TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietoliikennetekniikka

Tutkintotyö

Timo Helmijoki

Ohjelmistoradio, USRP-kehitysalusta

Työn valvoja Yliopettaja Mauri Inha

Tampere 2008

Tietotekniikka
Tietoliikennetekniikka
Timo Helmijoki
USRP – kehitysalusta
Tutkintotyö 26 sivua + 6 liitesivua
Työnvalvoja Mauri Inha
Työnteettäjä Tampereen ammattikorkeakoulu
Joulukuu 2008
Hakusanat USRP, software radio, software defined radio, Gnu Radio

TIIVISTELMÄ

Ohjelmistoradio on laajasti käytetty termi eikä nykyään ole aina varmuutta, kun puhutaan ohjelmistoradiosta, millaisesta radiosta puhutaan. Tämän takia käyn ensimmäisenä läpi radion määrittelemisen. Tämän jälkeen mainitsen muutaman projektin, joissa ohjelmistoradiota kehitetään. Tämän jälkeen päästään itse asiaan, eli käyn läpi ohjelmistoradion USRP-kehitysalustan ominaisuudet ja käyttämisen Gnu Radio ohjelmistolla.

Joe Mitol on sanonut seuraava ohjelmistoradiosta:

”Ohjelmistoradio on radio, jossa kanavan modulaation aaltomuotoon määritelty ohjelmallisesti. Toisin sanoen aaltomuodot generoidaan digitaalisesta signaalista. Digitaalisignaali muunnetaan analogiseksi signaaliksi laajakaistaisella DA-muuntimella ja mahdollisesti ylös-muunnetaan IF-taajuudesta RF-taajuudelle. Vastaanotin toimii käänteisesti, käyttäen laajakaistaista AD-muunninta, joka nappaa kaikki ohjelmistoradion kanavat. Kanavan aaltomuoto erotetaan, alas-muunnetaan ja ilmaistaan ohjelmistolla. Ohjelmistoa ajetaan yleiskäyttöisellä prosessorilla./1;2;4/

Computer science
Telecommunications engineering
Helmijoki, Timo writer
Thesis 26 pages + 16 appendices
Thesis supervisor Mauri Inha
Comissioning Comapany Tampere University Of Applied-Sciences
December 2008
Keywords USRP, software radio, software defined radio, Gnu Radio

ABSTRACT

The software-defined radio will be a term that has been widely used and nowadays there will be not always a certainty when it is talked about the software-defined radio about what kind of radio it is talked. Because of this I go through the specifying of the radio as the first one. After this I mention a few projects in which the software-defined radio is developed. After this the properties of the development base of the software-defined radio and use are achieved a matter itself, in other words are gone through. Universal Software Radio Peripheral (USRP) is as a development base equipment which is used on Gnu Radio software.

Joe Mitol has said about the software-defined radio, following:

"A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband DAC and then possibly upconverted from IF to RF. The receiver, similarly, employs a wideband Analog to Digital Converter (ADC) that captures all of the channels of the software radio node. The receiver then extracts, downconverts and demodulates the channel waveform using software on a general purpose processor." / 1;2;4/

KÄYTETYT TERMIT JA LYHENTEET

AD	Analog to Digital
BFN	Beamforming Network
CF	Core Framework
CIC	Cascade Integrator-comb
CORBA	Common Object Request Broker Architecture
DA	Digital to Analog
DDC	Digital Down Converter
DUC	Digital Up Converter
Flow Graph	Vuokaavio, vuoverkko
FSRP	Finnish Foftware Radio Programme
GPL	General Public License
GRC	Gnu Radio Companion
HPSDR	High Performance Software Defined Radio
HR	Hardware Radio
IF	Intermediate frequency
ISR	Ideal Software Radio
JTRS	Joint Tactical Radio System
MEMS	Micro-Electro-Mechanical-System
MIMO	Multiple-input and multiple-output
OBSAI	Open Base Station Architecture Initiative
PGA	Programmable Gain Amplifer
POSIX	Portable Operating System Interface
RF	Radio Frequency
USRP	Universal Software Radio Peripheral
USRP2	Universal Software Radio Peripheral 2
SA	Smart Antenna
SCA	Software Communication Architecture
SCR	Software Controlled Radio
SDR	Software Defined Radio
STAP	Space-Time Adaptive Processing
USRP	Universal Software Radio Peripheral
WF	Waveform

ALKUSANAT

Erään kurssin tehtävän myötä tutustuin ohjelmistoradioon ja löysin Universal Software Radio Peripheral (USRP), joka on ohjelmistoradion kehitysalusta ja avoin sellainen. Alustaa käytetään Gnu Radiolla ohjelmistolla, jota levitetään GPL-lisenssillä.

Tahtoisin kiittää Tietekniikan koulutuspäällikköä Ari Rantalaa ja muita, jotka mahdollistivat USRP-laitteiston hankinnan ja pääsin tutustumaan ohjelmistoradion maailmaan konkreettisesti.

Tampereella 1. joulukuuta 2008

Timo Helmijoki

SISÄLLYS

1 JOHDANTO

Radioita on kehitetty pitemmän aikaa ohjelmistolla hallittavaksi ja toteutuksia on ainakin vuosikymmenen verran ollut käytössä (GSM puhelimet). Nykyään ollaan tilanteessa, jossa radiot ovat siirtymässä ohjelmallisesti määriteltäväksi. Tästä on alkanut tulla todellisuutta viimeisten 5 vuoden aikana, kun yleiskäyttöiset prosessorit ovat saavuttaneet ratkaisevan laskentakapasiteetin.
Ohjelmistolla määritetty radio mahdollistaa kognitiivisen radion kehittämisen, koska kognitiivinen radio edellyttää nopeaa spektrianalysaattoria radiossa.
Työssä tarkoituksena oli kehittää tietoliikennetekniikan laboratoriotyö ohjelmistoradioon liittyen. Työssä käytin USRP-laittestoa ja Gnu Radio ohjelmistoa.

2 RADIOIDEN MÄÄRITTÄMINEN

Ensin määritellään mitä tarkoitetaan ohjelmistoradiolla. Yksi tapa jakaa radiot eri luokkiin on määritellä ne ohjelmiston käytön mukaan. SDR Forumin mukaan radiot jaetaan viiteen eri luokkaan.

- HW Radio (HR):
  Perinteinen radio, joka on toteutettu analogisilla tai lähes vain analogisilla komponenteilla. Radio voi siis sisältää logiikkaa, mutta ei varsinaisesti ohjelmistoa. LA-puhelimet ovat tästä esimerkki.

- Ohjelmallisesti kontrolloitu radio (SCR):
  Säätötoimintoja on toteutettu ohjelmistolla, tosin vain rajoitetusti. Taajuusaluetta ja modulaatiotyyppiä ei voi esimerkiksi vaihtaa ohjelmallisesti. GSM-puhelimet ovat tästä hyvä esimerkki.

- Ohjelmistolla määritetty radio (SDR):
  Ohjelmistolla on mahdollista kontrolloida modulaatiota, valita laaja- tai kapeakaista toiminta, yhteyden turvallisuutta ja aaltomuotoja. Taajuusalueen vaihtoa varten kuitenkin saattaa olla tarpeellista vaihtaa heterodyne RF-etupää (Front End) ja antenni.

- Ideaalinen ohjelmistoradio (ISR):
  RF-taajuus muutetaan suoraan analogisesta digitaaliseksi. Toisin sanoen etuosa poistetaan välistä. Analogiamuunnos tehdään ainoastaan antennille, kaiuttimelle ja mikrofonille.

– Lopullinen ohjelmistoradio (USR):
  Määritelmässä ainoastaan vertailua varten. Kyseessä on idealistinen kuvaus radiosta, joka kykenee toimimaan missä tahansa radiojärjestelmässä ja voi vaihtaa ilmarajapintaa yhdessä millisekunnissa.

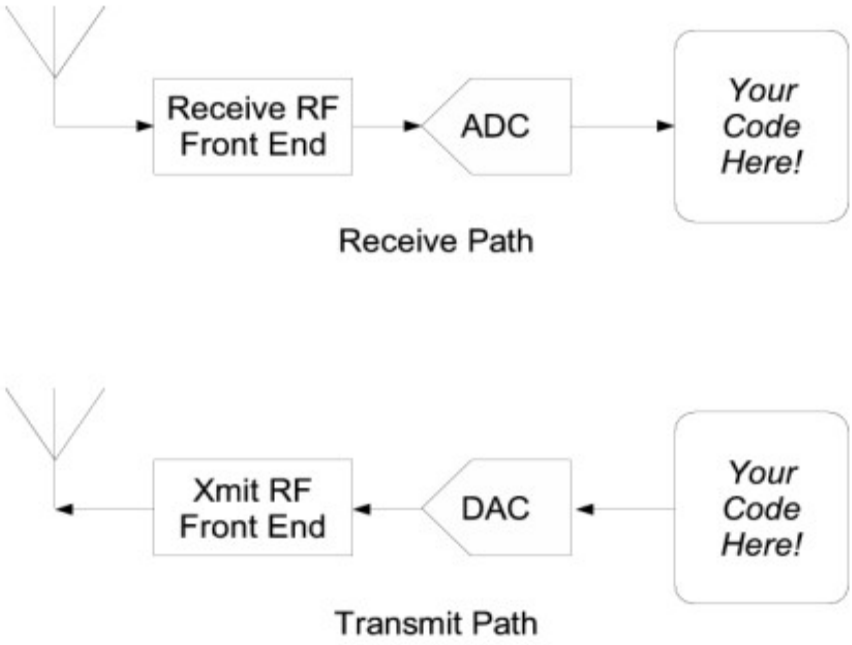
Tästä eteenpäin kun puhun ohjelmistoradiosta, tarkoitan ohjelmallisesti määritettyä radiota. /1/

3 OHJELMISTORADIO (SDR)

Joe Mitol on määrittänyt ohjelmistoradion seuraavasti vuonna 1992:

"A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband DAC and then possibly upconverted from IF to RF. The receiver, similarly, employs a wideband Analog to Digital Converter (ADC) that captures all of the channels of the software radio node. The receiver then extracts, downconverts and demodulates the channel waveform using software on a general purpose processor."/1;2;4/

Toisin sanoen ohjelmistoradiossa RF-signaalin AD ja DA-muunnos tapahtuu mahdollisimman lähellä antennia ja ohjelmistoa pyöritetään yleiskäyttöisellä prosessorilla. Ohjelmistoradiossa radion toimintataajuus, RF-kaistanleveys, modulaatio ja kaikki kantataajuiset signaalinkäsittelytoiminnot ovat ohjelmallisesti määriteltävissä.
Etuna saadaan se, että samalla laitteistolla voidaan olla mahdollisesti yhteydessä useampaan radiotie rajapintaan tai yhtä aikaa useammassa radiojärjestelmässä vain aaltomuotoa vaihtamalla. Tämä mahdollistaa halvemman laitteiston tekemisen, ettei tarvitse erikseen suunnitella piirejä esimerkiksi WLANia, GPS:ää ja Wibreetä varten erikseen, vaan piiriksi riittää yksi tarpeeksi tehokas signaalinkäsittelyyn tehty prosessori. Tällä hetkellä ollaan ohjelmistoradiossa kolmannessa sukupolvessa. Tämä tarkoittaa sitä, että ohjelmistoarkkitehtuuri on oikeasti avoin, mikä mahdollistaa aaltomuodot kolmannelta osapuolelta. Toisin sanoen laitteiston ja ohjelmiston kehitys on eriytetty toisistaan. Ensimmäisen sukupolven ohjelmistoradiototeutukset perustuivat suljettuun arkkitehtuuriin ja rajatusti uudelleenohjelmoitaviin ohjelmistoihin. Toisen sukupolven radiot perustuivat suljettuun laitteisto- ja ohjelmistoarkkitehtuuriin. /9/



***Kuva 1**.Tyypillinen ohjelmistoradion lohkokaavio./2/*

Ohjelmistoradion käytännön toteutukset perustuvat kuvassa olevaan lohkokaavioon (kuva 1). Ennen AD/ DA-muunnosta löytyy siis joku heterodyne tyyppinen RF-etupää. RF-etupäähän kuuluvat etuaste- ja välitaajuusosat.
Alle 40 MHz:n taajuuteen on nykyään mahdollista toteuttaa ideaalinen ohjelmistoradio. Tällaisessa ohjelmistoradiossa ei ole etupäätä. Etupään sijaan löytyy vaan etuaste, joten AD/ DA-muunnin kytketään suoraan RF-taajuudelle.
Ohjelmistoradio mahdollistaa sellaisen mielenkiintoisen radion toteuttamisen kuin kognitiivisen radion. Kognitiivinen radio tarkoittaa sitä , että radio tarkkailee RF-spektriä ja käyttää taajuutta, joka on vapaana. Toisin sanoen kognitiivisesta radiosta täytyy löytyä spektrianalysaattori.

4 SIGNAALIEN MUUNTAMINEN

Kun halutaan muuntaa jatkuva-aikainen 20 MHz:n signaalia digitaaliseksi. Silloin haluamme ottaa näytteen vähintään 20 µs:n välein, eli näytteenottotaajuus on 40 MHz:ä. Tarvittava näytteenottotaajuus saadaan seuraavasta kaavasta 1. Nyquistin taajuudeksi kutsutaan taajuutta, joka puolet näytteenottotaajuudesta ($F_s/2$).

$$F_s = \frac{1}{T}$$
$$F_s = näytteenottotaajuus$$
$$T = kahden\ peräkkäisen\ näytteen\ väli$$

Kaava 1

4.1 AD-muuntimien toimintaperiaatteita

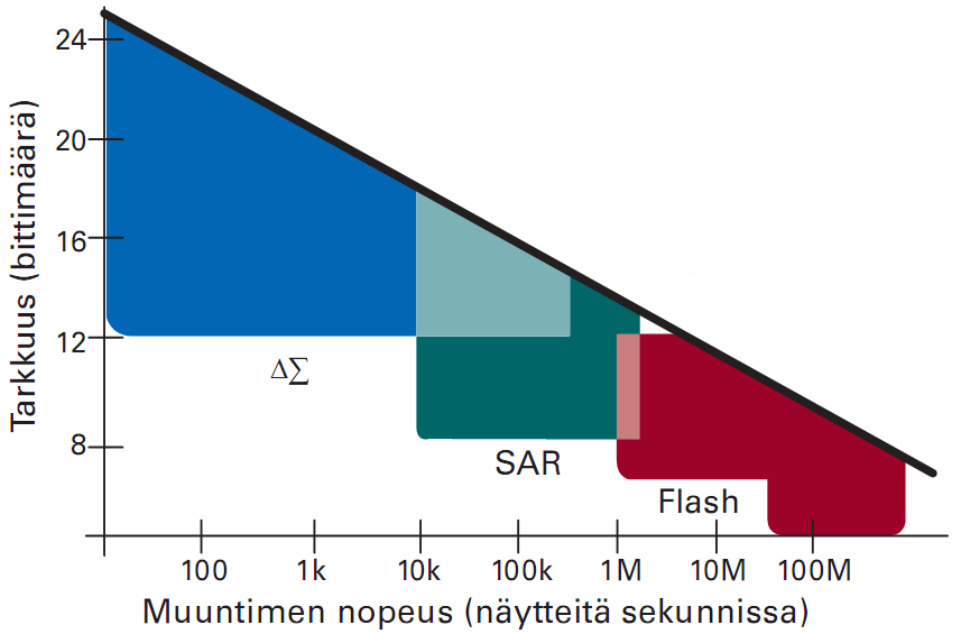
Signaalinmuuntimille on kehitetty useita erilaisia toimintaperiaatteita. AD-muuntimista nopein ja suoraviivaisin periaate on flash-muunnin. Siinä signaalijännite viedään samanaikaisesti vertailupiireihin. Flash-muuntimet ovat nopeimpia, mutta epätarkimpia (kuva 2). Tämän tyyppisellä muuntimella viive on jokseenkin sama kuin yhden vertailupiirin viive.

Peräkkäis-approksimaatioon (SAR) perustuva muunnin on yleisin tyyppi, kun muunnetaan enintään muutaman megahertsin näytetaajuuksista yhdistettynä enintään 16 bitin nimelliseen tarkkuuteen. Peräkkäis-approksimaatiossa jännitealue ensin jaetaan kahteen osaan ja tarkistetaan kumpaan puoliskoon muunnettava jännite kuuluu. Tämän jälkeen tarkistetaan kumpaan neljäsosaan jännite asettuu, ja näin muunnos etenee bitti kerrallaan. Toimintaperiaate on melko hidas, mutta soveltuu valmistusteknisesti CMOS-mikropiiriteknologiaan.

Kaksoisluiskamuunnin on kaikkein hitain muunnintyyppi. Kyseinen muunnin sietää melko hyvin suuritaajuisi häiriöitä. Tämä on yleisin käytetty muunnintyyppi digitaalisissa yleismittareissa.

Deltasigma-modulaattoria käytetään, kun pyritään suurimpaan mahdolliseen tarkkuuteen eli 24 bitin tarkkuuteen. ΔΣ-modulaattorissa ensin muodostetaan erotus muunnettavan signaalin ja jännitealueen ylä- tai alarajaa vastaavan jännitteen välillä. Nämä jännitteet saadaan takaisinkytkennällä modulaattorin binäärisestä antosignaalista. Erosignaali integroidaan ja muunnetaan digitaaliseksi yhden bitin tarkkuuksisena. ΔΣ-modulaattorissa näytetaajuus on aina suurempi kuin Nyquist-taajuus.

Tällä hetkellä RF-käyttöön paras muunnin tyyppi on liukuhihnatyyppiset muuntimet. Tämä muunnintyyppi on sukua peräkkäisaproksimaatio muuntimille. /14/



***Kuva 2**. Kolmen yleisimmän muuntotekniikan nopeus/tarkkuus./14/*

4.2 DA-muuntimien toimintaperiaatteita

DA-muuntimissa käytetään yleensä kahta menetelmää. R-2R-vastusverkkoa käytetään kohtalaisilla tarkkuuksilla. Vastusverkossa signaalien bitit kytkevät vastuksia joko referenssijännitteeseen tai nollaan. Muuntimen antosignaali on virtamuotoinen. Kyseisen muuntimen merkittävin ongelma on antosignaalin virtapiikit vastusverkon kytkimien nopeuserojen takia.
Tarkimmat DA-muuntimet perustuvat ΔΣ-modulaattoriin, joka voidaan toteuttaa analogisena tai digitaalisena. Muuntimen antosignaali pitää interpoloida, jotta siitä saadaan jatkuva-aikainen signaali. /14/

5 ÄLYKKÄÄT ANTENNIT

Ohjelmistoradion eräs idea on mahdollistaa pääsy useisiin eri langattomiin verkkoihin ja vielä toimia useissa verkoissa samaan aikaan. Tämä asettaa antenni teknologialle uusia haasteita.
Tähän haasteeseen etsitään ratkaisua älykkäillä antenneilla (Smart antenna). Älykäs antenni koostuu useista antenneista/ antenni elementeistä (tyypillisesto 4-12) ja signaalinkäsittely kapasiteetista. Antennin tekee siis älykkääksi signaalinkäsittely, toisin sanoen antennin toiminta määritetään ohjelmallisesti. Älykkäitä antennijärjestelmiä on olemassa kahta perustyyppiä keilaa kytkeviä (eng. switched beam) ja adaptiivisia järjestelmiä (adaptive array). Näistä kehittyneempi on adaptiivinen antennijärjestelmä.

Seuraavaksi esitetty jako on SDR Forumin älyantennin sovellusliitynnän määritelmässä (Smart Antenna Api Specification) älyantennit jaetaan neljään luokkaan: /17;20/

- Beamforming (keilanmuodostus)
- Space time equalization (suom. tila-aika korjaus)
- Diversity combining (suom. diversiteetin yhdistäminen)
- Multiple input multiple output (MIMO) processing.

5.1 Beamforming (keilanmuodostus)

Älykkäässä keilanmuodostus antennissa vastaanotossa algoritmi lähinnä vertaa haluttua ja ei halutta signaalia. Digitaalisella käsittelyllä täytyy olla kyky muokata säteilykuviota vastaanotossa/lähetyksessä ja kyky ohjata keila halutun signaalin suuntaan ja olla ohjaamatta mitään häiritsevää signaalia kohti. Tällä saadaan aikaiseksi alhainen kanavasignaalien keskinäinen häiriö ja suuri antennivahvistus halutulle signaalille. Keilanmuodostus järjestelmä voidaan toteuttaa kahdella tavalla. Keilaa kytkevällä tai adaptiivisella antennijärjestelmällä. Keilaa kytkevässä järjestelmässä keilanmuodostus verkkoa (BFN) seuraa RF kytkimet, jotka toimivat analogisessa RF-signaalissa .Kytkimiä ohjataan kontrollilogiikalla, joka valitsee sopivan keilan. Tässä vaadittava tiedonkäsittely on minimaalista, koska ohjauslogiikalla on esivalitut määritelmät halutulle signaalille. Adaptiivisessa keilanmuodostuksessa signaali valitaan antennivahvistuksen tai painotuksen perusteella algoritmillä, joka toimii järjestelmän digitaalisella puolella. Adaptiivinen toteutustapa on näistä kahdesta kehittyneempi. /17; 20, s.5/

5.2 Space-time equalization (suom. Tila-aika korjaus)

Tila-aika korjauksessa (Space-time equalization) yleensä oletetaan, että kiinnostava signaali on kapeakaistainen verrattuna kanavan koherenssiseen kaistanleveyteen ja näin ollen kärsii tasaisesta häipymästä signaalin kaistanleveydellä. Monitiehäipymä voidaan ilmaista myös sirontana vastaanotossa.
Poistaaksemme taajuusvääristymän vaikutuksen, otamme käyttöön ajallisen tietojenkäsittelyn (eng. temporal processing) jokaiselle antennielementille. Ajallinen yhdistäminen (eng. spatial combining) lieventää kanavaan indusoituvaa selektiivistä taajuushäipymää ja tuottaa antennivahvistusta. Tällaista järjestelmää kutsutaan mukautuvaksi tila-aika adaptiiviseksi käsittelyksi (eng. space-time adaptive processing) tai tila-aika taajuuskorjaukseksi (space-time adaptive equalization). /17/

5.3 Diversity combining (suom. Diversiteetti yhdistäminen)

Langattomassa tietoliikenteessä huomattava rajoittava tekijä on monitiehäipymä, jossa vastaanotetun signaalin amplitudi vaihtelee kokoajan. Tietoliikenneyhteyksissä esiintyvä syvä häipymä (deep fade) voi häiritä tavanomaista- tai yksiantennista järjestelmää. Syvässä häipymässä signaalin amplitudista tulee hyvin pieni.
Kun käytetään useita antenneja tulee vähemmän todennäköiseksi, että kaksi tai useampi antenni joutuisi syvään häipymään samanaikaisesti. Moni yksinkertainen algoritmi, kuten suurimpien yhdistäminen (eng. maximal ratio combining), yhtäsuurten vahvistuksien yhdistäminen (eng. equal gain combining) ja eriaikaisten valinta (selection diversity) on kehitetty hyödyntämään antenniryhmä järjestelmiä diversiteetti vastaanotossa. Algoritmit valitsee vastaanotettavat signaalit samoin kuin keilan muodostuksessakin, mutta pohjautuen eri kriteereihin. /17/

5.4 Multiple-Input Multiple-Output (MIMO)

Kuten nimikin sanoo antenni ryhmä käsittelee vastaanottoa ja lähetystä. MIMO järjestelmiä löytyy kahta erilaista versiota. Yksi versio käyttää tilallista limitystä (spatial multiplexing) kasvattamaan datan määrä annetulla kaistanleveydellä ja toinen käyttää tila-aika koodausta (eng. space time coding) diversiteettien yhdistämiseen häipymän vähentämiseksi. Limitetyssä järjestelmässä data muunnetaan sarjasta rinnakkaiseen ja lähetetään samanaikaisesti useammasta antennielementistä. Vastaanotin käyttää myös useita antennielementtejä signaalin vastaanottoon ja käyttää suurimman todennäköisyyden algoritmiä palauttaakseen samanaikaisesti lähetetyt symbolit. Pääoletuksena lähetysympäristö tuottaa voimakasta sirontaa. /17/

6 SCA-OHJELMISTORADIO ARKKITEHTUURI

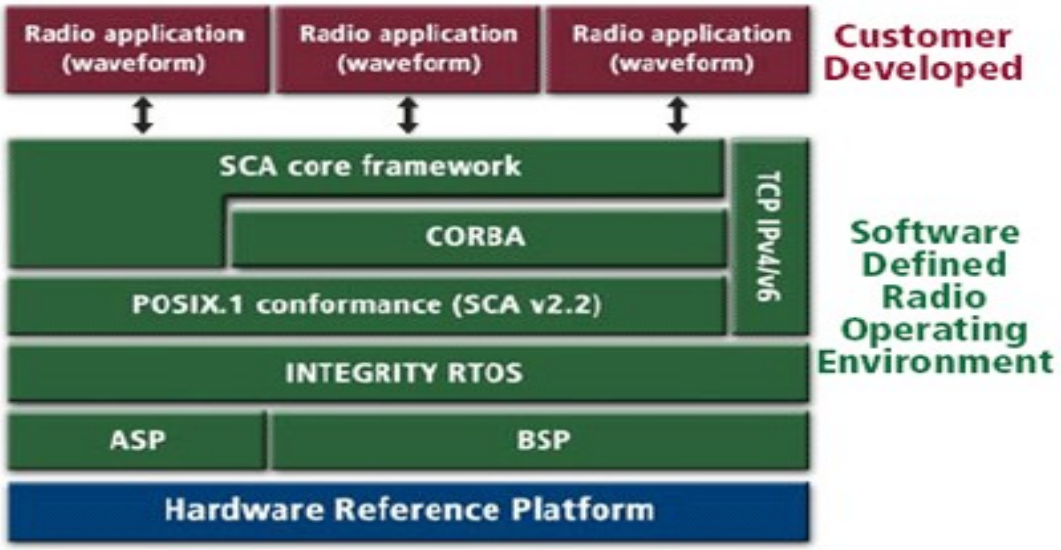
Software Communications Architecture (SCA) on avoin ohjelmistoradio arkkitehtuuri, joka on kehitetty Yhdysvaltojen JTRS projektissa. SCA-määrittely koostuu säännöistä ja protokollista, jotka määrittelevät avoimen arkkitehtuurin ohjelmistoradion sovelluksille. Siitä on muodostunut De-facto standardi sotilaskäyttöön. Tällä hetkellä standardi on versiossa 2.2.2. SDR foorumi on työstää muutos ehdotuksia SCA standardiin, jotta se sopisi paremmin siviilikäyttöön.

SCA:n määrittelemän ohjelmistoarkkitehtuurin tärkeimmät tavoitteet ovat seuraavat:

- Käyttää kaupallisia protokollia ja tuotteita
- Erottaa ydin- ja sovellusohjelmistot laitteistosta avoimien, kaupallisten ohjelmistorajapintojen avulla

– Tarjota hajautettu käyttäjäympäristö CORBA-arkkitehtuurin avulla sovellusohjelmistojen siirrettävyyden, uudelleenkäytön ja skaalattavuuden takaamiseksi.

Ohjelmistoradion ydin (kuva 3) CF-hallintaohjelmisto (Core Framework). Tämä osa ohjaa aaltomuotojen (Waveform) lataamista. Aaltomuodolla tarkoitetaan koko radiosovellusta modeemista verkkotoimintoihin. Tämä pitää sisällään siis ISO-mallin kolme ensimmäistä kerrosta. /18;7/



Kuva 3.*SCA rakenne./18/*

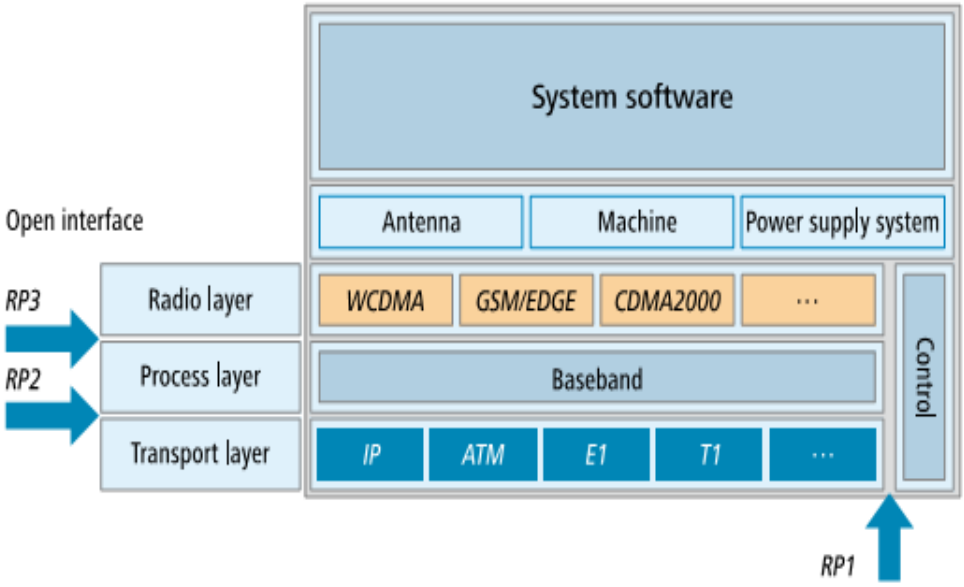
7 OHJELMISTORADIO TUKIASEMASSA

Ohjelmistoradiotekniikka on sinällään käytössä monissa tukiasematuotteissa. Kehitystä on kuitenkin hidastanut tukiaseman rakenteen standardoimattomuus. Tämä on merkinnyt, että tukiaseman sisäinen rakenne on vaihdellut/ vaihtelee valmistajan mukaan. Rakenteen standardoimiseksi on menossa kaksi hanketta Open Base Station Architecture Initiative (OBSAI) ja Common Public Radio Interface(CPRI). OBSAI perustettiin 2002 Nokian, LG Electronicsin ha Samsungin toimesta ja CPRI perustettiin 2003 Ericssonin, Nec:in, Nortel Networks:in, Siemensin ja Huawein toimesta. Projektit kilpailevat ainakin epäsuorasti avoimen tukiasema arkkitehtuurin luomisessa. /11/

7.1 Open Base Station Architecture Initiative

OBSAI-standardissa tukiasema muodostuu neljästä pääyksiköstä. Pääyksiköt ovat kantataajuusyksikkö (base station module), radiotaajuusyksikkö (radio module), siirtoyhteysyksikkö (transport module) ja ohjausyksikkö (control module). Yksiköiden

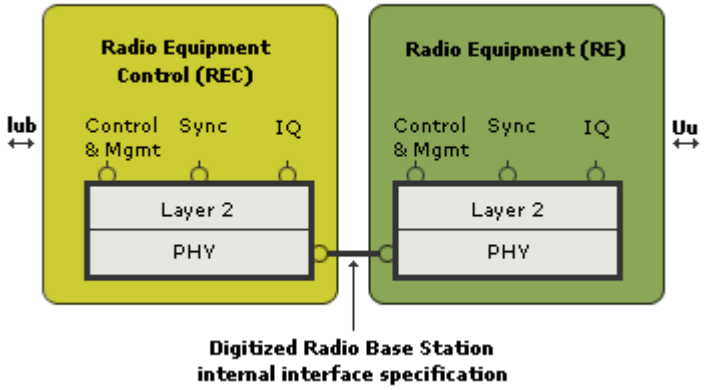
välille on määritetty kolme digitaalista rajapintaa RP1, RP2 ja RP3. Standardiin sisältyy neljäs rajapinta RP4 (kuva 4), joka sijoittuu tehonsyötön ja tukiaseman välille. Kantataajuusyksikössä tapahtuu kantataajuisen signaalin käsittely. Kantataajuusyksikkö liittyy RP3-rajapinnalla radiotaajuusyksikköön. Siirtoyhteysyksikkö hoitaa yhteydet muuhun verkkoon. Siirtoyhteysyksikkö liittyy RP2-rajapinnalla kantataajuusyksikköön. Ohjausyksikkö liittyy kantataajuusyksikköön RP1-rajapinnalla. /11/



***Kuva 4**.OBSAI:n rakenne./8/*

7.2 Common Public Radio Interface

CPRI-standardissa tukiasema jaetaan kahteen osaan, ohjausyksikköön (Radio equipment Control) ja radiotaajuusyksikköön (Radio Frequency Unit). Näitä yhdistää yksi rajapinta, joka on digitaalinen ja sarja muotoinen (kuva 5). /20/



***Kuva 5.** CPRI:n rakenne./8/*

8 OHJELMISTORADIOPROJEKTEJA

Ohjelmistoradiota kehitetään armeijoiden, kaupallisten ja avointen projektien voimin. Armeijoiden projekteja ovat esimerkiksi seuraavat:
Joint Tactical Radio System (JTRS) järjestelmä, joka on Yhdysvaltain puolustusvoimien projekti, ja Finnish Software Radio Programme (FSRP) on Suomen puolustusvoimien hanke.
Kolme ehkä tärkeintä avointa projektia ovat Flexradion FLEX-5000, GNU Radio ja HPSDR (High Performance SDR). Yhteinen piirre näillä projekteilla on se, että lopullinen signaalin käsittely tapahtuu PC:ssä. Muita avoimia ja kaupallisia ohjelmistoradio projekteja ovat mm. SoftRock, TinySDR. /6;11/

8.1 JTRS-projekti

JTRS (Joint Tactical Radio System) on Yhdysvaltojen ohjelmistoradio hanke sotilaskäyttöön. Projektissa on kehitetty ohjelmistoradioarkkitehtuuri SCA (Software Communication Architecture), joka on tullut sotilaspuolella De facto standardiksi.

8.2 FSRP-projekti

FSRP on puolustusvoimien ohjelmistoradio hanke. Projektissa kehitetään kaupallisia komponentteja hyödyntävä ohjelmistoradioalusta, mukautuva laajakaistainen aaltomuoto, paikannusaaltomuoto ja adaptiivinen antennijärjestelmä.
Ensimmäinen vaihe oli kehittää demonstraatio ohjelmistoradiosta laboratorioympäristössä. 2001 projektin päätoimijaksi valittiin Elektrobit. Vuonna 2003 puolustusvoimat ja Elektrobit solmivat 3-vuotisen sopimuksen demonstraatiohankkeesta, jonka arvo on 12 miljoonaa. Projektissa ohjelmistoradion kehittäminen perustuu radiolaitteiden kehittämiseen SCA arkkitehtuuria noudattaen, joka mahdollisti suunnittelun "third party waweforms on third party platforms"-periaatteella.

8.3 Flexradion FLEX-5000

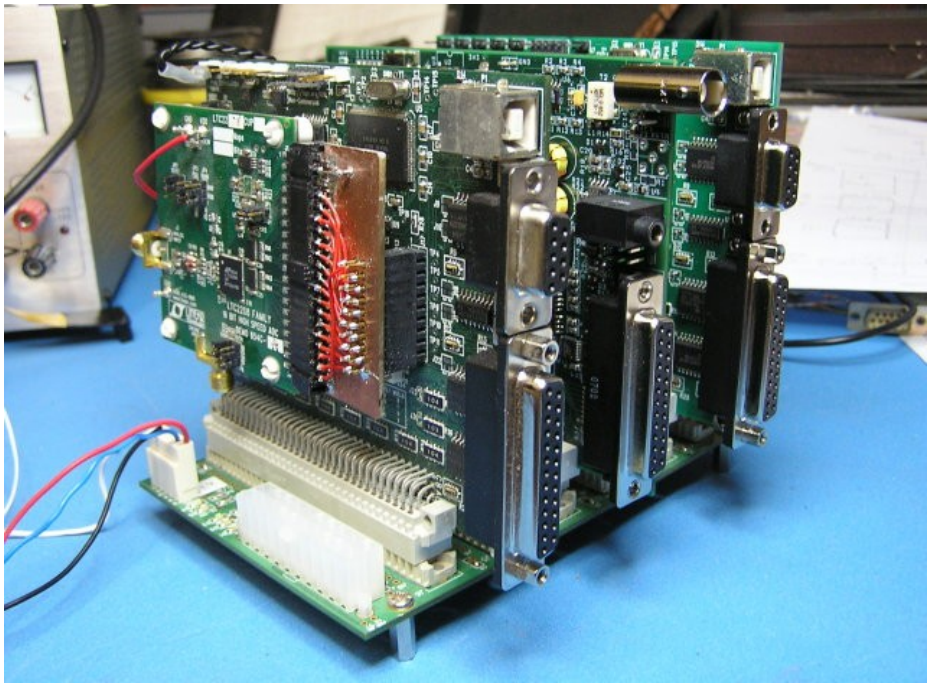
FlexRadio Systems kehittää ohjelmistoradiota radioamatööreille ja sen perustaja on Gerald Youngblood. Ensimmäinen ohjelmistoradio julkaistiin vuonna 2003 ja se tunnetaan nimellä SDR-1000. Vuonna 2007 julkaistiin seuraava sukupolvi, joka kantaa nimeä FLEX-5000. FLEX-5000 on olemassa kahta versiota, jossa toisessa tietokone on integroitu samaan pakettiin FLEX-5000C (kuva 5) ja toisessa radio pitää liittää erilliseen

tietokoneeseen FireWire-liitännällä (IEEE-1394a) FLEX-5000a. Tänä vuonna 2008 julkaistiin myös FLEX-3000, joka on liikuteltava eli huomattavasti kevyempi kuin aikaisemmat mallit. Ensi vuonna julkaistaan halpa (alle 500$) ja pieni tehoinen (1W) FLEX-1500. FLEX-3000 ja FLEX-5000 malleissa puhutaan 1500$ alkavista hinnoista. Radioita hallitaan FlexRadion PowerSDR ohjelmistolla, joka on julkaistu GPL lisenssillä./4/

***Kuva 6**. FLEX-5000C edestä./4/*

8.4 High Performance SDR

HPSDR on avoimen koodin ja raudan ohjelmistoradio projekti, joka on suunnattu radioamatööreille ja lyhyiden aaltojen kuuntelijoille. Projektin tavoitteena on suunnitella radio, joka koostuu moduuleista. Moduulit on suunniteltu toimimaan yksittäin tai ryhmissä ja ne kytketään ennalta määrättyyn väylään. Kuvassa (kuva 6) olevassa kokoonpanossa moduulit ovat vasemmalta oikealle seuraavat: Digitaalinen alas muunnin, Digitaalinen



***Kuva 7**. HPSDR./19/*

ylös muunnin ja kontrollointikortti./19/

8.5 GNU Radio

GNU Radio on avoin ohjelmistoradioprojekti, jossa kehitetään ohjelmistoa ja laitteistoa (USRP ja USRP2). GNU Radio on siis kokoelma työkaluja ohjelmistoradion toteuttamiseen ja kehittämiseen. Gnu Radiota jaetaan GNU GPL vapaa ohjelmistolisenssillä.

GPL lisenssillä kehitetyn ohjelmiston lähdekoodi pitää olla saatavilla, jos ohjelmistoa tai sen muunnosta levitetään. Muunnosta ei tarvitse kuitenkaan antaa toiselle käyttöön, jos ohjelmistoa ei levitetä. Ohjelmiston jakelua eikä myyntiä ole rajoitettu. Tämä tarkoittaa, että ohjelmisto ei ole välttämättä ilmainen vaikka lähdekoodi on saatavilla.

GNU radion signaalinkäsittely lohkot ovat kirjoitettu C++:lla. Lohkot yhdistetään toisiinsa Python:lla, jolla käyttöliittymäkin on luoto. Gnu Radion ylläpitää nykyään mm. Eric Blossom. GNU radio tukee ”kaikkia käyttöjärjestelmiä”, siis Linuxia, NetBSD:tä, MAC OS X:ää ja Windowsia. GNU radiota käytetään esimerkiksi yliopistollisessa opetuksessa ja tutkimuksessa.

Projektissa on kehitetyt ohjelmistoradion kehitysalustat, joista käytetään nimeä USRP ja USRP2, ne ovat myös avoimia. Toisin sanoen emolevyn ja tytärkorttien kytkentäkaaviot, Layout ja firmware/ ajurit ovat vapaasti saatavilla.

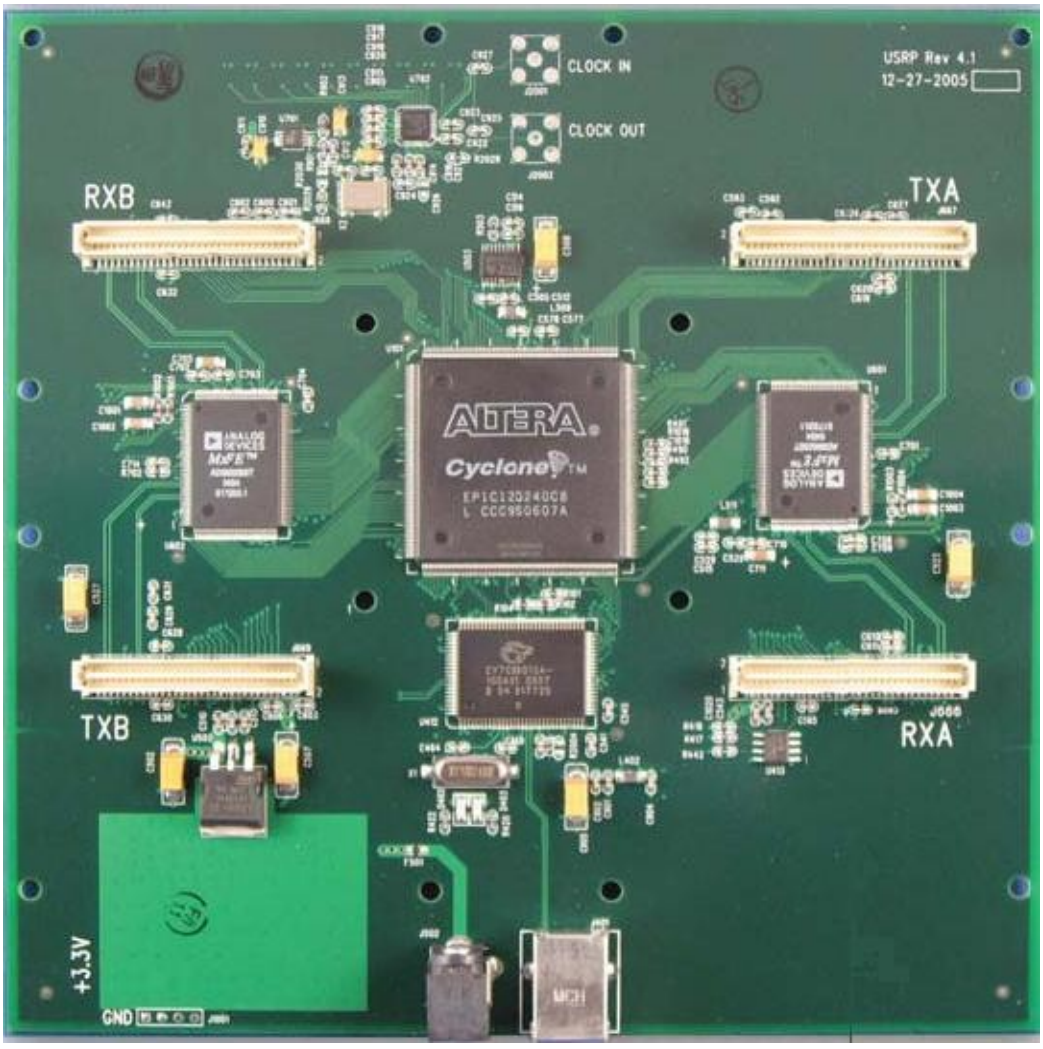
USRP- ja USRP2-kehitysalustoja kehittää Matt Ettus, joka myy alustoja yrityksensä Ettus Research LLC:n kautta. USRP:n emolevyn hinta on 700$ ja USRP2:n hinta on 1400$. Tytärkorttien hinnat vaihtelevat 75$ - 400$ välillä. USRP alusta kytketään PC:en USB2 väylään ja USRP2 Gigabitin Ethernet liitäntään. /2;3/

9 USRP LAITTEISTO

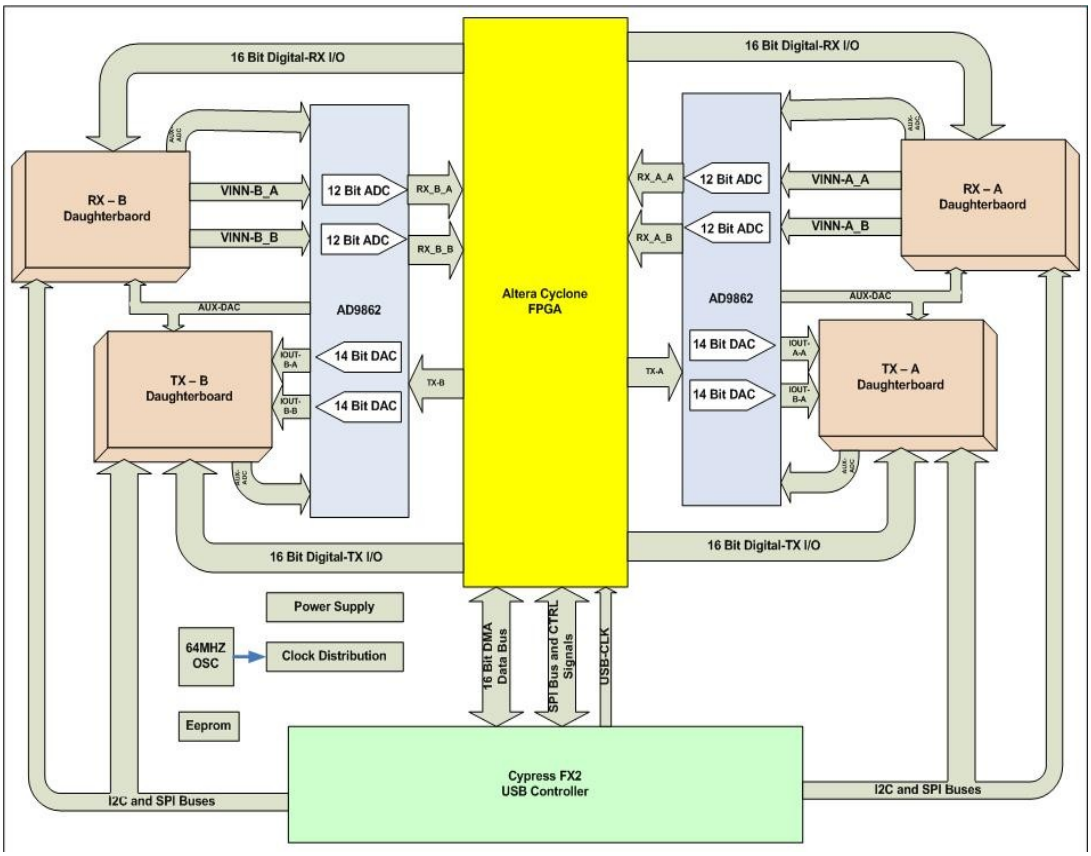
USRP laitteisto koostuu emolevystä ja tytärkorteista. Suunnittelufilosofia on ollut se, että pc:n prosessori hoitaa aaltomuodon käsittelyn, kuten moduloinnin ja demoduloinnin.

9.1 USRP Emolevy

USRP:n FPGA hoitaa digitaaliset ylös(DUC) ja alas muunnokset (DDC), decimoinnin ja interpoloinnit, toisin sanoen USRP-alustan emolevy on osa RF-etupäätä.
USRP:n emolevyssä (kuva 7) on 4 AD-muunninta (64 MHz 12-bit) ja 4 DA-muunninta (128 MHz 14bit). AD/DA muuntimet on yhdistetty Altere Cyclone FPGA-prosessoriin. FPGA prosessori on yhteydessä USB2 liitäntäpiiriin Cypress FX2. Mukana tuleva virtalähde tuottaa 6V 4A DC:tä. Alusta itse tarvitsee 5V 1,5A, mutta useimmat tytärkortit vaativat 6V ja 1A virtaa. Emolevyltä löytyy neljä slot-liitintä tytärkorteille.
Suurimpiin rajoituksia USRP-kehitysalustalla on USB2, joka rajoittaa PC:ltä USRP:lle siirrettävän signaalinkaistanleveyden 8 MHz:n. Emolevy itsessään kuitenkin voi käsitellä 16 MHz leveää signaalia. /2;3/



***Kuva 8**. USRP:n emolevy./2/*

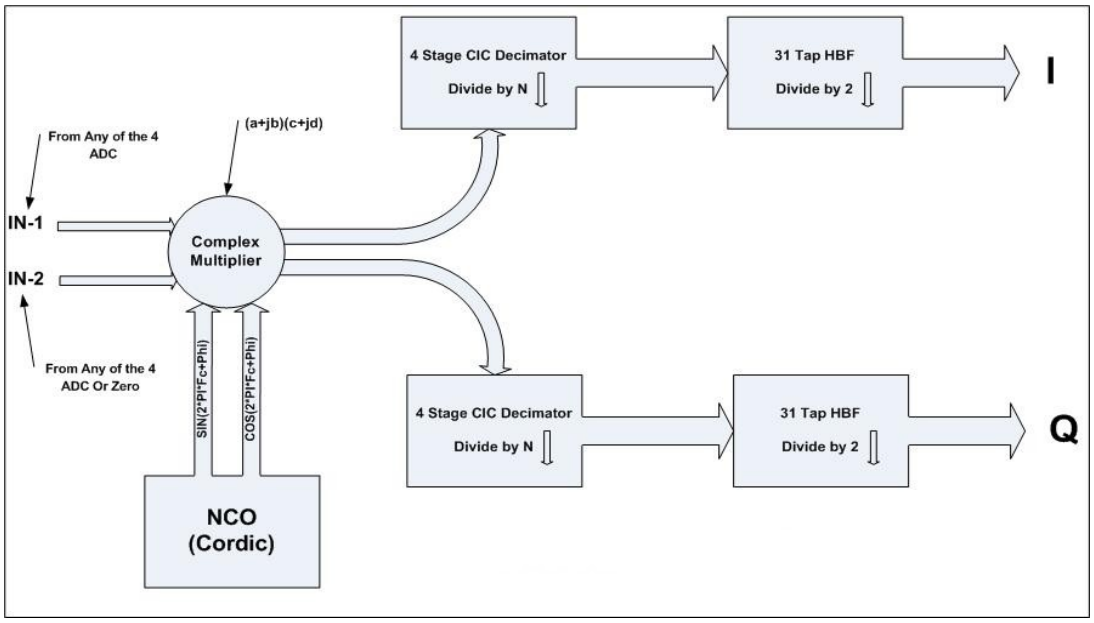


***Kuva 9**.USRP:n yksinkertainen lohkokaavio/2, s.8/*

Yllä olevasta lohkokaaviosta (kuva 9) nähdään, että FPGA on yhdistetty suoraan USB2 ohjainpiiriin. USB-ohjainpiiristä lähtee I2C-väylä ja SPI-väylä. I2C-väylää mahdollistaa tytärkorttien EEPROMin ohjelmoimisen ja SPI väylän kautta siirtyy data USB-ohjainpiirille ja toisinpäin. FPGA:sta lähtee 16 bit:et I/O-väylät tytärkorteille. Lohkokaaviosta selviää myös, että emolevyltä löytyy paikat neljälle tytärkotille, kahdelle vastaanottavalle ja kaksi lähettävälle. /2/

9.1.1 FPGA-piiri

Emolevystä löytyy Alteran Cyclonen perheen EP1C12 FPGA prosessori.
FPGA sisältää digitaaliset alas- ja ylös muuntimet (DDC ja DUC), jotka on varustettu 4 vaiheisilla CIC-suotimilla (Cascade Integrator-comb). CIC-suotimet ovat nopeita, koska niissä käytetään vain viivettä ja summausta. CIC-suotimen kanssa sarjassa on myös 31 tapin puolenkaistan suodatin spektrin muokkaamista ja kaistan ulkopuolisen signaalin hylkäämiseen. FPGA:sta löytyy myös 4 vaiheiset digitaaliset alasmuuntimet ilman puolenkastan suodatinta, joka mahdollistaa 1,2, tai 4 erillistä vastaanotto kanavaa. Kyseistä ominaisuutta tarvitaan MIMO vastaanotossa. /2, s.8/



Kuva 10.*FPGA:n DDC /2, s.9/*

Digitaalinen alas muunnin toimii seuraavasti. Ensiksi muunnetaan kompleksinen signaali IF-taajuudelta kantataajuudelle. Tämän jälkeen signaali voidaan desimoida, jotta se voidaan sovittaa USB2-väylän nopeuteen. Kun signaali on kantataajuudelle, se desimoidaan tekijällä n (n= 8–256 välillä). Desimointi toteutetaan neljässä vaiheessa. Desimointia voidaan ajatella alipäästösuodattimena, jonka jälkeen seuraa näytteiden vähentäjä (eng. down sampler).
Kaistanleveyden (32 MHz) suhteen voimme siirtää 32 MB/s USB väylällä. Näytteet lähetetään USB2-väylää pitkin 16-bit etumerkeillä varustettuna kokonaislukuina IQ formaatissa. Tämä merkitsee 4 tavua (16 bit/ 4 = 4 tavua) jokaista kompleksista näytettä kohden. Tästä saadaan, että yksi kompleksinen näyte tarvitsee 8 MB/s tiedonsiirto kapasiteettia./2, s.9/

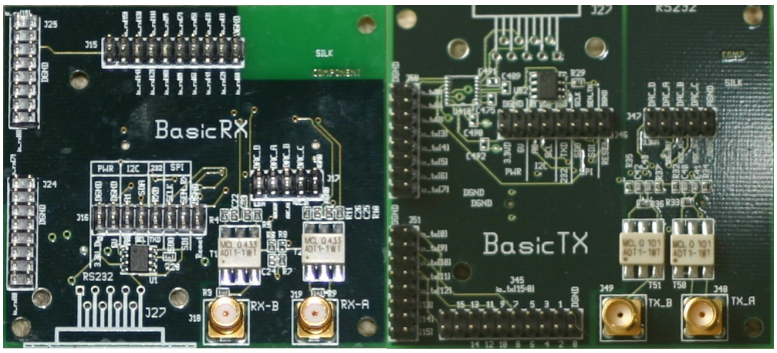
9.2 Tytärkortit

Jokaisella tytärkortilla on EEPROM, joka mahdollistaa tytärkortin automaattisen tunnistamisen. EEPROM:iin on esimerkiksi tallennettu kalibrointi arvot. Tytärkorttien paikat emolevyllä on nimetty TXA, TXB, RXA ja RXB. Jokaisesta tytärkortin paikasta on pääsy kahteen neljästä AD/ DA-muuntimesta. Tytärkortit ovat MIMO yhteensopivia TVRX-korttia lukuunottamatta. /2;3/

9.2.1 Basic-sarja, 1 - 250 MHz lähetin/ vastaanotin

Basic-sarjan tytärkortit (kuva 11) on suunniteltu käytettäväksi ulkoisen RF etupään kanssa. AD/DA muuntimet on kytketty suoraan SMA liittimiin, joten välistä ei löydy

sekoittimia, suodattimia tai vahvistimia. Tytärkortit tarjoavat suoran pääsyn rajapinta signaaleihin. Basic sarjassa on oma kortti vastaanottoon ja lähetykseen. /2/



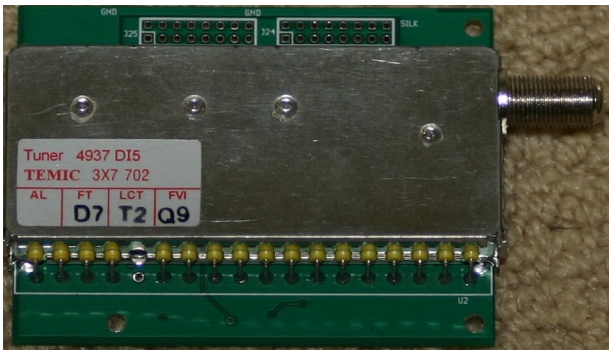
***Kuva 11**.BasicRX ja BasicTX tytärkortit /3/*

9.2.2 LF-sarja, DC -30 MHz lähetin/ vastaanotin

LFTX ja LFRX korteilla on kaksi merkittävää eroa Basic-sarjan kortteihin. LFTX ja LFRX käyttävät differentiaali vahvistinta muuntimen sijaa, minkä takia taajuusalue ulottuu tasavirtaan. Korteista löytyy myös 30 MHz:n alipäästösuodin laskostumisen estämiseksi./2;3/

9.2.3 TVRX, 50 - 860 MHz vastaanotin

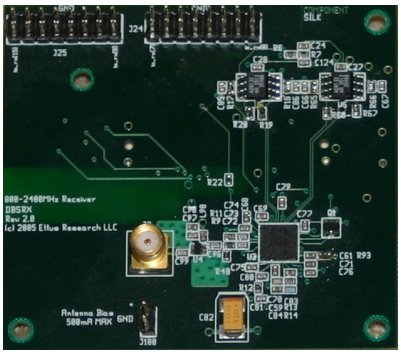
TVRX tytärkortti (kuva 12) on VHF ja UHF vastaanotin, joka perustuu TV-viritin moduuliin. Ainoa tytärkortti, jonka lähdön impedanssi on 75Ω .Vastaanotettava kaistan leveys on 6 Mhz. /2;3/



***Kuva 12.** TVRX tytärkortti./3/*

9.2.4 DBSRX, 0.8 - 2.4 GHz vastaanotin

DBSRX vastaanotin (kuva 13) kortti 800 MHz:stä 2,4 GHz:iin 3 - 5 dB kohinaluvulla. Kortista löytyy ohjelmallisesti säädettävä 1 - 60 MHz kanava suodatin. /2;3/



Kuva 13. *DBSRX tytärkortti./3/*

9.2.5 WBX0510, 50 Mhz -1Ghz lähetin/ vastaanotin

WBX0510 kortti sisältää lähettimen ja vastaanottimen. Kortilla on mahdollista lähettää/vastaanottaa noin 30 MHz levyistä signaalia. Kaikkia toimintoja voidaan hallita ohjelmallisesti tai FPGA:lla. Korteista löytyy myös analoginen RSSI mittaus ja 16 digitaalista I/O väylää esimerkiksi antenni kytkimen ohjaukseen. Lähetysteho on 100 mW (20 dBm). /2;3/

9.2.6 RFX-sarja, lähetin/ vastaanotin

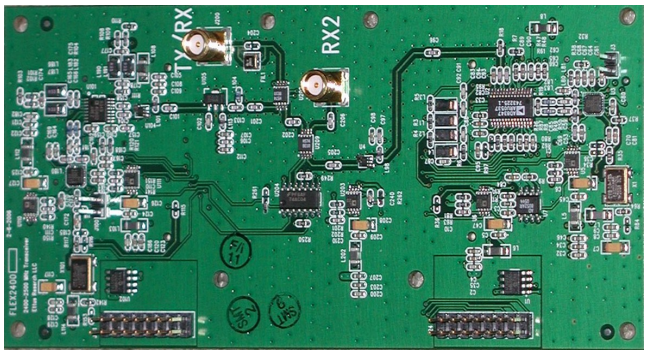
RFX-sarjan kortit sisältävät lähettimen ja vastaanottimen. Korteilla on mahdollista lähettää/vastaanottaa noin 30 MHz levyistä signaalia. Kaikkia toimintoja voidaan hallita ohjelmallisesti tai FPGA:lla. Korteista löytyy myös analoginen RSSI mittaus ja 16 digitaalista I/O väylää esimerkiksi antenni kytkimen ohjaukseen.

RFX900, 750 - 1050 MHz. Kortissa on kaistanpäästö suodatin 902 - 928 MHz (ISM-kaista). Lähetysteho 200 mW (23 dBm). Suodatin voidaan ohittaa juottamalle rinnalle 100pf:n kondensaattori ja katkaisemalla johdin suotimelta. Kun suodatin on ohitettu voidaan kortti muuttaa RFX1800:ksi polttamalla EEPROMille 1800 firmware.

RFX1200, 1150 - 1450 MHz
Lähetysteho 200 mW (23 dBm).

RFX1800, 1.5 - 2.1 GHz
Lähetysteho 200 mW (20 dBm).

RFX2400, 2.3 - 2.9 GHz (kuva 14). Kortissa on kaistanpäästö suodatin 2400 - 2483 MHz (noin ISM-kaista). Lähetysteho 50 mW (17 dBm). Suodatin voidaan ohittaa juottamalla rinnalle 50pf kondensaattori. Kun suodatin on ohitettu voidaan kortti muuttaa RFX1200 polttamalla EEPROMille 1200 firmware. /2;3/

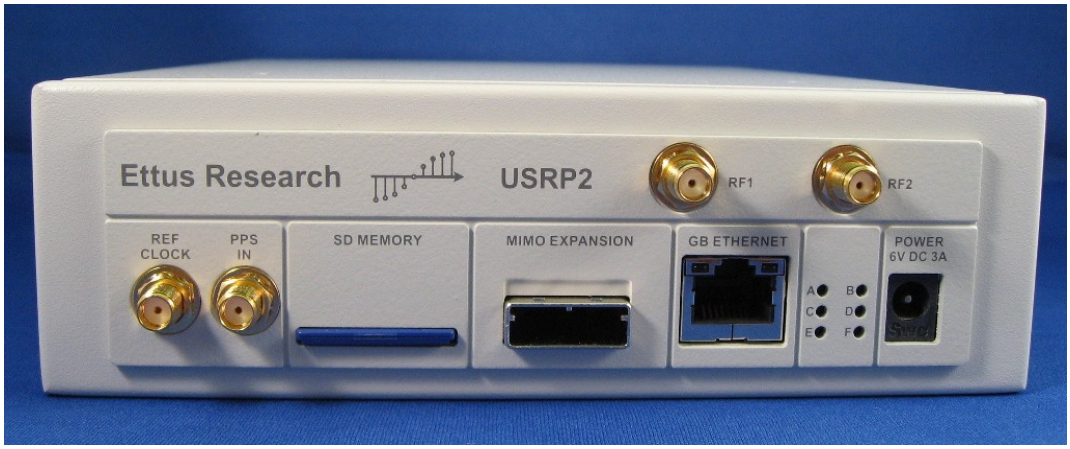


Kuva 14. *RFX2400 tytärkortti./3/*

9.2.1 XCVR2450, 2,4 - 2,5 GHz ja 4,9 - 5,85 GHz lähetin/ vastaanotin

XCVR2450 kortti sisältää lähettimen ja vastaanottimen. Kortilla on mahdollista lähettää/vastaanottaa noin 30 MHz levyistä signaalia. Kaikkia toimintoja voidaan hallita ohjelmallisesti tai FPGA:lla. Korteista löytyy myös analoginen RSSI mittaus ja 16 digitaalista I/O väylää esimerkiksi antenni kytkimen ohjaukseen. Lähetysteho on 100 mW (20 dBm). /2;3/

9.3 USRP2 emolevy



Kuva 15*.USRP2-kehitysalusta [22]*

USRP2 (kuva 15) emolevyssä kaksi AD-muunninta (100 MHz 14-bit) ja kaksi DA-muunninta (400 MHz 16bit). USB2 liitäntä on vaihtunut 1 Gb:n ethernet-liitäntään. Tämän

takia siirrettävän signaalinleveys voi olla 25 MHz, joka kuuluu tärkeimpiin uudistuksiin verrattaessa USRP:hen. Toki FPGA-piiri on myös vaihtunut nopeampaan Xilinx Spartan 3-2000 ja on saanut nopeata SRAM:a 1 Mb. Uusi emolevy tukee samoja tytärkortteja, mutta kortti paikat on vähennetty yhteen RX ja TX paikkaan. Uutena ominaissutena on SD-muistikortti paikka, jotta USRP2:ta voidaan käyttää ilman tietokonetta. SD-muistikortti tarvitaan, jotta USRP2 voidaan käyttää. Korttiin on tallennettu FPGA:an asetukset ja mikroprosessorin firmware. Levyssä on myös nopea sarjalinkki 1,6 Gb suuntaansa. USRP2 hinta on 1400$. /21/

10 GNU RADION KÄYTTÖ USRP-KEHITYSALUSTAN KANSSA

Käyttämäni USRP-kehitysalustan versio oli 4.5. USRP-kehityalustaa käytetään GNU Radio ohjelmistolla. Gnu Radio ohjelmisto koostuu työkaluista, joita ovat mm signaalinkäsittely lohkot. Signaalikäsittely lohkot ovat koodattu C++ ja ne on jaettu neljään pääryhmään. Suodattimiin (filters), signaalikohteisiin (sinks), signaalilähteisiin (sources) ja tyyppimuunnoksiin (type conversions). Lohkot liitetään toisiinsa Python-koodilla (flow graph). Python kielellä voidaan myös luoda graafinen käyttöympäristö. Jos sopivaa valmista signaalinkäsittely lohkoa ei löydy, sen voi koodata itse. Yleisimmät välitettävät datatyypit ovat kompleksiluvut, reaaliset lyhyet tai pitkät kokonaisluvut ja liukuluvut. Käyttöjärjestelmänä käytin Linuxia. Linux jakeluista valitsin Ubuntu 8.04, koska se on varsin suosittu ja ongelmatapauksiin löytyy luultavasti aika helposti ratkaisu. Jakelu oli juuri julkaistu keväällä (2008) ja osoittautui aluksi vähän epävakaaksi, mutta syksyyn mennessä alkoi toimia moitteettomasti. Gnu Radiosta asensin kehitysversion, jonka mukana tuli GRC. GRC on graafinen käyttöliittymä Gnu Radiolle. Gnu Radion käänsin lähdekoodista ja kääntämisohjeet löytyi Ubuntulle Gnu Radion Wikistä. Kääntäminen sujui ilman suurempia ongelmia. Pieniä ongelmia tuli vastaan, mutta ongelmiin löytyi aika nopeasti ratkaisut. /2/

10.1 FM-vastaanotin graafisella käyttöliittymällä

Käyn FM-vastaanottimen koodia läpi, joka tulee Gnu Radio ohjelmiston mukana (usrp_wfm_rcv.py) LIITE 1. Olen muokannut alkuperäistä koodia poistamalla turhia rivejä, joita ei tarvita BasicRX kortin kanssa. Kun koodia katsellaan ensimmäinen rivi on seuraavanlainen.

#!/usr/bin/env python
Yläpuolella oleva ensimmäinen rivi kertoo komentotulkille, että kyseessä on Python tiedosto. Tämä rivi tarvitaan, jotta koodi voidaan suorittaa komentokehotteesta.

from gnuradio import gr, gru, eng_notation, optfir
Gr on Gnu Radion pääkirjasto moduuli*, gru* moduuli tuo sekalaisia työkaluja, matikkaa ja muuta, *eng_notation* tuo tuen insinööri formaatin numeroille*, optifir* tuo ohjelman FIR suotimen suunnitteluun.

from gnuradio import audio
Äänikortin hallinta. (sources ja sinks)

from gnuradio import usrp
Sources (lähteet), sinks (kohteet) ja controls (hallinnan)

from gnuradio import blks2
Täydentäviä lohkoja, jotka on kirjoitettu pythonilla. Sisältää usein käytettyjä lohkoja, kuten modulaattori ja demodulaattori

from gnuradio.eng_option import eng_option
Laajentaa *optparsen* ymmärtämään insinööri merkintöjä (SI-järjestelmä).

from gnuradio.wxgui import slider, stdgui2, fftsink2, form
Välineet graafisen käyttöliittymän luomiseen

from optparse import OptionParser
Mahdollistaa komentokehote vaihtoehdot

from usrpm import usrp_dbid
Mahdollistaa symboliset nimet tytärkorteille

import sys
import math
import wx
Yllä olevat käskyillä otetaan moduuleita käyttöön (**from** .... **import** MODUULI). Moduulin tuonti komennon alle olen selventänyt mitä ominaisuuksia ne tuo. Komennot vastaa C++ #include komentoa.

def pick_subdevice(u):
return usrp.pick_subdev(u, (usrp_dbid.BASIC_RX,))
Ylhäällä olevilla riveillä valitaan käytettäväksi kortiksi BasicRx Tytärkortti Emolevyn A –puolelta.

```
class wfm_rx_block (stdgui2.std_top_block):
   def __init__(self,frame,panel,vbox,argv):
      stdgui2.std_top_block.__init__ (self,frame,panel,vbox,argv)

      parser=OptionParser(option_class=eng_option)
      parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                        help="select USRP Rx side A or B (default=A)")
      parser.add_option("-f", "--freq", type="eng_float", default=100.1e6,
                        help="set frequency to FREQ", metavar="FREQ")
      parser.add_option("-g", "--gain", type="eng_float", default=40,
```

```
                help="set gain in dB (default is midpoint)")
        parser.add_option("-V", "--volume", type="eng_float", default=None,
                help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string", default="",
                help="pcm device name.  E.g., hw:0,0 or surround51 or /dev/dsp")

        (options, args) = parser.parse_args()

if len(args) != 0:
        parser.print_help()
        sys.exit(1)

        self.frame = frame
        self.panel = panel

        self.vol = 0
        self.state = "FREQ"
        self.freq = 0
```

Ylhäällä olevalla riveillä määritetään, mitä ominaisuuksia voidaan määritellä ohjelmaa käynnistäessä komentokehotteesta.

Seuraava komento komentokehotteessa käynnistäisi fm-vastaanoton taajuudella 93.7 MHz:ä ja 20 dB:n vahvistuksella, joka on suurin mahdollinen.

./usrp_wfm_rcv.py -f 93.7M -g 20

Palataan takaisin koodin läpi käymiseen. s*elf.vol = 0* käskyllä kutsutaan metodi ja samalla määritellään arvoksi 0. Metodia kutsuttaessa tarvitaan aina *self*. *Self* on pythonissa kuin *this C++:ssa.*

```
self.u = usrp.source_c()                    # usrp is data source
```

Määrittää data lähteeksi valitun tytärkortin, joka on siis BasicRx.

```
adc_rate = self.u.adc_rate()                # 64 MS/s
```

Määrittää DAC:in nopeudeksi 64 MHz.

```
usrp_decim = 200
```

Asetetaan desimointi arvoksi 200.

```
self.u.set_decim_rate(usrp_decim)
```

Asetetaan desimointi arvoksi 200.

```
usrp_rate = adc_rate / usrp_decim           # 320 kS/s
```

Asetetaan vastaanotetun signaalinkaistanleveyden 320 kHz:ä

```
chanfilt_decim = 1
```

Asetetaan kana suotimen desimointi arvoksi 1

```
demod_rate = usrp_rate / chanfilt_decim
```

Asetetaan modulointi nopeudeksi 320 kHz

```
audio_decimation = 10
```

Asetetaan äänen desimointi arvoksi 10.

```
audio_rate = demod_rate / audio_decimation  # 32 kHz
```

Määritetään ääni signaalin näytetaajuudeksi 32 kHz.

```
if options.rx_subdev_spec is None:
    options.rx_subdev_spec = pick_subdevice(self.u)
```

Haetaan tytärkortin määrittelyt.

```
self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
```

Asetetaan sekoittimen arvo tytärkortin määritelmien mukaiseksi ja asetetaan tytärkortti.

```
print "Using RX d'board %s" % (self.subdev.side_and_name(),)
```

Tulostetaan komentokehotteeseen tyärkortin sijainti (side a tai side b) ja nimi.

```
chan_filt_coeffs = optfir.low_pass (1,          # gain
                                    usrp_rate,  # sampling rate
                                    80e3,       # passband cutoff
                                    115e3,      # stopband cutoff
                                    0.1,        # passband ripple
                                    60)         # stopband attenuation
```

Määritetään kanava suotimeksi, FIR-tyyppinen alipäästö suodatin.

```
#print len(chan_filt_coeffs)
chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)
self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)
self.volume_control = gr.multiply_const_ff(self.vol)

# sound card as final sink
audio_sink = audio.sink (int (audio_rate),
                         options.audio_output,
                         False)  # ok_to_block
```

Määritetään audion kohteeksi äänikortti.

```
# now wire it all together
self.connect (self.u, chan_filt, self.guts, self.volume_control, audio_sink)

self._build_gui(vbox, usrp_rate, demod_rate, audio_rate)

if options.gain is None:
    # if no gain was specified, use the mid-point in dB
    g = self.subdev.gain_range()
    options.gain = float(g[0]+g[1])/2

if options.volume is None:
    g = self.volume_range()
    options.volume = float(g[0]+g[1])/2

if abs(options.freq) < 1e6:
    options.freq *= 1e6
```

yhdistetään lohkot toisiina *(self.connect(..))* ja tehdään graafisille lohkoille sama *(self._build_gui(...)).* Määritetään myös oletukset vahvistukselle, äänen voimakkuudelle ja halutulle kanavan taajudelle, jos ohjelmaa suoritettaessa ei ole annettu alkuarvo (./usrp_wfm_rcv.py).

```
# set initial values

self.set_gain(options.gain)
```

```
        self.set_vol(options.volume)
        if not(self.set_freq(options.freq)):
            self._set_status_msg("Failed to set initial frequency")


    def _set_status_msg(self, msg, which=0):
        self.frame.GetStatusBar().SetStatusText(msg, which)


    def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

        def _form_set_freq(kv):
            return self.set_freq(kv['freq'])


        if 1:
            self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from USRP",
                                               fft_size=512, sample_rate=usrp_rate,
                                              ref_scale=32768.0, ref_level=-40, y_divs=5)
            self.connect (self.u, self.src_fft)
            vbox.Add (self.src_fft.win, 4, wx.EXPAND)

        if 1:
            post_filt_fft = fftsink2.fft_sink_f(self.panel, title="Post Demod",
                                                fft_size=1024, sample_rate=usrp_rate,
                                                y_per_div=10, ref_level=0)
            self.connect (self.guts.fm_demod, post_filt_fft)
            vbox.Add (post_filt_fft.win, 4, wx.EXPAND)

        if 0:
            post_deemph_fft = fftsink2.fft_sink_f(self.panel, title="Post Deemph",
                                                  fft_size=512, sample_rate=audio_rate,
                                                  y_per_div=10, ref_level=-20)
            self.connect (self.guts.deemph, post_deemph_fft)
            vbox.Add (post_deemph_fft.win, 4, wx.EXPAND)
```

Asetetaan graafisien näyttöjen oletus arvot.

```
        # control area form at bottom
        self.myform = myform = form.form()

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)
        myform['freq'] = form.float_field(
            parent=self.panel, sizer=hbox, label="Freq", weight=1,
            callback=myform.check_input_and_call(_form_set_freq, self._set_status_msg))

        hbox.Add((5,0), 0)
        myform['freq_slider'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox, weight=3,
                                        range=(87.9e6, 108.1e6, 0.1e6),
                                        callback=self.set_freq)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)

        myform['volume'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Volume",
                                        weight=3, range=self.volume_range(),
```

```
                                callback=self.set_vol)
        hbox.Add((5,0), 1)

        myform['gain'] = \
           form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Gain",
                                weight=3, range=self.subdev.gain_range(),
                                callback=self.set_gain)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)


    def on_rotate (self, event):
        self.rot += event.delta
        if (self.state == "FREQ"):
            if self.rot >= 3:
                self.set_freq(self.freq + .1e6)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_freq(self.freq - .1e6)
                self.rot += 3
        else:
            step = self.volume_range()[2]
            if self.rot >= 3:
                self.set_vol(self.vol + step)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_vol(self.vol - step)
                self.rot += 3

    def on_button (self, event):
        if event.value == 0:       # button up
            return
        self.rot = 0
        if self.state == "FREQ":
            self.state = "VOL"
        else:
            self.state = "FREQ"
        self.update_status_bar ()


    def set_vol (self, vol):
        g = self.volume_range()
        self.vol = max(g[0], min(g[1], vol))
        self.volume_control.set_k(10**(self.vol/10))
        self.myform['volume'].set_value(self.vol)
        self.update_status_bar ()
```

Määritetään Graafisen ikkuna koko, liukujen paikat ja säätöjen rajaarvot (taajuus, äänenvoimakkuus ja RF-signaalin vahvistus).

```
def set_freq(self, target_freq):
    """
    Set the center frequency we're interested in.

    @param target_freq: frequency in Hz
    @rypte: bool

    Tuning is a two step process.  First we ask the front-end to
    tune as close to the desired frequency as it can.  Then we use
    the result of that operation and our target_frequency to
```

```
        determine the value for the digital down converter.
        """
        r = usrp.tune(self.u, 0, self.subdev, target_freq)

        if r:
            self.freq = target_freq
            self.myform['freq'].set_value(target_freq)         # update displayed value
            self.myform['freq_slider'].set_value(target_freq)  # update displayed value
            self.update_status_bar()
            self._set_status_msg("OK", 0)
            return True

        self._set_status_msg("Failed", 0)
        return False

    def set_gain(self, gain):
        self.myform['gain'].set_value(gain)     # update displayed value
        self.subdev.set_gain(gain)

    def update_status_bar (self):
        msg = "Volume:%r  Setting:%s" % (self.vol, self.state)
        self._set_status_msg(msg, 1)
        self.src_fft.set_baseband_freq(self.freq)

    def volume_range(self):
        return (-20.0, 0.0, 0.5)
```

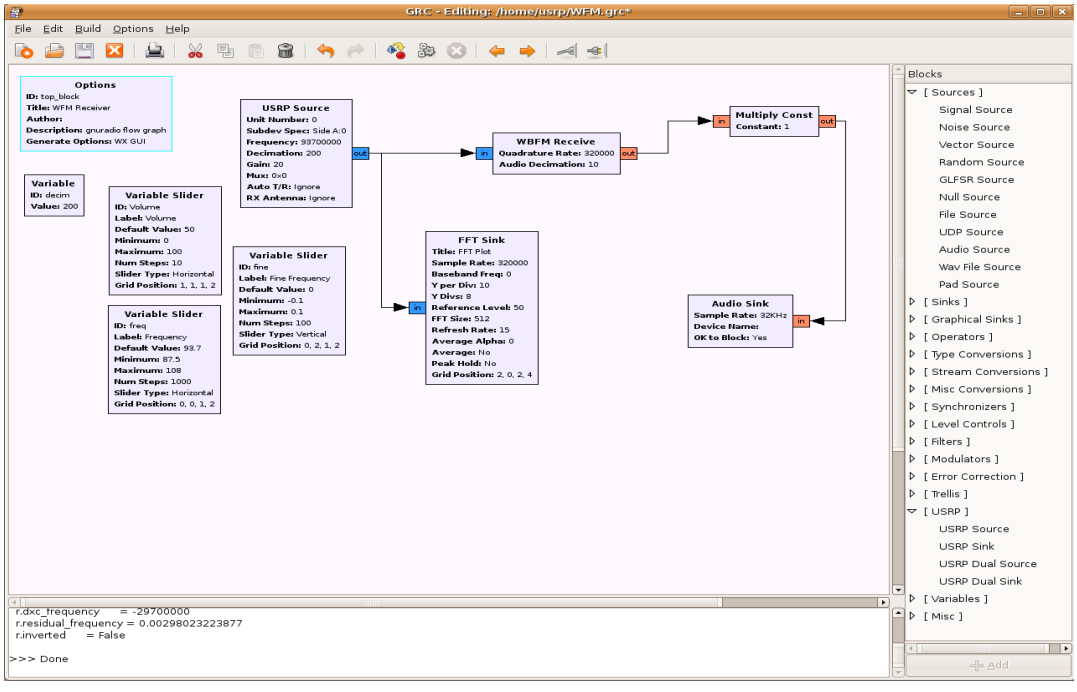
Tehdään viritys ja graafisessa käyttöliittymässä näkyvien arvojen päivitys.

```
if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP WFM RX")
    app.MainLoop ()
```

Viimeiset kaksi riviä käynnistävät ohjelman pyörimään silmukassa.

10.2 Graafinen työkalu

Gnu Radioon on kehitteillä graafinen työkalu (kuva 16) lohkojen määrittelemiseen ja



***Kuva 16**.GRC FM-vastaanotin BasicRX kortille*

yhdistämiseen, joka tunnetaan nimellä GNU Radio Companion (GRC) toistaiseksi. Toistaiseksi GRC:n kehittäjänä toiminut Josh Blum. Kyseinen mies on myös tehnyt uusia graafisia sink lohkoja Gnu Radioon OpenGL ohjelmointikielellä, korvaamaan ”hitaat” Pythonilla kirjoitetut graafiset lohkot.

11 OHJELMISTORADION TULEVAISUUS

Ohjelmistoradion kehitys näyttää tuovan mukanaan tehokkaamman tavan käyttää taajuuskaistaa (kognitiivinen radio), joka näyttää erittäin tarpeelliselta lisääntyvän taajuuskaistan tarpeen takia. Luultavasti saadaan toteutettua useampaa radiorajapintaa tukevia radioita halvemmalla kuin nykyään, kun siirrytään vähitellen ohjelmistolla kontrolloidusta radiosta ohjelmistolla määritettyyn radioon.
Haasteisiin kuuluu matkalla AD/ DA-muuntimien nopeuden ja tarkkuuden nostaminen, nopeampien prosessorien kehittäminen ja samalla näiden hyötysuhteen nostaminen. Hyötysuhteen nostemisen tekee ongelmalliseksi se, että kellotaajuuden nostaminen yleensä lisää tehon kulutusta.
Ensimmäiset ohjelmistoradiot ovat tukiasemia, koska virran kulutus niissä ei ole yhtä kriittistä, kuin kannettavissa laitteissa.

12 YHTEENVETO

Työssä käytiin läpi ohjelmistoradion määritteleminen ja tärkeimmät asiat, jotka vaikuttavat ohjelmistoradion toteuttamiseen. USRP ja Gnu Radio ohjelmiston muodostama ohjelmistoradion kehitysalusta ei ole välttämättä paras, eikä halvin mikä markkinoilta löytyy, mutta käytettävä taajuusalue on laaja. Tällä hetkellä, kun mennään kohti yhä leveämpi kaistaisia radiojärjestelmiä USRP:n kaistan leveys on vähän turhan kapea. USRP2 onneksi vähentää rajoituksen 25 MHz:in. Markkinoilta löytyy myös huomattavasti halvempia ohjelmistoradion kehitysalustoja, mutta ovat huomattavasti rajoittuneempia ominaisuuksiltaan.

LÄHDELUETTELO

SÄHKÖISET LÄHTEET

1. SDR Forum [www-sivu] Saatavissa: http://www.sdrforum.org/ [viitattu 28.11.2008]
2. USRP Documentation [PDF tiedosto] Saatavissa: http://gnuradio.org/trac/attachment/wiki/UsrpFAQ/USRP_Documentation.pdf [viitattu 28.11.2008]
3. Ettus Research LLC, USRP Ordering [www-sivu] Saatavissa: http://www.ettus.com/orderpage.html [viitattu 28.11.2008]
4. FlexRadio Systems [www-sivu] Saatavissa: http://www.flex-radio.com/Default.aspx [viitattu 28.11.2008]
5. SCA-dokumentit [www-sivu] Saatavissa: http://sca.jpeojtrs.mil/downloads.asp?ID=jtrs_infrastructure_documentation [viitattu 28.11.2008]
6. Wikipedia: SDR [www-sivu] Saatavissa: http://en.wikipedia.org/wiki/Software_radio [viitattu 28.11.2008]
7. Wikipedia: SCA [www-sivu] Saatavissa: http://en.wikipedia.org/wiki/Software_Communications_Architecture [viitattu 28.11.2008]
8. WCDMA Base Station Development Trend_Distributed Base Stations [PDF tiedosto] Saatavissa: http://www.huawei.com/file/download.do?f=784 [viitattu 28.11.2008]
9. Prosessori: Ohjelmistoradio tuo 4G verkot [PDF-tiedosto] Saatavissa: www.prosessori.fi/es05/ARKISTO/PDF/4G-VERKOT.pdf [viitattu 28.11.2008]
10. Prosessori: Työkaluja 4G-tutkimukseen [PDF-tiedosto] Saatavissa: http://www.prosessori.fi/es04/ARKISTO/PDF/OHJELMISTORADIOTEKNIIKKA.PDF [viitattu 28.11.2008]
11. Prosessori: Tukiasema taipuu ohjelmistolle [PDF-tiedosto] Saatavissa: http://www.prosessori.fi/arkisto/artikkelit/2007-1/PDF/OHJELMISTORADIO.pdf [viitattu 28.11.2008]
12. Prosessori: Ohjelmistoradio USB-liitäntään [PDF-tiedosto] Saatavissa: http://www.prosessori.fi/arkisto/artikkelit/2007-9/PDF/SOVELLUS.pdf [viitattu 28.11.2008]
13. Prosessori: Radiospektriä hyödynnetään heikosti [PDF-tiedosto] Saatavissa: http://www.prosessori.fi/arkisto/artikkelit/2006-11B/PDF/RADIOSPEKTRI.pdf [viitattu 28.11.2008]
14. Prosessori: Kasvavasti signaaleja digitoitavaksi Muunnin perustana[PDF-tiedosto] Saatavissa: http://www.prosessori.fi/arkisto/artikkelit/2008-5/PDF/MUUNNIN.pdf [viitattu 28.11.2008]
15. Puolustusvoimat: FSRP [PDF-tiedosto] Saatavissa: http://www.mil.fi/laitokset/pvtt/fsrpbook.pdf [viitattu 28.11.2008]
16. Puolustusvoimat: Sähkömagnetiikka 07. Seminaarijulkaisu [PDF-tiedosto]

Saatavissa: http://www.mil.fi/laitokset/pvtt/sahkomagn07.pdf [viitattu 28.11.2008]

17. SDR-Forum: Smart Antenna API Specification [PDF tiedosto] Saatavissa: http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-07-S-0016-V1_0_0.pdf [viitattu 28.11.2008]
18. Green Hills Platform for Software Defined radio (SDR) [www-sivu] Saatavissa: http://www.ghs.com/products/SDR.html [viitattu 28.11.2008]
19. HPSDR [www-sivu] Saatavissa: http://hpsdr.org/index.html [viitattu 28.11.2008]
20. Smart Antenna Systems [www-sivu] Saatavissa: http://www.iec.org/online/tutorials/smart_ant/index.asp [viitattu 28.11.2008]
21. USRP2 System Datasheet [PDF tiedosto] Saatavissa: http://www.ettus.com/downloads/ettus_ds_usrp2_v2.pdf [viitattu 28.11.2008]
22. The USRP2 [www-sivu] Saatavissa: http://gnuradio.org/trac/wiki/USRP2 [viitattu 28.11.2008]

LIITTEET

Fm Radio-vastaanottimen usrp_wfm_rcv.py koodi

```
#!/usr/bin/env python

from gnuradio import gr, gru, eng_notation, optfir
from gnuradio import audio
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
import math
import wx

def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    Try for one of these, in order: TV_RX, BASIC_RX, whatever is on side A.

    @return a subdev_spec
    """
    return usrp.pick_subdev(u, (usrp_dbid.BASIC_RX,))


class wfm_rx_block (stdgui2.std_top_block):
    def __init__(self,frame,panel,vbox,argv):
        stdgui2.std_top_block.__init__ (self,frame,panel,vbox,argv)

        parser=OptionParser(option_class=eng_option)
        parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=None,
                          help="select USRP Rx side A or B (default=A)")
        parser.add_option("-f", "--freq", type="eng_float", default=100.1e6,
                          help="set frequency to FREQ", metavar="FREQ")
        parser.add_option("-g", "--gain", type="eng_float", default=40,
                          help="set gain in dB (default is midpoint)")
        parser.add_option("-V", "--volume", type="eng_float", default=None,
                          help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string", default="",
                          help="pcm device name.  E.g., hw:0,0 or surround51 or /dev/dsp")

        (options, args) = parser.parse_args()
        if len(args) != 0:
            parser.print_help()
            sys.exit(1)

        self.frame = frame
        self.panel = panel

        self.vol = 0
        self.state = "FREQ"
        self.freq = 0

        # build flow graph

        self.u = usrp.source_c()                # usrp is data source
```

```
        adc_rate = self.u.adc_rate()            # 64 MS/s
        usrp_decim = 200
        self.u.set_decim_rate(usrp_decim)
        usrp_rate = adc_rate / usrp_decim       # 320 kS/s
        chanfilt_decim = 1
        demod_rate = usrp_rate / chanfilt_decim
        audio_decimation = 10
        audio_rate = demod_rate / audio_decimation  # 32 kHz

        if options.rx_subdev_spec is None:
            options.rx_subdev_spec = pick_subdevice(self.u)

        self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
        self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
        print "Using RX d'board %s" % (self.subdev.side_and_name(),)


        chan_filt_coeffs = optfir.low_pass (1,          # gain
                                    usrp_rate,   # sampling rate
                                    80e3,        # passband cutoff
                                    115e3,       # stopband cutoff
                                    0.1,         # passband ripple
                                    60)          # stopband attenuation
        #print len(chan_filt_coeffs)
        chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)

        self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)

        self.volume_control = gr.multiply_const_ff(self.vol)

        # sound card as final sink
        audio_sink = audio.sink (int (audio_rate),
                            options.audio_output,
                            False)  # ok_to_block

        # now wire it all together
        self.connect (self.u, chan_filt, self.guts, self.volume_control, audio_sink)

        self._build_gui(vbox, usrp_rate, demod_rate, audio_rate)

        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.subdev.gain_range()
            options.gain = float(g[0]+g[1])/2

        if options.volume is None:
            g = self.volume_range()
            options.volume = float(g[0]+g[1])/2

        if abs(options.freq) < 1e6:
            options.freq *= 1e6

        # set initial values

        self.set_gain(options.gain)
        self.set_vol(options.volume)
        if not(self.set_freq(options.freq)):
            self._set_status_msg("Failed to set initial frequency")


    def _set_status_msg(self, msg, which=0):
        self.frame.GetStatusBar().SetStatusText(msg, which)
```

```
    def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

        def _form_set_freq(kv):
            return self.set_freq(kv['freq'])


        if 1:
            self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from USRP",
                                               fft_size=512, sample_rate=usrp_rate,

ref_scale=32768.0, ref_level=0, y_divs=12)
            self.connect (self.u, self.src_fft)
            vbox.Add (self.src_fft.win, 4, wx.EXPAND)

        if 1:
            post_filt_fft = fftsink2.fft_sink_f(self.panel, title="Post Demod",
                                                fft_size=1024, sample_rate=usrp_rate,
                                                y_per_div=10, ref_level=0)
            self.connect (self.guts.fm_demod, post_filt_fft)
            vbox.Add (post_filt_fft.win, 4, wx.EXPAND)

        if 0:
            post_deemph_fft = fftsink2.fft_sink_f(self.panel, title="Post Deemph",
                                                  fft_size=512, sample_rate=audio_rate,
                                                  y_per_div=10, ref_level=-20)
            self.connect (self.guts.deemph, post_deemph_fft)
            vbox.Add (post_deemph_fft.win, 4, wx.EXPAND)


        # control area form at bottom
        self.myform = myform = form.form()

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)
        myform['freq'] = form.float_field(
            parent=self.panel, sizer=hbox, label="Freq", weight=1,
            callback=myform.check_input_and_call(_form_set_freq, self._set_status_msg))

        hbox.Add((5,0), 0)
        myform['freq_slider'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox, weight=3,
                                        range=(87.9e6, 108.1e6, 0.1e6),
                                        callback=self.set_freq)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)

        myform['volume'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Volume",
                                        weight=3, range=self.volume_range(),
                                        callback=self.set_vol)
        hbox.Add((5,0), 1)

        myform['gain'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox, label="Gain",
                                        weight=3, range=self.subdev.gain_range(),
                                        callback=self.set_gain)
        hbox.Add((5,0), 0)
```

```
        vbox.Add(hbox, 0, wx.EXPAND)


    def on_rotate (self, event):
        self.rot += event.delta
        if (self.state == "FREQ"):
            if self.rot >= 3:
                self.set_freq(self.freq + .1e6)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_freq(self.freq - .1e6)
                self.rot += 3
        else:
            step = self.volume_range()[2]
            if self.rot >= 3:
                self.set_vol(self.vol + step)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_vol(self.vol - step)
                self.rot += 3

    def on_button (self, event):
        if event.value == 0:        # button up
            return
        self.rot = 0
        if self.state == "FREQ":
            self.state = "VOL"
        else:
            self.state = "FREQ"
        self.update_status_bar ()


    def set_vol (self, vol):
        g = self.volume_range()
        self.vol = max(g[0], min(g[1], vol))
        self.volume_control.set_k(10**(self.vol/10))
        self.myform['volume'].set_value(self.vol)
        self.update_status_bar ()

    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool

        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital down converter.
        """
        r = usrp.tune(self.u, 0, self.subdev, target_freq)

        if r:
            self.freq = target_freq
            self.myform['freq'].set_value(target_freq)         # update displayed value
            self.myform['freq_slider'].set_value(target_freq)  # update displayed value
            self.update_status_bar()
            self._set_status_msg("OK", 0)
            return True

        self._set_status_msg("Failed", 0)
```

```
            return False

        def set_gain(self, gain):
            self.myform['gain'].set_value(gain)     # update displayed value
            self.subdev.set_gain(gain)

        def update_status_bar (self):
            msg = "Volume:%r  Setting:%s" % (self.vol, self.state)
            self._set_status_msg(msg, 1)
            self.src_fft.set_baseband_freq(self.freq)

        def volume_range(self):
            return (-20.0, 0.0, 0.5)


    if __name__ == '__main__':
        app = stdgui2.stdapp (wfm_rx_block, "USRP WFM RX")
        app.MainLoop ()
```