Toni Niittyjoki

Langaton anturiverkko

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Kone- ja tuotantotekniikka

Insinöörityö

9.4.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Toni Niittyjoki Langaton anturiverkko 36 sivua + 2 liitettä 9.4.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Kone- ja tuotantotekniikka
Suuntautumisvaihtoehto	Koneautomaatio
Ohjaaja	Lehtori Jari Savolainen, Metropolia Ammattikorkeakoulu

Tämän insinöörityön tarkoituksena oli kehittää langaton mittausjärjestelmä kahden Texas Instrumentsin CC430F6137-mikrokontrollerin välille. Työ tehtiin Metropolia Ammattikorkeakoululle, osana Flexwise-projektia.

Mittausjärjestelmässä mitattiin kiihtyvyyttä ja kulmanopeutta langattomasti kolmessa ulottuvuudessa. Antureina käytettiin Murata Electronicsin SCA3100-kiihtyvyysanturia ja CMR3000-kulmanopeusanturia. Mittausjärjestelmässä anturit kytkettiin mikrokontrolleriin, josta SPI-väylää pitkin saadut akselien arvot lähetettiin radion kautta toiselle mikrokontrollerille. Radion kautta saadut arvot lähetettiin eteenpäin sarjaväylää pitkin tietokoneelle, jossa arvoja voitiin analysoida ja tallentaa.

Molemmille mikrokontrollerille suunniteltiin ohjelmisto C-kielellä, Code Composer Studio-kehitysympäristöä hyväksi käyttäen.

Avainsanat	SPI, mikrokontrolleri, SCA3100, CMR3000

Author Title Number of Pages Date	Toni Niittyjoki Wireless sensor network 36 pages + 2 appendices 9 April 2013
Degree	Bachelor of Engineering
Degree Programme	Mechanical Engineering
Specialisation option	Machine Automation
Instructor	Jari Savolainen, D.Sc (tech.)

The purpose of this Bachelor's thesis was to develop wireless measuring equipment between two Texas Instruments CC430F6137 microcontrollers. The study was carried out for Metropolia University of Applied Sciences, as part of Flexwise project.

The meaning of the measuring equipment was to measure acceleration and angular velocity wirelessly in three dimensions. Murata Electronics SCA3100 acceleration sensor and CMR3000 gyroscope were used as sensors. The sensors were connected to a microcontroller and axel values from the sensors were read through the SPI bus. Value information was transmitted by radio to another microcontroller. This information was then forwarded to the computer via serial bus, where it was analyzed and saved.

This study was successful in developing the wireless measuring equipment. Software was also developed for both microcontrollers by using the C programming language, in Code Composer Studio development environment.

Keywords	SPI, microcontroller, SCA3100, CMR3000

Sisällys

Liitteet

Liite 1. EM430F6137RF900+SCA3100+CMR3000, lähetyspuolen koodi

Liite 2. EM430F6137RF900+SCA3100+CMR3000, vastaanottopuolen koodi

Lyhenteet

ADC	Analog-to-Digital converter. Muuntaa analogisen signaalin digitaaliseksi.
I2C	Inter-Intergrated Circuit. Philipsin kehittelemä kaksisuuntainen tiedonsiirtoväylä.
JTAG	Joint Test Action Group. JTAG on yleinen nimi IEEE 1149.1 -standardille. Se määrittää mm. laitteiston testaamiseen ja debuggaukseen käytettävän JTAG-portin.
LSB	Least significant byte. LSB tarkoittaa vähiten merkitseviä tavuja, jos jokin arvo ilmaistaan vaikka kahtena eri tavuna.
MSB	Most significant byte. MSB tarkoittaa eniten merkitseviä tavuja. LSB ja MSB tavut voidaan laskea yhteen laittamalla kaikki MSB -bitit LSB -bittien vasemmalle puolelle.
RISC	Reduced Instruction Set Computer. Yksi suorintinarkkitehtuurien suunnittelufilosofioista, jossa käskyt pyritään pitämään hyvin yksinkertaisina.
SoC	System on chip. Integroitu piiri, jossa on yhdistetty kaikki tietokoneen osat yhdeksi mikrosiruksi.
SPI	Serial Peripheral Interface bus. Motorolan kehittelemä synkroninen datansiirtostandardi.
UART	Universal Asynchronous Receiver/Transmitter. Osa tietokoneen laitteistosta, joka muuttaa rinnakkaismuotoisen tiedon sarjamuotoiseksi.
USB	Universal Serial Bus. Sarjaväyläarkkitehtuuri, käytetään usein kommmunikointiin ja käyttöjännitteeseen tietokoneen ja oheislaitteiden välillä.

1 Johdanto

Tässä insinöörityössä oli tarkoitus kehittää langaton mittausjärjestelmä Texas Instrumentsin CC430F6137-mikrokontrollerille. Työ tehtiin Metropolian koneautomaatiolaboratoriossa ja se oli osana Flexwise-projektia. Langaton anturiverkkoprojekti oli alkanut jo vuonna 2011 kurssimuotoisena, jolloin projektia tehtiin ryhmätyönä. Tuolloin projekti jäi kuitenkin alkumetreille, eikä antureita saatu luettua mikrokontrollereilla. Työ päätettiin viedä loppuun insinöörityön merkeissä.

Työn päätarkoituksena oli saada anturit toimimaan mikrokontrollerilla, jotta niillä voitiin mitata kiihtyvyyttä sekä kulmanopeutta kolmessa ulottuvuudessa. Työssä antureina käytettiin Murata Electronics Oy:n 3D MEMS -kulmanopeusanturia CMR3000 ja kiihtyvyysanturia SCA3100. Järjestelmässä on toisessa mikrokontrollerissa liitettynä molemmat CMR3000 ja SCA3100 -anturit. Tämä mikrokontrolleri-anturi-yhdistelmä lähettää anturien arvoja toiselle laudalle langattomasti. Toinen lauta vastaanottaa arvot ja lähettää ne edelleen sarjaväylää pitkin tietokoneelle, jossa tehdään tarvittavat laskutoimenpiteet saaduille antureiden mittausarvoille.

2 Laitteisto ja ohjelmisto

2.1 Mikrokontrolleri

Mikrokontrolleria voidaan sanoa erittäin pieneksi tietokoneeksi jota ohjelmoidaan toimimaan oman käyttötarkoituksen mukaisesti. Mikrokontrollerit sisältävät usein lähtö- ja tulopinnejä joihin voidaan liittää lisälaitteita tai joilla voidaan vastaanottaa ja lähettää tietoa erilaisia väyliä pitkin. Usein mikrokontrollereihin on lisätty erilaisia ajastimia ja datan siirtoon tarvittavia komponentteja kuten SPI ja UART. Joissakin mikrokontrollereissa voi olla jo valmiiksi asennettuja antureita kuten lämpötila-anturi, mutta yleensä anturit liitetään mikrokontrolleriin lisälaitteena.

2.1.1 MSP430-mikrokontrolleriperhe

MSP430 on Texax Instrumentsin valmistama mikrokontrolleriperhe, joka on rakennettu 16-bittisen RISC-suorittimen ympärille. MSP430 on keskittynyt pieneen virran kulutukseen ja pieneen hintaan. Perheeseen kuuluu monta eri tuotesarjaa, mutta tärkeimmät sarjat ovat 3xx, 4xx, 5xx ja 6xx. Jokaisella numerolla on tietty merkitys. Toinen numero eli ensimmäinen numero sarjanumeron jälkeen on mallin numero, mitä isompi numero sitä kehittyneempi malli. Kolmas numero ilmoittaa, kuinka paljon muistia kyseisessä mallissa on.

Mikrokontrollereita on siis melkoinen liuta. Kaikki kuitenkin ovat hieman erilaisia, jotta jokaiseen käyttötarkoitukseen löytyisi sopiva malli. Kaikissa sarjoissa on ADC-muunnin vakiona. Muita lisäosia ovat mm. timerit, lämpötila-anturi ja erilaiset väylät datan siirtoon, esim. SPI. [1]

2.1.2 CC430F6137-mikrokontrolleri

Tässä työssä käytettiin RF SoC (CC430) -sarjan mallia CC430F6137 (kuva 1), koska tämä kyseinen malli mahdollisti radioyhteyden muodostamisen kahden mikrokontrollerin välille ja SPI-väylän käytön mikrokontrollerin ja anturien välillä.

Kuva 1. CC430F6137 mikrokontrolleri.

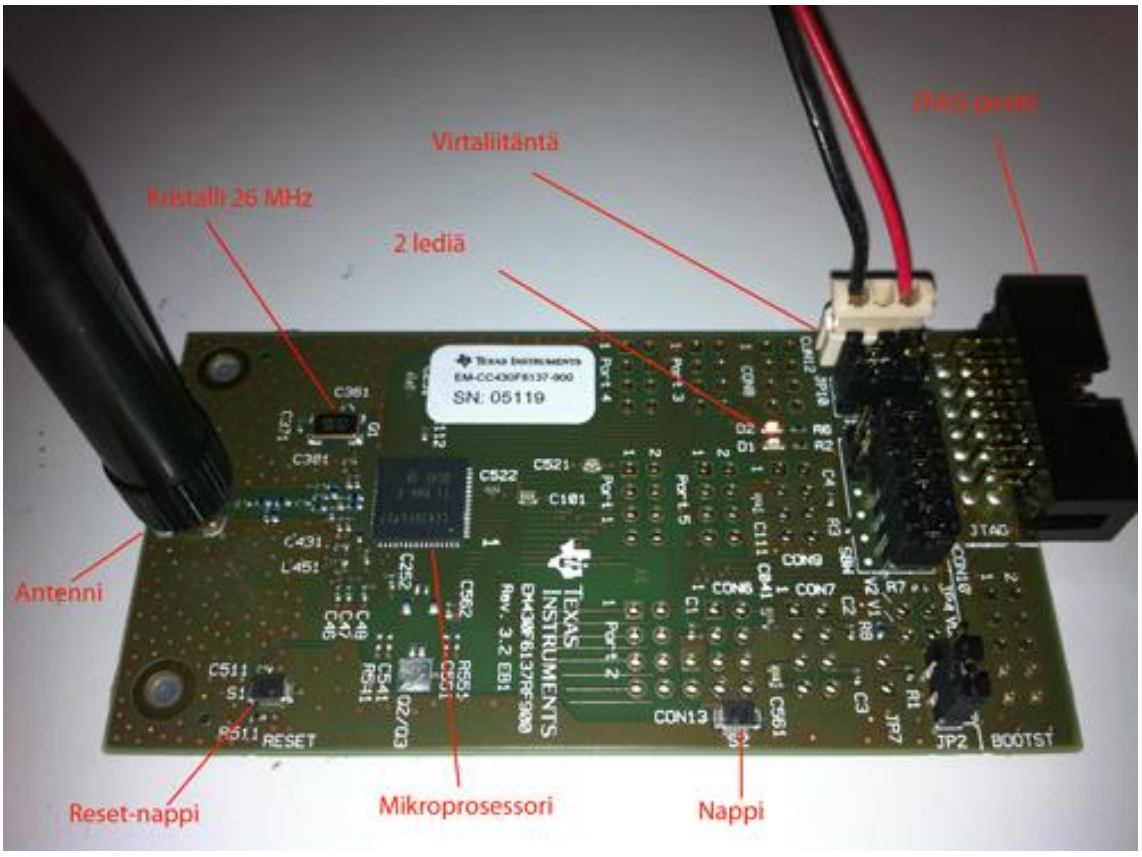
Tärkeimmät ominaisuudet:

- System-on-Chip (Soc) langaton kommunikointi pienellä virrankulutuksella
- Laaja käyttöjännite: 3.6 V – 1.8 V
- Pienet virrankulutukset:
    - Aktiivi tila: 160 µA/MHz
    - Virransäästötila: 2 µA
    - Radio vastaanotto modessa: 15 mA
- MSP430:n koneisto ja lisälaitteet
    - 16-bittinen RISC-suoritinarkkitehtuuri

- Flash-muisti 32 KB
- Herätys virransäästötilasta < 6 µs
- Kaksi 16-bittistä ajastinta
- Kaksi mahdollista paikkaa datansiirtoon, toisessa mm. UART ja toisessa SPI ja I2C
- 12-bittinen AD-muunnin [2]

2.1.3 EM430F6137RF900-kehitysalusta

EM430F6137RF900 (kuva 2) on valmis paketti, jolla voidaan luoda langaton yhteys kahden mikrokontrollerin kanssa. Langattoman yhteyden luomiseen käytetään SimpliciTI-radiotaajuusprotokollaa. Paketti sisältää kaksi langatonta CC430-alustaa, joihin on valmiiksi asennettu CC430F6137-mikrokontrolleri, 868/915 MHz antenni, 2 ledivaloa, 2 painiketta, 18 kpl 4x2 pin-liitintä, liitännät pattereille sekä JTAG-portti debuggerille. EM430F6137RF900-kehitystyökalupaketissa on kaikki mitä tarvitsee yksinkertaiseen langattomaan yhteyteen, eikä käyttäjän tarvitse itse tehdä minkäänlaisia liitäntöjä. Laudassa on runsaasti vapaita pinnejä mahdollisille lisälaitteille sekä tiedonsiirtoväylille. [3.]



Kuva 2. EM430F6137RF900-kehitysalusta.

2.2 Anturit

Microelectromechanical systems, lyhyemmin MEMS, ovat komponentteja, joissa tapahtuu vähintään kahden eri toiminnon yhdistämistä samaan pakettiin. Toiminnot voivat olla esimerkiksi mekaanisia, optisia, akustisia ja biologisia. MEMS-komponentit ovat erittäin pieniä, koot ovat 1 - 100 µm. MEMS-komponentteja käytetään useissa eri sovelluksissa mm. kiihtyvyysantureina autojen turvatyynyissä, peliohjaimissa, uusimmissa älypuhelimissa, digitaalikameroissa, kulmanopeusantureina autoissa ja veneissä ja mikrofoneina puhelimissa, kannettavissa tietokoneissa sekä kuulokkeissa.

Murata Electronics Oy eli entinen VTI Technologies tarjoaa laajan valikoiman MEMS-antureita. Murata valmistaa antureita mm. autoteollisuuteen, kulutuselektroniikkaan, terveysteknologiaan ja ilmailuteollisuuteen.

2.2.1 SCA3100-kiihtyvyysanturi

Projektissa käytettiin Murata Electronicsin valmistamaa SCA3100-kiihtyvyysanturia (kuva 3). Kiihtyvyyden lisäksi SCA3100-kiihtyvyysanturilla pystytään mittaamaan myös lämpötilaa, mutta tässä työssä lämpötilan mittaus ei ollut oleellista. SCA3100 on suunniteltu erilaisiin autosovelluksiin, esimerkiksi mäkilähtö-, jarrutus- ja jousitustoimintoihin.

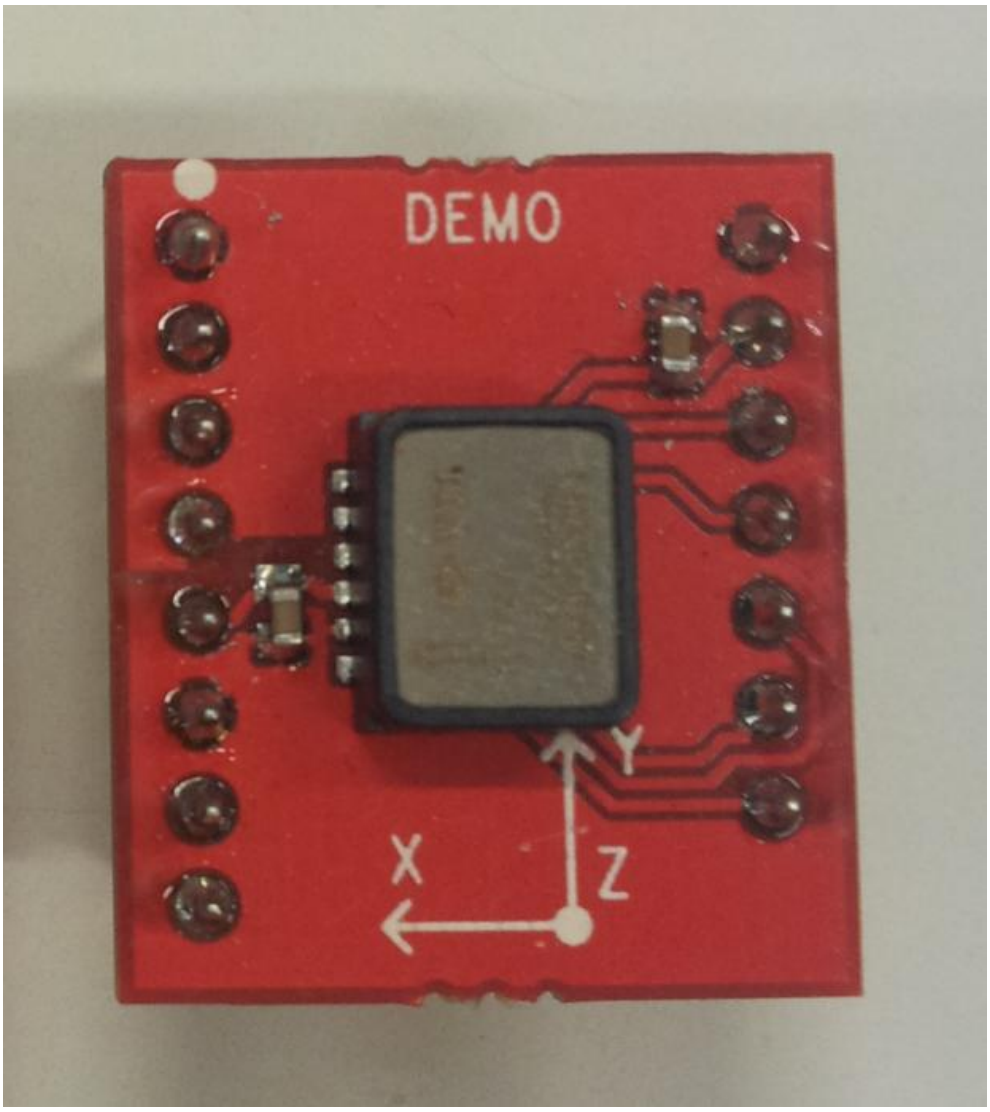
Kuva 3. SCA3100-kiihtyvyysanturi.

SCA3100-kiihtyvyysanturilla voidaan mitata kolmea eri akselia, x, y ja z. Kiihtyvyysanturin mittausalue on ±2 g. Anturilla voidaan siis mitata melko suuria kiihtyvyyksiä. Tarjolla on myös suuremmilla ja pienemmillä mittausalueilla olevia antureita, välillä ± 0.5 g – 6 g.

Anturin koko leveyssuunnassa on 7.6 mm, pituussuunnassa 8.6 mm ja korkeussuunnassa 3.3 mm. Syöttöjännitealue on 3 – 3.6 V. Virrankulutus on vain 3 mA ja virransäästötilassa vain 0.12 mA. Toimintalämpötila-alue -40…+125°C.

SCA3100-kiihtyvyysanturia voidaan kontrolloida ja lukea SPI- tai I2C-väylää hyväksi käyttäen. Tässä työssä käytettiin SPI-väylää. SCA3100 tarjoaa SPI:n kellontaajuudeksi jopa 8 MHz, työssä pystyttiin käyttämään nopeutena vain 500 kHz, koska CMR3000-kulmanopeusanturi ei tukenut suurempaa nopeutta. [4.]

Projektissa käytetyt anturit oli valmiiksi asennettu noin 15x20 mm:n piirilevyille, joihin oli valmiiksi kiinnitetty tarvittavat kondensaattorit ja liitännät, mikä helpotti anturin liittämistä omaan sovellukseen. Kiihtyvyysanturi oli hieman suurempi kuin kulmanopeusanturi, mutta se oli kuitenkin kiinnitetty samankokoiseen piirilevyyn, joten niiden asentaminen ja paikkojen vaihto onnistui erittäin sujuvasti. (Kuva 4.)



Kuva 4. SCA3100-D04, SCA3100-kiihtyvyysanturi kiinnitettynä piirilevyyn.

2.2.2 CMR3000-kulmanopeusanturi

Toisena anturina projektissa käytettiin Murata Electronicsin vuonna 2010 julkaisemaa CMR3000-kulmanopeusanturia (kuva 5). CMR3000 oli tällöin markkinoiden pienin ja vähävirtaisin kulmanopeusanturi. Anturi on suunniteltu erilaisiin liikeohjaussovelluksiin, pelilaitteisiin ja matkapuhelimiin. Sittemmin Murata Electronics on lopettanut CMR3000-kulmanopeusanturin tuotannon.

Kuva 5. CMR3000-kulmanopeusanturi.

CMR3000 on 3-akselinen kulmanopeusanturi. Kulmanopeutta voidaan mitata samanaikaisesti x-, y- ja z-akseleiden ympäri. Mittausalue on ±2000 °/s, joten anturilla voidaan mitata erittäin nopeita liikkeitä. CMR3000 on todella pieni, leveys 3.1 mm, pituus 4.1 mm ja paksuus vain 0.83 mm. Syöttöjännitealue anturilla on 2.5 V - 3.6 V ja virrankulutus vain 5 milliampeeria. Toimintalämpötila-alue -40…+85°C. CMR3000-kulmanopeusanturi ei siis kestä aivan niin kovia lämpötiloja kuin SCA3100-kiihtyvyysanturi. (Kuva 6.)

Samoin kuin SCA3100-anturissa, CMR3000-anturissakin on mahdollisuus käyttää joko SPI- tai I2C-väylää. CMR3000 tarjosi SPI kellontaajuuden vain 500 kHz. [5.]

Kuva 6. CMR3000-D01, CMR3000-kulmanopeusanturi kiinnitettynä piirilevyyn.

2.2.3 Rekisterit

Rekisterit ovat muistialueita, jotka sisältävät pienen määrän bittejä. Rekistereihin voidaan tallentaa arvoja, jotka esimerkiksi ohjaavat laitteen toimintaa. Rekistereistä voidaan myös lukea tietoja. Tässä työssä kaikki akselien arvot luetaan antureiden rekistereistä. Rekisterit ovat aina yksilökohtaisia, joten jokaisen laitteen rekisterit pitää tarkistaa laitteen teknisistä tiedoista.

CMR3000- ja SCA3100-anturien rekisterit ovat melko samanlaisia, lukuun ottamatta osoitteita. Taulukossa 1 SCA3100-kiihtyvyysanturin rekisterit sekä taulukossa 2 CMR3000-kulmanopeusanturin rekisterit.

Taulukko 1. SCA3100-kiihtyvyysanturin rekisterit .

Osoite	**Nimi**	**Kuvaus**	**Mode (R/RW)**
0x00	REVID	ASIC tarkistus ID numero	R
0x01	CTRL	Control	RW
0x02	STATUS	Status	R
0x03	RESET	Reset	RW
0x04	X_LSB	X-akselin LSB	R
0x05	X_MSB	X-akselin MSB	R
0x06	Y_LSB	Y-akselin LSB	R
0x07	Y_MSB	Y-akselin MSB	R
0x08	Z_LSB	Z-akselin LSB	R
0x09	Z_MSB	Z-akselin MSB	R
.....			
0x12	TEMP_LSB	Lämpötilan LSB	R
0x13	TEMP_MSB	Lämpötilan MSB	R

Taulukko 2. CMR3000-kulmanopeusanturin rekisterit.

Osoite	**Nimi**	**Kuvaus**	**Mode (R/RW)**
0x00	WHO_AM_I	Tunnistusrekisteri	R
0x01	REVID	ASIC tarkistus ID numero	R
0x02	CTRL	Ctrl	RW
0x03	STATUS	Status	R
...	...	...	...
0x0C	X_LSB	X-akselin MSB	R
0x0D	X_MSB	X-akselin MSB	R
0x0E	Y_LSB	X-akselin MSB	R
0x0F	Y_MSB	X-akselin MSB	R
0x10	Z_LSB	X-akselin MSB	R
0x11	Z_MSB	X-akselin MSB	R

REVID-rekisterillä voidaan tarkistaa, että kyseessä on oikea anturi. Tässä tapauksessa REVID-rekisteri oli hyvä tapa tarkistaa, että esimerkiksi SPI-väylä toimi oikein. Lukemalla REVID-rekisteri antureista, SCA3100-kiihtyvyysanturin arvo heksoina on aina 0x22 sekä CMR3000-kulmanopeusanturin arvo on 0x21.

CTRL-rekisterin kautta voidaan kontrolloida mm. ovatko I2C ja keskeytykset päällä ja mikä mittaustila laitetaan päälle. CMR3000:ssa täytyy CTRL-rekisterissä laittaa mittaustila päälle ennen kuin voidaan lukea arvoja akselien rekistereistä. Tämä onnistuu kirjottamalla CTRL-rekisteriin arvo 0x16. Tälloin kytketään I2C pois päältä ja käynnistetään 80 Hz:n mittaustila. SCA3100-anturi on valmiiksi mittaustilassa, kun sen käynnistää.

Jokaisen akselin arvo on jaettu kahteen 8-bittiseen rekisteriin (taulukot 3 ja 4). Toinen on vähiten merkitsevä LSB-tavu ja toinen eniten merkitsevä MSB-tavu. Lopullisen akselin arvon saa liittämällä MSB-tavun LSB-tavun eteen vasemmalle puolelle, eli MSB-rekisteriä siirretään 8 bittiä vasemmalle, jolloin LSB-tavusta ja MSB-tavusta tulee yksi kokonainen 16-bittinen akselin arvo.[6;7]

Taulukko 3. CMR3000-kulmanopeusanturin akselien MSB- ja LSB-bitit.

MSB								LSB							
B7	B6	B5	B4	B3	B2	B1	B0	B7	B6	B5	B4	B3	B2	B1	B0
x	x	s	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	x

Taulukko 4. SCA3100-kiihtyvyysanturin akselien MSB- ja LSB-bitit.

MSB								LSB							
B7	B6	B5	B4	B3	B2	B1	B0	B7	B6	B5	B4	B3	B2	B1	B0
s	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	x	x	x	x

2.3 UART-USB-muunnin

Kerätty tieto antureilta lähetettiin mikrokontrollerilta tietokoneelle sarjaväylää hyväksi käyttäen. Mikrokontrollerin ja tietokoneen väliin asennettiin UART-USB-muunnin (kuva 7), joka mahdollisti kommunikoinnin mikrokontrollerissa olevan UART:n sekä tietokoneessa olevan USB:n välillä. Hankittu kaapeli oli kuitenkin pakko purkaa UART:n päästä, jotta piuhat saatiin kytkettyä oikeisiin pinneihin mikrokontrollerissa.

Kuva 7. UART-USB-muunnin.

2.4 Code Composer Studio

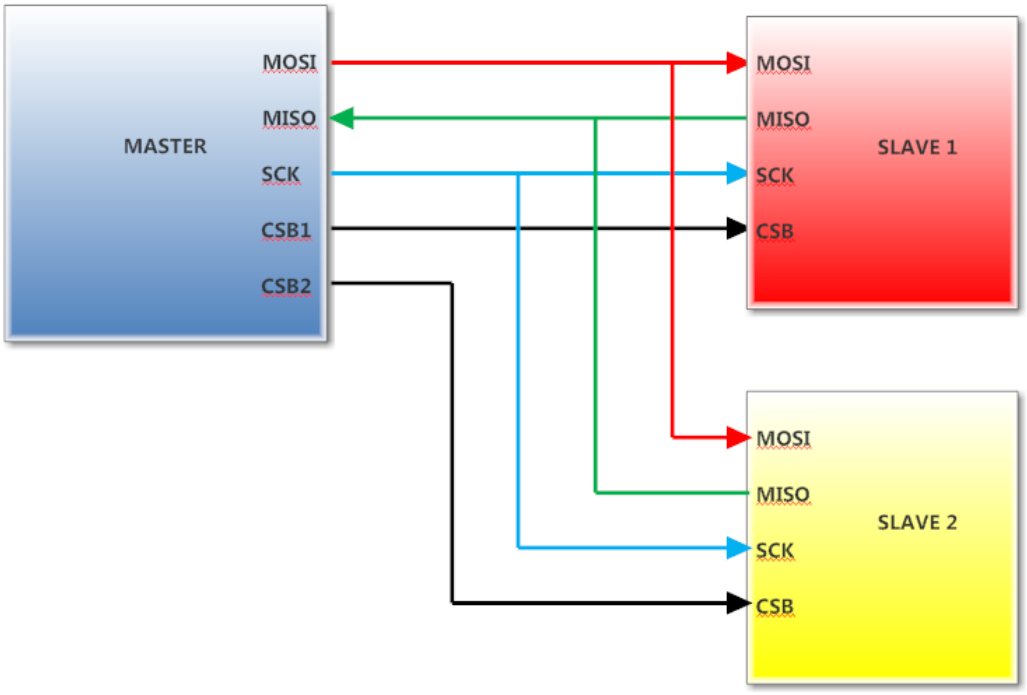
CCS eli Code Composer Studio on Texax Instrumentin kehittelemä ohjelmointiympäristö. Ohjelmalla voidaan kehittää, debugata sekä ladata sulautettuja sovelluksia mikrokontrollereihin. CCS tarjoaa myös kattavan kirjaston sekä paljon esimerkkiohjelmia TI:n laitteistoille.

3 SPI ja SimpliciTI

3.1 SPI-väylä

SPI on hyvin yksinkertainen ja helppo tapa siirtää tietoa kaksisuuntaisesti ja samanaikaisesti. SPI:tä käytetään hyvin paljon mikrokontrollereiden ja niiden oheislaitteiden väliseen kommunikointiin, sillä SPI on tarkoitettu hyvin lyhyisiin tiedonsiirtomatkoihin ja sen virrankulutus on erittäin pieni. SPI-linkissä on aina isäntä- ja orjapuoli, toinen siis toimii lähettäjänä ja toinen vastaanottajana. Yleensä mikrokontrolleri toimii isäntänä ja siihen liitettävä lisälaite orjana. Siirrettävän tiedon koko on yleensä 8 bittiä, mutta suurempiakin bittimääriä pystytään SPI:n avulla siirtämään. Tiedon siirron nopeus on yleensä 1 - 100 MHz.

SPI-väylän muodostamiseen tarvitaan MOSI (Master Out Slave In), suomeksi isäntä lähettää ja orja vastaanottaa, MISO (Master In Slave Out), SCK (Serial Clock) sekä CSB (Chip Select Bit) pinnit. Jokaisella SPI-kellon syklillä isäntä lähettää MOSI:n kautta orjalle tiedon mitä haluaa kysyä, jolloin orja lähettää MISO:a pitkin halutun tiedon isännälle. Orja lähettää isännälle siis ainostaan MISO:n kun taas isäntä lähettää MOSI:n, kellon ja CSB:n. Kuvassa 8 on kuvattu SPI-väylän toiminta, kun isännälle on liitetty kaksi orjaa.



Kuva 8. SPI-väylän toiminta.

SPI-väylään on todella helppo liittää monta eri laitetta. Isäntiä kuitenkin voi olla vain yksi. Jokainen orja liitetään isännässä samoihin MOSI-, MISO- ja kello-pinneihin, mutta jokaisella orjalla on oma paikkansa isännällä CSB-pinnille. CSB-pinnillä isäntä määrittää, miltä tietyltä orjalta isäntä haluaa tietoja. Järjestelmään voidaan siis liittää todella paljon orjia, riippuen tietenkin siitä montako CSB-paikkaa isännällä on tarjota. [8]

3.2 SimpliciTI

SimpliciTI on radioprotokolla joka on kehitetty pieniin radioverkkoihin, jotka toimivat pienellä virran kulutuksella, esim pattereilla. Sen yksinkertaisuus takaa myös vähäisen muistin käytön, vaikkakin asetuksilla on tähän myös vaikutuksensa. SimpliciTI:ä voidaan käyttää mm. autotallin oven ohjaimissa, palovaroittimissa ja kaikenlaisissa muissa pienissä hälytyslaitteissa.

SimpliciTI saadaan toimimaan vain kuudella funktiolla**,** jotka löytyvät suoraan SimpliciTI:n omasta kirjastosta:

- SMPL_Init() alustaminen
- SMPL_Ioctl() radion asetukset
- SMPL_Link() ja SMPL_LinkListen() linkin muodostamiseen
- SMPL_Send() ja SMPL_Receive() viestin lähettämiseen ja vastaanottamiseen [9]

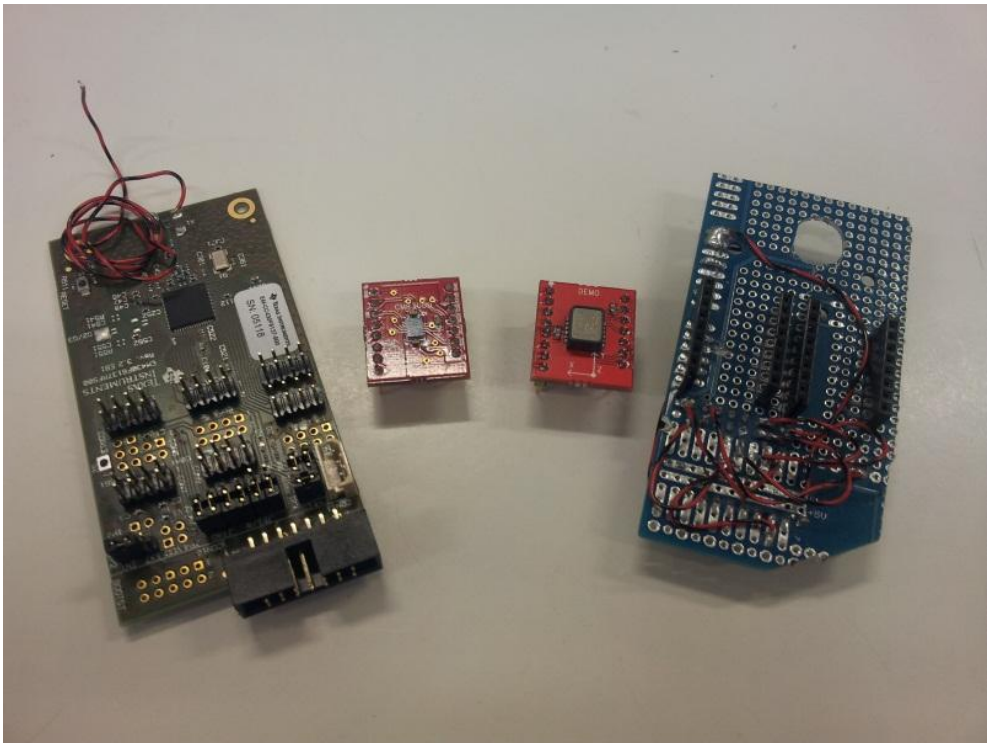
4 Mittausjärjestelmä ja kytkennät

Tässä luvussa käsitellään tehty mittausjärjestelmä, joka koostuu kahdesta EM430F6137RF900-kehitysalustasta, CMR3000-kulmanopeusanturista ja SCA3100-kiihtyvyysanturista.

Mittausjärjestelmän kokoonpanoon ei käytetty juurikaan aikaa, vaan se tehtiin tarvikkeista mitä koneautomaatiolaboratorion hyllystä löytyi, eikä siihen tilattu mitään osia erikseen. Tästä syystä paketista ei tullut kovin kaunista, mutta se ei vaikuttanut järjestelmän toimintaan.

4.1 Lähetyspuoli

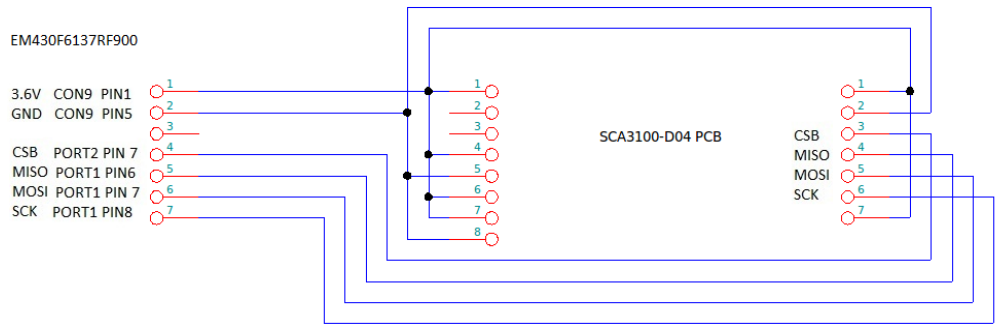
Mittausjärjestelmän lähetyspuolen kokoonpanoon kuului EM430F6137RF900-kehitysalusta, SCA3100-kiihtyvyysanturi sekä CMR3000-kulmanopeusanturi, jotka näkyvät kuvassa 9.



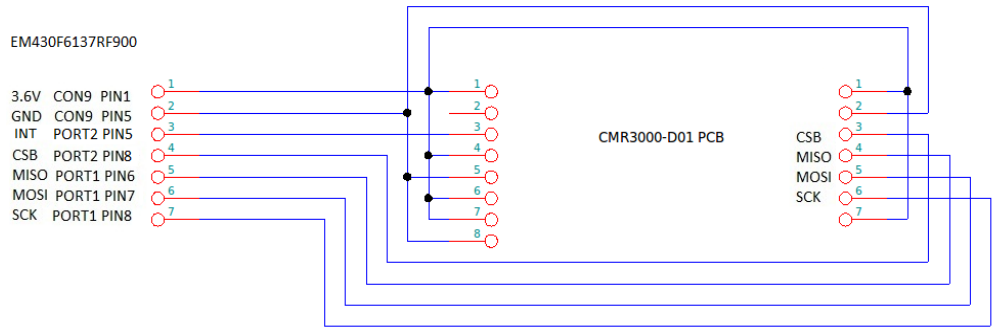
Kuva 9. EM430F6137RF900-mikrokontrolleri, CMR3000, SCA3100 ja asennuslevy antureille.

Suurin ongelma lähetyspuolen kokoonpanossa oli se, miten anturit saadaan kytkettyä tukevasti kehitysalustaan. Anturit päätettiin kiinnittää asennuslevyyn, joka oli tehty arduino-mikrokontrolleria varten, mutta siitä pystyttiin työstämään hyvin tukeva ja pieni paketti. Asennuslevyn toiselle puolelle asennettiin liittimet molempien anturien pinnejä varten ja toiselle puolelle liittimet, jotka sopivat kehitysalustassa oleville tarvittaville pinneille. Tämän jälkeen liittimien väliin kolvattiin johdot oikeille paikoille.

Kuvassa 10 SCA3100-kiihtyvyysanturin ja kuvassa 11 CMR3000-kulmanopeusanturin kytkennät EM430F6137RF900-lautaan. Kytkennät ovat melkein samat, lukuun ottamatta CSB (Chip select) -pinnejä ja CMR3000-anturin INT-pinniä. Kytkentöjä tehdessä piti olla erittäin varovainen, koska anturien liittimet olivat erittäin lähellä toisiaan, eivätkä anturit kestäneet oikosulkuja. Tämän takia jokainen kytkentä tarkistettiin yleismittarilla, jolla pystyttiin selvittämään, koskevatko väärät johdot toisiinsa.

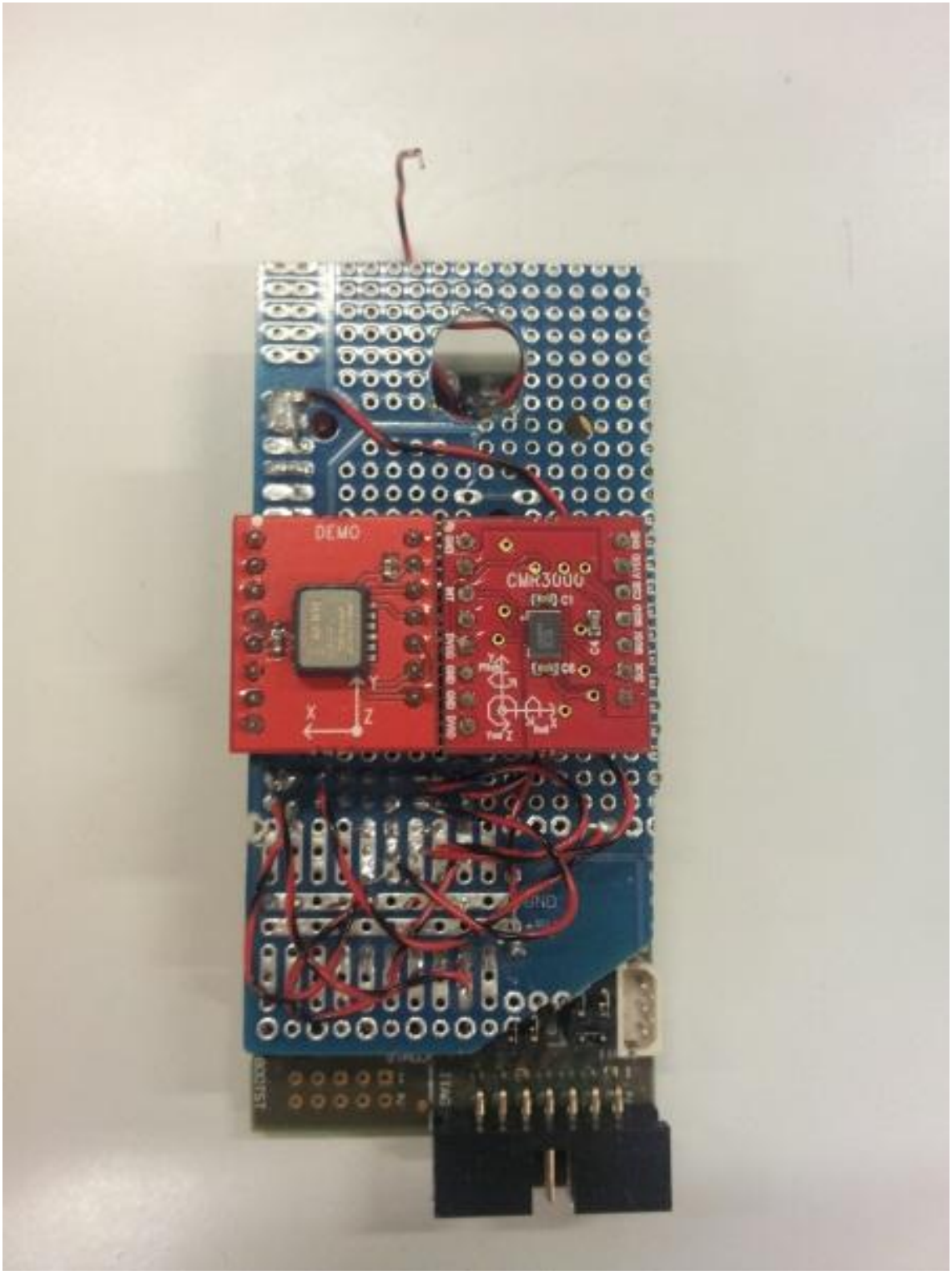


Kuva 10. SCA3100-kiihtyvyysanturin kytkentäkaavio.



Kuva 11. CMR3000-kulmanopeusanturin kytkentäkaavio.

Kuvassa 12 on lähetyspuolen kokoonpano valmiina. Asennuslevy antureineen sopi kehitysalustan päälle erittäin hyvin, eikä kokoonpanon koko suurentunut kuin korkeudessa noin 15 mm. Antureiden vaihto onnistui vaivatta pikaliittimien avulla.



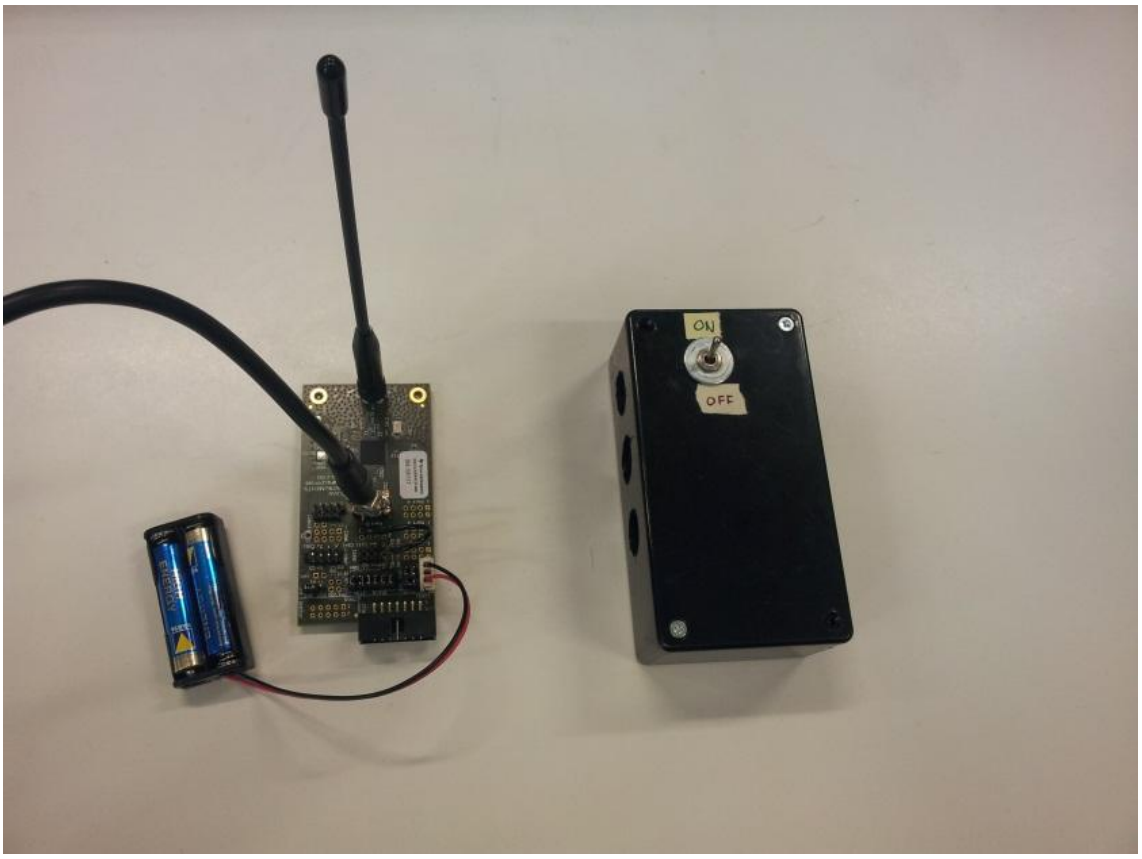
Kuva 12. Anturit kiinnitettynä mikrokontrolleriin.

Kuvassa 13 on lähetyspuolen kokoonpano laitettu mustaan muovikoteloon, jonka avulla kokonaisuudesta saatiin erittäin kiinteä ja suojattu paketti. Komponentit tuettiin kotelon sisällä vaahtomuovilla, jotta kehitysalusta pysyisi paikallaan eikä vaurioituisi kovasta liikkeestä ja mahdollisista iskuista. Koteloon varattiin myös tila kahdelle AAA-sormiparistolle, joista kehitysalusta sai virtansa. Kotelon kanteen lisättiin myös virtakytkin, jolla pystyttiin tarvittaessa käynnistämään kehitysalusta ilman, että paristoja tarvitsisi ottaa irti ja koteloa avata aina, kun järjestelmää ei käytetä. Kotelosta saadaan tarvittaessa pienillä muutoksilla roisketiivis.

Kuva 13. Lähetyspuoli kotelossa.

4.2 Vastaanottopuoli

Järjestelmän vastaanottopuolen kokoonpano oli erittäin yksinkertainen. Siihen kuului vain EM430F6137RF900-kehitysalusta sekä TLL-USB-muunnin. TLL-USB-muuntimessa lähetyspinni TX kytkettiin mikrokontrollerissa portti 2 pinni 7:aan ja vastaanottopinni RX portti 2 pinni 6:een. Kuvassa 14 näkyy vastaanottopuoli vasemmalla ja oikealla lähetyspuoli.



Kuva 14. Valmis mittausjärjestelmä.

5 Ohjelmat

Työssä kehitettiin ohjelmat molempiin mikrokontrollereihin. Toinen ohjelma käsitti anturien luvun sekä arvojen lähetyksen radion kautta toiselle mikrokontrollerille, joka taas vastaanotti arvot ja lähetti edelleen sarjaväylää pitkin tietokoneelle. Molemmat ohjelmat kehiteltiin CCS-kehitysympäristössä. Seuraavissa kappaleissa käydään läpi molempien ohjelmien keskeiset asiat. Liitteenä ohjelmat kokonaisuudessaan.

5.1 EM430F6137RF900 + CMR3000 + SCA3100, lähetys

Tässä kappaleessa käydään läpi lähetyspuolen ohjelman määrittelyt, muuttujat ja tärkeimmät funktiot.

Tavoitteena oli luoda yhteys toiseen mikrokontrolleriin, saada luettua SCA3100-kiihtyvyysanturin ja CMR3000-kulmanopeusnanturin akselien arvot mikrokontrollerilla, sekä lähettää saadut arvot radion kautta toiselle mikrokontrollerille.

5.1.1 Määrittelyt ja muuttujat

Ohjelman alussa lisätään tarvittavat kirjastot kuten bsp.h ja mrfi.h. Bsp.h-kirjaston kautta päästään käsiksi kehitysalustaan vaikuttaviin komentoihin, kuten laudan alustaminen. Mrfi.h-kirjasto sisältää tärkeät komennot radioyhteyden luomiseen.

Kirjastojen lisäämisen jälkeen luodaan tarvittavat muuttujat. Muuttujia luodaan moniin erilaisiin käyttötarkoituksiin: tarvittavien viiveiden luomiseen, SPI-kelloon, molempien anturien porttiasetuksiin ja akselien arvojen tallettamiseen, sekä CMR300-kulmanopeusanturin asetuksia varten muutama muuttuja.

XTAL, TICKSPERMS ja TICKSPERUS (koodi 1) muuttujien avulla luotiin viivefunktiot wait_ms millisekunnin ja wait_us mikrosekunnin viiveille, joita käytettiin myös vastaanottopuolen ohjelmassa.

```
#define XTAL 16000000L
#define TICKSPERMS (XTAL / 1000 / 5 - 1)
#define TICKSPERUS (TICKSPERMS / 1000)
```

Koodi 1. Viiveiden määrittelyt.

SPI-muuttujien SPIFRAMEDELAY ja SPICSBDELAY (koodi 2) avulla saadaan tarvittavat viiveet, kun suoritetaan datansiirtoa SPI:n kautta. Muuttujia käytetään wait_us-funktiossa.

```
#define SPICLOCK 500 //SPI clock = 500kHz
#define SPIFRAMEDELAY ((1000 / SPICLOCK) * 6) // SPI interframe delay
[us] = 6 * Tsck

#define SPICSBDELAY ((1000 / SPICLOCK) / 2) // SCK -> CSB delay [us]
= 0.5 * Tsck
```

Koodi 2. SPI-kellon ja viiveen määrittely.

Tärkeimmät määrittelyt tehtiin muuttujiin, jotka ilmaisivat anturien rekisterien paikat, joista luettiin anturien akselien arvot. Molemmille antureille tehtiin omat muuttujat ja määritelmät. Koodissa 3 SCA3100-kiihtyvyysanturin akselien rekisterien määritys.

```
#define SCA3100_X_LSB 0x04
#define SCA3100_X_MSB 0x05
#define SCA3100_Y_LSB 0x06
#define SCA3100_Y_MSB 0x07
#define SCA3100_Z_LSB 0x08
#define SCA3100_Z_MSB 0x09
```

Koodi 3. SCA3100-kiihtyvyysanturin akselien rekisterien määritys.

Anturien akselien arvojen muuttujat (koodi 4) nimettiin SCA3100_x_lsb sekä CMR3000_x_lsb, molempien antureiden kaikille akseleille omat lsb:t ja msb:t. Muuttujista tehtiin unsigned short-tyyppisiä muuttujia, joiden arvoalue oli 0...65535.

```
unsigned short SCA3100_x_msb;
unsigned short SCA3100_x_lsb;
short SCA3100_x_value;
unsigned short SCA3100_y_msb;
unsigned short SCA3100_y_lsb;
short SCA3100_y_value;
unsigned short SCA3100_z_msb;
unsigned short SCA3100_z_lsb;
short SCA3100_z_value;
```

Koodi 4. SCA3100-kiihtyvyysanturin akselien arvoja varten tehdyt muuttujat.

5.1.2 Ohjelman funktiot ja kulku

Funktiot sisältävät erilaisia lauseita, jotka määrittävät millaisia toimenpiteitä tietokone tekee. Jokainen C- ja C++-ohjelma alkaa main-funktiolla. Main-funktio sisältää kaikki funktiot, joita ohjelmassa käytetään.

Lähetyspuolen main-funktiossa (koodi 5) alustetaan anturit, mikrokontrolleri ja radio. Tämän jälkeen ohjelma menee funktioon linkTo(), jossa mikrokontrolleri jää odottamaan, että se saa yhteyden toiseen mikrokontrolleriin.

```
void main(void)
{
    Init_CMR3000_SCA3100();
    BSP_Init();                // laudan alustus
    SMPL_Init(sRxCallback);    // radion ja SimpliciTI protokollan alustus
    wait_ms(100);
    while(1)
    {
        linkTo();           // muodostetaan yhteys toisen laudan kanssa ja
                            //jäädään odottamaan CMR3000 anturin interruptia
    }
}
```

Koodi 5. Lähetyspuolen main-funktio.

Init_CMR3000_SCA3100-funktio alustaa anturit mikrokontrollerille. Funktio asettaa molempien anturien MOSI-, MISO-, SCK- sekä CSB-pinnit ja sekä tekee tarvittavat asetukset SPI-väylää varten.

LinkTo-funktiossa (koodi 6) tapahtuvat kaikki loput toimenpiteet. LinkTo-funktio odottaa, että se saa muodostettua radioyhteyden toisen mikrokontrollerin kanssa. Kun yhteys on muodostettu, laitetaan sen merkiksi vihreä ledi päälle. Tämän jälkeen ohjelma asettaa mikrokontrollerin radion vastaanoton päälle sekä resetoi CMR3000-kulmanopeusanturin ja asettaa sen mittaustilaan. SCA3100-kiihtyvyysanturi menee automaattisesti mittaustilaan, joten sitä ei tarvitse erikseen asettaa päälle. Kun mittaustila on laitettu päälle, asetetaan mikrokontrolleri virransäästötilaan, joka sallii yleiset keskeytykset. CMR3000-kulmanopeusanturi herättää mikrokontrollerin aina, kun sillä on tarjota uusia arvoja.

```
static void linkTo()
{
            //uint8_t len, tid;
            // odotetaan että linkki lautojen välille on muodostettu
            while (SMPL_SUCCESS != SMPL_Link(&sLinkID1))
            {
                        SPIN_ABOUT_A_SECOND;
            }
            BSP_TURN_ON_LED1();
            // radion asetukset, laitetaan vastaanotto päälle
            SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);
            // resetoi cmr3000 anturi
            CMR3000_Data = CMR3000_WriteRegister(CTRL, RESET);
            // odotetaan että resetointi valmis
            wait_ms(20);

/*laitetaan I2C pois päältä ja 80hz mittaus mode vielä kerran
varmuuden vuoksi päällle*/
            CMR3000_Data=CMR3000_WriteRegister(CTRL,I2C_DIS | MODE_80);
            wait_ms(20);

/* mennään virransäästötilaan, mutta sallii yleiset interruptit*/
            __bis_SR_register(LPM4_bits + GIE);

            while (1)
            // jäädään odottamaan CMR3000 anturin interruptia
            {

            }
}
```

Koodi 6. LinkTo-funktio.

Kun keskeytys uuden kulmanopeusarvon saatua tapahtuu, siirrytään keskeytys-funktioon (koodi 7). Funktion alussa laitetaan kulmnanopeusanturi mittautilaan uudelleen, CMR3000-kulmnanopeusanturi ei pysynyt jostain syystä mittaustilassa, joten se jouduttiin laittamaan uudelleen päälle.

```
#pragma vector=CMR3000_PORT_INT_VECTOR
__interrupt void Port_INT_ISR(void)
{
          /* laitetaan I2C pois päältä ja 80hz mittaustila päälle
          CMR3000_Data=CMR3000_WriteRegister(CTRL,I2C_DIS  |  MODE_80)
          /* kun uusi uuden arvot ovat valmiita, luetaan molempien
antureiden arvot*/
if (CMR3000_PORT_INT_IN & CMR3000_PIN_INT)
          {
          Read_CMR3000_SCA3100();  // antureiden arvojen luku
          }
}
```

Koodi 7. CMR3000-kulmanopeusanturin keskeytys-funktio.

Read_CMR3000_SCA31000-funktiolla (koodi 8) hoidetaan anturien arvojen lukeminen SPI-väylältä sekä arvojen lähettäminen radion kautta toiselle mikrokontrollerille. Molempien anturien jokaisen akselin lsb- ja msb-arvo luetaan CMR3000_ReadRegister- ja SCA31000_ReadRegister-funktioilla muuttujiin. Arvot lähetetään tämän jälkeen SMPL_Send-funktiolla toiselle mikrokontrollerille.

```
void Read_CMR3000_SCA3100()
{
          uint8_t  msg[12];
          BSP_TURN_ON_LED2(); // ledi päälle kun uusi tieto on valmis
/*  Luetaan  CMR3000,  jokaisesta  akselista  luetaan  LSB(vähiten
merkitsevät) ja MSB(eniten merkitsevät) arvot.  Molemmat LSB ja MSB
ovat 8bit kokoisia */
          CMR3000_x_lsb = CMR3000_ReadRegister(CMR3000_X_LSB);
          CMR3000_x_msb = CMR3000_ReadRegister(CMR3000_X_MSB);
          CMR3000_y_lsb = CMR3000_ReadRegister(CMR3000_Y_LSB);
          CMR3000_y_msb = CMR3000_ReadRegister(CMR3000_Y_MSB);
          CMR3000_z_lsb = CMR3000_ReadRegister(CMR3000_Z_LSB);
          CMR3000_z_msb = CMR3000_ReadRegister(CMR3000_Z_MSB);
/*  Luetaan  SCA3100,  jokaisesta  axelista  luetaan  LSB(vähiten
merkitsevät) ja MSB(eniten merkitsevät) arvot.*/
          SCA3100_x_lsb = SCA3100_ReadRegister(SCA3100_X_LSB);
          SCA3100_x_msb = SCA3100_ReadRegister(SCA3100_X_MSB);
          SCA3100_y_lsb = SCA3100_ReadRegister(SCA3100_Y_LSB);
          SCA3100_y_msb = SCA3100_ReadRegister(SCA3100_Y_MSB);
          SCA3100_z_lsb = SCA3100_ReadRegister(SCA3100_Z_LSB);
          SCA3100_z_msb = SCA3100_ReadRegister(SCA3100_Z_MSB);
          msg[0] = SCA3100_x_msb;  // tallennetaan arvot
          msg[1] = SCA3100_x_lsb;  // viestin muuttujiksi
          msg[2] = SCA3100_y_msb;
          msg[3] = SCA3100_y_lsb;
          msg[4] = SCA3100_z_msb;
          msg[5] = SCA3100_z_lsb;
          msg[6] = CMR3000_x_msb;
          msg[7] = CMR3000_x_lsb;
          msg[8] = CMR3000_y_msb;
          msg[9] = CMR3000_y_lsb;
          msg[10] = CMR3000_z_msb;
          msg[11] = CMR3000_z_lsb;
          /* Lähetetään arvot toiselle laudalle radion kautta*/
          SMPL_Send(sLinkID1, msg, sizeof(msg));
          BSP_TURN_OFF_LED2();
          wait_ms(15);
}
```

Koodi 8. Anturien lukufunktio Read_CMR3000_SCA3100.

Kun arvot on lähetetty radion kautta, ohjelma jää odottamaan seuraavia arvoja CMR3000-kulmanopeusanturilta ja toistaa äsken käydyt toimenpiteet. Ohjelma kokonaisuudessaan liitteessä 1. [10.

5.2 EM430F6137RF900 + CMR3000 + SCA3100, vastaanotto

Tässä kappaleessa käydään läpi vastaanottopuolen ohjelman määrittelyt, muuttujat ja tärkeimmät funktiot.

Vastaanottopuolen mikrokontrollerin tavoitteena oli vastaanottaa toisen mikrokontrollerin radion kautta lähettämät arvot ja lähettää ne edelleen sarjaväylän kautta tietokoneelle.

5.2.1 Määrittelyt ja muuttujat

Ohjelman alussa tehtiin samoja määrittelyjä kuin lähetyspuolen ohjelmassa, eli määriteltiin tarvittavat muuttujat viiveille. Muita muuttujia tarvittiin vain arvojen vastaanottamiseen, esim. tx_CMR3000_x_msb_value. Samalla muuttujalla anturien arvot vastaanotettiin sekä lähetettiin taas eteenpäin sarjaväylällä. Tämän jälkeen ohjelma siirtyi odottamaan uusia radioviestejä.

5.2.2 Ohjelman funktiot ja kulku

Vastaanottopuolen ohjelman main-funktio on hyvin samanlainen kuin lähetyspuolen main-funktio. Ainoana erona on se, että linkTo-funktion sijaan käytetään linkFrom-funktiota (koodi 9), jolla muodostetaan linkki mikrokontrollerien välille. Muodostettuaan linkin toisen mikrokontrollerin välille, ohjelma jää odottamaan sieltä radion kautta saapuvaa viestiä. Kun viesti saapuu, ohjelma tallettaa arvot muuttujiin ja lähettää ne saman tien sarjaväylää pitkin tietokoneelle. Sarjaväyläviestien loppuun lisättiin vielä kaksi ylimääräistä viestiä, joista tietokone pystyi tarkistamaan, että se lukee oikeat arvot. Ohjelma kokonaisuudessaan liitteessä 2.

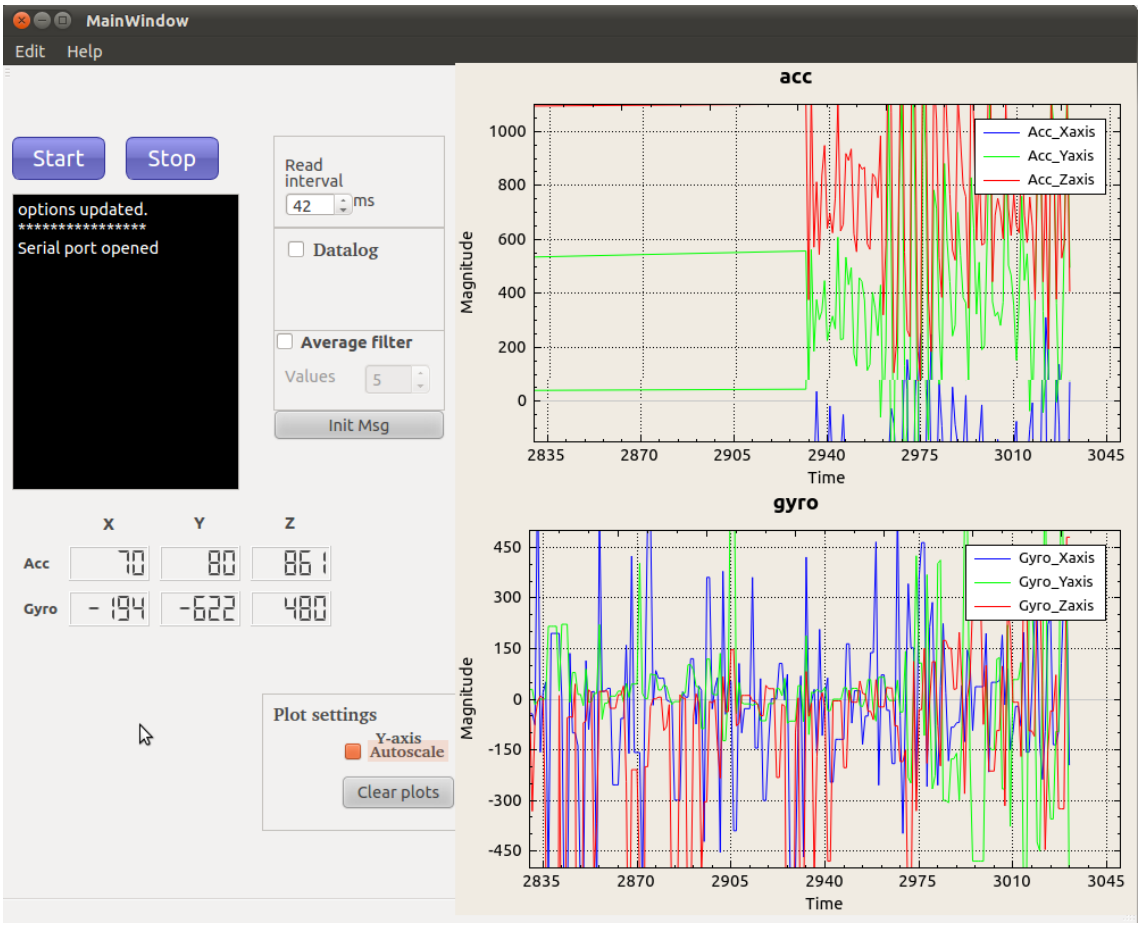
```
static void linkFrom()
{
/* Jäädään odottamaan, että linkki on muodostettu*/
  while (1)
  {
    if (SMPL_SUCCESS == SMPL_LinkListen(&sLinkID2))
    {
      BSP_TURN_ON_LED1();    // Kun linkki on muodostettu laitetaan
vihreä ledi päälle
      break;
    }
  }
   SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0); // radion
asetukset, laitetaan vastaanotto päälle
   while (1)
   {
    /* Odotetaan että viesti saapuu. vastaanotto säie antaa
interruptin */
           }
}
/* Saapuvan viestin käsittely. Kun uusi viesti on vastaanotettu,
 * laitetaan punainen ledi päälle. Kun viesti on lähetetty
 * eteenpäin sarjaväylälle, laitetaan punainen ledi pois päältä */
static uint8_t sRxCallback(linkID_t port)
{
  uint8_t msg[12], len;
  /* Tarkistetaan että viesti tulee halutusta laitteesta */
  if (port == sLinkID2)
  {
    /* Otetaan saapuva viesti vastaan */
     if ((SMPL_SUCCESS == SMPL_Receive(sLinkID2, msg, &len)) && len)
     {
        /* Luetaan saapuvat viestit*/
        BSP_TURN_ON_LED2();
        tx_SCA3100_x_msb_value = msg[0];
        tx_SCA3100_x_lsb_value = msg[1];
        tx_SCA3100_y_msb_value = msg[2];
        tx_SCA3100_y_lsb_value = msg[3];
        tx_SCA3100_z_msb_value = msg[4];
        tx_SCA3100_z_lsb_value = msg[5];
```

```
        tx_CMR3000_x_msb_value = msg[6];
        tx_CMR3000_x_lsb_value = msg[7];
        tx_CMR3000_y_msb_value = msg[8];
        tx_CMR3000_y_lsb_value = msg[9];
        tx_CMR3000_z_msb_value = msg[10];
        tx_CMR3000_z_lsb_value = msg[11];
        /* Lähetetään viestit sarjaväylälle */
        Serial_TX(tx_CMR3000_x_msb_value);
        Serial_TX(tx_CMR3000_x_lsb_value);
        Serial_TX(tx_CMR3000_y_msb_value);
        Serial_TX(tx_CMR3000_y_lsb_value);
        Serial_TX(tx_CMR3000_z_msb_value);
        Serial_TX(tx_CMR3000_z_lsb_value);
        Serial_TX(tx_SCA3100_x_msb_value);
        Serial_TX(tx_SCA3100_x_lsb_value);
        Serial_TX(tx_SCA3100_y_msb_value);
        Serial_TX(tx_SCA3100_y_lsb_value);
        Serial_TX(tx_SCA3100_z_msb_value);
        Serial_TX(tx_SCA3100_z_lsb_value);
        /* Qt. tehdylle ohjelmalle tarkistus bitit*/
        Serial_TX(0x37);
        Serial_TX(0x01);
        BSP_TURN_OFF_LED2();
        wait_ms(10);
        return 1;
     }
  }
  return 0;
}
```

Koodi 9. LinkFrom-funktio.

5.3 PC-ohjelma

Arvoja voitiin lukea tietokoneelta Pasi Yrjölän kehittelemällä Qt-ohjelmalla, joka oli suunniteltu tätä projektia varten, käsittelemään ja esittämään anturien arvoja. Ohjelman avulla voitiin helposti seurata anturien liikkeitä graafisesti sekä tallentaa saadut arvot tekstitiedostoihin. Kuvassa 15 on menossa testi, jossa antureita heilutetaan satunnaisesti nopeaan tahtiin. Kuvan vasemmassa laidassa on ilmoitettu anturien akselien arvot ja oikealla arvot on esitetty kuvaajissa. Kuvasta näkee, että kiihtyvyysanturissa on ollut jokin häiriö, sillä sen arvot kuvaajassa muuttuvat myöhemmin kuin kulmanopeusanturin. Tällaisiä ongelmia saattoi välillä ilmetä, ja ne johtuivat suurimmaksi osaksi kosketushäiriöistä.



Kuva 15. Qt-ohjelma anturien arvojen tarkasteluun ja tallentamiseen.

6 Tulokset

Mittausjärjestelmä saatiin valmiiksi ja tulokset olivat toivottuja. Mikrokontrollereille kehiteltiin ohjelmistot anturien lukua varten. Molemmat anturit saatiin toimimaan halutulla tavalla ja niistä saatiin luettua akselien arvoja SPI-väylää pitkin. Antureilta saatiin päivitettyä arvoja noin 17 kertaa sekunnissa ja mikrokontrollerien välimatkaksi todettiin noin 25 metriä ilman ongelmia.

Mittausjärjestelmän testaukseen ei käytetty paljon aikaa, eikä sitä saatu kiinnitettyä kunnon testipenkkiin. Testit toteutettiin liikuttelemalla anturipakettia kädellä. Muutaman viikon lisätyöllä mittausjärjestelmälle olisi voitu rakentaa kunnollinen testipenkki, mutta tällä kertaa tyydyttiin siihen, että saatiin anturit toimimaan.

7 Yhteenveto

Insinöörityön tuloksena saatiin toimiva langaton anturiverkko kahden mikrokontrollerin välille. Ohjelmistoissa on kuitenkin vielä paljon kehittämisen varaa, mutta niiden pohjalta on helppo jonkun jatkaa tarvittaessa kehitystä. Kehitys voisi liittyä siihen, kuinka mikrokontrollerit keskustelevat tietokoneen kanssa. Tässä työssä tietokoneen kanssa keskusteleva mikrokontrolleri vain lähettää tietoa, kun se voisi esimerkiksi saada tietokoneelta tiedon siitä, mitä antureita on käytössä. Tietokoneelle tehtyyn ohjelmaan lisättäisiin uusi välilehti, jossa voidaan määrittää käytössä olevat anturit, ja tieto valituista antureista päätyisi antureita ohjaavaan mikrokontrolleriin, joka tekisi tällöin tarvittavat toimenpiteet. Tällöin anturien vaihtaminen ja lisääminen helpottuisi, eikä mikrokontrollereihin tarvitsisi aina ladata erikseen ohjelmia riippuen siitä, mitä antureita on käytössä.

Mittausjärjestelmän kotelo, missä anturit sijaitsivat, olisi voitu suunnitella CAD-ohjelmalla, jotta siitä olisi saatu mahdollisimman tukeva ja mahdollisesti vesitiivis. Kotelo pitäsi suunnitella siten, että siinä olisi erittäin monipuoliset kiinnitysmahdollisuudet, jolloin sitä voitaisiin käyttää monissa eri tilanteissa. Tämänhetkinen kotelo on toimiva ja tukeva, mutta sen kiinnittäminen onnistuu vain esimerkiksi teipillä tai kiristysliinalla.

Työssä haastavin osuus oli selvittää, miten antureilta saatin halutut tiedot ulos SPI-väylää pitkin. SPI-väylä oli jo ennestään tuttu työn tekijälle, mutta sitä oli käytetty eri mikrokontrollerin kanssa. Suurin ongelma oli saada SPI:n ajoitus kohdalleen, eli säätää tarvittavat viiveet oikein. Työn alussa ongelmia oli myös SPI:n kytkennän osalta. SPI-väylä saatiin kuitenkin toimimaan pitkän väännön ansiosta, jonka jälkeen työ eteni vauhdilla loppua kohden.

Tästä työstä tehtiin laajat dokumentaatiot Metropolian wiki-sivuille, joiden avulla Murata Electronicsin anturien käyttöönotto SPI-väylää pitkin on hyvin helppoa.

Lähteet

1 TI MSP430 mikrokontrolleriperhe. Verkkojulkaisu, http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page?DCMP=MCU_other&HQS=msp430 Luettu 1.4.2013

2 CC430F6137 mikrokontrolleri. Verkkojulkaisu, http://www.ti.com/lit/ds/symlink/cc430f6137.pdf Luettu 1.4.2013

3 EM430F6137RF900-kehitysalusta. Verkkojulkaisu, http://www.ti.com/tool/em430f6137rf900#technicaldocuments Luettu 3.4.2013

4 SCA3100-D04-kiihtyvyysanturi. Verkkojulkaisu, http://www.muratamems.fi/sites/default/files/documents/sca3100-d04_accelerometer_datasheet_82_688_00_e_0.pdf Luettu 4.4.2013

5 CMR3000-D01-kulmanopeusanturi. Verkkojulkaisu, http://www.muratamems.fi/sites/default/files/documents/cmr3000_d01_datasheet_82106500.a02.pdf Luettu 5.4.2013

6 SCA8X0 / 21X0 / 31X0 Product Family Specification. Verkkojulkaisu, http://www.muratamems.fi/sites/default/files/documents/sca8x0_21x0_3100_product_family_specification_82_694_00f.pdf Luettu 5.4.2013

7 CMR3000-D0X Series Product Family Specification. Verkkojulkaisu, http://www.muratamems.fi/sites/default/files/documents/cmr3000-d0x_product_family_specification_82112900.a.03.pdf Luettu 6.4.2013

8 SPI. Verkkojulkaisu, http://cna.mikkeliamk.fi/Oppilas/mikro_ohjaimet_matskua/OSA_2/6.5.%20AVR_rauta.%20SPI-ohjelmointia.pdf Luettu 7.4.2013

9 SimpliciTI. Verkkojulkaisu, http://www.vscp.org/wiki/lib/exe/fetch.php/downloads/swru130_simplicity_principle.pdf Luettu 8.4.2013

10 Kernighan Brian W. 2004. Ohjelmointi. Helsinki. IT Press.

EM430F6137RF900+SCA3100+CMR3000, lähetyspuolen koodi

```
/*****************************************************************************
 * EM430F6137RF900 + CMR3000 + SCA3100 viestien lähetys.
 * Tämä puoli hoitaa viestien keräämisen antureilta ja lähettää arvot radion
 * kautta toiselle laudalle. Laudassa on kiinni anturit CMR300 ja SCA3100. Kun radio-
 * linkki on muodostettu, lautojen välille syttyy vihreä ledi. Tämän jälkeen ohjelma jää
 * odottamaan uutta tietoa antureilta. Anturien arvot luetaan aina kun CMR3000-anturi
 * antaa keskeytyksen (interrupt), kun tiedot ovat valmiit. Tämän jälkeen luetaan anturien
 * rekistereistä x, y ja z akselien arvot. Jokaisesta akselista luetaan LSB
 * (vähiten merkitsevät) ja MSB (eniten merkitsevät) arvot. Arvot lähetetään
 * radion kautta toiselle laudalle. Punainen ledi menee aina päälle kun uusi
 * arvo antureilta on valmiina ja menee pois päältä kun viesti on lähetetty
 * radion kautta eteenpäin.
 *
 * Anturien liitännät lautaan:
 *
 * Liitännät molemmissa antureissa:
 *
 * AVSS/DVSS    =          CON9        pin 1
 * AVDD/DVDD    =          CON9        pin 5
 * MISO         =          PORT1       pin 6
 * MOSI         =          PORT1       pin 7
 * SCK          =          PORT1       pin 8
 *
 * CMR3000:
 * CSB          =          PORT2       pin 8
 * INT          =          PORT2       pin 5
 *
 * SCA3100:
 * CSB          =          PORT2       pin 7
 */
// LIBRARIES
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "cc430x613x.h"
#include "bsp_leds.h"
// SPI delays
#define XTAL 16000000L
#define TICKSPERMS (XTAL / 1000 / 5 - 1)
#define TICKSPERUS (TICKSPERMS / 1000)
#define SPICLOCK 500 //SPI clock = 500kHz
#define SPIFRAMEDELAY ((1000 / SPICLOCK) * 6) // SPI interframe delay [us] = 6 * Tsck
#define SPICSBDELAY ((1000 / SPICLOCK) / 2) // SCK -> CSB delay [us] = 0.5 * Tsck
```

```
static void linkTo(void);
static uint8_t sRxCallback(linkID_t);
static linkID_t sLinkID1 = 0;
#define SPIN_ABOUT_A_SECOND  NWK_DELAY(1000)

// PORT DEFINITIONS
#define PORT_CSB_OUT P2OUT
#define PORT_CSB_DIR P2DIR

//CMR3000 DEFINE
#define CMR3000_PORT_INT_IN P2IN
#define CMR3000_PORT_INT_OUT P2OUT
#define CMR3000_PORT_INT_DIR P2DIR
#define CMR3000_PORT_INT_IE P2IE
#define CMR3000_PORT_INT_IES P2IES
#define CMR3000_PORT_INT_IFG P2IFG
#define CMR3000_PORT_INT_VECTOR PORT2_VECTOR
#define CMR3000_PIN_INT BIT4
//CMR300 AXIS REG
#define CMR3000_X_LSB 0x0C
#define CMR3000_X_MSB 0x0D
#define CMR3000_Y_LSB 0x0E
#define CMR3000_Y_MSB 0x0F
#define CMR3000_Z_LSB 0x010
#define CMR3000_Z_MSB 0x11
#define CMR3000_PIN_CSB BIT7

//CMR3000 muuttujat
unsigned short CMR3000_x_msb;
unsigned short CMR3000_x_lsb;
short CMR3000_x_value;
unsigned short CMR3000_y_msb;
unsigned short CMR3000_y_lsb;
short CMR3000_y_value;
unsigned short CMR3000_z_msb;
unsigned short CMR3000_z_lsb;
short CMR3000_z_value;
unsigned char CMR3000_Data;

#define CTRL 0x02
#define RESET 0x80 // Set device in reset
#define INT_LEVEL_LOW 0x40 // INT active high
#define I2C_DIS 0x10 // I2C disabled
#define MODE_80 0x06 // Measurement mode BW=80 Hz
```

```
// SCA3100 init
// SCA3100 AXIS REG
#define SCA3100_X_LSB 0x04
#define SCA3100_X_MSB 0x05
#define SCA3100_Y_LSB 0x06
#define SCA3100_Y_MSB 0x07
#define SCA3100_Z_LSB 0x08
#define SCA3100_Z_MSB 0x09
#define SCA3100_PIN_CSB BIT6

//SCA3100 muuttujat
unsigned short SCA3100_x_msb;
unsigned short SCA3100_x_lsb;
short SCA3100_x_value;
unsigned short SCA3100_y_msb;
unsigned short SCA3100_y_lsb;
short SCA3100_y_value;
unsigned short SCA3100_z_msb;
unsigned short SCA3100_z_lsb;
short SCA3100_z_value;

//SPI init
#define TX_BUFFER UCA0TXBUF
#define RX_BUFFER UCA0RXBUF
#define SPI_CTL0 UCB0CTL0
#define SPI_CTL1 UCB0CTL1
#define SPI_BR0 UCB0BR0
#define SPI_BR1 UCB0BR1

//CMR3000 FUNCTION PROTOTYPES
unsigned char CMR3000_ReadRegister(unsigned char Address);
unsigned char CMR3000_WriteRegister(unsigned char Address, unsigned char Data);
//SCA3100 FUNKTION PROTOTYPES
unsigned char SCA3100_ReadRegister(unsigned char Address);
//COMMON FUNKTION PROTOTYPES
unsigned short realValues(unsigned short values);
void wait_ms(unsigned short ms);
void wait_us(unsigned short us);

void Init_CMR3000_SCA3100();
void Read_CMR3000_SCA3100();

/* Tehdään tarvittavat alustukset ja muodostetaan linkki toiseen lautaan*/
 void main(void)
{
                        Init_CMR3000_SCA3100();
                        // alustaa SPI:n antureille
                        BSP_Init();
                                                        // laudan alustus
```

```
                SMPL_Init(sRxCallback);
                // radion ja SimpliciTI protokollan alustus
                wait_ms(100);
                while(1)
                {
                /* muodostetaan yhteys toisen laudan kanssa ja jäädään odottamaan CMR3000 anturin interruptia*/
                                linkTo();
                }
}
/* Antureiden alustus */
void Init_CMR3000_SCA3100()
{
                WDTCTL = WDTPW+WDTHOLD;     // sammuta watchdog timer
                UCSCTL5 |= DIVS_3;

                PMAPPWD = 0x02D52;        // hanki kirjoitusoikeudet porttitaulukon registereihin
                P1MAP6 = PM_UCA0SIMO;             // UCA0SIMO lähtö P1.6 (portti 1, pinni 6)
                P1MAP5 = PM_UCA0SOMI;             // UCA0SOMI lähtö P1.5 (portti 1, pinni 5)
                P1MAP7 = PM_UCA0CLK;              // UCA0CLK lähtö P1.7 (portti 1, pinni 7)

                P2REN |= BIT4;                    // ota käyttöön sisäinen vastus
                P2OUT |= BIT4;                    // Set P1.4 as pull-Up resistance
                P2IE |= BIT4;                     // interrupt päällä P1.4
                P2IES |= BIT4;                    // P1.4 Hi/Lo edge
                P2IFG &= ~BIT4;                   // P1.4 IFG cleared

                PORT_CSB_DIR |= CMR3000_PIN_CSB;
                PORT_CSB_OUT |= CMR3000_PIN_CSB;      // otetaan CMR3000 pois käytöstä
                PORT_CSB_DIR |= SCA3100_PIN_CSB;
                PORT_CSB_OUT |= SCA3100_PIN_CSB;       // otetaan SCA3100 pois käytöstä

                P1DIR |= BIT5 + BIT6 + BIT7;          // asettaa portin 1, pinnit 6, 7 ja 8 lähdöiksi
                P1SEL |= BIT5 + BIT6 + BIT7;                    // avaa portin 1, pinnit 6, 7 ja 8

                UCA0CTL1 |= UCSWRST;                            // Resetoi USCI moduulin
                UCA0CTL0 |= UCMST+UCSYNC+UCCKPH+UCMSB; // valitaan: UCMST =                masterin
                valinta,
// UCSYNC = valitaan SPI mode,
                                                                // UCCKPH = kellon vaihe 0x80,
//UCMSB = MSB ensin

                UCA0MCTL = 0;                     // ei modulointia

                UCA0CTL1 |= UCSSEL1;                            // SMCLK, USCI 0 Clock Source Select 1
                SPI_BR0 = 0x04;
                                // Low byte of division factor for baud rate (500kHz)
                SPI_BR1 = 0x00;
                                // High byte of division factor for baud rate
```

```
                UCA0CTL1 &= ~UCSWRST;                   // alustetaan USCI



}

/* Linkin muodostin. Tässä laitetaan samalla CMR3000 anturi 80Hz modeen*/
static void linkTo()
{
                //uint8_t len, tid;
                while (SMPL_SUCCESS != SMPL_Link(&sLinkID1))
                // odotetaan että linkki lautojen välille on muodostettu
                {
                                SPIN_ABOUT_A_SECOND;
                }
                BSP_TURN_ON_LED1();
                SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);  // radion asetukset,
laitetaan vastaanotto päälle
                CMR3000_Data = CMR3000_WriteRegister(CTRL, RESET);          // resetoi cmr3000 anturi
                wait_ms(20);                                    // odotetaan että resetointi valmis

/*laitetaan I2C pois päältä ja 80hz mittaus mode vielä kerran varmuuden vuoksi päällle*/
                CMR3000_Data = CMR3000_WriteRegister(CTRL, I2C_DIS | MODE_80);
                wait_ms(20);

/* mennään virransäästötilaan, mutta sallii yleiset interruptit*/
                __bis_SR_register(LPM4_bits + GIE);

                while (1)
                // jäädään odottamaan CMR3000 anturin interruptia
                {

                }
}


static uint8_t sRxCallback(linkID_t port)
{
  uint8_t msg[3], len;

  // tarkistetaan otetaan vastaan oikealta laitteelta
  if (port == sLinkID1)
  {
                // vastaanotetaan viesti
    if ((SMPL_SUCCESS == SMPL_Receive(sLinkID1, msg, &len))) //
    {
/*tähän voidaan laittaa jotakin mitä halutaan tehdä kun viesti saapui takaisin toiselta laudalta, jos on tarvetta*/
      return 1;
```

```
    }
  }
  return 0;
}
/* CMR3000 anturin keskeytys funktio. */
#pragma vector=CMR3000_PORT_INT_VECTOR
__interrupt void Port_INT_ISR(void)
{
                /* laitetaan I2C pois päältä ja 80hz mittaus mode vielä kerran varmuuden vuoksi päällle*/
                CMR3000_Data = CMR3000_WriteRegister(CTRL, I2C_DIS | MODE_80);

                /* kun uusi uuden arvot ovat valmiita, luetaan molempien antureiden arvot*/
if (CMR3000_PORT_INT_IN & CMR3000_PIN_INT)
                {
                                Read_CMR3000_SCA3100();
                                                                                // antureiden arvojen luku
                }
}

/* CMR3000 anturin rekistereiden luku*/
unsigned char CMR3000_ReadRegister(unsigned char Address)
{
                unsigned char Result;
                Address <<= 2;                                  // shiftataan osoitetta vasemmalle kahdella
                PORT_CSB_OUT &= ~CMR3000_PIN_CSB;               // valitaan CMR3000 anturi
                Result = RX_BUFFER;                             // pyyhitään interrupt lippu lukemalla RX bufferi
                TX_BUFFER = Address;                            // kirjoitetaan osoite TX bufferiin
                while (!(UCA0IFG&UCRXIFG));                     // odotetaan että uusi tieto on kirjoitettu RX bufferiin
                Result = RX_BUFFER;                             // pyyhitään interrupt lippu lukemalla RX bufferi
                TX_BUFFER = 0;                                  // kirjoitetaan TX bufferiin arvo 0
                while (!(UCA0IFG&UCRXIFG));                     // odotetaan että uusi tieto on kirjoitettu RX bufferiin
                Result = RX_BUFFER;                             // luetaan RX bufferi
                wait_us(SPICSBDELAY);
                PORT_CSB_OUT |= CMR3000_PIN_CSB;                // otetaan CMR3000 pois käytöstä
                wait_us(SPIFRAMEDELAY);
                return Result;                                  // palautetaan RX bufferista saatu data
}

/* CMR3000 anturin rekistereihin kirjoitus*/
unsigned char CMR3000_WriteRegister(unsigned char Address, unsigned char Data)
{
                unsigned char Result;
                Address <<= 2;                  // shiftataan osoitetta vasemmalle kahdella
                Address |= 0x02;                // asetetaan toinen bitti ykköseksi, tekee siitä silloin kirjoittavan
                PORT_CSB_OUT &= ~CMR3000_PIN_CSB;               // valitaan CMR3000 anturi
                Result = RX_BUFFER;             // pyyhitään interrupt lippu lukemalla RX bufferi
                TX_BUFFER = Address;            // kirjoitetaan osoite TX bufferiin
                while (!(UCA0IFG&UCRXIFG));     // odotetaan että uusi tieto on kirjoitettu RX bufferiin
                Result = RX_BUFFER;             // pyyhitään interrupt lippu lukemalla RX bufferi
```

```
                TX_BUFFER = Data;               // kirjoitetaan haluttu tieto TX bufferiin
                while (!(UCA0IFG&UCRXIFG));     // pyyhitään interrupt lippu lukemalla RX bufferi
                Result = RX_BUFFER;             // luetaan RX bufferi
                wait_us(SPICSBDELAY);
                PORT_CSB_OUT |= CMR3000_PIN_CSB;            // otetaan CMR3000 pois käytöstä
                wait_us(SPIFRAMEDELAY);
                return Result;
}


/* SCA3100 anturin rekistereiden luku*/
unsigned char SCA3100_ReadRegister(unsigned char Address)
{
                unsigned char Result;
                Address <<= 2;                  // shiftataan osoitetta vasemmalle kahdella
                PORT_CSB_OUT &= ~SCA3100_PIN_CSB;   // valitaan SCA3100 anturi
                Result = RX_BUFFER;             // pyyhitään interrupt lippu lukemalla RX bufferi
                TX_BUFFER = Address;            // kirjoitetaan osoite TX bufferiin r
                while (!(UCA0IFG&UCRXIFG));     // odotetaan että uusi tieto on kirjoitettu RX bufferiin
                Result = RX_BUFFER;             // pyyhitään interrupt lippu lukemalla RX bufferi
                TX_BUFFER = 0;                  // kirjoitetaan TX bufferiin arvo 0
                while (!(UCA0IFG&UCRXIFG));     // odotetaan että uusi tieto on kirjoitettu RX bufferiin
                Result = RX_BUFFER;             // luetaan RX bufferi
                wait_us(SPICSBDELAY);
                PORT_CSB_OUT |= SCA3100_PIN_CSB;    // otetaan SCA3100 pois käytöstä
                wait_us(SPIFRAMEDELAY);
                return Result;                  // palautetaan RX bufferista saatu data
}



/* Lukufunktio antureiden rekistereille
  Tähän tullaan aina kun CMR3000-anturi lähettää interruptin
  SCA3100:ssa ei ole interrupteja vaan dataa voidaan lukea koko ajan */
void Read_CMR3000_SCA3100()
{
                uint8_t  msg[12];
                BSP_TURN_ON_LED2(); // ledi päälle kun uusi tieto on valmis
/* Luetaan CMR3000, jokaisesta akselista luetaan LSB(vähiten merkitsevät) ja MSB(eniten merkitsevät) arvot. Molemmat LSB ja MSB ovat 8bit kokoisia */
                CMR3000_x_lsb = CMR3000_ReadRegister(CMR3000_X_LSB);
                CMR3000_x_msb = CMR3000_ReadRegister(CMR3000_X_MSB);
                CMR3000_y_lsb = CMR3000_ReadRegister(CMR3000_Y_LSB);
                CMR3000_y_msb = CMR3000_ReadRegister(CMR3000_Y_MSB);
                CMR3000_z_lsb = CMR3000_ReadRegister(CMR3000_Z_LSB);
                CMR3000_z_msb = CMR3000_ReadRegister(CMR3000_Z_MSB);

/* Luetaan SCA3100, jokaisesta axelista luetaan LSB(vähiten merkitsevät) ja MSB(eniten merkitsevät) arvot. Molemmat LSB ja MSB ovat 8bit kokoisia*/
```

```
                SCA3100_x_lsb = SCA3100_ReadRegister(SCA3100_X_LSB);
                SCA3100_x_msb = SCA3100_ReadRegister(SCA3100_X_MSB);
                SCA3100_y_lsb = SCA3100_ReadRegister(SCA3100_Y_LSB);
                SCA3100_y_msb = SCA3100_ReadRegister(SCA3100_Y_MSB);
                SCA3100_z_lsb = SCA3100_ReadRegister(SCA3100_Z_LSB);
                SCA3100_z_msb = SCA3100_ReadRegister(SCA3100_Z_MSB);

                // tallennetaan arvot viestin muuttujiksi,
                msg[0] = SCA3100_x_msb;
                msg[1] = SCA3100_x_lsb;
                msg[2] = SCA3100_y_msb;
                msg[3] = SCA3100_y_lsb;
                msg[4] = SCA3100_z_msb;
                msg[5] = SCA3100_z_lsb;
                msg[6] = CMR3000_x_msb;
                msg[7] = CMR3000_x_lsb;
                msg[8] = CMR3000_y_msb;
                msg[9] = CMR3000_y_lsb;
                msg[10] = CMR3000_z_msb;
                msg[11] = CMR3000_z_lsb;
                /* Lähetetään arvot toiselle laudalle radion kautta*/
                SMPL_Send(sLinkID1, msg, sizeof(msg));

                BSP_TURN_OFF_LED2();
                wait_ms(15);
}


void wait_ms(unsigned short ms)
{
                unsigned short a, b;
                for (a = ms; a > 0; a--)
                for (b = TICKSPERMS; b > 0; b--)
                asm(" nop");
}

void wait_us(unsigned short us)
{
                unsigned short a;
                us *= TICKSPERUS;
                for (a = us; a > 0; a--)
                asm(" nop");
}
```

EM430F6137RF900+SCA3100+CMR3000, vastaanottopuolen koodi

```
/*****************************************************************************
 * EM430F6137FR900 + CMR3000 + SCA3100 viestien vastaanotto.
 * Tämä puoli hoitaa viestien vastaanoton toiselta laudalta, jossa on
 * kiinni CMR3100 ja SCA3100 anturit. Muodostetaan yhteys kahden
 * EM430F6137FR900 kehityslaudan välille. Kun linkki on muodostettu,
 * syttyy laudassa vihreä ledi. Ohjelma jää odottamaan uuden viestin
 * saapumista toiselta laudalta. Aina kun uusi viesti saapuu, laitetaan
 * punainen ledi päälle. Kun viesti on käsitelty ja lähetetty eteenpäin
 * sarjaväylälle, sammutetaan punainen ledi. Arvot lähetetään sarjaväylälle
 * 8 bittisenä tavuna. Sarjaväylälle lähtee yhteensä 14 tavua. 12 tavua arvoja
 * ja 2 tarkistus tavua lopussa, jotta Qt tietää mistä kohtaa lukea arvoja.
 *
 *
 *   Sarjaväylän siirtonopeus = 9600
 *
 *               CC430F6137
 *                 -----------------
 *         /|\|                                   |
 *          | |                                   |
 *          --|RST                                |
 *            |                                   |
 *            |                   P2.7/UCA0TXD |------------>
 *            |                                  | 9600 - 8N1
 *            |                   P2.6/UCA0RXD |<------------
 *
 * */

#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "math.h"
static void linkFrom(void);

void toggleLED(uint8_t);

static          linkID_t sLinkID2 = 0;
static volatile uint8_t  sSemaphore = 0;
unsigned short realValues(unsigned short values);
void wait_ms(unsigned short ms);
void wait_us(unsigned short us);
void Serial_TX(unsigned int value);
```

```
void Init();
#define XTAL 16000000L
#define TICKSPERMS (XTAL / 1000 / 5 - 1)
#define TICKSPERUS (TICKSPERMS / 1000)
/* Rx callback handler */
static uint8_t sRxCallback(linkID_t);

char tx_CMR3000_x_msb_value;
char tx_CMR3000_x_lsb_value;
char tx_CMR3000_y_msb_value;
char tx_CMR3000_y_lsb_value;
char tx_CMR3000_z_msb_value;
char tx_CMR3000_z_lsb_value;
char tx_SCA3100_x_msb_value;
char tx_SCA3100_x_lsb_value;
char tx_SCA3100_y_msb_value;
char tx_SCA3100_y_lsb_value;
char tx_SCA3100_z_msb_value;
char tx_SCA3100_z_lsb_value;
void main (void)
{
  BSP_Init();                                                    // Laudan alustus
  SMPL_Init(sRxCallback);                                        // Radion ja SimpliTi alustus
  Init();

  linkFrom();                                                    // Linkin muodostaminen
  while (1) ;
}
/* Muodostetaan linkki "lähettäjä" laitteen kanssa, kun linkki on muodostettu laitetaan vihreä ledi päälle.  */
static void linkFrom()
{
/* Jäädään odottamaan, että linkki on muodostettu*/
  while (1)
  {
    if (SMPL_SUCCESS == SMPL_LinkListen(&sLinkID2))
    {
      BSP_TURN_ON_LED1();                                                                    //      Kun
linkki on muodostettu laitetaan vihreä ledi päälle
      break;
    }
  }
   SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0); // radion asetukset, laitetaan vastaanotto päälle
   while (1)
   {
    /* Odotetaan että viesti saapuu. vastaanotto säie antaa interruptin */

                        }
}
```

```
/* Saapuvan viestin käsittely. Kun uusi viesti on vastaanotettu,
 * laitetaan punainen ledi päälle. Kun viesti on lähetetty
 * eteenpäin sarjaväylälle, laitetaan punainen ledi pois päältä */
static uint8_t sRxCallback(linkID_t port)
{
  uint8_t msg[12], len;

  /* Tarkistetaan että viesti tulee halutusta laitteesta */
  if (port == sLinkID2)
  {
    /* Otetaan saapuva viesti vastaan */
    if ((SMPL_SUCCESS == SMPL_Receive(sLinkID2, msg, &len)) && len)
    {
      /* Luetaan saapuvat viestit*/
      BSP_TURN_ON_LED2();
      tx_SCA3100_x_msb_value = msg[0];
      tx_SCA3100_x_lsb_value = msg[1];
      tx_SCA3100_y_msb_value = msg[2];
      tx_SCA3100_y_lsb_value = msg[3];
      tx_SCA3100_z_msb_value = msg[4];
      tx_SCA3100_z_lsb_value = msg[5];
      tx_CMR3000_x_msb_value = msg[6];
      tx_CMR3000_x_lsb_value = msg[7];
      tx_CMR3000_y_msb_value = msg[8];
      tx_CMR3000_y_lsb_value = msg[9];
      tx_CMR3000_z_msb_value = msg[10];
      tx_CMR3000_z_lsb_value = msg[11];

      /* Lähetetään viestit sarjaväylälle */
      Serial_TX(tx_CMR3000_x_msb_value);
      Serial_TX(tx_CMR3000_x_lsb_value);
      Serial_TX(tx_CMR3000_y_msb_value);
      Serial_TX(tx_CMR3000_y_lsb_value);
      Serial_TX(tx_CMR3000_z_msb_value);
      Serial_TX(tx_CMR3000_z_lsb_value);
      Serial_TX(tx_SCA3100_x_msb_value);
      Serial_TX(tx_SCA3100_x_lsb_value);
      Serial_TX(tx_SCA3100_y_msb_value);
      Serial_TX(tx_SCA3100_y_lsb_value);
      Serial_TX(tx_SCA3100_z_msb_value);
      Serial_TX(tx_SCA3100_z_lsb_value);

      /* Qt. tehdylle ohjelmalle tarkistus bitit*/
      Serial_TX(0x37);
      Serial_TX(0x01);

      BSP_TURN_OFF_LED2();
      wait_ms(10);
```

```
      return 1;
    }
  }
  return 0;
}
// Wait ms
void wait_ms(unsigned short ms)
{
 unsigned short a, b;
 for (a = ms; a > 0; a--)                    // outer loop takes 5 ck per round
 for (b = TICKSPERMS; b > 0; b--)             // inner loop takes 5 ck per round
 asm(" nop");
}
// Wait us
void wait_us(unsigned short us)
{
 unsigned short a;
 us *= TICKSPERUS;
 for (a = us; a > 0; a--)                     // loop takes 5 ck per round
 asm(" nop");
}

// Sarjaporttiin lähetys
void Serial_TX(unsigned int value)
{
_delay_cycles(12000);                        // lisätään pieni väli lähetyksien välille
      UCA0TXBUF = value;                     // byten lähetys
}

void Init()
{
WDTCTL = WDTPW+WDTHOLD;                      // Stop watchdog timer

 P5SEL |= 0x03;                              // Enable XT1 pins

 do
 {
   UCSCTL7 &= ~(XT1LFOFFG + DCOFFG);
                                             // Clear XT2,XT1,DCO fault flags
   SFRIFG1 &= ~OFIFG;                        // Clear fault flags
   __delay_cycles(100000);                   // Delay for Osc to stabilize
 }while (SFRIFG1&OFIFG);                     // Test oscillator fault flag

 P1OUT = 0x000;                  // P1.0/1 setup for LED output
 P1DIR |= BIT0+BIT1;                         //

 PMAPPWD = 0x02D52;                           // Get write-access to port mapping regs
 P1MAP5 = PM_UCA0RXD;                        // Map UCA0RXD output to P2.6
 P1MAP6 = PM_UCA0TXD;                        // Map UCA0TXD output to P2.7
```

```
  PMAPPWD = 0;                    // Lock port mapping registers

  P1DIR |= BIT6;                                  // Set P2.7 as TX output
  P1SEL |= BIT5 + BIT6;                           // Select P2.6 & P2.7 to UART function

  UCA0CTL1 |= UCSWRST;                            // **Put state machine in reset**
  UCA0CTL1 |= UCSSEL_1;                           // CLK = ACLK
  UCA0BR0 = 0x03;                 // 32k/9600 - 3.41
  UCA0BR1 = 0x00;
  UCA0MCTL = 0x06;                                // Modulation
  UCA0CTL1 &= ~UCSWRST;                           // **Initialize USCI state machine**
}
```