LANGATTOMAT PERÄVAUNUN TAKAVALOT

Hinausautokäyttöön

Timo Pekkanen

Opinnäytetyö
Joulukuu 2013

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Tekijä(t)	Julkaisun laji	Päivämäärä
PEKKANEN, Timo	Opinnäytetyö	9.12.2013
	Sivumäärä 69	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
Langattomat Perävaunun Takavalot (Hinausautokäyttöön)

Koulutusohjelma
Ohjelmistotekniikan Koulutusohjelma

Työn ohjaaja(t)
VÄÄNÄNEN, Olli
KOTKANSALO, Jouko

Toimeksiantaja(t)
PEKKANEN, Timo

Tiivistelmä
Opinnäytetyössä suunnitellaan ja rakennetaan langattomat valot hinausautokäyttöön. Pimeällä kun hinataan 2 autoa, niin toinen niistä nostetaan eturenkaista ilmaan. Tällöin hinausauton omat takavalot peittyvät ja hinattavaan ajoneuvoon tarvitaan valot. Työssä kuvataan laitteen spesifikaatiot, suunnittelutyökalut, materiaalihankinnat ja ohjelmiston suunnittelu. Tavoitteena oli tehdä valmis laite, joka toimii sille asetettujen rajojen sisällä. Laite koostuu lähettimestä ja vastaanottimesta ja vastaanottimeen liitetyistä LED-valoista. Laite toimii 12 V jännitteellä. Lähetin ja vastaanotin sisältävät molemmat Atmelin prosessorin sekä Microchipin radiopiirin. Laite toimii 433MHz:n vapailla taajuuksilla. Laite suunnitellaan KiCad-ohjelmistolla, jolla piirretään piirikaavio sekä suunnitellaan piirilevy. Levystä tehdään Gerber-tiedostot ja levyt tilataan kiinalaisesta iTead-studio-nimisestä verkkokaupasta. Osatilaukset tehdään Farnellilta. Valmiit levyt kalustetaan sekä sille suunnitellaan ohjelmisto Atmel Studio-ohjelmistolla.

Laitteen kalustamis- ja ohjelmointivaiheessa ensimmäisessä protoversiossa huomataan muutamia virheitä. Nämä virheet eivät kuitenkaan estä laitteen toimintakuntoon laittamista. Laite toimii ja signaalin kantomatka on tällä hetkellä n. 15-20 metriä avoimessa maastossa. Laitteen sietokykyä erilaisille häiriöille ei tässä työssä ole testattu.

Työssä esitetään korjauskohteet laitteen jatkokehitystä varten. Piirilevy pitää uusia seuraavaa versiota varten. Laitteen sovelluskohteet ovat langattomat kytkinsovellukset niin ajoneuvokäytössä kuin kodin elektroniikassakin. Laite on muokattavissa useille eri käyttöjännitteille ja taajuusalueille.

Avainsanat (asiasanat)
Langaton, Elektroniikka, Radiotekniikka, Sulautettu Elektroniikka, MRF49XA

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Pekkanen, Timo	Bachelor´s Thesis	09122013
	Pages 69	Language Finnish
		Permission for web publication (X)

Title
Wireless Tail Light (For tow truck)

Degree Programme
Degree Programme in Software Engineering

Tutor(s)
VÄÄNÄNEN, Olli
KOTKANSALO, Jouko

Assigned by

Pekkanen, Timo

Abstract

This bachelor's thesis represents the building and programming of wireless tail lights to be used with tow trucks. The problem in Finland is that when tow truck is towing two vehicles at once, the rear vehicle is lifted from the front wheels. The rear vehicle then blocks the tow truck's rear lights. Finnish road law states that rear lights must be visible at all times.

This bachelor's thesis describes the designing, schematics and PCB planning, bill of materials and the final building process of the device. Additionally, the programming part is included and the basic programming process is described. This device consists of three parts: transmitter, receiver and LED-lights and operates at 12 V DC. The device can be altered to function on many different voltages, depending on the operating environment. The device uses 433 MHz free channels and has a range of approximately 15 to 20 meters in free air. Both receiver and transmitter have Atmel processor and Microchip radio chip. The schematics and PCB design have been made with KiCad, which is a freeware program. The software has been designed with Atmel Studio.

There were some bugs with the 1st proto version and fixes for those bugs have been presented in this thesis. EMC testing was not conducted for the 1st proto. The device can be altered for home wireless use, e.g., switches, dimming, remote control. All the information needed to design and build the device is included in this thesis.

Keywords

Wireless, radio, MRF49XA, Atmel, Electronics, Embedded devices

Miscellaneous

SISÄLTÖ

Kuviot

1. TYÖN LÄHTÖKOHDAT

1.1. Ongelman kuvaus

Hinausautoilla tulee usein vastaan tilanne, jolloin täytyy kuljettaa kerralla useampia kuin yksi ajoneuvo. Useinkaan ei ole käytössä autoa, johon molemmat autot saisi lavan päälle. Tällöin toinen hinattavista ajoneuvoista otetaan kyytiin nostamalla joko ajoneuvon eturenkaat tai ajoneuvon takarenkaat ns. ”grillin” päälle. Kuviossa 1 on kuva ”grillistä”. Kyseessä on ritilä, jonka koloihin hinattavan ajoneuvon eturenkaat (tai takarenkaat) laitetaan. Kuviossa 2 näkyy ajoneuvo lastattuna ”grillin” päälle. Usein kyytiin otettavat ajoneuvot ovat vaurioituneena eri tavoin, ja monesti tilanne

on sellainen, että ajoneuvon omat valaisimet eivät toimi. Vikana voi olla kolarin aiheuttamat vauriot, pelastuslaitoksen vahingontorjuntatoimenpiteet (akunnavat irrotettu) tai muu vastaava vaurio. Kesäisin ja valoisaan aikaan tämä ei aiheuta mitään ongelmaa, mutta tilanne muuttuu syksyllä, talvella sekä keväällä. Silloin tieliikennelaki velvoittaa hinausauton kuljettajan merkitsemään hinattavan ajoneuvon siten, että sen ääripisteet pystytään havaitsemaan takana tulevasta autosta luotettavasti. Ongelman ratkaisuksi on olemassa markkinoilla takavalot, jotka asennetaan hinattavan ajoneuvon taakse, ja siitä vedetään johdot hinausautoon, josta tulee käyttösähkö sekä signaalit. Tämän kyseisen ratkaisun ongelmakohta on se, että usein hinattavat ajoneuvot ovat pituudeltaan erilaisia, jolloin johto joko ei riitä tai sitä on liikaa. Samoin johdon kiinnittäminen hinattavaan ajoneuvoon on ongelmallista. Jos auto on muuten ehjä, mutta siinä ei ole sähköä, silloin hinattavaan ajoneuvoon ei saa tulla mitään vaurioita. Ratkaisuna ongelmaan on se, että tehdään hinattavaan ajoneuvoon liitettävät valot langattomiksi.

Kuvio 1. "Grilli"

Kuvio 2. Hinausauto lastattuna.

1.2. Laitteen toiminnan kuvaus

Suunniteltavana oli laite, johon kuuluvat hinausautoon liitettävä lähetin sekä hinattavaan ajoneuvoon liitettävä vastaanotin. Vastaanottimeen kuuluvat myös virtalähde sekä LED-valot. Kun hinausautosta laitetaan vilkku päälle tai painetaan jarrua, siitä

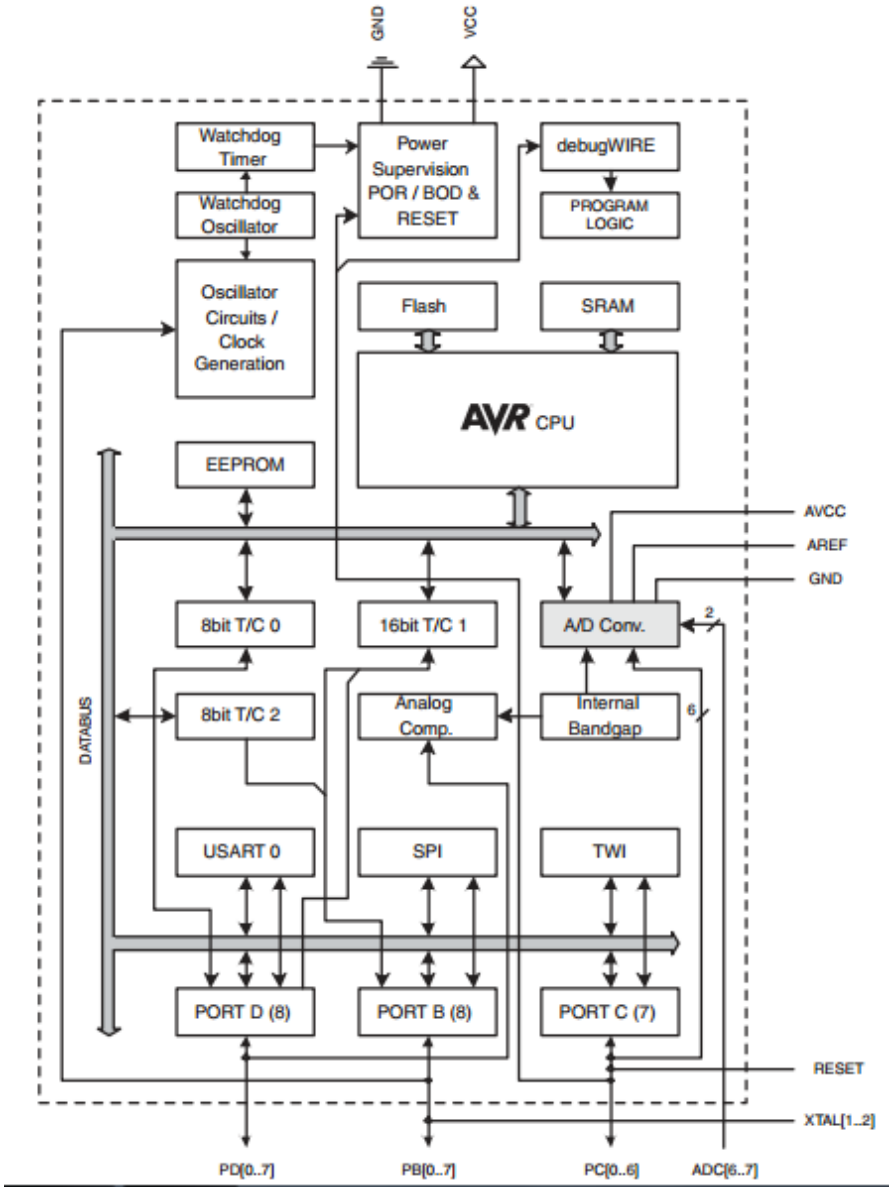
lähtee signaali prosessorille, joka prosessoi tiedon ja lähettää sen edelleen lähetin-vastaanotin yksikköön. Se lähettää tiedon vastaanottimelle, jossa vastaava lähetin-vastaanotinyksikkö vastaanottaa datan ja lähettää sen edelleen prosessorille. Prosessori sitten käsittelee tiedon ja sen pohjalta sytyttää FET-kytkennän kautta oikean LED-valon. Alkuunsa tieto tulee olemaan yksisuuntaista, eli dataa kulkee vain lähetin-yksiköltä vastaanottimelle ja mitään kuittausta ei tule. Mahdollisesti tulevat kenttätestit osoittavat, tarvitaanko kuittausta. Vastaanotin tullaan liittämään hinattavaan ajoneuvoon joko magneeteilla tai vaihtoehtoisesti imukupeilla.

2. ATMEL megaAVR-SARJA

megaAVR-prosessoriperheeseen kuuluu useita erikokoisia prosessoreita. megaAVR-prosessorit ovat 8-bittisiä RISC teknologiaan perustuvia prosessoreita. Pienimmät prosessorit sisältävät 4 kilotavua flash-muistia ja ovat 28-32-jalkaisia piirejä. Suurimmat puolestaan sisältävät jopa 256 kilotavua flash-muistia. Eri prosessoreissa on erilainen kokoelma liitäntöjä, kuten USB, LCD-kontrollerit, CAN, LIN sekä tehotasokontrollerit.

3. ATmega328A

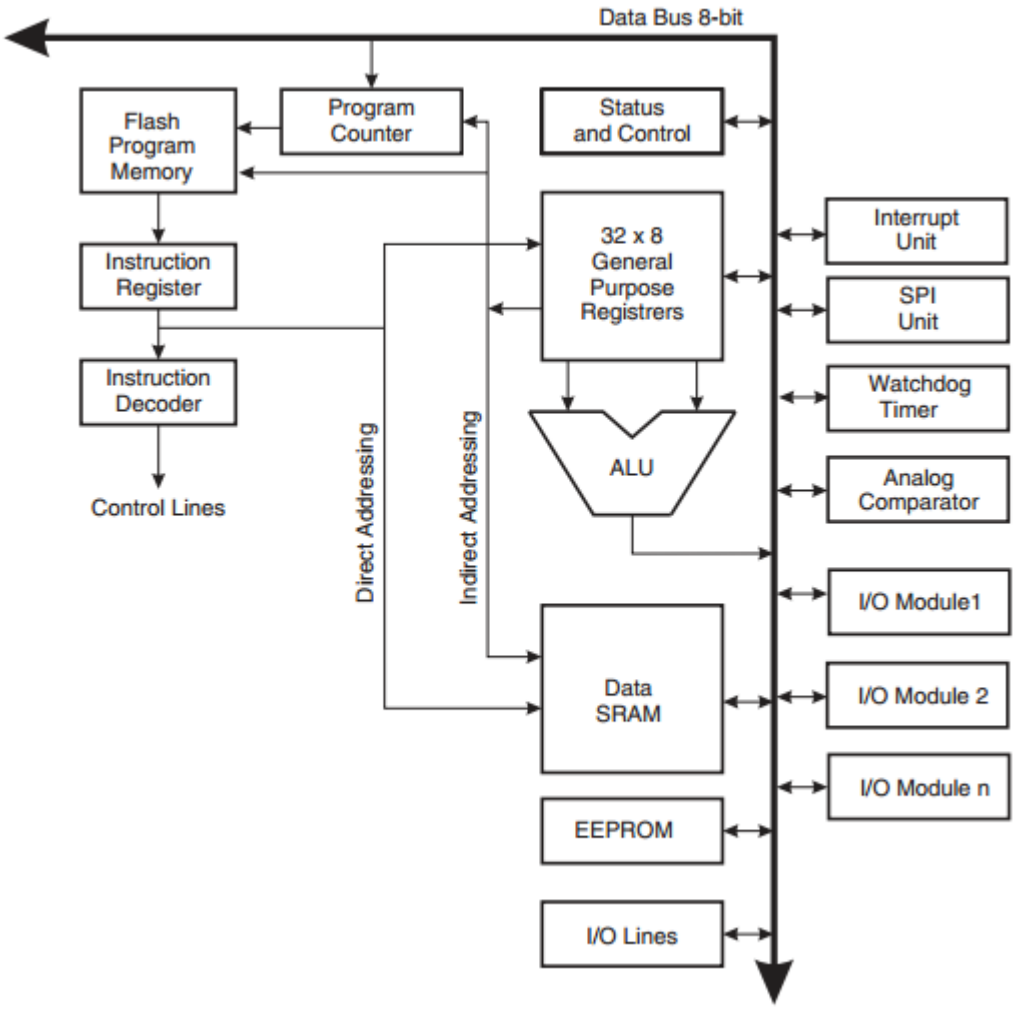
Laitteeseen valittiin ATmega328A-prosessori. Se on matalakulutuksinen 8-bittinen CMOS-mikrokontrolleri, joka perustuu AVR:n tehostettuun RISC-arkkitehtuuriin. (ATMega328 Technical data sheet, 2007) ATmega328A kykenee jopa 1 MIPSin suorituskykyyn 1 MHz:n kellon nopeudella. Prosessorin lohkokaavio näkyy kuviossa 3. ATmega328A sisältää 32 kilotavua flash-muistia, 1 kilotavun EEPROM-muistia, sekä 2 kilotavua RAM-muistia. Prosessorissa on 3 porttia, joista kahdessa on käytössä 8 I/O-linjaa, sekä yksi portti, jossa on 7 I/O-linjaa. Liitäntämahdollisuuksia antavat mm. USART- ja SPI-liitännät, joista SPI:llä ohjataan tässä työssä radiopiiriä.



Kuvio 3. ATmega 328A lohkokaavio (ATMega Technical data sheet, 2007)

Prosessorin ydin on AVR cpu core. Se perustuu Harvard-arkkitehtuuriin, jossa muistit ja linjat ovat erilliset ohjelmaa ja dataa varten. Useimmat tietokoneet nykyään käyttävät von Neumann-arkkitehtuuria, jossa data ja ohjelma ovat samassa muistissa ja käyttävät samaa linjaa. Tällöin ei pystytä samaan aikaan ajamaan ohjelmaa ja hakemaan dataa. Harvard-arkkitehtuurissa puolestaan erillisten linjojen ansiosta voidaan suorittaa ohjelmaa ja hakea dataa samaan aikaan. Tämän teknologian käyttö mahdollistaa myös seuraavan komennon esihaun valmiiksi. Tietokoneissa ei käytetä tätä arkkitehtuuria enää, lähinnä nopeutuneiden prosessoreiden ansiosta. Myös ohjelmointi on helpompaa, koska ei tarvitse eritellä ohjelmamuistia ja datamuistia erik-

seen, joilla on myös eri osoiteavaruudet. Kuviossa 3 kuvataan tarkemmin ATmega 328A-prosessorin core-osuutta.



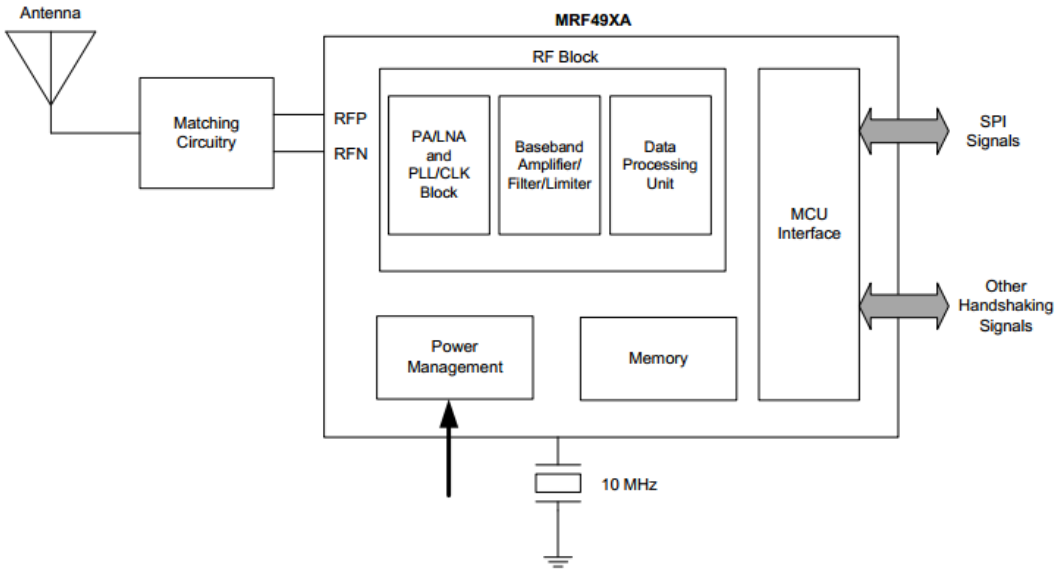
Kuvio 4. AVR core (ATMega328 Technical data sheet, 2007)

AVR core sisältää 32x8 nopeaa yleiskäyttöistä rekisteriä ns. rekisteritiedostossa. Ne ovat kaikki yhden kellojakson hakuajalla, joten se mahdollistaa yhden kellojakson operaatiot AVR-prosessorille. Yhden kellojakson operaatiolla tarkoitetaan sitä, että 2 rekisterin arvoa haetaan rekisteritiedostosta, suoritetaan niille operaatio ja tulos palautetaan takaisin rekisteritiedostoon. Kuten kuviosta 4 nähdään, käytössä on myös useita I/O-rekistereitä, watchdog-ajastin sekä keskeytykset ja SPI-rekisterit.

4. MRF49XA RADIOPIIRI

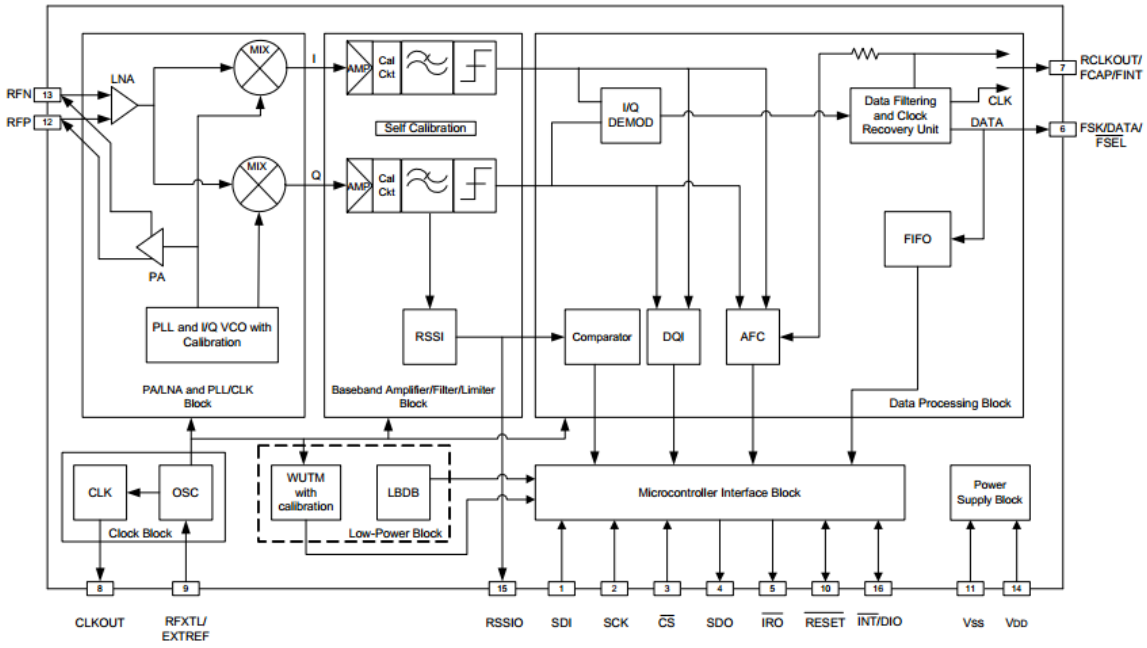
4.1. Yleistä

Seuraavana laitteeseen valittiin radiopiiri, joka hoitaa informaation kulun lähettimen ja vastaanottimen välillä. Tarkoituksena oli valita piiri, joka pystyy kohtuullisiin välimatkoihin, olisi prosessorilla ohjattavissa ja toimii niin sanotuilla vapailla taajuuksilla. Näillä kriteereillä ja hinnan perusteella päädyttiin Microchipin tekemään MRF49XA-piiriin. Piiri on lähetin-vastaanotinmallinen, eli sisältää molemmat. Se on pakattu TSSOP16-koteloon, joka on pintaliitoskotelo. Se tukee suoraan kolmea bandia: 433, 868 sekä 915 MHz:n bandeja. Tämä antaa mahdollisuuden käyttää jotain muuta taajuutta, mikäli tarve niin tulee. Täten itse kytkentää ei tarvitse juurikaan muuttaa, vaan mahdolliset taajuusalueen muutokset voidaan hoitaa komponenttiarvoja muuttamalla. Modulaatio voidaan valita kahdesta vaihtoehdosta: FSK (Frequency Shift Keying) tai FHSS (Frequency Hopping Spread Spectrum). Näistä kahdesta FSK on toiminnaltaan yksinkertaisempi, kun taas FHSS modulaatiolajina pärjää paremmin ulkoisia häiriölähteitä vastaan. Yksi tärkeä valintakriteeri oli myös piirin pieni virrankulutus. Piiri kuluttaa lähetin päällä vain 15 milliampeeria ja lepotilassa vain 0,3 µA. Piirin lohkokaavio löytyy kuviosta 5.



Kuvio 5. MRF49XA Lohkokaavio (MRF49XA User Guide, 2009)

Oikeasta laidasta edeten, piiriin saapuu SPI signaalit prosessorilta, jotka MCU liitäntä selvittää. Sen pohjalta muistia hyväksi käyttäen data etenee data processing unitille. Kun data on prosessoitu, se etenee edelleen baseband-vahvistimien ja harmoniikka-filttereiden läpi PA (Power Amplifier) yksikölle, jossa on myös vaihelukittu silmukka (PLL). Kun signaali on vahvistettu ja muodostettu, se lähetetään differentiaalisignaalina edelleen balunille. Balun on komponenteista kasattu sovitusyksikkö, jossa differentiaalisesta signaalista tehdään vaiheen osalta ja impedanssin osalta sovitettu 50 Ω:n signaali. Balun on lyhenne englanninkielisestä termistä ”from balanced to unbalanced”. Tarkempi esittely piirin toiminnasta löytyy kuviosta 6.



Kuvio 6. MRF49XA Lohkokaavio (MRF49XA User Guide, 2009)

Signaali tulee vasemmalta RFN/RFP-nastoilta. Siitä signaali menee matalakohinaisen vahvistimen (LNA) kautta miksereille. Miksereissä signaaliin sekoitetaan VCO:n (Variable Crystal Oscillator) tuottama signaali ja taajuus ja signaali jaetaan I- ja Q-signaaleiksi. Molemmat sekä I- ja Q-signaalit suodatetaan ja signaalit viedään IQ-demodulaattorille, jossa data erotetaan signaaleista. Tämä data menee eteenpäin ja kellon mukaan synkronoidaan ennen kuin se lähetetään prosessorille. Tähän radiopiiriin on laitettu vaihelukittu silmukka, eli signaali viedään lisäksi AFC:lle (Automatic Frequency Correction) sekä muille vaihelukitun silmukan komponenteille.

4.2. Vaihelukittu silmukka (PLL)

Vaihelukitun silmukan periaatteena on säätää lähtevän pulssin vaihe samaan vaiheeseen tulevan pulssin kanssa. Tällä saadaan parannettua vastaanottimen herkkyyttä. Kun vastaanottimeen tulee signaali, se on alun perin lähetetty tietyllä taajuudella ja tietyssä vaiheessa. Kun se sitten matkaa x-matkan, se saattaa muuttaa taajuuttaan tai lähetin lämmetessään muuttaa hieman omaa taajuuttaan. Vastaanottimessa vastaanotettu signaali syötetään vaihe-ilmaisimeen, joka muuttaa taajuuden vaiheeksi. Samaan vaihe-ilmaisimeen syötetään vastaanottimen värähtelijän taajuus, ja vaihe-ilmaisin ilmaisee niiden erotuksen. Tämä erotus suodatetaan ja syötetään värähtelijälle, joka täten muuttaa omaa taajuuttaan erovahvistimen signaalin mukaan. Vastaanotin siis säätyy automaattisesti vastaanottamaan tietoa oikealta taajuudelta. Kun vaihe on täysin synkronoitu, on taajuus vastaavasti täysin synkronoituna. (Van Roon, 2001).

5. Perävaunun pistoke

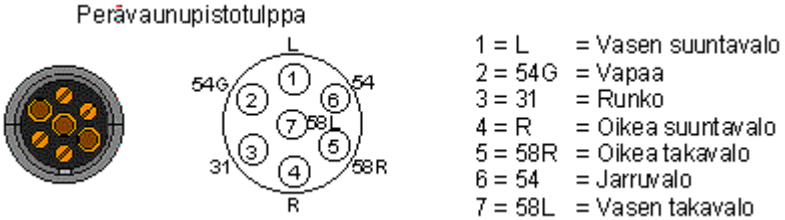
Lähettimen ohjelmiston tarkoituksena on vastaanottaa ajoneuvosta, oli se sitten hinausauto tai henkilöauto, perävaunulle tuleva signaali. Hinausautoissa on samanlainen 7-napainen pistoke kuin henkilöautoissa. Kuviossa 7 on kuva kyseisestä liittimestä. On olemassa myös muunlaisia pistokkeita. Uudemmassa pistorasiassa on 13 pinniä ja luonnollisesti sen kytkentä on erilainen. Kuviossa 8 näkyy uudemman mallinen pistoke.

(Virtasenkauppa, 2013)

Kuvio 7. 7-napainen pistoke

Kuvio 8. 13-napainen pistoke
(Teohydrauli, 2013)

Näiden keskinäiset erot kytkennässä ovat toki merkittäviä, mutta eivät vaikuta oikeastaan laitteen toimintaan millään tavalla. Lähetin on suunniteltu ottamaan vastaan vain ennalta määrätyt signaalit: vasen vilkku, oikea vilkku, jarruvalo ja parkki. 7-napaisen pistokkeen kytkentä selviää kuviosta 9. Vasen ja oikea parkki tulevat pistokkeelle eri nastoihin. Tarvitsemme vain signaalin kertomaan, ovatko parkit päällä, joten otetaan signaali vain yhdestä nastasta. Laite voidaan tehdä myös 13-napaiseen pistokkeeseen sopivaksi. Kuviossa 10 on 13-napaisen perävaunupistokkeen kytkentä.



Kuvio 9. 7-napaisen perävaunupistokkeen kytkentä

Perävaunun sähkötoiminto	Napa nro:	Stand.:
Vasen suuntavilkku	1	L
Takasumuvalo	2	52-G
Maadoitus	3	31
Oikea suuntavilkku	4	R
Oikea takavalo	5	58-R
Jarruvalot	6	54
Vasen takavalo	7	58-L
Peruutusvalo	8	-
Jatkuva virta	9	30+
Akun lataus, virtalukon kautta	10	15+
Maadoitus (akun lataukselle [pistoke 10])	11	10
Kytketyn perävaunun tunnistin	12	-
Maadoitus (jatkuvalle virralle [pistoke 9])	13	11

Kuvio 10. 13-napaisen perävaunupistokkeen kytkentä

6. SYSTEEMISUUNNITTELU

6.1. Vaatimukset

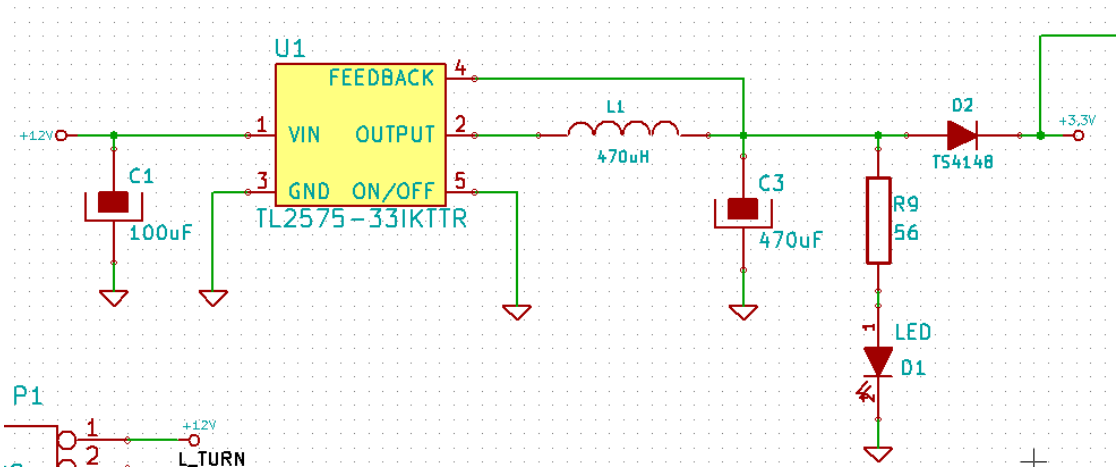
Järjestelmä koostuu lähettimestä, vastaanottimesta sekä valoyksiköistä. Lähetinyksikkö saa virtansa hinausauton perävaunupistokkeesta. Lähettimen kantomatkan tulee olla vähintään 10 metriä. Lähettimen sekä vastaanottimen on oltava vähintään roiskevedeltä suojattu.

Lähetinyksikkö vastaanottaa perävaunupistokkeelta signaalit, jotka ovat 12 V:n suuruisia. Tämä signaali skaalataan alemmalle jännitetasolle, joka sitten luetaan ATmegan prosessorilla. Tästä tiedosta muodostetaan datapulssi, joka lähetetään MRF49XA-lähetin-vastaanotinpiirille. Piiri lähettää tiedon eteenpäin piirilevylle tehdyn antennin kautta.

Vastaanotinyksikkö on akkukäyttöinen. Se käyttää virtalähteenä 12 V:n akkua, nimelliskapasiteetti akulla tulee olemaan vähintään 2 Ah. Vastaanotinyksikkö vastaanottaa tiedon MRF49XA-lähetin-vastaanotinpiiriltä, josta tieto menee ATmegan prosessorille. Sitten datan perusteella ohjataan FET-transistoreita, jotka johtavat valoille tarvittavan virran. Valojen maataso on erotettu optisesti optoerottimilla.

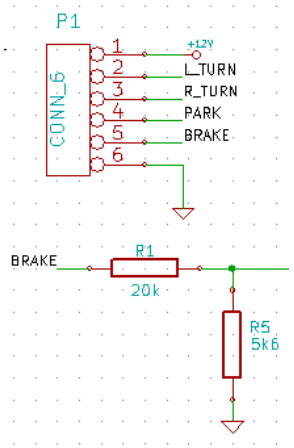
6.2. Lähetinyksikkö

Lähettimen virta tulee hinausauton perävaunupistokkeelta, josta se ohjataan ensimmäisenä hakkuripiirille. Hakkuri on mitoitettu tuottamaan hinausauton 12 V:sta 3,3 V:n tasajännitettä. Tätä samaa jännitettä käytetään sekä ATmega328:n käyttöjännitteenä (prosessori), että myös MRF49XA:n käyttöjännitteenä (radiopiiri). Hakkuriksi on valittu Texas Instrumentsin TL2575-33IKTTR hakkuripiiri. Kyseessä oleva piiri pystyy tuottamaan useita eri jännitteitä kuten 3.3 V, 5 V, 12 V sekä 15 V. Se pystyy ottamaan vastaan jännitteitä alueelta 4,75–40 V. Ensimmäisenä suunniteltiin hakkurin oheiskomponentit hakkuripiirin datalehden avulla. Hakkurin kytkentä komponentteineen löytyy kuviosta 11.



Kuvio 11. Lähettimen hakkurin kytkentä

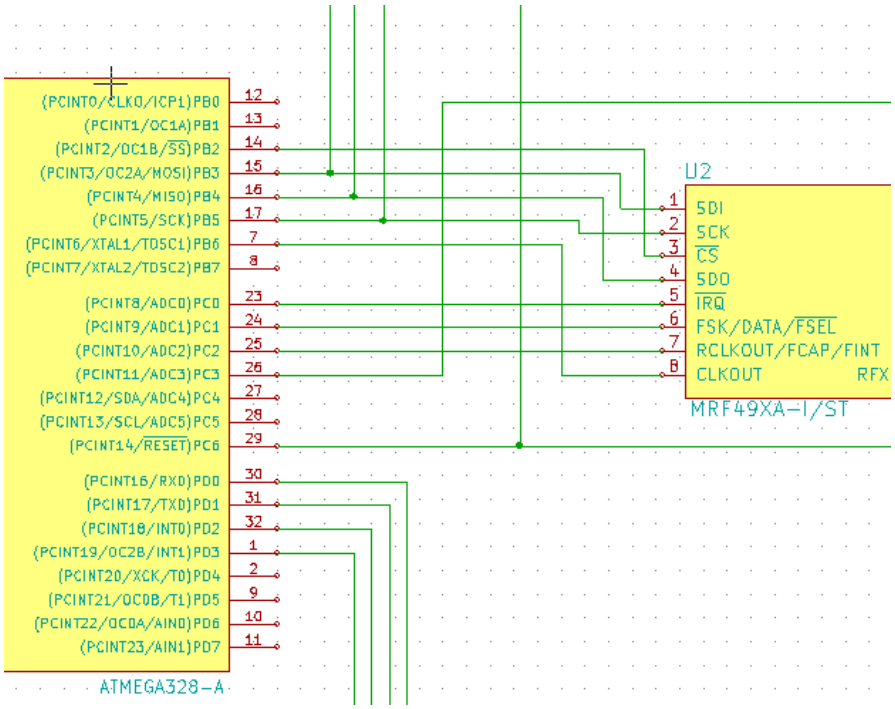
Kuviossa 11 mitoitettavia komponentteja ovat C1, C3, L1 sekä D2. C1 oli annettu suoraan datalehdellä, eikä sitä tarvitse muuttaa, vaikka suunnitelmasta tehtäisiin 5 V järjestelmä. Kela L1 sekä kondensaattori C3 mitoitettiin datalehden mukaan. D2 on suojadiodi, joka estää hakkurille takaisin päin tulevat piikit. R9 ja ledi D1 on laitettu mukaan sen vuoksi, että voidaan visuaalisesti todeta toimiiko hakkuri kun laitteeseen kytketään virta päälle. Kuviossa 12 nähdään kuinka hinausautosta tuleva signaali skaalataan prosessorille sopivaksi.



Kuvio 12. Jännitteen skaalaaminen prosessorille

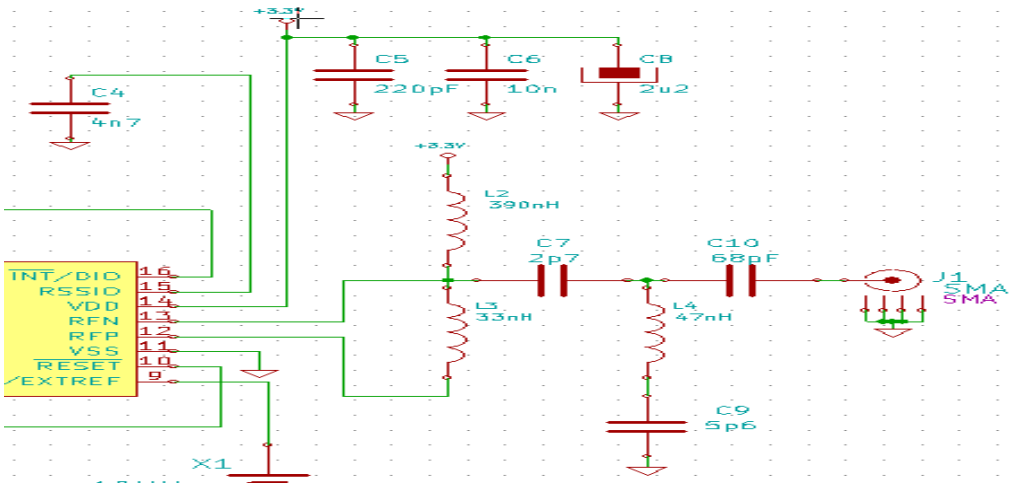
Jännitteen skaalaamisessa käytetään normaalia vastusjakoa, koska prosessori ei ole kovin tarkka sen sisään tulevasta jännitteestä. Jännitteen vaihteluväli pysyy koko ajan prosessorin loogisen 1:sen puolella. Vaihteluväli tulojännitteen vaihdellessa 11–14 V:n välillä saa aikaa lähtöjännitteeksi 2,4 V-3,06 V. Koska prosessori erottelee kaikki yli 1,65 V:n ylittävät jännitteet loogiseksi 1:seksi, ei virheen vaaraa ole.

Prosessori käskee radiopiiriä SPI (Serial Peripheral Interface) yhteyden kautta. SPI:n kautta radiopiiri käsketään oikealle taajuudelle ja määritellään, mitä modulaatiota se käyttää. Kun radiopiiri on alustettu, se vastaanottaa datan pinniin nro 6. Kuvio 13 näyttää ATmegan ja MRF49XA:n välisen yhteyden.



Kuvio 13. ATmega328:n ja MRF49XA:n välinen kytkentä

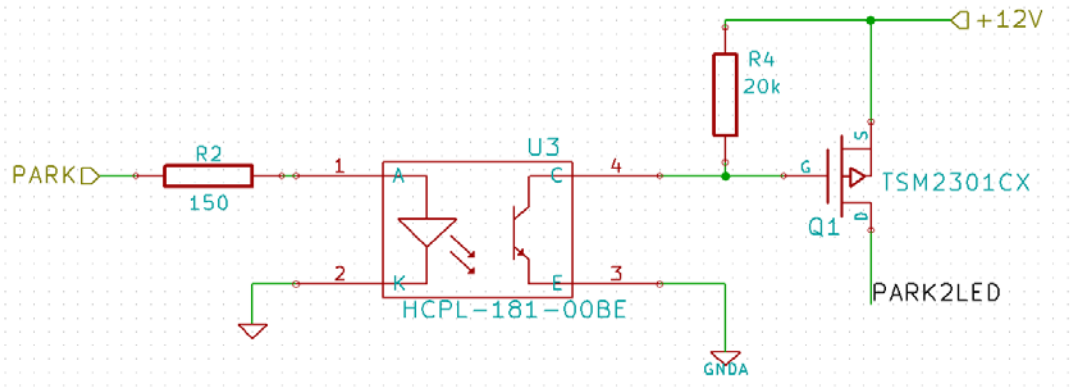
Kellopulssi tulee MRF49XA:lle, joka antaa kellon puolestaan ATmega328:lle. Kun data on prosessoitu MRF49XA:lla, se moduloi signaalin ja lähettää sen eteenpäin antennille. Ennen antennia on vielä Balun, joka sovittaa differentiaalisen signaalin balansoimattomaksi signaaliksi ennen antennia. Balun-kytkentä näkyy kuviossa 14. Balun on mitoitettu MRF49XA:n datalehden mukaan, ja se vaihtuu taajuuden mukaan. Tässä työssä radiopiiri on tarkoitettu käytettäväksi lupavapailla taajuuksilla. Lähetin käyttää 433 MHz:n taajuuskaistaa. Antenni on suoraan KiCadin kirjastoista, ja se on suunniteltu 433 MHz:n taajuuskaistalle. Etäisyyden ollessa kuitenkin aika pieni ei antennia tarvitse tarkkaan sovittaa halutulle taajuudelle.



Kuvio 14. Balun kytkentä

6.3. Vastaanotinyksikkö

Vastaanotin on rakennettu suurilta osin saman mallin mukaan kuin lähetinkin. Hakkuri tuottaa 3,3 V:n jännitteen prosessorille sekä radiopiirille. Erot alkavat siinä, että vastaanottimen virtalähteenä toimii 12 V:n akku. Nimelliskapasiteetiltaan akun tulee olla vähintään 2 Ah luokkaa, jotta laitteelle voidaan taata riittävä toiminta-aika. Kun vastaanotin vastaanottaa signaalin lähettimeltä, radiopiiri prosessoi signaalin ja kääntää sen takaisin dataksi, joka puolestaan lähetetään prosessorille. ATmega328 vastaanottaa datan ja alkaa ohjata valoja. Valojen virta tulee akulta, jota puolestaan ohjataan N-kanava FETeillä. Kytkentä tälle näkyy kuviossa 15.



Kuvio 15. Valojen ohjauspiiri

Signaali valojen ohjaukseen tulee vasemmalta, josta se menee optoerottimelle. Optoerotin erottaa ohjauspiirin maan valojen maasta. Tämä on välttämätöntä, koska valoilla on oma elektroniikkansa, johon ei pystytä vaikuttamaan. Kun optoerotin oh-

jaa FETin hilan maahan, syntyy potentiaaliero hilan ja lähteen välille. Se saa FETin johtavaan tilaan, ja virta alkaa kulkea lähteen ja nielun välillä. Tällöin LED-valo syttyy. Parkit ovat oletusarvoisesti päällä, ja tässä vaiheessa suunnitelmaa on kytkennässä 4 optoerotinta ja 4 FET-transistoria. Tämä määrä on mahdollista vähentää 3:een. Se onnistuu ohjaamalla jarruvaloa PWM-pulssilla. Näin toimiessa ohjataan jarruvaloa n. 25 prosentin teholla ja aikaansaadaan jarruvalo ja jos jarrua painetaan, ohjataan kyseistä pinniä 100 % teholla.

6.4. LED-valot

Vastaanotin käyttää valoina marketeista saatavaa LED-takavaloa. Ne ovat kustannustehokkaita ja helppo ottaa käyttöön langattomissa takavaloissa. Kyseiset valot on pakattu valmiiksi jo vedenpitäviin koteloihin, joten niiden kosteussuojauksesta ei tarvitse huolehtia. Kuva valoista löytyy kuviosta 16.

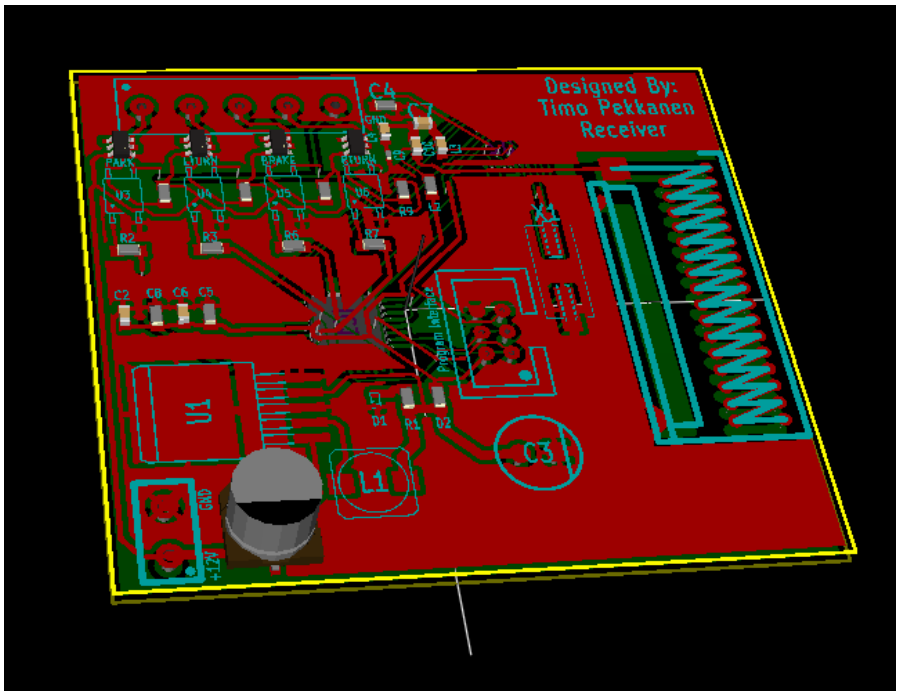
Kuvio 16. LED-takavalo

Kyseisissä LED-takavaloissa on lisäksi mukana heijastin. Akun loppuessa kesken matkan on näkyvyys silti jotenkin turvattuna, koska heijastin ilmoittaa takana ajavalle, että edessä on mahdollinen este.

6.5. Piirilevyjen suunnittelu

6.5.1 Yleistä

Laitteeseen piti suunnitella 2 eri piirilevyä: toinen lähetinyksikköä ja toinen vastaanotinyksikköä varten. Lähettimen levyn kooksi tuli 81.41 milliä leveä ja 49.28 milliä korkea. Vastaanottimen levy puolestaan oli 68,26 milliä leveä ja 61,6 millimetriä korkea. Levyn olisi mahdollisesti saanut tiivistettyä vieläkin pienemmäksi ulkoisilta mitoiltaan, mutta ainakaan ensimmäisellä protokierroksella sitä ei nähty tarpeelliseksi. Levyt suunniteltiin KiCad-ohjelmistolla, joka on vapaan lähdekoodin ohjelmisto. Siinä oli kaikki tarvittava tämän kokoluokan projektin tekemiseen. Levyt ovat 2-kerroksisia FR-4 levyjä. Levyjen paksuus on 1,6 mm. Projektin levyt tilattiin iTead-mall-nimisestä verkkokaupasta, joka tekee levyt Kiinassa. Levyistä tuli niin monimutkaiset, joten oli loogista teettää levyt tehtaassa, jotta toimintavarmuus pystyttäisiin takaamaan. Kuviossa 17 on vastaanottimen levy 3D-mallinnuksena.



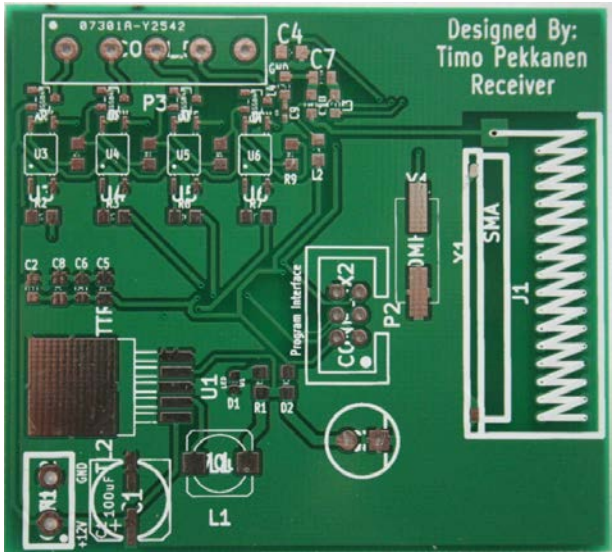
Kuvio 17. Vastaanottimen valmis piirilevy 3D-mallina

Levyn oikeassa laidassa näkyy antenni, joka oli otettu valmiina KiCadin kirjastoista. iTead pystyy tulostamaan myös silkkikerroksen levyille, joten levyjen kalustaminen

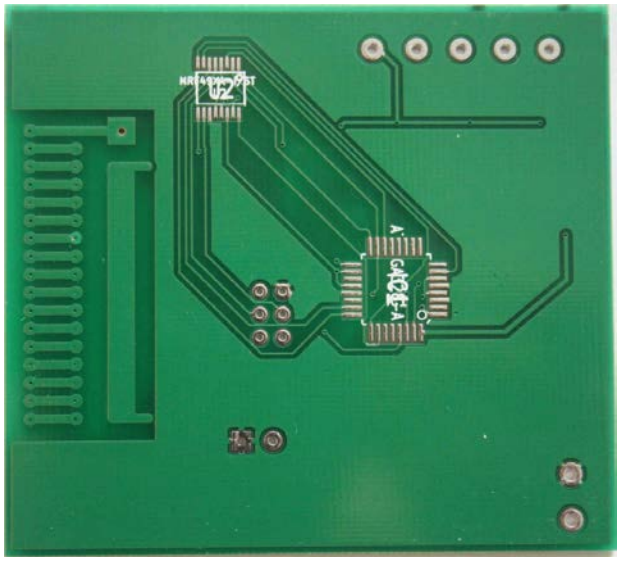
tulee olemaan helpompaa tämän myötä. Tarkat projektin kustannukset löytyvät liitteestä 4.

6.5.2 Valmiit levyt

Valmiiden levyjen saapumiseen meni aikaa tilauksesta n. 3 viikkoa. Levyt saapuivat hyvin pakattuina ilmatiiviisiin pusseihin. Pintapuolinen tarkastelu osoitti, että levyt olivat kohtuullisen hyvää laatua. Reikien kohdistus oli kohdallaan, ja levyt vaikuttivat olevan kunnossa. Virheitä levyissä oli kyllä, mm. silkkipainossa oli tekstejä, jotka eivät sinne kuuluneet. Tämä oli selkeä suunnittelijan virhe. Kuvioissa 18 sekä 19 näkyy vastaanottimen kalustamaton levy.



Kuvio 18. Vastaanotin, Top layer



Kuvio 19. Vastaanotin, Bottom layer

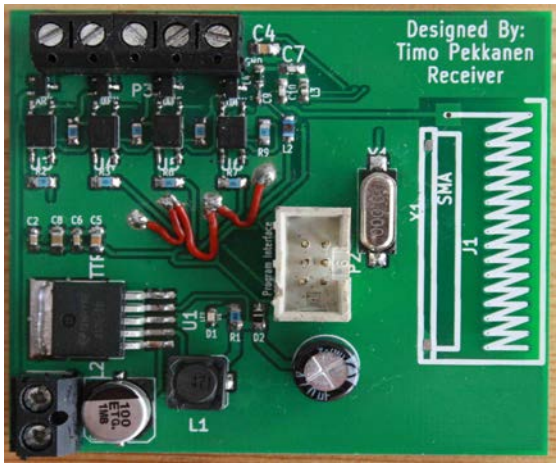
Maatasoja ei ollut supistettu ja ohuetkin vedot olivat siistiä työtä. Koko levy on pade- ja lukuun ottamatta päällystetty juotteenestopinnoitteella, joten kalustaminenkin oli kohtuullisen helppoa.

7. LAITTEEN KOKOONPANO

Kokoonpanovaihe voidaan jakaa osiin. Ensimmäisenä on levyjen kalustaminen, eli osien kiinni juottaminen ja liittimien kiinnittäminen. Toisena vaiheena on levyn testaaminen. Kolmantena vaiheena levylle ladataan ohjelmisto ja tehdään uusi testi levylle. Tämä on niin sanottu toiminnallinen testi. Kun ohjelmisto on todettu toimivaksi ja rautapuoli toimii myös, voidaan siirtyä kenttätestaukseen.

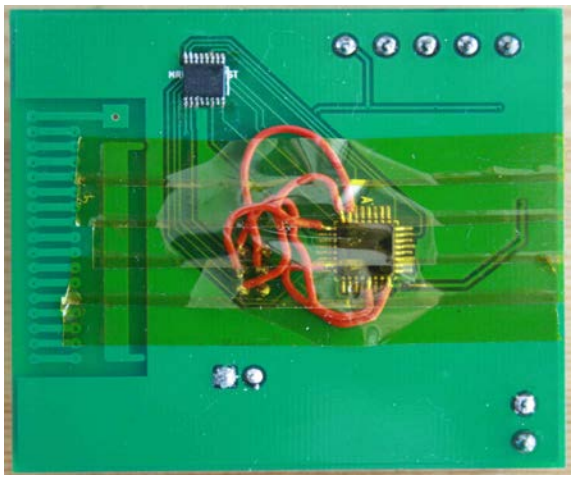
7.1. Levyjen kalustaminen

Levyt olivat kaksipuolisia, eli komponentteja tuli molemmille puolille levyä. Kalustaminen tapahtui kotona, mutta käytössä oli teline, johon levyn sai kiinni. Tämä helpotti komponenttien asettelua ja kiinnijuottamista. Osat kävivät padeihin hyvin, eikä kalustuksen aikana tullut suurempia yllätyksiä. Ohjelmointiliittimen reiät olivat liian pienet, joten niitä joutui hieman suurentamaan asennuksen yhteydessä. Kalustaminen on melko työlästä, ja kahden levyn kalustamiseen menikin aikaa n. 3 tuntia. Kuviossa 20 näkyy kalustettuna vastaanottimen päällimmäinen kerros.



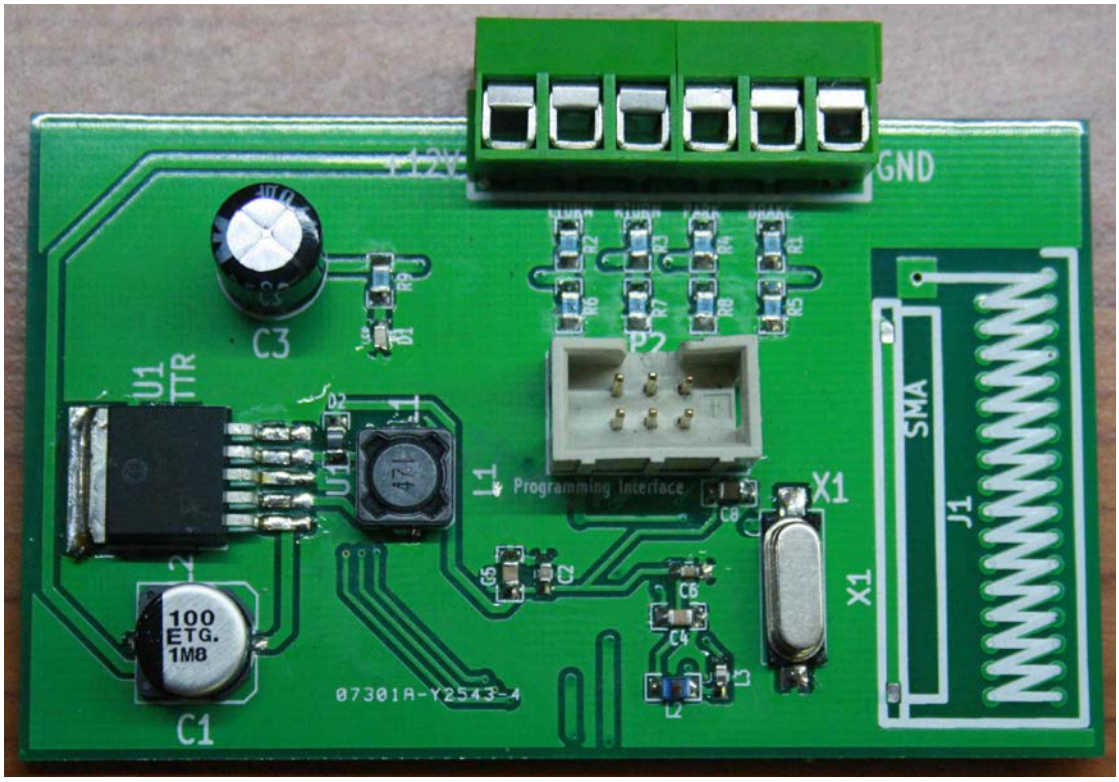
Kuvio 20. Vastaanotin, Top layer kalustettuna

Ja kuviossa 21 näkyy vastaanottimen alempi kerros kalustettuna.



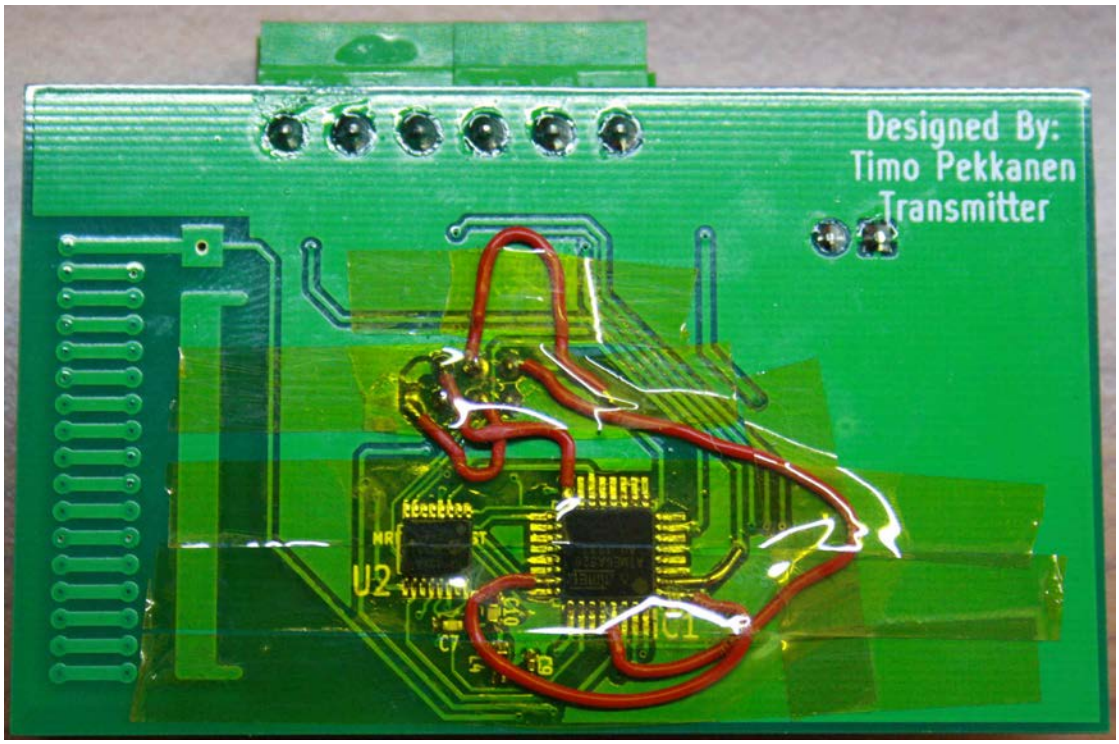
Kuvio 21. Vastaanotin, bottom layer kalustettuna

Kuten kuvioista 20 ja 21 voidaan nähdä, oli levyissä suunnitteluvirheitä, jotka ilmenivät ohjelmointiliittymän testauksen yhteydessä. Päällimmäisen kerroksen hyppylangat johtuivat puuttuvasta maavedosta. Koska kyseistä maavetoa ei ollut vedetty, ei osaan transistoreista kytkeytynyt ollenkaan jännitettä, koska ohjausvirta ei päässyt maihin. Alemmassa kerroksessa olevat hyppylangat puolestaan johtuivat väärin kytketystä ohjelmointiliittimestä. Ongelman ratkaisuksi jouduttiin katkaisemaan liittimelle menevät piirilevyn vedot ja korvaamaan ne johtimilla. Sama virhe toistui molemmissa levyissä. Tarkoituksena on korjata kyseinen virhe seuraavassa erässä, jos sellaisen tekemiseen päädytään. Kuviossa 22 on puolestaan lähettimen päällimmäinen kerros kalustettuna.



Kuvio 22. Lähetin, top layer kalustettuna

Kuten huomataan, lähettimen päällimmäinen kerros onnistui sähköisesti. Täten sinne ei tarvinnut vetää hyppylankoja. Kuviossa 23 on lähettimen pohjakerros.



Kuvio 23. Lähetin, bottom layer kalustettuna

Pohjakerroksessa puolestaan on jälleen hyppylankoja johtuen ohjelmointiliittimen väärästä kytkennästä.

7.2. Levyjen sähköinen testaus

Kuten aikaisemminkin jo mainittiin, levyjen sähköisessä testauksessa havaittiin heti suunnittelussa tapahtuneita virheitä. Testaus aloitettiin laittamalla laitteeseen sähköt. Heti saatettiin havaita, että hakkurivirtalähde toimii, koska laitteeseen sijoitetut LED-valot syttyivät. Yleismittarilla mitattuna myös havaittiin, että hakkuri tuotti halutut 3,3 V jännitettä, joka sitten jaettiin ympäri levyä. Tässäkin kohtaa tosin suunnittelija oli tehnyt virheen. Prosessorille ja radiopiirille menee levyllä tällä hetkellä 2,7 V:n jännite. 0,6 V:n jännite-ero johtuu diodista joka sijoitettiin levylle suojaamaan hakkurin ulostuloa. Tähän suojadiodiin jää 0,6 V:n jännite. Laite on testeissä toiminut myös 2,7 V:lla, johtuen piirien suuresta jännitehaarukasta jolla ne toimivat. Luultavasti seuraavaan kehitysversioon diodi joko poistetaan tai siihen keksitään jotain muuta, jotta jännite saadaan 3,3 V:n tasolle. Seuraavana sähköiseen testaamiseen kuului toki ohjelmointiliittimen testaus. Ohjelmointilaitteena käytettiin AVRISP mkii-ohjelmointilaitetta. Laitteen kuva löytyy kuviosta 24.

Kuvio 24. AVRISP mkii ohjelmointilaite.(Wikipedia Commons, 2011)

Kyseisen ohjelmointilaitteen ominaisuuksiin kuuluu, että kun liitin kytketään kiinni prosessoriin, laitteessa palava led-valo muuttuu punaisesta vihreäksi, mikäli ohjelmointilaite tunnistaa prosessorin. Jos prosessorissa ei ole käyttöjännitettä, led-valo palaa punaisena. Kuviossa 25 esitellään mahdolliset tilat ohjelmointilaitteelle.

LED Color	Description
Red	Idle - No target power
Green	Idle - With target power
Orange	Busy - Programming
Orange blinking	Reversed target cable connection
Red blinking	Short-circuit on target
Red - Orange blinking	Upgrade mode

Kuvio 25. AVRISP mkii laitteen tilat.(AVRISP mkii User Guide, 2005)

Ohjelmointilaitteen kytkeminen levyyn sai aikaan tilanteen, jolloin ohjelmointilaitteen oranssi led-valo vilkkui. Tarkempi tarkastelu osoitti, että ohjelmointiliitin oli kytketty suunnitteluvaiheessa väärin. Sille vialle ei mahda levyllä enää muuta, kuin katkaista liittimelle menevät vedot ja korvata ne hyppylangoilla. Korjausten jälkeen liitin alkoi toimia, ja testien jälkeen myös prosessori ohjelmoitui. Vastaanottimen testaukseen kuului myös FET-ohjauksen toiminnan testaus. Ensimmäisessä testissä vaikka prosessori ohjasi signaalin päälle, ei led-valo syttynyt. Ensimmäinen syy toimimattomuuteen oli maiden erotus. Levyllä on optoerottimet, jotka erottavat 12 voltin maan 3,3 voltin maasta. Koska laitteessa on vain yksi virtalähde, on näiden maiden oltava molempien yhdistettynä virtalähteen maahan. Siltikään laite ei toiminut. Piirikaaviosta ja layout-kuvasta selvisi, että levyltä oli unohtunut kytkeä yksi maaveto. Tämän korjaamiseksi oli laitettava päällimmäiseen kerrokseen 3 hyppylankaa maiden kytkemiseen. Korjausten jälkeen led-valon ohjaus prosessorilla toimii.

8. OHJELMISTO

8.1. Yleistä

Tähän laitteeseen tarvittiin ohjelmisto ohjaamaan laitteen toimintaa. Molempiin sekä lähettimeen, että vastaanottimeen tarvittiin oma ohjelmistonsa. Ohjelmiston tarkoituksena on ohjata laitteen toimintaa vastaanottamalla tietoa ja prosessoimalla se. Prosessoitu tieto sitten ohjataan ulostuloille tekemään tarpeellisia ohjaustoimenpiteitä. Ohjelmisto, jota laite käyttää on melko pitkälti lainattu erään harrastelijan projektista. Projektin tekijä on William Dillon ja hän on tehnyt samaan radiopiiriin perustuvan projektin. Projekti löytyy Internetistä ja on nimeltään: ”RF Transceiver using the MRF49XA”. Linkki kyseisen projektin sivulle löytyy lähdeluettelosta.

8.2. SPI-ajuri

Laitteen käyttöä varten oli rakennettava SPI-ajuri. SPI on lyhenne sanoista Serial Peripheral Interface. Se on standardoitu tietoliikenneprotokolla ja mahdollistaa liikenteen Full-duplex-tilassa. Tätä sovellusta varten on tehty erikseen 16-bittinen SPI-ajuri, koska MRF49XA-radiopiiri käyttää 16-bittistä dataliikennettä. ATMega328-piirissä on sisäänrakennettu rutiini, joka osaa käyttää SPI-väylää, mutta se toimii ainoastaan 8-bittisenä. SPI-liikenne koostuu neljästä pinnistä: SCLK, MOSI, MISO ja SS. Näistä SS toimii invertoituna eli on aktiivinen kun pinnin jännitetaso on 0. SCLK on kellopulssi ja jokaisella kellopulssin nousevalla reunalla prosessori lukee MISO-pinnin. Samaan aikaan prosessorilta menee tieto toiselle laitteelle MOSI-pinnin kautta. SS-pinni toimii Slave Select-pinninä eli prosessori voi valita tämän pinnin avulla, mikä laite kuuntelee lähetystä.

Ajuri itse koostuu header-tiedostosta, jossa määritellään mm. SPI-liikenteeseen käytettävät portit. Itse ajurin tiedostossa on funktiot 8- sekä 16-bittiselle SPI-kirjoittamiselle. Ajurissa on myös funktiot 8- ja 16-bittiselle lukemiselle. Tehty ajuri, kuten myös koko ohjelmakoodi on liitteessä 5. Kuvio 26 esittelee 16-bittisen SPI-kirjoittamisen.

```
void spi_write16(uint16_t data, volatile uint8_t *enable_port, uint8_t enable_pin)
{
    int i = 0;

    // Spin on the mutex
    while (mutex);
    mutex = 1;

    // Bring the enable pin low to enable SPI on the peripheral
    *enable_port &= ~(1 << enable_pin);
    spi_read_buffer = 0;

    for (i = 0; i < 16; i++) {
        if (data & 0x8000) {
            SPI_MOSI_PORTx |=  (1 << SPI_MOSI_BIT); // Set port value to 1
        } else {
            SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
        }

        // Shift read buffer contents
        spi_read_buffer = spi_read_buffer << 1;

        // Read MISO
        if (bit_is_set(SPI_MISO_PINx, SPI_MISO_BIT)) {
            spi_read_buffer |= 0x01;
        }

        // Data valid on rising edge of the clock
        SPI_SCLK_PORTx |=  (1 << SPI_SCLK_BIT); // Set clock value to 1

        // Shift input data
        data = data << 1;

        // Data change on falling edge
        SPI_SCLK_PORTx &= ~(1 << SPI_SCLK_BIT); // Set clock value to 0
    }

    // Return the enable pin and MOSI to the inactive state
    SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
    *enable_port |= (1 << enable_pin);

    mutex = 0;
}
```

Kuvio 26. 16-bittinen SPI-kirjoitus väylään

Radiopiirin ajuri käyttää SPI-ajuria kaikkeen liikenteeseen. SPI-ajurin aliohjelmia ja funktioita ei kutsuta suoraan pääohjelmasta.

8.3. MRF49XA:n ajuri

Radiopiirille piti rakentaa oma ajurinsa, jolla saadaan radiopiiri helposti ja nopeasti alustettua oikeaan tilaan tätä sovellusta varten. Radiopiirin ajuri on jälleen suurelta osin William Dillonin käsialaa. Tässä kappaleessa käydään radiopiirin ajuri pääpiirteissään lävitse.

Prosessorin ja radiopiirin väliseen liikenteeseen on varattu useita prosessorin pinnejä. 4 pinniä menee pelkästään SPI-liikenteelle ja tämän lisäksi on useita pinnejä varattu mm. keskeytysten hallintaan ja dataliikenteen hoitamiseen. Kaikki prosessorin ja radiopiirin porttivalinnat on hoidettu omassa header-tiedostossaan, josta osa näkyy kuviossa 27.

```
/* These defines set the chip select line
 * and interrupt for the MRF module
 *
 */
#define MRF_CS_PINx     PINB
#define MRF_CS_PORTx    PORTB
#define MRF_CS_DDRx     DDRB
#define MRF_CS_BIT      2

#define MRF_IRO_PINx    PIND
#define MRF_IRO_PORTx   PORTD
#define MRF_IRO_DDRx    DDRD
#define MRF_IRO_BIT     2
#define MRF_IRO_VECTOR  INT0_vect

#define MRF_FSEL_PORTx  PORTC
#define MRF_FSEL_DDRx   DDRC
#define MRF_FSEL_BIT    1
```

Kuvio 27. Prosessorin ja radiopiirin porttimäärityksiä

Radiopiirin alustamiseen ja määrittämiseen täytyi radiopiirin datalehdeltä etsiä tarvittavat määritykset. Helpointa oli ottaa määritykset ja antaa niille omat nimet omassa header-tiedostossaan. Kuviossa 28 on esitelty yksi määritys.

```
#pragma mark Power Management Configuration Register
#define MRF_PMCREG      0x8200      // Power Mgmt. Config. Register address
#define MRF_RXCEN       0x0080      // Receiver chain enable
#define MRF_BBCEN       0x0040      // Baseband chain enable
#define MRF_TXCEN       0x0020      // Transmitter chain enable
#define MRF_SYNEN       0x0010      // Synthesier enable
#define MRF_OSCEN       0x0008      // Oscillator enable
#define MRF_LBDEN       0x0004      // Low Battery Detector Enable
#define MRF_WUTEN       0x0002      // Wakeup timer enable
#define MRF_CLKODIS     0x0001      // Clock output disable
```

Kuvio 28. Datalehden määrityksiä ohjelmistossa

Kuten kuviosta 28 nähdään, prosessorin täytyy lähettää ensin komentosana ylimmässä 8 bitissä. Sitten loput 8 bittiä määrittelee eri tiloja radiopiirille, kuten rx-ketjun päällä olon tai baseband-ketjun päällä olon. Samanlaisia määrityksiä on n. 20 kappaletta datalehdellä ja kaikki on kuvattu vastaavalla tavalla header-tiedostoon. Määritysten käyttäminen ja koodin lukeminen helpottuu, kun bittikuvioille annetaan nimet.

Kun määritykset on tehty, alkaa varsinainen ajuri. Itse ohjelma toimii keskeytyspohjaisesti eli ajuri tutkii erään pinnin tilaa. Ajurissa on eri funktiot, eli aliohjelmat, joilla mm. alustetaan radiopiiri käyttöön, vastaanotetaan viesti tai lähetetään viesti. Esimerkkinä on paketin lähettämiseen tarkoitettu aliohjelma, joka näkyy kuviossa 29

```
void MRF_transmit_packet(MRF_packet_t *packet)
{
    uint8_t i;
    uint8_t wait = 1;

    // We can check, without synchronization
    // (because it doesn't change in the ISR)
    // whether we're in a testing mode.  If so, simply return.
    // This is important, so we don't hard lock.
    if (mrf_state & MRF_TX_TEST_MASK) {
        return;
    }

    // Wait for the module to be idle (this is cheezy synchronization)
    do {
        cli();  // Disable interrupts, this is a critical section
        if (mrf_state == MRF_IDLE) {
            mrf_state = MRF_TRANSMIT_PACKET;
            wait = 0;
            sei();  // Atomic operation complete, reenable interrupts
        } else {
            sei();  // Atomic operation complete, reenable interrupts
            //_delay_ms(1); // This should be roughly 1 byte period
        }
    } while (wait);

    // Initialize the constant parts of the transmit buffer
    counter = 0;
    transmit_buffer[0] = 0xAA;
    transmit_buffer[1] = 0x2D;
    transmit_buffer[2] = 0xD4;
    transmit_buffer[3] = packet->length;

    // Copy the packet contents into the buffer
    for (i = 0; i < packet->length; i++) {
        transmit_buffer[i + 4] = packet->payload[i];
    }

    RegisterSet(MRF_PMCREG);                        // Turn everything off
    RegisterSet(MRF_GENCREG_SET | MRF_TXDEN);       // Enable TX FIFO
    // Reset value of TX FIFO is 0xAAAA

    RegisterSet(MRF_PMCREG | MRF_TXCEN);            // Begin transmitting
    // Everything else is handled in the ISR
}
```

Kuvio 29. Paketin lähettämiseen tarkoitettu funktio

Kuten funktiosta voidaan huomata, toimii ajuri eri tiloissa. Nämä eri tilat määrittää mitä prosessori komentaa radiopiirin tekevän. Samoin tilasta voidaan päätellä mitä prosessorin tarvitsee suorittaa, jos lähetetään pakettia. Paketti itsessään koostuu niin sanotusta preamble-kohdasta, jolla vain on tarkoitus saada radion sisäinen viritys kohdalleen. Seuraavana ovat synkronointipaketit, joiden perusteella vastaanotin tunnistaa, että nyt on paketti tulossa. Synkronointipaketit koostuvat kahdesta 8-bittisestä kuviosta. Sitten seuraa paketin dataosuus.

8.4. Lähetin

Lähettimen ohjelmisto vastaanottaa tietoa perävaunupistokkeelta ja ohjaa sen eteenpäin radiopiirille. Lähettimen ohjelmiston täytyy myös käskeä radiopiiriä toimimaan oikein. Kun perävaunupistokkeella joku signaali nostaa jännitetason tarpeeksi korkeaksi, lähettää lähetin tiettyä kuviota jatkuvasti. Lähettimen ei tarvitse säästellä virtaa, koska se saa käyttösähkönsä perävaunupistokkeelta.

8.5. Vastaanotin

Vastaanotin saa virtansa akulta. Vastaanottimen ohjelmistossa on samat ajurit kuten lähettimessäkin: SPI-ajuri sekä radiopiirin ajuri. Vastaanottimeen on ohjelmoitu tietty bittikuvio, jonka luettuaan ohjelma alkaa tallentaa vastaanottobufferiin tietoa. Mikäli vastaanotettu tieto vastaa tiettyä bittikuviota, sen perusteella prosessori nostaa I/O-portin tason ylös. Ylösnostettu pinni sitten ohjaa optoerottimen kautta FET-transistoria, joka sytyttää kyseisen valon.

9. VIAT JA NIIDEN KORJAUS

9.1. Yleistä

Kuten kaikissa isommissa projekteissa tuotteen luominen on prosessi. Tämä versio oli ensimmäinen protoversio ja siihen jäikin useita virheitä, jotka pitää korjata ennen tuotteen valmistumista. Vikoja laitteessa on sekä piirilevyjen puolella, mutta myös ohjelmiston osalta. Seuraavakaan proto ei tulisi luultavasti olemaan vielä täysin virheetön, mutta moni asia tulisi olemaan siinä paremmin. Laitteen RF-osien suorituskykyä on mahdoton arvioida, koska tarvittavia mittalaitteita ei ole saatavilla käyttöön tässä vaiheessa. RF-osien sovitus, varsinkin balunin osalta ei ole luultavasti optimaalinen. Vetojen pituudet pitäisi mitoittaa ja laskea ja avaukset myös RF-vetojen osalta pitäisi mitoittaa paremmin. Tällä saataisiin mahdolliset heijastumat poistettua ja kaikki saatavilla oleva teho saataisiin antennille asti. Myös levyllä oleva 10MHz kide on sijoitettu melko lähelle antennia ja balunia ja se saattaa häiritä radiopiirin toimintaa. Näiden vaikutusten arviointi on kuitenkin vaikeaa koska mittauksia ei saa tehtyä levyltä ilman tarvittavia mittalaitteita.

9.2. Lähetin

Lähettimessä ensimmäinen ja suurin virhe on ISP-ohjelmointiliittimen väärä kytkentä. Suunnitteluvaiheessa liitin jostain syystä tuli liitettyä väärin ja siksi tämän proton kalustusvaiheessa jouduttiin katkomaan vetoja ja laittamaan hyppylankoja. Korjaaminen tälle vialle on helppo, suunnitteluohjelmassa kytketään vain vedot oikein. Toinen virhe ilmeni ohjelmiston tekovaiheessa. ATMega328-prosessorissa on tiettyjä toimintoja tietyissä porteissa. Missään muissa porteissa ei välttämättä ole tiettyjä toimintoja. Lähettimen ohjelmisto käyttää keskeytysrutiinia, joka laukaistaan vain pulssin nousevalla reunalla. Portti, jossa oli saatavana kyseinen keskeytys, oli jo varattu toiseen käyttöön. Tämän vian korjaamiseksi leikattiin kirurgin veitsellä piirilevyltä veto poikki. Seuraavaksi raaputettiin varovasti vedon pinnasta juotteenesto-pinnoite pois. Esiin tulleeseen kupariin juotettiin kytkentälanka kiinni ja se juotettiin piirin jalkaan

kiinni. Toki jos jalkaan tulee toinen veto, niin se on katkaistava. Pienin tehdyistä virheistä oli hakkurin ulostulon kohdalla. Hakkuria suojaamaan laitettiin yksi diodi. Tämä diodi laskee prosessorille ja radiopiirille menevän jännitteen 3,3 V:sta 2,6 V:iin. Suoranaista vaikutusta toimintaan ei pystytty todentamaan, mutta jännitteen lasku saattaa aiheuttaa suorituskyvyn alenemista jossain määrin.

9.3. Vastaanotin

Vastaanottimen kytkentä prosessorin, hakkurin ja radiopiirin osalta oli kopioitu suoraan lähettimen piirikaaviosta. Täten vastaanottimeen tuli sama vika kuin lähettimeen ISP-liittimen osalta. ISP-liittimen hätäkorjaus toteutettiin samalla lailla kuin lähettimessäkin: katkomalla vedot ja liittämällä ne kytkentälangoilla oikeille paikoilleen. Tämän lisäksi vastaanottimessa oli muutamia muitakin vikoja. Tärkein näistä oli puuttuva maaveto LED-valojen ohjauselektroniikassa. Vastaanottimeen laitettiin optoerottimet, jotta LED-valojen maataso saataisiin erotettua prosessorin maatasosta. Mutta maatasot oli kuitenkin yhdistettävä, koska ei ole käytettävissä kahta erillistä virtalähdettä. Tämän lisäksi optoerottimien maatasot oli yhdistämättä. Tämäkin vika saatiin korjattua protoversiossa lisäämällä kytkentälangat oikeisiin paikkoihin. Vastaanottimen prosessorin kytkentä oli samoin väärä. Keskeytyksiä varten tarvittavat pinnit oli varattu toiseen käyttöön ja täten ohjelmiston tekeminen ei onnistunut kunnolla. Korjaus oli sama kuin lähettimessä. Samoin vastaanottimen hakkurin ulostulossa on diodi, joka laskee käyttöjännitettä alas diodin kynnysjännitteen verran.

10. TYÖN LOPPUARVIOINTI

10.1. Tulosten arviointi

Tätä työtä tehdessä on tavoitteena alusta lähtien ollut tehdä toimiva tuote. Mielestäni siinä on onnistuttu, sillä laite toimii. Jos ajatellaan laitteen toimintaa hinausautokäytössä, niin se sytyttää LED-valot annettujen signaalien mukaan. Pakettiliikenne ei

ole täysin aukoton: Kaikki paketit eivät tule perille puhtaasti. Osa pakettien korruptoitumisesta saattaa aiheutua RF-osien sovituksen puutteesta ja ohjelmiston huonoudesta. Myös muiden häiriöiden kestoa ei ole pystytty tämän työn aikana arvioimaan. Mitä tulee suunnittelijaan itseensä, niin eihän tämä ole aivan niin aukoton kuin oli tarkoitus. Levyssä on vielä paljon parannettavaa ja levyjen kokoa pystyisi vielä pienentämään aika paljonkin. Toiminnan epävarmuutta lisää levyssä olevien virheiden lisäksi ohjelmiston puutteet.

Tämän kokoluokan projektissa ennakkosuunnittelu nousee suureen arvoon. Pitää pystyä ennakoimaan tarvittava toimintaympäristö, vaadittavat markkinat, laitteen koko, suojaukset, ohjelmiston vaatimukset, osatilauksen koko ja tarvittava suunnitteluaika. Tehty projekti on vasta alkuvaiheissaan vaikkakin laite toimii. Mekaniikkasuunnittelu on tekemättä kokonaan. Lähetin pitäisi koteloida vedenpitävästi ja liittää perävaunupistokkeeseen. Lisäksi laitteen toimintavarmuutta parantaisi mm. kaksisuuntainen liikenne, eli jokainen lähetetty paketti kuitataan vastaanottimen puolesta. Sen avulla voitaisiin saada aikaiseksi virhetilanteiden korjaus ja hätätilanteiden varalle vaikka varoitusvilkkutoiminto.

10.2. Mitä opittiin?

Tämä työ opetti minulle paljon lisää projektin läpiviennistä. Tiesin aikaisemman kokemuksen perusteella, että isomman projektin läpivientiin tarvitaan useampia suunnittelijoita, jotka ovat kaikki oman erikoisalansa osaajia. Huolellinen suunnittelu antaisi paljon paremmin mahdollisuuden hallita laitteeseen tulevia ominaisuuksia. Samoin aikataulu olisi helpommin hallittavissa. Kulut voitaisiin varmaankin saada vähintään 30 % alemmas huolellisella suunnittelulla ja hankinnan tehostamisella. Ymmärsin tätä tehdessä, että käytetystä ajasta vähintään 75 % meni uuden opetteluun. Radiopiirin toiminnan ymmärtäminen edes jollakin tasolla otti paljon aikaa. Samoin RF-osien mitoitus ja sovitus on opettelematta ja tekemättä kokonaan. Elektroniikan suunnittelu ja levyn teko oli tehtävistä hommista ehkä tutuinta. Myös ohjelmistoa tehdessä tuli havaittua, että ohjelmointi ei ole minulle sitä kaikkein tehokkainta työskentelyä. Ymmärrän miten ohjelmalla voidaan vaikuttaa laitteen toimintaan, mutta

tiedän myös, että ohjelmiston toiminta voitaisiin saada paremmaksi ja kevyemmäksi jos sen tekisi joku ohjelmistosuunnittelija. Vanha sanonta ”suutari pysyköön lestissään” pitää paikkansa projektisuunnittelussa. Isomman projektin nopea ja sujuva läpivienti vaatii ympärilleen osaavan ryhmän, jossa vastuut ja tehtävät on jaettu osaamisalueiden mukaan. Vain siten voidaan saada aikaan järkevä tuote jota on järkevä valmistaa ja markkinoida. Myös raportointi ja katselmointi ovat asioita, jotka pitää olla kunnossa. Varsinkin raportointi on tärkeää tuotteen jatkoa ajatellen. Jos tuotteen valmistus ja toiminta on hyvin raportoitua, on helppo jatkaa tuotteen kehittelyä tai korjaamista pidemmänkin ajan jälkeen. Katselmoinnin avulla olisi suurin osa suunnittelussa tapahtuneista virheistä saatu korjattua ennen levyjen tilaamista. Näihin lukeutuu mm. ISP-liittimen viat, maatasojen kytkemättömyys, RF-sovitusten teko, sekä vääriin pinneihin kytketyt signaalit.

10.3. Mitä jatkossa?

Tuotteen jatkokehitys jatkuu edelleen. Tarkoitus olisi valmistaa seuraavaksi protoversio 2. Siinä korjataan ainakin levyillä olevat viat ja tarkoitus olisi myös pienentää levyjen kokoa. RF-sovitukset olisi myös hyvä saada tehtyä. Se tosin vaatii melko paljon lukemista ennen kuin on edellytyksiä alkaa niitä tekemään. Ohjelmistoa joudutaan myös muuttamaan kun saadaan keskeytyssignaalit oikeisiin pinneihin. Laitteen kotelointi pitäisi miettiä myös sekä LED-valojen kiinnitykset. Kun kotelointi, vastaanottimen akku ja uudet levyt on kalustettu ja ohjelmoitu, päästään kenttätesteihin. Kenttätestit kertovat meille paljon informaatioita siitä mm. kuinka häiriöherkkä laite on. Hinausautoissa on nykyään melko paljon elektroniikkaa, mm. lava-automatiikkaa joka toimii kaukosäätimellä. Sen vaikutus laitteen toimintaan on täysin arvoitus.
Samaa laitetta pienin muutoksin pystyttäisiin käyttämään moneen muuhunkin toimintaan. Kotona sitä voitaisiin käyttää sytyttämään valoja, käynnistämään pesukonetta ym. Samoin ajoneuvotekniikassa tällä laitteella voidaan saada kytkettyä eri laitteita päälle, mm. varoitusmajakoita, ilman johtojen vetämistä. Käyttömahdollisuuksia on monia, laite pitää vain kehittää sille tasolle, että se vastaa markkinoiden vaatimuksiin.

LÄHTEET

ATMega328 Technical data sheet, 2007. Viitattu 10.2.2013.
http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Summary.pdf

MRF49XA User Guide, 2009. Viitattu 17.3.2013.
http://www.farnell.com/datasheets/1669470.pdf

van Roon, T. 2001. What exactly is a PLL? Viitattu 5.12.2012.
http://www.sentex.ca/~mec1995/gadgets/pll/pll.html.

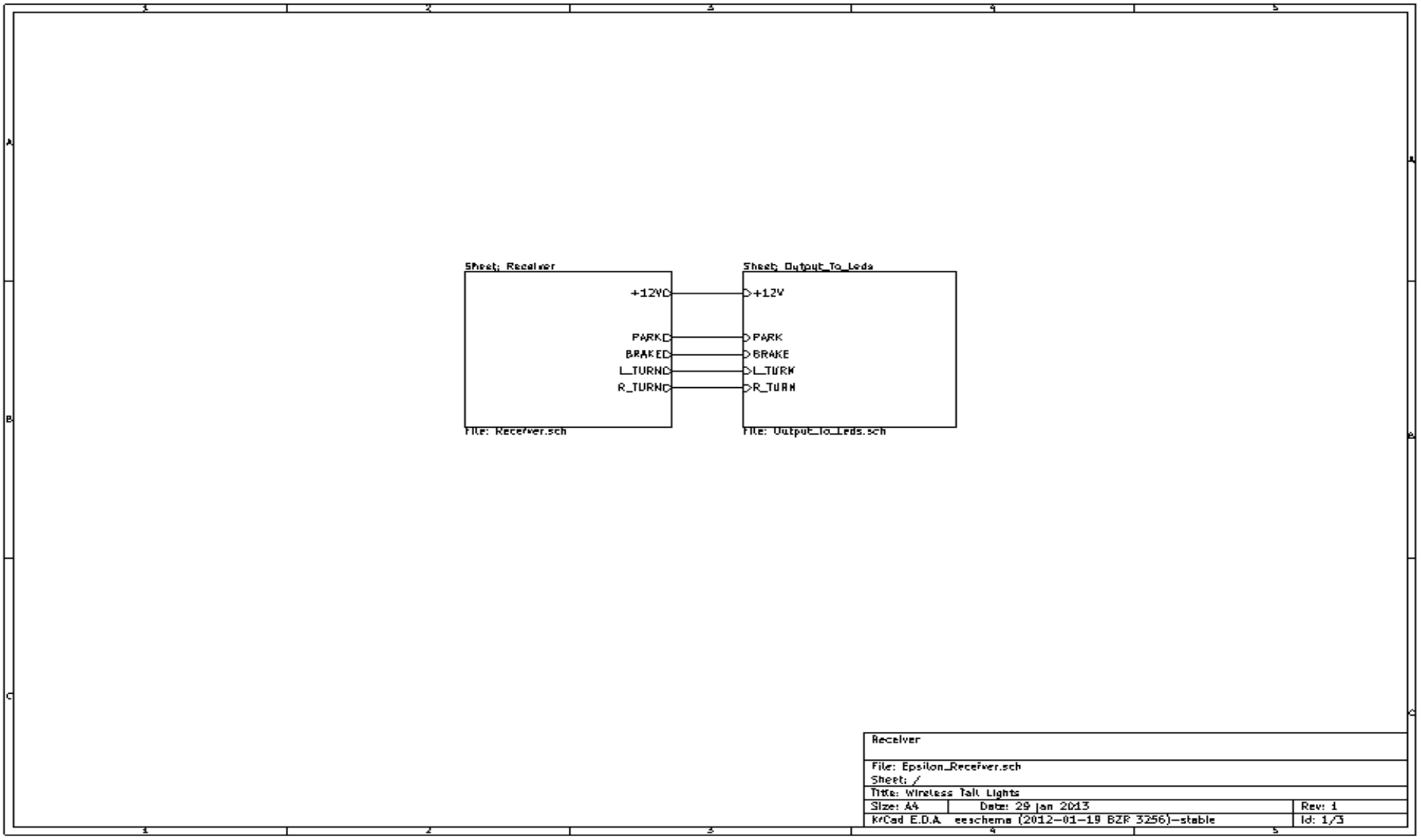
Wikipedia Commons, 2011. Viitattu 23.9.2013.
http://commons.wikimedia.org/wiki/File:AVRISP_mkII.jpg

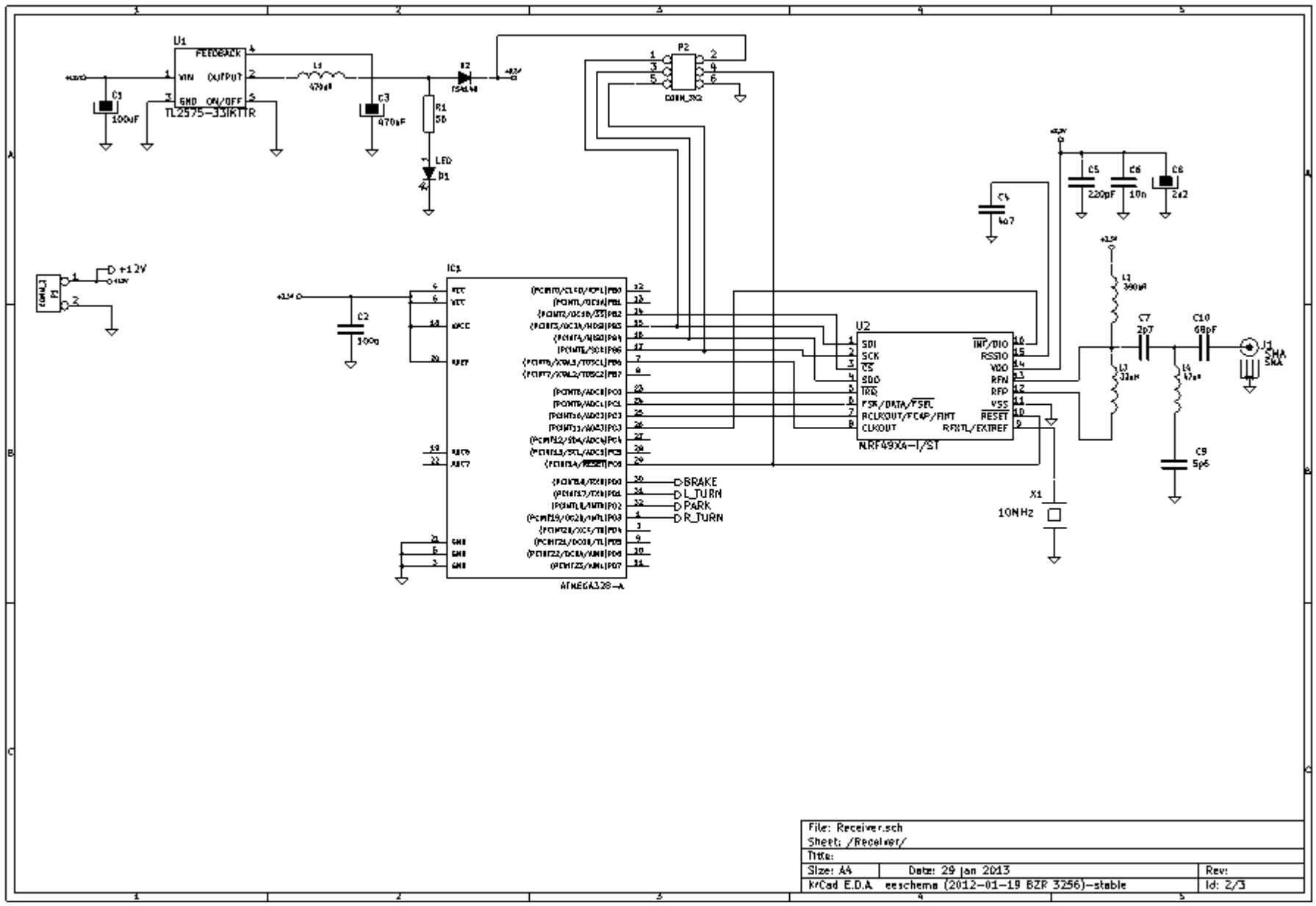
Virtasenkauppa, 2013. Viitattu 18.10.2013.
http://www.virtasenkauppa.fi/tuote/63601/sahkotarvikkeet/peravaunupistorasia-7-nap

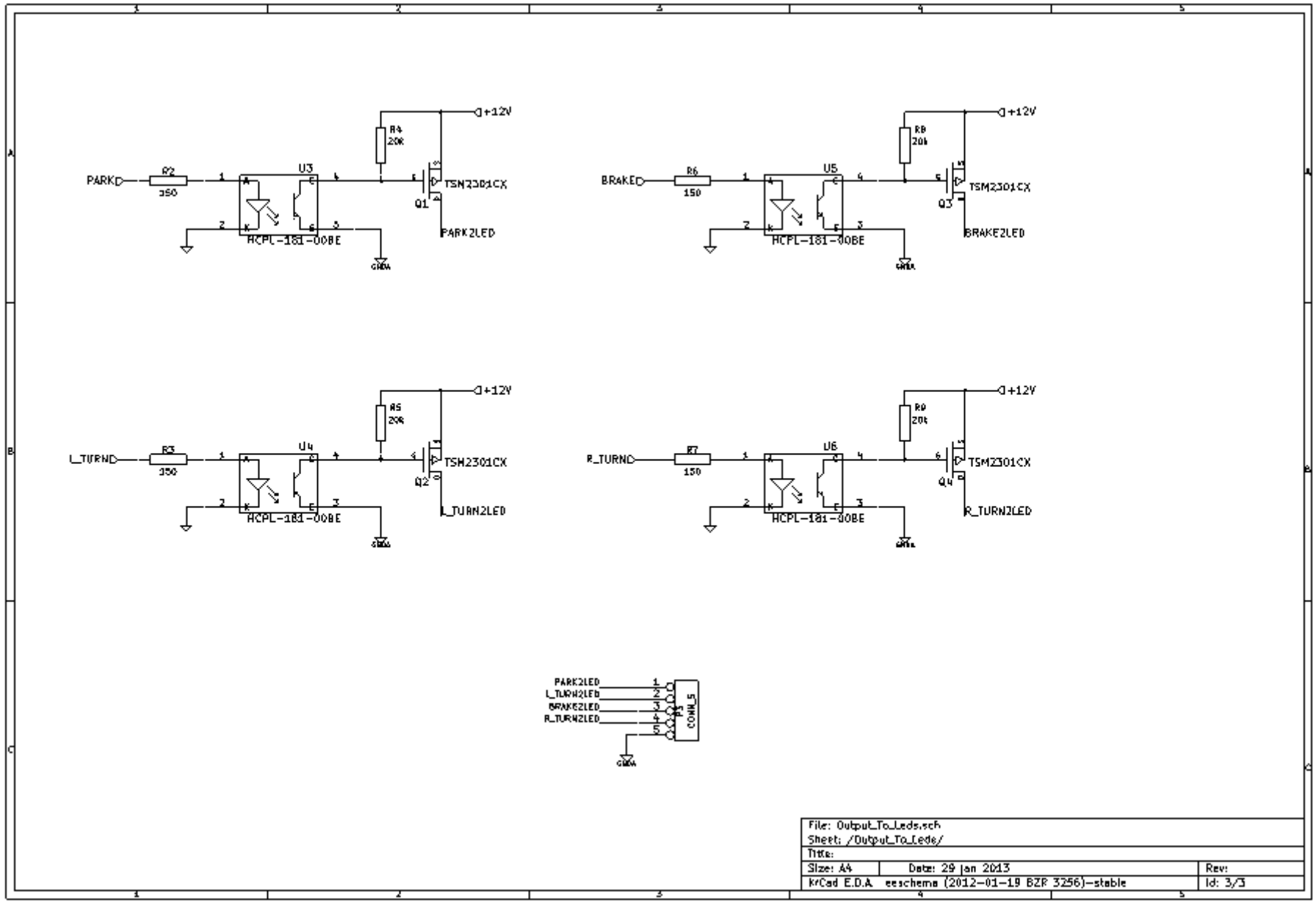
Teohydrauli, 2013. Viitattu 18.10.2013.
http://www.teohydrauli.fi/PublishedService?file=page&pageID=9&itemcode=15546

LIITTEET

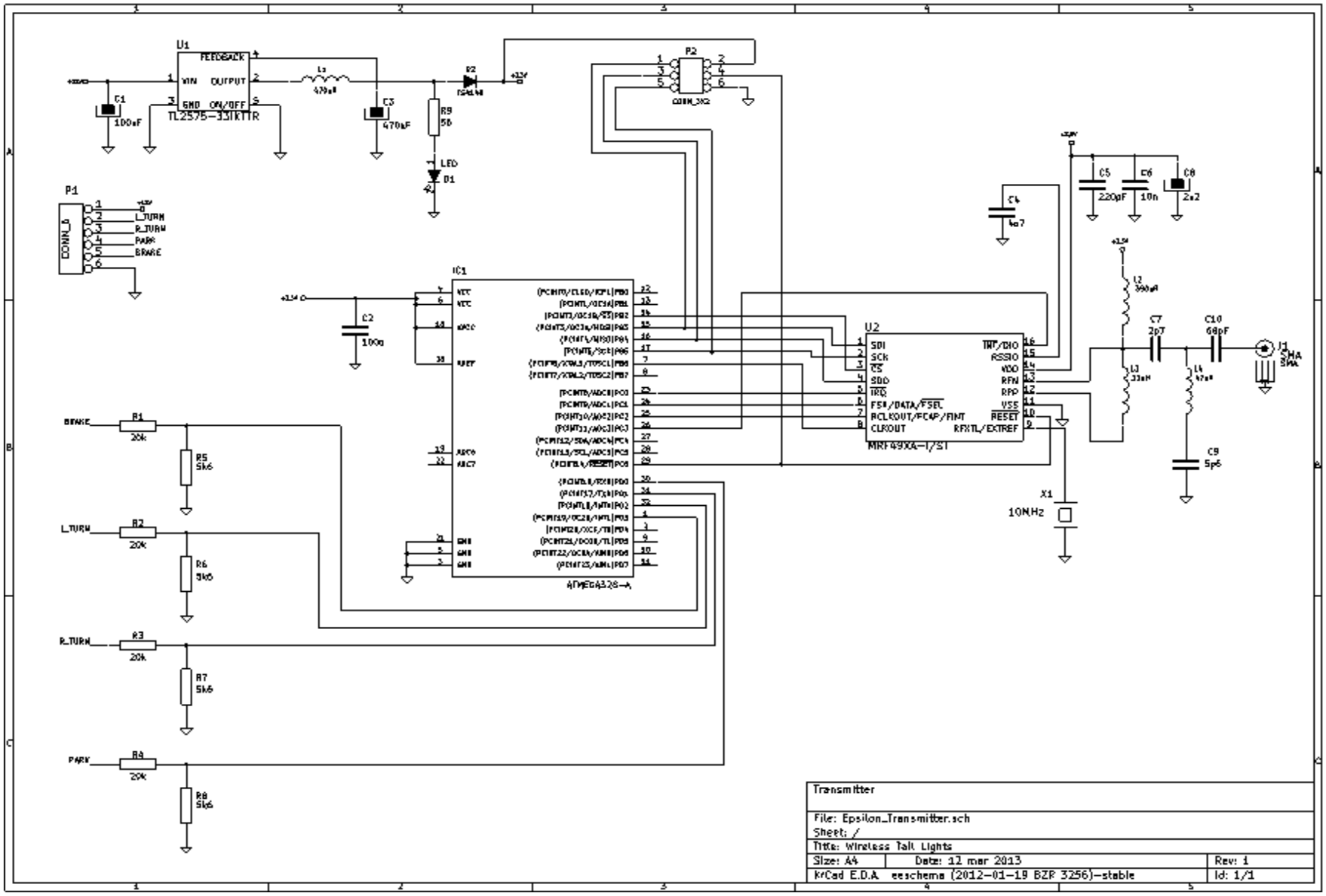
Liite 1 – Vastaanotin




U1
FEEDBACK
VIN
OUTPUT
GND
ON/OFF
TL2575-33IKTTR
C1
100uF
L1
470uH
C3
470uF
R1
5R
D2
LED
D1
P2
CON_3X2
C4
4u7
C5
220pF
C6
10n
C8
2u2
L2
390uH
C7
2p7
C10
68pF
J1
SMA
SMA
L3
33nH
L4
47nH
C9
5p6
U2
SDI
SCK
CS
SDO
IRQ
FSK/DATA/FSEL
RCLKOUT/FCAP/FINT
CLKOUT
INT/DIO
RSSIO
VDD
RFN
RFP
VSS
RESET
RFXTL/EXTREF
NRF49X4-I/ST
X1
10MHz
IC1
ATMEGA328-A
C2
100n
CON_2
P1
+12V
BRAKE
L_TURN
PARK
R_TURN
File: Receiver.sch
Sheet: /Receiver/
Title:
Size: A4
Date: 29 jan 2013
Rev:
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable
Id: 2/3


PARKD
R2
350
U3
HCPL-181-00BE
R4
20R
+12V
TSM2301CX
Q1
PARK2LED
BRAKED
R6
150
U5
R8
20R
Q3
BRAKE2LED
L_TURND
R3
350
U4
R5
20R
Q2
L_TURN2LED
R_TURND
R7
150
U6
R9
20R
Q4
R_TURN2LED
PARK2LED
L_TURN2LED
BRAKE2LED
R_TURN2LED
P3
CONN_5
GND
File: Output_To_Leds.sch
Sheet: /Output_To_Leds/
Title:
Size: A4
Date: 29 Jan 2013
Rev:
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable
Id: 3/3

Liite 2 – Lähetin



Liite 3 – Bill Of Materials

Order No	Part Ref.	Value	Quantity	Price á	total
2079278	C1	100uF	10	0,79	7,9
1759122	C2	100n	100	0,009	0,9
1902884	C3	470uF	10	0,073	0,73
1759241RL	C4	4n7	100	0,013	1,3
1759208RL	C5	220pF	100	0,013	1,3
3019561	C6	10n	20	0,007	0,14
721890RL	C7	2p7	20	0,024	0,48
1759473RL	C8	2u2	100	0,02	2
1611949	C9	5p6	100	0,004	0,4
1759064RL	C10	68pF	100	0,009	0,9
1058359RL	D1	LED	10	0,06	0,6
8150206	D2	TS4148	10	0,058	0,58
1972086	IC1	ATMEGA328-AU	2	3,27	6,54
1864487RL	L1	470uH	10	0,21	2,1
1711841	L2	390nH	10	0,094	0,94
1457818RL	L3	33nH	10	0,046	0,46
1198386	L4	47nH	10	0,141	1,41
1357328	P1	CONN_6	10	0,14	1,4
2112438	P2	CONN_3X2	2	1,49	2,98
1755280	U1	TL2575-33IKV	2	2,46	4,92
1755200	U2	MRF49XA-I/ST	2	2,84	5,68
1842273	X1	10MHz	2	0,51	1,02
1131855	P1	CONN_2	10	0,28	2,8
1717004	P3	CONN_5	2	0,64	1,28
1829184RL	Q1, Q2, Q3, Q4	2N7002PW	12	0,042	0,504
1244527RL	U3, U4, U5, U6	HCPL-181-00BE	6	0,24	1,44
KOKONAISSUMMA:					**50,70 €**

Liite 4 – Projektin kokonaiskustannukset

Kohta	Kuvaus	Määrä	Hinta
LED	LED-valot	2	20
Osat	Osatilaukset, sis. useamman kappaleen osat	770	50,70
Levyt	20 kpl piirilevyjä	20	46,92
YHTEENSÄ			**117,62 €**

Liite 5 – Lähettimen ja vastaanottimen ohjelmistokoodi.

SPI.h

```
/*
 *  spi.h
 *  MRF49XA
 *
 *  Created by William Dillon on 11/1/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
 *
 */
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#include "hardware.h"


// Use these functions even with hardware SPI for portability
void spi_init(void);

// Read functions for 8 and 16 bit transfers
uint8_t  spi_read8(void);
uint16_t spi_read16(void);

// Write functions for 8 and 16 bit transfers
void spi_write8(uint8_t data, volatile uint8_t *enable_port, uint8_t enable_pin);
void spi_write16(uint16_t data, volatile uint8_t *enable_port, uint8_t enable_pin);

/* These defines must be present in a file called "hardware.h"
 *
 * You may copy and paste them into such a file and modify as needed.

#ifdef SOFTWARE_SPI

// These defines must be provided if using software SPI
#define SPI_MISO_PORTx   PORTB
#define SPI_MISO_PINx    PINB
#define SPI_MISO_DDRx    DDRB
#define SPI_MISO_BIT     7

#define SPI_MOSI_PORTx   PORTB
#define SPI_MOSI_PINx    PINB
#define SPI_MOSI_DDRx    DDRB
#define SPI_MOSI_BIT     0

#define SPI_SCLK_PORTx   PORTB
#define SPI_SCLK_PINx    PINB
#define SPI_SCLK_DDRx    DDRB
#define SPI_SCLK_BIT     2

#else

// These defines must be provided for hardware SPI.  If 16 bit transferes are
// used, it is imperitive that the hardware SS pin NOT be used.
#define SPI_MOSI_PORTx   PORTB
#define SPI_MOSI_PINx    PINB
#define SPI_MOSI_DDRx    DDRB
#define SPI_MOSI_BIT     0

#define SPI_SCLK_PORTx   PORTB
#define SPI_SCLK_PINx    PINB
#define SPI_SCLK_DDRx    DDRB
```

```
#define SPI_SCLK_BIT     2

#endif

 */
```

SPI.c

```
/*
 *  spi.c
 *  MRF49XA
 *
 *  Created by William Dillon on 11/1/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
 *
 */

#include "spi.h"

// This variable is vital for SW SPI, and 16 bit HW SPI.
volatile uint16_t spi_read_buffer;

volatile uint8_t mutex;

// This function assumes that the CS pins for peripherals are already "inactive"
void spi_init(void)
{
#ifndef SOFTWARE_SPI
            // Setup the hardware SPI interface
            SPCR    =  0x50;                        // set up SPI mode
            SPSR    =  0x01;                        // double speed operation
#endif

            // Setup the port pins for HW and SW SPI interface
            SPI_MISO_PORTx &= ~(1 << SPI_MISO_BIT); // Set port value to 0 (pullups off)
            SPI_MISO_DDRx  &= ~(1 << SPI_MISO_BIT); // Enable input on MISO
            SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
            SPI_MOSI_DDRx  |=  (1 << SPI_MOSI_BIT); // Enable output on MOSI
            SPI_SCLK_PORTx &= ~(1 << SPI_SCLK_BIT); // Set port value to 0
            SPI_SCLK_DDRx  |=  (1 << SPI_SCLK_BIT); // Enable output on SCLK
}

// In HW SPI mode, we still write to spi_read_buffer to enable 16 bit reads
uint8_t spi_read8()
{
            uint8_t     temp = spi_read_buffer;
            spi_read_buffer = 0x0000;
            return temp;
}

uint16_t spi_read16()
{
            return spi_read_buffer;
}

// To perform the SPI write functions, we have to do more work in the SW mode.
#ifdef SOFTWARE_SPI
void spi_write8(uint8_t data, volatile uint8_t *enable_port, uint8_t enable_pin)
{
            int i = 0;

            // Spin on the mutex
            while (mutex);
            mutex = 1;
```

```
		// Bring the enable pin low to enable SPI on the peripheral
		*enable_port &= ~(1 << enable_pin);
		spi_read_buffer = 0;

		for (i = 0; i < 8; i++) {
			if (data & 0x80) {
				SPI_MOSI_PORTx |=  (1 << SPI_MOSI_BIT); // Set port value to 1
			} else {
				SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
			}

			// Shift read buffer contents
			spi_read_buffer = spi_read_buffer << 1;

			// Data valid on rising edge of the clock
			SPI_SCLK_PORTx |=  (1 << SPI_SCLK_BIT); // Set clock value to 1

			// Read MISO
			if (bit_is_set(SPI_MISO_PINx, SPI_MISO_BIT)) {
				spi_read_buffer |= 0x01;
			}

			// Shift output data
			data = data << 1;

			// Data change on falling edge
			SPI_SCLK_PORTx &= ~(1 << SPI_SCLK_BIT); // Set clock value to 0

		}

		// Return the enable pin and MOSI to the inactive state
		SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
		*enable_port |= (1 << enable_pin);

		mutex = 0;
}

void spi_write16(uint16_t data, volatile uint8_t *enable_port, uint8_t enable_pin)
{
		int i = 0;

		// Spin on the mutex
		while (mutex);
		mutex = 1;

		// Bring the enable pin low to enable SPI on the peripheral
		*enable_port &= ~(1 << enable_pin);
		spi_read_buffer = 0;

		for (i = 0; i < 16; i++) {
			if (data & 0x8000) {
				SPI_MOSI_PORTx |=  (1 << SPI_MOSI_BIT); // Set port value to 1
			} else {
				SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
			}

			// Shift read buffer contents
			spi_read_buffer = spi_read_buffer << 1;
```

```
                        // Read MISO
                        if (bit_is_set(SPI_MISO_PINx, SPI_MISO_BIT)) {
                                spi_read_buffer |= 0x01;
                        }

                        // Data valid on rising edge of the clock
                        SPI_SCLK_PORTx |=  (1 << SPI_SCLK_BIT); // Set clock value to 1

                        // Shift input data
                        data = data << 1;

                        // Data change on falling edge
                        SPI_SCLK_PORTx &= ~(1 << SPI_SCLK_BIT); // Set clock value to 0

                }

                // Return the enable pin and MOSI to the inactive state
                SPI_MOSI_PORTx &= ~(1 << SPI_MOSI_BIT); // Set port value to 0
                *enable_port |= (1 << enable_pin);

                mutex = 0;
}

#else

void spi_write8(uint8_t data, volatile uint8_t *enable_port, uint8_t enable_pin)
{
                // Spin on the mutex
                while (mutex);
                mutex = 1;

                // Bring the enable pin low to enable SPI on the peripheral
                *enable_port &= ~(1 << enable_pin);

                SPDR = data;

                loop_until_bit_is_set(SPSR, SPIF);

                spi_read_buffer = (uint16_t)SPDR;

                // Return the enable pin to the inactive state
                *enable_port |= (1 << enable_pin);

                mutex = 0;
}

void spi_write16(uint16_t data, volatile uint8_t *enable_port, uint8_t enable_pin)
{
                // Spin on the mutex
                while (mutex);
                mutex = 1;

                // Bring the enable pin low to enable SPI on the peripheral
                *enable_port &= ~(1 << enable_pin);

                // Byte 1
                SPDR = (uint8_t)((data & 0xFF00) >> 8);

                loop_until_bit_is_set(SPSR, SPIF);

                spi_read_buffer = (((uint16_t)SPDR) << 8) & 0xFF00;

                // Byte 2
                SPDR = (uint8_t)(data & 0x00FF);
```

```
            loop_until_bit_is_set(SPSR, SPIF);

            spi_read_buffer |= ((uint16_t)SPDR) & 0x00FF;

            // Return the enable pin to the inactive state
            *enable_port |= (1 << enable_pin);

            mutex = 0;
}

#endif
```

mrf49xa.h

```
/*
 *  MRF49XA.h
 *  MRF49XA
 *
 *  Created by William Dillon on 11/1/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
 *
 *  Adapted from Microchip MRF49XA sample code (for register states)
 */

#include "MRF49XA_definitions.h"
#include "hardware.h"

/*****************************************************************************
 * This section of the header file includes the interface used by the user
 * application.  This includes an initialization routine, and functions to set
 * desired center frequency, baud rate, deviation, etc.
 *
 * In addition to initialization an configuration functions, functions are
 * provided for sending a packet, as well as receiving one.
 *
 *****************************************************************************/
void MRF_init(void);

uint8_t MRF_is_idle();
uint16_t MRF_statusRead(void);

// Packet structures
#define MRF_PAYLOAD_LEN 40                                      // the maximum payload size
// Space for preamble, sync, length and dummy
#define MRF_TX_PACKET_LEN            MRF_PAYLOAD_LEN + 5

typedef struct {
            uint8_t                  length;
            char                     payload[MRF_PAYLOAD_LEN];
} MRF_packet_t;

// Packet based functions
void MRF_transmit_packet(MRF_packet_t *packet);
MRF_packet_t* MRF_receive_packet();

void MRF_set_baud(uint16_t baud);     // Sets the baud rate in kbps

// Testing functions
void MRF_transmit_zero();
void MRF_transmit_one();
void MRF_transmit_alternating();
void MRF_packet_reflect();
```

```
void MRF_packet_generator();
void MRF_receive();
```

mrf49xa.c

```
/*
 *  MRF49XA.c
 *  MRF49XA
 *
 *  Created by William Dillon on 11/1/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
 *
 */

#define MRF49XA_C
#include "MRF49XA.h"
#include "spi.h"

#include <util/delay.h>

// Bit position:   7  6  5  4  3  2  1  0
// Normal modes:                 <X  X  X>
// Testing modes: <X  X  X>

// Normal modes
#define MRF_IDLE                                0x00        // Listening for packets,
nothing yet
#define MRF_TRANSMIT_PACKET 0x01      // Actively transmitting a packet
#define MRF_RECEIVE_PACKET  0x02      // Actively receiving a packet

// Testing modes
#define MRF_RECEIVE_ALL               0x20        // Not yet implemented
#define MRF_TRANSMIT_ZERO             0x40        // Transmits all '0'
#define MRF_TRANSMIT_ONE 0x80         // Transmits all '1'
#define MRF_TRANSMIT_ALT 0xC0         // Transmits alternating '01'

#define MRF_TX_TEST_MASK 0xC0         // Singles out spectrum test modes

static volatile uint8_t mrf_state;    // Defaults to idle

// There are 2 Rx_Packet_t instances, one for reading off the air
// and one for processing back in main.
MRF_packet_t Rx_Packet_a;
MRF_packet_t Rx_Packet_b;

// The hasPacket flag means that the finished_packet variable contains
// a fresh data packet.  The receiving_packet always contains space for
// received data.  If finished_packet isn't read before the a second packet
// comes
volatile uint8_t         hasPacket;               // Initialized to 0 by default
volatile MRF_packet_t *finished_packet;
volatile MRF_packet_t *receiving_packet;

volatile uint8_t         transmit_buffer[MRF_TX_PACKET_LEN];

volatile uint16_t        mrf_status;

#define RegisterSet(setting) spi_write16(setting, &MRF_CS_PORTx, MRF_CS_BIT)

uint16_t MRF_statusRead(void)
{
            spi_write16(0x0000, &MRF_CS_PORTx, MRF_CS_BIT);
```

```
            return spi_read16();
}

static inline uint8_t MRF_fifo_read(void)
{
            RegisterSet(MRF_RXFIFOREG);
            return spi_read8();
}

static void MRF_reset(void)
{
            RegisterSet(MRF_PMCREG);
            RegisterSet(MRF_FIFOSTREG_SET);
            RegisterSet(MRF_GENCREG_SET);
            RegisterSet(MRF_PMCREG | MRF_RXCEN);
            RegisterSet(MRF_GENCREG_SET | MRF_FIFOEN);
            RegisterSet(MRF_FIFOSTREG_SET | MRF_FSCF);

}

volatile uint8_t counter;
ISR(MRF_IRO_VECTOR, ISR_BLOCK)
{
            PORTD |= (1<<PORTD0);
            uint8_t bl;

            // Set the CS pin for the MRF low
            MRF_CS_PORTx &= ~(1 << MRF_CS_BIT);

            // Wait for the synchronizer
            _delay_us(1);

            // Perform operations
            if (bit_is_set(SPI_MISO_PINx, SPI_MISO_BIT)) {

                        switch (mrf_state) {
                                    case MRF_IDLE:
                                                bl = MRF_fifo_read();

                                                // The first byte is the length field
                                                if ((bl > 0) &&
                                                            (bl <= MRF_PAYLOAD_LEN)) {
                                                            mrf_state =
MRF_RECEIVE_PACKET;
                                                            receiving_packet->length =
bl;
                                                            counter = 0;

                                                            // The length field was obvi-
ously bogus, reset the module
                                                } else {
                                                            //PORTC &= ~(1 << 4);
                                                            counter = 0;
                                                            receiving_packet->length = 0;
                                                            MRF_reset();
                                                            return;
                                                }
                                                break;

                                    case MRF_TRANSMIT_PACKET:
                                                // Transmit the contents of the packet
                                                // (including preamble, sync, length, and
dummy byte)
```

```
                                        RegisterSet(MRF_TXBREG | trans-
mit_buffer[counter++]);

                                        // Derivation of the +6:
                                        // Preamble:      1 + (before packet)
                                        // Sync word:     2 + (before packet)
                                        // Length byte:   1 + (Packet byte 1, not
included in length)
                                        // Dummy byte:    1 + (Not actually trans-
mitted)
                                        // Pre-increment: 1 =
                                        // Total:         6
                                        if (counter >= (transmit_buffer[3] + 6)) {
                                                // Disable transmitter, ena-
ble receiver
                                                RegisterSet(MRF_PMCREG |
MRF_RXCEN);
                                                RegisterSet(MRF_GENCREG_SET |
MRF_FIFOEN);
                                                Register-
Set(MRF_FIFOSTREG_SET);
                                                RegisterSet(MRF_FIFOSTREG_SET
| MRF_FSCF);

                                                // Return the state
                                                mrf_state = MRF_IDLE;
                                                counter = 0;
                                        }

                                        break;

                                case MRF_RECEIVE_PACKET: // We've received at least
the size
                                                PORTD |= (1<<PORTD3);
                                                receiving_packet->payload[counter++] =
MRF_fifo_read();

                                                // End of packet?
                                                if (counter >= receiving_packet->length) {

                                                        // Reset the FIFO
                                                        Register-
Set(MRF_FIFOSTREG_SET);
                                                        RegisterSet(MRF_FIFOSTREG_SET
| MRF_FSCF);

                                                        // Swap packet structures
                                                        finished_packet = receiv-
ing_packet;
                                                        hasPacket = 1;
                                                        if (receiving_packet ==
&Rx_Packet_a) {
                                                                receiving_packet
=  &Rx_Packet_b;
                                                        } else {
                                                                receiving_packet
=  &Rx_Packet_a;
                                                        }

                                                        // Restore state
                                                        mrf_state = MRF_IDLE;
                                                        counter = 0;
                                                        receiving_packet->length = 0;
```

```
                                        }

                                        break;

                                case MRF_TRANSMIT_ZERO:
                                        RegisterSet(MRF_TXBREG | 0x0000);
                                        break;

                                case MRF_TRANSMIT_ONE:
                                        RegisterSet(MRF_TXBREG | 0x00FF);
                                        break;

                                case MRF_TRANSMIT_ALT:
                                        RegisterSet(MRF_TXBREG | 0x00AA);
                                        break;

                                default:
                                        break;
                        }
                }

                // There was no FIFO flag, just leave.
                else {
                        MRF_CS_PORTx |=  (1 << MRF_CS_BIT);
                        return;
                }
}
void MRF_init(void)
{

                // Enable the IRO pin as input w/o pullup
                MRF_IRO_PORTx &= ~(1 << MRF_IRO_BIT);
                MRF_IRO_DDRx  &= ~(1 << MRF_IRO_BIT);

                // Enable the CS line as output with high value
                MRF_CS_PORTx  |=  (1 << MRF_CS_BIT);
                MRF_CS_DDRx   |=  (1 << MRF_CS_BIT);

                // Enable the FSEL as output with high value (default, low for receive)
                MRF_FSEL_PORTx |= (1 << MRF_FSEL_BIT);
                MRF_FSEL_DDRx |=  (1 << MRF_FSEL_BIT);

                // Enable the External interrupt for the IRO pin (falling edge)
                MRF_INT_SETUP();

                // configuring the MRF49XA radio
                RegisterSet(MRF_FIFOSTREG_SET);       // Set 8 bit FIFO interrupt count
                RegisterSet(MRF_FIFOSTREG_SET | MRF_FSCF);        // Enable sync. latch
                RegisterSet(MRF_GENCREG_SET);         // From the header: 434mhz, 10pF
                RegisterSet(MRF_AFCCREG_SET);
                RegisterSet(MRF_CFSREG_SET);
                RegisterSet(MRF_DRSREG_SET);                                        // Approx. 9600
baud
                RegisterSet(MRF_PMCREG | MRF_CLKODIS);            // Shutdown everything
                RegisterSet(MRF_RXCREG_SET);
                RegisterSet(MRF_TXCREG_SET);
                RegisterSet(MRF_BBFCREG | MRF_ACRLC | (4 & MRF_DQTI_MASK));

                // antenna tuning on startup
                RegisterSet(MRF_PMCREG | MRF_CLKODIS | MRF_TXCEN); // turn on the transmitter
                _delay_ms(5);                           // wait for oscillator to stablize
                // end of antenna tuning

                // turn off transmitter, turn on receiver
```

```
		RegisterSet(MRF_PMCREG | MRF_CLKODIS | MRF_RXCEN);
		RegisterSet(MRF_GENCREG_SET | MRF_FIFOEN);
		RegisterSet(MRF_FIFOSTREG_SET);
		RegisterSet(MRF_FIFOSTREG_SET | MRF_FSCF);


		// Setup the packet pointers
		receiving_packet = &Rx_Packet_a;

		// Dummy read of status registers to clear Power on reset flag
		mrf_status = MRF_statusRead();

		// Enable interrupt last, just in case they're already globally enabled
		MRF_INT_MASK();
}


// Testing functions
void MRF_transmit_zero()
{
		uint8_t wait = 1;
		do
		{
			cli(); // disable interrupts for the checking
			if (mrf_state == MRF_IDLE) {
				mrf_state = MRF_TRANSMIT_ZERO;
				wait = 0;
				sei();		// State change done, re-initialize inter-
rupts.
			}		else
			{
				sei();
				_delay_ms(1);  // Wait for a while
			}
		} while (wait);

		mrf_state = MRF_TRANSMIT_ZERO;

		// Enable the TX Register
		RegisterSet(MRF_GENCREG_SET | MRF_TXDEN);

		// The transmit register is filled with 0xAAAA, we want it to be zeros
		RegisterSet(MRF_TXBREG | 0x0000);

		// Enable the transmitter
		RegisterSet(MRF_PMCREG | MRF_CLKODIS | MRF_TXCEN);

		// Upon completion of a byte !IRO should toggle
		return;
}

void MRF_transmit_one()
{
		uint8_t wait = 1;
		do
		{
			cli(); // disable interrupts for the checking
			if (mrf_state == MRF_IDLE) {
				mrf_state = MRF_TRANSMIT_ONE;
				wait = 0;
				sei();		// State change done, re-initialize inter-
rupts.
			}		else
			{
```

```
                                    sei();
                                    _delay_ms(1);  // Wait for a while
                        }
            } while (wait);

            mrf_state = MRF_TRANSMIT_ONE;

            // Enable the TX Register
            RegisterSet(MRF_GENCREG_SET | MRF_TXDEN);

            // The transmit register is filled with 0xAAAA, we want it to be ones
            RegisterSet(MRF_TXBREG | 0x00FF);

            // Enable the transmitter
            RegisterSet(MRF_PMCREG | MRF_CLKODIS | MRF_TXCEN);

            // Upon completion of a byte !IRO should toggle
            return;
}

void MRF_transmit_alternating()
{
            uint8_t wait = 1;
            do
            {
                        cli(); // disable interrupts for the checking
                        if (mrf_state == MRF_IDLE) {
                                    mrf_state = MRF_TRANSMIT_ALT;
                                    wait = 0;
                                    sei();      // State change done, re-initialize inter-
rupts.
                        }           else
                        {
                                    sei();
                                    _delay_ms(1);  // Wait for a while
                        }
            } while (wait);

            // Enable the TX Register
            RegisterSet(MRF_GENCREG_SET | MRF_TXDEN);

            // The transmit register is filled with 0xAAAA, we can leave it alone

            // Enable the transmitter
            RegisterSet(MRF_PMCREG | MRF_CLKODIS | MRF_TXCEN);

            // Upon completion of a byte !IRO should toggle
            return;
}

MRF_packet_t* MRF_receive_packet()
{

            if (hasPacket) {
                        hasPacket = 0;
                        return finished_packet;
            } else {
                        return 0;
            }
}

uint8_t MRF_is_idle(void)
{
            if (mrf_state == MRF_IDLE) {
```

```
                return 1;
        } else {
                return 0;
        }
}

void MRF_transmit_packet(MRF_packet_t *packet)
{
        uint8_t     i;
        uint8_t wait = 1;

        // We can check, without synchronization
        // (because it doesn't change in the ISR)
        // Whether we're in a testing mode.  If so, simply return.
        // This is important, so we don't hard lock.
        if (mrf_state & MRF_TX_TEST_MASK) {
                return;
        }

        // Wait for the module to be idle (this is cheezy synchronization)
        do {
                cli();      // Disable interrupts, this is a critical section
                if (mrf_state == MRF_IDLE) {
                        mrf_state = MRF_TRANSMIT_PACKET;
                        wait = 0;
                        sei();      // Atomic operation complete, reenable
interrupts
                } else {
                        sei();      // Atomic operation complete, reenable
interrupts
                        //_delay_ms(1);          // This should be roughly 1
byte period
                }
        } while (wait);

        // Initialize the constant parts of the transmit buffer
        counter = 0;
        transmit_buffer[0] = 0xAA;
        transmit_buffer[1] = 0x2D;
        transmit_buffer[2] = 0xD4;
        transmit_buffer[3] = packet->length;

        // Copy the packet contents into the buffer
        for (i = 0; i < packet->length; i++) {
                transmit_buffer[i + 4] = packet->payload[i];
        }

        RegisterSet(MRF_PMCREG);                                            //
Turn everything off
        RegisterSet(MRF_GENCREG_SET | MRF_TXDEN);         // Enable TX FIFO
        // Reset value of TX FIFO is 0xAAAA

        RegisterSet(MRF_PMCREG | MRF_TXCEN);              // Begin transmitting
        // Everything else is handled in the ISR
}
```

hardware.h

```
/*
 *  hardware.h
 *  MRF49XA
 *
 *  Created by William Dillon on 11/1/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
```

```
 *
 */

#ifndef HARDWARE_H
#define HARDWARE_H

#define F_CPU 1000000UL  // 8 Mhz internal clock

#include <avr/io.h>
#include <avr/interrupt.h>

/* These defines are included for the SPI library
 *
 */
#define SOFTWARE_SPI

#define SPI_MISO_PORTx   PORTB
#define SPI_MISO_PINx    PINB
#define SPI_MISO_DDRx    DDRB
#define SPI_MISO_BIT     4

#define SPI_MOSI_PORTx   PORTB
#define SPI_MOSI_PINx    PINB
#define SPI_MOSI_DDRx    DDRB
#define SPI_MOSI_BIT     3

#define SPI_SCLK_PORTx   PORTB
#define SPI_SCLK_PINx    PINB
#define SPI_SCLK_DDRx    DDRB
#define SPI_SCLK_BIT     5

/* These defines set the chip select line and interrupt for the MRF module
 *
 */
#define MRF_CS_PINx                 PINB
#define MRF_CS_PORTx     PORTB
#define MRF_CS_DDRx                 DDRB
#define MRF_CS_BIT                  2

#define MRF_IRO_PINx     PIND
#define MRF_IRO_PORTx    PORTD
#define MRF_IRO_DDRx     DDRD
#define MRF_IRO_BIT                 2
#define MRF_IRO_VECTOR   INT0_vect



#define MRF_FSEL_PORTx   PORTC
#define MRF_FSEL_DDRx    DDRC
#define MRF_FSEL_BIT     1

// These are the settings for the external interrupt pins
#define MRF_INT_SETUP()  EICRA = (1 << ISC01)
#define MRF_INT_MASK()   EIMSK = (1 << INT0)

/*********************************************************************************
 * These defines set either the soldered-on characteristics of the MRF module,
 * or they are specific to this application. This includes the frequency band,
 * clock output, and pin settings.
 *
 *********************************************************************************/
// Set the module to 434 Mhz band, with 10pF series capactance crystal
#define MRF_GENCREG_SET             (MRF_GENCREG | (MRF_LCS & MRF_LCS_MASK) | MRF_FBS_434)
```

```
#define MRF_AFCCREG_SET             (MRF_AFCCREG | MRF_AUTOMS_INDP | MRF_ARFO_3to4 |
MRF_HAM | MRF_FOREN | MRF_FOFEN)
#define MRF_PLLCREG_SET             (MRF_PLLCREG | MRF_CBTC_5p)
#define     MRF_CFSREG_SET                      (MRF_CFSREG | (MRF_FREQB &
MRF_FREQB_MASK))
#define MRF_RXCREG_SET              (MRF_RXCREG | MRF_RXBW_67K | MRF_DRSSIT_103db)
#define MRF_TXCREG_SET              (MRF_TXCREG | MRF_MODBW_30K | MRF_OTXPWR_17D5)
#define MRF_DRSREG_SET              (MRF_DRSREG | (MRF_DRPV_VALUE & MRF_DRPV_MASK))
#define MRF_FIFOSTREG_SET           (MRF_FIFORSTREG | MRF_DRSTM | 0x0080)

#endif
```

mrf49xa_definitions.h

```
/*
 *  MRF49XA_definitions.h
 *  MRF49XA
 *
 *  Created by William Dillon on 11/2/10.
 *  Copyright 2010 Oregon State University. All rights reserved.
 *
 */

/*******************************************************************************
 * Bit definitions for MRF registers
 *
 * These defines are provided for use configuring the MRF49XA module.  The
 * register address is the first define in each section, and must be written
 * to the MOSI pins to access the register.
 *
 * To set register states, bit-wise or the address with the desired set bits
 * for the register.
 *
 * To read the register states, you must also write to any bits that are R/W
 * Luckily, the only register worth reading is STSREG, and has no settable bits
 *
 * Some registers contain values that need to be derived, or select a range of
 * settings.  Each of these has their own section containing the equation
 * and/or individual defines for each setting.
 *
 ******************************************************************************/
#pragma mark Status read register
#define MRF_STSREG                  0x0000                  // Status read register ad-
dress
#define MRF_TXRXFIFO     0x8000                 // FIFO state (1 = ready)
#define MRF_POR                                 0x4000                  // Power-on-Reset
flag
#define MRF_TXOWRXOF     0x2000                 // Underrun/Overwrite/Overflow
#define MRF_WUTINT                  0x1000                  // Wakeup timer overflow in-
terrupt
#define MRF_LCEXINT                 0x0800                  // Logic change interrupt
#define MRF_LBTD                    0x0400                  // Low Battery threshold de-
tect
#define MRF_FIFOEM                  0x0200                  // Receiver FIFO empty (1 =
empty)
#define MRF_ATTRSSI                 0x0100                  // Antenna Tuning and RSSI
indicator
#define MRF_DQDO                    0x0080                  // Data Quality De-
tect/Indicate output
#define MRF_CLKRL                   0x0040                  // Clock recovery lock bit
```

```
#define MRF_AFCCT                   0x0020                  // Automatic frequency con-
trol cycle toggle
#define     MRF_OFFSV               0x0010                  // Measured frequency offset
of AFC cycle
#define MRF_OFFSET_MASK  0x000F                  // Value of the AFC offset (see datasheet)

/*************************************************************************
 * Convenience definitions for Automatic frequency control configuration reg.
 *
 * These defines are provided for use configuring the MRF49XA module.
 *
 * The AutoMS field sets the mode, and ARFO sets the allowable tuning range.
 * The MFCS, HAM, FOREN, and FOFEN bits set the manual control strobe, high
 * accuracy mode, frequency offset register, and frequency offset respectively
 *
 *************************************************************************/
#pragma mark Automatic frequency control configuration register
#define MRF_AFCCREG                 0xC400                  // AFC configuration register
address

// Automatic frequency mode selection
#define MRF_AUTOMS_INDP  0x00C0                  // Offset independent for state of DIO
sig.
#define MRF_AUTOMS_RECV  0x0080                  // Offset only during receive
#define MRF_AUTOMS_ONCE  0x0040                  // Offset once after power-cycle
#define MRF_AUTOMS_OFF   0x0000                  // Auto mode off

// Allowable tuning range selection
#define MRF_ARFO_3to4    0x0030                  // +3 to -4 Fres (tuning bits)
#define MRF_ARFO_7to8    0x0020                  // +7 to -8 Fres
#define MRF_ARFO_15to16  0x0010                  // +15 to -16 Fres
#define MRF_ARFO_unlim   0x0000                  // Unlimited

// Other flags in automatic tuning set register
#define MRF_MFCS                    0x0008                  // Manual Frequency control
strobe
#define MRF_HAM                                 0x0004                  // High accuracy
mode
#define MRF_FOREN                   0x0002                  // Frequency Offset Register
Enable
#define MRF_FOFEN                   0x0001                  // Frequency Offset Enable

/*************************************************************************
 * Convenience definitions for Transmit byte register
 *
 * This register allows for filling the transmit register one byte at a time.
 * The top byte of the command is the address, the bottom byte is the data.
 *************************************************************************/
#pragma mark Transmit Byte Register
#define MRF_TXBREG                  0xB800                  // Transmit byte register
address
#define MRF_TXDB_MASK    0x00FF                  // Transmit byte mask


#pragma mark Baseband Filter Configuration Register
#define MRF_BBFCREG                 0xC228                  // Baseband filter config.
register address
#define MRF_ACRLC                   0x0080                  // Automatic clock recovery
lock control
#define MRF_MCRLC                   0x0040                  // Manual clock recovery lock
control
#define MRF_FTYPE                   0x0010                  // Filter type (0 = digital,
1 = Ext. RC)
#define MRF_DQTI_MASK    0x0007                  // Data quality threshold indicator
```

```
#pragma mark Receiver FIFO read register
#define MRF_RXFIFOREG    0xB000                  // Receiver FIFO read register
#define MRF_RXDB_MASK    0x00FF                  // Receive data byte mask

/*******************************************************************************
 * Convenience definitions for FIFO and Reset mode configuration register
 *
 * These defines are provided for use configuring the MRF49XA module.
 *
 * The FFBC field of this register sets the number of bits to receive before
 * a FIFO full interrupt is generated.  A reasonable number is 8.  The maximum
 * is 15 bits.
 *
 *******************************************************************************/
#pragma mark FIFO and reset mode configuration register
#define MRF_FIFORSTREG   0xCA00                  // FIFO/Reset mode configuration register
#define MRF_FFBC_MASK    0x00F0                  // FIFO Fill Bit Count (maximum = 15)
#define MRF_SYCHLEN                  0x0008                  // Synchronous Character
length
#define MRF_FFSC                     0x0004                  // FIFO Fill Start condition
#define MRF_FSCF                     0x0002                  // FIFO Synchronous Character
fill
#define MRF_DRSTM                    0x0001                  // Disable (sensitive) reset
mode

#pragma mark Synchrnonous byte configuration register
#define MRF_SYNBREG                  0xCE00                  // Synchrnous byte config.
register address
#define MRF_SYNCB                    0x00FF                  // Sync byte configuration

#pragma mark Power Management Configuration Register
#define MRF_PMCREG                   0x8200                  // Power Mgmt. Config. Regis-
ter address
#define MRF_RXCEN                    0x0080                  // Receiver chain enable
#define MRF_BBCEN                    0x0040                  // Baseband chain enable
#define MRF_TXCEN                    0x0020                  // Transmitter chain enable
#define MRF_SYNEN                    0x0010                  // Synthesier enable
#define MRF_OSCEN                    0x0008                  // Oscillator enable
#define MRF_LBDEN                    0x0004                  // Low Battery Detector Ena-
ble
#define     MRF_WUTEN                0x0002                  // Wakeup timer enable
#define MRF_CLKODIS                  0x0001                  // Clock output disable

#pragma mark Wakeup timer value set register
#define MRF_WTSREG                   0xE000                  // Wakeup timer value regis-
ter address
#define MRF_WTEV_MASK    0x1F00                  // Wakeup timer exponential value
#define MRF_WTMV_MASK    0x00FF                  // Wakeup timer multiplier exponential
value

#pragma mark Duty cycle value set register
#define MRF_DCSREG                   0xC800                  // Duty cycle value set reg-
ister
#define MRF_DCMV_MASK    0x00FE                  // Duty cycle multiplier value mask
#define MRF_DCMEN                    0x0001                  // Duty cycle mode enable

#pragma mark Battery threshold and clock output value set register
#define MRF_CSREG                    0xC000                  // Battery thres. and clock
set reg. address
#define MRF_COFSB_MASK   0x00E0                  // Clock output frequency set
#define MRF_LBDVB_MASK   0x0008                  // Low battery detect value

/*******************************************************************************
```

```
 * Convenience definitions for PLL Configuration register
 *
 * These defines are provided for use configuring the MRF49XA module.
 *
 * The default setting is for 5-10 MHz output clock, Phase detector delay
 * disabled, PLL dithering disabled, and PLL bandwidth set to -102dBc/Hz
 * which is appropriate for higher bit rates.
 *
 * > 90kbps -> -102dBc/Hz (max 256kbps)
 * < 90kbps -> -107dBc/Hz (max 86.2kbps)
 *
 *****************************************************************************/
#pragma mark PLL Configuration register
#define MRF_PLLCREG                 0xCC12                  // PLL configuration register

// Clock buffer time control settings
#define MRF_CBTC_5p                 0x0060                  // Clock buffer 5-10 Mhz
#define MRF_CBTC_3                  0x0040                  // Clock buffer 3.3 Mhz
#define MRF_CBTC_2p                 0x0020                  // Clock buffer > 2.5 Mhz
#define MRF_CBTC_2m                 0x0000                  // Clock buffer < 2.5 Mhz

#define MRF_PDDS                    0x0008                  // Phase detector delay
#define MRF_PLLDD                   0x0004                  // PLL Dithering Disable
#define MRF_PLLBWB                  0x0001                  // PLL Bandwidth (102dBc/Hz)

/*****************************************************************************
 * Convenience definitions for band setting
 *
 * These defines are provided for use configuring the MRF49XA module.
 * Select the frequency band for the given hardware by uncommenting the
 * appropriate line.  Set the crystal load capactiance using the MRF_XTAL_LD_CAP
 * value.
 *
 * The load capactiance is given by the following equation:
 *
 * Cap pF = 8.5 + (LCS / 2)
 * LCS = (10 - 8.5) * 2
 *
 * For 10pF: LCS = (10 - 8.5) * 2 = 1.5 * 2 = 3
 *
 *****************************************************************************/
#pragma mark General Configuration Register
#define MRF_GENCREG                 0x8000                  // General configuration reg-
ister addres
#define MRF_TXDEN                   0x0080                  // TX Data Register enable
bit
#define MRF_FIFOEN                  0x0040                  // FIFO enable bit
#define MRF_FBS_MASK     0x0030                 // Mask for the band selection
#define MRF_LCS_MASK     0x000F                 // Mask for the crystal load capactiance

// 10pF Crystal load capacitance
#define MRF_LCS                                 15                                  //
Crystal Load capactiance

// Frequency band settings
#define MRF_FBS_434                 0x0010                  // 434 mHz band
#define MRF_FBS_868                 0x0020                  // 868 mHz band
#define MRF_FBS_915                 0x0030                  // 915 mHz band

/*****************************************************************************
 * Convenience definitions for center frequency
 *
 * These defines are provided for use configuring the MRF49XA module.
 * Select the center frequency using the following equation:
```

```
 *
 * Fo = 10 * FA1 * (FA0 + Fval/4000)
 * Fval = Decimal value of FREQB, 96 < Fval < 3903
 *
 * Choose FA1 anf FA0 using the following table according to band:
 *
 *                      Range                   FA1                                     FA0
 *                      434                                 1
           43
 *                      868                                 2
           43
 *                      915                                 3
           30
 *
 * Fo = 10 * (43 + Fval/4000) = 10*43 + 10*(Fval/4000) = 430 + Fval/400
 *
 * Fval = (Fo - 430) * 400 = (432.10 - 430) * 400 = 2.1 * 400 = 840
 *
 ********************************************************************************/
#pragma mark Center Frequency Value Set Register
#define MRF_CFSREG                  0xA000                  // Center Frequency value
register address
#define MRF_FREQB_MASK   0x0FFF                 // Center Frequency value (see datasheet)

// Frequency setting for 432.10 mHz
#define MRF_FREQB                   840

/********************************************************************************
 * Convenience definitions for receiver control register
 *
 * These defines are provided for use configuring the MRF49XA module.
 * It is possible to set the values for Data Integrity integration time,
 * Baseband bandwidth values, and LNA gain.
 *
 * Use the MRF_RXCREG_SET define for Microchip defaults
 *
 ********************************************************************************/
#pragma mark Receive control register
#define MRF_RXCREG                  0x9000                  // Receive control register
address
#define MRF_FINTDIO                 0x0400                  // Function interrupt/dio
output
#define MRF_DIORT_MASK   0x0300                 // Data indicator response time
#define MRF_RXBW_MASK    0x00E0                 // Receiver baseband bandwidth
#define MRF_RXLNA_MASK   0x0018                 // Receiver LNA gain
#define MRF_DRSSIT_MASK 0x0007                  // Digital RSSI threshold

// Data indicator output response time
#define MRF_DIORT_CONT   0x0300                 // Continuous
#define MRF_DIORT_SLOW   0x0200                 // Slow
#define MRF_DIORT_MED    0x0100                 // Medium
#define MRF_DIORT_FAST   0x0000                 // Fast

// Receiver Baseband bandwidth
#define MRF_RXBW_67K     0x00C0                 // Receiver Bandwidth 67 khz
#define MRF_RXBW_134K    0x00A0                 // Receiver Bandwidth 134 khz
#define MRF_RXBW_200K    0x0080                 // Receiver Bandwidth 200 khz
#define MRF_RXBW_270K    0x0060                 // Receiver Bandwidth 270 khz
#define MRF_RXBW_340K    0x0040                 // Receiver Bandwidth 340 khz
#define MRF_RXBW_400K    0x0020                 // Receiver Bandwidth 400 khz

// Receiver LNA Gain
#define MRF_RXLNA_20DB   0x0018                 // LNA Gain -20dB
#define MRF_RXLNA_14DB   0x0010                 // LNA Gain -14dB
```

```
#define MRF_RXLNA_6DB    0x0008                  // LNA Gain -6dB
#define MRF_RXLNA_0DB    0x0000                  // LNA Gain  0dB

// Digital RSSI threshold
#define MRF_DRSSIT_73db  0x0005                  // -73dB Threshold
#define MRF_DRSSIT_79db  0x0004                  // -79dB Threshold
#define MRF_DRSSIT_85db  0x0003                  // -85dB Threshold
#define MRF_DRSSIT_91db  0x0002                  // -91dB Threshold
#define MRF_DRSSIT_97db  0x0001                  // -97dB Threshold
#define MRF_DRSSIT_103db 0x0000                  // -103dB Threshold

/*********************************************************************************
 * Convenience definitions for transmitter control register
 *
 * These defines are provided for use configuring the MRF49XA module.
 * Select the modulation bandwidth and output power by bit-wise oring them
 * with the register address.
 *
 * Use the MRF_TXCREG_SET for Microchip defaults
 *
 *********************************************************************************/
#pragma mark Transmit Configuration register
#define MRF_TXCREG                   0x9800                  // Transmit configuration
register address
#define MRF_MODPLY                   0x0100                  // Modulation polarity
#define MRF_MODBW_MASK   0x00F0                  // Modulation bandwidth
#define MRF_OTXPWR_MASK 0x0007                   // Output transmit power

// Modulation bandwidth settings
#define MRF_MODBW_240K   0x00F0                  // 240kHz modulation bandwidth
#define MRF_MODBW_225K   0x00E0                  // 240kHz modulation bandwidth
#define MRF_MODBW_210K   0x00D0                  // 240kHz modulation bandwidth
#define MRF_MODBW_195K   0x00C0                  // 240kHz modulation bandwidth
#define MRF_MODBW_180K   0x00B0                  // 240kHz modulation bandwidth
#define MRF_MODBW_165K   0x00A0                  // 240kHz modulation bandwidth
#define MRF_MODBW_150K   0x0090                  // 240kHz modulation bandwidth
#define MRF_MODBW_135K   0x0080                  // 240kHz modulation bandwidth
#define MRF_MODBW_120K   0x0070                  // 240kHz modulation bandwidth
#define MRF_MODBW_105K   0x0060                  // 240kHz modulation bandwidth
#define MRF_MODBW_90K    0x0050                  // 240kHz modulation bandwidth
#define MRF_MODBW_75K    0x0040                  // 240kHz modulation bandwidth
#define MRF_MODBW_60K    0x0030                  // 240kHz modulation bandwidth
#define MRF_MODBW_45K    0x0020                  // 240kHz modulation bandwidth
#define MRF_MODBW_30K    0x0010                  // 240kHz modulation bandwidth
#define MRF_MODBW_15K    0x0000                  // 240kHz modulation bandwidth

// Output power settings
#define MRF_OTXPWR_17D5  0x0007                  // -17.5dB Transmit Output Power
#define MRF_OTXPWR_15D0  0x0006                  // -15.0dB Transmit Output Power
#define MRF_OTXPWR_12D5  0x0005                  // -12.5dB Transmit Output Power
#define MRF_OTXPWR_10D5  0x0004                  // -10.5dB Transmit Output Power
#define MRF_OTXPWR_7D5   0x0003                  // -7.5dB Transmit Output Power
#define MRF_OTXPWR_5D0   0x0002                  // -5.0dB Transmit Output Power
#define MRF_OTXPWR_2D5   0x0001                  // -2.5dB Transmit Output Power
#define MRF_OTXPWR_0     0x0000                  //  0dB Transmit Output Power

/*********************************************************************************
 * Convenience definitions for data rate value set register
 *
 * These defines are provided for use configuring the MRF49XA module.
 *
 * To calculate the DRPV value using the following equation:
 *
 * DRPV = 10000/[29 * (1 + DPRE * 7) * DREx] - 1
```

```
 *
 * Where DPRE is either 1 or 0, and DREx is the desired data rate (kbps)
 *
 * 10000/[29 * 1 * 9.6] - 1 = 10000/278.4 - 1 = 35.9 - 1 = 34.9 = 35
 *
 ******************************************************************************/
#pragma mark Data Rate Value set register
#define MRF_DRSREG                  0xC600                  // Data rate value set regis-
ter address
#define MRF_DRPE                    0x0080                  // Data rate prescaler enable
#define MRF_DRPV_MASK    0x007F                  // Data rate value mask

// Settings for approx. 9600 baud
#define MRF_DRPE_ENABLE  0x0080                  // Data rate prescaler enable bit
#define MRF_DRPV_VALUE   35                              // Derived value for 9579
baud
```

Epsilon.c (sisältää molemmat vastaanottimen ja lähettimen ohjelmakoodit)

```
/*
 * Epsilon.c
 *
 * Created: 20.3.2013 15:33:51
 *  Author: Timo Pekkanen
 *
 * This is the SW for wireless tail lights project Epsilon.
 * This SW covers the receiver operations.
 */

#include "spi.h"
#include "mrf49xa.h"
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define F_CPU 1000000UL

#define RegisterSet(setting) spi_write16(setting, &MRF_CS_PORTx, MRF_CS_BIT)
#define VON 1
#define VOFF 0
#define STATEJ 1
#define STATEVV 2
#define STATEOV 3
#define STATEJOV 4
#define STATEJVV 5
#define STATEJSOS 113
#define STATESOS 112
#define STATENOSIGNAL 99

volatile uint8_t state = STATENOSIGNAL, icounter = 0;

ISR(PCINT1_vect, ISR_BLOCK)
{
//          PORTD ^= (1<<PORTD0);
//          if (bit_is_set(PINC, PORTC2)) {
                        uint8_t temp, temp2;
                        RegisterSet(MRF_RXFIFOREG);
                        temp = spi_read8();
                        if (temp == 2) {
                                    temp = 0;
                                    RegisterSet(MRF_RXFIFOREG);
                                    temp = spi_read8();
```

```
                                        switch (temp) {
                                                case 0xae:  // Jarru + Oikea vilkku
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0xae) {
                                                                state = STATEJOV;

                //MRF_statusRead();
                                                                //_delay_ms(100);
                                                        }
                                                break;
                                                        case 0xbe:   // Jarru
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0xbe) {
                                                                state = STATEJ;

                //MRF_statusRead();
                                                                //_delay_ms(100);
                                                        }
                                                break;
                                                case 0xce:  // Jarru + vasen vilkku
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0xce) {
                                                                state = STATEJVV;

                //MRF_statusRead();
                                                                //_delay_ms(100);
                                                        }
                                                break;
                                                case 0xde:  // Vasen Vilkku
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0xde) {
                                                                state = STATEVV;

                //MRF_statusRead();
                                                                //_delay_ms(100);

                                                        }
                                                break;
                                                case 0xfe:  // Oikea vilkku
                                                        icounter = 0;
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0xfe) {
                                                                state = STATEOV;

                //MRF_statusRead();
                                                                //_delay_ms(100);
                                                        }
                                                break;
                                                case 0x0c:
                                                        icounter = 0;
                                                        RegisterSet(MRF_RXFIFOREG);
                                                        temp2 = spi_read8();
                                                        if (temp2 == 0x0c) {
                                                                state =
STATENOSIGNAL;
                                                        }
                                                break;
                                                case 0x99:
                                                        icounter = 0;
```

```
                                                                RegisterSet(MRF_RXFIFOREG);
                                                                temp2 = spi_read8();
                                                                if (temp2 == 0x99) {
                                                                        state =
STATEJSOS;
                                                                }

                                                        break;
                                                        case 0x77:
                                                                icounter = 0;
                                                                RegisterSet(MRF_RXFIFOREG);
                                                                temp2 = spi_read8();
                                                                if (temp2 == 0x77) {
                                                                        state = STATESOS;
                                                                }

                                                        break;
                                                        default:
                                                                //MRF_statusRead();
                                                                //_delay_ms(10);
                                                                break;
                                        }
                                }
//          }
}
int main(void)
{
/*          int tauko = 300;
            int i = tauko;
            MRF_packet_t jarru, vasen_vilkku, oikea_vilkku, jvv, jov, tyhja, jsos, sos;
            jarru.length = 2;
            jarru.payload[0] = 0xbe;
            jarru.payload[1] = 0xbe;
            vasen_vilkku.length = 2;
            vasen_vilkku.payload[0] = 0xde;
            vasen_vilkku.payload[1] = 0xde;
            oikea_vilkku.length = 2;
            oikea_vilkku.payload[0] = 0xfe;
            oikea_vilkku.payload[1] = 0xfe;
            jvv.length = 2;
            jvv.payload[0] = 0xce;
            jvv.payload[1] = 0xce;
            jov.length = 2;
            jov.payload[0] = 0xae;
            jov.payload[1] = 0xae;
            tyhja.length = 2;
            tyhja.payload[0] = 0x0c;
            tyhja.payload[1] = 0x0c;
            jsos.length = 2;
            jsos.payload[0] = 0x99;
            jsos.payload[1] = 0x99;
            sos.length = 2;
            sos.payload[0] = 0x77;
            sos.payload[1] = 0x77;
            DDRD &= ~(1<<PORTD0);
            PORTD &= ~(1<<PORTD0);
            DDRD &= ~(1<<PORTD1);
            PORTD &= ~(1<<PORTD1);
            DDRD &= ~(1<<PORTD2);
            PORTD &= ~(1<<PORTD2);

            // Initialize the SPI ports and operations
            spi_init();
            // Delay from startup to make sure MRF49XA is ready
```

```
            _delay_ms(50);
            // Initialize the MRF49XA ready for operations
            MRF_init();

            while (1)
            {
                        if (bit_is_set(PIND, PORTD0) && (bit_is_clear(PIND, PORTD1)) &&
(bit_is_clear(PIND, PORTD2))) {
                                    state = STATEJ;
                        }
                        if (bit_is_clear(PIND, PORTD0) && bit_is_set(PIND, PORTD1) &&
bit_is_clear(PIND, PORTD2)) {
                                    state = STATEOV;
                        }
                        if (bit_is_clear(PIND, PORTD0) && bit_is_clear(PIND, PORTD1) &&
bit_is_set(PIND, PORTD2)) {
                                    state = STATEVV;
                        }
                        if (bit_is_set(PIND, PORTD0) && bit_is_clear(PIND, PORTD1) &&
bit_is_set(PIND, PORTD2)) {
                                    state = STATEJVV;
                        }
                        if (bit_is_set(PIND, PORTD0) && bit_is_set(PIND, PORTD1) &&
bit_is_clear(PIND, PORTD2)) {
                                    state = STATEJOV;
                        }
                        if (bit_is_set(PIND, PORTD0) && bit_is_set(PIND, PORTD1) &&
bit_is_set(PIND, PORTD2)) {
                                    state = STATEJSOS;
                        }
                        if (bit_is_clear(PIND, PORTD0) && bit_is_set(PIND, PORTD1) &&
bit_is_set(PIND, PORTD2)) {
                                    state = STATESOS;
                        }
                        if (bit_is_clear(PIND, PORTD0) && bit_is_clear(PIND, PORTD1) &&
bit_is_clear(PIND, PORTD2)) {
                                    state = STATENOSIGNAL;
                        }

                        switch (state) {
                                    case STATEJ:
                                                MRF_transmit_packet(&jarru);
                                                _delay_ms(10);
                                                break;
                                    case STATEOV:
                                                MRF_transmit_packet(&oikea_vilkku);
                                                _delay_ms(10);
                                                break;
                                    case STATEVV:
                                                MRF_transmit_packet(&vasen_vilkku);
                                                _delay_ms(10);
                                                break;
                                    case STATEJOV:
                                                MRF_transmit_packet(&jov);
                                                _delay_ms(10);
                                                break;
                                    case STATEJVV:
                                                MRF_transmit_packet(&jvv);
                                                _delay_ms(10);
                                                break;
                                    case STATEJSOS:
                                                MRF_transmit_packet(&jsos);
                                                _delay_ms(10);
                                                break;
```

```
                        case STATESOS:
                                MRF_transmit_packet(&sos);
                                _delay_ms(10);
                                break;
                        case STATENOSIGNAL:
                                MRF_transmit_packet(&tyhja);
                                _delay_ms(10);
                                break;
                        default:
                                MRF_transmit_packet(&tyhja);
                                _delay_ms(10);
                                break;
                }
}*/
uint16_t ovcounter = 0, tauko = 100;
uint8_t vtila = VON;
spi_init();
_delay_ms(50);
MRF_init();
DDRD |= (1<<PORTD3) | (1<<PORTD1) | (1<<PORTD0);
DDRC |= (1<<PORTC3);
DDRC &= ~(1<<PORTC2);
PORTC &= ~(1<<PORTC2);

PCICR = (1 << PCIE1);
PCMSK1 = (1 << PCINT10);
sei();

// D0 = Jarru
// D1 = vasen vilkku
// D3 = oikea vilkku
PORTC &= ~(1<<PORTC3);
while (1)
{
        switch (state) {
                        case STATEJ:
                                PORTD |= (1<<PORTD0);
                                PORTD &= ~(1<<PORTD1);
                                PORTD &= ~(1<<PORTD3);
                                icounter = 0;
                                sei();
                                break;
                        case STATEOV:
                                icounter = 0;
                                switch (vtila){
                                        case VON:
                                        PORTD &= ~(1<<PORTD0);
                                        PORTD &= ~(1<<PORTD1);
                                        PORTD |= (1<<PORTD3);
                                        _delay_ms(tauko);
                                        vtila = VOFF;
                                        break;
                                        case VOFF:
                                        PORTD &= ~(1<<PORTD0);
                                        PORTD &= ~(1<<PORTD1);
                                        PORTD &= ~(1<<PORTD3);
                                        _delay_ms(tauko);
                                        vtila = VON;
                                        break;
                                }
                                sei();
                                break;
                        case STATEVV:
                                icounter = 0;
```

```
                switch (vtila){
                        case VON:
                        PORTD &= ~(1<<PORTD0);
                        PORTD |= (1<<PORTD1);
                        PORTD &= ~(1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VOFF;
                        break;
                        case VOFF:
                        PORTD &= ~(1<<PORTD0);
                        PORTD &= ~(1<<PORTD1);
                        PORTD &= ~(1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VON;
                        break;
                }
                sei();
                break;
        case STATEJOV:
                icounter = 0;
                switch (vtila){
                        case VON:
                        PORTD |= (1<<PORTD0);
                        PORTD &= ~(1<<PORTD1);
                        PORTD |= (1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VOFF;
                        break;
                        case VOFF:
                        PORTD |= (1<<PORTD0);
                        PORTD &= ~(1<<PORTD1);
                        PORTD &= ~(1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VON;
                        break;
                }
                sei();
                break;
        case STATEJVV:
                icounter = 0;
                switch (vtila){
                        case VON:
                        PORTD |= (1<<PORTD0);
                        PORTD |= (1<<PORTD1);
                        PORTD &= ~(1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VOFF;
                        break;
                        case VOFF:
                        PORTD |= (1<<PORTD0);
                        PORTD &= ~(1<<PORTD1);
                        PORTD &= ~(1<<PORTD3);
                        _delay_ms(tauko);
                        vtila = VON;
                        break;
                }
                sei();
                break;
        case STATEJSOS:
                icounter = 0;
                switch (vtila){
                        case VON:
                        PORTD |= (1<<PORTD0);
                        PORTD |= (1<<PORTD1);
```

```
                                                PORTD |= (1<<PORTD3);
                                                _delay_ms(tauko);
                                                vtila = VOFF;
                                                break;
                                                case VOFF:
                                                PORTD |= (1<<PORTD0);
                                                PORTD &= ~(1<<PORTD1);
                                                PORTD &= ~(1<<PORTD3);
                                                _delay_ms(tauko);
                                                vtila = VON;
                                                break;
                                        }
                                        sei();
                                        break;
                                case STATESOS:
                                        icounter = 0;
                                        switch (vtila){
                                                case VON:
                                                PORTD &= ~(1<<PORTD0);
                                                PORTD |= (1<<PORTD1);
                                                PORTD |= (1<<PORTD3);
                                                _delay_ms(tauko);
                                                vtila = VOFF;
                                                break;
                                                case VOFF:
                                                PORTD &= ~(1<<PORTD0);
                                                PORTD &= ~(1<<PORTD1);
                                                PORTD &= ~(1<<PORTD3);
                                                _delay_ms(tauko);
                                                vtila = VON;
                                                break;
                                        }
                                        sei();
                                        break;
                                case STATENOSIGNAL:
                                        PORTD &= ~(1<<PORTD0);
                                        PORTD &= ~(1<<PORTD1);
                                        PORTD &= ~(1<<PORTD3);
                                /*      icounter = 0;
                                        switch (vtila){
                                                case VON:
                                                PORTD |= (1<<PORTD0);
                                                PORTD &= ~(1<<PORTD1);
                                                PORTD &= ~(1<<PORTD3);
                                                _delay_us(500);
                                                vtila = VOFF;
                                                break;
                                                case VOFF:
                                                PORTD &= ~(1<<PORTD0);
                                                PORTD &= ~(1<<PORTD1);
                                                PORTD &= ~(1<<PORTD3);
                                                _delay_ms(5);
                                                vtila = VON;
                                                break;
                                        }*/
                                        sei();
                                        break;
                        }
                }
        }
```