OULUN AMMATTIKORKEAKOULU

Jussi Laitala

SIGNAALINKÄSITTELYLOHKOJEN OHJELMOINTI GNU RADIOSSA

SIGNAALINKÄSITTELYLOHKOJEN OHJELMOINTI GNU RADIOSSA

Jussi Laitala
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Jussi Laitala
Opinnäytetyön nimi: Signaalinkäsittelylohkojen ohjelmointi GNU Radiossa
Työn ohjaaja: Kari Jyrkkä
Työn valmistumislukukausi ja -vuosi: Kevät 2024 Sivumäärä: 43

Opinnäytetyön tavoitteena oli tutustua GNU Radio -ohjelmistoon sekä tutkia, millä tavalla uusia signaalinkäsittelylohkoja voidaan toteuttaa ja käyttää yhdessä ohjelmiston standardilohkojen kanssa. Lisäksi työssä toteutettiin omia lohkoja hyödyntävä signaalinkäsittelyketju GNU Radiolle, jolla pystyttiin vastaanottamaan FM-radiosignaalia ja demoduloimaan sitä äänisignaaliksi.

Työn alussa olevassa teoriaosassa selitetään, miten FM-modulointi toimii ja millaisilla signaalinkäsittelytekniikoilla demoduloiminen voidaan toteuttaa. Luvussa pureudutaan tarkemmin työn toteutuksessa käytettyyn kvadratuuriseen FM-demodulointitekniikkaan sekä kerrotaan, miten demoduloinnissa käytetty alipäästösuodatus toteutettiin.

Työssä käytiin läpi GNU Radion asennusprosessi ja siihen tarvittavien ohjelmistoriippuvuuksien asentaminen Windows- ja Linux-käyttöjärjestelmille. Uusien signaalinkäsittelylohkojen ohjelmointiin käytettiin C++-ohjelmointikieltä. Radiosignaalin vastaanottamiseen käytettiin USB-väylään liitettävää RTL-SDR-vastaanotinta.

Työn tuloksena saatiin toteutettua toimiva FM-demodulaattori GNU Radiolle, jonka suorituskyky oli riittävä radiolähetysten kuuntelemiseen.

Asiasanat: GNU Radio, demodulointi, signaalinkäsittely

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Jussi Laitala
Title of thesis: Programming signal processing blocks in GNU Radio
Supervisor: Kari Jyrkkä
Term and year when the thesis was submitted: Spring 2024
Number of pages: 43

The purpose of the thesis was to find out how new signal processing blocks can be programmed into the GNU Radio software and use them with the software's standard blocks. The work also implemented a signal processing chain for GNU Radio using own blocks, which could receive an FM radio signal and demodulate it back into an audio signal.

The theory part at the beginning of the work explains how FM modulation works and what signal processing techniques demodulation can be used for. In the chapter, the quadrature FM demodulation technique used in the implementation of the work is explained in more detail, and how the low-pass filtering used in the demodulation was implemented.

The work went through the installation process of GNU Radio and the installation of the necessary software dependencies for Windows and Linux operating systems. The C++ programming language was used to program the new signal processing blocks. An RTL-SDR receiver connected to the USB bus was used to receive the radio signal.

As a result of the work, a functioning FM demodulator was programmed into GNU Radio, the performance of which was sufficient for listening to radio broadcasts.

Keywords: GNU Radio, demodulation, signal processing

SISÄLLYS

1 JOHDANTO

GNU Radio on avoimeen lähdekoodiin perustuva ohjelmistokehitysalusta, joka tarjoaa erilaisia signaalinkäsittelylohkoja, joiden avulla voidaan rakentaa ohjelmistoradioita graafista käyttöliittymää hyödyntäen. Alustaa voidaan käyttää ulkoisesta vastaanottimesta saadun sähkömagneettisen signaalin käsittelyyn tai käyttää lähteenä generoitua signaalia simulaatioympäristössä. GNU Radio on ladattavissa ilmaiseksi ja se tukee Windows-, Linux- ja macOS-käyttöjärjestelmiä. (1.)

Tämän opinnäytetyön tavoitteena oli tutustua GNU Radion toimintaperiaatteeseen Windows- ja Linux-alustoilla, sekä käyttää ohjelmistoa RTL-SDR-vastaanottimen kanssa sähkömagneettisten signaalien vastaanottoon ja käsittelyyn. Kehitysalustan asennuksen mukana tulee valmiina erilaisia signaalinkäsittely- ja ohjelmistolohkoja, joista voidaan rakentaa signaalinkäsittelyketjuja ja kääntää niitä valmiiksi ohjelmistoradioksi. Kehitysalusta mahdollistaa myös uusien käsittelylohkojen ohjelmoinnin Python- ja C++-ohjelmointikielillä alustan tarjoamaa ohjelmistorajapintaa hyödyntäen. Työssä tutkittiin tapoja, joilla uusia signaalinkäsittelylohkoja voidaan ohjelmoida sekä lisätä graafisen käyttöliittymän saataville. Lopuksi omia ohjelmistolohkoja hyödyntäen toteutettiin käsittelyketju, jonka avulla vastaanotettu FM-radiolähetys prosessoitiin valmiiksi audioksi asti.

Sähkömagneettinen spektri on erittäin laaja ja signaalinkäsittely monimutkainen aihealue. Tämän vuoksi työ rajoittuu FM-radiolähetysten käyttämän signaalimuodon tutkimiseen ja analysointiin. Työn tilaajana toimi Oulun ammattikorkeakoulu ja aihe valittiin, koska signaalinkäsittely ja siihen liittyvä matematiikka kiinnostavat henkilökohtaisesti. Aiheen tutkimisesta ja perehtymisestä on myös hyötyä työssäni ohjelmistokehittäjänä HF-radiotekniikan parissa.

2 TAAJUUSMODULAATIO

2.1 Radioaallot ja modulointi

Guglielmo Marconi rakensi vuonna 1895 ensimmäisen tiedonsiirtolaitteen, jolla pystyttiin siirtämään signaalia langattomasti pitkien matkojen ylitse. Marconin laite käytti hyväkseen aiemmin löydettyä sähkömagneettista säteilyä, kun hän ymmärsi, että säteilyn avulla pystyttiin siirtämään tietoa ilman kiinteää yhteyttä lähettäjän ja vastaanottajan välillä. Laite pystyi muuntamaan elektronisen signaalin radioaalloiksi, jotka siirtyvät valonnopeudella ilmakehässä. Vastaanotin havaitsi nämä radioaallot ja muutti ne takaisin elektroniseksi signaaliksi. Marconin keksintö aloitti uuden aikakauden tiedonsiirron historiassa. (2.)

Modulaatiolla tarkoitetaan signaalin muokkaamista toisella signaalilla, jossa lähetettävä informaatio liitetään kantoaaltoon, jonka avulla tieto siirretään lähettäjältä vastaanottajille. Moduloitu signaali käännetään takaisin alkuperäiseksi signaaliksi demodulaation avulla. Signaalia voidaan moduloida useilla erilaisilla tavoilla ja näiden yhdistelmillä, mm. amplitudi-, taajuus- ja vaihemodulaatiomenetelmillä. Ensimmäisissä radiolähetyksissä käytettiin amplitudimodulointia, joka on helpompi tuottaa ja vastaanottaa, mutta on huomattavan herkkä häiriöille. Myöhemmin kehitetyssä taajuusmodulaatiossa, jota FM-radiot käyttävät, häiriönsietokyky ja äänenlaatu on huomattavasti parempi AM-lähetyksiin verrattuna. FM-radiot vaativat kuitenkin monimutkaisempaa tekniikkaa lähetykseen ja vastaanottoon. FM-radioiden signaali ei kanna yhtä laajalle kuin AM-radioissa, koska FM-lähetykset käyttävät tyypillisesti huomattavasti korkeampia taajuuksia kuin AM-lähetykset. (3.)

2.2 Taajuusmodulaatiotekniikat

Tässä työssä pureudutaan tarkemmin taajuusmoduloinnin teoriaan ja käytettyihin teknologisiin ratkaisuihin. Koska työn tavoitteena on ohjelmoida signaalinkäsittelylohkoja, joiden avulla vastaanotetuista signaalinäytteistä pystytään demoduloimaan FM-radiolähetys, tulee tarvittavien prosessointivaiheiden teoria tuntea, jotta prosessointivaiheet voidaan muuttaa ohjelmalliseen muotoon.

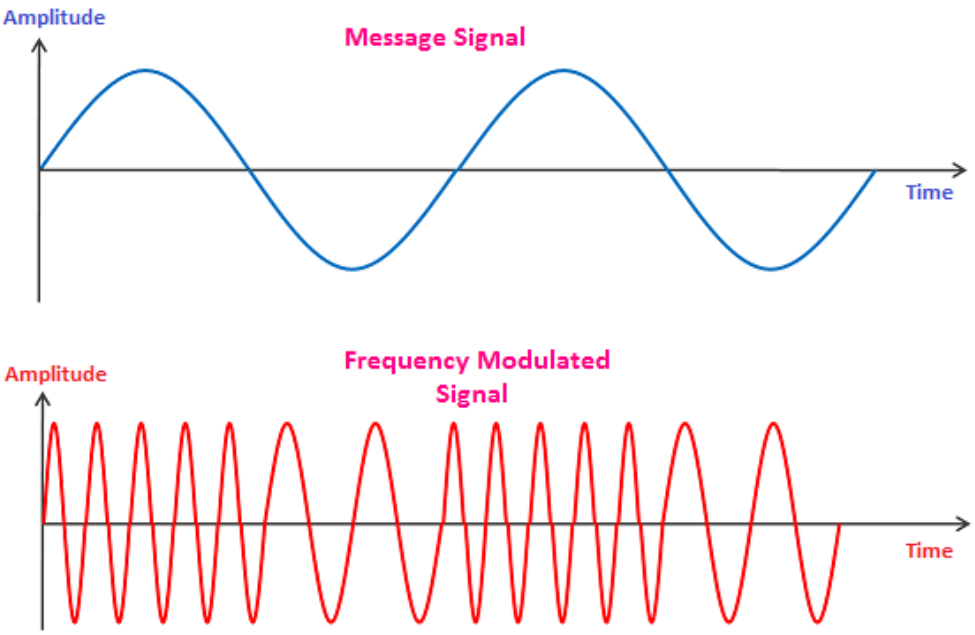
2.2.1 Signaalin modulointi ja lähettäminen radiosignaalina

Taajuusmoduloinnissa signaalin amplitudin vaihtelu moduloidaan kantoaaltoon, jonka taajuus vaihtelee signaalin voimakkuuden mukaan (kuva 1). Moduloidun signaalin voimakkuus pysyy kuitenkin vakiona. Kantoaalto on FM-lähetyksen keskitaajuus, jonka taajuus vaihtelee lähetyksen aikana moduloidun signaalin mukana. Taajuuspoikkeama ilmaisee, kuinka paljon kantoaallon taajuus voi vaihdella lähetyksen aikana ja on suoraan verrannollinen siihen, miten suuri amplitudin vaihtelu moduloitavassa signaalissa on. Taajuuspoikkeaman lisäksi, kun tiedetään signaalin maksimitaajuus, voidaan lähetyksen kaistanleveys laskea Carsonin säännön avulla (kaava 1). (5.)

$$FMkaistanleveys = 2(\Delta f + fm)$$

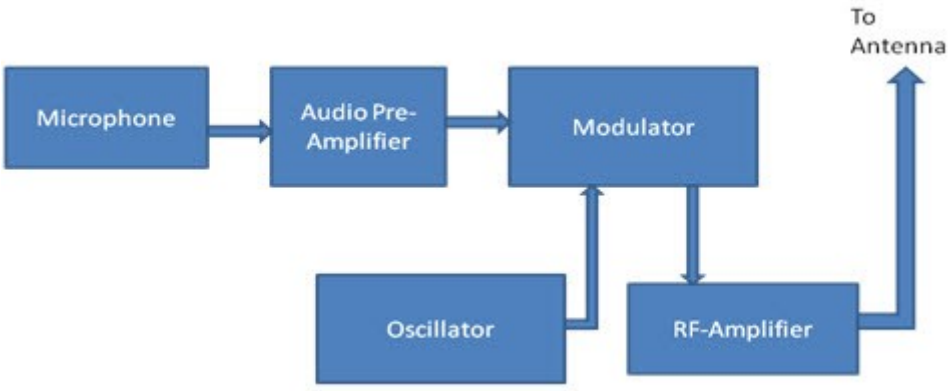
KAAVA 1. Carsonin kaistanleveyssääntö (5)

Yhtälössä ***Δf*** on taajuuspoikkeama ja ***fm*** tarkoittaa signaalin maksimitaajuutta. Esimerkiksi eurooppalaisessa standardissa FM-lähetykset sijoittuvat taajuusvälille 87,5–108 MHz, joka on jaettu 200 kHz:n levyisiin kanaviin. Lähetyksen taajuuspoikkeama on maksimissaan ±75 kHz:ä ja lähetettävän äänisignaalin maksimitaajuus on 15 kHz:ä. Tästä saamme laskettua FM-lähetyksen tarvitseman kaistanleveyden **2 * (75 kHz + 15 kHz) = 180 kHz**, johon 98 % lähetystehosta jakautuu. Näin ollen 200 kHz:n kanavat riittävät jakamaan lähetykset toisistaan. (6.)



KUVA 1. Signaalin taajuusmodulointi kantoaaltoon (4)

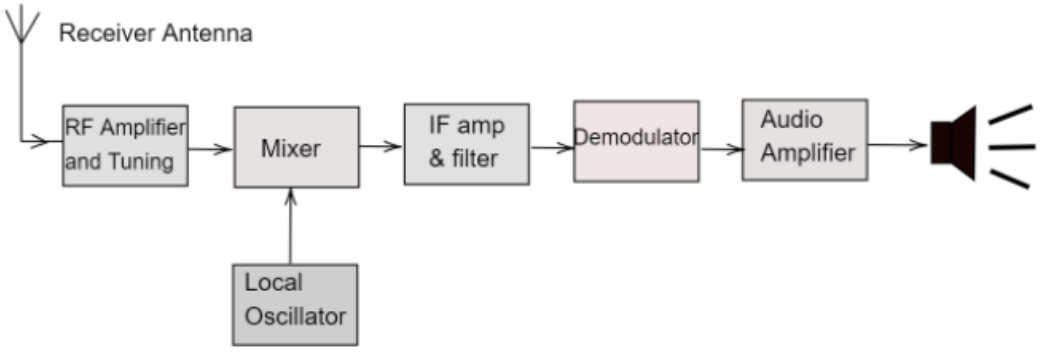
FM-lähettimessä taajuusmoduloitu signaali voimistetaan ja ohjataan antennille, joka lähettää signaalin radioaaltoina eteenpäin (kuva 2). Lähettämiseen tarvittava tekniikka on yksinkertaisempaa verrattuna vastaanottimeen, jossa signaali pitää seuloa kaiken muun radioliikenteen ja melun seasta demoduloitavaksi.



KUVA 2. FM-lähettimen lohkokaavio (7)

2.2.2 Radiosignaalin vastaanotto

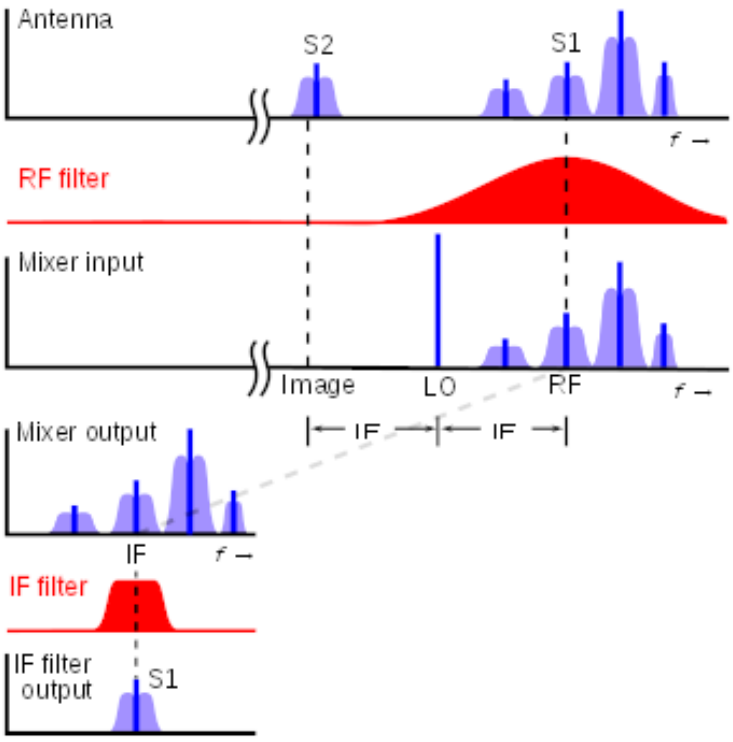
Taajuusmoduloidun signaalin vastaanotto on monimutkaisempi prosessi kuin lähettäminen. Vastaanottimessa oleva antenni muuttaa radioaallot elektroniseksi signaaliksi, josta eri vaiheiden kautta pystytään lähetetty FM-signaali demoduloimaan takaisin audioksi (kuva 3).



KUVA 3. FM-vastaanottimen lohkokaavio (8)

Vastaanottimessa ensimmäisessä vaiheessa radiosignaali ***RF*** vahvistetaan ja siitä suodatetaan pois korkeammat taajuuskomponentit halutun taajuusalueen ulkopuolelta. Tämän jälkeen signaali sekoitetaan oskillaattorin ***LO*** tuottaman vakiotaajuuden kanssa, jotta haluttu radiosignaalin saadaan siirrettyä alemmalle keskitaajuudelle ***IF*** demoduloitavaksi (kuva 4). Radiosignaalin

sekoittaminen oskillaattorin tuottaman signaali kanssa tuottaa signaalin keskitaajuudelle, jonka taajuus on oskillaattorin ja radiosignaalin taajuuksien erotus. Miksaus tuottaa samalla myös toisen signaalin, joka on radiosignaalin ja oskillaattorin signaalin taajuuksien summa. Tällä samalla periaatteella FM-lähettimessä siirretään signaali moduloinnin jälkeen korkeammalle taajuudelle lähetettäväksi. (9.)



KUVA 4. Signaalin suodattaminen ja siirtäminen alemmalle taajuudelle (9)

FM-vastaanottimessa muut taajuuskomponentit keskitaajuuden ulkopuolelta suodatetaan miksauksen jälkeen pois, jolloin jäljelle jää vain moduloitu signaali seuraavaan vaiheeseen demoduloitavaksi. Suodattaminen poistaa haitallista melua signaalin ympäriltä ja parantaa signaali-kohinasuhdetta. (10.)

2.2.3 Signaalin demodulointi

Signaalin demodulointivaiheessa voidaan käyttää erityyppisiä tekniikoita, joilla moduloitu signaali saadaan irroitettua takaisin alkuperäiseksi signaaliksi. Näillä jokaisella on omat etunsa ja haittansa. Ohessa muutamia demodulaatiotekniikoita, joita voidaan käyttää FM-vastaanottimissa. (11.)

Kaltevuuden tunnistus

Kaltevuuden tunnistus on hyvin yksinkertainen FM-demodulaatiomuoto, jota voidaan käyttää silloin, jos vastaanottimessa ei ole FM-toimintoa. Signaalin taajuusvaihtelut muutetaan amplitudin vaihteluiksi, josta alkuperäinen signaali saadaan demoduloitua AM-tekniikoita käyttäen. Tämä demodulaatiomuoto ei ole järin tehokas ja on herkkä signaalin amplitudin vaihteluille. (12.)

Kvadratuurinen FM-demodulaattori

Tässä demodulaatiomuodossa signaali jaetaan I/Q-komponentteihin**,** jotka kuvaavat signaalin reaali- ja imaginääriosia kompleksitasolla. Näiden signaalien vaihe-eron poikkeamasta saadaan kantoaallon taajuuspoikkeama, eli alkuperäinen signaali demoduloitua. Kvadratuurinen FM-ilmaisin on helppo toteuttaa ja tarjoaa hyvän suorituskyvyn. (13.)

Vaihelukitun silmukan demodulaattori

PLL-demodulointi (Phase Locked Loop) perustuu kvadratuurisen demodulaattorin tavoin kahden eri signaalin vaihe-erojen havaitsemiseen. Menetelmä vertaa jänniteohjatun oskillaattorin signaalia tulosignaalin ja tarjoaa takaisinkytkentämekanismin oskillaattorin signaalin taajuuden muuttamiseksi. Tämä demodulaatiotekniikka pääsee hyvin lineaariseen tulokseen ja pystyy toimimaan laajalla taajuusalueella. (14.)

2.2.4 Alkuperäisen signaalin toistaminen ja RDS

FM-vastaanottimen viimeisessä vaiheessa demoduloitu signaali vahvistetaan ja johdetaan kaiuttimille toistettavaksi ääniaaltoina. Demoduloitu signaali voi sisältää äänisignaalin sijaan myös digitaalista tietoa, kuten RDS (Radio Data System), jota FM-radiolähetyksen ohessa voidaan lähettää. RDS-järjestelmän avulla voidaan lähettää esimerkiksi radiokanavan nimi ja tietoa toistettavasta ohjelmasta tai kappaleesta. (15.)

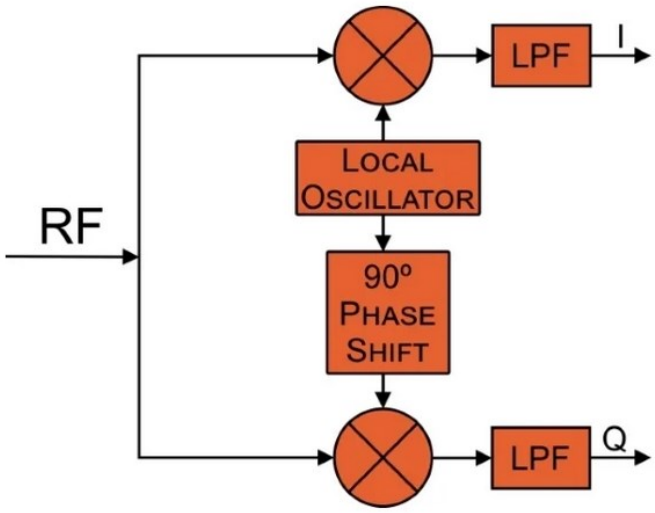
2.3 Työssä käytetyt signaalinkäsittelytekniikat

Tämän opinnäytetyön tavoitteena oli luoda GNU Radion signaalinkäsittelyketju, jolla pystytään demoduloimaan vastaanotettu FM-radiosignaali audioksi ja kuuntelemaan paikallisia radiokanavia. Päädyin toteuttamaan signaalinkäsittelyketjun lohkot, jotka vastaavat laiteajurilta saatujen signaalin IQ-näytteiden prosessoinnista audiosignaaliksi asti. Ketjun ensimmäisenä ja viimeisenä lohkona käytin GNU Radion standardilohkoja, jotka vastasivat signaalinäytteiden lukemisesta laiteajurilta ja audiosignaalin lähettämisestä äänikortille toistettavaksi. Koska työn tarkoituksena oli ymmärtää FM-signaalin demodulointiprosessia, päätin siksi toteuttaa vain siihen liittyvät prosessointilohkot työn rajaamiseksi. Seuraavaksi kerron hieman tarkemmin, mitä signaalinkäsittelytekniikoita demodulointiprosessissa käytin ja miksi päädyin juuri niihin.

2.3.1 Kvadratuurinen FM-demodulointi

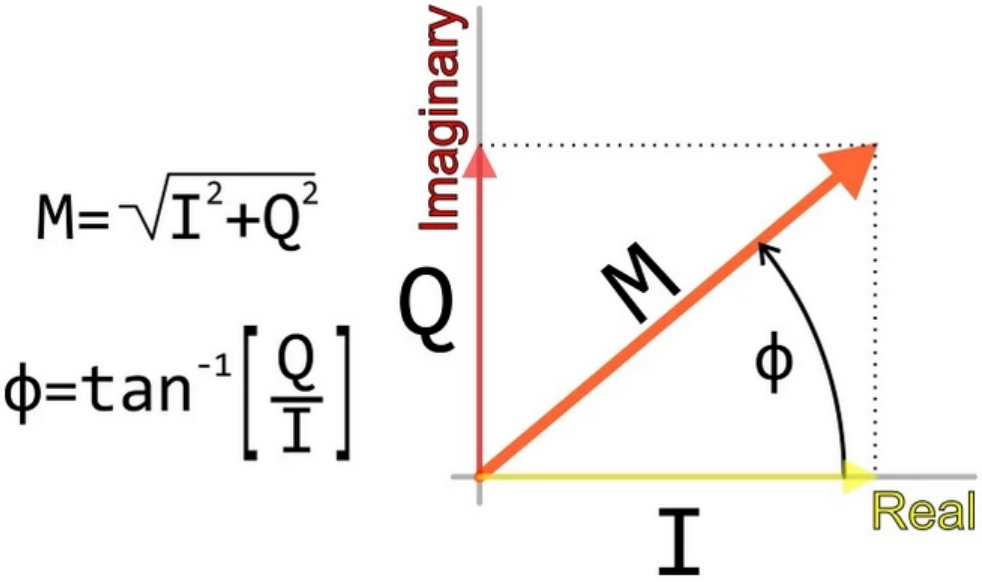
Työssä käytettävää demodulointitekniikkaa pohtiessani päädyin valitsemaan kvadratuurisen demodulointitekniikan signaalin prosessointiin. Tämä demodulointitekniikka on suhteellisen yksinkertaista toteuttaa ohjelmallisesti ja se on suorituskykyinen heikollakin signaalilla. Demodulointitekniikan toteuttamista helpotti myös se, että signaalinkäsittelyketjun ensimmäinen lohko, joka lukee RTL-SDR-vastaanottimen saamaa signaalia laiteajurilta, jakaa näytteet jo valmiiksi I- ja Q-komponentteihin ketjun seuraavalle lohkolle.

Kvadratuurisessa demoduloinnissa radiosignaalin jaetaan I- ja Q-komponentteihin (In phase ja Quadrature), joka tarkoittaa signaalin jakamista kahteen erilliseen signaaliin, joista toinen signaali on vaihesiirretty 90 astetta ensimmäiseen signaaliin nähden (kuva 5). Tämä saadaan kertomalla jaetut signaalit kantoaallon taajuisilla signaaleilla, joista ensimmäinen on kosini- ja toinen sinimuotoinen signaali, eli näiden signaalien vaihe-ero on 90 astetta. Signaalit suodatetaan tämän jälkeen alipäästösuodattimien avulla, jotta signaalien sekoittamisessa tuotetut korkeamman taajuuden signaalit saadaan vaimennettua ja kohinatasoja pienennettyä. (16.)



KUVA 5. Signaalin jakaminen I- ja Q-komponentteihin (16)

I- ja Q-komponentit eivät matemaattisesta näkökulmasta ole varsinaisia signaaleja, vaan edustavat alkuperäisen signaalin reaali- ja imaginääriosia kompleksitasolla (17). Näiden avulla voidaan muodostaa vektori, joka kuvaa alkuperäisen signaalin voimakkuutta ja vaihetta tietyllä ajan hetkellä (kuva 6). Vertaamalla IQ-signaalinäytteen vaihe-eroa aiempaan näytteeseen saadaan selville signaalin taajuuden muutos ja demoduloitua kantoaaltoon moduloitu signaali. Tämä tehdään kertomalla aiemman IQ-näytteen kompleksiluvun konjugaatio senhetkisen IQ-näytteen kompleksiluvun kanssa ja laskemalla arkustangentti tuloksena saadun kompleksiluvun imaginääri- ja reaaliosan suhteesta (kaava 2).



KUVA 6. I- ja Q-komponentit kuvattuna kompleksitasolla (16)

$$Y = (I - jQ)_{n-1} * (I + jQ)_n \Rightarrow S_{out} = arctan\left(\frac{\mathrm{Im}(Y)}{\mathrm{Re}(Y)}\right)$$

KAAVA 2. FM-signaalin demodulointi IQ-näytteistä (18)

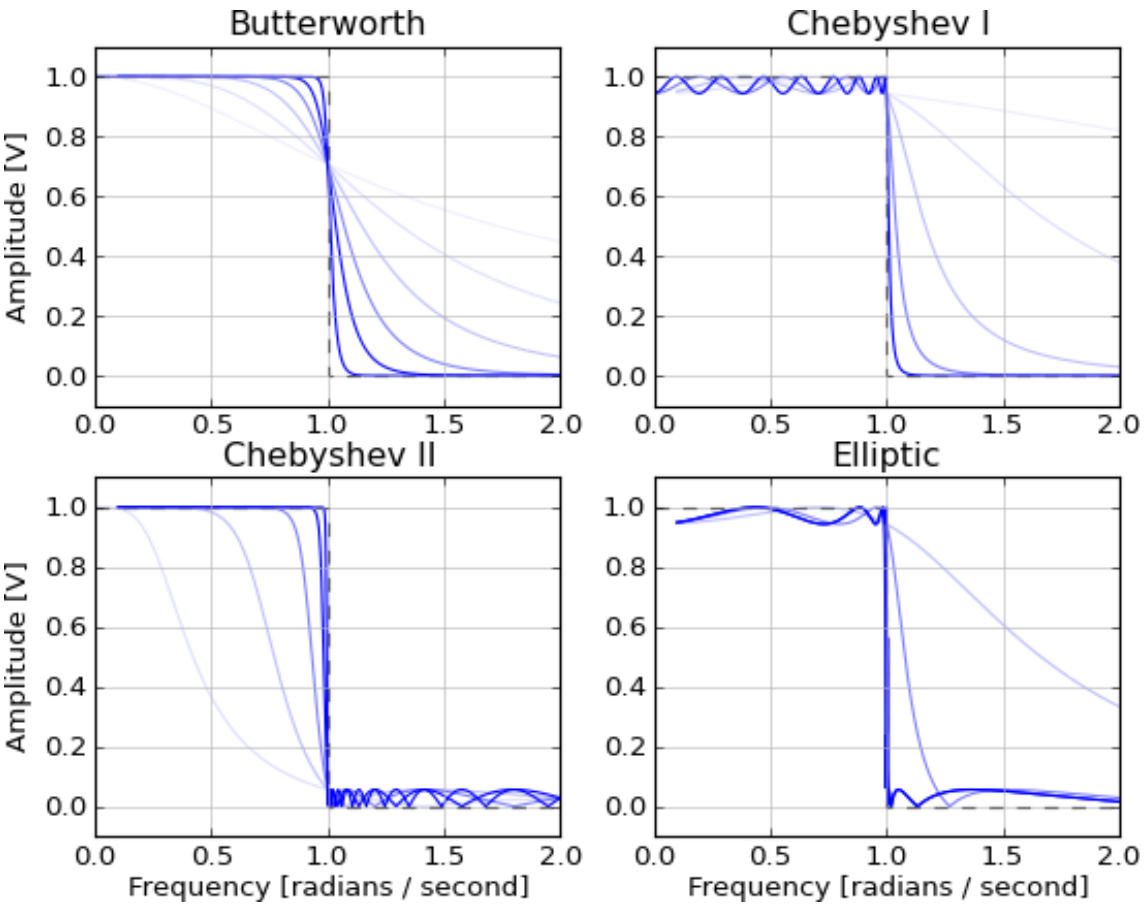
2.3.2 Signaalin suodattaminen

Signaalin eri taajuusalueiden suodattaminen on olennainen osa signaalinkäsittelyä, jotta kohina ja ylimääräiset taajuudet saadaan poistettua signaalinäytteistä. Erilaisia suodatinratkaisuja on monen tyyppisiä. Digitaaliset suodattimet perustuvat yleensä analogisen suodatinpiirin toimintaperiaatteen muuttamiseen matemaattisilla kaavoilla digitaalisesti toteutettavaksi. Digitaaliset suodattimet jaetaan pääsääntöisesti kahteen luokkaan: FIR (Finite Impulse Response) - ja IIR (Infinite Impulse Response) -tyyppisiin suodattimiin. (19.)

Digitaalisten suodattimien toiminta perustuu signaalinäytteiden kertomiseen suodatinfunktion impulssivasteen kertoimilla, joka suodatintyypistä riippuen vaimentaa haluttujen taajuuksien amplitudia signaalissa. FIR- ja IIR-suodattimien matemaattinen ero on se, että IIR-suodattimet toimivat rekursiivisesti, eli signaalinäytteiden kertomisessa otetaan huomioon aiempien näytteiden tuloksia. Näin IIR-suodattimet vaativat huomattavasti vähemmän laskentatehoa kuin FIR-tyypin suodattimet ja sopivat siksi signaalin suodattamiseen reaaliaikaisesti. (20.)

Erityyppiset IIR-suodattimet sopivat erilaisiin käyttötarkoituksiin, mutta työssä on käytetty vain Butterworth-tyypin suodatinta, koska sen taajuusvaste on tasainen suodatetuilla ja läpipääsevillä taajuuksilla, eikä signaalin voimakkuuden vaimennuksessa ole aaltoilua. Butterworth-suodattimen siirtofunktion kertoimien laskeminen ohjelmallisesti oli myös helpommin toteutettavissa kuin muilla tutkimillani suodatintyypeillä.

Toisaalta Butterworth-suodattimen taajuusvaste ei kasva yhtä nopeasti katkaisutaajuuden kohdalla, eli suodatettavien taajuuksien voimakkuus ei heikkene yhtä nopeasti kuin muissa suodatintyypeissä (kuva 7). Taajuusvasteen jyrkkenemistä voidaan kuitenkin kasvattaa lisäämällä suodattimen rekursiivisuutta eli suodattimen järjestystä, mutta se kasvattaa samalla laskentatehovaatimusta ja lisää signaalin vaihesiirtoa alkuperäiseen signaaliin verrattuna (21).



KUVA 7. Erityyppisten IIR-alipäästösuodattimien taajuusvasteet (20)

Digitaalisen suodattimen toteuttaminen vaatii suodattimen toimintaa taajuusalueella kuvaavan funktion siirtämiseen matemaattisilla muunnosfunktioilla muotoon, josta saadaan selville tarvittavat kertoimet, joilla käsiteltävät signaalinäytteet kerrotaan halutun suodatustoiminnon aikaansaamiseksi. Kertoimien määrittäminen riittävän tarkasti on tärkeää, jotta suodatin pysyy stabiilina. Kun suodattimen järjestystä kasvatetaan, voi laskettujen kertoimien epätarkkuus tehdä suodattimen toiminnasta epävakaata. Tämän vuoksi digitaalisesti IIR-suodattimet toteutetaan yleensä peräkkäisillä toisen tai ensimmäisen asteen suodattimilla, joihin kertoimien epätarkkuus ei vielä vaikuta. (22.)

Samalla tavalla toteutettiin ohjelmallisesti työssä käytetty Butterworth-alipäästösuodatin, joka annettujen parametrien avulla määritti haluttua suodattimen järjestystä vastaavat kertoimet peräkkäisillä alemman asteen suodattimilla. Tällä tavalla suodattimen toteuttaminen ohjelmallisesti oli myös huomattavasti yksinkertaisempaa ja sillä päästiin samaan suodatustulokseen kuin vastaavalla korkeamman asteen suodattimella.

3 GNU RADION ASENTAMINEN

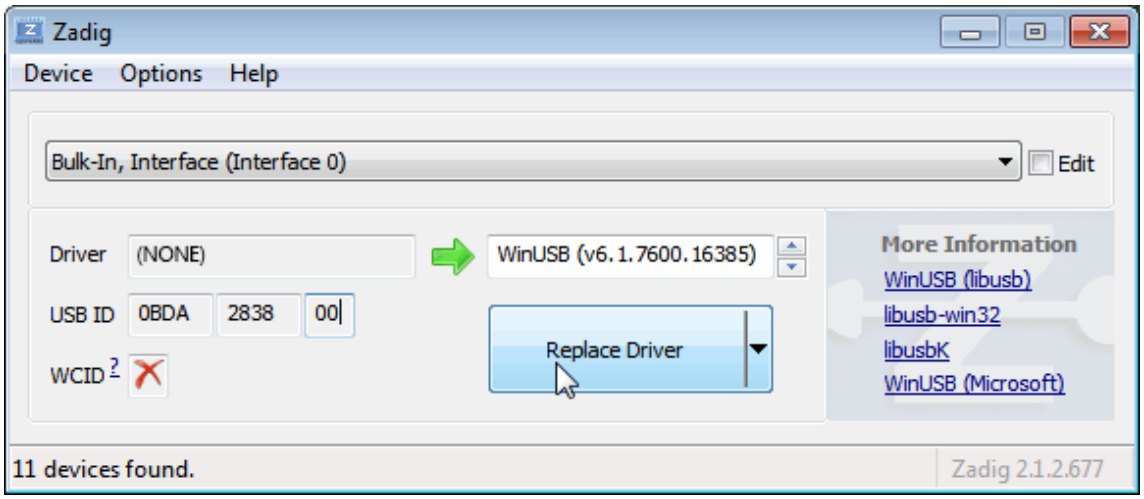
Ohjeet GNU Radion asentamiseen tuetuille käyttöjärjestelmille löytyvät GNU Radio projektin wiki-sivustolta (23). Tässä luvussa käydään lävitse GNU Radion asennusprosessi, RTL-SDR-USB-tikun käyttöönotto ja tarvittavien laiteajureiden asennus. Ohjelmisto asennettiin Windows 10 - ja Ubuntu 20.04 -käyttöjärjestelmille.

3.1 Asentaminen Windows-käyttöjärjestelmälle

GNU Radion wiki-sivuston asennusta käsittelevässä osiossa löytyy linkki Ryan Volzin Github-sivulle, jonka kautta ladattiin Windows Radioconda Installer -sovellus (24). Sovellus on valmiiksi käännetty binäärimuotoinen suoritettava ohjelma, joka hoitaa GNU Radion asennusprosessin Windows-alustalle. Radio-ohjelmiston asentaminen Radiocondan avulla oli hyvin suoraviivaista ja onnistui asennusohjelman ohjeita seuraamalla. Asennuksen jälkeen GNU Radion käynnistäminen onnistui pikakuvakkeen kautta.

Radiosignaalin vastaanottamiseen tarkoitettu RTL-SDR-USB-tikku tarvitsee myös erillisen laitteistoajurin asennettavaksi, jotta käyttöjärjestelmä osaa keskustella USB-väylään liitetyn RTL-SDR-laitteen kanssa ja vastaanottaa tulevia signaalinäytteitä. Työssä käytetty RTL-SDR-tikku sisälsi RTL2832U-piirisarjan ja tarvittavat ohjeet laitteistoajurin asentamiseen löytyivät laitteen kotisivuilta (25). Asennusohjeissa ohjeistettiin asentamaan SDRSharp-ohjelmisto, jonka mukana tulee tarvittavat laitteistoajurit ja ohjelmistopaketti RTL-SDR-vastaanottimen käyttämiseen. Koska työn tarkoituksena oli käyttää vastaanotinta yhdessä GNU Radion kanssa, ladattiin ainoastaan laitteistoajuri RTL-SDR-tiimin Github-sivustolta (26) ja tarvittava Zadig-ohjelma (27) USB-ajurin asentamista varten.

Seuraavaksi kytkettiin RTL-SDR-USB-tikku tietokoneen USB-porttiin ja käynnistettiin Zadig-ohjelma, joka tunnisti liitetyn laitteen ja ehdotti ajurin asentamista oikeaan USB-rajapintaan. Tässä kohtaa oli tärkeää varmistaa, että laitteen USB-ID oli oikea, jotta laitteistoajuria ei asennettaisi vahingossa johonkin toiseen USB-väylään kytkettyyn laitteeseen (kuva 8).



KUVA 8. RTL-SDR-laitteistoajurin asennus Zadig-ohjelman avulla

Asennusprosessin jälkeen varmistettiin ajurin toimivuus ajamalla ajuriohjelmiston kansiossa olevan *rtl_test.exe*-testiohjelma, jonka avulla pystyttiin todentamaan, että kytketty RTL-SDR-tikku toimi oikein laitteistoajurin kanssa. Laitteen toiminta varmistettiin myös GNU Radiossa luomalla käyttöliittymän avulla uusi projekti, jossa muutamia prosessointilohkoja yhdistelemällä todennettiin laitteen oikeanlainen toiminta.

3.2 Asentaminen Linux-käyttöjärjestelmälle

GNU Radion asennusohjeet tuetuille Linux-jakeluille löytyivät kätevästi projektiin wiki-sivustolta. Koska työssä käytetyn Ubuntu 20.04:n paketinhallintajärjestelmässä ei oletuksena GNU Radion pakettia ollut saatavilla, tuli ensiksi lisätä GNU Radion arkisto paketinhallintajärjestelmän saataville. Tämän jälkeen päivitettiin paketinhallintajärjestelmän tietokanta ja asennettiin GNU Radio paketinhallinnan kautta.

```
sudo add-apt-repository ppa:gnuradio/gnuradio-releases
sudo apt-get update
sudo apt-get install gnuradio
```

Radiosovelluksen asennuksen jälkeen piti myös RTL-SDR-tikun laitteistoajurit asentaa. Tämä prosessi oli hieman monimutkaisempi kuin Windowsin laitteistoajurin asennus, sillä Linuxilla ajurin lähdekoodi piti ladata versionhallintasivulta internetistä ja kääntää suoritettavaksi ohjelmistoksi käyttöjärjestelmälle. Kenn Ranousin blogisivuilta löytyi ladattava PDF-tiedosto, jossa laitteistoajurin asennusprosessi oli selkeästi ohjeistettu. (28.)

Aluksi tuli jakelun tietokannan päivityksen jälkeen asentaa tarvittavat ohjelmistopaketit laitteistoajurin lähdekoodin lataamiseksi Github-versionhallinnasta ja lähdekoodin kääntämiseksi suoritettavaksi ohjelmistoksi.

```
sudo apt-get install git cmake build-essential
```

Näiden jälkeen asennettiin C-kirjasto, joka mahdollistaa USB-laitteiden hallinnan.

```
sudo apt-get install libusb-1.0-0-dev
```

Seuraavaksi ladattiin laitteistoajurin lähdekoodi (29) Git-versionhallintatyökalun avulla ja käännettiin ohjelmisto suoritettavaksi binääriksi CMake-ohjelman avulla.

```
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
```

Ohjelmiston kääntämisen jälkeen päivitettiin kernelin ajonaikainen linkittäjä ja kopioitiin tarvittavat säännöt lähdekoodista rules.d-kansion alle.

```
sudo ldconfig
sudo cp ../rtl-sdr.rules /etc/udev/rules.d/
```

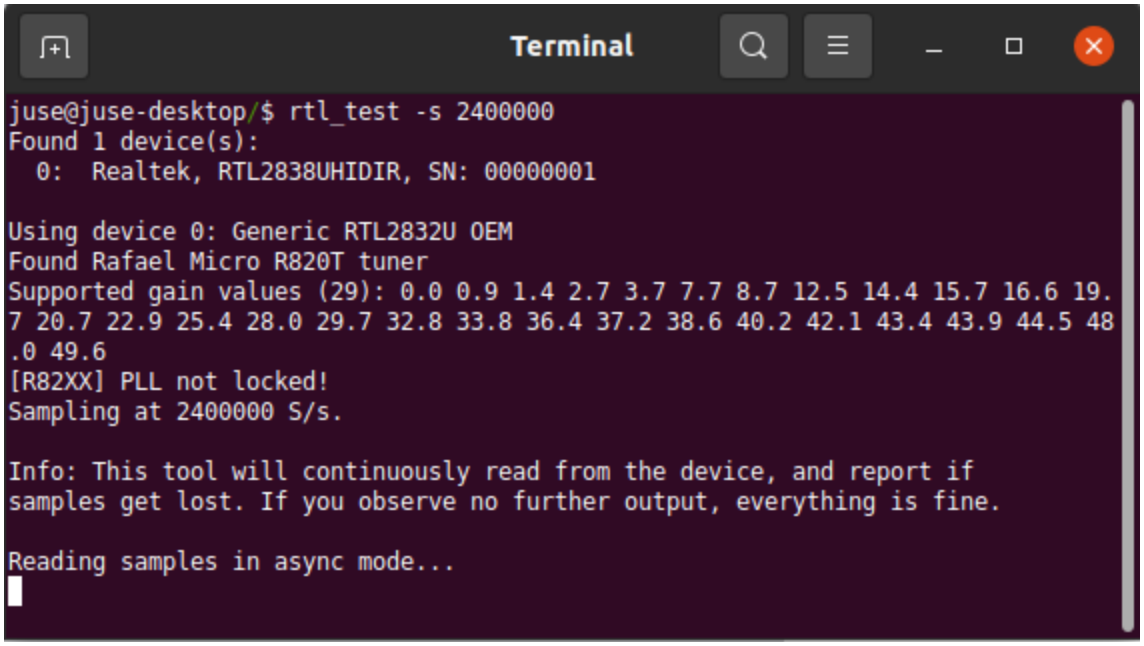
Linux-kerneliin on oletuksena asennettu laitteistoajuri RTL-SDR-tikulle, joka käynnistetään automaattisesti, kun USB-laite kytketään USB-väylään. Tämä ajuriohjelmisto tulee kytkeä pois käytöstä, jotta juuri asennettu ajuriohjelmisto korvaisi oletusajurin ja RTL-SDR-laitetta voitaisiin käyttää siihen tarkoitukseen mihin sitä tarvitaan. Tämän tapahtuu lisäämällä modprobe.d-kansioon uuden asetustiedoston, johon lisätään tarvittava rivi oletusajurin kytkemiseksi pois päältä.

```
sudo bash -c "echo 'blacklist dvb_usb_rtl28xxu' > /etc/modprobe.d/blacklist-rtl.conf"
```

Tämän jälkeen kaikki tarvittavat vaiheet ajuriohjelmiston asennuksessa on tehty ja jotta muutokset tulevat voimaan, on tietokone käynnistettävä uudelleen. Tämän jälkeen ajamalla testiskripti voidaan todentaa asennuksen toimivuus, kun RTL-SDR-tikku on kytkettynä USB-väylään.

```
rtl_test -s 2400000
```

Asennus on onnistunut ja RTL-SDR-tikku on käyttövalmiina, kun testiohjelma pystyy vastaanottamaan näytteitä ajurilta onnistuneesti (kuva 9).



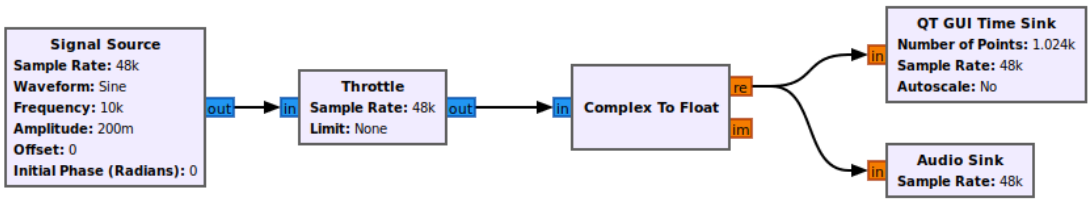
KUVA 9. RTL-SDR-laitteistoajurin testitulostus Ubuntun terminaalissa

4 SIGNAALINKÄSITTELYLOHKOT

4.1 Signaalinkäsittelylohkojen toimintaperiaate

GNU Radion toimintaperiaate perustuu erillisiin signaalinkäsittelylohkoihin, joiden tehtävänä on vastaanottaa signaalinäytteitä, käsitellä niitä ja lähettää käsitellyt näytteet eteenpäin seuraavalle lohkolle. Näitä lohkoja ketjuttamalla voidaan toteuttaa erilaisia signaalinkäsittelyketjuja, joiden toimintaperiaate vaihtelee yksinkertaisesta hyvinkin monimutkaisiin järjestelmiin.

Lohkot voivat toimia myös näytteiden tuottajina, jolloin lohko ei vastaanota signaalinäytteitä, vaan tuottaa niitä seuraavan lohkon käsiteltäväksi. Lohko, joka lukee RTL-SDR-vastaanottimen saamaa radiosignaalia laiteajurilta ja lähettää sen eteenpäin, on esimerkki näytteiden tuottajasta, joka toimii signaalinkäsittelyketjun ensimmäisenä lohkona. Signaalinkäsittelyketjun täytyy myös loppua johonkin, joten lohko voi toimia myös ketjun päätepisteenä, joka ei käsittelemiään näytteitä enää lähetä eteenpäin. Signaalin taajuusalueen visualisointilohko tai signaalinäytteiden ääneksi muuntava lohko ovat esimerkkejä signaalinkäsittelylohkon viimeisestä prosessointilohkosta. (Kuva 10.)



KUVA 10. GNU Radion lohkoista koottu signaalinkäsittelyketju

Lohko määrittelee, minkä tyyppisiä näytteitä se vastaanottaa ja minkä tyyppisiä näytteitä lohko tuottaa käsittelyn jälkeen. Erilaisia tyyppejä ovat esimerkiksi kompleksiluvut tai liukuluvut. Signaalinkäsittelylohkojen toimintaa voidaan ohjata syötettävillä parametreilla, jotka hienosäätävät lohkon toimintoja. Yhden lohkon tuottamat näytteet voidaan ohjata useampaankin eri lohkoon ja näitä erityyppisiä lohkoja yhdistelemällä voidaan toteuttaa monenlaisia signaalinkäsittelyketjuja eri käyttötarkoituksiin.

4.2 Uusien signaalinkäsittelylohkojen luominen

4.2.1 Lohkon luominen Linuxissa

GNU Radion asennusprosessin mukana asennetaan ohjelmaskriptejä, joiden avulla uusien OOT (Out-Of-Tree) -moduulien pohjakoodin luominen on suoraviivaista ja helppoa. Out-Of-Tree-moduuli tarkoittaa GNU Radion ohjelmakomponenttia, jota hallinnoidaan GNU Radion lähdekoodin ulkopuolella. Jos GNU Radion toimintaa halutaan laajentaa omilla signaalinkäsittelylohkoilla tai funktioilla, luodaan tällainen OOT-moduuli, jonka lähdekoodin hallinta on ohjelmoijan käsissä. (30.)

Moduulien luominen, uusien lohkojen lisääminen ja niiden hallinnointi, on helpointa toteuttaa käyttämällä gr_modtool-nimistä ohjelmaskriptiä. Uuden moduulin tekeminen aloitetaan luomalla uusi kansio ja ajamalla sen sisällä seuraava komento.

```
gr_modtool newmod uusi_moduuli
```

Skripti luo uuden gr-uusi_moduuli-nimisen kansion, joka sisältää uuden moduulin tarvitsemat kansiot ja tiedostorakenteet, jotta moduuli voidaan kääntää ja lisätä GNU Radion käyttöliittymän saataville. Seuraavassa vaiheessa siirrytään luomaan uusi lohko moduulin sisälle gr_modtool ohjelmaskriptin avulla.

```
gr_modtool add testi_lohko
```

Ohjelma tulostaa näytölle listan kysymyksiä, joilla luotava lohko määritellään. Valitaan mm. lohkon tyyppi, ohjelmointikieli ja mahdolliset käynnistysargumentit. Lohkolle voidaan myös valita luotavaksi pohjakoodi testeille, joilla lohkon oikea toiminnallisuus voidaan varmistaa. (Kuva 11.)

```
GNU Radio module name identified: uusi_moduuli
('sink', 'source', 'sync', 'decimator', 'interpolator', 'general', 'tagged_stream', 'hier', 'noblock')
Enter block type: sync
Language (python/cpp): cpp
Language: C++
Block/code identifier: testi_lohko
Please specify the copyright holder:
Enter valid argument list, including default arguments:
 float kerroin
Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] n
```

KUVA 11. Uuden lohkon luominen GNU Radiolle gr_modtool-ohjelmaskriptillä

Ohjelmaskripti luo moduulin kansioihin tarvittavat tiedostot ja lähdekoodin uudelle lohkolle, sekä muokkaa Cmakelists.txt-tiedostoja, jotta lohkon lähdekoodi saadaan mukaan moduulin rakentamisprosessiin CMake-ohjelmalla. Työssä käytettiin C++-ohjelmointikieltä lohkojen toteuttamiseen, joten seuraavaksi käydään läpi, mitä kyseisellä kielellä luodun lohkon lähdekoodista pitää muokata, jotta moduulin saa käännettyä ja asennettua GNU Radion saataville.

Uuden lohkon pääluokka, missä lohkon signaalinkäsittelyn logiikka tapahtuu, sijaitsee moduulin kansiorakenteessa nimellä *lib/testi_lohko_impl.h*. Luotu uusi lohko käsittelee signaalinäytteitä synkronisesti, eli se tuottaa saman verran näytteitä ulos, kuin mitä se saa käsiteltäväksi. Signaalinäytteiden käsittely tapahtuu luokan work-funktion sisällä. Lohkon muodostajafunktio saa argumentikseen float-tyyppisen muuttujan, joka määriteltiin uuden lohkon luomisen yhteydessä gr_modtool-ohjelmaskriptillä (kuva 12). Testilohkon funktioiden määrittelykoodi löytyy *lib/testi_lohko_impl.cc*-nimellä moduulin kansiosta. Lohkon koodipohjaan tehdään seuraavaksi muutamia muutoksia sen toimintaperiaatteeseen. Tämä esimerkkilohko kertoo vastaanotetut näytteet määritellyllä kertoimella ja syöttää näytteet eteenpäin seuraavalle lohkolle.

```
namespace gr {
  namespace uusi_moduuli {

    class testi_lohko_impl : public testi_lohko
    {
     private:
      // Nothing to declare in this block.

     public:
      testi_lohko_impl(float kerroin);
      ~testi_lohko_impl();

      // Where all the action really happens
      int work(
              int noutput_items,
              gr_vector_const_void_star &input_items,
              gr_vector_void_star &output_items
      );
    };

  } // namespace uusi_moduuli
} // namespace gr
```

KUVA 12. Uuden lohkon pääluokka tiedostossa testi_lohko_impl.h

Testilohkon vastaanottamien ja tuottamien näytteiden tyyppi määritellään aluksi input_type- ja output_type-aliaksilla ja poistetaan generoidut pragma-varoitusviestit niiden välistä. Esimerkkilohko vastaanottaa ja tuottaa liukulukutyyppisiä näytteitä. Lohkon muodostajafunktiossa määritellään, kuinka monesta portista tulevia signaalinäytteitä vastaanotetaan, ja kuinka monta porttia ulostulevia näytteitä lohko tuottaa. Tässä esimerkissä määritellään vain yksi sisään- ja ulostuloportti signaalinäytteille. Lohkon vastaanottama kerroin-argumentti tallennetaan luokan sisäiseksi muuttujaksi. (Kuva 13.)

```
using input_type = float;
using output_type = float;
testi_lohko::sptr
testi_lohko::make(float kerroin)
{
  return gnuradio::make_block_sptr<testi_lohko_impl>(
    kerroin);
}


/*
 * The private constructor
 */
testi_lohko_impl::testi_lohko_impl(float kerroin)
  : gr::sync_block("testi_lohko",
                   gr::io_signature::make(1 /* min inputs */, 1 /* max inputs */, sizeof(input_type)),
                   gr::io_signature::make(1 /* min outputs */, 1 /*max outputs */, sizeof(output_type))),
                   kerroin_(kerroin)
{}
```

KUVA 13. Lohkon muodostukseen liittyvät funktiot tiedostossa testi_lohko_impl.cc

Lohkon work-funktiossa saadut näytteet käydään läpi, kerrotaan tallennetulla kerroin_-muuttujalla ja tallennetaan saatu arvo ulostuleviin näytteisiin. Myös pragma-varoitukset poistetaan funktion sisältä (kuva 14). Esimerkkilohkon käsittelykoodi on nyt valmis, ja seuraavaksi muokataan moduulin kansiossa olevaa *grc/uusi_moduuli_testi_lohko.block.yml*-tiedoston sisältöä vastaamaan lohkon määrittelyä.

```
int
testi_lohko_impl::work(int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)
{
  auto in = static_cast<const input_type*>(input_items[0]);
  auto out = static_cast<output_type*>(output_items[0]);

  // Input samples are multiplied by the given factor
  for (size_t i = 0; i < static_cast<size_t>(noutput_items); i++) {
    out[i] = in[i] * kerroin_;
  }

  // Tell runtime system how many output items we produced.
  return noutput_items;
}
```

KUVA 14. Lohkon work-funktio, missä signaalinkäsittely tapahtuu

Tiedostoon täydennetään lohkon hyväksymien argumenttien tyypit ja niiden selitystekstit, jotka näkyvät GNU Radion käyttöliittymässä. Esimerkkilohko vastaanottaa vain yhden argumentin, jolla määritellään signaalin kerroin. Lisäksi määritellään vastaanotettavien ja tuotettavien näytteiden tyyppi, eli tässä tapauksessa molemmat ovat liukulukuja. (Kuva 15.)

```
id: uusi_moduuli_testi_lohko
label: testi_lohko
category: '[uusi_moduuli]'

templates:
  imports: from gnuradio import uusi_moduuli
  make: uusi_moduuli.testi_lohko(${kerroin})

parameters:
- id: kerroin
  label: Signaalin kerroin
  dtype: float
  default: 1.0

inputs:
- label: in
  dtype: float

outputs:
- label: out
  dtype: float

file_format: 1
```

KUVA 15. Lohkon datatyyppien määrittely GNU Radion käyttöliittymää varten

Yksinkertaiseen esimerkkilohkoon tarvittavat määrittelyt on nyt tehty ja ohjelmakoodi on valmis kääntämistä varten. Lohkoa luodessa gr_modtool-ohjelmaskripti loi valmiiksi tarvittavat Python-sidokset, joiden avulla Python-tulkki voi kutsua lohkon suorittavaa koodia ajon aikana. Näihin tiedostoihin ei tarvitse koskea, koska lohkon argumentteihin ei ole tullut muutoksia lohkon luomisen jälkeen.

4.2.2 Lohkon kääntäminen ja asentaminen Linuxissa

Kun moduuli ja sen ainoa esimerkkilohko on kirjoitettu, niin seuraavana on vuorossa moduulin kääntäminen ja asentaminen GNU Radion käyttöliittymän saataville CMake-ohjelman avulla. Ennen moduulin kääntämistä asennetaan tarvittavat ohjelmat paketinhallinnan avulla.

```
sudo apt-get install gnuradio-dev cmake libspdlog-dev clang-format
```

Seuraavaksi luodaan uusi kansio, jonka sisällä kääntäminen tapahtuu. Kansion sisällä ajetaan komento, joka lukee projektin CMakelist.txt-tiedostot ja tarkistaa, että tarvittavat ohjelmistot ovat saatavilla kääntämistä varten.

```
mkdir build
cd build
cmake ..
```

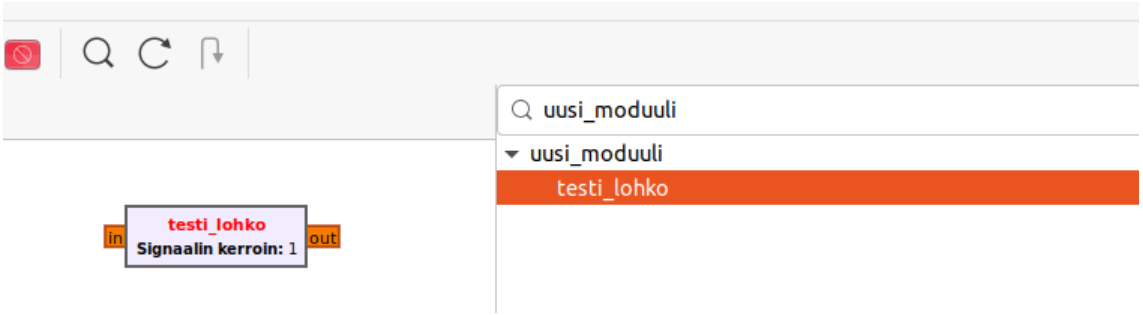
CMake-alustuksen läpimeno ilman virheitä tarkoittaa sitä, että tarvittavat ohjelmat löytyivät ja projekti on valmiina kääntämistä sekä asentamista varten. Tämä voidaan tehdä seuraavilla komennoilla, jotka kääntävät lähdekoodin binääriksi ja asentavat tarvittavat tiedostot oikeaan paikkaan, josta GNU Radion käyttöliittymä ne löytää.

```
make
sudo make install
sudo ldconfig
```

Kun moduulin kääntäminen ja asentaminen on suoritettu onnistuneesti, tulee moduuli näkyviin GNU Radion käyttöliittymässä (kuva 16). Moduuliin voidaan lisätä myöhemmin uusia lohkoja gr_modtool-ohjelmaskriptillä. Lohkoja voidaan myös muuttaa tai poistaa moduulin sisältä. Moduulin asennus GNU Radiolta voidaan poistaa seuraavalla komennolla.

```
sudo make uninstall
```

Linux-käyttöjärjestelmässä uusien moduulien luominen ja kääntäminen on suoraviivaista ja helppoa. Tämä johtuu siitä, että GNU Radio on alun perin kehitetty Linux-alustalle.



KUVA 16. Uusi moduuli GNU Radion käyttöliittymässä

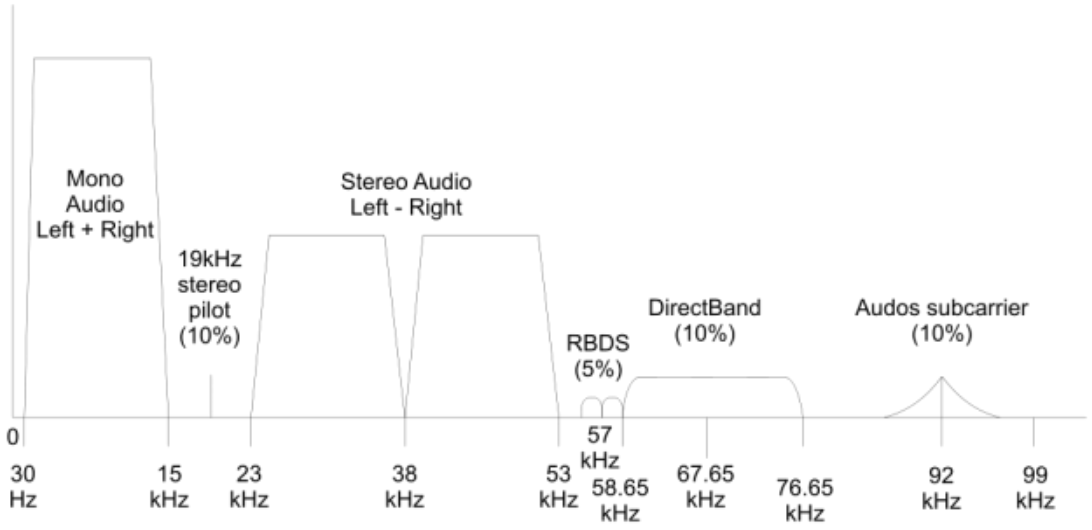
4.2.3 Lohkon kääntäminen ja asentaminen Windowsissa

GNU Radion asentaminen Windowsille on yksinkertainen prosessi, mutta C++-ohjelmointikielellä kirjoitettujen OOT-moduulien kääntäminen ja asentaminen on huomattavasti hankalampaa. Syynä tähän on se, että vaikka GNU Radion koodi on pyritty pitämään alustariippumattomana, on se alun perin kehitetty Linux ympäristössä käyttäen sille tarkoitettuja työkaluja ja skriptejä. Kehityksessä on käytetty myös useita ulkoisia kirjastoja, jotka vaativat usein omat käyttöjärjestelmäriippuvaiset asennusprosessinsa. Koska uudet OOT-moduulit joudutaan kääntämään paikallisesti, on käännösympäristön ja riippuvuuksien konfiguroiminen Windowsissa huomattavasti haastavampaa kuin Linuxissa. (31.)

5 DEMODULOINNISSA TARVITTAVIEN LOHKOJEN TOTEUTUS

5.1 FM-radiolähetyksen rakenne

Suomessa analogiset FM-radiolähetykset noudattavat eurooppalaista standardia, jossa lähetykset sijoittuvat 87,5–108 MHz:n taajuusalueelle, joka on jaettu 200 kHz:n radiokanaviin (32). FM-radiolähetys kantataajuudella koostuu mono- ja stereolähetyksistä, joiden välillä on 19 kHz:n kohdalla ohjaussignaali ilmaisemassa, että lähetyksellä on stereosignaali ja helpottamassa sen demodulointia (33). Lisäksi lähetys voi sisältää myös mahdollisen RDS-signaalin, jolla voidaan välittää digitaalista tietoa radiolähetyksestä. (Kuva 17.)



KUVA 17. FM-radiolähetyksen komponentit kantataajuudella (34)

Opinnäytetyössä keskityttiin demoduloimaan FM-radiolähetyksen audio monosignaalina kaiuttimille, mikä vaatii vastaanottimelta yksinkertaisempaa rakennetta kuin stereosignaalin demoduloiminen. Tässä luvussa kerrotaan, miten GNU Radion lohkot FM-signaalin demoduloimiseen toteutettiin. Luvussa myös pohditaan demoduloinnin onnistumista omia lohkoja hyödyntäen, verrattuna GNU Radion standardilohkoista kootun FM-vastaanottimen suorituskykyyn.

5.2 FM-moduulin lohkojen toteutus

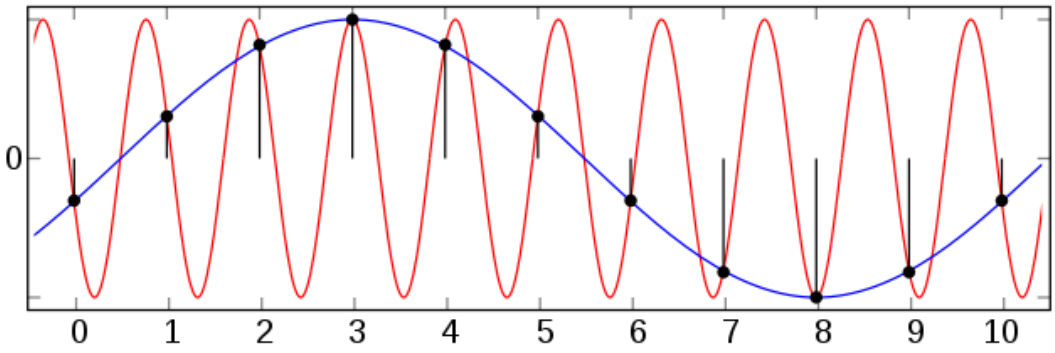
Demodulaattorin toteutus aloitettiin luomalla aluksi uusi moduuli gr_modtool-asennuskriptin avulla, jonka käyttämisestä on kerrottu edellisessä luvussa. Uuden moduulin nimi on FM-demodulator ja sen sisälle luodaan uudet lohkot, joita demoduloinnissa tarvitaan. Vastaanotetun radiosignaalin lukemiseen on käytetty GNU Radion standardilohkoa, joka lukee signaalinäytteet laiteajurilta, ja lähettää I/Q-signaalinäytteet eteenpäin seuraavalle lohkolle. Tälle lohkolle annetaan argumentteina taajuus, mistä signaali siirretään kantataajuudelle, sekä haluttu näytteistystaajuus.

RTL-SDR-vastaanottimen näytteistystaajuudet ovat hyvin korkeita, ja pienin taajuus, joilla näytteistyksiä voidaan lukea laitteistoajurilta, on 1,024 MHz. Ensimmäisenä lohkona toteutetaan desimaattori, jonka toimintaperiaatteena on näytteistystaajuuden pienentäminen, jotta seuraavien lohkojen ei tarvitse prosessoida näytteitä suuremmilla näytteistystaajuuksilla kuin on tarpeen.

5.2.1 Desimaattori

Desimaattorin tehtävänä signaalinkäsittelyketjussa on pienentää käsiteltävien näytteiden näytteistystaajuutta. Tällä voidaan vähentää seuraavien lohkojen prosessointityötä, jos alkuperäinen näytteistystaajuus on suurempi kuin lohkon prosessi tarvitsee optimaaliseen toimintaan. Lohkon toimintaperiaate voi myös vaatia tietynlaisen näytteistystaajuuden, kuten audiosignaalin käsittelylohko, joka lähettää prosessoitavat näytteet äänikortille 48 kHz:n taajuudella.

Desimaattori käytännössä valitsee saapuvista näytteistä joka N:nnen näytteen, riippuen asetetusta arvosta, millä alkuperäinen näytteistystaajuus halutaan jakaa. Desimaattorin vastakohta on interpolaattori, jolla näytteistystaajuutta voidaan kasvattaa. Ennen kuin signaalin näytteistystä voidaan pienentää, tulee alipäästösuodattimen avulla suodattaa signaalista liian korkeat taajuudet pois, jotta vältytään signaalin laskostumiselta. Tämä johtaa signaalin vääristymiseen ja johtuu liian pienestä näytteistystaajuudesta signaalin kaistanleveyteen nähden (kuva 18).



KUVA 18. Signaalin laskostuminen liian pienellä näytteistystaajuudella (35)

Signaalinkäsittelyketjussa tarvitaan desimaattoria myös toisessa vaiheessa demodulointilohkon jälkeen, jossa demoduloidun äänisignaalin näytteistystaajuus tuli sovittaa äänentoiston 48 kHz:n taajuudelle. Ainoa ero näiden kahden vaiheen välillä on näytteiden tyyppi. Ensimmäisen vaiheen desimaattorin signaalinäytteet ovat kompleksilukuja, kun taas demoduloinnin jälkeinen desimaattori käsittelee liukulukuja. Tämän vuoksi desimaattorin lähdekoodissa hyödynnettiin C++- templateja, jolla samasta koodipohjasta voidaan kääntää lohkot erityyppisien signaalinäytteiden käsittelyyn. GNU Radion käyttöliittymässä voidaan dynaamisesti vaihtaa desimaattorin tyyppi kulloisessakin signaalinkäsittelyketjun kohdassa.

Kun luokasta ja sen jäsenfunktioista tehdään template-mallit, voidaan sama koodipohja kääntää eri tyyppisten datatyyppien käsittelyyn. Ainoastaan datatyyppikohtaiset käsittelyt pitää ohjelmoida erikseen. Desimaattorin tapauksessa käytetään filter_iq_t-tyypin luokkaa signaalin suodattamiseen, jos datatyyppinä on kompleksiluku. Kyseinen suodatinluokka sisältää tällöin kaksi Butterworth-tyypin alipäästösuodatinta, joilla suodatetaan syötettyjen signaalinäytteiden I- ja Q-komponentit. Muiden datatyyppien kohdalla riittää vain yksi suodatin, jolloin käytetään filter_t-tyypin luokkaa. Kyseiset luokat on määritelty moduulin nimiavaruuden sisällä ja käytetty luokka valitaan käännösvaiheessa. (Kuva 19.)

```
template<class T>
class decimator_xx_impl : public decimator_xx<T>
{
 private:
   You, 1 second ago | 1 author (You)
  using filter_type = typename std::conditional<
    std::is_same<T, gr_complex>::value, filter_iq_t, filter_t>::type;

  int decimation_;
  filter_type IIR_filter;
 public:
  decimator_xx_impl(int decimation, float sample_hz, float cutoff_hz, int filter_order);
  ~decimator_xx_impl();

  int work(      You, 2 weeks ago • Added new template decimator block
           int noutput_items,
           gr_vector_const_void_star &input_items,
           gr_vector_void_star &output_items
  );
};
```

KUVA 19. Desimaattorin pääluokan template-malli

Desimaattorin yml-tiedostossa määritellään desimaattorille enum-tyyppinen type-parametri, jonka perusteella Python-tulkki osaa kutsua oikeaa muodostusfunktiota, riippuen käytetystä datatyypistä, joka GNU Radion käyttöliittymässä on valittuna (kuva 20). Eri datatyyppjä tukeville luokille on myös luotava omat sidosfunktionsa Python-tulkille, jotka määritellään moduulin hakemiston *python/{moduuli}/bindings/{lohko}.cc*-tiedostossa (kuva 21). Desimaattorille luodaan funktiot float- ja complex-tyypeille, joita desimaattorin C++-toteutus tukee.

```
label: Decimator
category: '[fm_module]'
flags: [ python, cpp ]

templates:
  imports: from gnuradio import fm_module
  make: fm_module.decimator_${type.fcn}(${decimation}, ${sample_hz}, ${cutoff_hz}, ${filter_order})

parameters:
- id: type
  label: IO Type
  dtype: enum
  options: [complex, float]
  option_attributes:
      fcn: [cc, ff]
  hide: part
```

KUVA 20. Desimaattorin tyypin määrittely yml-tiedostossa

```
template<typename T>
void bind_decimator_xx_template(py::module& m, const char* name)
{
    using decimator_xx = gr::fm_module::decimator_xx<T>;

    py::class_<decimator_xx, gr::sync_decimator,
        std::shared_ptr<decimator_xx>>(m, name, D(decimator_xx))
        .def(py::init(&decimator_xx::make),
           D(decimator_xx,make)
        );
}

void bind_decimator_xx(py::module& m)
{
    bind_decimator_xx_template<float>(m, "decimator_ff");
    bind_decimator_xx_template<gr_complex>(m, "decimator_cc");
}
```

KUVA 21. Desimaattorin python sidosten muodostusfunktiot

Desimaattorille syötettävissä parametreissa määritellään käsiteltävien signaalinäytteiden näytteistystaajuus ja digitaalisen suodattimen järjestys, jolla signaalinäytteet aluksi suodatetaan laskostumisen ehkäisemiseksi. Parametreissa on mahdollista määritellä myös tarvittavaa alhaisempi katkaisutaajuus suodattimille, jolloin signaalinkäsittelyketjussa tarvittava suodatus voidaan hoitaa jo desimointivaiheessa. Tällä tavalla desimaattori hoitaa signaalin suodattamisen ketjun seuraavana lohkona olevalle demodulaattorille, jolloin FM-demodulointi voidaan tehdä saaduista näytteistä ilman uutta suodatusta.

5.2.2 Kvadratuurinen FM-demodulaattori

Signaalinkäsittelyketjun seuraavana lohkona on FM-demodulaattori, joka vastaa taajuusmoduloidun radiosignaalin demoduloinnista alkuperäiseksi ääneksi, käyttäen kvadratuurista demodulointitekniikkaa. Lohkolle syötetään desimaattorin käsittelemät I/Q-signaalinäytteet, jotka demoduloidaan. Tuloksena saadut signaalinäytteet menevät ketjun seuraavalle lohkolle käsiteltäviksi.

Ennen kuin lasketaan signaalin I- ja Q-komponenttien vaihepoikkeamaa arkustangenttifunktiolla, on tärkeää varmistaa, ettei näytteen reaaliosa ole nolla, sillä muuten jakolasku johtaisi nollalla jakamiseen. Demoduloidut signaalinäytteet normalisoidaan välille ±1,0 jakamalla arkustangentin tulos puolella piin arvosta. Käsittelemätön signaalinäyte tallennetaan prev_sample_-muuttujaan, jota käytetään aina seuraavan näytteen vertailuarvona, jotta näytteiden välinen vaihe-ero saadaan laskettua. (Kuva 22.)

```
int
frequency_demodulator_impl::work(int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)
{
  auto in = static_cast<const input_type*>(input_items[0]);
  auto out = static_cast<output_type*>(output_items[0]);

  for (size_t i = 0; i < static_cast<size_t>(noutput_items); i++) {
    auto sample = in[i];
    auto reference = sample * prev_sample_;
    auto output = reference.real() ? atan(reference.imag() / reference.real()) / M_PI_2 : 1.0;
    out[i] = static_cast<output_type>(output);
    prev_sample_ = gr_complex(sample.real(), -sample.imag());
  }

  return noutput_items;
}
```

KUVA 22. FM-demodulaattorilohkon signaalinkäsittelyfunktio

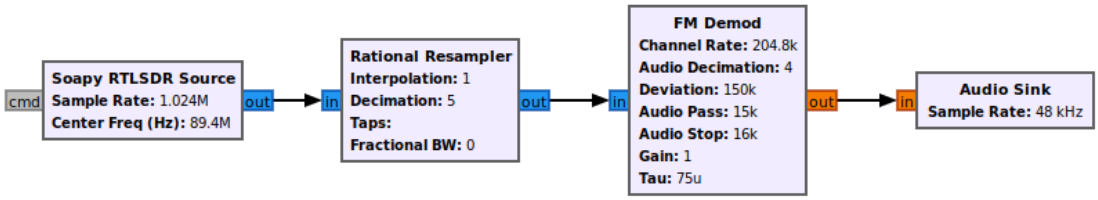
Signaalinkäsittelyketjun seuraavassa vaiheessa demodulaattorilohkon tuottamat float-tyyppiset signaalinäytteet syötetään ketjun toiselle desimaattorille, jossa suodatetaan signaalista yli 20 kHz:n taajuudet pois ja lasketaan näytteistystaajuutta lähelle 48 kHz:n taajuutta. Ketjun viimeisessä lohkossa signaalinäytteet lähetetään äänikortille äänimuodossa toistettavaksi.

6 FM-SIGNAALIN DEMODULOINTI

Tässä viimeisessä luvussa verrataan kahden GNU Radion signaalinkäsittelyketjuilla toteutetun FM-demodulaattorin suorituskykyä eri taajuuksilla vastaanotettujen radiolähetysten demoduloinnissa. Toinen signaalinkäsittelyketju on rakennettu käyttäen GNU Radion standardilohkoja, kun taas toisessa ketjussa demoduloinnista vastaavat lohkot on korvattu edellisessä kappaleessa esitellyillä omilla lohkoilla.

6.1 Standardilohkoista koottu FM-demodulaattori

Standardilohkoista tehty signaalinkäsittelyketju on toteutettu Per Vices Support -sivustolta löytyvän esimerkin mukaisesti (36.). Yksinkertaisimmillaan ketjussa on RTLSDR-lohko, joka lukee signaalinäytteet laiteajurilta määritellyllä keskitaajuudella käyttäen asetettua näytteistystaajuutta. Seuraavassa lohkossa näytteistystaajuutta pienennetään ja syötetään näytteet FM-demodulaattorin käsiteltäviksi. Demodulaattorilohko vastaa signaalin demoduloinnista, audiosignaalin suodattamisesta sekä näytteistystaajuuden pienentämisestä määriteltyjen parametrien perusteella. Viimeisessä lohkossa äänisignaali syötetään äänikortille toistettavaksi. (Kuva 23.)

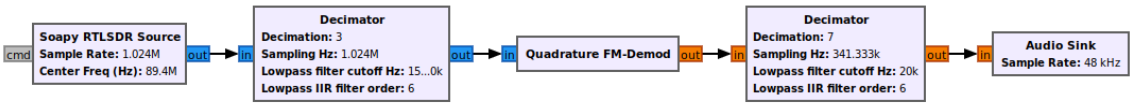


KUVA 23. GNU Radion standardilohkoista rakennettu FM-demodulaattori

6.2 Omilla lohkoilla rakennettu FM-demodulaattori

Toisessa FM-demodulaattorissa käytettiin ketjun ensimmäisenä ja viimeisenä lohkona samoja standardilohkoja kuin ensimmäisen FM-demodulaattorin signaalinkäsittelyketjussa. Ketjun demoduloinnista vastaavat lohkot korvattiin edellisessä luvussa esitellyillä, itse ohjelmoiduilla signaalinkäsittelylohkoilla. Standardilohkoista kootun signaalinkäsittelyketjun demodulaattorilohko vastaa demoduloidun äänisignaalin suodattamisesta ja uudelleennäytteistyksestä annettujen parametrien

perusteella, mutta omista lohkoista kootussa ketjussa tarvittavat suodatukset signaalille tapahtuvat desimaattorilohkoissa. Demoduloinnista vastaava lohko ei näin ollen tarvitse parametreja, koska vastaa vain IQ-signaalinäytteiden demoduloinnista äänisignaaliksi kvadratuurisella demodulointitekniikalla. (Kuva 24.)



KUVA 24. Omia lohkoja hyödyntävän FM-demodulaattorin signaalinkäsittelyketju

6.3 FM-signaalien demodulointi radiolähetyksistä

Demodulointitestissä valittiin kuusi paikallista Oulun alueella kuuluvaa radiokanavaa, joiden välillä tehtiin vastaanottotestit signaalinkäsittelyketjuilla. Testiin otettiin radiokanavat 93,2–107,4 MHz:n taajuusalueilta, joiden lähetysteho vaihteli 0,5–50 kW:n välillä (kuva 25). Demodulaattorien suorituskyky arvioitiin korvakuulolta ja tekemällä sen perusteella päätelmiä tuloksena olevan audiosignaalin taajuusnäytteistä.

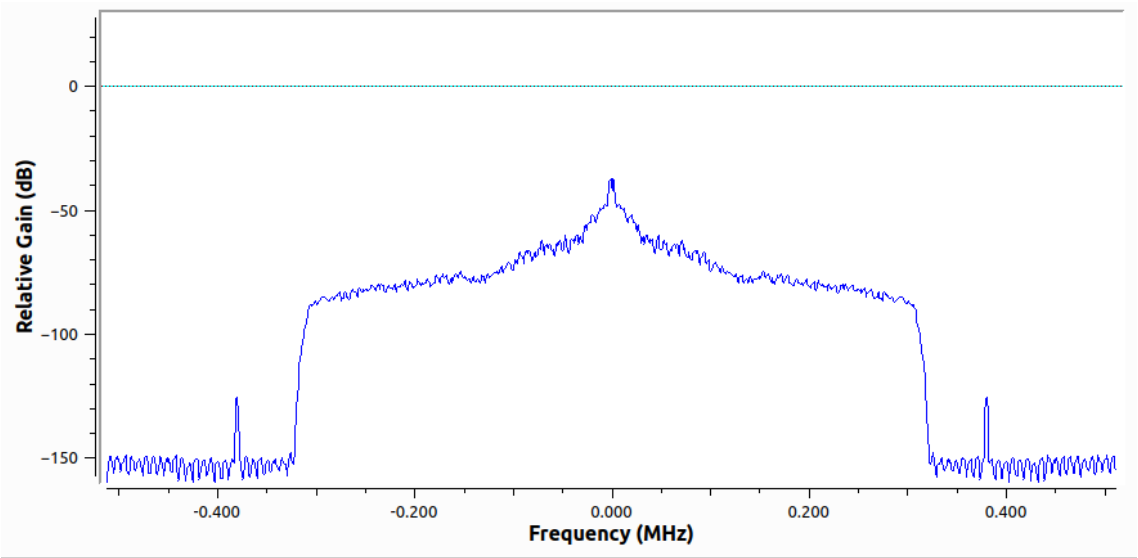
TAULUKKO 1. Demodulointitestin radiokanavat Oulun alueella (37.)

Radiokanava	Lähetystaajuus (MHz)	Lähetysteho (kW)
YleX	93,2	50
Suomi Rock	102,6	5
Radio Rock	95,8	3
Järviradio	107,4	2
NRJ	99,1	1
Yle Radio 1	104,4	0.5

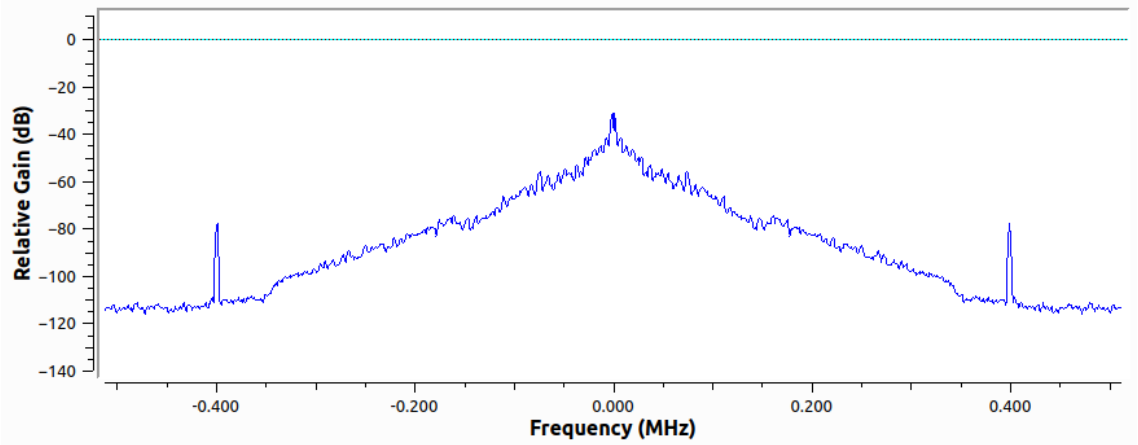
Omista lohkoista koottu signaalinkäsittelyketju suoriutui testistä lähes yhtä hyvin, kuin standardilohkoista koottu FM-demodulaattori. Radiolähetysten kuuluvuus molemmilla demodulaattoreilla oli suhteellisen hyvä kaikilla testatuilla kanavilla. Ainoastaan Järviradion ja Yle Radio 1:n kohdalla oli havaittavissa selvää rätinää äänisignaalissa. Nämä häiriöt tulivat esille molemmissa demodulaattoreissa, ehkä hiukan selvempänä omista lohkoista kootun demodulaattorin kohdalla.

Standardilohkon FM-demodulaattorin lähdekoodista selvisi, että myös siinä on käytetty kvadratuurista demodulointitekniikkaa kuten omassa FM-demodulaattorissani (38).

Suurin ero näiden signaalinkäsittelyketjujen välillä oli signaalin suodattamiseen käytetyt digitaaliset alipäästösuodattimet. Standardilohkoissa käytettiin FIR-tyyppistä suodatinta, kun taas omissa lohkoissani suodattaminen tehtiin IIR-tyyppisellä Butterworth-alipäästösuodattimella. FIR-tyyppisillä suodattimilla standardilohkoista kootussa ketjussa, saatiin signaali suodatettua katkaisutaajuden kohdalta huomattavasti jyrkemmässä kulmassa (kuva 26), kuin IIR-tyyppisellä suodattimella toisessa ketjussa (kuva 27). Tämä luultavasti mahdollisti selkeämmän erottelun haitallisista taajuuksista, jotka ilmenivät hieman selkeämpänä rätinänä äänisignaalissa omista lohkoista kootussa ketjussa.



KUVA 26. Äänisignaalin taajuuksien voimakkuudet FFT-muunnoksen jälkeen FIR-suodattimella



KUVA 27. Äänisignaalin taajuuksien voimakkuudet FFT-muunnoksen jälkeen IIR-suodattimella

Testin johtopäätöksenä voidaan todeta, että FM-signaalin kvadratuurinen demodulointi on yksinkertainen toteuttaa ja se on suhteellisen suorituskykyinen heikollakin radiosignaalilla. Signaalin voimakkuuden vaihtelut vaikuttivat demoduloidun äänen laatuun vain vähän, mutta oleellista oli suodattaa korkeammat taajuudet pois signaalista ennen demodulointia, sillä häiriöt signaalissa aiheuttivat helposti äänisignaalin säröytymistä.

7 YHTEENVETO

Opinnäytetyön tavoitteena oli tutustua GNU Radion toimintaan Windows- ja Linux-alustoilla sekä tutkia, millä tavalla uusia OOT-moduuleja voitaisiin toteuttaa ohjelmiston käyttöliittymän saataville standardimoduulien lisäksi. Tavoitteena oli myös luoda omia lohkoja hyödyntävä signaalinkäsittelyketju, joka pystyisi demoduloimaan RTL-SDR-vastaanottimen vastaanottamaa FM-radiosignaalia äänisignaaliksi.

Työn tuloksena onnistuttiin toteuttamaan omista lohkoista koottu signaalinkäsittelyketju, joka suoritui hyvin vastaanotetun FM-signaalin demoduloinnista. Uusien moduulien toteuttaminen Linux-käyttöjärjestelmälle oli suhteellisen yksinkertaista, ja kaikki tarvittavat ohjelmistotyökalut olivat helposti asennettavissa Linuxin paketinhallinnan kautta. Demoduloinnissa tarvittavien tekniikoiden toteuttaminen ohjelmallisesti onnistui myös suhteellisen hyvin, ja suurin työ oli tarvittavan digitaalisen alipäästösuodattimen ohjelmoiminen. Se vaati syvempää perehtymistä matemaattisiin algoritmeihin, joiden avulla työssä käytetyn suodattimen toimintaa kuvaava funktio saatiin siirrettyä muotoon, jolla digitaalisen suodattimen ohjelmointi muokattavilla parametreilla onnistui.

GNU Radiolle ohjelmoiduissa signaalinkäsittelylohkoissa käytetty kvadratuurinen FM-demodulointitekniikka oli suorituskyvyltään riittävä ja sen toteuttaminen oli suhteellisen yksinkertaista. Demoduloinnissa käytetty IIR-alipäästösuodatus ei kuitenkaan yltänyt yhtä hyvään suodatustulokseen, kuin lopun demodulointitestissä verrattu standardilohkoista koottu ketju, joka käytti suodattamiseen FIR-tyyppistä alipäästösuodatinta. Tämä ilmeni selkeämpänä säröilynä demoduloidussa äänisignaalissa omista lohkoista kootun ketjun kohdalla.

C++-ohjelmointikielellä toteutettujen OOT-moduulien kääntämien Windows-käyttöjärjestelmälle osoittautui kuitenkin liian haastavaksi prosessiksi, vaikka sitä useaan otteeseen yritettiin tehdä. GNU Radio on kehitetty alun perin Linux-alustalle hyödyntäen sille suunnattuja ohjelmistotyökaluja, ja siksi Windows-alustalle OOT-moduulien kääntäminen vaatisi enemmän, kuin mitä tämän työn puitteissa pystyttiin tekemään.

Opinnäytetyön tekeminen oli antoisa prosessi oman oppimisen kannalta ja sen myötä oma osaamiseni laajeni digitaalisen signaalinkäsittelyn puolella. GNU Radio osoittautui monipuoliseksi ja

joustavaksi ohjelmistoksi, jonka avulla pystytään rakentamaan ja visualisoimaan tehokkaasti digitaalisen signaalinkäsittelyn eri vaiheita.

LÄHTEET

1. GNU Radio 2023. About GNU Radio. Hakupäivä 10.10.2023. https://www.gnuradio.org/.

2. Nobel Prize 2023. Guglielmo Marconi. Hakupäivä 15.11.2023. https://www.nobelprize.org/prizes/physics/1909/marconi/biographical/.

3. Diffen 2023. AM vs FM. Hakupäivä 15.11.2023. https://www.diffen.com/difference/AM_vs_FM.

4. JavaTpoint 2023. Frequency Modulation. Hakupäivä 16.11.2023. https://www.javatpoint.com/frequency-modulation.

5. Wostu, Stu 2023. Carson's Rule. Ham Radio School. Hakupäivä 16.11.2023. https://www.hamradioschool.com/post/carson-s-rule-g8b06.

6. Electronics-notes 2023. Frequency Modulation, FM Sidebands & Bandwidth. Hakupäivä 15.11.2023. https://www.electronics-notes.com/articles/radio/modulation/frequency-modulation-fm-sidebands-bandwidth.php.

7. Elporocus 2023. Block diagram of FM Transmitter. Hakupäivä 16.11.2023. https://www.elprocus.com/making-of-fm-transmitter-circuit-working-application/.

8. Vedantu 2023. Block diagram of FM Receiver. Hakupäivä 16.11.2023. https://www.vedantu.com/question-answer/with-the-help-of-a-block-diagram-explain-the-class-12-physics-cbse-5fdb906f7dd0d60c2b41422f.

9. Wikipedia 2023. Superheterodyne receiver. Hakupäivä 17.11.2023. https://en.wikipedia.org/wiki/Superheterodyne_receiver.

10. Tutorialspoint 2023. How does an FM radio set work. Hakupäivä 16.11.2023. https://www.tutorialspoint.com/how-does-an-fm-radio-set-work.

11. Electronics-notes 2023. Frequency Modulation, FM Detection, Demodulation, Discrimination. Hakupäivä 20.11.2023. https://www.electronics-notes.com/articles/radio/modulation/fm-frequency-demodulation-detection-discrimination.php.

12. Electronics-notes 2023. FM Slope Detector. Hakupäivä 20.11.2023. https://www.electronics-notes.com/articles/radio/modulation/fm-frequency-demodulation-slope-detector-discriminator.php.

13. Electronics-notes 2023. Quadrature FM Demodulator. Hakupäivä 20.11.2023. https://www.electronics-notes.com/articles/radio/modulation/fm-frequency-demodulation-quadrature-coincidence-detector-demodulator.php.

14. Electronics-notes 2023. PLL FM demodulator. Hakupäivä 20.11.2023. https://www.electronics-notes.com/articles/radio/modulation/fm-frequency-demodulation-phase-locked-loop-pll-detector-demodulator.php.

15. Wikipedia 2021. RDS. Hakupäivä 20.11.2023. https://fi.wikipedia.org/wiki/RDS.

16. All About Circuits 2024. Understanding Quadrature Demodulation. Hakupäivä 17.1.2024. https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-demodulation/understanding-quadrature-demodulation/.

17. Keim, Robert 2018. How to Process I/Q Signals in a Software-Defined RF Receiver. All About Circuits. Hakupäivä 23.1.2024. https://www.allaboutcircuits.com/technical-articles/how-to-process-iq-signals-software-defined-rf-receiver-dsp-digital-signal/.

18. Mankovskyy, Spartak & Matiieshyn, Yurij 2023. Digital FM Demodulator with Reduced Computational Complexity. Lviv Polytechnic National University. Hakupäivä 23.1.2024. http://pe.org.pl/articles/2023/10/28.pdf.

19. Siemens 2020. Introduction to Filters: FIR versus IIR. Hakupäivä 13.2.2024. https://community.sw.siemens.com/s/article/introduction-to-filters-fir-versus-iir.

20. Stack Exchange 2013. Which IIR filters approximate a Gaussian filter. Hakupäivä 11.2.2024. https://dsp.stackexchange.com/questions/9745/which-iir-filters-approximate-a-gaussian-filter.

21. Thai, Jason 2022. Filtering Basics: Importance of Linear Phase. ErdosMiller. Hakupäivä 11.2.2024. https://info.erdosmiller.com/blog/filtering-basics-importance-of-linear-phase.

22. Wikipedia 2024. Digital biquad filter. Hakupäivä 14.2.2024. https://en.wikipedia.org/wiki/Digital_biquad_filter.

23. GNU Radio Wiki 2023. Main Page. Hakupäivä 1.11.2023. https://wiki.gnuradio.org.

24. Volz, Ryan 2023. ryanvolz / radioconda. GitHub. Hakupäivä 1.11.2023. https://github.com/ryanvolz/radioconda.

25. RTL-SDR 2023. RTL-SDR.COM. Hakupäivä 1.11.2023. https://www.rtl-sdr.com.

26. Github 2023. rtlsdrblog / rtl-sdr-blog. Hakupäivä 1.11.2023. https://github.com/rtlsdrblog/rtl-sdr-blog/.

27. Batard, Pete 2023. USB driver installation made easy. Zadig. Hakupäivä 1.11.2023. https://zadig.akeo.ie.

28. Ranous, Kenn 2020. RTL-SDR for Linux. Ranous. Hakupäivä 2.11.2023. https://ranous.files.wordpress.com/2020/05/rtl-sdr4linux_quickstartguidev20.pdf.

29. Osmocom 2023. RTL2832 laitteistoajurin lähdekoodi. Hakupäivä 2.11.2023. https://github.com/osmocom/rtl-sdr.

30. GNU Radio Wiki 2024. OutOfTreeModules. Hakupäivä 12.1.2024. https://wiki.gnuradio.org/index.php/OutOfTreeModules.

31. GNU Radio Wiki 2024. Windows Install. Hakupäivä 28.1.2024. https://wiki.gnuradio.org/index.php/WindowsInstall.

32. Traficom 2024. Frequency planning ensures the comprehensive coverage of radio and TV services throughout Finland. Hakupäivä 4.2.2024. https://www.traficom.fi/en/communications/radio-licences-and-frequencies/frequency-planning-ensures-comprehensive-coverage.

33. International Telecommunication Union (ITU) 2019. Transmission standards for FM sound broadcasting at VHF. Hakupäivä 4.2.2024. https://www.itu.int/dms_pubrec/itu-r/rec/bs/R-REC-BS.450-4-201910-I!!PDF-E.pdf.

34. Wikipedia 2024. FM broadcasting. Hakupäivä 4.2.2024. https://en.wikipedia.org/wiki/FM_broadcasting.

35. Wikipedia 2024. Laskostuminen. Hakupäivä 7.2.2024. https://fi.wikipedia.org/wiki/Laskostuminen.

36. Per Vices Support 2024. Building an FM Receiver with gnuradio. Hakupäivä 18.2.2024. https://support.pervices.com/tutorials/pvt-1/.

37. Traficom 2024. Radioasemat Suomessa. Hakupäivä 22.2.2024. https://www.traficom.fi/fi/viestinta/tv-radio-ja-muut-mediapalvelut/radioasemat-suomessa.

38. GitHub 2024. gnuradio / gnuradio. Hakupäivä 25.2.2024. https://github.com/gnuradio/gnuradio/blob/main/gr-analog/python/analog/fm_demod.py.