit*- ja *digit_count* -muuttujat. Desimaaliluku muutetaan numero kerrallaan ASCII-merkeiksi ja lähetetään niin ikään numero kerrallaan. Koska osamäärän desimaalit häviävät eikä osamäärää pyöristetä ylöspäin, saadaan sata- ja kymmenkerrat laskettua helposti. Jokaisen kierroksen jälkeen desimaaliluvusta vähennetään muunnetun sata- tai kymmenkerta, riippuen onko kyseessä ensimmäinen vai toinen kierros. Aliohjelman lopuksi lähetetään rivinvaihto.

```
void dec_to_ascii(unsigned int value)
{
      volatile unsigned int digit = 0x0000;
      volatile unsigned char digit_count = 0x00;
      for( digit_count = 3; digit_count > 0; digit_count-- )
      {
            digit = (value / decade[digit_count-1]);
            transmit(digits[digit]);
            value = value - digit * decade[digit_count-1];
      }
      transmit('\n');
}
```

4.2.4 Keskeytysaliohjelmat

Keskeytysaliohjelmissa ei suoriteta paljoa käskyjä, koska on hyvä pitää keskeytykset lyhyinä, jottei sitä kautta ilmaannu toimintahäiriöitä.

Timer_A-ajastimen keskeytystä käytetään viiveen luomiseen, jotta AD-muuntimen vertailujännite ehtii asettua alustuksen jälkeen. Timer_A:n keskeytysaliohjelmassa vain poistutaan virransäästötilasta. Myöhemmin keskeytystä ei enää tarvita AD-muuntimen tahdistamiseen.

```
__interrupt void TA0_ISR(void)
{
      __bic_SR_register_on_exit(CPUOFF);        // Exit LPM0 on return
}
```

COMPARATORA_ISR-keskeytysaliohjelmassa sytytetään LED ja käynnistetään AD-muunnos. Virransäästötilasta ei vielä poistuta, vaan se tapahtuu AD-muunnoksen valmistuttua.

```
__interrupt void COMPARATORA_ISR(void)
{
      P1OUT |= 0x40;                            // Turn on LED
      start_conversion();
}
```

AD-muuntimen keskeytys suoritetaan, kun DTC on lopettanut viimeisen muistipaikan kirjoittamisen. Tällöin asetetaan *dtc_finished*-lippu ja poistutaan virransäästötilasta.

```
__interrupt void ADC10_ISR(void)
{
      dtc_finished = 1;
      stop_adc();
      __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

USCI0TX_ISR-keskeytysaliohjelmaan ei itse asiassa koskaan tulla, koska keskeytystä ei milloinkaan sallita. On kuitenkin hyvä määritellä kaikki keskeytykset, jotta säästytään mahdollisilta toimintahäiriöiltä.

```
__interrupt void USCI0TX_ISR(void)
{
      __bic_SR_register_on_exit(CPUOFF);
}
```

USCI0RX_ISR-keskeytysaliohjelma suoritetaan, aina kun vastaanotetaan merkki. Jos vastaanotettu merkki on u, asetetaan *start_transmit*-lippu.

```
__interrupt void USCI0RX_ISR(void)
{
      if ( UCA0RXBUF == 'u' )              // 'u' received?
      {
            start_transmit = 1;
      }
      __bic_SR_register_on_exit(CPUOFF);
}
```

4.2.5 Pääohjelma

Pääohjelmassa kirjoitetaan DTC:lle aloitusosoite ja mennään virransäästötilaan. Virransäästötilassa ollaan niin pitkään, kunnes DTC on kirjoittanut lohkon täyteen ja aiheuttaa keskeytyksen. Muunnostulokset kirjoitetaan sarjaporttiin, jos lohko on kirjoitettu täyteen tai on vastaanotettu ennalta määritetty kirjain. Keskeytykset estetään lähetyksen ajaksi, etteivät ne häiritse lähetystä tai käynnistä AD-muunninta kesken lähetyksen. Ennen lähetystä nollataan *start_transmit* ja *dtc_finished* -liput. LED sammutetaan lähetyksen loputtua.

```
while(1)
{
      start_dtc((unsigned int)&samples[0]);      // Data buffer start
      __bis_SR_register(CPUOFF + GIE);           // Go to Low Power Mode 0

      if( dtc_finished || start_transmit )
      {
            __disable_interrupt();
            start_transmit = 0;
            dtc_finished = 0;

            for(i = 0; i < (ARRAY_SIZE); i++)
            {
                  dec_to_ascii(samples[i]);
            }
            P1OUT &= ~0x40;                      // Turn off LED
            __enable_interrupt();
      }
}
```

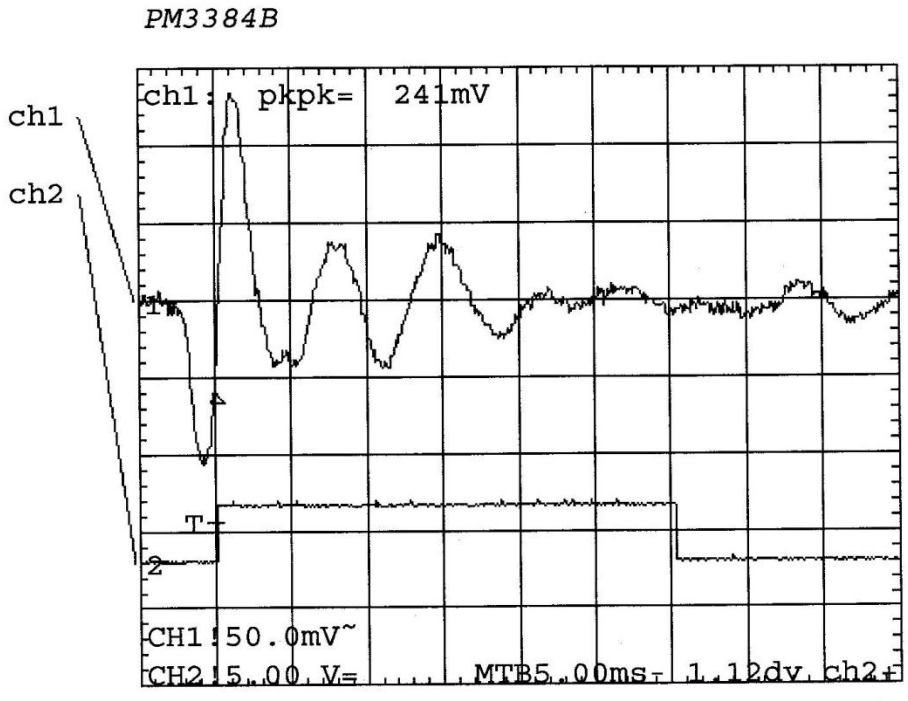
4.3 Kytkentä

Kytkentä suunniteltiin EAGLE 6.3.0-ohjelmalla, jolla voidaan suunnitella myös piirilevy. MSP430G2553 löytyy suoraan TI:n EAGLE-kirjastosta, mutta MMA2301EG-kiihtyvyysanturille täytyi luoda malli itse. Kytkentää ei päästä vielä testaamaan, vaan testaus tapahtuu MSP430 Launchpad -kehitysalustan avulla. Kytkentä on esitetty liitteessä 1.

Laite tarvitsee kaksi käyttöjännitettä, koska kiihtyvyysanturin käyttöjännite on 5 V ja prosessorin käyttöjännite on 3,6 V, jotka luodaan jänniteregulaattoreilla. AD-muuntimen sisääntulon ja kiihtyvyysanturin ulostulon väliin tulee ensimmäisen asteen passiivinen alipäästösuodin kiihtyvyysanturin datalehden mukaisesti. Komparaattoria varten signaalia täytyy vahvistaa, joka tapahtuu operaatiovahvistimella. Vahvistimelle menevästä signaalista suodatetaan tasajännitetaso pois ensimmäisen asteen passiivisella ylipäästösuotimella. Muunnoksen käynnistymistä ilmaistaan LEDillä. USB-väylään liittämistä varten tarvitaan USB-kontrolleripiiri, josta voidaan myös saada käyttöjännite pariston tai akun lisäksi. Tähän kytkentään ei USB-kontrolleripiiriä kuitenkaan vielä ole lisätty.

4.4 Testaus

Toimintaa testattiin mittaamalla iskukohtaa oskilloskoopilla ja vertailemalla oskilloskoopin tulostetta AD-muunnokseen. Iskurilla iskettiin nurinpäin olevan säilyketölkin pohjaa. Ennen varsinaista mittausta varmistettiin, että perättäisten iskujen tulokset ovat lähellä toisiaan, josta voitiin päätellä mittauksen olevan luotettava. Ohjelmaan tehtiin myös pieni muutos testausta varten, jossa P1.6-nasta asetetaan komparaattorin keskeytysaliohjelmassa ja nollataan DTC:n valmistuttua AD-muuntimen keskeytysaliohjelmassa. Näin nähdään helposti miltä väliltä AD-muunnos on tehty. AD-muuntimen tahdistus Timer_A:n avulla toimi hyvin, ja muunnoksen kesto on tarkalleen haluttu eli 30 ms. Tässä tapauksessa TACCR0-rekisterin arvo on 004Bh eli 75 µs vastaava arvo, jolloin muunnos liipaistaan 150 µs:n välein. Vaikka näytteiden määrää ei saatukaan lisättyä, huomattiin 200 näytteen olevan hyvin riittävä määrä. Oskilloskooppituloste on esitetty kuvassa 9. Kuvasta nähdään myös jonkin verran esiintyvää kohinaa, jota ehkä saataisiin pienennettyä lisäämällä datalehden mukaisesti kiihtyvyysanturin alle maataso. Testikytkennässä tämä ei kuitenkaan ollut mahdollista.



KUVA 9. Oskilloskoopin tuloste. CH1 on kiihtyvyysanturilta tuleva signaali ja CH2 on AD-muunnoksen pituuden ilmaiseva signaali

AD-muunnoksen loputtua ohjelma lähettää muunnostuloksen kokonaislukuarvoina USB-porttiin, ja se voidaan kopioida ja liittää terminaalista taulukkolaskentaohjelmaan. Desimaaliluku muunnettiin kiihtyvyyttä vastaavaksi jännitearvoksi V_a kaavalla 3:

$$V_a = V_{REF}\left(\frac{G-G_0}{1024}\right) \quad (3)$$

jossa V_{REF} = AD-muuntimen vertailujännite (2,5 V)
G = Kiihtyvyyttä vastaava kokonaisluku
G_0 = Kiihtyvyyttä vastaava kokonaisluku, kun iskuri on lepotilassa

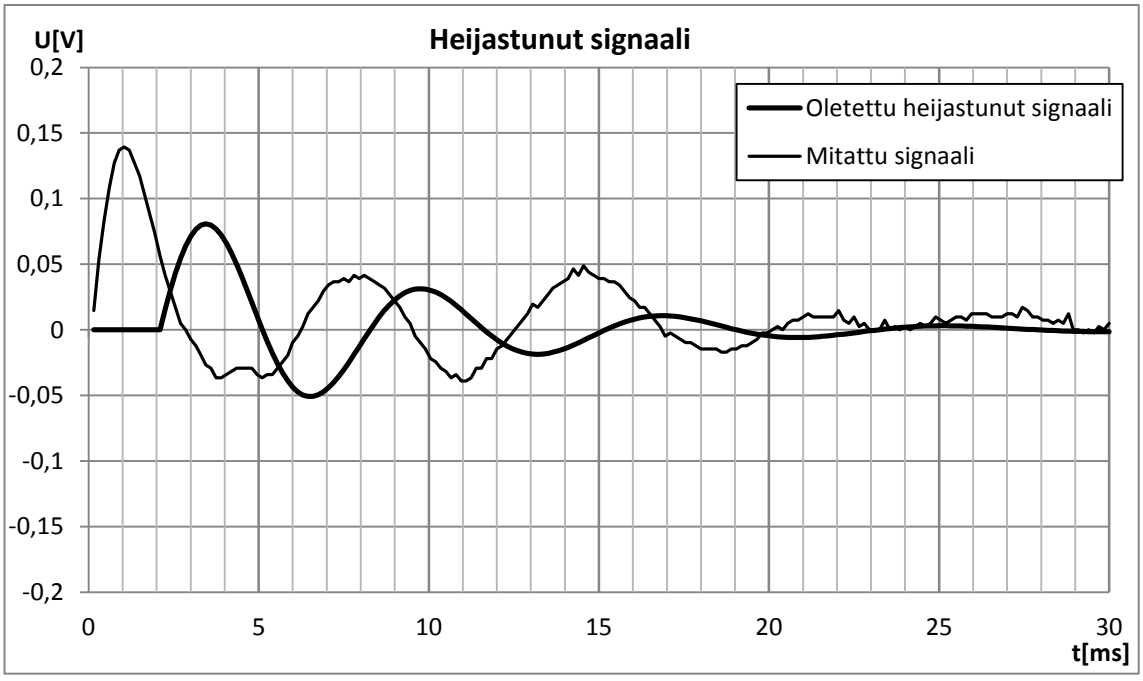
Kuvaajalla on sama asteikko kuin oskilloskoopilla, joka helpottaa kuvien vertailua keskenään. Jännitteestä saadaan kiihtyvyys jakamalla sitä vastaava jännitearvo V_0 kiihtyvyysanturin herkkyydellä S, joka on 10 mV/g. Kiihtyvyyden saa siis helposti kertomalla jännitteen sadalla. Kiihtyvyyden suurin arvo oli tässä mittauksessa noin 14 g, joka näkyy myös taulukkolaskentaohjelmalla piirretystä kuvaajasta. Kuvaaja vastaa hyvin oskilloskoopin tulostetta, joten AD-muunnosta voidaan sanoa luotettavaksi. Kertoimilla ja asteikkoa muuttamalla kuvasta saataisiin ehkä luettavampi, mutta nyt keskityttiin vain vertailemaan kuvia keskenään. Piirretty kuvaaja on esitetty kuvassa 10.

KUVA 10. Taulukkolaskentaohjelmalla piirretty kuvaaja mittauksesta

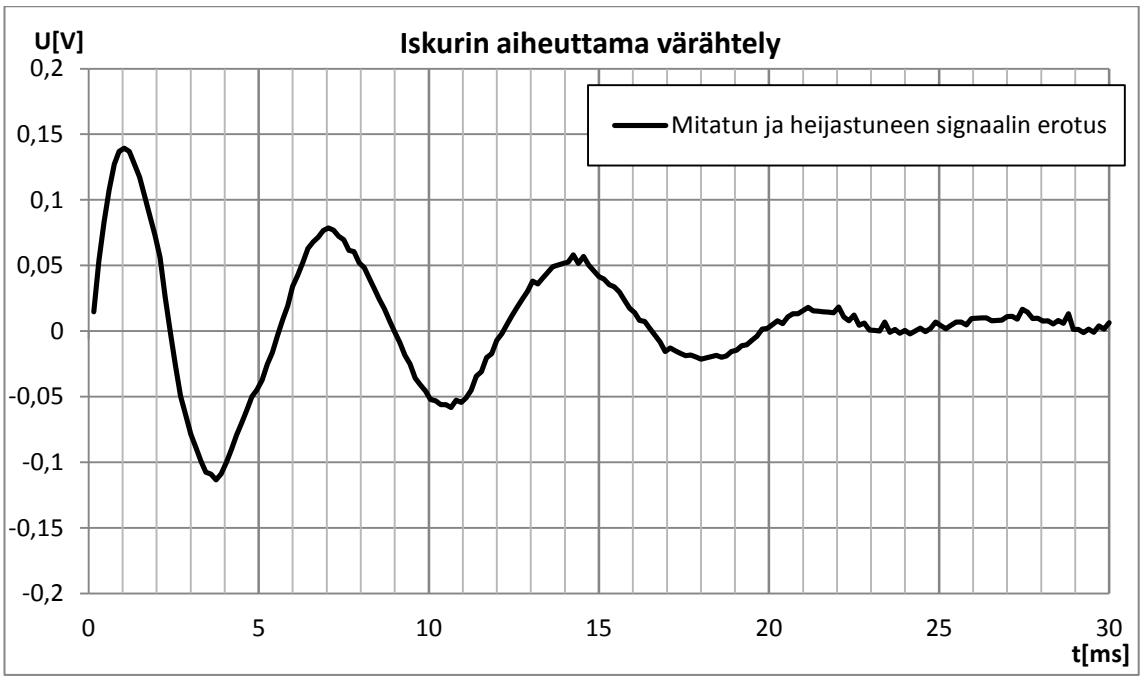
Koska mittaustulos on todennäköisesti pääasiassa iskurin aiheuttaman värähtelyn ja materiaalista heijastuneen värähtelyn summa, luotiin heijastunutta signaalia kuvaava signaali. Heijastunutta signaalia viivästettiin sen verran, että se osuu mitatun signaalin ensimmäiseen epäjatkuvuuskohtaan. Vähentämällä luotu heijastunut signaali mittaustuloksesta pyritään saamaan näkyviin pelkkä iskun aiheuttama värähtely tai ainakin sitä lähellä oleva kuvaaja. Signaali luotiin kaavalla 4, joka on tasaisesti vaimenevan sinisignaalin kaava:

$$y(t) = e^{-\lambda t} \cdot \sin(2\pi f t) \tag{4}$$

Koska värähtelyn taajuus oletettavasti pienenee liikkeen hidastuessa, luodun signaalin taajuus *f* pienenee tasaisesti 170 Hz:stä 130 Hz:iin. Nämä taajuudet löydettiin kokeilemalla ja niiden todettiin tuottavan järkevimmän tuloksen. Tuotettu heijastunut signaali on esitetty kuvassa 11 ja sen avulla saatu iskurin aiheuttama värähtely on esitetty kuvassa 12.

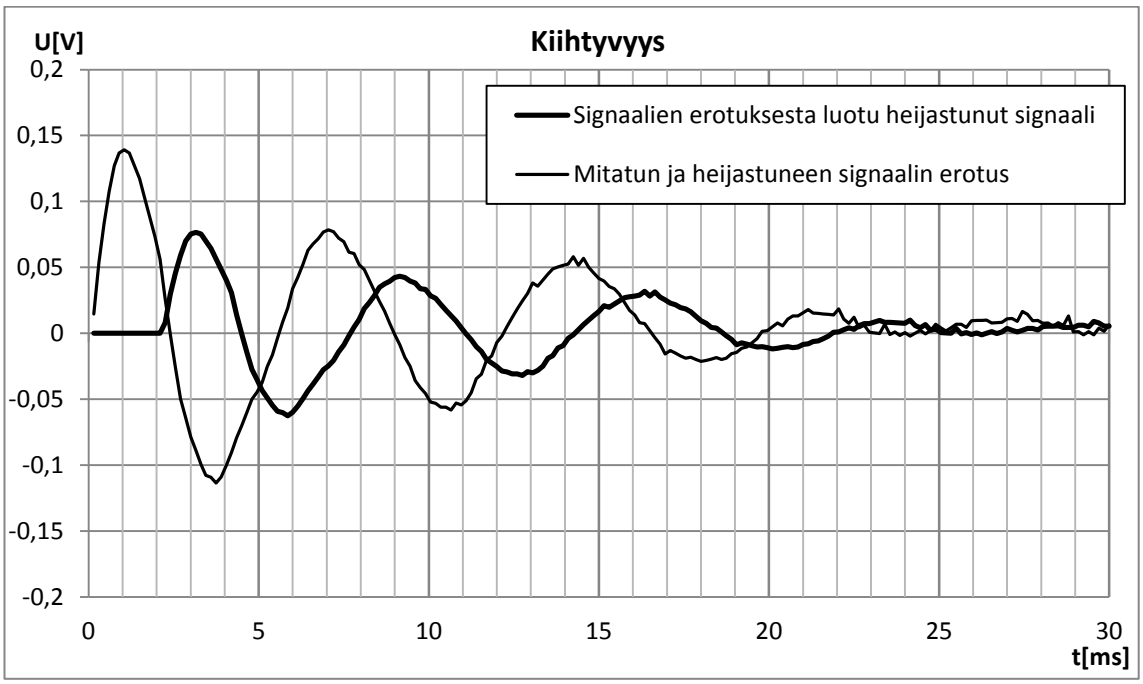


KUVA 11. Taulukkolaskentaohjelmassa luotu heijastunut signaali

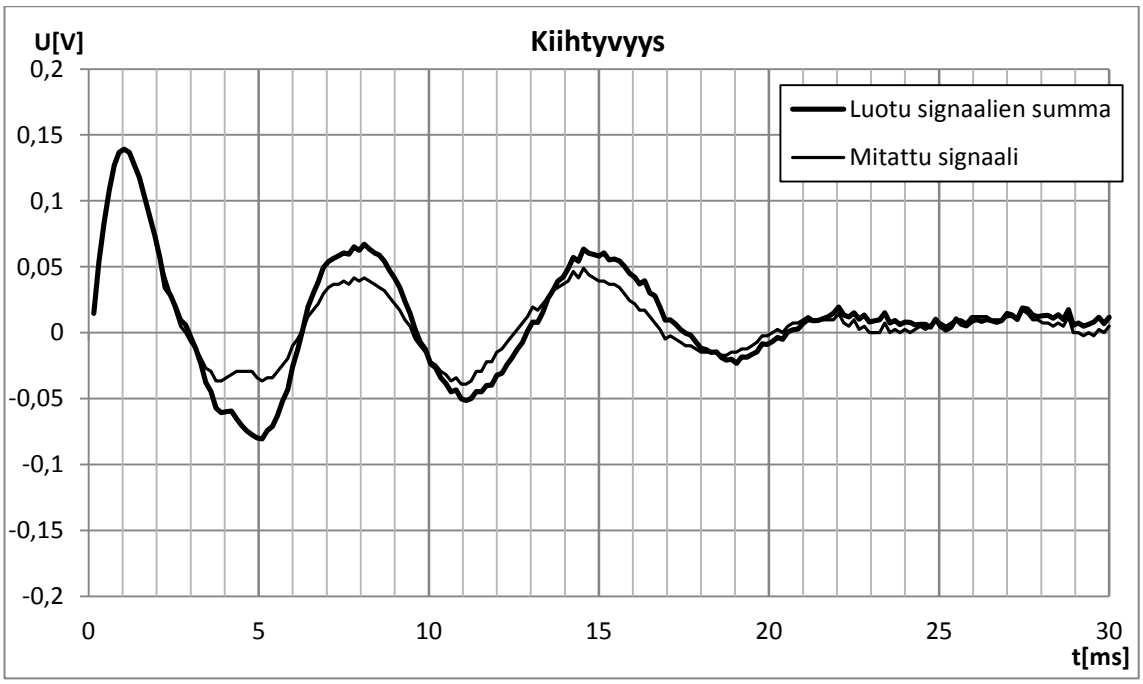


KUVA 12. Luodun heijastuneen signaalin avulla saatu iskurin aiheuttama värähtely

Tuloksena saatu iskurin aiheuttama värähtely muistuttaa jo paljon vaimenevaa sinisignaalia. Toimenpide tehtiin vielä käänteisenä, eli summattiin tulos samasta signaalista viivästetyn ja vaimennetun signaalin kanssa, joka on esitetty kuvassa 13. Luotu mittaustulos on esitetty kuvassa 14.



KUVA 13. Signaalien erotuksesta luotu heijastunut signaali



KUVA 14. Luotu mittaustulos

Tulos on samansuuntainen kuin oikea mitattu signaali, joten voidaan olettaa oletetun heijastuneen signaalin olevan oikean kaltainen. Käytetty menetelmä on kuitenkin melko alkeellinen ja tarkempaa signaalien erotusta varten pitäisi tarkastella mitatun signaalin spektriä.

5 POHDINTA

Vaikka aihe vaikuttaa yksinkertaiselta, löytyi siitä paljon syvyyttä heti työn alkuvaiheessa. Varsinkin MSP430-prosessoreissa on paljon eri toimintoja joita voi käyttää monipuolisesti kyseessä olevan sovelluksen mukaan. Valittu kiihtyvyysanturi MMA2301EG toimi sovelluksessa hyvin, mutta jatkossa kannattaa ehkä valita jokin herkempi kiihtyvyysanturi, jotta saataisiin mitattua pienempiä kiihtyvyyksiä tarkemmin. Komponentit valittiinkin lähinnä edullisen hinnan perusteella, eikä kaikkia ominaisuuksia osattu siinä vaiheessa ottaa huomioon. Nyt huomattiin, että isku täytyy olla melko voimakas, ennen kuin kunnollisia lukemia saatiin aikaiseksi. Toinen vaihtoehto on iskurin iskuvoiman kasvattaminen.

Ohjelman tekeminen oli siinä mielessä hankalaa, että vaadittavia ominaisuuksia ei oltu määritelty. Tosin valmiiden esimerkkien ja kokeilujen kautta saatiin koodi toimimaan halutulla tavalla. Jatkossa panostetaan mittaustulosten tarkentamiseen, mittaustulosten tallentamiseen myöhempää tarkastelua varten, käyttöliittymään ja vähävirtaisuuteen sekä lisätään laitteeseen näyttö tuloksien lukua varten. Laitteen käyttötarkoituksen takia prototyypistä pitää saada myös kannettava versio, jotta mittauksia voidaan suorittaa myös oikeassa ympäristössä eli luonnonjäällä. Virtalähteeksi sopisi tällöin pitkäikäinen paristo tai ladattava akku, jota voitaisiin ladata esimerkiksi USB-väylän kautta.

Mittauksissa ei ehditty kokeilla useita eri materiaaleja eri paksuuksilla eikä mittaustuloksia ehditty tarkastella tarpeeksi, jotta voitaisiin luotettavasti laskea isketyn kappaleen paksuus. Jos luotettavia tuloksia ei saada laajemmalla testauksella ja tarkemmilla mittaustuloksilla, voidaan tilanteen mukaan harkita myös muita mittaustekniikoita, kuten mikroaaltolähetintä ja -vastaanotinta.

Sovelluksella voi olla muitakin käyttökohteita jään paksuuden mittaamisen lisäksi, kuten virheiden etsintä kappaleista, veden etsiminen maaperästä ja muiden epäsäännöllisyyksien etsintä erilaisista kappaleista ja materiaaleista.

LÄHTEET

[1] Wilson, Jon S. 2005. Sensor Technology Handbook. Elsevier.

[2] Freescale Semiconductor Inc. 2009. MMA2301EG Datasheet Rev 0.
http://cache.freescale.com/files/sensors/doc/data_sheet/MMA2301KEG.pdf

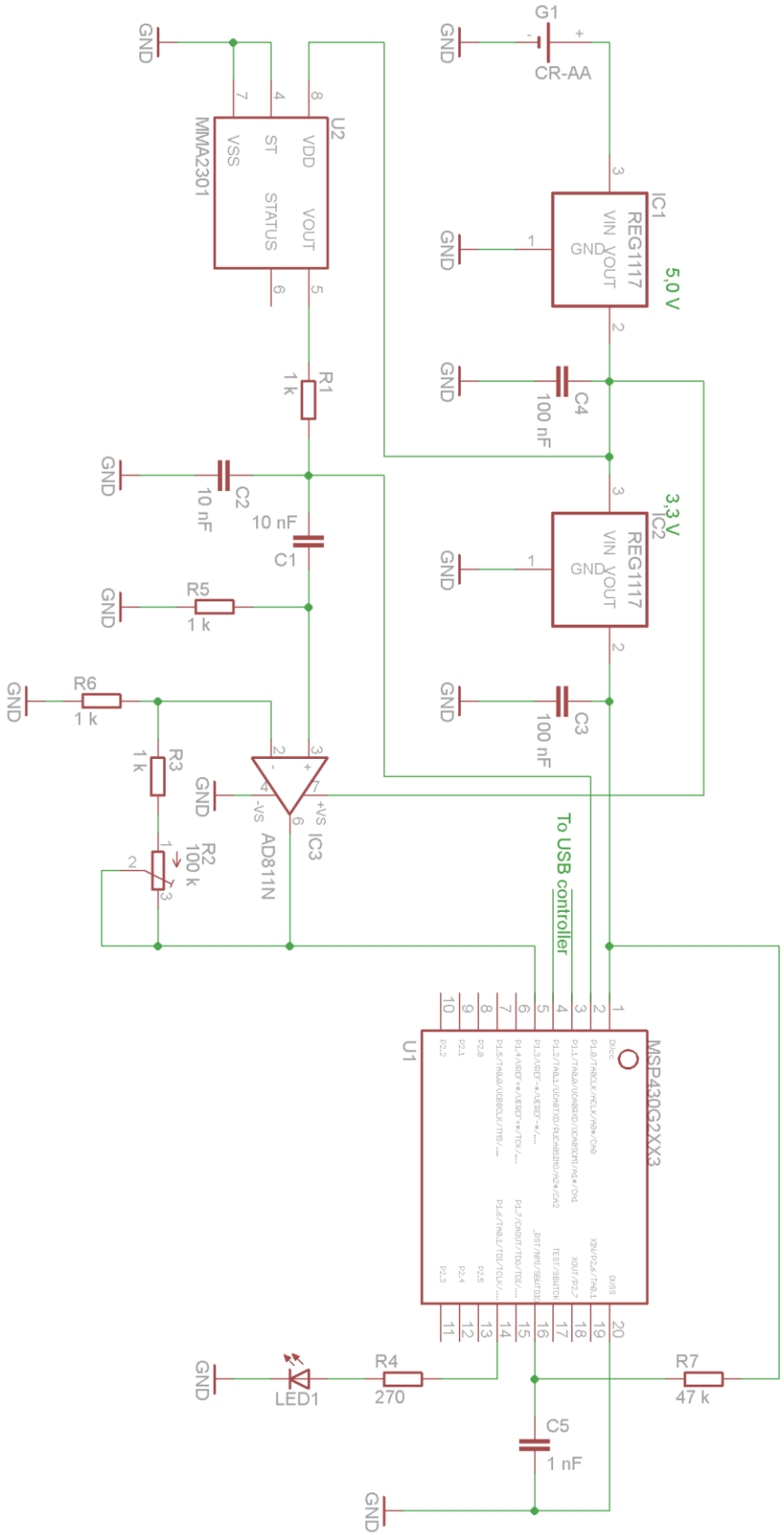
[3] Texas Instruments. 2012. MSP-EXP430G2 LaunchPad Experimenter Board User's Guide.
http://www.ti.com/lit/ug/slau318b/slau318b.pdf

[4] Texas Instruments. 2012. MSP430x2xx Family User's Guide.
http://www.ti.com/lit/ug/slau144i/slau144i.pdf

[5] Texas Instruments. 2012. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. E).
http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

LIITTEET

Liite 1. Kytkentäkaavio



Liite 2. Ohjelman lähdekoodi

```
/*************************************
* Niko Heikkinen
* Opinnäytetyö, kiihtyvyysanturin luku
*
* 12/2012
*************************************/


#include  "msp430g2553.h"

#define ARRAY_SIZE 200

static volatile unsigned short int i = 0;
static volatile unsigned char start_transmit = 0, dtc_finished = 0;

volatile const char digits[10] =
                        { 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39 };
volatile const short int decade[3] = { 1, 10, 100 };
volatile unsigned short int samples[ARRAY_SIZE] = { 0 };

void stop_adc(void);
void start_conversion(void);
void start_dtc(unsigned short int);
void transmit(unsigned char);
void dec_to_ascii(unsigned short int);


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

      // AD-muuntimen alustus
      ADC10CTL1 = CONSEQ_2 + INCH_0 +           // Repeat single channel, A0
                  + SHS_2;                      // SHI source Timer_A OUT0
      ADC10CTL0 = SREF_1 + REFON + REF2_5V +    // Reference 2,5 V on,
                  ADC10SHT_1 +                  // sample-and-hold 8 × ADC10CLKs.
                  ADC10ON + ADC10IE;            // ADC10 on, ADC interrupt enable

      // Viive, jotta ADC referenssi ehtii asettua
      __enable_interrupt();                     // Enable interrupts.
      TACCR0 = 0x03E8;                          // 1 ms delay to allow Ref to settle
      TACCTL0 |= CCIE + OUTMOD_4;               // Compare-mode interrupt.
      TACTL = TASSEL_2 + MC_1;                  // TACLK = SMCLK, Up mode.
      __bis_SR_register(CPUOFF + GIE);          // Wait for delay.
      TACCTL0 &= ~CCIE;                         // Disable timer Interrupt
      __disable_interrupt();                    // Disable interrupts.
      // Viive loppuu
```

```
	TACCR0 = 0x0032;                          // time between conversions / 2

	ADC10DTC1 = ARRAY_SIZE;                   // number of conversions stored by DTC
	ADC10AE0 |= 0x01;                         // P1.0 ADC option select

	P1DIR = 0x76;                             // P1.0 and P1.7 as inputs
	P1OUT = 0;                                // All P1.x reset

	// UARTin alustus
	P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
	P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
	UCA0CTL1 |= UCSSEL_2;                     // SMCLK
	UCA0BR0 = 104;                            // 1MHz 9600
	UCA0BR1 = 0;                              // 1MHz 9600
	UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
	UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
	IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

	// Komparaattorin alustus
	CACTL1 = CAEX + CAREF_2 + CAON +          // 0.25 Vcc = -comp, on
	         CAIES + CAIE;
	CACTL2 = P2CA2 + P2CA1 + CAF;             // P1.1/CA1 = +comp
	CACTL1 &= ~CAIFG;                         // Reset Comparator Interrupt Flag

	while(1)
	{
		start_dtc((unsigned short int)&samples[0]);     // Data buffer start
		__bis_SR_register(CPUOFF + GIE);                // Go to Low Power Mode 0

		if(dtc_finished || start_transmit)
		{
			__disable_interrupt();
			start_transmit = 0;                 // Reset flags
			dtc_finished = 0;                   // Reset flags
			for(i = 0; i < (ARRAY_SIZE); i++)   // Send each value in table to PC
			{
				dec_to_ascii(samples[i]);     // Convert dec to ASCII and send
			}
			P1OUT &= ~0x40;                     // Turn off LED1
			__enable_interrupt();
		}
	}
}
```

```
/*
*     TimerA0 keskeytysaliohjelma. Käytetään vain viiveen luomiseen.
*     Timer pysäytetään ja poistutaan virransäästötilasta.
*/
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TA0_ISR(void)
{
      P1OUT ^= 0x40;
      __bic_SR_register_on_exit(CPUOFF);        // Exit LPM0 on return
}

/*
*     ADC10 keskeytysaliohjelma
*     Kun DTC kirjoittaa lohkon loppuun, keskeytys tapahtuu ja
*     dtc_finished-lippu asetetaan sekä pysäytetään ADC.
*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
      dtc_finished = 1;
      stop_adc();
      __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}

/*
*     UART TX-keskeytysaliohjelma
*     Ei käytössä, koska keskeytystä ei sallita missään vaiheessa.
*     Tähän tultaisiin, kun lähetetään merkki.
*/
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
      __bic_SR_register_on_exit(CPUOFF);
}

/*
*  UART RX-keskeytysaliohjelma
*     Tähän tullaan, kun vastaanotetaan merkki.
*     Jos merkki on 'u', asetetaan start_transmit-lippu.
*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
      if ( UCA0RXBUF == 'u' )                   // 'u' received?
      {
            start_transmit = 1;
      }
      __bic_SR_register_on_exit(CPUOFF);
}
```

```
/*
*     Komparaattorin keskeytysaliohjelma
*     Tähän tullaan, kun operaatiovahvistimelta tuleva signaalin amplitudi
*     ylittää vertailujännitteen. Sitten sytytetään LED ja käynnistetään AD-muunnos.
*/
#pragma vector=COMPARATORA_VECTOR
__interrupt void COMPARATORA_ISR(void)
{
      P1OUT |= 0x40;                                  // Turn on LED1
      start_conversion();
}

/*
*     AD-muuntimen pysäytys.
*     Pysäyttää käynnissä olevan muunnoksen ja estää AD-muuntimen keskeytyksen
*/
void stop_adc(void)
{
      ADC10CTL0 &= ~ENC;                          // Stop conversion
      while (ADC10CTL1 & BUSY);                   // Wait if ADC10 core is active
      ADC10CTL0 &= ~ADC10IE;                      // Disable ADC10 interrupt
}

/*
*     AD-muuntimen käynnistys
*     Sallii AD-muuntimen keskeytyksen ja käynnistää muunnoksen
*/
void start_conversion(void)
{
      ADC10CTL0 |= ADC10IE;                       // Enable ADC10 interrupt
      ADC10CTL0 |= ENC + ADC10SC;                 // Start conversion
}

/*
*     DTC:n käynnistys
*     Kirjoittaa DTC:lle muunnosten määrän ja aloitusosoitteen, joka on taulukon osoite.
*     Osoite tuodaan parametrina taulukko_address.
*/
void start_dtc(unsigned short int taulukko_address)
{
      ADC10DTC1 = ARRAY_SIZE;                     // Number of conversions stored by DTC
      ADC10SA = taulukko_address;                 // DTC starting address set as table
                                                  // samples address.
}
```

```
/*
*     Merkin lähetys
*     Lähettää merkin kirjoittamalla sen lähetyspuskuriin.
*     Lähetettävä merkki tuodaan parametrina digit, ja sen on oltava jo ASCII-muodossa.
*/
void transmit(unsigned char digit)
{
      while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX buffer ready?
      UCA0TXBUF = digit;                        // TX character
}

/*
*     Desimaaliarvon muunto ASCII-merkeiksi
*     Desimaaliarvo tuodaan parametrina value, ja muunnetaan ASCII-merkeiksi numero
*     kerrallaan. ASCII-merkit ovat taulukossa digits[], ja alkion indeksiä vastaa sama
*     luku ASCIIna. Jokainen merkki lähetetään erikseen heti, kun se on muunnettu.
*     Lopuksi lähetetään rivinvaihto.
*/
void dec_to_ascii(unsigned short int value)
{
      volatile unsigned short int digit = 0x0000;
      volatile unsigned char digit_count = 0x00;

      for( digit_count = 3; digit_count > 0; digit_count-- )
      {
            digit = (value / decade[digit_count-1]);
            transmit(digits[digit]);
            value = value - digit * decade[digit_count-1];
      }
      transmit('\n');
}
```