TARTU ÜLIKOOL
Loodus- ja täppisteaduste valdkond
Füüsika instituut

Füüsika eriala

LAUR JOOST

BPSK DEMODULEERIMINE TARKVARALISE RAADIOGA

Bakalaureusetöö (12 EAP)

Juhendajad: Viljo Allik, MSc
Indrek Sünter, MSc

Tartu 2016

BPSK demoduleerimine tarkvaralise raadioga

Bakalaureusetöö osana koostatakse tarkvaralisel raadiol põhinev BPSK vastuvõtja satelliitside maajaamade jaoks, kasutades USRP N210 tarkvaralist raadiot ning GnuRadio raamistikku. Töö osana koostatakse GnuRadio lisaplokk, mis sõltumata välistest programmidest teostab satelliidi Doppleri nihke korrektsiooni. Vastuvõtja saavutab bitiveatiheduse sõltuvuse signaal-müra suhtest 0.5 dB piires optimaalsest.

Töö käigus koostatud kood on vabalt kättesaadav asukohas [18].

Märksõnad: SDR, DSP, GnuRadio

CERCS: T121 – Signaalitöötlus

Demodulation of BPSK using Software Defined Radio

As part of this Bachelor's Thesis, a Software-Defined BPSK Receiver for satellite ground stations using Ettus USRP N210 software defined radio and the GnuRadio framework is implemented. A part of the results is a Doppler correction block for GnuRadio that has no external dependencies. The receiver achieves a bit-error rate dependency of signal-to-noise ratio within 0.5 dB of optimal.

The resulting code is freely accessible at [18].

Keywords: SDR, DSP, GnuRadio

CERCS: T121 – Signal Processing

Sisukord

Joonised

Lühendid

AWGN Additive White Gaussian Noise — Normaal (e. Gaussi) jaotusega juhuslik liituv müra.

BER Bit Error Rate — Bitivea tõenäosus.

BPSK Binary **PSK** — digitaalne kahendfaasmodulatsioon.

C-riba Raadiosageduste vahemik 4–8 GHz.

DBPSK Differential **BPSK** — Diferentsiaalne **BPSK**. Andmeid antakse edasi faasi muutumise või mittemuutumisega.

DSP Digital Signal Processing — digitaalne signaalitöötlus.

EC-1 ESTCube-1.

EC-2 ESTCube-2.

EC-3 ESTCube-3.

FSK Frequency Shift Keying — digitaalne sagedusmodulatsioon.

GFSK Gaussian **FSK** — Gaussi funktsiooniga filtreeritud **FSK**.

I/Q In-phase/Quadrature — sünfaas/kvadratuuresitus[30].

IEEE Institute of Electrical and Electronics Engineers — Elektri- ja elektroonikainseneride instituut. USA-s baseeruv ülemaailmne mittetulunduslik organisatsioon.

ITU-T "International Telecommunication Union – Telecommunication Standardization Sector".

MCS Mission Control System — Missioonijuhtimissüsteem.

MSK Minimum Shift Keying — **FSK**, mille puhul sagedusdeviatsioon on $1/2$ bitisagedusest.

PRBS9 Pseudo-Random Binary Sequence 9 — 9-bitine kahend-pseudomüra.
9-bitise tagasisidestatud nihkeregistriga genereeritav maksimaalse pikkusega unikaalne bitijada.

PSK Phase Shift Keying — digitaalne faasmodulatsioon.

RRC Root-Raised Cosine — tõstetud ruutjuur-koosinusfilter.
Filter, mille sageduskoste servad on sagedustelje kohale tõstetud koosinuse veerandperioodi ruutjuured.

SDR Software-Defined Radio — tarkvaraline raadio.
1. Raadio, mille ehitus on peamiselt defineeritud tarkvaraga, jättes riistvara hooleks vaid sagedusnihke,
2. sellise raadio riistvaraline komponent.

SNR Signal-to-Noise Ratio — signaali/müra suhe.

USRP Universal Software Radio Peripheral — Ettus Researchi toodetav **SDR** riistvara.

1 Motivatsioon

Eesti Tudengisatelliidi programmi eesmärgiks on nanosatelliitide arendamise kaudu edasi viia nii Eesti inseneriharidust taset kui kohalikku võimekust kosmosetehnoloogia hangetest osavõtmiseks. Tähtis osa sellest on satelliidi opereerimine kosmoses. Selleks on vaja töökindlat ning piisava läbilaskevõimega sidet satelliidi ja maajaama vahel. Käesolev töö keskendub maapealse vastuvõtuvõimekuse arendamisele tarkvaralise raadio (SDR[1]) abil, rõhuga ESTCube-2 (EC-2) C-riba[2] saatja vastuvõtmisel.

1.1 Tulevased missioonid

ESTCube-3 (EC-3[3]) eesmärgiks on Kuu orbiidil testida elektrilist päikesepurje. EC-2 on ettevalmistav missioon, mille ülesandeks testida ja valideerida kasutatav tehnoloogia ja tehnika EC-3-l kasutamiseks. Satelliidi pardale on kavas paigutada maavaatluskaamera (Earth Observation Payload), millega kaasneb vajadus suurte sidekiiruste järgi.

1.2 Hetkeseis

EC-2 peamine sidesüsteem kasutab ka ESTCube-1 (EC-1[4]) sides rakendatud sagedusmoduleeritud kanali ning amatöör-raadioside protokolli AX-25 kombinatsiooni. Selle vastuvõtmiseks on olemas nii riistvaralised kui tarkvaralised lahendused. EC-1 puhul olid side kiirusteks 9600 b/s maale ning 1200 b/s satelliidile. EC-2 puhul on ette nähtud sidekiiruste 38,4 kb/s-ni tõstmine.

Lisaks sellele on EC-2 pardal Ventspilsi Ülikooli tudengite ehitatud C-riba saatja, mis kasutab diskreetset kahendfaasmodulatsiooni (BPSK[5]). Kasutatav protokoll on hetkel määramata, sidekiirused on plaanitud Mb/s suurusjärku. Selle vastuvõtmiseks on plaanitud kasutada SDR-il põhinevat raadiot (käesolev töö). See võimaldab vastuvõtjat kergesti lõplikule saatjale kohandada ning kasutab ära maajaama juba olemasolevat riistvara.

1.3 Ristkasutus

Lisaks paindlikusele on SDR eeliseks portatiivsus. EC-2 vastuvõttu on kavas testida ka Irbene Raadioastronoomia Keskuse seadmetega, et valmistuda EC-3 kuu–missiooni sideks. SDR kasutamine võimaldab vältida sõltuvust riistvarast – muudatused vastuvõtjas on võimalik digitaalselt kohale toimetada.

[1] Software Defined Radio
[2] 4–8 GHz
[3] ESTCube-3
[4] ESTCube-1
[5] Binary Phase Shift Keying

Sama vastuvõtjaprogrammi saavad minimaalsete riistvarast sõltuvate modifikatsioonidega kasutada ka raadioamatöörid. Kuna satelliit kasutab amatöör–raadio sagedusi, siis peavad side parameetrid olema avalikult kättesaadavad. EC-1 kogemus näitab ka, et võimalus raadioamatööride vastuvõetud andmed otse missioonijuhtimis-süsteemiga (MCS[6]) ühendada on kasulik meetod vajadusel satelliidi vastuvõtuala suurendamiseks. [37]

[6]Missioonijuhtimissüsteem

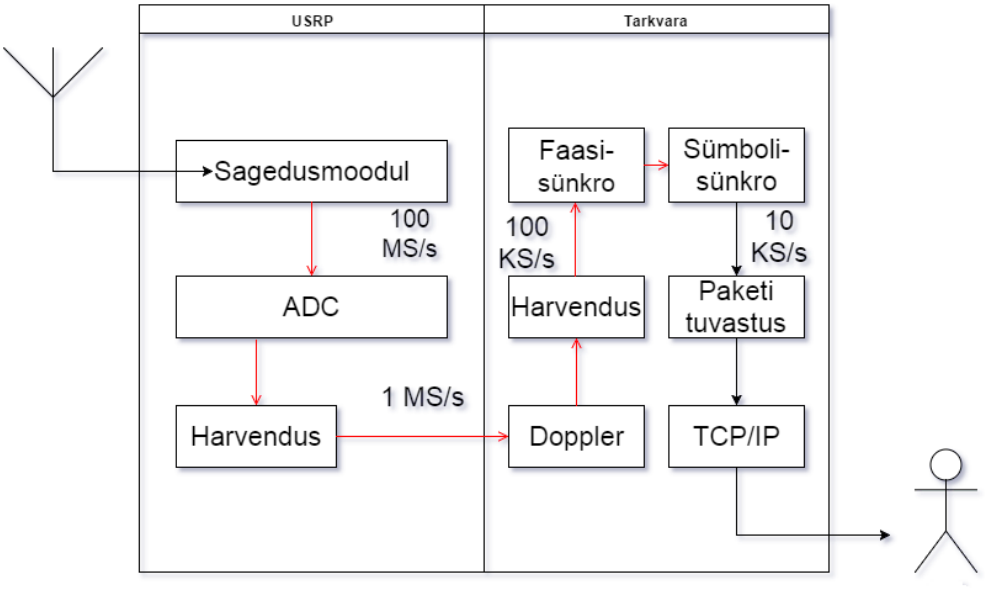
2 Ülevaade

2.1 Tarkvaraline raadio

IEEE[7] defineerib tarkvaralise raadio järgnevalt:

Tarkvaraline raadio on raadio, milles osa või kõik füüsilise kihi funktsioonid on tarkvaraliselt defineeritud — selle operatiivfunktsioonid (kuid mitte tingimata fikseeritud funktsioonide valimine) on tarkvaraliselt muudetav.[1, lk 4,6]

Käesolevas töös on riistvara osaks pääsuriba signaali vahesagedusele teisendamine ning diskreetimine. Tarkvaras toimub lõplik sageduskorrektsioon, demodulatsioon ning paketituvastus (Joonis 1).



Joonis 1: Tarkvaralise raadio plokkskeem

2.2 I/Q[8] esitus ja selle tarvilikkus

Olgu kandevsagedus, millele andmeid moduleeritakse, kirjeldatud kui

$$s(t) = A_0 \sin(\omega_0 \cdot t + \phi_0), \quad A_0, \omega_0, \phi_0 = const, \tag{2.1}$$

kus

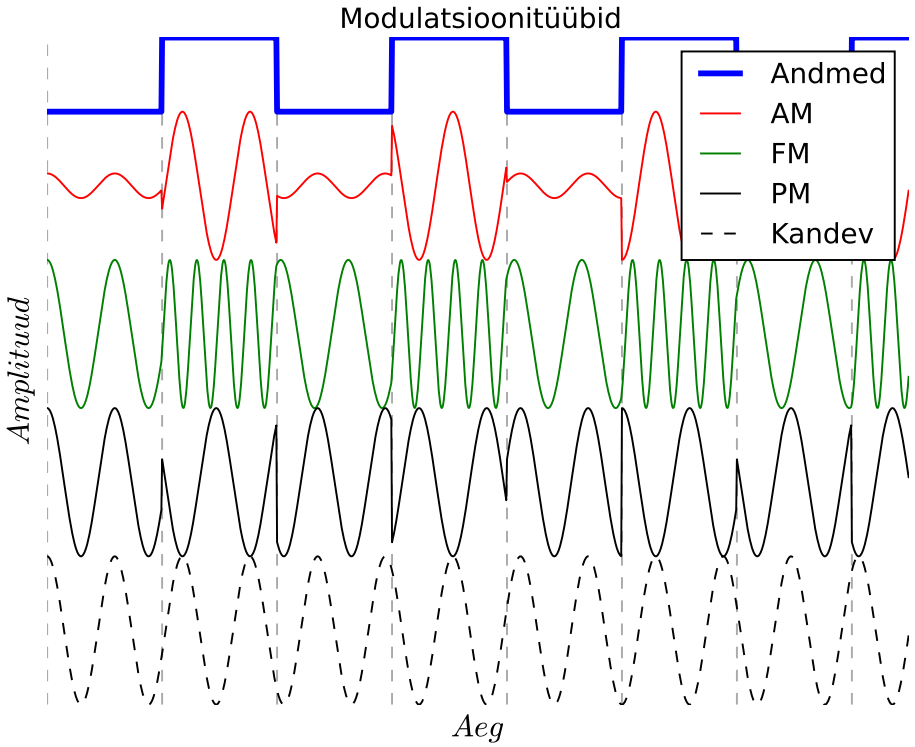
[7] Institute of Electrical and Electronics Engineers
[8] In-Phase/Quadrature

A_0 on amplituud,

ω_0 nurksagedus,

ϕ_0 konstantne faasinihe.

Andmeid saab kandevsagedusele moduleerida muutes amplituudi, sagedust või faasi (Joonis 2). Sagedusmodulatsiooni võib käsitleda kui faasmodulatsiooni erijuhtu, kuna nurksagedus ω on faasi ϕ ajaline tuletis.



Joonis 2: Näited faas-, sagedus-, ja amplituudmodulatsioonist

Saatjas on lihtne jälgida, mis väärtusega mingi konkreetne parameeter parajasti on: saatja defineerib need oma tööga. Vastuvõtjas puudub suvalisel ajahetkel piisav info mõlema parameetri määramiseks kohaliku referentssignaali suhtes, isegi kui teada on nii hetkenivoo kui selle tuletis.

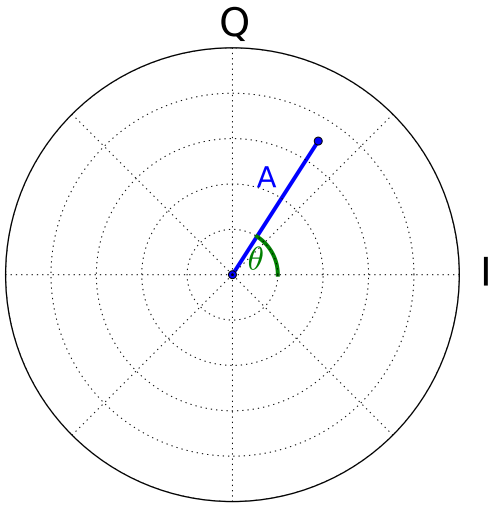
Olgu $a = A\sin(\phi)$ signaali nivoo hetkel t ja $b = \frac{da}{dt} = A\cos(\phi)\omega$ selle ajaline tuletis, siis

$$\begin{cases} a = A\sin(\phi) \\ b = A\cos(\phi)\omega \end{cases} \tag{2.2}$$

On näha, et süsteem ei ole üheselt lahenduv: tundmatuid väärtusi on rohkem kui võrrandeid. Diskreetsel juhul, asendades tuletise kahe võendi vahega, kehtib sama ebamäärasus.

Komplekskuju, mille reaalosa moodustab signaal ise, ning imaginaarosa selle signaali Hilberti teisendus, nimetatakse analüütiliseks signaaliks. Sellisel signaalil on igal ajahetkel defineeritud nii amplituud kui faas, ning faasi tuletis määrab hetksageduse. Lisaks on signaali kompleksosa iga sageduskomponent reaalosa vastava komponendi $\frac{\pi}{2}$ faasis pööratud variant.[34]

Sellist signaali ajahetkel t võib kujutada komplekstasandil (Joonis 3).



Joonis 3: Signaali võendi esitus komplekstasandil: A on signaali amplituud, θ faas.

Võendi projektsioon reaalteljele on signaali nivoo ajas. Seda nimetatakse I-komponendiks (In-phase). Projektsioon imaginaarteljele on Q-komponent (Quadrature).

Faas on defineeritud kohalikult genereeritud referentssignaali suhtes, mida kirjeldab (vrdl. (2.1)):

$$c_I(t) = A_c \sin(\omega_c t). \tag{2.3}$$

Kuna signaal on genereeritud vastuvõtjas, võib faasinihke valida 0-ks.

Vastuvõetud signaali korrutis selle signaaliga annab

$$\begin{aligned} s(t)c_I(t) &= A(t)A_c \sin(\omega t + \phi(t)) \sin(\omega_c t) = \\ &= A(t)A_c(\cos(\omega_t + \phi(t) - \omega t) - \cos(\omega_c t + \phi(t) + \omega t). \end{aligned}$$

Kuna kandevsagedused on võrdsed, siis

$$s(t)c_I(t) = A(t)A_c(\cos(\phi(t)) - \cos(2\omega_c t + \phi(t)). \tag{2.4}$$

Filtreerides välja kõrgsagedusliku komponendi, jääb alles baasriba I-komponent

$$I(t) = A(t)A_c(\cos(\phi(t))). \tag{2.5}$$

Q-komponendi saamiseks tuleb tähele panna, et imaginaartelg on referentsi suhtes $\frac{\pi}{2}$ faasinihkega, seega Q-komponendi saamiseks tuleb korrutada signaal referentsi samavõrra nihutatud koopiaga. Kuna referents on siinussignaal, siis Q-komponendi saamiseks on vaja signaal korrutada koosinussignaaliga:

$$\begin{aligned} c_Q(t) &= A_c \cos(\omega_c \cdot t), \\ s(t)c_Q(t) &= A(t)A_c \sin(\omega t + \phi(t)) \sin(\omega_c t + \pi/2) = \\ &= A(t)A_c(\cos(\omega_t + \phi(t) - \omega t - \pi/2) - \cos(\omega_c t + \phi(t) + \omega t + \pi/2)) = \\ &= A(t)A_c(\cos(\phi - \pi/2) - \cos(2\omega_c t + \phi(t) + \pi/2)) \end{aligned}$$

ja analoogselt (2.5)-ga:

$$Q(t) = A(t)A_c(\cos(\phi(t) - \pi/2)) = A(t)A_c \sin(\phi(t)) \tag{2.6}$$

Signaali komplekskujul esitamine defineerib signaali üheselt, mis lihtsustab edasist töötlust. Näiteks saab sagedusteisendustel vältida lisaks vahesagedusele ka summaarse sageduse teket. Samuti lihtsustub Costase faasituvastusahela töö (vt. sektsioon 3.1.2).

2.3 Riistvara

Riistvara ülesandeks on raadiosagedusliku signaali teisendamine arvutil töötlemiseks sobivasse sagedusvahemikku ning diskreetimine, arvestades Nyquist-Shannoni diskreetimisteoreemi piirangut[14, ptk 2.4].

Levinud SDR tava-arvutiga ühendatavad riistvara lahendused võib jagada kolmeks:

1. Seadmed, mis teostavad sagedusnihke, ning väljastavad signaali analoogkujul arvuti helikaardi sisendisse.

2. Seadmed, mis teostavad analoogtehnoloogia abil sagedusnihke ning diskreedivad signaali, väljastades signaali digitaalkujul arvutile USB- või võrguühenduse kaudu

3. Seadmed, mis diskreedivad otse raadiosagedusel, ning teostavad sagedusnihke digitaalse signaalitöötluse (DSP[9]) abil, enne signaali arvutile edasi andmist.

[9]Digital Signal Processing

Esimesse kategooriasse, mille eelisteks on lihtsus ja odavus, kuuluvad näiteks SoftRock[44] ja FlexRadio SDR-1000[17]. Arvuti helikaardi võib sellisel juhul lugeda raadio osaks, ning seade ja helikaart kokku moodustavad tinglikult teise kategooria seadme. Piiravateks teguriteks on arvuti helikaardi diskreetimissagedus ja sageduskoste ning müra, ning kvaliteetne helikaart võib kogu süsteemi oluliselt kallimaks teha.[17] Käesoleva töö jaoks need ei sobi, kuna nende ribalaius seab ülempiiri maksimaalsele saavutatavale andmesidekiirusele.

Teise kategooriasse kuuluvad näiteks RTL-SDR[22], Funcube Dongle[25], Ettus Research USRP[10]-seeria[9] ja HackRF One[29]. Tegu on levinuima lahendusega, kuna võimaldab kasutada suuremaid ribalaiuseid kui esimese kategooria seadmed. RTL-SDR ja HackRF on mõlemad levinud hobivastuvõtjad, kuid nende 8-bitise dünaamilise ulatusega ADC-d seavad alampiiri saavutatavale signaal-müra suhtele nõrga signaali korral.

Kolmandas kategoorias on peamiselt lühilainealas ja madalamatel sagedustel töötavad raadiod, kuna disain eeldab töösagedusest suuremat analoog-digitaalmuunduri diskreetimissagedust. Näited selles kategoorias on FlexRadio SDR-6000 seeria[12], Microtelecom Perseus[27] ja ICOM IC-7300[16]. Samuti on see võimekus mõningatel teise kategooria seadmetel, näiteks USRP raadiod toetavad seda vastavate raadiosagedusmoodulite kasutamisel, ning RTL-SDR vastuvõtjaid saab otse-diskreetimiseks ümber ehitada[23]. Ehkki eksisteerib GHz-võimekusega analoog-digitaalmuundureid [38, 3], on need enamasti suunatud ultra-lairiba erirakendustele. Vähese leviku põhjuseks on raskused nii suure andmehulga aegsal töötlemisel.

Saatmisel (kui seade seda võimaldab) toimuvad kirjeldatud protsessid vastupidises järjekorras, ning kõik mainitud piirangud rakenduvad samaväärselt.

Käesolevaks tööks peab SDR riistvara vastama järgnevatele nõuetele:

- Ribalaius suurem kui 2 MHz
- Dünaamiline ulatus vähemalt 16 bitti
- 5.8 GHz vastuvõtu võimekus

2.3.1 USRP N210

Ettus Research USRP N210 on SDR, mis toetab kuni 50 MS/s 8-bitises, ning kuni 25 MS/s 16-bitises täis-dupleks režiimis. Seade ühendub arvutiga kasutades 1000BASE-T võrguühendust, mis seab ka piiri saavutatavale ribalaiusele.[8, 10]

N210 toetab erinevaid raadiosagedusmooduleid (*daughterboard*), mis võimaldavad ligipääsu erinevatele sagedusaladele vahemikus 0 – 6000 MHz.

ESTCube maajaamal Tartu Observatooriumis on olemas USRP N210 SDR seadmed erinevate sagedusmoodulitega, millest aga hetkel ükski ei toeta 5.8 GHz signaali

[10]Universal Software Radio Peripheral

vastuvõttu. Kuna seadmed vastavad ülejäänud nõuetele, on viimane tingimus rahuldatud välise sagedusmuunduriga, mis teisendab 5.8 GHz signaali 600 MHz peale. Ettus Research pakub ka sagedusmoodulit mis katab vajaliku sagedusala, võimaldades hiljem süsteem integreerida[7].

2.4 Tarkvara

SDR tarkvara tegeleb vastuvõtul riistvarast saabuvate I/Q võendite töötlemisega vastavalt rakendusele. Peamiselt võib jagada lahendused kaheks:

1. Signaalitöötlusraamistikud, mis pakuvad erinevaid funktsioone, mida programmaatiliselt kokku ühendades on võimalik defineerida uut funktsionaalsust;

2. Fikseeritud funktsioonidega rakendused, mis defineerivad teatud modulatsioonidemodulatsiooniskeemid või analüüsimeetodid, mille vahel võib olla võimalik valida.

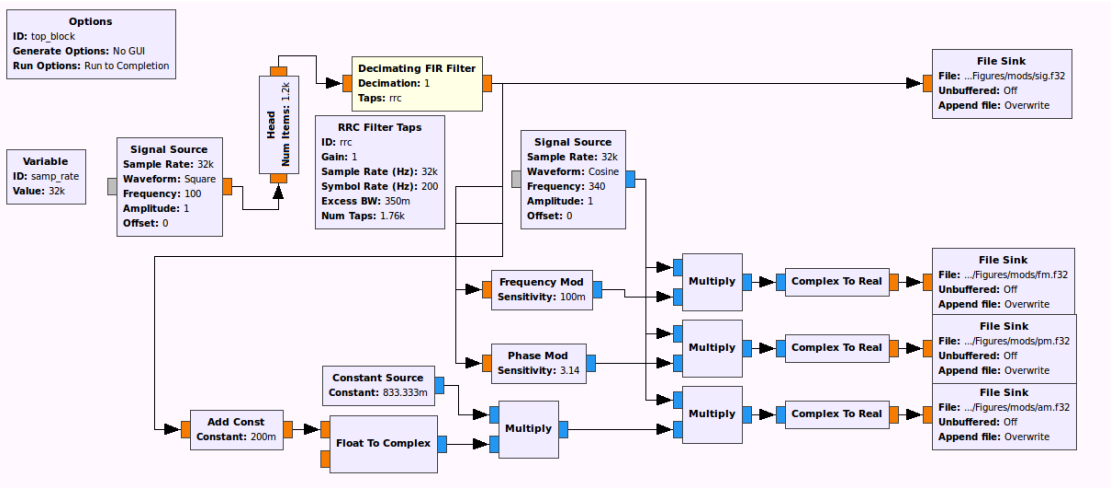
Esimesse kategooriasse kuuluvad näiteks GnuRadio [40], Simulink [42], LabView [28] ja Vivado [45]. Kõik kolm pakuvad võimalust signaalitöötlusrakenduste visuaalprogrammeerimiseks, lastes kasutajal ühendada signaalitöötlusplokke graafilises kasutajaliideses. Tegu on üldiste tööriistadega, mis kasutamiseks eeldavad tutvust signaalitöötlusega. Vivado on näide SDR disainist mitte tavaarvuti, vaid FPGA peal.

Teises kategoorias on näiteks GQRX [6], HDSDR [43], SDR Radio [35], SmartSDR [13], Linrad [4]. Enamasti on tegu SDR riistvaratootjate või raadioamatööride poolt välja antud programmidega, mis on suunatud amatöörraadio režiimidel kasutamiseks.

2.4.1 GnuRadio

GnuRadio on vabavaraline signaalitöötlusraamistik, mille põhistruktuur on voodiagramm (ingl. k. *flowgraph*): varem programmeeritud töötlusplokid ühendatakse omavahel järjestikku. Plokid on kirjutatud C++ programmeerimiskeeles ning neid saab omavahel siduda nii C++, Pythonit kui graafilist programmeerimist kasutades. Gnuradio-companion (Joonis 4), mis esitab plokid graafilisel kujul, võimaldab neid omavahel siduda, kontrollib ühenduste õigsust andmetüübi järgi, ning genereerib selle põhjal iseseisva Pythoni programmi.

Uute moodulite ja plokkide kirjutamise lihtsustamiseks tuleb programmina kaasa *gr_modtool*, mis etteantud parameetrite (ploki tüüp, sisendid, väljundid, parameetrite arv) genereerib raamistiku, kuhu kuuluvad CMake-l[19] põhinev projektigenereerimissüsteem, Pythoni ja C++ ühiktestide raamistikud, liidesed plokkide Pythonist välja kutsumiseks ning XML failid, mida kasutatakse gnuradio-companionis plokkide esituse defineerimiseks.



Joonis 4: gnuradio-companion näide: jooniste 2 ja 5 andmete generaator

GnuRadio eelisteks käesoleva projekti raames on avatus, mitmekülgne riistvara tugi ning nendest tulenev laiapõhjaline kasutajabaas nii raadioamatööride seas, akadeemias kui tööstuses. Peamisteks puudusteks on kiireltarenevatele avatud tarkvara projektidele omased pinnapealne dokumentatsioon ning kohati lünklik baasfunktsionaalsus.

2.5 Kosmoseside

Kuupsatelliidid kasutavad sideks enamasti FSK[11] modulatsiooni ning AX.25 kadreerimist[20]. AX.25 on amatöörkasutuses levinud punktist-punkti pakettandmeside protokoll, ning integreeritud FSK raadioga mikrokontrollerid, enamasti disainitud nutistus kasutamiseks, on odavad ja levinud. Olulisel kohal on inerts: lahendus on korduvalt testitud, töötav, ning enamikul juhtudel piisav.

2.6 BPSK - digitaalne kahendfaasmodulatsioon

BPSK on modulatsioon, mille puhul andmetega moduleeritakse kandevsageduse faasinihet. Olgu andmed esitatud bipolaarkujul:

$$s_n = \begin{cases} 1, & \text{kui } d_n = 1, \\ -1, & \text{kui } d_n = 0, \end{cases} \tag{2.7}$$

kus d_n on n-is andmebitt, siis moduleerides muudetakse saadetava signaali faasi $s_n \frac{pi}{2}$ võrra.

[11] Frequency Shift Keying

BPSK puhul on probleemiks faasi ebamäärasus: Vastuvõtjas ei ole võimalik lisainfota määrata, kumb olek tähendab 0, kumb 1. Sellest üle saamiseks saadetakse paketi päises mingi tuntud bitijada ning võrreldakse seda vastuvõtul.

Eksisteerib alternatiivne kuju: diferentsiaalne BPSK (DBPSK), kus andmed on moduleeritud mitte faasi absoluutse nihkena mingist referentsist, vaid faasi nihkena võrreldes eelmise sümboliga.

Lisaks andmete ühesele esitusele on DBPSK eeliseks BPSK ees see, et demoduleerimiseks ei ole vajalik vastuvõtjas rekonstrueerida absoluutset faasireferentsi – piisab töödeldava sümboli eelmisega võrdlemisest. Seega puudub vajadus kandevlaine faasilukustamise järgi. See võimaldab lihtsustada vastuvõtja konstruktsiooni, ning vältida lisaks sünkroniseerimise ajakulu paketi alguses.

Peamiseks puuduseks on 1-2 db suurem vajalik signaal-müra suhe antud bitivigade tõenäosuse saavutamiseks [32, ptk 5.2.8].

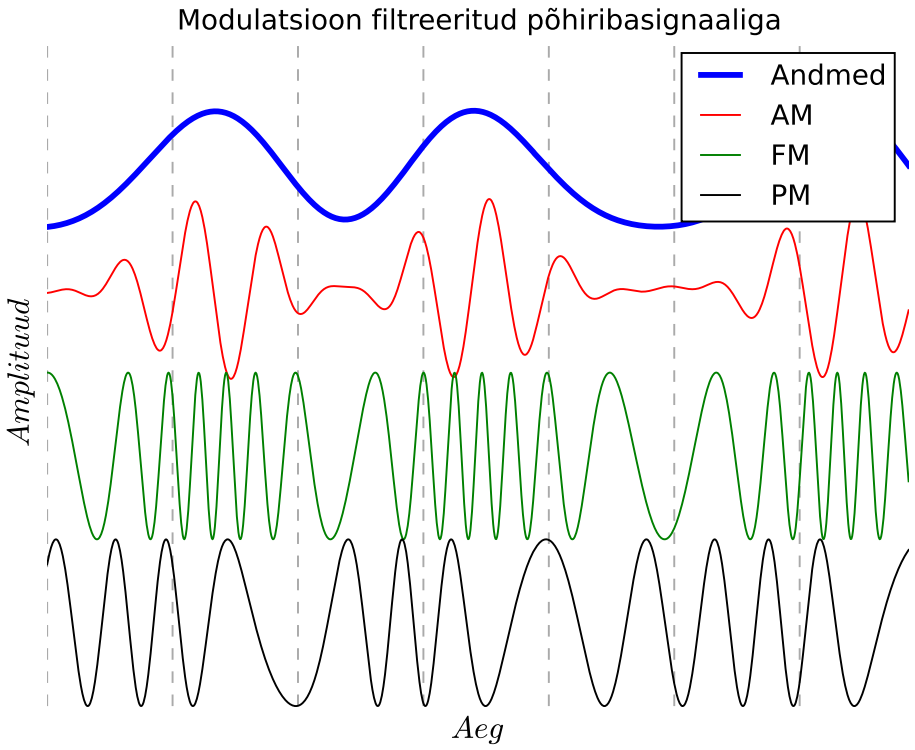
2.7 Filtreerimine

Kirjeldatud modulatsiooni korral toimub faasi muutus väga lühikese aja jooksul, mistõttu signaali sagedusesitus on lai. See on vastuolus raadiospektri kasutamise põhimõttega, mille järgi teisi spektri kasutajaid tuleb võimalikult vähe segada. Lugedes töösageduse läheduses müra spektraalse tiheduse konstantseks, võib lisaks täheldada, et vastuvõtja peab tegema kompromissi signaali energia püüdmise ning müra vältimise vahel. Seetõttu filtreeritakse saatjas signaale.

Kuna raadiosagedusel filtreerimine on keerukas (filtri suhteline pääsuriba läheb väga väikseks, mis viitab väga suurele kvaliteeditegurile), eelistatakse põhiribafiltreerimist ehk impulsi kujundamist (ingl. k. *pulse shaping*): filtrit rakendatakse moduleerivale signaalile. Tulemus on kujutatud joonisel 5

Selliselt filtreeritud signaali tuvastamiseks vastuvõtjas, AWGN[13] kanali puhul, on efektiivseim filter, mille impulsskoste on saatjas kasutatud filtri ajas pööratud versioon. Sellist filtrit nimetatakse sobitatud filtriks (ingl. k. *matched filter*. Tulemuseks on vastuvõetud signaali korrelatsioon saadetud impulsi kujuga. [32, ptk 5.1.2]

[13] Additive White Gaussian Noise



Joonis 5: RRC[12]-filtreeritud signaal, liiasus $\alpha = 0.35$

3 Vastuvõtja disain

Tarkvaras on võimalik eristada nelja osa:

1. sageduskorrektsioon ja harvendus,
2. faasisünkronisatsioon,
3. demodulatsioon,
4. paketituvastus.

3.1 Sageduskorrektsioon

Sageduskorrektsioon koosneb satelliidi liikumise tõttu tekkinud Doppleri nihke korrektsioonist ning saatja ja vastuvõtja mitteideaalsuse kompenseerimiseks vajalikust sagedussünkronisatsioonist.

3.1.1 Doppleri nihke kompensatsioon

Nagu helile, rakendub ka raadiolainetele Doppleri nihe: lähenev saatja näib saatvat kõrgemal, kaugenev madalamal sagedusel, kui saatja tegelik sagedus. Satelliitsides võib selline nihe ulatuda olenevalt sagedusest ning orbiidist mitmesaja kHz-ni, kusjuures sageduse muutus võib olla 10 kHz/s suurusjärgus [21].

Seetõttu on vajalik kas saatjas või vastuvõtjas Doppleri nihe kompenseerida. Mõlemal juhul on korrigeerival poolel vaja teada või oletada osapoolte suhtelist liikumiskiirust. Lihtsustamaks satelliidi disaini ning mitme maajaama kasutamist, teostatakse käesolevas töös korrektsioon maajaamas.

Satelliitside hõlbustamiseks eksisteerib tarkvarapakett GPredict [5]. Programm suudab kontrollida nii antennikeerajaid kui amatöörraadioid HamLib[11] liideseteekide abil. EC-1 kogemus näitas, et ehkki GPredicti genereeritud andmed on usaldusväärsed, on programm ise järelvalveta kasutamiseks liiga ebastabiilne.

GPredict kasutab satelliitide positsioonide jälgimiseks C-s kirjutatud teeki sgp4sdp4 [36]. See evitab Põhja-Ameerika Õhukaitse Peastaabi (NORAD) poolt avaldatud algoritmid [15] SGP4 ja SDP4 (vastavalt maalähedaste ja kaugemate satelliitide jaoks), mis töötavad NORADi poolt avaldatud jälgimisandmetega.

Käesoleva töö raames valmis GnuRadio plokk, mis kutsub otse selle teegi funktsioone, tagamaks GPredictist sõltumatu Doppleri nihke korrigeerimine.

Üks olulisi eesmärke oli saavutada sageduse ja faasi pidevus. Varasem kogemus Costase faasisünkronisatsiooni plokiga on näidanud, et järsud sageduse muutused on peamine faasipöörete põhjus.

Plokk oskab lisaks süsteemiaja järgi töötamisele ära kasutada USRP võimekust lisada esimesele vastuvõetud signaali I/Q võendile vastuvõtuaja lipp. Seega hoolitseb USRP

aja monotoonsuse tagamise eest. Ühtlasi võimaldab see võendi täpsusega reageerida kadunud pakettidele või ajutiselt peatatud tööle.

Süsteemiaja järgi töötades ei ole efektiivne ega enamasti võimalik samasugust reageerimisvõimet tagada: Süsteemiaja päring on võrreldes diskreetimissagedusega aeglane, ning tihti puudub platvormil piisava lahutusega aja allikas. Lisaks pole ploki lõim pidevalt aktiivne: GnuRadio plaanur koordineerib plokkide tööd vastavalt järgnevate plokkide nõudlusele ja eelnevate tootlikkusele.

Ploki tööprotsess on järgnev (vt joonis 6):

- Plokk saab arvu n võendeid, mida töödelda.
  - Kui nende seas on vastuvõtuaja lipuga võendeid, töödeldakse vaid sellest eelneva võendini.
  - Kui lippe pole, töödeldakse kõik võendid, mille jaoks on olemas ette valmistatud sagedusi.
- Kui esimesel võendil on lipp või ette valmistatud sagedused saavad otsa, genereeritakse uus sagedustenimekiri.
  - Kui põhjuseks on ettevalmistatud sageduste otsa saamine, päritakse sgpsdp teegilt 1 sekund hilisema aja kohta doppleri nihke hinnang. Algpunktina kasutatakse eelmise jada lõppsagedust.
  - Kui põhjuseks on lipp, päritakse hinnangud nii alguse kui lõpu jaoks sgpsdp teegilt.
  - Nende kahe punkti vahel interpoleeritakse lineaarselt diskreetimissageduse järgi hetksagedused.
- GnuRadio plaanurile antakse teada, mitu võendit töödeldi.

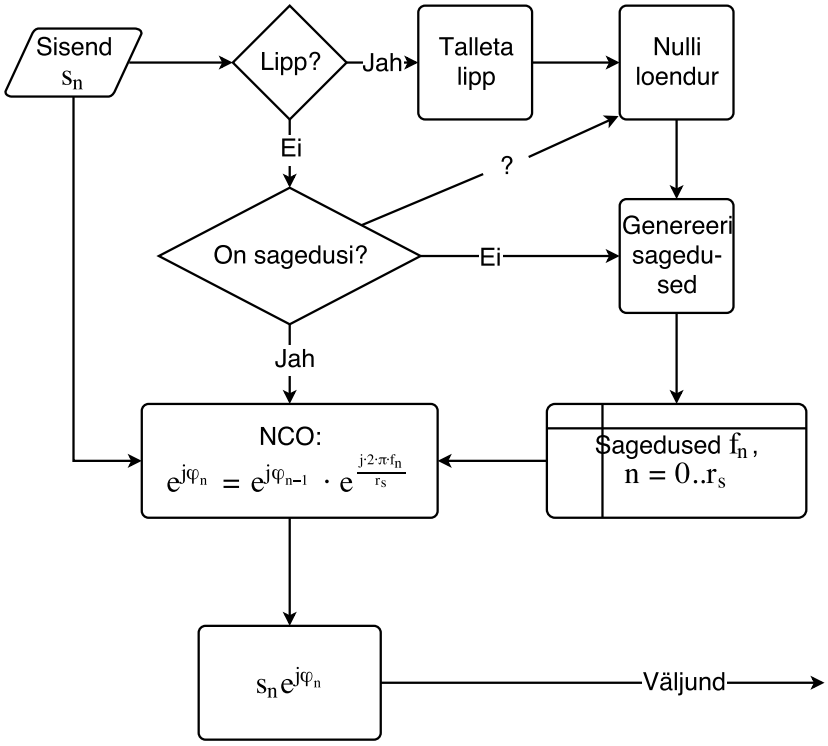
Koodiga on võimalik tutvuda lisas 6.2 ning veebilehel [18, /blocks/gr-doppler/]

3.1.2 Sagedussünkronisatsioon

Kuna ebatäiused tootmises, keskkonna mõjutused ning vananemine võivad saatja või vastuvõtja sagedusstandardi täpsust mõjutada, on raadiotes harilikult kasutusel sagedussünkronisatsioon.

Kuna antud juhul on saatja alati sama ning hästi karakteriseeritud, ning kuna faasikorrektsioon (vt. järgmine sektsioon) suudab +- 2 kHz piires seda ülesannet iseseisvalt täita, ei ole praeguse seisuga vajalik vastuvõtjas lisakeerukust tekitada.

Eksperimenteeriti mõningate GnuRadio standardteegi lahendustega:



Joonis 6: Doppleri ploki töö. s_n - sisendvõendid, $e^{j\phi}$ - sisemise ostsillaatori olek, f_n doppleri nihke hinnang, r_s - diskreetimissagedus

.

- *FLL Band Edge* kasutab sobitatud filtrist tuletatud kahte filtrit, mis asuvad signaali spektri servadel. Nende diferentsiaalne korrelatsioon vastuvõetud signaaliga on tagasisideahela veasignaaliks. Puuduseks on ploki fikseeritus RRC filtritega kasutamiseks. Lisaks raskendab antud töös selle ploki kasutamist asjaolu, et testseade, mille vastuvõttu katsetati, vahetab perioodiliselt nullfaasi: Kui andmed on kodeeritud üheselt $\{0, 1\} \mapsto \{\frac{\pi}{2}, -\frac{\pi}{2}\}$, siis üleminekud võivad olla nii läbi 0 kui π. See muudab sagedusesituse kuju, tehes spektri servadel põhineva lahenduse ebasobivaks. Lisapiirang ploki tööks on see, et lukustumiseks peab signaal sisaldama statistiliselt enam-vähem juhuslikke andmeid: plokk ei lukustu puhtale kandevlainele, küll aga suudab järgida aktiivset kanalit.

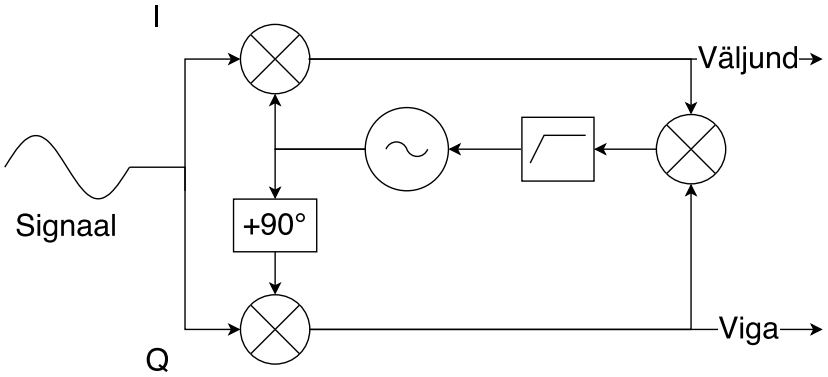
- *PLL Carrier Tracking* lukustub kiiresti kandevsagedusele, ent ei sobi aktiivse kanali järgimiseks. Selle ploki kasutamiseks on vaja tagasisidet ülejäänud vastuvõtjalt, mis sageduse paketi dekodeerimise ajaks lukustaks.

3.2 Faasisünkronisatsioon

PSK[14] vastuvõtmiseks on vaja signaali faasi mingi referentsiga võrrelda. Kuna vastuvõtjal puudub originaalreferentsile ligipääs, tuleb see koha peal genereerida.

Costase ahel on analoog-raadioskeemidest pärit faasisükronisatsiooniahel [32, lk 355-356], mille digitaalne signaalitöötlus on lihtsamaks teinud[39] (vt. joonis 7):

Olgu ω nurksageduse hinnang ning ϕ kohaliku ostsillaatori hetkefaas. Plokki töötlemiseks saabuv võend s korrutatakse kõigepealt $e^{-j\phi}$-ga. Sellega on s-le rakendatud sagedusparand ning saab arvutada sageduse veahinnangu. BPSK faasi veasignaal on $\epsilon = \Re(s) \cdot \Im(s)$. Veasignaal madalpääs-filtreeritakse ning liidetakse ω-le, mille alusel uuendatakse kohaliku ostsillaatori faasi ϕ. ϵ märk on alati selline, et kui liita see ω-le, rakendatakse järgmisele võendile vastavalt vajadusele suuremat või väiksemat sagedusnihet, et parandi tulemusena oleks võend reaalteljel.



Joonis 7: Costase ahel

Skeemi eeliseks on, et faasinihe π võrra ei põhjusta lukustuse kaotust. See teeb Costase ahela väga sobivaks BPSK sünkroniseerimiseks: korrigeerimine toimub pidevalt.

Töös on madalpääsufiltri laius on määratud eksperimentaalselt piisavalt kitsas, et vältida genereeritud referentssignaali faasi pöördumist π võrra müra tõttu. See määrab ka alampiiri signaal-müra suhtele, millel vastuvõtja on võimeline töötama (vt. sektsioon 4.1).

3.3 Sümbolitakti sünkroniseerimine

Lisaks kandevlaine sageduse- ja faasisünkronisatsioonile on vajalik kindlaks määrata samad parameetrid sümbolitakti jaoks. Sümbolite ajastus on määratud saatva mikrokontrolleri poolt, ning sagedusstandard mis selle määrab ei ole üldjuhul sünkroonis vastuvõtva seadme sagedusega.

[14] Phase Shift Keying

Sümbolitakti tuvastamiseks on võimalik ära kasutada saatjas kasutatud põhiribafiltri omadusi. Olgu meil kasutusel sobitatud filter (vt. ptk 2.7). Hetked, mil korrelatsioon filtri ning vastuvõetud signaali vahel on suurim, ongi võendi võtmiseks optimaalsed.[14, ptk. 13.3].

GnuRadio standardteekides on polüfaas-sümbolisünkronisaator (*Polyphase Clock Sync*). See võimaldab teostada ühe blokiga nii filtreerimise kui sünkroniseerimise. Plokk vajab parameetrina kõrgemat järku filtrit, kui diskreetimissagedus ette näeks: olgu osafiltrite arv n = 32, sümbolitakt 10 korda väiksem diskreetimisagedusest ning filtri rühmaviide 7 sümbolit, siis filtri järk peab olema $32 \cdot 7 \cdot 10$[14, ptk 13.3.2][41].

See filter jagatakse kaardipaki laadselt laiali n osafiltriks. Võib tähele panna, et filtrid on üksteise suhtes $\frac{2\pi}{n}$ võrra faasinihkes, ning katavad kokku kogu võimaliku faaside vahemiku sama suure sammuga.

Lisaks genereeritakse teine osafiltrite pakk, mille koefitsiendid on esimese paki kõrvutiolevate osafiltrite koefitsientide vahed.

Sissetulevale signaalile rakendatakse korraga ühte osafiltrit põhipakist ja sellega seotud osafiltrit vahepakist. Põhipaki filter leiab korrelatsiooni, vahepaki filter aga suuna, kuhu poole peab korrelatsiooni tõstmiseks liikuma. Kui plokk on saavutanud korrelatsiooni maksimumi, jääb selle olek kahe kõrvutiasetseva osafiltri vahel vahetuma. `n`-nda osafiltri rakendamine interpoleerib sümboli maksimumi (mis muidu võis olla kahe võendi vahel) kindla võendi peale, misjärel see võend väljastatakse kui parim sümboli väärtuse hinnang.[14, ptk13.3.2]

Pärast seda eraldatakse komplekssest sümbolist reaalosa ujukomaarvuna, teisendatakse märgi järgi üheks või nulliks, ning saadetakse bitijada paketituvastusse.

3.4 Sõnepõhine faasi- ja sümbolisünkronisatsioon

Nii *Costas loop* kui *PFB Clock Sync* tunnistavad voolippusid (vastavalt `phase_est` ja `time_est`), mille väärtuste järgi korrigeerivad oma hetkeseisu. Nende lippude genereerimiseks on standardteegis plokk *Correlation Estimator*. Plokile on võimalik ette anda sünkrosõna moduleeritud kuju, mille leidmisel see hindab faasi- ja ajastusnihet. Algoritm on kirjeldatud [26].

Plokk märgib lippudena signaalivoos ära sünkrosõna alguse, lõpu ning faasi ja ajastuse. Faasi ja ajastuse lippude asukohta on võimalik valida. Arvestada tuleb aga, et kui signaal ei ole sagedussünkroniseeritud, siis on faasi hinnang korrektne vaid korrelatsiooni maksimumi kohal, seega on otstarbekas märkida hinnangud sõna lõppu. Sünkrosõna peaks lõppema sümboli keskel: faas on hästi defineeritud, ning sümbolitevaheline interferents ei mõjuta tulemust.

Antud töös prooviti sõnepõhist sünkroniseerimist lahendamaks BPSK faasi kahesuse probleemi. Lõplikus disainis on see lahendus kõrvale jäetud. Järgneb lühike kokkuvõte kasutuselevõtmiseks vajalikust ning mittekasutamise põhjustest.

3.4.1 Sünkrosõna valik

Sünkrosõna autokorrelatsioon peab olema võimalikult terava piigi ning kergete sabadega. Optimaalsed sellised jadad on Barkeri koodid, mille autokorrelatsioonid $c_n \leq 1$ kõikide nihete $n \neq 0$ korral. Kahjuks ei ole Barkeri koode, mis oleks pikemad kui 13 sümbolit. Eksperiment käesolevas töös näitas, et see ei ole piisav mürases signaalis valetuvastuste vältimiseks.

Seetõttu katsetati 24-bitist optimaalset koodi[24]. Probleemiks jäi siinkohal see, et sünkrosõna tuvastamine küllaldase usaldusväärsuse ning madala valetuvastuste arvuga nõudis vähemalt ligikaudset sagedussünkroniseeritust. Kuna faasisünkronisaator lukustus enamasti kiiremini kui sagedus-sünkronisaator, tekitas sünkrosõne abil leitud faasile lülitumine pigem faasimüra, ning pikendas sünkronisatsiooni stabiliseerumise aega.

Tasub tähelepanu juhtida asjaolule, et *PFB Clock Sync* plokis on teadaolev viga lippude töötlemisel. Kuna olemuselt ei ole plokk fikseeritud harvendusteguriga, ei ole lippude asukoha määramine väljundis triviaalne. Olemas on veaparandus [33], mis vähendab nihet. Seejärel saab lõpliku korrektsiooni teha *PFB Clock Sync* plokis kasutatava filtripaki nihutamisega. Joonisel 8 kujutatud filtri maksimumi nihe keskjoonelt on sellise korrektuuri tulemus.

Kuna BPSK faasipööre on (samuti sünkrosõna abil) lihtsalt tuvastatav ka pärast demoduleerimist ja dekodeerimist, ja seda harvendusteguri jagu väiksema arvutuskoormusega, ei olnud käesoleval juhul otstarbekas sellist faasi- ja ajastussünkroniseerimismeetodit kasutada. Võib arvata, et meetod omab rohkem väärtust kõrgemat järku modulatsioonimeetodite sünkroniseerimisel, mille puhul demoduleeritud kujul nihke tuvastamine pole triviaalne.

Lisaraskuseks käesolevas töös on juba sektsioonis 3.1.2 mainitud ADF4158 BPSK signaali eripära, et modulatsiooni faas sümbolite vahel ei ole ühene. Selline signaali konjugatsioon tingis vajaduse kasutada kaht korrelatsiooniplokki erinevate filtritega, mis tõstis arvutuskoormust veelgi.

3.5 Paketituvastus ja väljastamine

Kuna tegelik kasutusele tulev protokoll ei ole veel teada, on hetkel testimiseks kasutusele võetud GnuRadio standardteegis olev lihtne sünkrosõnaga paketituvastus. Sünkrosõnanan kasutatakse 34-bitist minimaalsete sabadega autokorrelatsiooniga bitijada[24].

Olles vastuvõetud bitijadas tuvastanud sünkrosõna, eeldab plokk, et järgmised 32 bitti kirjeldavad paketi andmete pikkust bittides. Pikkus on kirjeldatud kahe 16-bitise arvuna, ning kui need ei ole võrdsed, loetakse pakett vigaseks ja seda ei edastata.

Kui pikkus tuvastatakse edukalt, märgitakse järgmine bitt peale pikkuseväljasid lipuga `packet_len`, mille väärtuseks on paketi pikkus bittides.

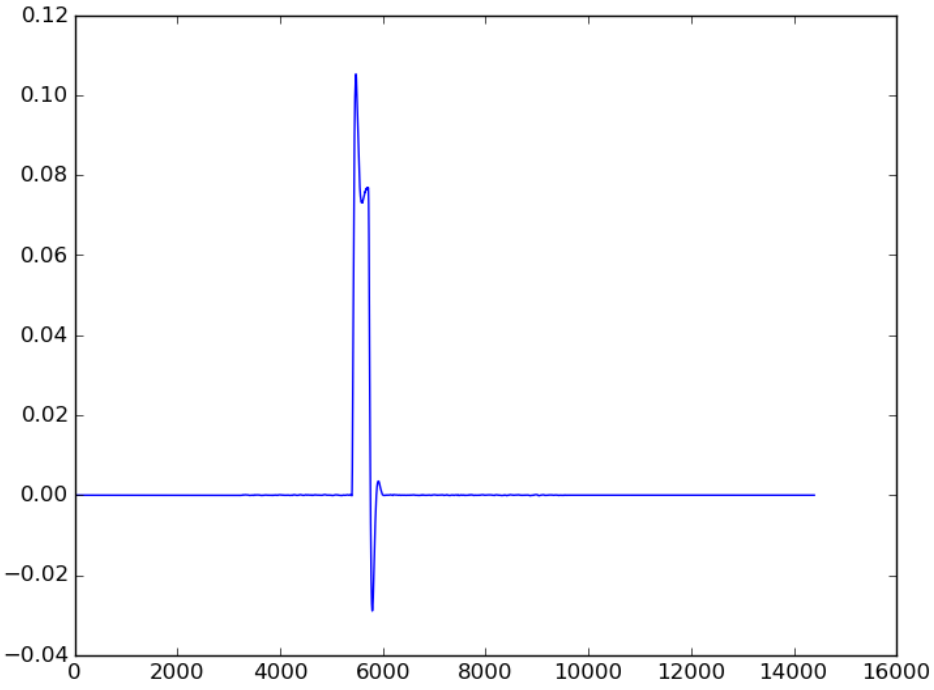
Tuvastamaks ka pööratud faasiga pakette, on selliseid plokke kaks, komplementaarsete sünkrosõnedega. Kui tuvastakse pööratud faasiga pakett, rakendatakse ülejäänud

paketile XOR bitiseisu vahetust.

Bitid pakitakse baitideks kokku standardteegi plokiga *Repack Bits*, mis oskab intelligentselt ümber käia ka paketi pikkust tähistava lipuga. Edasi saadetakse pakett üle võrgu (klient või server, UDP või TCP) töötlemiseks.

3.6 Filtrid

Kuna hetkel reaalset saateriistvara spetsifikatsiooni ei ole, ei saa selle alusel disainida ka süsteemile vajalikke sobitatud filtreid. Vastuvõtja testimiseks ning tundmatu saatja jaoks kohandamise metoodika väljatöötamiseks kasutatakse seetõttu olemasolevat ADF5148 sagedussüntesaatorit, mis suudab PSK otsesünteesida.



Joonis 8: Saatja ühikimpulss

Sagedussüntesaatorit käsitletakse kui nn „musta kasti”, kuhu saadetakse andmed, ning millest tuleb välja moduleeritud signaal. Sobitatud filtri välja töötamiseks on kõigepealt vaja määrata impulsskoste.

Selleks saadetakse süntesaatorisse 100 sümbolipikkust impulssi, ning salvestatakse väljundit. Tulemus sünkroniseeritakse käsitsi sageduses ning faasi sünkroniseerimiseks

kasutatakse väga madala ribalaiusega Costase ahelat. Seejärel jupitatakse salvestatud signaal üksikuteks impulssideks ning keskmistatakse, vabanemaks mürast. Selleks on koostatud komplekt Pythoni ja GnuRadio programme, mis on leitavad asukohas [18, /Pulses/].

Ikka veel komplekskujul tulemus teisendatakse faasiks. Kuna korrelatsioon signaaliga toimub vastuvõtuahela lõpus, pärast harvendusfiltrit, tuleb ka see tulemus samamoodi töödelda, tagamaks vastavus sümbolituvastuse hetkel. Tulemus konkreetse sagedussüntesaatori näitel on kujutatud joonisel 8.

Vajadusel diskreeditakse sobitatud filter sümbolituvastuse filtripaki jaoks sobivale diskreetimissagedusele (vt 3.3).

4 Testid

4.1 Bitivigade tiheduse hindamine

Vastuvõtja efektiivsust kirjeldab selle võime infot mürasest signaalist eraldada. Selle hindamiseks saadetakse vastuvõtjale teadaoleva sisuga andmepakett, ning võrreldakse demoduleeritud andmeid tegelikuga.

Testimise jaoks kasutatavad andmed peavad sisaldama esinduslikku valimit võimalikest saadetavatest bitijadadest. PSK puhul eriti olulised on jadad, mis sisaldavad järjestikku identseid sümboleid: vastuvõtja sümbolisünkronisatsioon kasutab ajastuse tuvastamiseks erinevate sümbolite vahelisi üleminekuid. Lahendus peab olema piisavalt täpne ja stabiilne, et selliste lõikude kohal ajastus säiliks.

Selliseks testimiseks on ITU-T[15] defineerinud soovituslikud jadad[2]. Jadad on defineeritud n-bitiste tagasisidestatud nihkeregistritega. Tagasiside on valitud selline, et väljundis oleks esindatud kõik võimalikud n-bitised jadad, peale ühe (maksimaalne järjestikuste nullide arv on $n-1$, kuna vastasel korral ei oleks registris enam midagi tagasi sidestada ning jada katkeks). Selline pseudojuhuslik väljund simuleerib küllalt hästi päris andmeid eeldusel, et sideprotokoll hoolitseb sünkroniseerimiseks piisava arvu bitimuutuste tagamise eest.

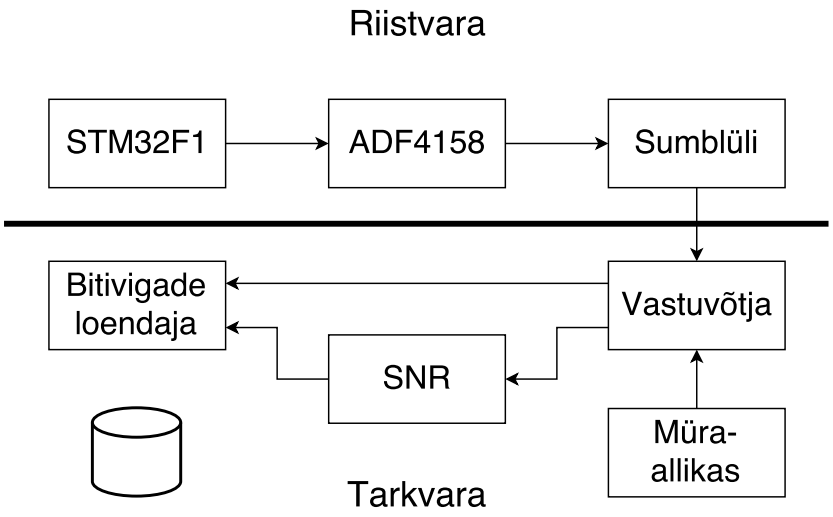
Vastuvõtja testimiseks genereeritakse 9-bitisele nihkeregistrile vastav selline jada (PRBS9[16]). Sellele lisatakse päises dubleeritult registri algväärtus, mis võimaldab paketi kao korral vastuvõtja taas-sünkroniseerida. Seejärel koostatakse sektsioonis 3.5 kirjeldatud pakett, mis sisaldab 256 baiti testandmeid. Pakett moduleeritakse ning signaal suunatakse muudetavasse sumblülisse, mille abil signaali tugevust vähendatakse süsteemi müratasemele lähedale.

Vastuvõtjas hinnatakse signaali-müra võimsuste suhet SNR[17] ning määratakse bitivigade arv ja tihedus. Info väljastatakse kord 10 sekundi tagant logisse.

Testpakettide koostamiseks kasutatakse STM32F1 arendusplaati, moduleerimiseks ADF4158 arendusplaati (vt. sektsioon 3.6), SNR määramiseks GnuRadio standardteegi plokki „MPSK SNR Estimator Probe" ning pakettide töötlemiseks ja bitivigade loendamiseks selleks otstarbeks kirjutatud plokki. Ülevaade katsest on antud joonisel 9.

Kuna reaalse signaalitugevuse piisavalt väikese sammuga muutmine sumblüli abil ei olnud võimalik, seati signaalitugevus konstantseks u. 12 dB SNR juures, ning vastuvõtutarkvaras lisati müra, kuni oli saavutatud järgmise mõõtmise jaoks sobiv signaali-müra suhe. Ülevaade katsest on antud joonisel 9. Testimisega seotud kood ja katseandmed on kättesaadavad [18, /Testing/BER]

[15] International Telecommunication Union – Telecommunication Standardization Sector
[16] Pseudo-Random Binary Sequence
[17] Signal-to-Noise Ratio



Joonis 9: Bitivea tiheduse testi ülevaade.

4.1.1 Signaal-müra suhte määramine

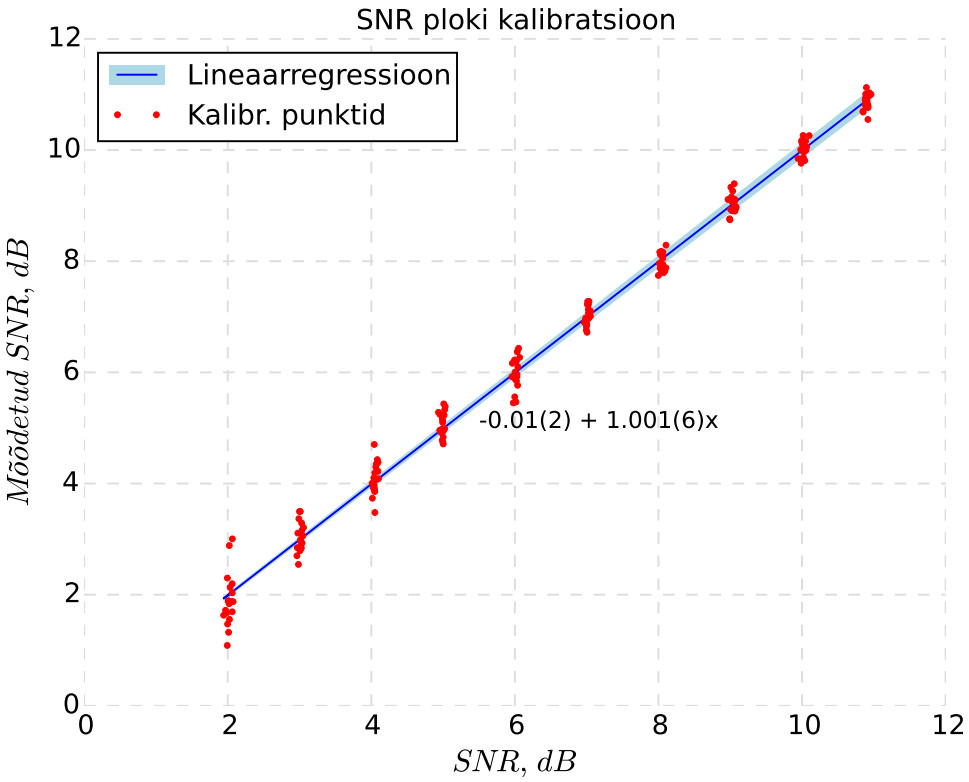
Signaali müratasemе määramiseks kasutatakse *MPSK SNR Estimator Probe* ploki algoritmi *2nd and 4th Moments*. See saadab regulaarselt sõnumeid SNR väärtusega bitivigade loendamise plokile, mis need kombineerib ja väljastab. SNR määramise algoritm on kirjeldatud [31]. Muuhulgas mainitakse vajadust teostada kalibratsioon juhul, kui vastuvõtja filter ei vasta Nyquisti filtri[14, ptk 4.2] kriteeriumitele.

Kalibreerimiseks genereeritakse GnuRadios simuleeritud müravaba saatja signaal, kasutades selleks teadaolevat saatja filtri impulsskostet, ning suunatakse see demodulaatori sisendisse. Lisaks genereeritakse Gaussi jaotusega valge müra, mis filtreeritakse kasutades sama filtrit, mida demoduleerimisel.

Signaali ja müra võimsused hinnatakse eraldi, ning arvutatakse signaal-müra suhe. Seda võrreldakse tulemusega SNR plokist, mille sisendis signaal ja müra liidetakse.

Kuna filter ei ole sümbolitevahelise interferentsi vaba, on vajalik hinnata signaali panust mürasse, kasutades sama hindamisalgoritmi, mis hiljem kasutuses müraga signaali puhul. Müra osakaaluks signaalist hindame $-27.8(1)dBc$ 95% usaldusnivool. See osa lahutatakse signaali võimsusest ning liidetakse müra võimsusele enne võimsuste suhte arvutamist.

Seejärel teostatakse kalibratsioonmõõtmised. Tulemused on esitatud joonisel 10.



Joonis 10: SNR ploki kalibratsioon.

4.1.2 Bitivigade loendamine

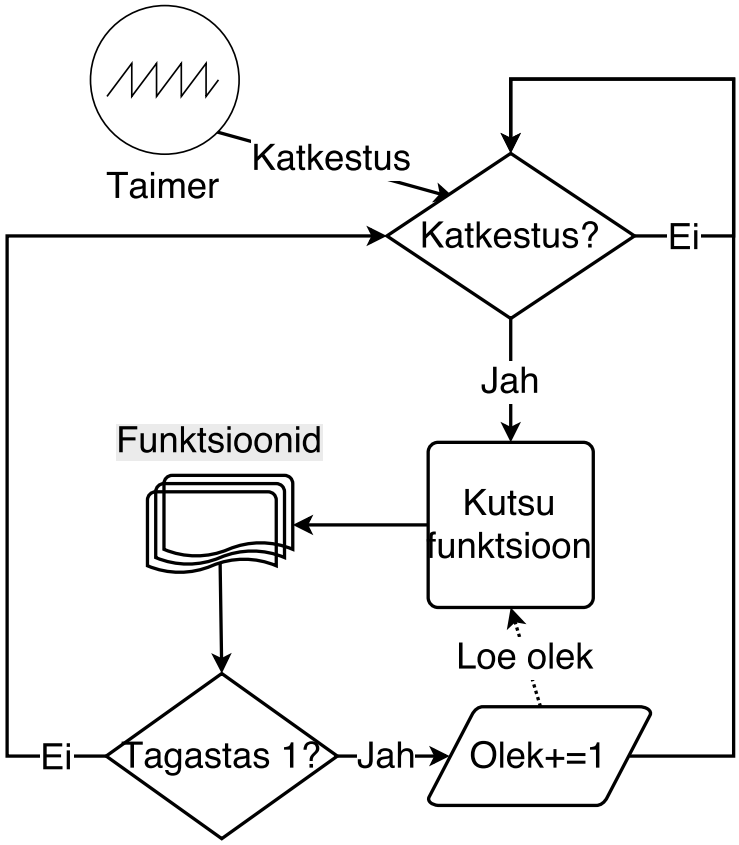
Testpakettide genereerimiseks on koostatud programm ST Microelectronics STM32F1 arendusplaadi jaoks. Paketi sisuks on juhuslik lõik PRBS9 jadast ja selle genereerimiseks vajalik nihkeregistri algseis. Programm kasutab bittide ajastamiseks riistvaralist taimerit, mis genereerib katkestuse. Katkestuskäsitleja on olekumasin, mis kutsub vastavalt paketi osale, mida parajasti saadetakse, välja alamkäsitleja. Alamkäsitleja teab, mis bitti antud osas saata on vaja, ning seab väljundi vastavalt. Kui tegu oli antud osa viimase bitiga, tagastatakse olekumasinale 1, mispeale too liigub järgmisesse olekusse(joonis 11).

Alamkäsitlejad on:

- faasi- ja sümboli-sünkrosõne (ei kasutata),
- paketi sünkrosõne,
- paketi päis (paketi pikkus),

- nihkeregistri algolek,
- genereeritud bitid.

Kood on leitav lisas 6.1.1 ning aadressil [18, /MCU].



Joonis 11: Testpakettide genereerija põhimõtteskeem.

Bitivigade loendamiseks on koostatud GnuRadio plokk, mis loeb paketist nihkeregistri väärtuse ning genereerib andmete kohaliku koopia, millega vastuvõetut võrreldakse. Kõigi pakettide peale loetakse kokku vastuvõetud ning vigaste bittide arvud, mille järgi arvutatakse bitivigade keskmine tihedus. Tulemused väljastatakse regulaarselt ekraanilogisse.

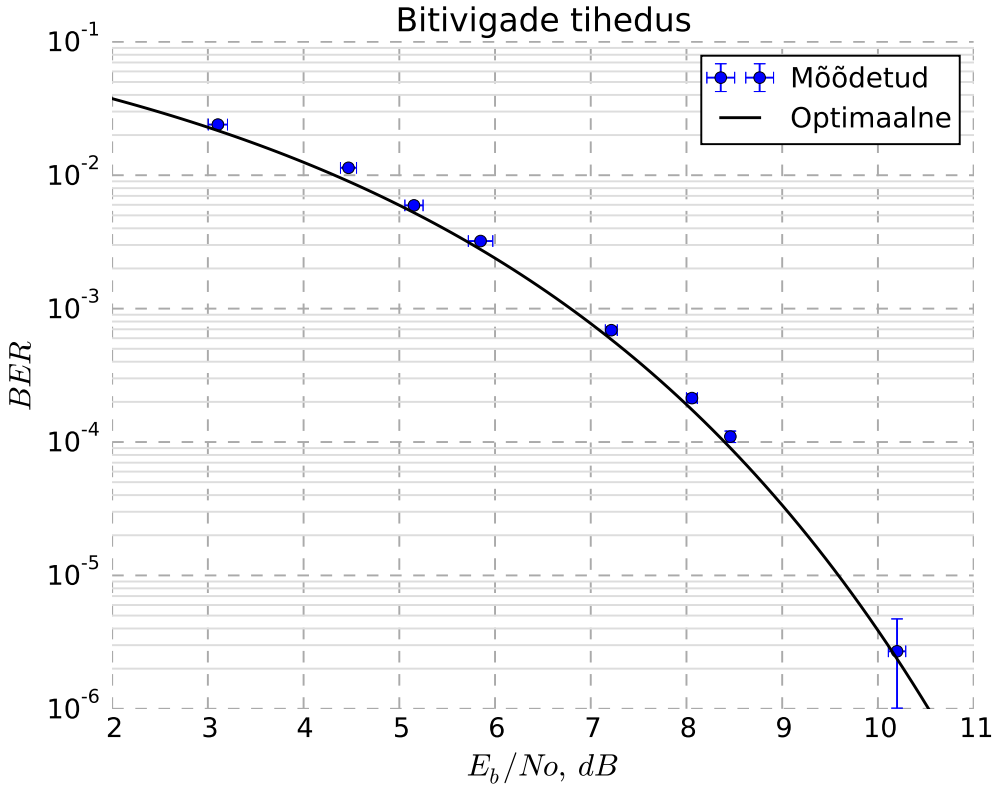
Andmed kopeeritakse ekraanilogist failidesse, mida seejärel töödeldakse Pythonis koostatud skripti abil. Signaal-müra suhet modelleeritakse kui normaaljaotusega juhuväärtust, bitivigade arvu ühe mõõtmise jooksul kui Poissoni jaotusega väärtust. Mõlema parameetri jaoks hinnatakse 95% usaldusnivoo. Programm on antud lisas 6.1.3 ning digitaalkujul leitav [18, /Testing/BER].

BPSK teoreetiline bitivigade tiheduse sõltuvus on antud [32, valem 5-2-5]:

$$P_d = \frac{1}{2} erfc(\sqrt{(\frac{S}{N})}), \tag{4.1}$$

kus

- P_d on bitivigade tihedus,
- S on müravaba signaali tugevus detektoris,
- N on müra tugevus detektoris.



Joonis 12: Bitivigade tiheduse võrdlus teoreetilisega.

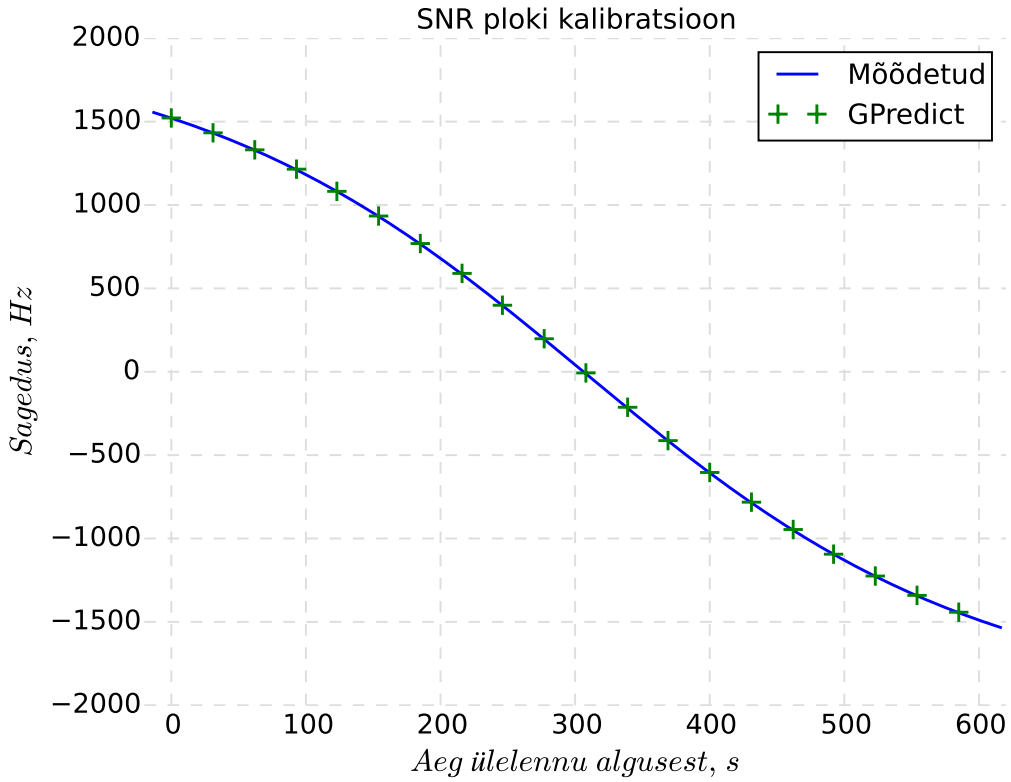
Võib näha head kooskõla teoreetilise kõvera ning mõõtepunktide vahel. Maksimaalne kõrvalekalle optimaalsest ei ületa 0.5 dB. Madalaima mõõdetud signaali-müra suhte määras Costase ahela stabiilsus: allpool $SNR = 3$ tõusis faasi pöördumiste tihedus, mis tegi terve paketi kättesaamise pea võimatuks.

4.2 Doppleri ploki võrdlus GPredictiga

Doppleri korrektsiooni ploki puhul on vaja kindlustada, et kasutatud teegi rakendamise erisused pole põhjustanud olulisi muutusi tulemustes. Selleks võrreldakse GPredicti väljastatud andmeid ploki genereeritud sagedusnihkega.

GPredict võimaldab väljastada ennustustabeli satelliidi järgmise ülelennu kohta. Väljundi algusaeg teisendatakse USRP ajakoodiks, mis antakse Doppleri korrektsiooni ploki sisendis esimese võendi vastuvõtuaja lipuna. Väljund ühendatakse GnuRadio standardteegi plokiga *Quadrature demod*, mis võenditevahelise faasinihke ning diskreetimissageduse järgi arvutab välja hetksageduse, mis talletatakse kõvakettale.

Testi tulemused on esitatud joonisel 13. Kood on kättesaadav [18, /Testing/Doppler].



Joonis 13: Simuleeritud satelliidi ülelend

Maksimaalne vahe GPredicti ja töös koostatud ploki ennustuste vahel ei ületa 6 Hz.

5 Arendusvõimalused

Ehkki vastuvõtja töötab, on arenguruumi veel. Järgnevalt on välja toodud valdkonnad, mis vajavad tähelepanu.

Kasutusmugavus: sageduse ja satelliidi muutmine ei peaks endas hõlmama programmi seismajätmist. Satelliidi valik peaks olema nimekirjast, ning loetama otse andmepakkuja poolt jagatavast failist.

Parandada saab Doppleri korrektsiooni ploki arvutuslikku efektiivsust: kombineerides selle harvendava filtriga, on võimalik töötlemist vajavate andmete hulka harvendusteguriga võrdeliselt vähendada – käesolevas töös tähendaks see vähemalt 10 korda suuremat jõudlust. Lisaks saab optimeerida arvutust võttes kasutusele VOLK teegid, mis defineerivad paljude tehete jaoks optimeeritud SIMD lahendused.

Edasi on vaja töötada ka kasutatud sagedussüntesaatoriga ADF4158. Ehkki EC-2-l seda ei rakendata, on vaja analüüsida selle sobivust saatjaks mõnes järgnevas projektis. See nõuab tutvumist süntesaatori tagasisidefiltriga: kui palju õnnestub piirata kiirgust kõrvalkanalitele ning kui suurt andmeedastuskiirust on võimalik saavutada.

5.1 Veaparandus

Kuna plaanitavad andmemahud on suured, on vaja uurida ning välja arendada veaparanduskoodide kasutamise võimekus. Peale eetriaja säästmise aitab see tasakaalus hoida ka satelliidi energiabilanssi: saatja on sees harva, kuid sel ajal on see suurim elektritarbija. Veaparandus on oluline ka EC-3 kuumissiooni perspektiivis: Kuna vahemaa satelliidi ja maajaama vahel on suur, on andmete taassaatmise nõudmine satelliidilt aeglane (edasi-tagasi side viide u. 2.5 sekundit)

Viited

[1] IEEE standard 1900.1-2008. *IEEE Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management*. 2008. DOI: 10.1109/IEEESTD.2008.4633734.

[2] ITU-T Study Group 4. *ITU-T Recommendation O.150. General Requirements for Instrumentation for Performance Measurement on Digital Transmission Equipment*. 1996. URL: http://www.itu.int/rec/T-REC-O.150-199605-I/en (külastatud 05/28/2016).

[3] Analog Devices, Inc. *AD9625. 12-Bit, 2.6 GSPS/2.5 GSPS/2.0 GSPS, 1.3 V/2.5 V Analog-to-Digital Converter*. URL: http://www.analog.com/en/products/analog-to-digital-converters/high-speed-ad-10msps/ad9625.html (külastatud 05/22/2016).

[4] Leif Åsbrink. *Linrad home page*. URL: http://www.sm5bsz.com/linuxdsp/linrad.htm (külastatud 05/23/2016).

[5] Alexandru Csete. *About GPredict*. URL: http://gpredict.oz9aec.net/index.php (külastatud 05/27/2016).

[6] Alexandru Csete. *Gqrx SDR*. URL: http://gqrx.dk/ (külastatud 05/23/2016).

[7] Ettus Research, Inc. *UBX 10-6000 MHz Rx/Tx (40 MHz, N Series and X Series)*. URL: https://www.ettus.com/product/details/UBX40 (külastatud 05/23/2016).

[8] Ettus Research, Inc. *USRP Bandwidth*. URL: https://www.ettus.com/kb/detail/usrp-bandwidth (külastatud 05/23/2016).

[9] Ettus Research, Inc. *USRP Software Defined Radio*. URL: https://www.ettus.com/product/ (külastatud 05/22/2016).

[10] Ettus Research, Inc. *USRP2 and N2x0 Series*. URL: http://files.ettus.com/manual/page_usrp2.html (külastatud 05/23/2016).

[11] Stéphane Fillod, Frank Singleton, ja the Hamlib Group. *Hamlib API Reference*. URL: http://hamlib.sourceforge.net/manuals/1.2.15/index.html (külastatud 05/22/2016).

[12] FlexRadio Systems. *FLEX Signature Series*. URL: https://www.flexradio.com/amateur-products/flex-6000-signature-series/ (külastatud 05/22/2016).

[13] FlexRadio Systems. *SmartSDR v1.7.30 Now Available for Download*. URL: https://community.flexradio.com/flexradio/topics/smartsdr-v1-7-30-now-available-for-download (külastatud 05/23/2016).

[14] Fredric J. Harris. *Multirade Signal Processing For Communication Systems*. Pearson India, 2007. ISBN: 978-8131715970.

[15] Felix R. Hoots, Ronald L. Roehrich, ja TS Kelso. *SPACETRACK REPORT NO. 3. Models for Propagation of NORAD Element Sets*. 1988. URL: http://www.celestrak.com/NORAD/documentation/spacetrk.pdf (külastatud 05/27/2016).

[16] Icom America Inc. *IC-7300 HF/50MHz Tranceiver*. URL: http://www.icomamerica.com/en/products/amateur/hf/7300/default.aspx (külastatud 05/22/2016).

[17] Steve Ireland ja Phil Harman. *Radio Communication Handbook*. Toim. Mike Dennison ja John Fielding. 11th ed. Bedford, GB: Radio Society of Great Britain, 2011. Chap. Software Defined Radio. ISBN: 9781-9050-8674-0.

[18] Laur Joost. *Bakalaureusetöö kood ja andmed*. URL: https://gitlab.com/jostikas/Thesis_final (külastatud 05/30/2016).

[19] Kitware. *About CMake*. URL: https://cmake.org/overview/ (külastatud 05/27/2016).

[20] Bryan Klofas ja Kyle Leveque. *A Survey of CubeSat Communication Systems: 2009–2012*. 2013. URL: http://www.klofas.com/papers/Klofas_Communications_Survey_2009-2012.pdf (külastatud 05/24/2015).

[21] Thorsten Kostulski ja Sam Reisenfeld. "K_a band Propagation Experiments on the Australian Low-Earth Orbit Microsatellite "FedSat"". Kogumikus: *Proceedings of the 6th Australian Communications Theory Workshop*. 2005. DOI: 10.1109/AUSCTW.2005.1624234.

[22] Carl Laufer. *About RTL-SDR*. URL: http://www.rtl-sdr.com/about-rtl-sdr (külastatud 05/22/2016).

[23] Carl Laufer. *RTL-SDR Direct-Sampling Mode*. URL: http://www.rtl-sdr.com/rtl-sdr-direct-sampling-mode/ (külastatud 05/22/2016).

[24] Anatolii N. Leukhin ja Egor N. Potekhin. "Optimal Peak Sidelobe Level Sequences up to Length 74". Kogumikus: *Proceedings of the 10th European Radar Conference*. 2013. ISBN: 978-2874870330. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6689222.

[25] Howard Long. *FUNcube Dongle*. URL: http://www.funcubedongle.com/ (külastatud 05/22/2016).

[26] S. Lawrence Marple Jr. "Estimating group delay and phase delay via discrete-time "analytic" cross-correlation". Kogumikus: *IEEE Transactions on Signal Processing* 47 (9 1999). DOI: 10.1109/78.782223.

[27] Microtelecom s.r.l. *Perseus Direct-Sampling SDR Receiver*. URL: http://www.microtelecom.it/perseus/ (külastatud 05/22/2016).

[28] National Instruments. *Software-Defined Radio*. URL: http://www.ni.com/sdr/# (külastatud 05/23/2016).

[29] Michael Ossman. *Great Scott Gadgets - HackRF One*. URL: https://greatscottgadgets.com/hackrf/ (külastatud 05/22/2016).

[30] Toomas Parve, toimetaja. *Technical Vocabulary*. URL: http://www.elin.ttu.ee/~parveto/EthnLing/TechVoca/Index.htm (külastatud 05/23/2016).

[31] David R. Pauluzzi ja Norman C. Beaulieu. "A Comparison Of SNR Estimation Techniques For The AWGN Channel". Kogumikus: *IEEE Transactions on Communications* 48 (10 2000). DOI: 10.1109/26.871393.

[32] John G. Proakis. *Digital Communication*. 4th ed. McGraw Hill, 2000. Chap. 5. ISBN: 978-0072321111.

[33] Tom Rondeau. *digital: fixing PFB clock sync block handling of tags.* URL: https://github.com/trondeau/gnuradio/commit/8b5169b933cb4d9443ed3a8e242a22bb981c1267 (külastatud 05/29/2016).

[34] Toomas Ruuben. *Pidevsignaalide töötlemine*. 2014. URL: http://www.lr.ttu.ee/pidevsignaalid/2014sygis/PIDEVSIGNAALID_11.pdf (külastatud 05/23/2016).

[35] SDR-Radio.com Limited. *It's All About The Software*. URL: http://sdr-radio.com/ (külastatud 05/23/2016).

[36] Bill Shupp, toimetaja. *libgpredict on GitHub*. URL: https://github.com/shupp/libgpredict/tree/master/sgpsdp (külastatud 05/27/2016).

[37] Andris Slavinskis, Silver Lätt, ja Mart et al. Noorma. "ESTCube-1 In-Orbit Experience and Lessons Learned". Kogumikus: *IEEE Aerospace and Electronic Systems Magazine* 30 (8 2015), lk. 12–22. DOI: 10.1109/MAES.2015.150034.

[38] Texas Instruments Inc. *RF-Sampling and GSPS ADCs*. 2012. URL: http://www.ti.com/lit/sg/snwt001/snwt001.pdf (külastatud 05/22/2016).

[39] The GNU Radio Foundation, Inc. *costas_loop_cc_impl.cc*. Versioon 3.7.9.2. May 27, 2016. URL: https://github.com/gnuradio/gnuradio/blob/master/gr-digital/lib/costas%5C_loop%5C_cc%5C_impl.cc.

[40] The GNU Radio Foundation, Inc. *GNU Radio*. URL: http://gnuradio.org/ (külastatud 05/23/2016).

[41] The GNU Radio Foundation, Inc. *gr::digital::pfb_clock_sync_ccf Class Reference*. URL: http://gnuradio.org/doc/doxygen/classgr%5C_1%5C_1digital%5C_1%5C_1pfb%5C_%5C_clock%5C_%5C_sync%5C_%5C_ccf.html (külastatud 05/27/2016).

[42] The MathWorks, Inc. *Software-Defined Radio (SDR)*. URL: http://se.mathworks.com/discovery/sdr.html (külastatud 05/23/2016).

[43] Mario Täubel. *HDSDR. High Definition Software Defined Radio*. URL: http://www.hdsdr.de/index.html (külastatud 05/23/2016).

[44] *WB5RVZ Software Defined Radio Homepage*. URL: http://wb5rvz.com/sdr/ (külastatud 05/22/2016).

[45] Xilinx Inc. *Vivado Design Suite*. URL: http://www.xilinx.com/products/design-tools/vivado.html (külastatud 05/23/2016).

6 Lisad

Failiviited on defineeritud `https://gitlab.com/jostikas/Thesis_final` ([18]) suhtes.

6.1 BER[18] testimine.

6.1.1 STM32F1 PRBS9 genereerimiskood

Fail `/MCU/Src/stm32f1xx_it.c`

```
1 void TIM1_UP_TIM16_IRQHandler(void)
2 {
3   /* USER CODE BEGIN TIM1_UP_TIM16_IRQn 0 */
4   output_next_bit();
5   /* USER CODE END TIM1_UP_TIM16_IRQn 0 */
6
7   HAL_TIM_IRQHandler(&htim1);
8 }
```

Fail `/MCU/Inc/main.h`:

```
1 #define ZEROS 16*8  /* 16 bytes of 0-s. */
2
3 /*! Syncword to use. */
4 const int g_syncword[] =
      {0,0,1,1,0,0,0,1,1,1,1,1,1,0,1,0,1,0,1,1,0,1,1,0};
5 const int g_syncword_len = sizeof (g_syncword) / sizeof
      g_syncword[0];
6
7 /*! Access code to use. */
8 const int g_access[] = {0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,
9                         1,1,1,1,0,0,1,0,1,1,0,1,0,1,0,1,0};
10 const int g_access_len = sizeof (g_access) / sizeof g_access[0];
11
12 /*! length field, 256 + 2*2 (for the regstate). 32 bits by definition
13 (2*uint16_t, as expected by the packet deinterleaver).
14 Hope this long row of 0-s doesn't cause sync issues... */
15 const int g_packetlen[32] = {0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,
16                              0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0};
17 const int g_packetlen_len = 32;
```

Fail `/MCU/Src/main.c`:

```
1 static uint16_t reg = 0x0001;  //State of the shift register
2
3 void output_bit(GPIO_PinState state)
4 {
```

[18] Bit Error Rate

```
5   HAL_GPIO_WritePin(PRBS_OUT_GPIO_Port, PRBS_OUT_Pin, state); //
      Write out to GPIO
6 }
7
8 int sync_next_bit(void)
9 {
10  static int count = 0;
11
12  output_bit( g_syncword[count++] ? GPIO_PIN_SET : GPIO_PIN_RESET );
13
14  if ( count >= g_syncword_len )
15  {
16    count = 0;
17    return 1;
18  }
19  else return 0;
20 }
21
22 int access_next_bit(void)
23 {
24  static uint8_t count = 0;
25
26  output_bit( g_access[count++] ? GPIO_PIN_SET : GPIO_PIN_RESET );
27  if (count >= g_access_len)
28  {
29    count = 0;
30    return 1;
31  }
32  else return 0;
33 }
34
35 int packetlen_next_bit(void)
36 {
37  static uint8_t count = 0;
38
39  output_bit( g_packetlen[count++] ? GPIO_PIN_SET : GPIO_PIN_RESET );
40  if (count >= g_packetlen_len)
41  {
42    count = 0;
43    return 1;
44  }
45  else return 0;
46 }
47
48 int regstate_next_bit(void)
49 {
50  static int count = 0;
51  static uint16_t regstate = 0;  /*< Actual register will never have
      0. */
```

```
52    if ( !count )  regstate = reg;
53
54    output_bit( (regstate & (1<<15)) ? GPIO_PIN_SET : GPIO_PIN_RESET );
55    regstate <<= 1;
56
57    if ( (++count) >= 16)
58    {
59      count = 0;
60      regstate = 0; //Will be 0 anyway, as all bits have been shifted
       out.
61      return 1;
62    }
63    else return 0;
64  }
65
66  int PRBS9_next_bit(void)
67  {
68    static uint16_t count = 0;
69    output_bit(reg&1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
70
71    /* Update the state */
72    int newbit = (((reg >> 8) ^ (reg >> 4)) & 0x0001); //x^9+x^5+1,
       PRBS9
73    reg = ((reg << 1) | newbit) & 0x01FF;
74
75    if (++count >= 256*8)
76    {
77      count = 0;
78      return 1;
79    }
80    else return 0;
81  }
82
83  int zeros_next_bit(void)
84  {
85    static int count = 0;
86
87    output_bit(GPIO_PIN_RESET);
88
89    if ( ++count >= ZEROS )
90    {
91      count = 0;
92      return 1;
93    }
94    else
95    {
96      return 0;
97    }
98  }
```

```
 99
100 /*
101 Output the frame in sections. If called function returns 1, go to
        next state.
102
103 syncword is used for demodulation.
104 Access code and packetlen are used by the packet deframer, but
        really we could
105 do without, as syncword should be enough, and our packet has
        constant structure.
106 Regstate (twice) follows, so if we lose a packet, we can detect it
        and resync.
107 PRBS9 outputs the actual test bits.
108 zeros is padded to the end.
109 */
110 void output_next_bit(void)
111 {
112   static int clock = 0;
113   static int state = 0;
114   static int (*state_funcs[7])() = {
115     sync_next_bit,
116     access_next_bit,
117     packetlen_next_bit,
118     regstate_next_bit,
119     regstate_next_bit, /* Register twice, lest we calculate based on
       wrong register value */
120     PRBS9_next_bit,
121     zeros_next_bit
122   };
123   /* On falling edge of clock */
124   if (clock)
125   {
126     /* If the corresponding function returns true, move to the next
       state */
127     if ( state_funcs[state]() ) state++;
128
129     if ( state >= 7 ) state = 0;
130   }
131   HAL_GPIO_TogglePin(HCLK_OUT_GPIO_Port, HCLK_OUT_Pin);  // Output
       clock.
132   clock = !clock;
133 }
134
135 void wait_for_PB(void)
136 {
137   while(HAL_GPIO_ReadPin(UserBTN_GPIO_Port, UserBTN_Pin))
138   {} //wait for button to be released.
139   HAL_Delay(10); //Debounce delay 10 ms
```

```
140   while(!HAL_GPIO_ReadPin(UserBTN_GPIO_Port, UserBTN_Pin))  // Wait
      for button to be pressed
141   {
142     HAL_Delay(10);
143     if(HAL_GPIO_ReadPin(UserBTN_GPIO_Port, UserBTN_Pin))
144     {
145       break;
146     }
147   }
148 }
```

6.1.2 SNR ploki kalibreerimisandmete töötlemine

Fail `/Testing/BER/extract_cal_data.py`

```
1  from __future__ import print_function, unicode_literals
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.stats import linregress
5
6  "Extract calibration coeffs from caldata."
7  indices=range(2,12)
8
9  # Read data from files.
10 divs = [[],[]]  # means and sigmas
11 snbs = [[],[]]
12 for snr in indices:
13     div = np.fromfile(open("caldata/div{}.f32".format(snr)),
       dtype=np.float32)
14     divs[0].append(np.mean(div))
15     divs[1].append(np.std(div, ddof=1) / np.sqrt(len(div)))
16     with open("caldata/snr_block{}".format(snr)) as snblock:
17         vals = []
18         for line in snblock:
19             try:
20                 vals.append(float(line))
21             except ValueError:
22                 pass
23         snbs[0].append(np.mean(vals))
24         snbs[1].append(np.std(vals, ddof=1) / np.sqrt(len(vals)))
25
26 divs = np.array(divs)
27 snbs = np.array(snbs)
28 slope, intercept, r_value, p_value, std_err = linregress(divs[0],
       snbs[0])
29
30 errs = snbs[0] - slope*divs[0] - intercept
31
32 # Estimate std of slope and intercept
```

```
33 n = len(divs[0])
34 num = np.sqrt(np.sum(divs[0]**2) * np.sum(errs**2))
35 den = (n-2) * (n*np.sum(divs[0]**2) - np.sum(divs[0])**2)
36 s_int = np.sqrt(num/den)
37
38 num = n*np.sum(errs**2)
39 # den is same.
40 s_slo = np.sqrt(num/den)
41
42 print("Slope: {}, intercept: {}, s_slope: {}, s_inter:
       {}".format(slope,
43
       intercept,
44
       s_slo,
45
       s_int))
46
47 result = np.array((intercept, slope, s_int, s_slo), dtype=np.float32)
48 result.tofile("cal-data.f32")
```

6.1.3 Bitivigade tiheduse andmete töötlemine

Programm on kutsumiseks failide nimekirjaga bash-ist.
Fail `/Testing/BER/extract_data.py`:

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3 from __future__ import print_function
4
5 """
6 Extract BER and SNR data from BER logs, process and output to csv.
7 """
8 import numpy as np
9 import scipy
10 from scipy import stats
11
12 def extract(filename):
13     snrs = []
14     tot = 0
15     be = 0
16     with open(filename, "r") as f:
17         for line in f:
18             if line.startswith("BE :"):
19                 be = int(line[4:])  # Only keep last.
20             elif line.startswith("BTS:"):
21                 tot = float(line[4:])
22             elif line.startswith("SNR:"):
```

```
23                  snrs.append(float(line[4:]))
24          snr = np.mean(snrs)
25          snr_sigma = np.std(snrs, ddof=1)
26          snr_conf = stats.norm.interval(0.95, 0, snr_sigma /
    len(snrs))
27
28          ber = be/tot
29          ber_conf = []
30          be_conf = stats.poisson.interval(0.95, be)
31          ber_conf.append(ber - be_conf[0] / tot)  # Need those values
    relative to data
32          ber_conf.append(be_conf[1] / tot - ber)
33          return (snr, ber, snr_conf[1], ber_conf[0], ber_conf[1])
34
35 def plot(results):
36
37      plt.text(6.5, 0.12, "Bitivigade tihedus", fontsize=17,
    ha="center")
38
39 ...
40
41      na = np.array(results).T
42      measured = plt.errorbar(na[0], na[1],
43                              yerr=[na[3], na[4]],
44                              xerr=na[2],
45                              fmt='o',
46                              label=u"M~o~odetud")
47      print(measured)
48
49      theory_x = np.linspace(2,11,100)
50      theory_y = 0.5*scipy.special.erfc(np.sqrt((10**(theory_x/10))))
51      bpsk, = plt.semilogy(theory_x, theory_y, 'k-',
    label="Teoreetiline", lw=1.5)
52
53      plt.legend(handles=[measured, bpsk])
54
55      plt.show()
56
57
58 if __name__ == "__main__":
59      import csv
60      import argparse
61      from matplotlib import pyplot as plt
62
63      parser = argparse.ArgumentParser(description="Process BER
    output")
64      parser.add_argument("files", type=str, nargs="+")
65      parser.add_argument("-o", dest="outfile",
    default="bers_out.csv", type=str)
```

```
66
67      args = parser.parse_args()
68
69      results = []
70      for filename in args.files:
71          result = extract(filename)
72          if float("nan") in result:  # remove nan-containing points,
        as they casue
73              continue
74          results.append(extract(filename))
75      results.sort(key=lambda x: x[0])
76
77      writer = csv.writer(open(args.outfile, "wb"))
78      writer.writerow("snr, ber, snr_conf_low, snr_conf_high,
        ber_conf_low, ber_conf_high")
79      writer.writerows(results)
80
81      plot(results)
```

6.2 Doppleri nihke korrigeerimise plokk

Fail: `/blocks/gr-doppler/lib/doppler_tle_cc_impl.cc`

```
1 #include <cstdio>
2 #include <ctime>
3 #include <stdexcept>
4 #include <gnuradio/io_signature.h>
5 #include <gnuradio/tags.h>
6 #include <gnuradio/expj.h>
7 #include "doppler_tle_cc_impl.h"
8
9 #include <cassert>
10
11 #define MILLION 1000000
12 #define BILLION 1000000000
13 #define DOP_PERIOD_IN_SECS 1  /*!< Period of updating the frequency
       prediction. */
14 #define DOP_PERIOD d_samp_rate*DOP_PERIOD_IN_SECS  /*!< Period of
       updating the frequency in samples */
15 #define LC_KM 299792.458 /*<! Speed of light in km/s */
16
17 namespace gr {
18   namespace doppler {
19
20     doppler_tle_cc::sptr
21     doppler_tle_cc::make(
22         double samp_rate,
23         double center_freq,
```

```
24          char tle[240],
25          std::vector<double> obs,
26          bool use_tags)
27      {
28        return gnuradio::get_initial_sptr
29          (new doppler_tle_cc_impl(samp_rate, center_freq, tle, obs,
        use_tags));
30      }
31
32      /*
33       * The private constructor
34       */
35      doppler_tle_cc_impl::doppler_tle_cc_impl(
36          double samp_rate,
37          double center_freq,
38          char tle_string[240],
39          std::vector<double> obs,
40          bool use_tags)
41        : gr::sync_block("doppler_tle_cc",
42                gr::io_signature::make(1, 1, sizeof(gr_complex)),
43                gr::io_signature::make(1, 1, sizeof(gr_complex)))
44      {
45        /* Allocate memory for structs */
46        d_sat = new sat_t();
47        d_obs = new geodetic_t();
48
49        /* Initialize other member variables */
50        d_renew = false;
51        d_rx_time = uhd::time_spec_t();
52        d_offset = uhd::time_spec_t();
53        d_systime_initialized = false;
54        d_samps_to_next = 0;
55        d_nco = gr_expj(0);
56        d_end_freq = 0;
57        d_curr_tag = tag_t();
58
59        set_samp_rate(samp_rate);
60        set_center_freq(center_freq);
61        set_tle(tle_string);
62        set_obs(obs);
63        set_use_tags(use_tags);
64      }
65
66      /*
67       * Our virtual destructor.
68       */
69      doppler_tle_cc_impl::~doppler_tle_cc_impl()
70      {
71        delete d_sat;
```

```
72        delete d_obs;
73      }
74
75      /*************************************/
76      /*       Getters and setters         */
77      /*************************************/
78
79      /* Sample rate */
80      void
81      doppler_tle_cc_impl::set_samp_rate(double samp_rate)
82      {
83        d_samp_rate = samp_rate;
84        d_freqs.reserve(int(DOP_PERIOD));
85        d_renew = true;
86      }
87
88      double
89      doppler_tle_cc_impl::samp_rate(void)
90      {
91        return d_samp_rate;
92      }
93
94      /* Center frequency */
95      void
96      doppler_tle_cc_impl::set_center_freq(double center_freq)
97      {
98        d_center_freq = center_freq;
99        d_renew = true;
100     }
101
102     double
103     doppler_tle_cc_impl::center_freq(void)
104     {
105       return d_center_freq;
106     }
107
108     /* TLE - ie. satellite selection */
109     void
110     doppler_tle_cc_impl::set_tle(char tle_string[240])
111     {
112       char tle_lines[3][80];
113
114       /* Split the TLE into lines */
115       std::stringstream ss(tle_string);
116       std::string to;
117       for (int i=0;i<3;i++)
118       {
119         std::getline(ss, to);
120         strncpy(tle_lines[i], to.c_str(), 80);
```

```
121         tle_lines[i][79] = '\0';
122       }
123
124       /* Process the TLE */
125       if(Get_Next_Tle_Set(tle_lines, &d_sat->tle) != 1)
126       {
127         throw std::invalid_argument("TLE string failed checksum!");
128       }
129       else
130       {
131         select_ephemeris(d_sat);
132         memcpy(d_tle_string, tle_string, 240);  /* Lookup buffer for
    returning only. */
133         d_renew = true;
134         if (d_debug) std::cout << "TLE: " << tle() << std::endl;
135       }
136     }
137
138     char *
139     doppler_tle_cc_impl::tle(void)
140     {
141       return d_tle_string;
142     }
143
144     void
145     doppler_tle_cc_impl::set_obs(std::vector<double> obs)
146     {
147       d_obs->lat = Radians(obs[0]);
148       d_obs->lon = Radians(obs[1]);
149       d_obs->alt = obs[2]/1000;  /* Model wants this in km-s */
150       d_obs->theta = 0;
151       d_renew = true;
152       if (d_debug) std::cout << "OBS: (" << this->obs()[0] << ", "
    << this->obs()[1] << ", " << this->obs()[2] << ")" << std::endl;
153     }
154
155     std::vector<double>
156     doppler_tle_cc_impl::obs(void)
157     {
158       double ret[] = {Degrees(d_obs->lat), Degrees(d_obs->lon),
    d_obs->alt*1000};
159       return std::vector<double>(ret, ret+3);
160     }
161
162     void
163     doppler_tle_cc_impl::set_use_tags(bool use_tags)
164     {
165       d_use_tags = use_tags;
166     }
```

```
167
168     bool
169     doppler_tle_cc_impl::use_tags(void)
170     {
171       return d_use_tags;
172     }
173
174     uhd::time_spec_t
175     doppler_tle_cc_impl::tag_to_time_spec(tag_t tag)
176     {
177       if (d_debug) {
178         std::cout << "TTTS: "<< tag.key << tag.value << std::endl;
179         assert (pmt::symbol_to_string(tag.key) == "rx_time");
180       }
181
182       time_t full_secs = (time_t)
        pmt::to_uint64(pmt::tuple_ref(tag.value, 0));
183       double frac_secs = pmt::to_double(pmt::tuple_ref(tag.value,
        1));
184       assert(0 <= frac_secs && frac_secs < 1);
185
186       return uhd::time_spec_t(full_secs, frac_secs);
187     }
188
189     inline double
190     doppler_tle_cc_impl::unix_to_jd(uhd::time_spec_t rx_time)
191     {
192       if (d_debug) std::cout << "UTJD: "<< rx_time.get_real_secs()
        << std::endl;
193       return double(rx_time.get_full_secs())/86400 +
        rx_time.get_frac_secs()/86400 + 2440587.5;
194     }
195
196     /* Does the actual calling of the library. Copied wholesale from
        the original C program.
197       Note that this section is only slightly modified for the
        thesis from earlier work by Viljo Allik.*/
198     void
199     doppler_tle_cc_impl::predict(uhd::time_spec_t rx_time)
200     {
201       obs_set_t      obs_set;
202       geodetic_t     sat_geodetic;
203       double         age;
204       double jdtime = unix_to_jd(rx_time);
205       if (d_debug)  std::cout << "PRED: JD"<< jdtime << std::endl;
206
207       d_sat->jul_epoch = Julian_Date_of_Epoch (d_sat->tle.epoch); //
        => tsince = 0.0
208       d_sat->jul_utc = jdtime;
```

```
209       d_sat->tsince = (d_sat->jul_utc - d_sat->jul_epoch) * xmnpda;
210     //     d_sat->tsince = (d_sat->jul_utc - d_sat->jul_epoch);
211
212       /* call the norad routines according to the deep-space flag */
213       if (d_sat->flags & DEEP_SPACE_EPHEM_FLAG)
214           SDP4 (d_sat, d_sat->tsince);
215       else
216           SGP4 (d_sat, d_sat->tsince);
217
218       /* Convert to km-s */
219       Convert_Sat_State (&d_sat->pos, &d_sat->vel);
220
221       /* get the velocity of the satellite */
222       Magnitude (&d_sat->vel);
223       d_sat->velo = d_sat->vel.w;
224       Calculate_Obs (d_sat->jul_utc, &d_sat->pos, &d_sat->vel,
      d_obs, &obs_set);
225       Calculate_LatLonAlt (d_sat->jul_utc, &d_sat->pos,
      &sat_geodetic);
226
227       while (sat_geodetic.lon < -pi)
228           sat_geodetic.lon += twopi;
229
230       while (sat_geodetic.lon > (pi))
231           sat_geodetic.lon -= twopi;
232
233       d_sat->az = Degrees (obs_set.az);
234       d_sat->el = Degrees (obs_set.el);
235       d_sat->range = obs_set.range;
236       d_sat->range_rate = obs_set.range_rate;
237       if (d_debug)  std::cout << "PRED: RR "<< d_sat->range_rate <<
      std::endl;
238       d_sat->ssplat = Degrees (sat_geodetic.lat);
239       d_sat->ssplon = Degrees (sat_geodetic.lon);
240       d_sat->alt = sat_geodetic.alt;
241       d_sat->ma = Degrees (d_sat->phase);
242       d_sat->ma *= 256.0/360.0;
243       d_sat->phase = Degrees (d_sat->phase);
244
245       /* same formulas, but the one from predict is nicer */
246       //d_sat->footprint = 2.0 * xkmper * acos (xkmper/d_sat->pos.w);
247       d_sat->footprint = 12756.33 * acos (xkmper /
      (xkmper+d_sat->alt));
248       age = d_sat->jul_utc - d_sat->jul_epoch;
249       d_sat->orbit = (long) floor((d_sat->tle.xno * xmnpda/twopi +
250                       age * d_sat->tle.bstar * ae) * age +
251                       d_sat->tle.xmo/twopi) + d_sat->tle.revnum - 1;
252     }
253
```

```
254     void
255     doppler_tle_cc_impl::linterp(std::vector<float> &freq,
256
257         double start_freq,
258         double end_freq,
259         int samps_to_next
260         )
261     {
262       freq.clear();
263       if (d_debug) {
264         std::cout << "LINT: SF "<< start_freq << std::endl;
265         std::cout << "LINT: EF "<< end_freq << std::endl;
266         std::cout << "LINT: STN "<< samps_to_next << std::endl;
267       }
268
269       for (int i=0;i < samps_to_next;i++)
270       {
271         freq.push_back(float(start_freq + i * (end_freq-start_freq)
        / samps_to_next));
272       }
273     }
274
275     uhd::time_spec_t
276     doppler_tle_cc_impl::get_system_time()
277     {
278       using namespace uhd;
279       time_spec_t ts;
280       struct timeval tmval;
281
282
283       if ( d_systime_initialized )
284       {
285         return time_spec_t::get_system_time() + d_offset;
286       }
287       else
288       {
289         d_systime_initialized = true;
290         ts = time_spec_t::get_system_time();
291         gettimeofday(&tmval, NULL);
292         d_offset = time_spec_t::get_system_time() + ts;
293         d_offset = time_spec_t(tmval.tv_sec, tmval.tv_usec, MILLION)
        -
294           time_spec_t(d_offset.get_real_secs() / 2);
295
296         if (d_debug) std::cout << "SYST: F "<< (ts +
        d_offset).get_real_secs() << std::endl;
297         return ts + d_offset;
298       }
299     }
```

```
300
301     int
302     doppler_tle_cc_impl::work(int noutput_items,
303         gr_vector_const_void_star &input_items,
304         gr_vector_void_star &output_items)
305     {
306       const gr_complex *in = (const gr_complex *) input_items[0];
307       gr_complex *out = (gr_complex *) output_items[0];
308
309       std::cout.precision(15);
310
311       using std::deque;
312
313       /*******************************/
314       /*        Predictor logic      */
315       /*******************************/
316
317       bool renew = false;
318       uhd::time_spec_t time_next = uhd::time_spec_t();
319
320       if ( !d_samps_to_next )
321       {  // If we're out of predictions
322         if (d_debug)  std::cout << "OOP" << std::endl;
323         renew = true;
324         d_samps_to_next = DOP_PERIOD;  // This will later be
    reduced, if necessary.
325       }
326
327       /* First case:
328        * Using accurate reception time stamps from the radio.
329        *
330        * If there are any tags not on the first sample, we'll only
    process up
331        * until them, and let the next work() call deal with the time
    change.
332        */
333       if (d_use_tags)
334       {
335         std::vector<tag_t> tags;
336         get_tags_in_window(tags, 0, 0, noutput_items,
    pmt::intern("rx_time"));
337         deque<tag_t> tags_d(tags.begin(), tags.end());
338         tags.clear();
339         while ( !tags_d.empty() )
340         {
341           if ( tags_d.front().offset == nitems_read(0) ) {
342             if (d_debug) {
343               std::cout << "TAG@FRONT: "<< tags_d[0].key <<
    std::endl;
```

```
344               std::cout << "       VAL: "<< tags_d[0].value <<
      std::endl;
345               std::cout << "    OFFSET: "<< tags_d[0].offset <<
      std::endl;
346             }
347
348             d_curr_tag = tags_d.front();
349             tags_d.pop_front();
350             d_samps_to_next = DOP_PERIOD;
351             renew = true;
352             d_end_freq = 0;  //Signals that we can't use cached
      frequency value.
353           }
354           else {
355             /* Process no more than up to the next tag or to current
      next time, if earlier. */
356             if (d_debug) {
357               std::cout << "TAG@LATER: "<< tags_d[0].key <<
      std::endl;
358               std::cout << "       VAL: "<< tags_d[0].value <<
      std::endl;
359               std::cout << "    OFFSET: "<< tags_d[0].offset <<
      std::endl;
360               std::cout << "RELOFFSET: "<<
      tags_d[0].offset-nitems_read(0) << std::endl;
361             }
362
363             uint rel_offset = tags_d[0].offset - nitems_read(0);
364             if (rel_offset < d_samps_to_next) {
365               d_samps_to_next = rel_offset;
366               if (d_debug) {
367                 std::cout << " STN: "<< d_samps_to_next << std::endl;
368                 std::cout << "SIZEFREQS: "<< d_freqs.size() <<
      std::endl;
369               }
370             }
371             break;
372           }
373         }
374         if (renew ) // Calculate timestamps for renewal
375         {
376           /* Calculate current time from last received tag and
      number of samples processed.
377            * That way we avoid error accumulation
378            */
379           if (d_curr_tag.key != pmt::intern("rx_time"))
380           {
381             std::cout << "Doppler block says: " << std::endl;
```

```
382           std::cout << "No starting rx_time tag found. Using
      current time as starting point.\n"
383                        "(If your hardware driver doesn't generate
      such tags, use 'use_tags=False',\n"
384                        "as tagged mode only updates based on tags
      and sample rate)." << std::endl;
385           uhd::time_spec_t curr_time = get_system_time();
386           d_curr_tag.key = pmt::intern("rx_time");
387           d_curr_tag.value = pmt::make_tuple(
388             pmt::from_uint64(curr_time.get_full_secs()),
389             pmt::from_double(curr_time.get_frac_secs())
390           );
391           d_curr_tag.offset = nitems_read(0);
392         }
393         d_rx_time = tag_to_time_spec(d_curr_tag) +
394           uhd::time_spec_t(double(nitems_read(0) -
      d_curr_tag.offset) / d_samp_rate);
395         time_next = d_rx_time +
      uhd::time_spec_t(double(d_samps_to_next)/d_samp_rate);
396         if (d_debug) {
397           std::cout << "RENEW_wTAGS" << std::endl;
398           std::cout << "      RXT: " << d_rx_time.get_real_secs()
      << std::endl;
399           std::cout << "      NXT: " << time_next.get_real_secs()
      << std::endl;
400         }
401       }
402     }
403
404     /*! Second case:
405      * Using wall clock time at the moment that work() is called.
406      * Will use a higher update rate, to recover quicker from
      dropped packets.
407      */
408     else
409     {
410       uhd::time_spec_t curr_time = get_system_time();
411       if ((curr_time - d_rx_time).get_real_secs() > DOP_PERIOD/2)
412       {
413         renew = true;
414         d_end_freq = 0;  // Also need to calculate new start
      frequency.
415         std::cout << "J";
416       }
417
418       if (renew)
419       {
420         d_rx_time = get_system_time();
421         d_samps_to_next = DOP_PERIOD/2;  //Update prediction
```

```
422           time_next = d_rx_time +
      uhd::time_spec_t(double(d_samps_to_next)/d_samp_rate);
423           if (d_debug) {
424             std::cout << "RENEW_wTIME" << std::endl;
425           }
426         }
427       }
428
429       /* If necessary, generate a new prediction vector. */
430       if (renew)
431       {
432         double start_freq;
433
434         /* Get the timestamps to use for prediction. */
435         if ( d_end_freq ) {  // Use cached value, if possible
436           start_freq = d_end_freq;
437         }
438         else {
439           predict(d_rx_time);
440           if (d_debug) {
441             std::cout << "RAW_SFREQ" << (d_sat->range_rate/LC_KM) *
      d_center_freq *(-1.0) << std::endl;
442           }
443           start_freq = (d_sat->range_rate/LC_KM) * d_center_freq
      *(-1.0) * twopi / d_samp_rate;
444         }
445         predict(time_next);
446         if (d_debug) {
447             std::cout << "RAW_EFREQ" << (d_sat->range_rate/LC_KM) *
      d_center_freq *(-1.0) << std::endl;
448           }
449         d_end_freq = (d_sat->range_rate/LC_KM) * d_center_freq
      *(-1.0) * twopi / d_samp_rate;
450
451         /* Update the d_freqs vector */
452         linterp(d_freqs, start_freq, d_end_freq, d_samps_to_next);
453         d_freqs_idx = 0;
454       }
455
456       /* Do actual work. */
457       if ( d_samps_to_next < noutput_items) {
458         noutput_items = d_samps_to_next;
459       }
460
461       for (int i = 0; i<noutput_items; i++)
462       {
463         d_nco *= gr_expj(d_freqs[d_freqs_idx+i]);
464         out[i] = d_nco * in[i];
465       }
```

```
466         d_samps_to_next -= noutput_items;
467         d_freqs_idx += noutput_items;
468 
469         //memcpy(out, in, noutput_items * sizeof(in[0]));
470         return noutput_items;
471       }
472     } /* namespace doppler */
473 } /* namespace gr */
```

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, LAUR JOOST ,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„BPSK vastuvõtt tarkvaralise raadioga"
mille juhendajad on VILJO ALLIK, MSc (Tartu Observatoorium) ja INDREK SÜNTER, MSc (Tartu Ülikool)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni; 1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 30.05.2016