Matti Tassi

Langaton ohjelmointi

Opinnäytetyö

Syksy 2010

Tekniikan yksikkö

Tietotekniikan Koulutusohjelma

Ohjelmistotekniikka

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Matti Tassi

Työn nimi: Langaton ohjelmointi

Ohjaaja: Heikki Palomäki

Vuosi: 2010 Sivumäärä: 32 Liitteiden lukumäärä: 2

Tämän työn tavoitteena oli toteuttaa langattoman ohjelmoinnin protokolla sekä siihen liittyvä laite- ja ohjelmistokokonaisuus. Tällä oli tarkoitus tukea uutta kehitteillä olevaa piirilevyä, joka pohjautui NRF24LE1-radioprosessoriin. Tarkoitus oli myös, että lähes kuka tahansa teknisesti orientoitunut henkilö pystyisi ohjelmoimaan toisen samanlaisen laitteen langattomasti, ilman suurta vaivannäköä. Langaton ohjelmointi mahdollistaisi myös laitteen jatkokehityksen ja se antaisi sille täysin uusia käyttömahdollisuuksia, esim. tuulivoimalassa olevan laitteen koodi voitaisiin päivittää langattomasti, joten virheiden korjaaminen ja jopa erilaisten ohjelmakoodien kokeileminen olisi helppo tehtävä.

Työssä käytettiin useita laitteita. NRF24LE1-radioprosessoria käyttävät laitteet olivat työssä pääosassa. Niitä käytettiin sekä langattoman ohjelmoinnin käskyjen antajina, sekä kohdelaitteina. AT90USB162-mikro-ohjainta käyttävä USB-tikku toimi projektissa USB-SPI-siltana PC:n ja NRF24LE1-radioprosessorin välillä.

Työssä NRF24LE1-radioprosessoria ohjelmoitiin USB-tikun kautta PC:llä. PC:llä oli mikroprosessorin ohjelmoinnin helpottamiseksi toteutettu SURFprogrammer-ohjelma, jolla ohjelmoinnin pystyi tekemään suhteellisen helposti. Varsinainen koodin kääntäminen tehtiin myös SURFprogrammer-ohjelmalla, mutta sisäisesti se käytti SDCC-kääntäjää. Langatonta ohjelmointia varten kehitettiin kaksi uutta ohjelmaa. Yhtä käytettiin ohjelmoijana toimivassa NRF24LE1-laitteessa ja toista käytettiin Bootloader-ohjelmana kaikissa kohdelaitteissa. Kohdelaitteeseen langattomasti laitettavan ohjelmakoodin varsinainen syöttö tehtiin ilmaiseksi saatavilla olevalla sarjaporttikommunikointiohjelmalla.

Avainsanat: AT90USB162, NRF24LE1, Atmel, langaton

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: The School of Technology
Degree programme: Information Technology
Specialisation: Software Engineering

Author: Matti Tassi

Title of thesis: Wireless programming

Supervisor: Heikki Palomäki

Year: 2010 Number of pages: 32 Number of appendices: 2

The purpose of this work was to implement a protocol for wireless programming and develop a hardware and software entity for it. All of that was meant to support the new microchip, which was built around NRF24LE1 radio processor. Another purpose was that any technically oriented person could easily program another similar device wirelessly. Wireless programming could also enable further development of the device and give it new usages. For example, you could program a device in a wind farm wirelessly, which would allow you to fix software bugs or even try different applications on the fly.

Several devices were used in this work. Main devices were those that used NRF24LE1 radio processor. They were used as devices that gave orders wirelessly and also as target devices. A USB-stick with AT90USB162 microcontroller was used as a USB to an SPI Bridge between a PC and NRF24LE1 device.

Devices using NRF24LE1 radio processor were programmed via a USB-stick with a PC. SURFprogrammer was a program that was developed in order to make it easier to program devices using NRF24LE1 radio processor. Code compiling was also done with SURFprogrammer, but internally it used SDCC compiler. Two new programs were created for wireless programming. One was for programming NRF24LE1 device and the other one was used as a Bootloader program for all target devices. The actual input of the program code, which was wanted to transfer wirelessly into the target device, was done with a freely available serial port communication program.

Keywords: AT90USB162, NRF24LE1, Atmel, wireless

SISÄLTÖ

Käytetyt termit ja lyhenteet

SPI	Serial Peripheral Interface. Kahden tai useamman laitteen välinen kommunikointiväylä, missä on 4 signaalia.
8051	Voidaan viitata sekä arkkitehtuurin tyyppiin, sekä ohjelmoinnin käyttämään käskykantaan.
NRF24LE1	Nordic Semiconductor yhtiön kehittämä radioprosessori.
AT90USB162	Atmelin kehittämä mikro-ohjain.
GENSEN	Geneeriset sensoriverkot. Projektin nimi.
SurfNet	Seinäjoki UAS RF. Laitteisto- ja ohjelmistokokonaisuuden nimi.
IHX	Intel Hex Format. Tiedostomuoto

(NRF24LE1 Product Specification 2010, 103–104, Virrankoski 2012, 1.)

KUVIO- JA TAULUKKOLUETTELO

1 JOHDANTO

1.1 Työn tausta

SeAMK:n langattomien laitteiden kehitys oli yksi avaintekijöistä tämän työn syntyyn. Tähän kehitykseen johti etenkin SeAMK:n osallistuminen GENSEN-hankkeeseen. Sen puitteissa kehitettiin muun muassa toimiva ohjelmointiympäristö NRF24LE1-radiopiirille (Huhta 2009, 50).

GENSEN oli TEKESin rahoittama projekti, jossa kehitettiin yleiskäyttöinen laite- ja ohjelmistoympäristö langattoman automaation tarpeisiin. SeAMK kehitti tästä kokonaisuudesta SurfNet-ympäristön, johon sisältyi kaksi eri laitteistoalustaa: yhteysprotokolla, sekä ohjelmistoympäristö. Tässä työssä käytettiin etenkin SurfNet alustan v 1.0 laitteistoa. Tämän pohjalta kehitettiin lisäksi viisi erilaista pilottisovellusta eri käyttötarkoituksiin. (Virrankoski 2012, 1, 45.)

SurfNet-ympäristön vaihtoehtona olisi ollut NRF24LE1-radioprosessoria varten tehty kaupallinen kehitysympäristö. Korkean hintansa takia tätä vaihtoehtoa ei kuitenkaan otettu, vaan oma avoin ja yksinkertainen kehitysympäristö tehtiin opiskelijavoimin. (Palomäki 2013, 290.)

Erilaisten projektien yhteydessä tuli tarve saada laite vielä pienemmäksi. Tämän esteeksi muodostui laitteen ohjelmointiväylä. Ohjelmointiväylän pois jättämiseksi haluttiin tuki langattomalle ohjelmoinnille.

1.2 Työn tavoite

Yksi työn tavoitteista oli uuden NRF24LE1-radioprosessoria käyttävän piirilevyn tukeminen kehittämällä sitä varten langattoman ohjelmoinnin protokolla ja siihen

yhteensopiva ohjelmistokokonaisuus. Kokonaisuuden tarkoitus oli, että lähes kuka tahansa teknisesti orientoitunut henkilö pystyisi ohjelmoimaan uuden laitteen langattomasti, ilman suurta vaivannäköä.

Tavoitteena oli myös pienentää laitteen vaatimuksia poistamalla tarve erillisestä ohjelmointiväylästä. Langaton ohjelmointi mahdollistaisi myös useita erilaisia ratkaisuja laitteiden ohjelmoimiseen ja helpottaisi sitä. Esimerkiksi tuulivoimalassa oleva laitteen koodi voitaisiin päivittää langattomasti, joten virheiden korjaaminen tai jopa täysin erilaisten ohjelmakoodien kokeilu olisi helppo tehtävä.

1.3 Työn rakenne

Työ aloitettiin kuvailemalla työssä käytettävää laitteistoa. Näitä olivat Atmelin AT90USB162-mikro-ohjainta käyttävä USB-tikku, sekä NRF24LE1-radiopiiriä käyttävä piirilevy. Tämän jälkeen kuvailtiin langattoman ohjelmoinnin toiminnan protokolla.

Suunnittelun jälkeen kuvailtiin varsinaisen työn toteutusta. Eri ohjelmien ja laitteiden käyttö kerrottiin loogisessa järjestyksessä. SURFprogrammer oli olennainen osa työtä, joten siitä kerrottiin sen kehityshistoria, sekä sen käyttötarkoitus projektissa. USB-SPI-tikusta kerrottiin sen käyttötarkoitus projektissa, sekä sen sisältämän ohjelman toiminta yleisellä tasolla. Lopuksi kuvailtiin kaksi opinnäytetyötä varten tehtyä ohjelmaa, joista toinen oli tarkoitus asentaa langattomasti ohjelmoivaan laitteeseen ja toinen kohdelaitteeseen. Viimeisenä työn toteutuksesta kerrottiin sen testaus alusta loppuun.

Työn toteutuksen jälkeen käytiin läpi työn tuloksia ja sen arviointia. Tuloksista kerrottiin laite- ja ohjelmistokokonaisuus, sekä langattoman ohjelmoinnin osuus. Arvioinnissa arvioitiin työn kokonaisuutta ja sen jälkeen käytiin läpi jatkokehitysideoita.

2 SUUNNITTELU

2.1 Laitteisto

Työssä käytettiin kahta erilaista laitetta. Toinen oli AT90USB162-USB-tikku ja toinen oli NRF24LE1-radiopiiri.

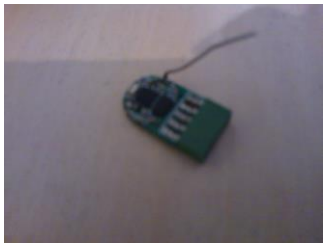
2.1.1 AT90USB162-USB-tikku

Kuva 1. AT90USB162-mikro-ohjainta käyttävä USB-SPI-tikku.

USB-tikkua (kuva 1) käytettiin työssä USB-SPI-siltana tietokoneen ja NRF24LE1-radiopiirin välillä. Laitetta käytettiin sekä NRF24LE1-radioprosessorien ohjelmoimiseen, että kommunikointiväylänä ohjelmointia suorittavan radiopiirin ja PC:n välillä.

AT90USB162 mikro-ohjain tuki sisäisesti sekä USB 2.0:aa että SPI:tä. Laitteessa oli myös erikseen varattu muistia 176 tavua USB:tä varten. Sekä USB:tä että SPI:tä varten oli useita rekistereitä, joilla näiden toimintaa pystyi ohjailemaan. (AT90USB82/162 Datasheet 2008.)

2.1.2 NRF24LE1-radiopiiri



Kuva 2. Työssä käytetty NRF24LE1-radiopiiri.

Työssä käytettiin Seinäjoen Ammattikorkeakoulussa kehitettyä piirilevyä (kuva 2), joka oli rakennettu NRF24LE1-radioprosessoria varten. NRF24LE1-radio-ohjainta käytettiin varsinaiseen langattomaan ohjelmointiin, sekä itse ohjelmointisignaaleja lähettävänä laitteena että kohdelaitteena.

NRF24LE1-radioprosessorin etuina olivat etenkin sisäänrakennettu langattoman tiedonsiirron käsittely, sekä se, että sen asetuksilla pystyi säätämään melkein mitä tahansa sisäistä ominaisuutta. Myös virrankulutus oli erittäin vähäinen. (NRF24LE1 Product Specification 2010, 10.)

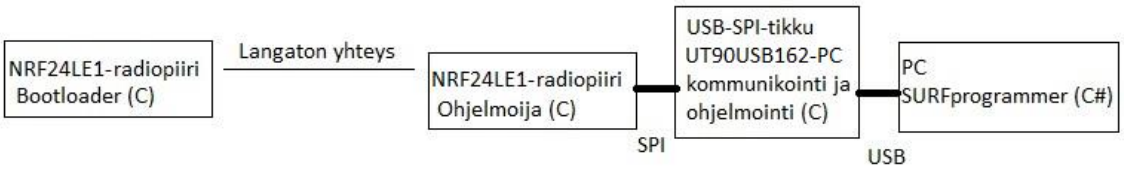
2.2 Rakenne

Suunnitteluvaiheessa ensiarvoisen tärkeää oli kokonaisuuden hahmottaminen. Alusta alkaen oli selvää, ettei yksi ohjelma riittäisi työn toteuttamiseen. Työ oli pakko hajauttaa mahdollisesti useille eri laitteille. Myös useiden eri ohjelmointikielien mahdollisuus kävi selväksi.

Välttämättömiksi ohjelmakokonaisuuksiksi muodostuivat kohdelaitteen ohjelmakoodi, ja sitä ohjelmoiva ohjelmointilaite. Näistä molemmat käyttäisivät NRF24LE1-radiopiiriä. Myös ohjelmointikielen valinta näihin tapauksiin oli selvä: C-kieli soveltui parhaiten laiteläheiseen ohjelmointiin.

Nämä kaksi NRF24LE1-laitetta olisivat jo riittäneet toistensa ohjelmointiin, mutta niihin ei varsinaisesti saanut liitettyä minkäännäköistä käyttöliittymää, joten se olisi rajoittunut vain tietynlaisen valmiiksi tehdyn koodin syöttöön. Lisäksi kohteen olisi pitänyt olla tiedossa etukäteen. Tarvittiin jotain, minkä välityksellä käyttäjä voisi syöttää haluamansa ohjelmakoodin ja valita ohjelmoitavan laitteen. Tähän tarkoitukseen käytettiin USB-SPI-tikkua. Tämän hyödyntäminen tapahtui USB-SPI-sillan kautta ja siitä edelleen tietokoneeseen sarjaportin välityksellä.

USB-tikun ohjelmakoodia ei tarvinnut tehdä aivan alusta, vaan käytössä oli Nordic Semiconductorin ohjelmointialusta. Kyseinen ohjelmakoodi huolehti etenkin USB-tikun ja tietokoneen välisestä liikenteestä. Sen avulla tietokone tunnisti USB-tikun sarjaporttina, jolloin sitä oli helppo käsitellä.



Kuva 3. Laite- ja ohjelmistokokonaisuus.

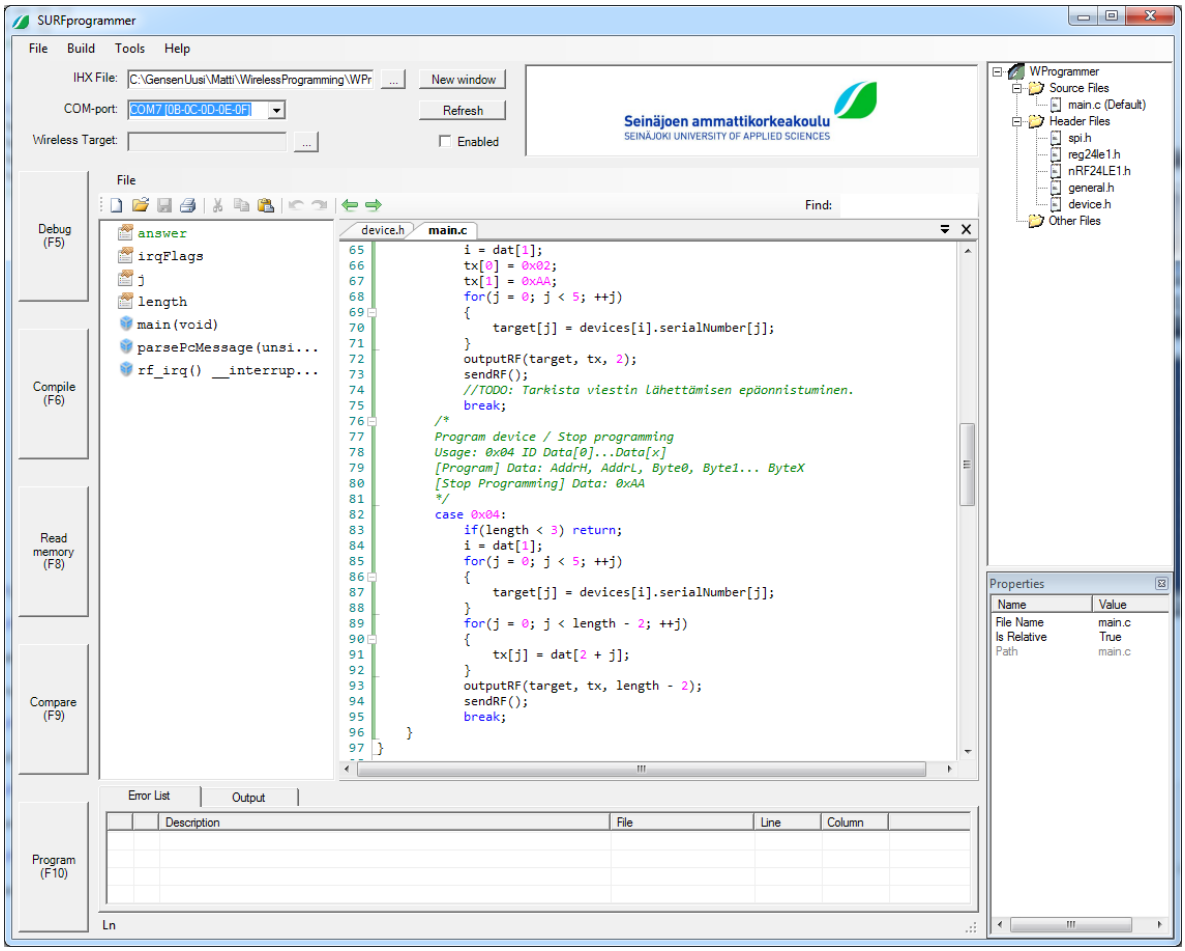
Toteutukseen tarvittava laite- ja ohjelmistokokonaisuus on esitelty kuvassa 3. Jokaiseen laitteeseen tarvittiin oma ohjelmansa, jonka ohjelmointikieli on kerrottu sulkujen sisällä.

3 TYÖN TOTEUTUS

3.1 Toteutus

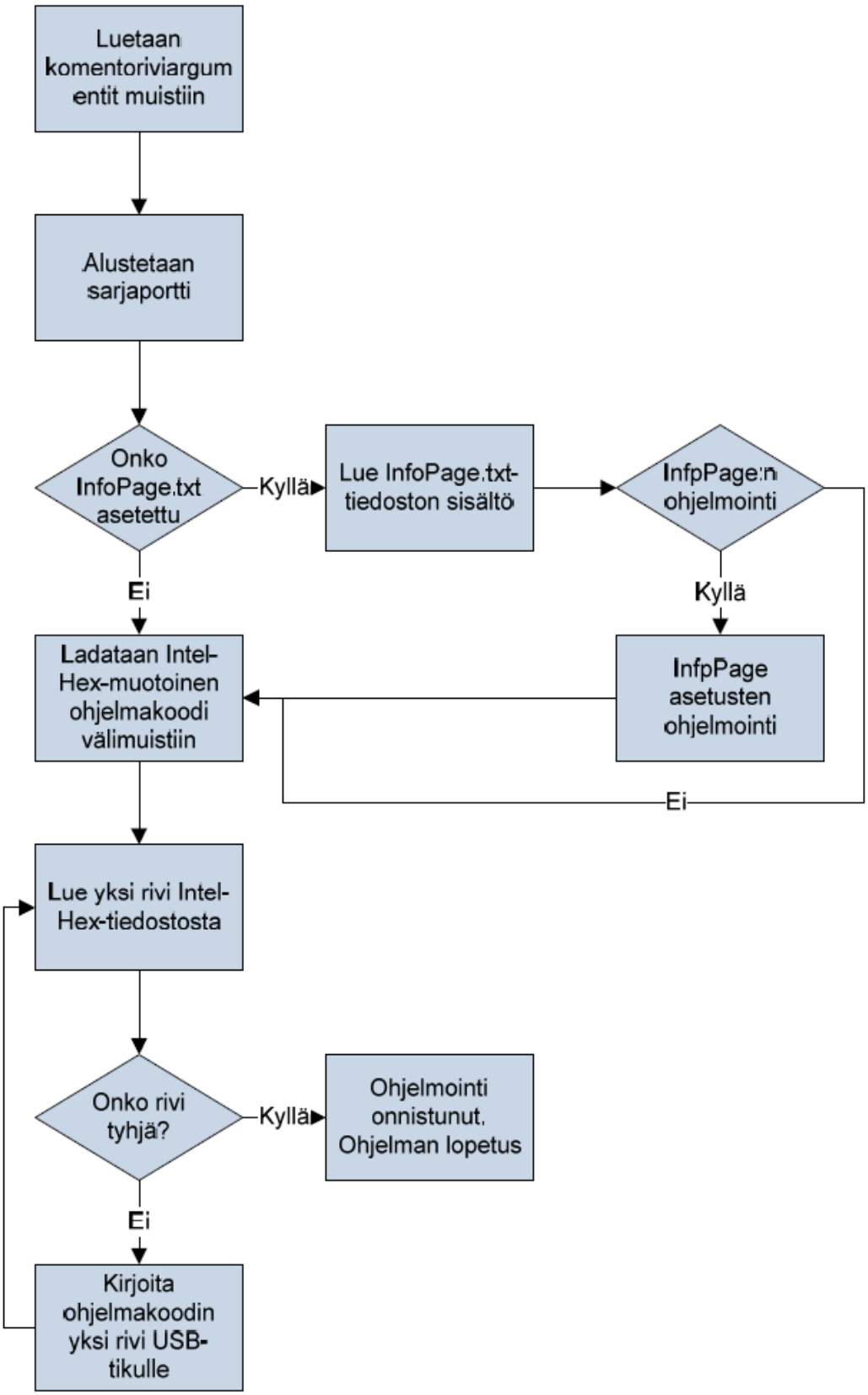
Työ toteutettiin ohjelma kerrallaan. Työssä toteutettavia ohjelmia olivat WProgrammer ja Bootloader. Työssä käytettiin myös muita jo ennen työtä valmistuneita ohjelmia, kuten SURFprogrammeria ja USB-tikun ohjelmaa.

3.1.1 SURFprogrammer



Kuva 4. Esimerkkikuva SURFprogrammer-ohjelmasta.

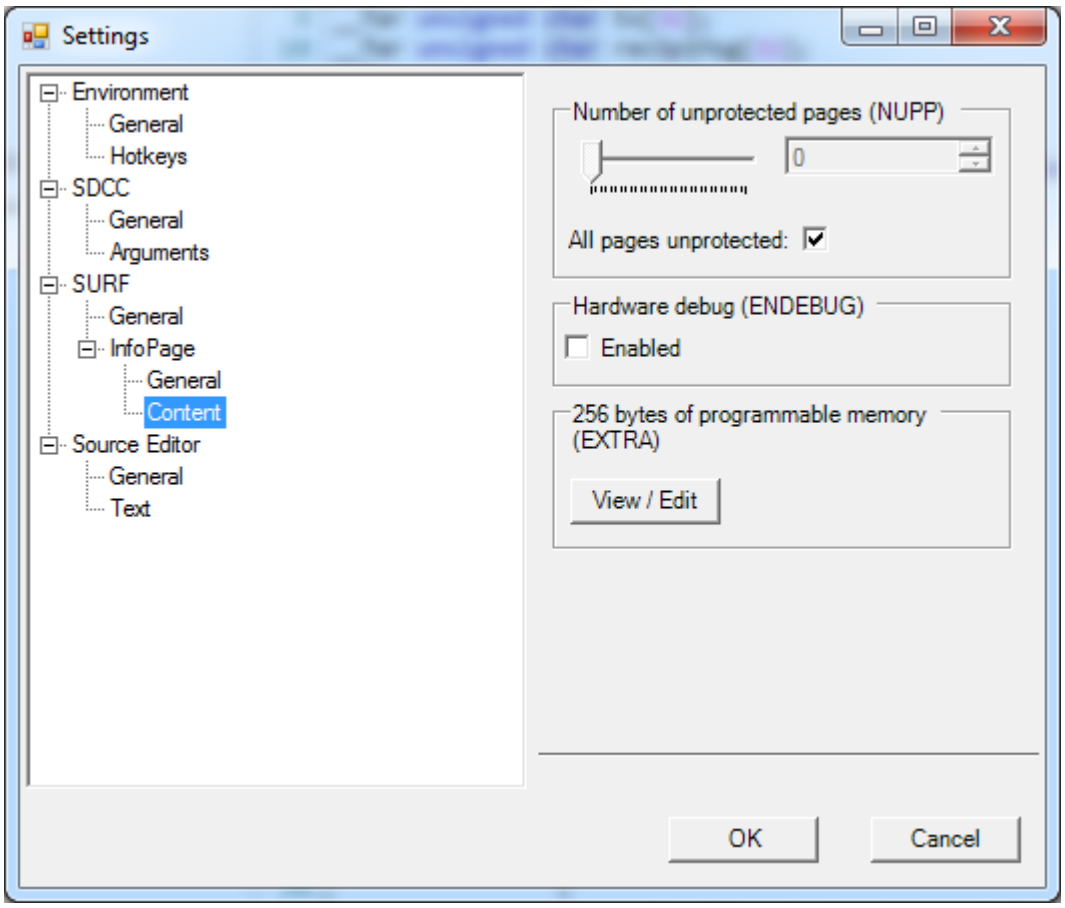
PC-puolelle oli toteutettu SURFprogrammer ennen opinnäytetyön aloittamista. Tämä ohjelma oli tarkoitettu NRF24LE1-laitteiden ohjelmoimiseen USB-SPI-tikun välityksellä. Ohjelman käyttöliittymä on esitelty kuvassa 4.



Kuva 5. LE1-programmer-ohjelman vuokaavio. (Huhta 2009, 27)

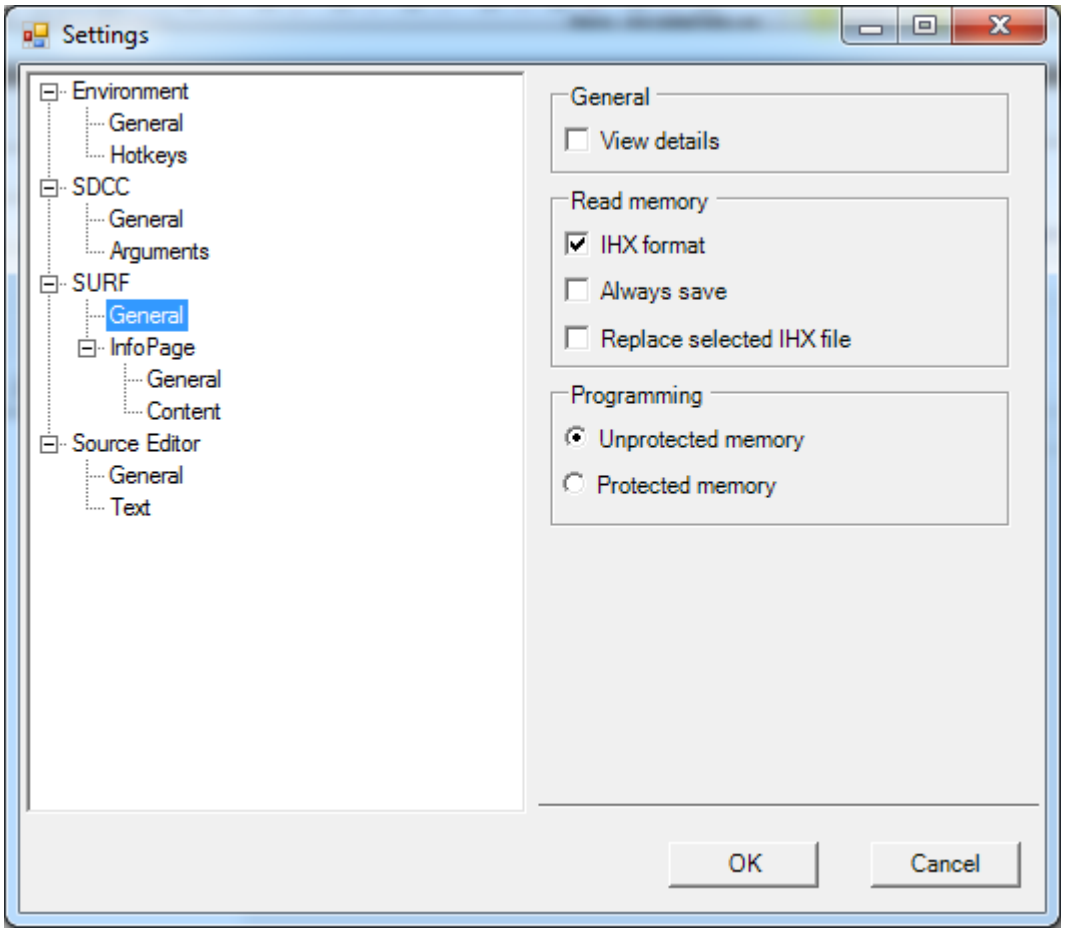
SURFprogrammer oli jatkokehitystä aiemmin toteutetulle Marko Huhdan tekemälle LE1-programmer-ohjelmalle. LE1-programmer-ohjelma oli alun perin pelkkä konsolipohjainen sovellus, mutta jatkokehitysideana oli tehdä siihen graafinen käyttöliittymä ja lisätä siihen joitain uusia ominaisuuksia. SURFprogrammer toteutettiin siten, että se noudattaisi edelleen suurimmaksi osaksi vanhaa LE1-programmer-ohjelman vuokaaviota (5), mutta ensimmäisenä vaiheena oli käyttöliittymän Program-napin painaminen ja InfoPage.txt-tiedoston sijaan asetukset tulivat suoraan graafisesta käyttöliittymästä. (Huhta 2009, 55–56.)

Ohjelmalla pystyi ylläpitämään ja kääntämään koodia laitteen vaatimaan muotoon. Lisäksi sillä pystyi ohjelmoimaan laitteet suoraan, mikä helpotti etenkin testaamista.



Kuva 6. SURFprogrammerin InfoPage-asetukset.

Ohjelmaan oli myös toteutettu suoraan joitain NRF24LE1:n tukemia toimintoja, esimerkiksi InfoPagen syöttö laitteeseen kuvan 6 mukaisesti.



Kuva 7. Asetuksia, joista voidaan muun muassa ohjelmoida suojattu muistialue.

Työn kannalta tärkeät ominaisuudet olivat InfoPagen suojattujen sivujen lukumäärän asetus, sekä suojatun muistialueen ohjelmointi. Suojatun muistin ohjelmoimisen voi kytkeä päälle kuvan 7 mukaisessa asetusikkunassa.

3.1.2 USB-SPI

Kaikkein tärkein osa kokonaisuutta oli USB-SPI-tikku, jonka avulla oli tarkoitus ohjelmoida molemmat NRF24LE1-laitteet. Toiseen laitteeseen tarvittiin Bootloader-ohjelmakoodi ja toiseen tarvittiin WProgrammer, jonka tarkoitus oli välittää USB-SPI-tikusta tullut ohjelmakoodi Bootloader-ohjelman päällä pyörivään NRF24LE1-

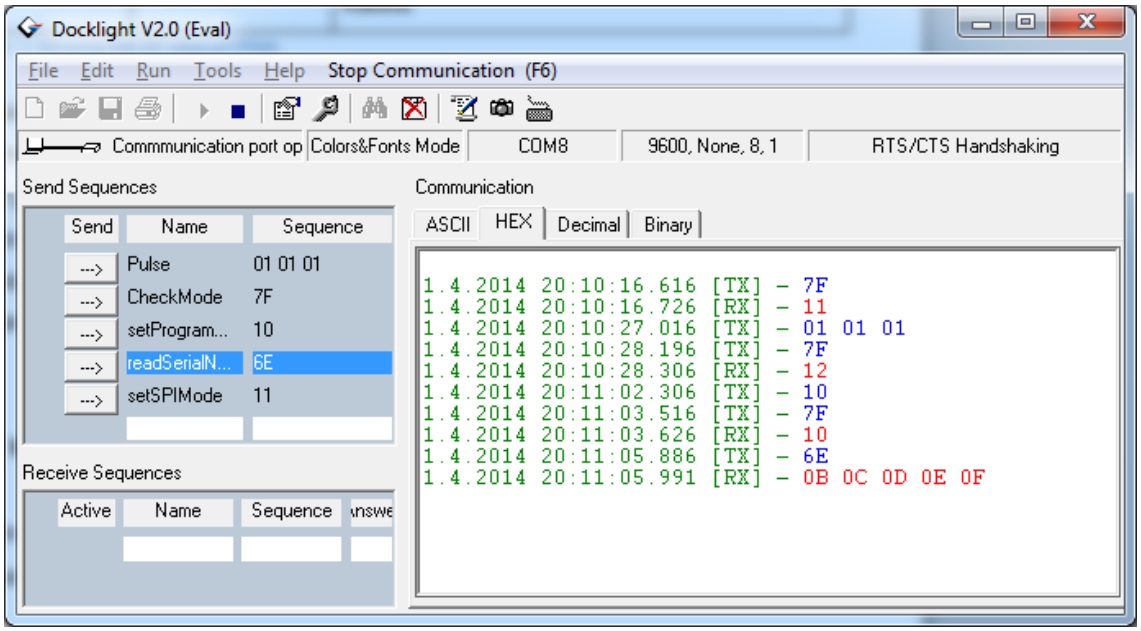
laitteeseen. Bootloader-laite pystyi ohjelmoimaan itse itsensä vastaanotettuaan ohjelmakoodin.

USB-SPI-tikkuun oli onneksi toteutettu oma ohjelma ennen työn aloitusta. Ohjelman toteutus oli myös työn tarkoituksiin nähden riittävä, joten muutoksia ei tarvinnut tehdä.

USB-SPI-tikun ohjelmassa oli neljä eri tilaa. Yksi näistä tiloista on pelkästään tilan vaihtoa varten.

Taulukko 1. USB-SPI-tikun eri tiloja.

Tilan nimi	**Valitaan ID:llä**	**Tarkoitus**
Ohjelmointitila	0x10	Tilan aikana voidaan ohjelmoida SPI-väylässä kiinni oleva laite.
SPI-tila	0x11	Tilaa käytetään viestien välitykseen ja vastaanottamiseen SPI-väylän kautta.
Tilanvaihto	0x12	Vaihdetaan toiseen tilaan.
Vastaanottotila	0x13	Luetaan tietoa nopeassa tahdissa SPI-väylän kautta.



Kuva 8. Esimerkissä vaihdeltu eri tilojen välillä ja lopuksi luettu laitteen sarjanumero.

Työssä käytettiin taulukossa 1 esitetyistä tiloista ohjelmointitilaa laitteiden ohjelmoimiseen ja SPI-tilaa varsinaisessa langattomassa ohjelmoinnissa. SURFprogrammer-ohjelma huolehti ohjelmointitilan käytöstä ja SPI-tilaa käytettiin sarjaporttikommunikointiin erikoistuneella Docklight-ohjelmalla. Docklight valittiin lähinnä sen tuttuuden ja ilmaisen näyteversion takia, mutta myös muita vastaavanlaisia ohjelmia olisi ollut tarjolla. Kuvan 8 esimerkki kuvastaa sekä Docklightin että myös USB-SPI:n yhteyskäyttöä. Alkutilana oli SPI-tila, mikä vaihdettiin lopulta ohjelmointitilaan. Viimeisenä kysyttiin USB-SPI-väylän kautta siihen kytketyn NRF24LE1-laitteen sarjanumero komennolla ”n”, joka on hex-muodossa 0x6E. Komennolla 0x7F kysyttiin nykyistä tilaa.

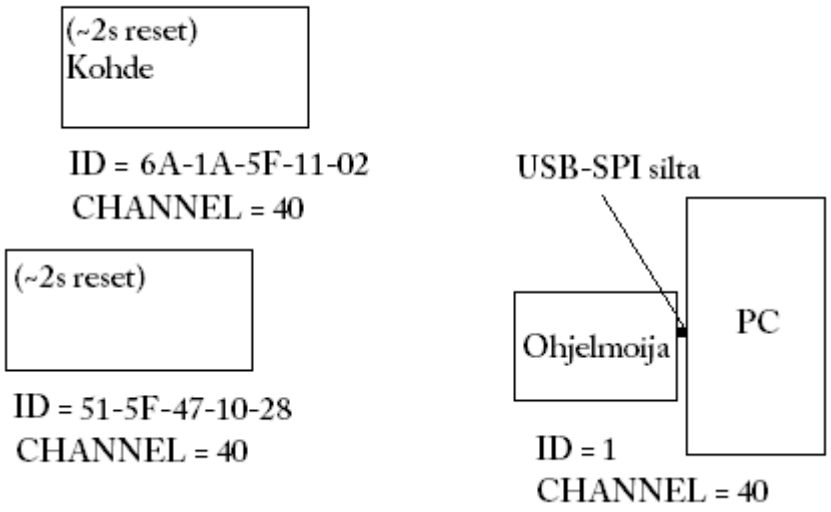
3.1.3 WProgrammer

Ohjelma oli tarkoitus asentaa NRF24LE1-laitteeseen, mikä toimisi langattomana ohjelmoijana. Laite pystyisi WProgrammer-ohjelman avulla ohjelmoimaan muita samanlaisia laitteita langattomasti.

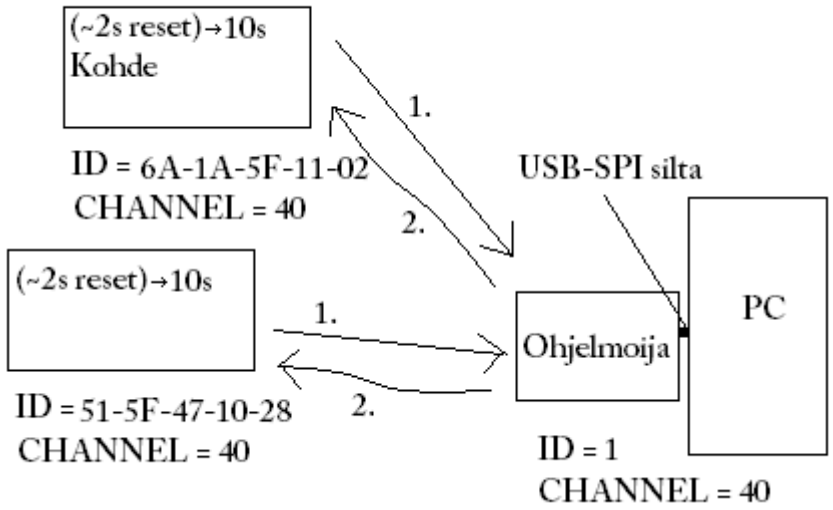
WProgrammer-ohjelmaa oli tarkoitus käyttää tietokoneen kautta käsittelemällä USB-SPI-tikkua sarjaporttina. USB-SPI-tikun oli tarkoitus toimia tässä ainoastaan välikätenä, mutta WProgrammer-ohjelman käyttämiseksi piti silti myös pitää mielessä USB-SPI-tikun ohjelman oma toimintalogiikka. WProgrammer ohjelman ohjelmakoodin päätiedosto on liitteenä (liite 1).

3.1.4 Bootloader

Bootloader-ohjelman oli tarkoitus pyöriä etäällä olevissa laitteissa, jotka oli tarkoitus ohjelmoida langattomasti. Bootloader-ohjelman ohjelmakoodin päätiedosto on liitteenä (liite 2).

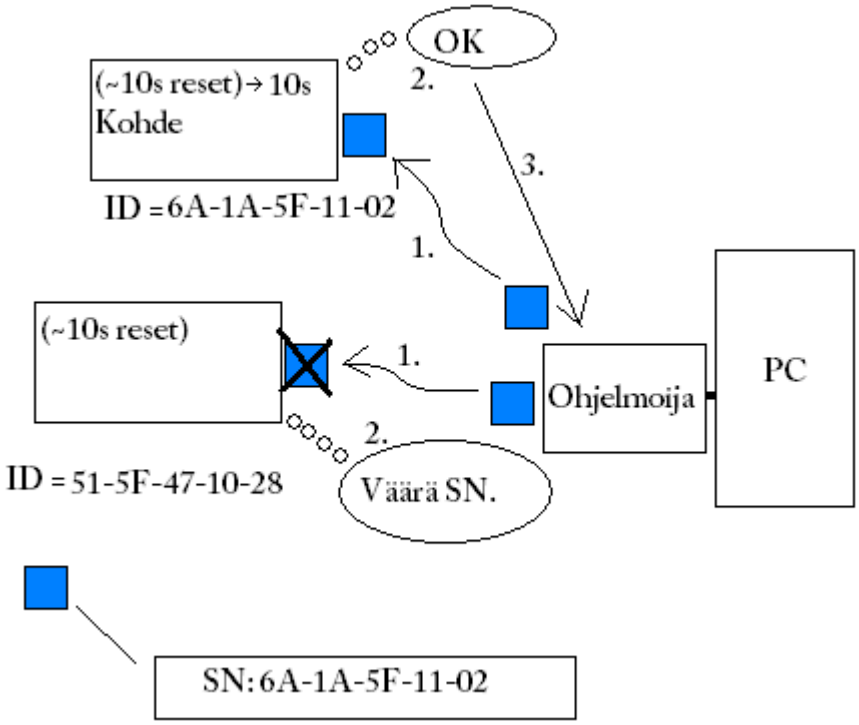


Kuva 9. Alkutilanteen esimerkkitapaus.



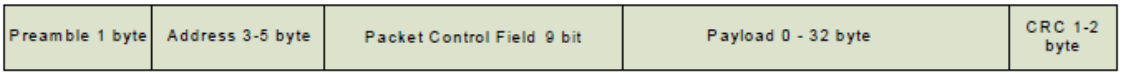
Kuva 10. Alkutilanteen aikana tapahtuva kommunikointi.

Bootloader-ohjelma oli suunniteltu kaikille langattomasti ohjelmoitaville kohdelaitteille. Normaalitilanteessa, kun ei ole tarkoitus ohjelmoida laitetta (9), laite käynnistyi ensin Bootloader-ohjelman, joka asetti laskurin 10 sekunnin päähän. Näiden 10 sekunnin ajan kuitenkin yritettiin myös ottaa yhteyttä ohjelmointilaitteeseen kuvan 10 mukaisesti. Mikäli yhteys saatiin ohjelmointilaitteeseen, niin ajastin asetettiin takaisin 10 sekuntiin. Uudelleenkäynnistyksen ajastus hoidettiin laitteen Watchdog-ominaisuudella (NRF24LE1 Product Specification 2010, 103–104).

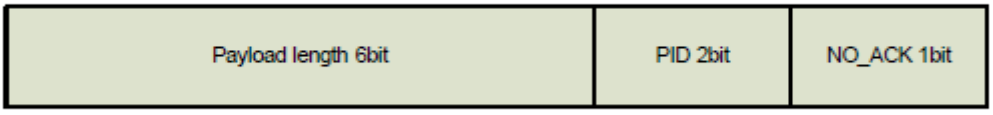


Kuva 11. Ohjelmoinnin aloitustilanne.

Ohjelmoinnin aloittamiseksi täytyi ensin valita laite sarjanumeron perusteella ja lähettää sille aloituspaketti.

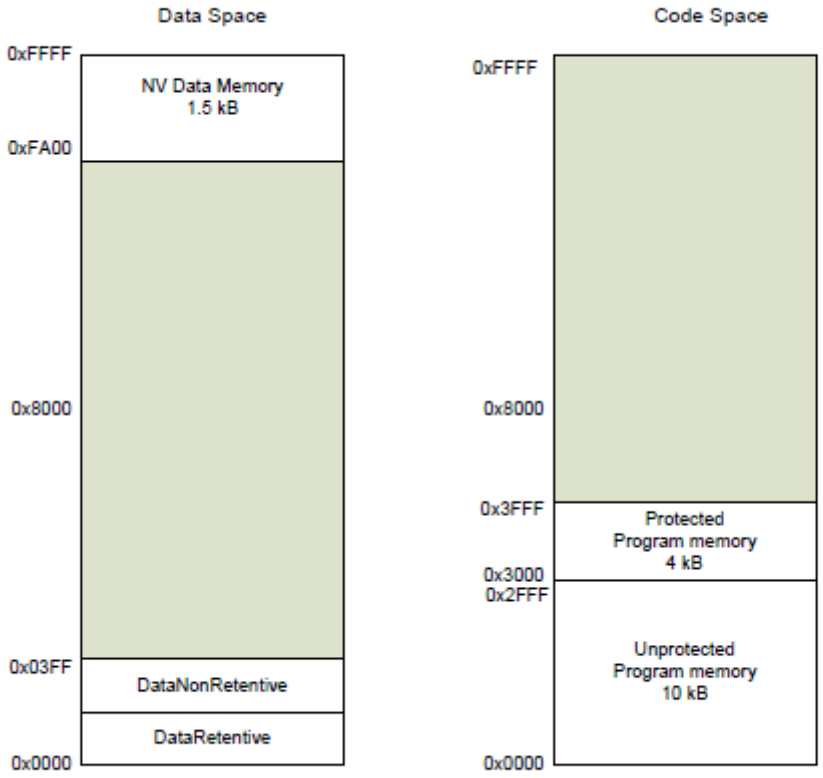


Kuva 12. Kuvaus laitteistopaketista. (NRF24LE1 Product Specification v1.6 2010, 23)



Kuva 13. Packet Control Field -kohdan tarkempi kuvaus. (NRF24LE1 Product Specification v1.6 2010, 24)

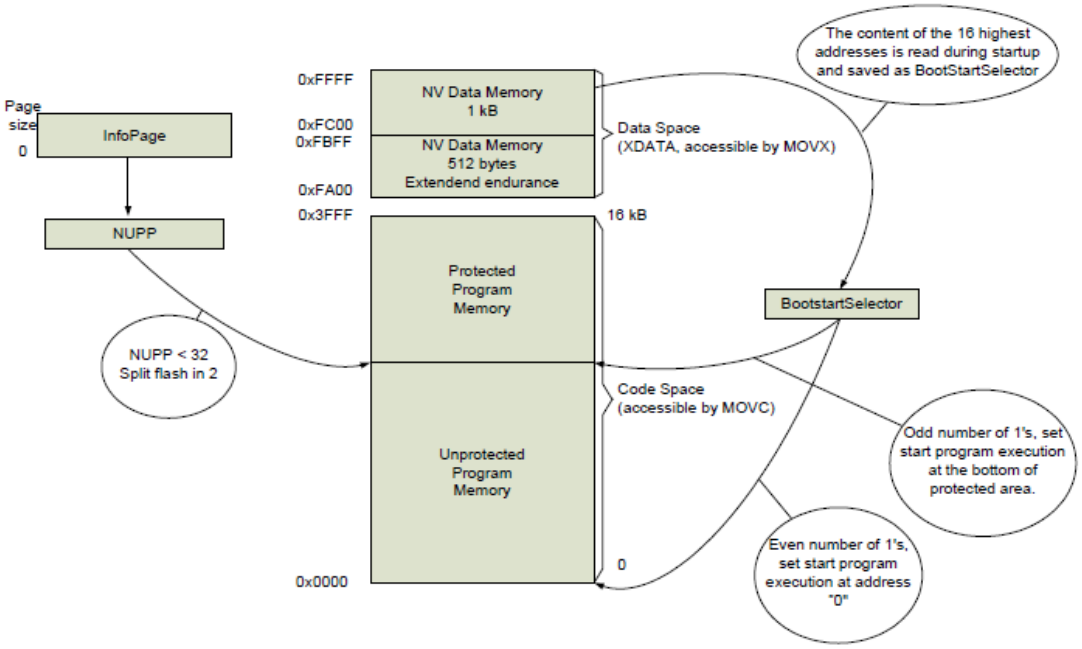
Kuva 12 kuvaa paketin eri osioita. Ensimmäisen tavun joka toinen bitti oli yksi ja joka toinen nolla. Aloittava bitti valittiin osoitteen ensimmäisen bitin perusteella. Osoite oli 5-tavuinen ja siihen asetettiin kohdelaitteen sarjanumero. Kuvan 11 mukainen paketin hylkääminen tai hyväksyminen tapahtui sarjanumeron perusteella. Payload-kohdassa oli varsinainen paketin välittämä tieto. Langattoman ohjelmoinnin eri vaiheissa Payload-osioon piti laittaa protokollan vaatimat tavut. (NRF24LE1 Product Specification 2010, 23–24.)



Kuva 14. Muistialueiden jako NRF24LE1-prosessorissa normaalitilanteessa. (NRF24LE1 Product Specification v1.6 2010, 82)

Bootloader-ohjelmaa ei voitu laittaa normaalilla tavalla laitteeseen, vaan se piti ohjelmoida laitteen suojatulle muistialueelle. Kuva 14 näyttää esimerkkinä mahdollisen muistijaon. InfoPage oli se, jolla pystyi kertomaan suojaamattoman ja suojatun muistin määrän. Sekä suojatulle muistialueelle ohjelmoimisen että Info Pagen

muuttamisen (6) pystyi tekemään helposti SURFprogrammer-ohjelmalla (kuva 4). (NRF24LE1 Product Specification 2010, 81–83.)

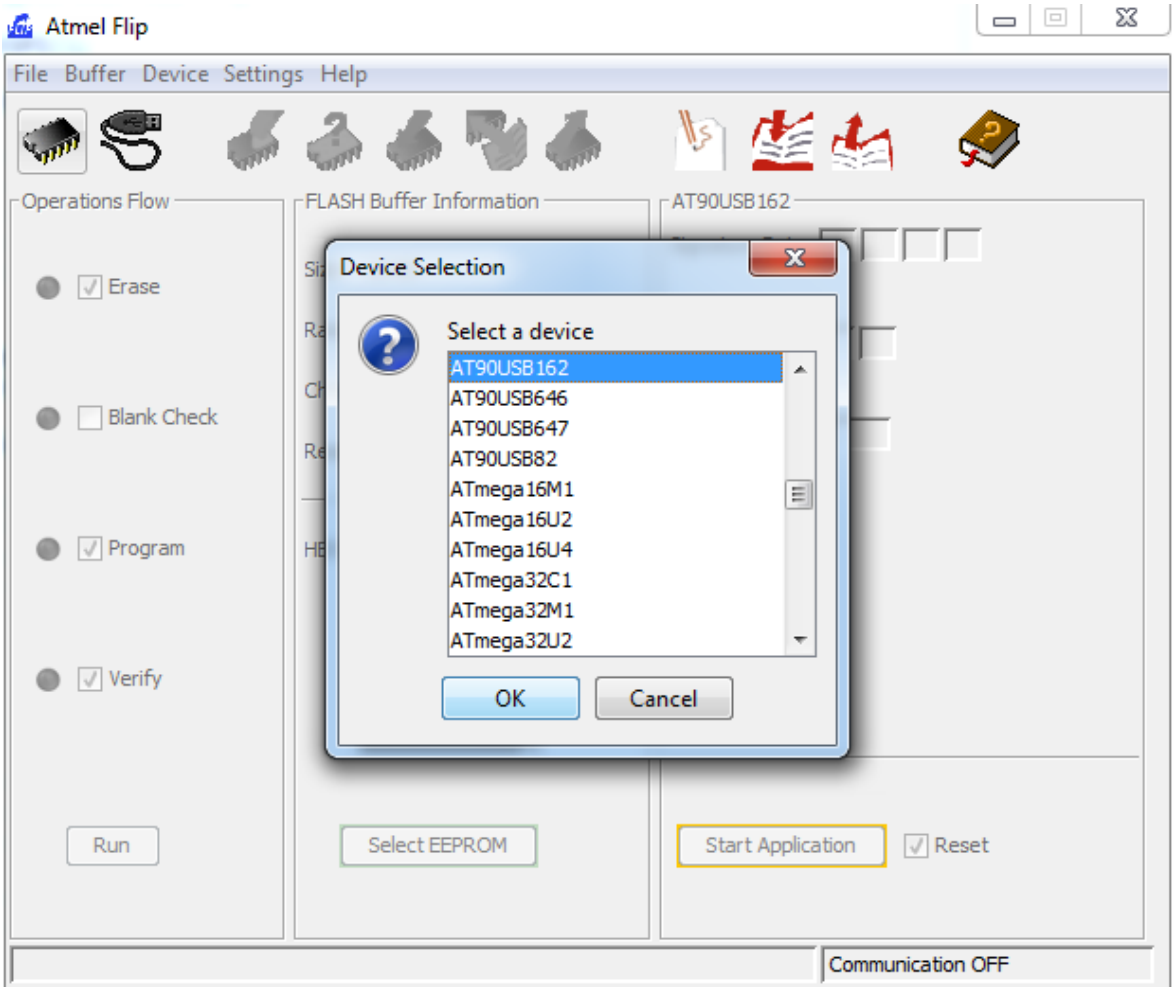


Kuva 15. Muistialueiden jakaminen ja käynnistyslohkon valinta. (NRF24LE1 Product Specification v1.6 2010, 73)

Pelkkä muistialueiden jakaminen ei kuitenkaan riittänyt, vaan laite piti saada käynnistymään suojatulta muistialueelta. Tätä tarkoitusta varten NV Data -muistin 16 ylintä osoitetta piti laittaa niin, että siellä olisi pariton määrä ”1” bittejä (kuva 15).

NV Data -muistin muokkaamisen jälkeen Bootloader-ohjelman asennukselle ei ollut enää mitään estettä.

3.2 Testaus

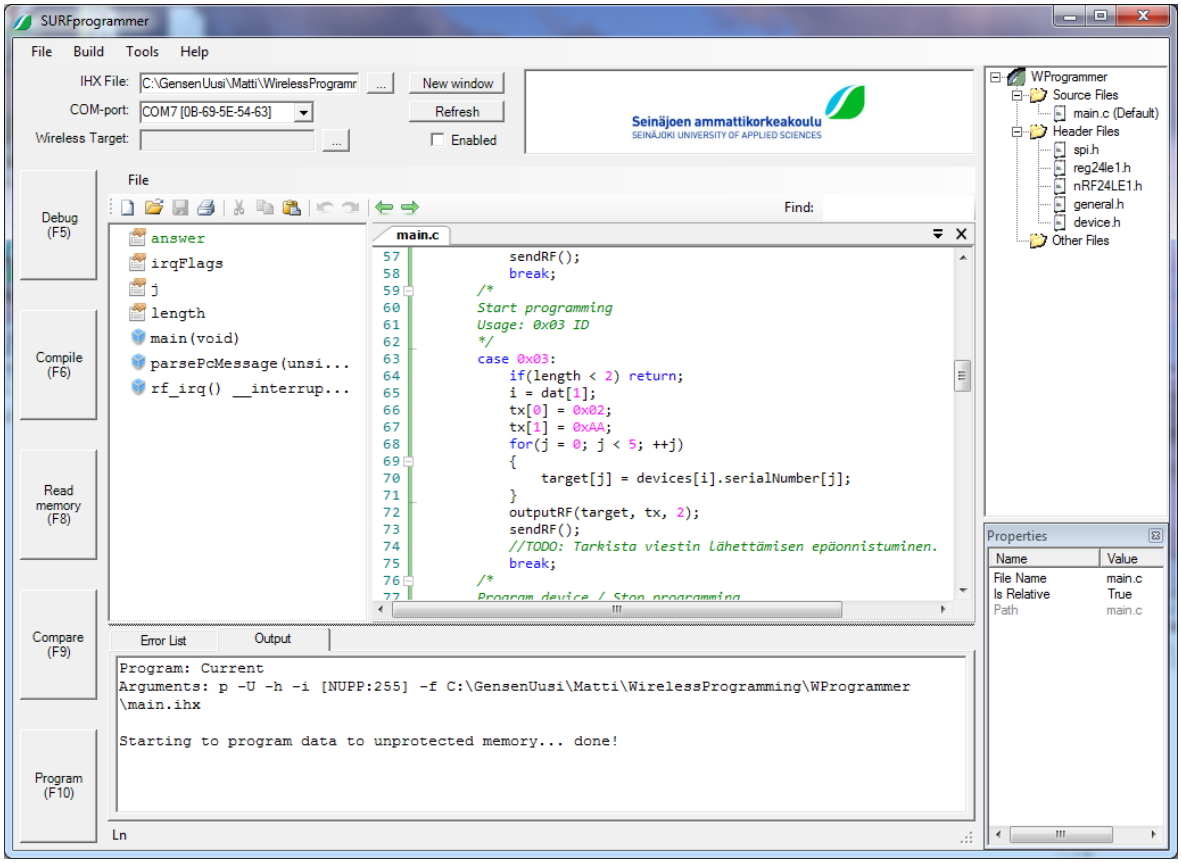


Kuva 16. Atmel Flip -ohjelmalla laitteen valinta.

Kokonaisuuden testaus päätettiin aloittaa lähes alusta asti eli USB-SPI-tikun ohjelmoinnista. Tähän oli valmis ohjelmapaketti, joka tarvitsi vain laittaa USB-tikkuun. Tätä varten käytettiin ohjelmaa Atmel Flip, joka oli tehty nimenomaan kyseistä tarkoitusta varten. Ensin valittiin laite kuvan (kuva 16) mukaisesti. Tämän jälkeen yhdistettiin GND- ja RST-nastat ja pidettiin yllä GND- ja HWB-nastoja koko ohjelmoinnin ajan. Koska testaus tehtiin Windows 7 -laitteella, niin ajurin asennuksen kanssa kului huomattavasti enemmän aikaa ja vaivaa, mutta lopulta USB-SPI-tikun tunnistaminen ja ohjelmointi onnistui. Tämän jälkeen laite piti saada näkymään sarjaporttina, joten PC:lle piti asentaa myös toinen ajuri. Myös tämän ajurin

asennuksen kanssa tuli joitain ongelmia käyttöjärjestelmän takia, mutta lopulta sekin saatiin asennettua.

Testikäytössä oli kaksi NRF24LE1-radiopiiriä, joten toiseen piti asentaa WProgrammer ja toiseen Bootloader.



Kuva 17. Ohjelmoitu suojaamaton muistialue SURFprogrammer-ohjelman avulla.

WProgrammer-ohjelman asennus oli suhteellisen helppo toimenpide, koska laitteen koko ohjelmamuistialue sai pysyä suojaamattomana, ja ohjelma voitiin asentaa suoraan siihen. Ohjelmointi tehtiin SURFprogrammer-ohjelmalla (kuva 17).

Ennen Bootloader-ohjelman asennusta laitteeseen piti ensin asettaa oikea määrä suojattuja ja suojaamattomia sivuja. Jokainen sivu vastasi 512 tavua. Suojaamatonta muistialuetta päätettiin varata 6 sivullista eli 3 kt. Muistialueen jakamisen lisäksi piti myös varmistua, että laite käynnistyisi ensisijaisesti suojatulta muistialueelta. Tämä tehtiin apuohjelmalla, joka muutti NV Data -muistin 16 ylintä tavua

sellaiseen muotoon, että siinä olisi pariton määrä ”1” bittejä (NRF24LE1 Product Specification v1.6 2010, 73). NV Data -muistin asetuksen jälkeen testattiin, että laite todellakin käynnistyi suojatusta muistista. Tämä testaus tehtiin yksinkertaisella SPI-viestiä lähettävällä ohjelmalla.

Bootloader asennettiin suojattuun muistiin. Tämän jälkeen voitiin aloittaa kokonaisuuden testaus. Tämä tehtiin Docklight-ohjelmalla. Asennettava ohjelma käännettiin ensimmäiseksi SURFprogrammer-ohjelmalla oikeaan IHX-muotoon. Tämä käännettiin tekstinkäsittelyohjelmalla USB-SPI-tikulle lähetettävään muotoon. Langattoman ohjelmoinnin vaatimat aloitustavut syötettiin manuaalisesti. Tämän jälkeen syötettiin itse ohjelmakoodi.

Tulos saatiin todettua tarkistamalla SPI-väylästä tuleva viesti. Langattomasti asennettava ohjelma oli yksinkertainen SPI-viestiä lähettävä ohjelma. Viestin tultua läpi voitiin olla varma, että langaton ohjelmointi onnistui.

4 TULOKSET JA ARVIOINTI

4.1 Tulokset

Työn tuloksia olivat sen laite- ja ohjelmistokokonaisuus, sekä itse langattoman ohjelmoinnin kehittäminen.

4.1.1 Laite- ja ohjelmistokokonaisuus

Tässä työssä saatiin laadittua ja toteutettua langattoman ohjelmoinnin vaatima laite- ja ohjelmistokokonaisuus. Kokonaisuus saatiin myös sellaiseksi, että NRF24LE1-mikroprosessorin varaan tehtyjen piirilevyjen suunnittelussa voitaisiin jatkossa harkita mahdollisuutta SPI-väylän pois jättämiseen.

4.1.2 Langaton ohjelmointi

Langaton ohjelmointi saatiin toteutettua siten, että NRF24LE1-radiopiirillä laite pystyi ohjelmoimaan toisen samanlaisen laitteen langattomasti. Tämä oli tosin mahdollista vain, jos molemmissa laitteissa oli sopiva ohjelmakoodi. Kohdelaitteissa piti lisäksi käyttää osa muistialueesta Bootloader-ohjelman säilyttämiseen. Kohdelaitteen ohjelmoitavuuden säilyttämiseksi siihen asennettavan ohjelman piti myös olla sellainen, että se pystyisi tarvittaessa käynnistämään itsensä suojatulta muistialueelta. Uudelleenohjelmoitavuuden säilyttämiseksi piti myös huolehtia siitä, että kohdelaite ei muokkaisi muistialue jakoa tai itse suojatun muistialueen sisältöä, koska tuo olisi tuhonnut Bootloader-ohjelman.

4.2 Arviointi

Työtä arvioitiin etenkin sen tavoitteiden onnistumisen suhteen. Lisäksi kerrottiin työn mahdollisesta jatkokehityksestä.

4.2.1 Työ

Työn tavoitteet ja vaatimustaso olivat suhteellisen korkeita, sillä sen teko vaati huomattavan paljon uusien asioiden oppimista. Näitä olivat etenkin sulautettujen laitteiden ohjelmointi, sekä siihen liittyvä mikropiirien ja prosessorien tunteminen. Myös peruselektroniikan hallitsemisessa vaadittiin uusien asioiden oppimista. Tavoitteisiin ja vaatimustasoon nähden työ onnistui hyvin.

Työn tavoitteissa saavutettiin langattoman ohjelmoinnin protokollan kehittäminen, sekä siihen liittyvän ohjelmistokokonaisuuden luominen. Kohdelaitteelle jäi tosin joitain ylimääräisiä vaatimuksia, jotka haittasivat hieman kokonaisuuden helppokäyttöisyyttä, mikä oli alkuperäinen tavoite.

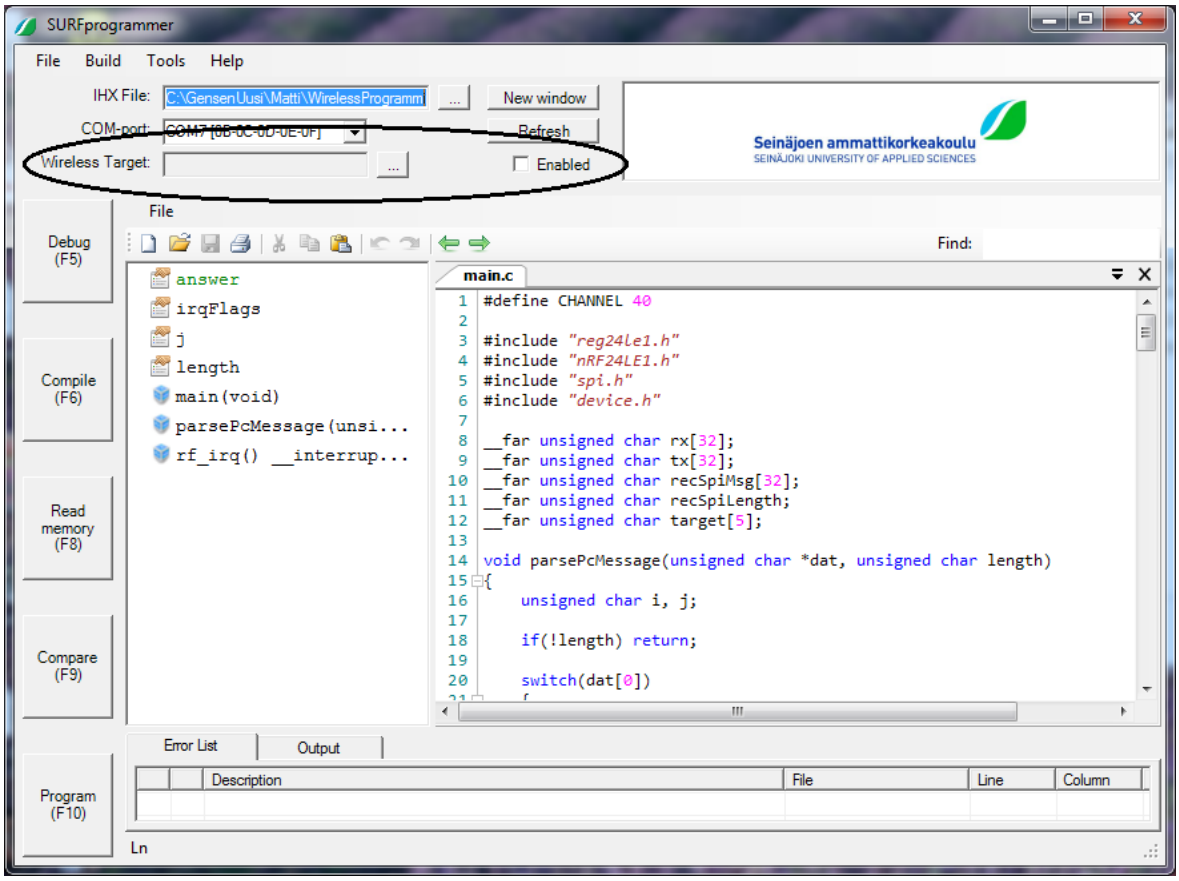
Ohjelmistokokonaisuuden suhteen jouduttiin tinkimään alkuperäisistä suunnitelmasta siinä määrin, että SURFprogrammer-ohjelmaan suunnitellusta langattoman ohjelmoinnin tuesta jouduttiin luopumaan. Luopuminen tehtiin koska langaton ohjelmointi pystyttiin tekemään myös ilman kunnollista käyttöliittymää.

Kohdelaitteelle jäi joitain ylimääräisiä vaatimuksia. Suurin osa näistä vaatimuksista oli täysin perusteltavissa ja hyväksyttävissä, mutta varsinaiselle laitteelle laitettavan ohjelmakoodin suhteen olisi ollut parempi, jos se ei olisi vaatinut tukea suojatulle muistialueelle käynnistämiseen. Tämä vaatimus oli vain niissä tilanteissa, joissa haluttiin laitteen pysyvän uudelleenohjelmoitavana. Tämä ei myöskään ollut ongelma perinteisten piirilevyjen kanssa, joissa oli SPI-väylä.

Työn laitetavoitteiden suhteen onnistuttiin, sillä työ mahdollisti pienempien NRF24LE1-mikroprosessorin varaan tehtyjen piirilevyjen suunnittelun antamalla mahdollisuuden jättää SPI-väylä pois ensimmäisen ohjelmointikerran jälkeen.

Kokonaisuus huomioon ottaen koko työn tavoitteet onnistuivat suhteellisen hyvin.

4.2.2 Jatkokehitys



Kuva 18. SURFprogrammer-ohjelman uusi tuki langattomalle ohjelmoinnille.

SURFprogrammer-ohjelmaan suunniteltu langattoman ohjelmoinnin tuki jäi jatkokehityksen varaan, sillä varsinaisen työn aikana tätä ei päätetty toteuttaa. Tarkoituksena olisi integroida kehitetty laite- ja ohjelmistokokonaisuus SURFprogrammer-ohjelmaan niin että ohjelma jäisi mahdollisimman helppokäyttöiseksi. Tarkoitus olisi, että nykyinen Program-napin painaminen tekisi saman minkä nytkin eli ohjelmoisi laitteen, mutta kohdelaite olisi USB-tikkuun kytketyn laitteen sijaan siihen langattomasti yhteydessä oleva laite. Ohjelmointi tapahtuisi langattomasti. Käyttöliittymään ei tulisi suuria muutoksia, vaan siihen tulisi vain pieniä lisävalinto-

ja langattoman ohjelmoinnin suhteen. Käyttöliittymän muutos saatiin jo työn aikana valmiiksi ja se on ympyröity alue kuvassa (kuva 18).

Tämän työn mahdollistama laitekehitys on myös mahdollista, mutta sen kehitys jäi SeAMK:n varaan. Työ mahdollisti yhä pienempien NRF24LE1-radioprosessoria käyttävien piirilevyjen valmistuksen, koska langattoman ohjelmoinnin ansiosta SPI-väylää ei välttämättä enää tarvinnut pitää pakollisena.

LÄHTEET

AT90USB82/162 Datasheet. 2008. [PDF]. Atmel. [Viitattu 25.5.2014]. Saatavissa: http://www.atmel.com/Images/doc7707.pdf

Huhta M. 2009. Radiopiirin ohjelmointiympäristön kehitys ja langattoman verkon tahdistus. [Opinnäytetyö]. Tietotekniikan koulutusohjelma, Tekniikan yksikkö. Seinäjoen ammattikorkeakoulu. [Viitattu 25.5.2014]. Saatavissa: http://urn.fi/URN:NBN:fi:amk-200912047110

NRF24LE1 Product Specification v1.6. 2010. [PDF]. Nordic Semiconductor. [Viitattu 4.4.2014] Saatavissa: http://www.nordicsemi.com/eng/nordic/download_resource/10875/3/24235901

Palomäki, H. 2013. Connecting objects - langaton tekniikka älykkäässä ympäristössä. Teoksessa: E. Varamäki & S. Päällysaho (toim.). Tapio Varmola - suomalaisen ammattikorkeakoulun rakentaja ja kehittäjä. Seinäjoen ammattikorkeakoulun julkaisusarja. 285–299. [Viitattu 25.5.2014]. Saatavissa: http://urn.fi/URN:ISBN:978-952-5863-48-2

Virrankoski R. 2012. Generic Sensor Network Architecture for Wireless Automation (GENSEN). Vaasa: University of Vaasa. [Viitattu 25.5.2014]. Saatavissa: http://www.uva.fi/materiaali/pdf/isbn_978-952-476-387-5.pdf

LIITTEET

Liite 1: WProgrammer

```
#define CHANNEL 40

#include "reg24le1.h"
#include "NRF24LE1.h"
#include "spi.h"
#include "device.h"

__far unsigned char rx[32];
__far unsigned char tx[32];
__far unsigned char recSpiMsg[32];
__far unsigned char recSpiLength;
__far unsigned char target[5];

void parsePcMessage(unsigned char *dat, unsigned char length)
{
    unsigned char i, j;

    if(!length) return;

    switch(dat[0])
    {
        /*
        NOP
        Usage: 0x00
        */
        case 0x00:
            break;
        /*
        Ask devices
        Usage: 0x01
        */
        case 0x01:
            spiMsg[0] = deviceCount;
            spiCommunication(spiMsg);
            for(i = 0; i < deviceCount; ++i)
            {
                for(j = 0; j < 5; ++j)
                {
                    spiMsg[j] = devices[i].serialNumber[j];
                }
                spiCommunication(spiMsg);
            }
```

```
    break;
/*
Keep target alive
Usage: 0x02 ID
*/
case 0x02:
    if(length < 2) return;
    i = dat[1];
    tx[0] = 0x01;
    for(j = 0; j < 5; ++j)
    {
        target[j] = devices[i].serialNumber[j];
    }
    outputRF(target, tx, 1);
    sendRF();
    break;
/*
Start programming
Usage: 0x03 ID
*/
case 0x03:
    if(length < 2) return;
    i = dat[1];
    tx[0] = 0x02;
    tx[1] = 0xAA;
    for(j = 0; j < 5; ++j)
    {
        target[j] = devices[i].serialNumber[j];
    }
    outputRF(target, tx, 2);
    sendRF();
    //TODO: Tarkista viestin lähettämisen epäonnistuminen.
    break;
/*
Program device / Stop programming
Usage: 0x04 ID Data[0]...Data[x]
[Program] Data: AddrH, AddrL, Byte0, Byte1... ByteX
[Stop Programming] Data: 0xAA
*/
case 0x04:
    if(length < 3) return;
    i = dat[1];
    for(j = 0; j < 5; ++j)
    {
        target[j] = devices[i].serialNumber[j];
    }
    for(j = 0; j < length - 2; ++j)
    {
        tx[j] = dat[2 + j];
    }
    outputRF(target, tx, length - 2);
```

```
            sendRF();
            break;
    }
}

void main(void)
{
    tx[0] = 0xAA;
    tx[1] = 1;
    for(i = 2; i < 5; ++i)
    {
        tx[i] = 0;
    }
    for(i = 0; i < 32; ++i)
    {
        spiMsg[i] = 0;
    }
    initSPI();
    initRF(tx);
    inputON();

    tx[0] = 0;
    outputACK(tx, 1, 1);

    EA = 1;                     // Enable interrupts
    RF = 1;                     // Enable radio interrupt

    while(1)
    {
        waitS(1);
        EA = 0;
        spiMsg[0] = 0x01; //Accepting commands.
        for(i = 1; i < 5; ++i)
        {
            spiMsg[i] = 0;
        }
        recSpiLength = spiCommunicationR(recSpiMsg, spiMsg);
        parsePcMessage(recSpiMsg, recSpiLength);
        cleanDeviceList();
        EA = 1;
    }//while 1
}//main

// Radio interrupt
void rf_irq() __interrupt INTERRUPT_RFIRQ
{
    unsigned char answer;
    unsigned char length = 0;
    unsigned char irqFlags = 0;
    EA = 0;
```

```
    READ_CLEAR_STATUS_REGISTER(&irqFlags);
    irqFlags &= 0b01110000;
    outputACK(tx, 1, 1);

    spiCommunication(spiMsg);

    // If data received
    if(irqFlags & 0b01000000)
    {
        // Read payload
        while(fifoRX() != EMPTY_FIFO)
        {
            length = inputRF(rx);

            tx[0] = 0;
            if(length)
            {
                if(length == 5)
                {
                    answer = checkDevice(rx);
                    if(answer)
                    {
                        spiMsg[0] = 0x00; //Unable to accept commands
(new event).
                        spiMsg[1] = 0x00; //New device contacted.
                        if(answer == 1)
                        {
                            spiMsg[2] = deviceCount - 1;    //ID
                            for(i = 0; i < 5; ++i)
                            {
                                spiMsg[i+3] = rx[i];
//SerialNumber
                            }
                        }
                        else if(answer == 2)
                        {
                            spiMsg[2] = MAX_DEVICES;        //Error:
List is full!
                        }
                        spiCommunication(spiMsg);
                    }
                }
            }
        }
    }
    EA = 1;
}
```

Liite 2: Bootloader

```
#define CHANNEL 40

#include "reg24le1.h"
#include "spi.h"
#include "NRF24LE1.h"
#include "flash.h"

__far unsigned char tx[32]= {0x00};
__far unsigned char rx[32] = {0x00};
__far unsigned char target[5] = {0x00};
unsigned char tryRefresh = 1;
unsigned char serialNumber[5];  //__far ei toimi tässä
unsigned char counter = 0;
unsigned char delay = 0;
unsigned char programmingMode = 0;
__far unsigned char pageContent[256];

void main(void)
{
    //TODO: Tarkistussumman varmistus, sen mukaan NVDataMemory Paril-
linen/Pariton
    unsigned char i=0;
    delay = 0;

    target[0] = 0xAA;
    target[1] = 1;
    for(i = 2; i < 5; ++i)
    {
        target[i] = 0;
    }

    FSR |= 0b00001000;
    for(i = 0; i < 5; ++i)
    {
        serialNumber[i] = infoPageStart[i];
    }
    FSR &= 0b11110111;

    initSPI();
    initRF(serialNumber);                // Own address, subnet
    selectPipe(0);
    initWatchdog();
    startWatchdogSec(10);

    EA = 1;
    RF = 1;
```

```
    while(1)
    {
        waitS(1);
        delay++;
        if(tryRefresh)
        {
            EA = 0;
            outputRF(target, serialNumber, 5);           // load mes-
sage
            EA = 1;
            sendRF();              // send message
        }
    }//while 1
}//main

void parsePacket(unsigned char *dat, unsigned char length)
{
    unsigned char i;
    unsigned char val = 0;
    unsigned int add;
    BitField removePages;

    startWatchdogSec(10);   //Ei resettiä

    if(!programmingMode)
    {
        switch(dat[0])
        {
            /*
            Device connected
            Usage: 0x00 (Automatic with ACK)
            */
            case 0x00:
                if(tryRefresh)
                {
                    tryRefresh = 0;
                    inputON();
                }
                break;
            /*
            Upkeep connection
            Usage: 0x01
            */
            case 0x01:
                break;
            /*
            Start programming
            Usage: 0x02
            */
            case 0x02:
                for(i = 0; i < MAX_PAGE; ++i)
```

```
                {
                    removePages.area[i] = 0;
                }
                if(length >= 4)
                {
                    for(i = dat[2]; i <= dat[3]; ++i)
                    {
                        removePages.area[i] = 1;
                    }
                }
                initFlash(&removePages);
                if(dat[1] == 0xAA) programmingMode = 1;
                break;
            }
        }

        if(programmingMode)
        {
            if(length == 1)
            {
                if(dat[0] == 0xAA)
                {
                    //TODO: Laske tarkistussumma, sen mukaan NVDataMemory
Parillinen/Pariton
                    if(STP == 0)
                    {
                        flashErasePage(35);
                        //for(add = 0xFE00; add <= 0xFFFF; ++add)
                        //{
                        //    flashReadByte(pageContent[add]);
                        //}
                        val = flashReadByte(0xFFFF);
                        val ^= 0b00000001;
                        flashWriteByte(0xFFFF, val);
                    }
                    programmingMode = 0;
                    inputOFF();
                    startWatchdogRaw(1);
                }
            }
            else if(length > 2)
            {
                length -= 2;
                add = 0;
                add |= (dat[0] << 8);
                add |= dat[1];
                for(i = 0; i < length; ++i)
                {
                    val = dat[2 + i];
                    flashWriteByte(add, val);
                }
```

```
            }
        }

        if(!tryRefresh)
        {
            outputACK(serialNumber, 5, 1);
        }
    }

    // Radio interrupt
    void rf_irq() __interrupt INTERRUPT_RFIRQ
    {
        unsigned char irqFlags;
        unsigned char length;
        EA = 0;

        READ_CLEAR_STATUS_REGISTER(&irqFlags);
        irqFlags &= 0b01110000;

        // If data received
        if(irqFlags & 0b01100000)
        {
            // Read payload
            while(fifoRX() != EMPTY_FIFO)
            {
                length = inputRF(rx);
                if(length && delay >= 2)
                {
                    parsePacket(rx, length);
                }
            }
        }
        EA = 1;
    }
```