Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Zero-IF architektūros siųstuvų/imtuvų I/Q disbalanso kompensacijos eksploatacijos metu metodo sukūrimas ir tyrimas

Baigiamasis magistro projektas

Tomas Ševenikas

Projekto autorius

Prof. dr. Žilvinas Nakutis

Vadovas

Kaunas, 2020

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Zero-IF architektūros siųstuvų/imtuvų I/Q disbalanso kompensacijos eksploatacijos metu metodo sukūrimas ir tyrimas

Baigiamasis magistro projektas

Elektronikos inžinerija (6211EX012)

Tomas Ševenikas

Projekto autorius

Prof. dr. Žilvinas Nakutis

Vadovas

Doc. Dr. Pranas Kuzas

Recenzentas

Kaunas, 2020

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Tomas Červenikas

Zero-IF architektūros siųstuvų/imtuvų I/Q disbalanso kompensacijos eksploatacijos metu metodo sukūrimas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Tomo Červenikas, baigiamasis projektas tema „Zero-IF architektūros siųstuvų/imtuvų I/Q disbalanso kompensacijos eksploatacijos metu metodo sukūrimas ir tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

Tomas Červenikas	
(vardą ir pavardę įrašyti ranka)	(parašas)

Šervenikas, Tomas. Zero-IF architektūros siųstuvų/imtuvų I/Q disbalanso kompensacijos eksploatacijos metu metodo sukūrimas ir tyrimas. Magistro baigiamasis projektas / vadovas prof. dr. Žilvinas Nakutis; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, inžinerijos mokslai.

Reikšminiai žodžiai: FPGA, Zero-IF, Direct Conversion, SDR, IQ disbalansas.

Kaunas, 2020. 65 p.

Santrauka

Šiame darbe aprašomas Zero-IF architektūros fazinio ir amplitudinio IQ disbalansų derinimo metodas, iš kitų literatūroje aprašomų metodų, išsiskiriantis tuo jog norint jį atlikti neprivaloma nutraukti siųstuvo/imtuvo eksploatacijos. Pateikiamos derinimo metodo įgyvendinimo, panaudojant programuojamą logiką, schemos, skirtingų metodo parametrų įtaka derinimo procesui.

Šervenikas, Tomas. Development and research of a method for online I/Q imbalance compensation of Zero-IF architecture transceivers. Master's Final Degree Project / supervisor prof. Žilvinas Nakutis; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics Engineering, Engineering Sciences.

Keywords: FPGA, Zero-IF, Direct Conversion, SDR, IQ imbalance.

Kaunas, 2020. 65 pages.

Summary

This thesis describes a new method for lessening the effects of phase and amplitude IQ imbalance in Zero-IF transceivers. This method is unique in that it does not require the normal operation of the transceiver to be suspended while performing the calibration. Thesis contains the schematic representation of the method, as well as the influence of various method’s parameters to the process of calibration.

Turinys

Santrumpų ir terminų sąrašas

Santrumpos

AGFT – (Atvirkštinė greitoji Furjė transformacija) matematinė operacija perkelianti skaičių seką iš dažnių srities į laiko sritį.

FDD – (angl. Frequency Division Duplex) Dupleksinių mainų įgyvendinimo metodas, kai abejų krypčių duomenys perduodami tuo pačiu laiku, bet skirtingais dažniais;

FFT – (angl. Fast Fourier Transform) algoritmas, perkeliantis signalą iš laiko srities į dažnių sritį;

FPGA – (angl. Field Programmable Gate Array) integrinis lustas, kurio konfigūracija gali būti keičiama po gamybos;

GFT – (Greitoji Furjė transformacija) matematinė operacija perkelianti skaičių seką iš laiko srities į dažnių sritį;

IF – (angl. Intermediate frequency) tarpinis dažnis naudojamas daugumoje siųstuvų/imtuvų architektūrų;

LTE – (angl. Long term evolution) Bevielių komunikacijų standartas;

PDU – (angl. Protocol Data Unit) LTE standarte protokolo paketui įvardinti naudojamas terminas;

RRC – (angl. Radio resource control) LTE standarto protokolas skirtas ryšio kokybei užtikrinti;

SC-FDA – (angl. Single carrief frequency division multiplexing) į OFDM panašus duomenų apdorojimo metodas, skirtas mažesnės galios siųstuvams;

SDR – (angl. Software defined radio) radijo komunikacijų sistema, kurioje dalį tradiciškai aparatinės įrangos atliekamų funkcijų atlieka programinė įranga;

SDU – (angl. Service Data Unit) LTE standarte protokolo pakete esantį aukštesnio lygio protokolo paketą įvardinti naudojamas terminas;

TDD – (angl. Time Division Duplex) Dupleksinių mainų įgyvendinimo metodas, kai abejų krypčių duomenys perduodami tuo pačiu dažniu, bet skirtingu laiku;

OFDM – (angl. Orthogonal frequency division multiplexing) metodas duomenims apdoroti prieš juos transliuojant;

Įvadas

Vienas iš faktorių darančių įtaką SDR sistemų lankstumui yra tai kaip lengva šios sistemos siųstuvą ir imtuvą perderinti skirtingiems dažniams. Dauguma siųstuvų/imtuvų architektūrų veikia norimą signalą perkeldamos į žemesnį, tarpinį, dažnį, tokiu būdu mažindamos techninius reikalavimus juostiniams filtrams, kurie reikalingi norimam dažnių ruožui išskirti. Tačiau tokių sistemų lankstumas apribojamas tuo, jog jų juostiniai filtrai turi būti pritaikyti tiek naudojamam tarpiniam dažniui, tiek naudingo signalo juostos pločiui. Šiuo trūkumu nepasižymi ZeroIF architektūra, kurios tarpinis dažnis yra 0Hz. Tokiu atveju naudingam signalui išskirti reikalingas daug paprastesnis (lyginant su juostiniu) žemųjų dažnių filtras[1].

Kartu su didesniu lankstumu ši architektūra pasižymi ir keliais jai būdingais trūkumais. Dėl naudingam signalui išskirti naudojamo žemų dažnių filtro sukuriama galimybė atsirasti nepageidaujamai nuolatinei dedamajai. Žemas tarpinis dažnis taip pat daro sistemą jautrią signalo kvadratūrinių dedamųjų fazių ir amplitudžių nesutapimams[2].

Tipinis metodas šioms problemoms spręsti - reguliariai atliekamas derinimas, kurio metu yra sustabdomas normalus sistemos veikimas ir naudojant determinuotus testinius signalus įvertinama kiek ir kaip sistema yra nukrypusi nuo pilno IQ balanso. Toks metodas, dėl būtinybės sustabdyti normalią sistemos veiklą nėra idealus situacijose, kuomet SDR yra naudojamas kaip 4G bazinė stotis, ar kitose situacijose, kuomet radijo ryšio nutraukimas reikštų reikšmingą paslaugos kokybės praradimą.

Šiame darbe aptariama galimybė SDR veikiantį kaip 4G bazinę stotį derinti naudojant tinklo vartotojų įrangos siunčiamą tarnybinę informaciją apie ryšio kokybę. Likusi darbo dalis aptaria naują derinimo metodą, bei jo veikimo rezultatus.

Aptariant metodus daroma prielaidos, jog SDR, kuriuose šie metodai turėtų būti įgyvendinami, panaudoja programuojamą logiką ir kad metodo įgyvendinimas atliekamas minėtoje programuojamoje logikoje.

Darbo tikslas – sukurti ZeroIF siųstuvų/imtuvų derinimo metodą, kurį naudojant nereikėtų laikinai nutraukti siųstuvo ar imtuvo eksploatacijos, bei ištirti šio metodo savybes. Tikslui pasiekti iškelti tokie uždaviniai:

1. Apžvelgti galimus Zero-IF architektūros siųstuvų / imtuvų IQ disbalanso derinimo metodus.
2. Sukurti ir modeliuoti naują disbalanso derinimo metodą.
3. Sukurtą metodą įgyvendinti realioje aparatūroje ir ištestuoti su realiais duomenimis.

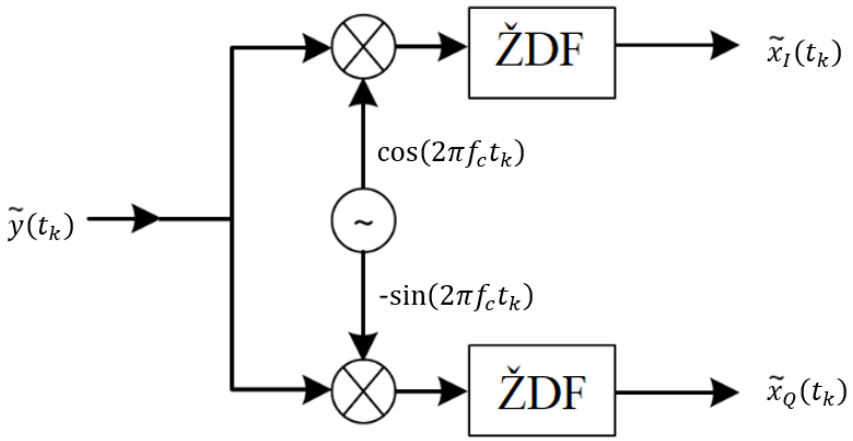
1. Literatūros apžvalga

1.1. ZeroIF siųstuvo / imtuvo architektūra ir jos trūkumai

ZeroIF architektūros siųstuvai bei imtuvai yra patrauklūs situacijose, kuomet reikia mažo komponentų kiekio, kompaktiškumo arba lankstumo dažnių srityje, kadangi duomenų perkėlimas iš arba į nešančiosios dažnį vyksta vienu etapu (priešingai negu kitose architektūrose, kuriose naudojami tarpiniai dažniai). Tačiau ši architektūra pasižymi dviem pagrindiniais trūkumais – IQ disbalansu ir nepageidaujama nuolatine dedamąja. Šios problemos vienodai pasireiškia tiek siųstuve, tiek imtuve, todėl paprastumo dėlei šiame poskyryje bus aprašomas tik imtuvas.

1.1.1. I/Q fazinis ir amplitudinis disbalansas

ZeroIF architektūroje priimtas signalas, norint jį perkelti į žemesnį dažnį, yra sudauginamas su dviem 90 -čia laipsnių besiskiriančiomis sinusoidėmis. Tam dažniausiai naudojamas vienas sinusinio signalo generatorius kartu su fazės pasukimą atliekančiais elementais. Ši architektūra pavaizduota 1.1. pav. Dėl šias funkcijas atliekančių elementų ne idealumo, bei išorinių aplinkos veiksnių gali būti sudėtinga užtikrinti idealų 90 -ties laipsnių skirtumą tarp minėtų sinusoidžių, bei jų amplitudės vienodumą.



1.1 pav. ZeroIF imtuvo architektūra. Pritaikyta iš [3]

Idealus priimtas signalas aprašomas taip:

$$y(t_k) = x_I(t_k)\cos(2\pi f_c t_k) - x_Q(t_k)\sin(2\pi f_c t_k) \qquad (1.1.1.1)$$

Čia $y(t_k)$ - priimtas signalas, $x_I(t_k), x_Q(t_k)$ – duomenų I ir Q komponentės, atitinkamai, t_k – diskretinis laikas.

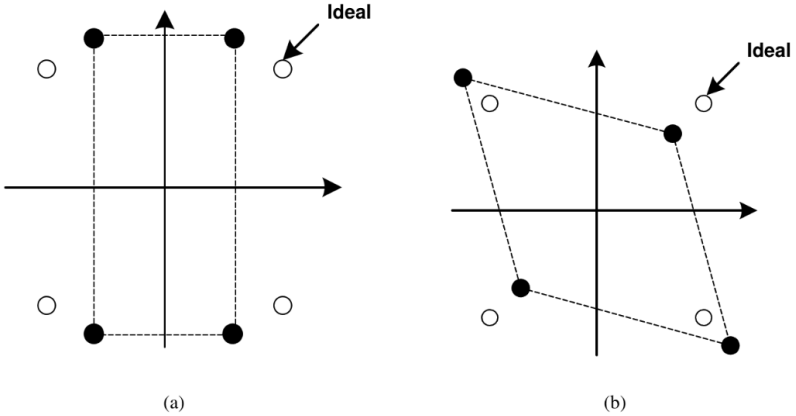
Aprašius amplitudinį disbalansą kaip $20\log[(1+\varepsilon_A)/(1-\varepsilon_A)]$ dB, o fazinį disbalansą kaip ε_θ laipsnių, priimtas, demoduliuotas duomenų dedamąsias galima aprašyti taip:

$$\tilde{x}_I(t_k, \varepsilon_A, \varepsilon_\theta) = (1 + \varepsilon_A)\left[x_I(t_k)\cos\left(\frac{\varepsilon_\theta}{2}\right) - x_Q(t_k)\sin\left(\frac{\varepsilon_\theta}{2}\right)\right] \quad (1.1.1.2)$$

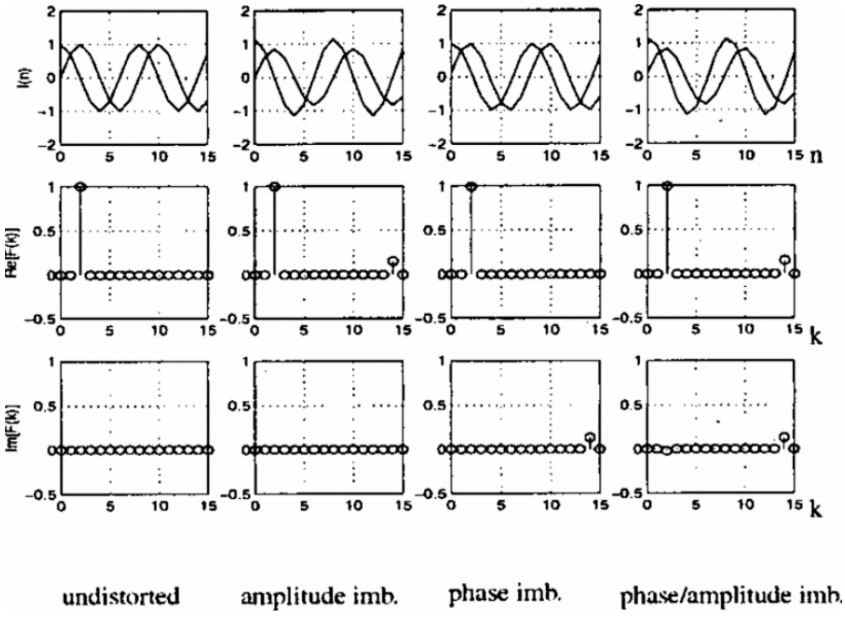
$$\tilde{x}_Q(t_k, \varepsilon_A, \varepsilon_\theta) = (1 - \varepsilon_A)\left[x_Q(t_k)\cos\left(\frac{\varepsilon_\theta}{2}\right) - x_I(t_k)\sin\left(\frac{\varepsilon_\theta}{2}\right)\right] \quad (1.1.1.3)$$

Čia $\tilde{x}_I(t_k, \varepsilon_A, \varepsilon_\theta)$ – IQ disbalanso paveikta duomenų I dedamosios imtis, $x_I(t_k)$ – tikroji duomenų I komponentės imtis, ε_A – amplitudinis disbalansas, ε_θ – fazinis disbalansas, $\tilde{x}_Q(t_k, \varepsilon_A, \varepsilon_\theta)$ – IQ disbalanso paveikta duomenų Q komponentės imtis, $x_Q(t_k)$ – tikroji duomenų Q komponentės imtis.

Iš šių išraiškų matyti, jog IQ disbalansas iškraipo priimamus duomenis[3]. Šis efektas pavaizduotas grafiškai 1.2. pav. Dažnių srityje šis iškraipymas pasireiškia kaip veidrodinio atspindžio atsiradimas : amplitudinio disbalanso atveju atspindys realus, fazinio disbalanso atveju atspindys menamasis[4].



1.2 pav. IQ disbalanso paveikti QPSK moduliuoti simboliai; (a) - amplitudinis disbalansas, (b) – fazinis disbalansas[4]



1.3 pav. IQ disbalanso efektai dažnių srityje [4]

1.1.2. Nepageidaujama nuolatinė dedamoji

Kitas ZeroIF architektūros trūkumas – nepageidaujamos nuolatinės dedamosios atsiradimas. Esant stipriai nuolatinei dedamajai išauga maksimali signalo amplitudė. Šiai amplitudei esant pakankamai didelei, signalo apdorojimo aparatūra įsisotina ir priimami duomenys yra iškraipomi.

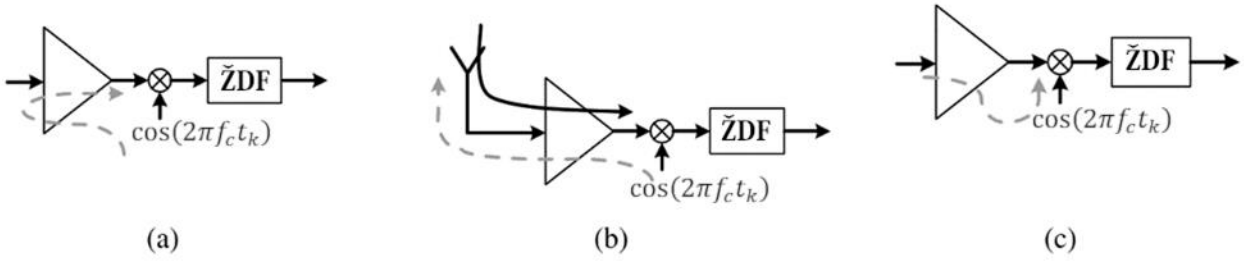
Nepageidaujama nuolatinė dedamoji gali atsirasti dėl dviejų priežasčių : komponentų netiesiškumo ir/arba dėl imtuvo generuojamo aukšto dažnio signalo susidauginimo su savimi (toliau - savimaiša).

Dėl netiesiškumo nuolatinė dedamoji atsiranda tuomet, kai imtuvo aparatūra pasižymi lyginio laipsnio netiesiškumu ir yra priimamas aukštos amplitudės signalas.

Savimaišos atveju nuolatinė dedamoji atsiranda, kai imtuvo vietinis aukšto dažnio signalas patenka į priimtam signalui skirtą įėjimą ir susidauginęs su savimi sukuria nuolatinę dedamąją. Tai gali nutikti vienu iš trijų būdų[3]:

a) Vietinis signalas indukuojasi prieš maišiklį ir maišiklyje yra sudauginamas su savimi.
b) Vietinis signalas indukuojasi antenoje, išsitransliuoja į aplinką, aplinkoje atsispindėjęs vėl patenka į anteną ir kartu su norimu signalu patekęs į maišiklį susidaugina su savimi.
c) Stiprus interferuojantis signalas gali indukuotis maišiklyje, įėjime skirtam vidiniam imtuvo aukšto dažnio signalui, ir susidauginti su savimi.

Žemiau pateiktoje iliustracijoje pavaizduoti visi šie trys variantai:



1.4 pav. Skirtingi savimaišos atvejai. Pritaikyta iš [3]

1.2. LTE fizinis lygmuo

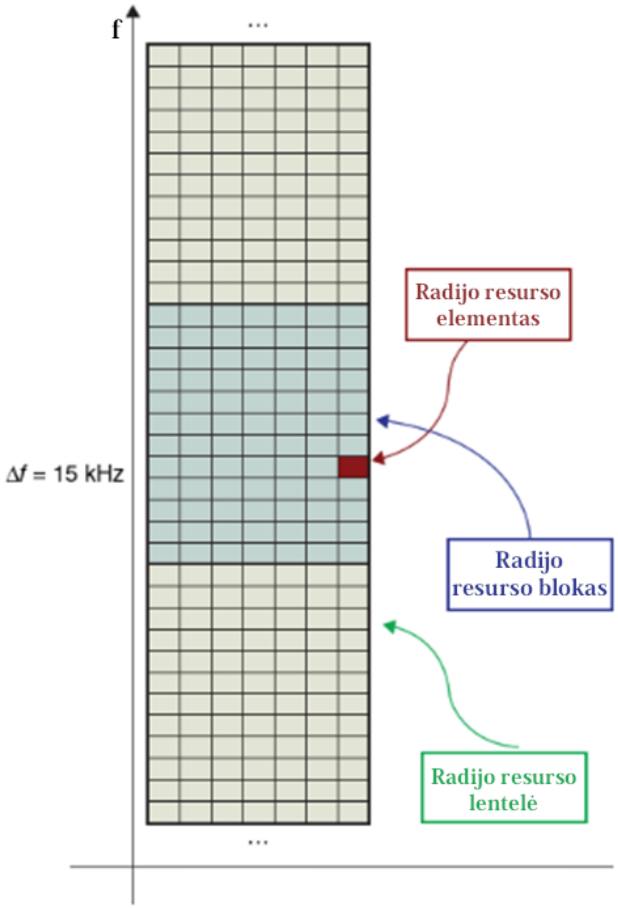
Siekiant derinimo uždavinį atlikti programuojamoje logikoje apdorojant grynus duomenis, reikėtų pirma aptarti LTE fizinį lygmenį.

1.2.1. Radijo resursas, jo paskirstymas

LTE standartas jam priskirtą dažnių ruožą panaudoja jį suskirstydamas į smulkesnius, nepriklausomus resursų blokus. Šie resursų blokai yra apriboti dažniu, bei laiku – 180kHz ir 0,5ms atitinkamai, bei yra dinamiškai priskiriami vartotojams, atsižvelgiant į ryšio kokybę ir/arba vartotojo

poreikius. LTE spektras gali būti sudarytas iš nuo 6 iki 100 resursų elementų, priklausomai nuo mobilaus ryšio operatoriui suteikto juostos pločio.

Resursų blokus sudaro mažesni resurso elementai, atspindintys dažnių srityje vieną 15kHz subnešančiąją, kurių resursų bloke yra 12, o laiko srityje vieno OFDM simbolio trukmę, kurių resursų bloke gali būti 6 arba 7[5].



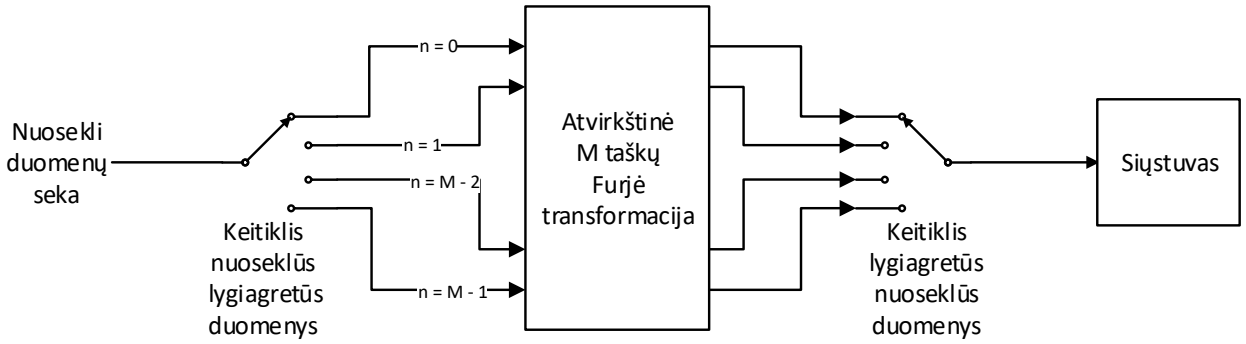
1.5 pav. LTE radijo resurso paskirstymas. Pritaikyta iš [5]

1.2.2. Žemynkryptis duomenų perdavimas

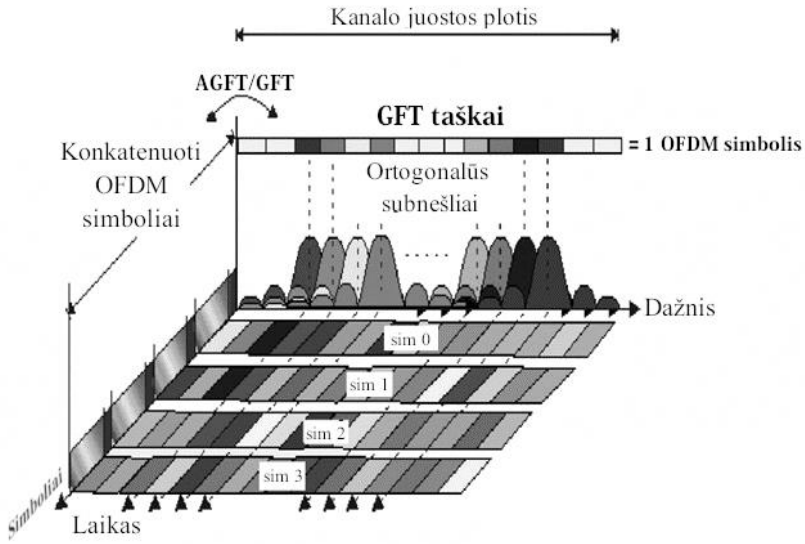
Žemynkryptis – tai terminas naudojamas apibūdinti duomenų perdavimą iš stoties vartotojui. LTE standartas abejoms kryptims naudoja šiek tiek skirtingus metodus. Pirma aptariamas žemynkripčiam ryšiui naudojamas metodas.

Remiantis 1.2.1. poskyryje pateikiama informacija, duomenų perdavimui gali prireikti iki 1200 subnešančiųjų. Akivaizdu, jog tai padaryti standartiniais metodais būtų nepraktiška, todėl LTE naudoja OFDM moduliaciją, kuri suformuoja sudėtingą signalą, kurio kiekviena dažninė dedamoji atspindi konkrečius perduodamus duomenis ir yra interpretuojama ne kaip bendra signalo dalis, bet

kaip atskira subnešančioji. OFDM moduliatoriaus struktūra pavaizduota 1.6 pav., iliustracijoje naudojama konstanta M atitinka naudojamų subnešančiųjų skaičių.



1.6 pav. OFDM moduliatoriaus struktūra



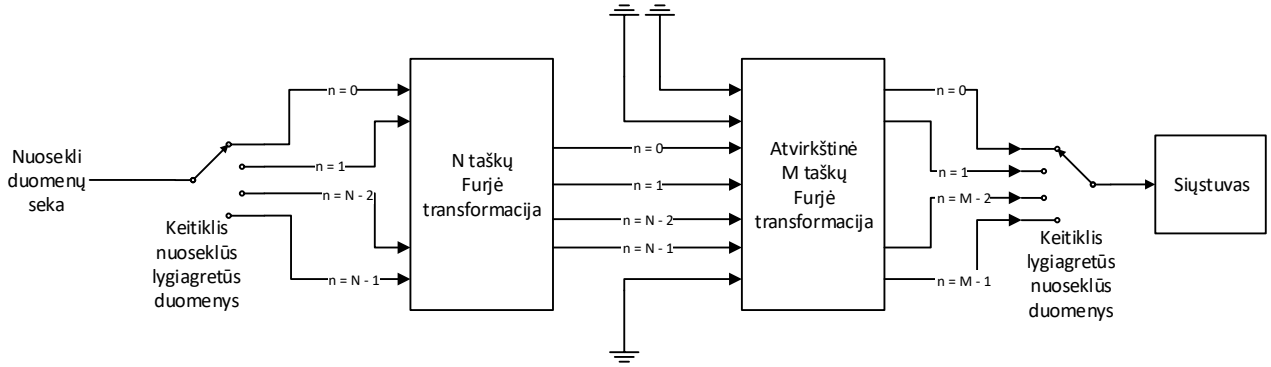
1.7 pav. OFDM signalo vaizdas. Pritaikyta iš [6]

Kaip matyti iš OFDM moduliatoriaus struktūros, didelis reikalingas nešančiųjų skaičius yra sukuriamas moduliuotus duomenis (duomenis jau praėjusius QPSK arba QAM moduliatorių) apdorojant atvirkštine Furjė transformacija. Transformacijos išėjime gaunamas laikinis signalas, kurio kiekviena dažninė dedamoji atspindi atskirą QPSK arba QAM simbolį[5].

1.2.3. Aukštynkryptis duomenų perdavimas

Aukštynkryptis – tai terminas naudojamas apibūdinti duomenų perdavimą iš vartotojo stočiai. Aukštynkrypčiui duomenų perdavimui OFDM moduliacija nepatraukli, kadangi jos sukuriami signalai pasižymi dideliu santykiu tarp momentinės galios pikų ir vidutinės galios vertės. Transliuojant tokius signalus pasiekiamas prastas siųstuvo efektyvumas, kas yra nesuderinama su poreikiu vartotojo aparatūrai naudoti kuo mažiau energijos, kadangi ji dažnai yra apribota baterijos talpa.

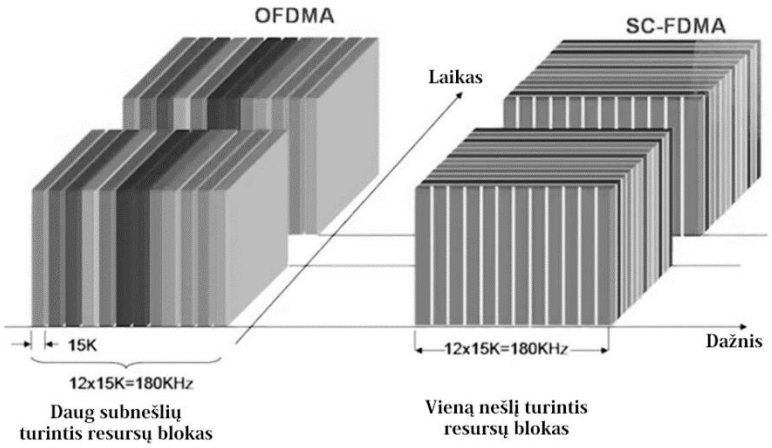
Šios problemos sprendimui vietoje OFDM naudojama jai gimininga SC-FDM moduliacija, kuri yra aparatiškai sudėtingesnė, tačiau padeda sumažinti siųstuvo sunaudojamos energijos kiekį. SC-FDM moduliatoriaus struktūra pateikta toliau. Struktūroje konstanta N atitinka vartotojui priskirtą subnešančiųjų skaičių, konstanta M atitinka visų kanalo naudojamų subnešančiųjų skaičių.



1.8 pav. SC-FDM moduliatoriaus struktūra

Iš moduliatoriaus struktūros matyti, jog esminis skirtumas tarp šio ir OFDM moduliatoriaus tai duomenims papildomai atliekama N taškų tiesioginė Furjė transformacija, bei atvirkštinės Furjė transformacijos ne visų įėjimų užpildymas (schemoje pavaizduota įžeminimu). Šio moduliatoriaus veikimą galima suskirstyti į du etapus:

Pirmuoju, N vartotojo duomenų simbolių yra perkeliami į dažnių sritį naudojant Furjė transformaciją. Antruoju, į dažnių sritį perkelti duomenys yra priskiriami prie N atvirkštinės Furjė transformacijos įėjimų, taip duomenis perkeliant atgal į laiko sritį (verta pastebėti, jog daugumoje atvejų N<M). Galutinio, į siųstuvą siunčiamo signalo dažninės charakteristikos tiesiogiai priklauso nuo to, kurie atvirkštinės Furjė transformacijos įėjimai buvo panaudoti. [5,7].



1.9 pav. Palyginimas tarp OFDM ir SC-FDM signalų. Pritaikyta iš [8]

Pastaba: įvairiuose šaltiniuose skiriasi ar minima OFDM ir SC-FDM ar OFDMA ir SC-FDMA. Šios abi sistemos tarpusavyje skiriasi tik radijo resurso perskirstymu – OFDMA ir SC-FDMA resursus gali perskirstyti lanksčiau, tačiau patys moduliavimo principai nesiskiria.

1.3. LTE protokolinis lygmuo

Kartu su vartotojo duomenimis vartojo įranga stočiai taip pat siunčia ir kontrolinę informaciją, nurodančią kaip vartotojo įranga suvokia ryšio kokybę tarp stoties ir jos pačios[5]. Dėl vartotojų judėjimo, oro sąlygų kaitos ir kitų išorinių faktorių šie kokybės vertinimai laikui bėgant kinta. Ilgesnį laiką vidurkinant skirtingų vartotojų ryšio įvertinimus turėtų sumažėti minėtų pašalinių faktorių įtaka gautam galutiniam rezultatai, taip gaunant koeficientą, kuris tiksliau atspindi stoties įrangos veikimo kokybę. Tačiau svarstant apie tokio metodo įgyvendinimą panaudojant programuojamą logiką susiduriama su keliomis problemomis :

1. Norint teisingai priimti duomenis, reikėtų programuojamoje logikoje įgyvendinti M dydžio Furjė transformaciją ($M \neq 2^n$), turint omenyje jog maksimalus M yra 1200, tam reikėtų labai didelių loginių elementų sąnaudų.
2. Taip pat reikėtų atlikti ir N dydžio atvirkštinę Furjė transformaciją kiekvienam vartotojui, kurio duomenis norima ištirti. Turint omenyje, jog kiekvieno vartotojo N reikšmė gali būti skirtinga, bei kad $N=x*12 \neq 2^n$, nėra galimybių šių skaičiavimų optimizuoti ir tam taip pat reikėtų didelių loginių elementų sąnaudų.
3. Kiekvienam vartotojui priskirtas dažnių ruožas ir jo plotis gali dinamiškai kisti dėl ryšio kokybės ar vartotojo poreikių pasikeitimo, dėl ko reikėtų jog aparatūra būtų lengvai ir greitai perderinama skirtingo dydžio transformacijų atlikimui. Tai yra sunkiai pasiekiama naudojant programuojamą logiką.

Kadangi LTE protokoliniame lygmenyje perduodamus konkrečius duomenis demoduliuoti naudojant programuojamą logiką yra nepraktiška, protokolinis lygmuo nebus aprašytas.

1.4. ZeroIF architektūros siųstuvo/imtuvo trūkumų šalinimo metodai

ZeroIF architektūros siųstuvai/imtuvai, dėl savo trūkumų (aprašyti poskyriuose 1.1.1. ir 1.1.2) yra problemiški juos naudojant situacijose, reikalaujančiose aukštos eilės moduliacijų, kaip LTE. Dėl šios priežasties architektūros derinimas (minėtų trūkumų šalinimas) yra literatūroje plačiai nagrinėjama tema. Literatūroje siūlomi sprendimai patenka į vieną iš dvejų kategorijų: derinimą, kuomet yra stengiamasi panaikinti pačio siųstuvo/imtuvo sukeliamus iškraipymus ir įvertinimą/kompensavimą – kuomet yra stengiamasi įvertinti bendrą duomenų iškraipymą priimtų duomenų sraute ir juos apdoroti sumažinant iškraipymo įtaką[9,10].

1.4.1. Literatūroje aprašomi ZeroIF derinimo metodai

Visi derinimo metodai, kurie buvo ištirti atliekant literatūros apžvalgą[9,10,11,12,13,14] iš principo yra labai panašūs. Šių metodų esmė – suderinti siųstuvą ir imtuvą taip, kad jie abu veiktų persiklojančiose dažnių ruožuose. Tai atlikus ištransliuojamas derinimui optimizuotas testinis signalas, kurį priėmus imtuvu pagal skirtumus tarp originalaus ir priimtojo signalų įvertinami siųstuve ir imtuve esantys iškraipymai. Įvertinus aparatūros sukeliamus iškraipymus į sistemą įvedamas papildomas, priešingas duomenų iškraipymas, kuris kompensuoja aparatūros neigiamą poveikį.

Daugumos šių metodų autoriai rekomenduoja tokį derinimą atlikti sistemos paleidimo metu ar kuomet yra mažas aktyvumas, kadangi atliekant derinimą negalima aparatūros naudoti pagal paskirtį – siųsti ar priimti naudingus duomenis. Tokį derinimą atliekant periodiškai, norint kompensuoti dinamiškai kintančius iškraipymus mažėja vidutinė siųstuvo/imtuvo kokybė (jei neveiksnumo periodą, kuomet atliekamas derinimas vertintume kaip periodą, kuomet visi duomenys ištransliuojami ir priimami su 100 % klaidų tikimybe).

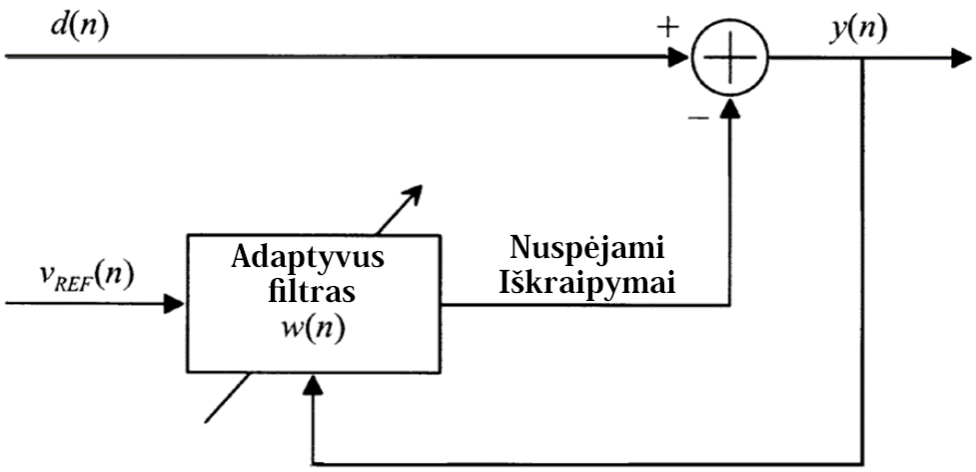
Skirtingi šios rūšies metodai tarpusavyje skiriasi savo naudojamais testiniais signalais ir algoritmais, kurie naudojami iškraipymams įvertinti, siekiant trumpesnio derinimo laiko, arba tikslesnio suderinimo.

1.4.2. Kompensaciniai metodai

Kompensaciniai ZeroIF architektūros trūkumų šalinimo metodai yra orientuoti pagrinde į imtuvo pusę ir jų tikslas yra – kokiu nors būdu įvertinti priimamų duomenų iškraipymą sukeliančius faktorius, jų amplitudę. Įvertinus šiuos faktorius priimti duomenys yra apdorojami, iš jų pašalinant iškraipymus. Skirtumas tarp šių metodų ir 1.4.1 skyrelyje aprašytų yra subtilus, bet reikšmingas – kompensaciniai metodai keičia jau priimtus duomenis, kuomet kiti metodai siekia padaryti kompensuojančią įtaką pačiai aparatūrai.

Daugelis metodų šią iškraipymus nusakančią informaciją siūlo išgauti panaudojant pilotinius signalus[15,16,17] t.y. iškraipymų nustatymui optimizuotus specialius signalus. Tokie metodai susiduria su tokiomis pačiomis problemomis, kaip ir metodai aprašyti 1.4.1. poskyryje.

Tačiau yra ir ne vien pilotinius signalus pasitelkiančių metodų. Yra ir metodas iš priimtų duomenų pašalinti iškraipymus panaudojant adaptyvųjį filtrą[18].



1.10 pav. Adaptyvusis filtras naudojamas signalo iškraipymams pašalinti. Pritaikyta iš [18]

Kaip žinoma naudojant adaptyvųjį filtrą reikalingi du įėjimo signalai – signalas, kurį norima filtruoti ir signalas, kuris koreliuoja su nepageidaujama signalo dedamąja, kurią norima pašalinti, tačiau nekoreliuoja, arba mažiau koreliuoja su naudingu signalu. Kaip apdorojamas signalas yra naudojami priimti duomenys, o kaip signalas, koreliuojantis su nepageidaujama dedamąja yra naudojamas priimto signalo kompleksinis jungtinis signalas (signalas su invertuota menamąja dalimi). Kompleksinis jungtinis signalas pasižymi savybe, kad jo spektras yra veidrodiškas atspindys originalaus signalo spektrui. Turint omenyje 1.1.1. poskyryje aprašytą I/Q disbalanso efektą dažnių srityje, nesunku pamatyti, kodėl toks signalas turėtų smarkiai koreliuoti su nepageidaujama signalo dalimi. Nesudėtinga pastebėti vieną iš ryškiausių šio metodo trūkumų – imtuvas, naudojantis tokią disbalanso kompensavimo sistemą nėra tinkamas priimti signalams, kurie pasižymi simetriškumu apie centrinį dažnį, kadangi adaptyvus filtras šias simetriškas dedamąsias pašalintų, taip iškraipydamas signalą. Verta paminėti, jog kadangi šio derinimo metu naudojamas filtras, kompensuojantis priimtuose duomenyse esantį disbalansą, šis metodas gali būti taikomas tik imtuvui.

1.5. Apibendrinimas

Iš literatūros apžvalgos matyti, jog LTE teikiamų tarnybinių duomenų panaudojimas sistemos derinimui yra nepraktiškas, kadangi tam reikalingi demoduliatoriai suvartotų itin daug programuojamos logikos resursų. Apžvelgus kitus, literatūroje aprašytus, derinimo metodus pastebėta, jog dauguma jų reikalauja prietaiso eksploatacijos nutraukimo, kas nėra suderinama su šio darbo uždaviniais. Paskutinis aptartas metodas – adaptyvaus filtro panaudojimas, kaip ir atitiktų išsikeltus šio darbo uždavinius, tačiau pasižymi keliais esminiais trūkumais – nėra tinkamas priimant signalus, pasižyminčius simetriškumu dažnių srityje, bei negali būti panaudojamas siųstuvo derinimui. Sekančiame skyriuje aprašomas derinimo metodas, galintis patenkinti visus išsikeltus darbo uždavinius, bei nepasižymintis minėtais trūkumais.

2. Derinimo metodo kūrimas

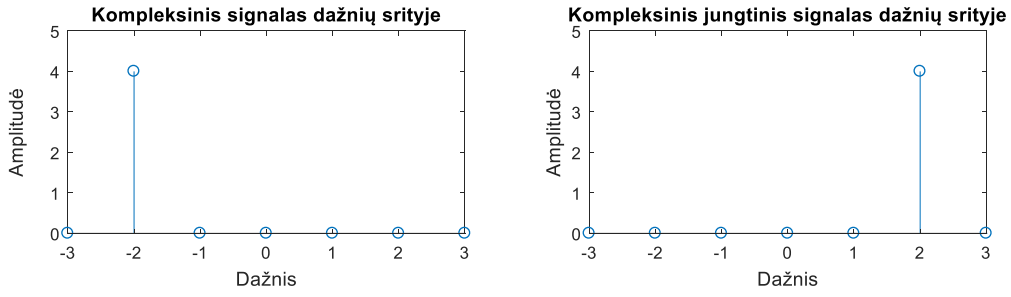
Daugelis derinimo metodų aprašų, kaip vieną svarbiausių metodo privalumų, akcentuoja laiką, reikalingą sistemos suderinimui, kadangi greitesnis derinimo atlikimas atitinka trumpesnį eksploatacijos pertraukimą. Kadangi šiame darbe aprašomas metodas veikia nepertraukdamas aparatūros eksploatacijos, šis parametras nėra tiek svarbus. Esant mažesnei minėto parametro svarbai, svarbiausiu metodo naudingumo vertinimo kriterijumi tampa metodo atliekamo derinimo tikslumas.

2.1. Derinimas, panaudojant koreliacijos koeficientą

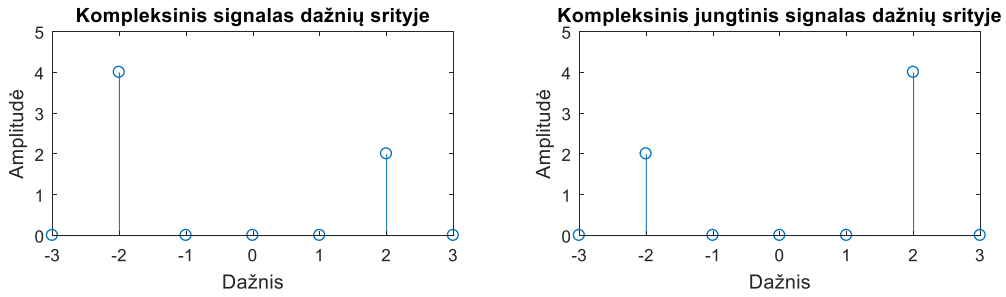
Pasak informacijos pateiktos 1.1.1. poskyryje, I/Q disbalansas dažnių srityje pasireiškia kaip tikrų, norimų transliuoti, duomenų veidrodinis atspindys. Kadangi veidrodinio atspindžio buvimas yra tiesiogiai susijęs su I/Q disbalansu, tai vienas būdų įvertinti disbalansą yra veidrodinio atspindžio įvertinimas.

Vienas būdų kaip tai padaryti – priimamus duomenis, panaudojant greitąją Furjė transformaciją, perkelti į dažnių sritį ir dažnių srityje paskaičiuoti koreliacijos koeficientą tarp teigiamų ir neigiamų dažnių. Esant veidrodiniam atspindžiui, teigiamų dažnių amplitudės kitimo dėsningumai atsispindi neigiamų dažnių srityje ir atvirkščiai. Dėl šios priežasties, esant atspindžiui koreliacijos koeficientas tarp abejų dažnių sričių yra didesnis, nei kad būtų įprastai.

Kitas, paprastesnis, metodas būtų skaičiuoti koreliacijos koeficientą tarp priimto kompleksinio signalo (I dedamoji = realioji dalis, Q dedamoji = menamoji dalis) ir jo kompleksinio jungtinio. Signalo kompleksinis jungtinis pasižymi savybe, jog jo teigiami ir neigiami dažniai yra apkeisti vietomis. Jeigu priimti duomenys yra paveikti IQ disbalanso, juose atsiradęs veidrodinis dažninių dedamųjų atspindys kompleksiniame jungtiniame signale turės tokią pačią poziciją, kaip ir tikroji dažninė dedamoji. Skaičiuojant koreliaciją tarp disbalanso paveikto signalo ir jo kompleksinio jungtinio koreliacijos koeficientas gaunamas didesnis, nei disbalansui nesant. Toliau pateiktose iliustracijose vaizduojami minėti dažninių dedamųjų pozicijų pasikeitimai (disbalansas įvestas naudojant formules (1.1.1.2) ir (1.1.1.3)



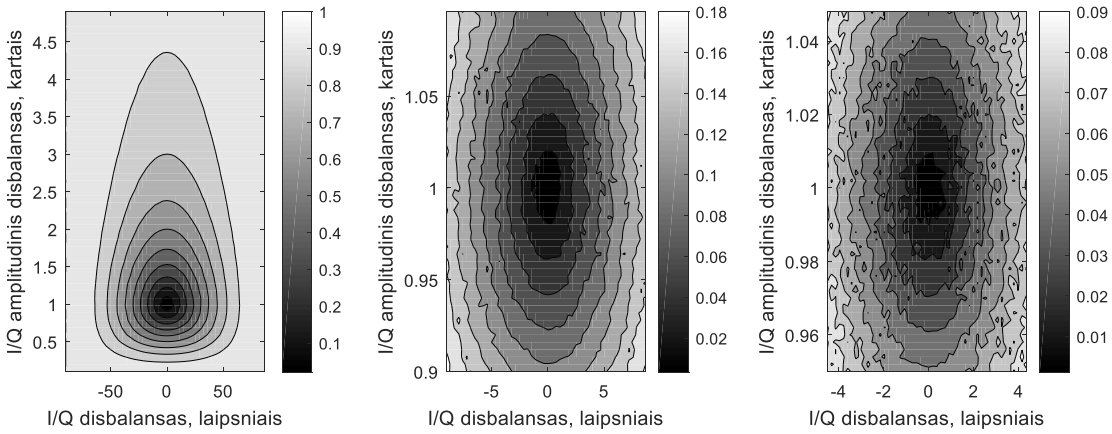
2.1 pav. Harmoninio kompleksinio signalo ir jo kompleksinio jungtinio spektrai be IQ disbalanso



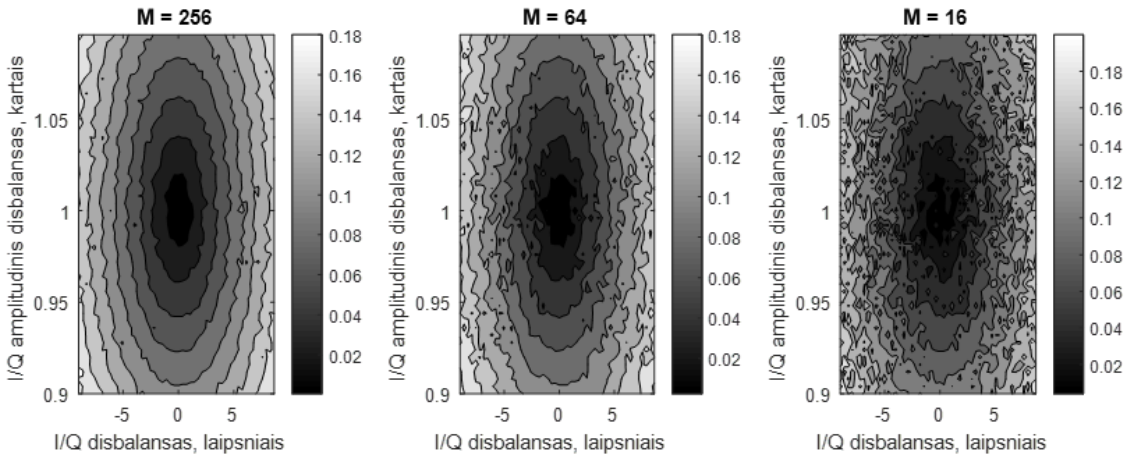
2.2 pav. Harmoninio kompleksinio signalo ir jo kompleksinio jungtinio spektrai su IQ disbalansu

Dėl palyginus sudėtingo Furjė transformacijos matematinio įgyvendinimo pasirinkta toliau nagrinėti antrą šiame poskyryje aprašytą metodą.

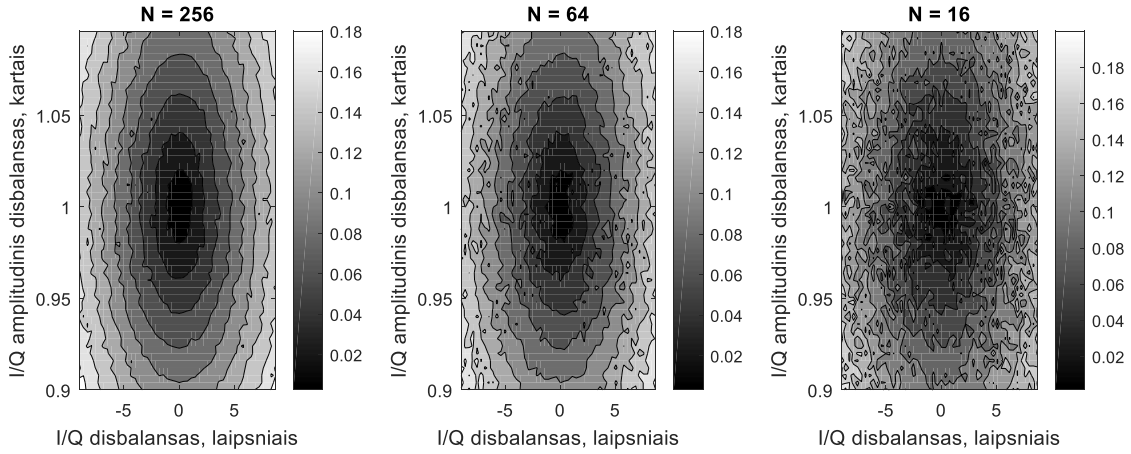
Šiam metodui išbandyti parašytas Matlab kodas generuojantis atsitiktinę tam tikro ilgio QPSK duomenų seką, kuri vėliau perskaičiuojama pagal išraiškas (1.1.1.2) ir (1.1.1.3) siekiant simuliuoti IQ disbalanso poveikį pradiniams duomenims. Skaičiavimai atliekami su skirtingomis fazinio ir amplitudinio disbalanso reikšmėmis. Su kiekviena fazinio ir amplitudinio disbalansų verčių kombinacija atliekamas iš anksto užsibrėžtas kiekis skaičiavimų, kurių metu kaskart sugeneruojami nauji duomenys. Atlikus minėtą kiekį skaičiavimų, randamas jų rezultatų vidurkis ir priskiriamas tai fazinio ir amplitudinio disbalansų kombinacijai. Tai atliekama siekiant iškompensuoti natūralias atsitiktiniuose duomenyse atsirandančias koreliacijos koeficiento variacijas. Toliau pateikiami modeliavimo rezultatai.



2.3 pav. Koreliacijos koeficiento priklausomybė nuo IQ disbalanso (vaizdai skiriasi disbalanso amplitudės rėžiais)



2.4 pav. Koreliacijos koeficiento priklausomybė nuo IQ disbalanso (keičiant vidurkinamų koeficientų skaičių)



2.5 pav. Koreliacijos koeficiento priklausomybė nuo IQ disbalanso (keičiant tiriamo signalo ilgį)

Grafikuose, esančiuose 2.3 pav., 2.4 pav. ir 2.5 pav. pavaizduotos koreliacijos koeficiento vertės gautos pagal formulę (2.1.1).

$$E[\rho(\varepsilon_A, \varepsilon_\theta)] = \frac{1}{M}\sum_{i=1}^{M} \rho(A(t_k, \varepsilon_A, \varepsilon_\theta)_i\, , \bar{A}(t_k, \varepsilon_A, \varepsilon_\theta)_i) \tag{2.1.1}$$

Čia $E[\rho(\varepsilon_A, \varepsilon_\theta)]$ – apskaičiuotų koreliacijos koeficientų vidurkis, kai amplitudės ir fazės disbalansai yra atitinkamai ε_A ir ε_θ, $\rho(A, \bar{A})$ – A ir $\bar{A}$ sekų tarpusavio koreliacijos koeficientas, $A(t_k, \varepsilon_A, \varepsilon_\theta)_i$ – i-toji sugeneruoja N imčių ilgio kompleksinių duomenų seka, M – vidurkinamų koreliacijos koeficientų skaičius, $\bar{A}$ – A sekos kompleksinis jungtinis, t_k – diskretinis laikas.

$$A(t_k, \varepsilon_A, \varepsilon_\theta) = (\, \tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta)_1 \;\; \tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta)_2 \;\; \ldots \;\; \tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta)_N \,) \tag{2.1.2}$$

$$\bar{A}(t_k, \varepsilon_A, \varepsilon_\theta) = (\, \bar{\tilde{y}}(t_k, \varepsilon_A, \varepsilon_\theta)_1 \;\; \bar{\tilde{y}}(t_k, \varepsilon_A, \varepsilon_\theta)_2 \;\; \ldots \;\; \bar{\tilde{y}}(t_k, \varepsilon_A, \varepsilon_\theta)_N \,) \tag{2.1.3}$$

Čia $A(t_k, \varepsilon_A, \varepsilon_\theta)$ – priimtų kompleksinių duomenų seka, kai amplitudės ir fazės disbalansai yra atitinkamai ε_A ir ε_θ, $\tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta)_1$ – pirmoji sekos imtis.

$$\tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta) = \tilde{x}_I(t_k, \varepsilon_A, \varepsilon_\theta) + j * \tilde{x}_Q(t_k, \varepsilon_A, \varepsilon_\theta) \quad (2.1.4)$$

$$\bar{\tilde{y}}(t_k, \varepsilon_A, \varepsilon_\theta) = \tilde{x}_I(t_k, \varepsilon_A, \varepsilon_\theta) - j * \tilde{x}_Q(t_k, \varepsilon_A, \varepsilon_\theta) \quad (2.1.5)$$

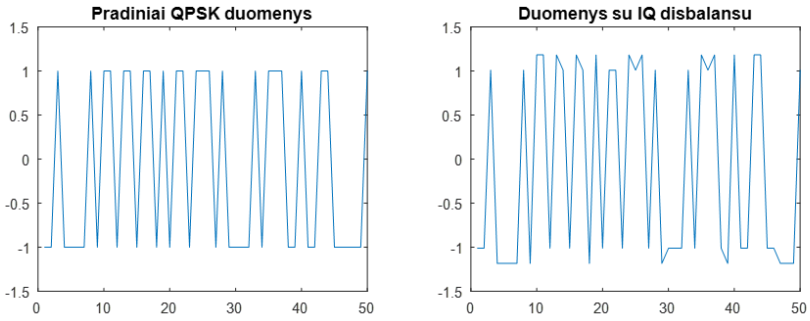
Čia $\tilde{y}(t_k, \varepsilon_A, \varepsilon_\theta)$ – priimtų kompleksinių duomenų imtis, o $\bar{\tilde{y}}(t_k, \varepsilon_A, \varepsilon_\theta)$ – priimtų kompleksinių duomenų kompleksinis jungtinis, kai amplitudės ir fazės disbalansai yra atitinkamai ε_A ir ε_θ, j – menamasis vienetas.

Vertės $\tilde{x}_I(t_k, \varepsilon_A, \varepsilon_\theta)$ ir $\tilde{x}_Q(t_k, \varepsilon_A, \varepsilon_\theta)$ yra apskaičiuojamos pagal formules (1.1.1.2) ir (1.1.1.3). Šiose formulėse naudojamos vertės $x_I(t_k)$ ir $x_Q(t_k)$ generuojamos naudojant toliau pateikiamą formulę (I ir Q dedamosios generuojamos naudojant tą pačią formulę).

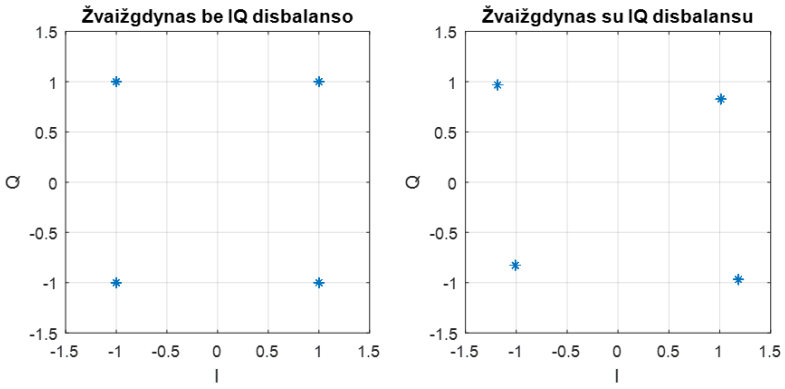
$$x_I(t_k) = x(R(t_k)) = \begin{cases} -1, & R < 0.5 \\ 1, & R \geq 0.5 \end{cases} \quad (2.1.6)$$

Čia $x(R)$ – generuojamo atsitiktinio QPSK simbolio dedamoji, R – tolygiojo skirstinio atsitiktinis skaičius, kurio vertė kinta ribose (0;1),

Toliau pateikiami sugeneruotų duomenų pavyzdžiai su ir be IQ disbalanso.



2.6 pav. Modeliavimo metu naudotų duomenų pavyzdys (I komponentė). IQ fazinis disbalansas 9 laipsniai, IQ amplitudinis disbalansas = 10 %



2.7 pav. Modeliavimo metu naudotų duomenų pavyzdys (žvaigždyno diagrama). IQ fazinis disbalansas 9 laipsniai, IQ amplitudinis disbalansas = 10 %

Iš grafikų pateiktų 2.3 pav. matyti, jog egzistuoja ryšys tarp IQ disbalanso ir koreliacijos koeficiento, bei kad šis ryšys neišnyksta ir esant mažesnės amplitudės I/Q disbalansams.

Iš grafikų pateiktų 2.4 pav. ir 2.5 pav. matyti koreliacijos skaičiavimo parametrų – analizuojamo signalo ilgio ir vidurkinamų koeficientų skaičiaus įtaką skaičiavimo rezultatams. Tai gali būti paaiškinama tuo, jog analizuojant ilgesnį signalą jo kitimo dėsnis darosi sudėtingesnis ir mažėja tikimybė, jog atsitiktinumo dėka signalas ir jo paties kompleksinis jungtinis reikšmingai koreliuos, o vidurkinant daugiau apskaičiuotų koeficientų mažėja bet kurių atsitiktinai per daug ar per mažai koreliuojančių signalų įtaka rezultatui. Iš to galima daryti išvadą, jog įgyvendinant šį metodą naudojant programuojamą logiką pagrindinis faktorius į kurį reikėtų atsižvelgti parenkant minėtų parametrų vertes yra loginių resursų sąnaudų augimas, susijęs su minėtų parametrų didinimu.

Verta paminėti, jog tiriant realius signalus konkreti koreliacijos koeficiento reikšmė neturi prasmės, kadangi skirtingi signalai dėl skirtingų moduliacijų, kodavimo, ar kitų priežasčių gali iš prigimties pasižymėti didesne ar mažesne koreliacija tarp teigiamų ir neigiamų dažnių. Siekiant užtikrinti kuo didesnį derinimo metodo universalumą reikėtų atsižvelgti išskirtinai į koeficiento pokytį.

2.2. Kompleksinio koreliacijos koeficiento skaičiavimas

Realus koreliacijos koeficientas

Ankstesniuose poskyriuose aptarus kompleksinės koreliacijos koeficiento naudojimą IQ disbalansui nustatyti reikėtų aptarti kaip būtent yra skaičiuojamas kompleksinis koreliacijos koeficientas ir kuo jis skiriasi nuo ne kompleksinio. Koreliacijos koeficientas skaičiuojamas pagal formulę:

$$\rho(A,B) = \frac{cov(A,B)}{\sigma(A) * \sigma(B)} \tag{2.2.1}$$

Čia ρ – koreliacijos koeficientas, cov(A,B) – A ir B duomenų sekų kovariacija, σ (A) – A duomenų sekos standartinė deviacija.

Koreliacijos koeficientui reikalingi koeficientai (kovariacija ir standartinė deviacija) skaičiuojami pagal formules :

$$cov(A,B) = \frac{\sum(A_i - E[A]) * (B_i - E[B])}{n} \tag{2.2.2}$$

$$\sigma(A) = \sqrt{Var(A)} = \sqrt{\frac{\sum(A_i - E[A])^2}{n}} \tag{2.2.3}$$

Čia A_i – i-tasis A duomenų sekos narys, $E[A]$ – A duomenų sekos reikšmių vidurkis, n -duomenų sekos narių skaičius, Var(A) – A duomenų sekos variacija, cov(A,B) – A ir B duomenų sekų kovariacija, σ (A) – A duomenų sekos standartinė deviacija.

Verta paminėti, jog statistikoje dažnai vietoje dalybos iš n naudojama dalyba iš (n+1) – tai yra naudojama siekiant pašalinti skaičiavimų metu atsirandančią adityvinę paklaidą. Kadangi siekiant įvertinti IQ disbalanso pokytį panaudojant koreliacijos koeficientą svarbi ne tiksli koreliacijos koeficiento reikšmė, o jos pokytis, formulės supaprastinamos ir naudojama dalyba iš n.

Kompleksinis koreliacijos koeficientas

Skaičiuojant koreliacijos koeficientą tarp kompleksinių duomenų sekų yra naudojamos šiek tiek kitokios kovariacijos ir variacijos išraiškos, kurios yra pateikiamos žemiau:

$$cov(A,B) = \frac{\sum(A_i - E[A]) * \overline{(B_i - E[B])}}{n} \tag{2.2.4}$$

$$Var(A) = \frac{\sum(A_i - E[A]) * \overline{(A_i - E[A])}}{n} \tag{2.2.5}$$

Čia $\overline{X}$ – X kintamojo kompleksinis jungtinis, A_i – i-tasis A duomenų sekos narys, $E[A]$ – A duomenų sekos reikšmių vidurkis, n -duomenų sekos narių skaičius, Var(A) – A duomenų sekos variacija, cov(A,B) – A ir B duomenų sekų kovariacija.

Svarbu pastebėti, jog kompleksinės duomenų sekos variacija yra realus skaičius, kadangi skaičiaus ir jo kompleksinio jungtinio sandaugos rezultatas yra realus skaičius.

2.2.1. Kompleksinio koreliacijos koeficiento skaičiavimas tarp signalo ir kompleksinio jungtinio

Skaičiuojant kompleksinį koreliacijos koeficientą tarp kompleksinių duomenų sekos ir jos kompleksinio jungtinio, koreliacijos koeficiento skaičiavimo išraiška supaprastėja dėl to, jog koreliuojant šis dvi sekas galima išprastinti dalį matematinių veiksmų. Išraiškos pateikiamos žemiau:

$$\sigma(A) * \sigma\left(\overline{A}\right) = \sigma(A)^2 = \sqrt{Var(A)}^2 = Var(A) \tag{2.2.6}$$

$$cov\left(A, \overline{A}\right) = \frac{\sum(A_i - E[A]) * \overline{\left(\overline{A_i} - \overline{E[A]}\right)}}{n} = \frac{\sum(A_i - E[A])^2}{n} \tag{2.2.7}$$

Čia $\overline{X}$ – X kintamojo kompleksinis jungtinis, A_i – i-tasis A duomenų sekos narys, $E[A]$ – A duomenų sekos reikšmių vidurkis, n -duomenų sekos narių skaičius, Var(A) – A duomenų sekos variacija, cov(A,B) – A ir B duomenų sekų kovariacija, σ (A) – A duomenų sekos standartinė deviacija.

Tuomet koreliacijos koeficientas apskaičiuojamas pagal formulę:

$$\rho(A,\overline{A}) = \frac{cov(A,\overline{A})}{Var(A)} = \frac{\frac{\sum(A_i - E[A])^2}{n}}{\frac{\sum(A_i - E[A]) * \overline{(A_i - E[A])}}{n}} \quad (2.2.8)$$

Čia $\overline{X}$ – X kintamojo kompleksinis jungtinis, A_i – i-tasis A duomenų sekos narys, $E[A]$ – A duomenų sekos reikšmių vidurkis, n -duomenų sekos narių skaičius, Var(A) – A duomenų sekos variacija, cov(A,B) – A ir B duomenų sekų kovariacija, ρ – koreliacijos koeficientas.

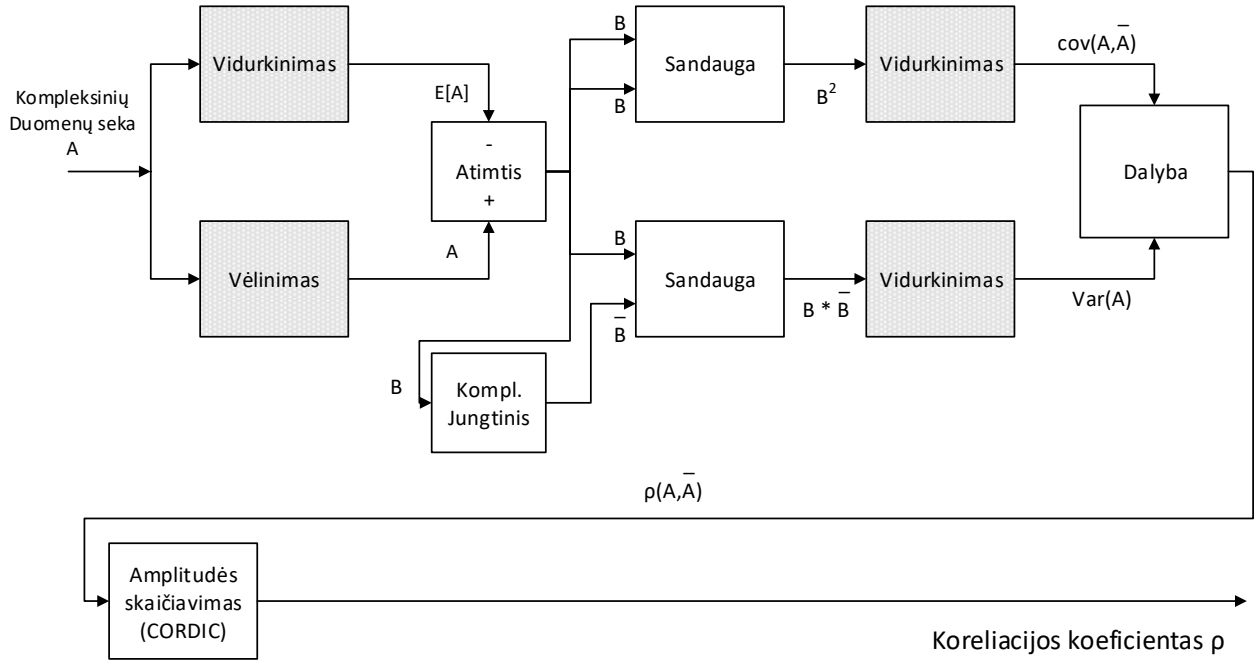
Išreiškus $B = (A_i - E[A])$ išraiška (1.5.8) atrodo taip:

$$\rho(A,\overline{A}) = \frac{\sum B^2}{\sum B * \overline{B}} \; arba \; \frac{\frac{\sum B^2}{n}}{\frac{\sum B * \overline{B}}{n}} \quad (2.2.9)$$

Čia ρ – koreliacijos koeficientas, $\overline{X}$ – X kintamojo kompleksinis jungtinis, n -duomenų sekos narių skaičius.

2.3. Koreliacijos koeficiento tarp kompleksinės sekos ir jos kompleksinio jungtinio įgyvendinimas programuojamoje logikoje

Norint koreliacijos koeficiento skaičiavimą įgyvendinti programuojamoje logikoje, reikia stengtis tai atlikti naudojant kuo mažiau perteklinių operacijų, taip taupant programuojamos logikos loginius resursus. Sugalvotos skaičiavimo struktūros schema pateikta toliau:



2.8 pav. Kompleksinės koreliacijos skaičiavimo programuojamoje logikoje struktūra

Iš 2.8 pav. struktūros matyti, jog pasirinkta (2.2.9) formulėje pateikta antra koreliacijos koeficiento išraiška (su vidurkinimu), nors tai yra perteklinis skaičiavimo veiksmas, tačiau jis sumažina bitų prieaugio problemą sumuojant sandaugų rezultatus.

Po dalybos blokelio esantis CORDIC blokelis yra skirtas be sandaugos ir šaknies traukimo operacijų apskaičiuoti kompleksinio koreliacijos koeficiento amplitudę.

Struktūroje blokeliai, kurių logikos elementų sąnaudos auga didinant tiriamo signalo ilgį pažymėti tamsiau. Iš pažymėtų blokelių daugiausiai resursų sunaudoja vėlinimo elementas, kadangi jis yra įgyvendinamas kaip atmintis, sauganti tiek kompleksinių imčių, iš kiek susidaro tiriama duomenų seka.

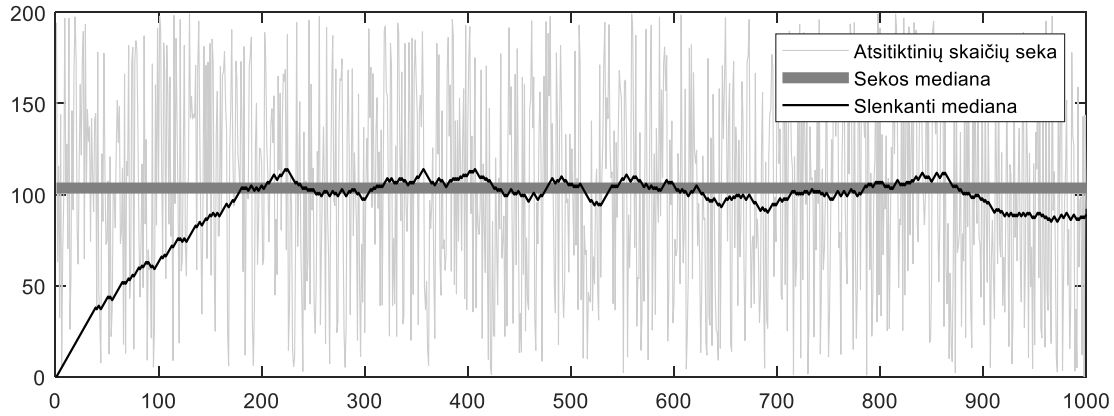
2.3.1. Skaičiavimo rezultatų apdorojimas

Atsižvelgus į informaciją pateiktą 2.5 pav. ir 2.8 pav. matyti, jog norint pasiekti ryškesnę priklausomybę tarp koreliacijos koeficiento ir sistemos I/Q disbalanso reikia arba koeficiento skaičiavimui naudoti ilgesnę duomenų seką, arba papildomai apdoroti rezultatus. Kadangi tiriamos duomenų sekos ilginimas yra susijęs su ženkliai didesnėmis loginių elementų sąnaudomis, nuspręsta naudoti papildomą koeficientų apdorojimą. Nors 2.1 poskyryje naudojamas vidurkinimas, tačiau šią struktūrą įgyvendinus naudojant programuojamą logiką ir išbandžius su realiais duomenimis buvo pastebėta, jog galutiniams sistemos rezultatams pastebimą įtaką daro smarkiai nuo rezultatų vidurkio nukrypusios koreliacijos koeficiento reikšmės, atsirandančios dėl natūraliai ir nenuspėjamai laike kintančio realių duomenų koreliavimo. Vienas iš šios problemos sprendimo būdų – vietoje vidurkio naudoti medianą, tačiau kadangi standartinis medianos skaičiavimo algoritmas – imčių saugojimas, išrūšiavimas ir vidurinės reikšmės išskyrimas reikalautų didelio kiekio programuojamos logikos resursų, buvo nuspręsta naudoti slenkančią medianą[19], kurios veikimo principas toliau pateikiamas formule.

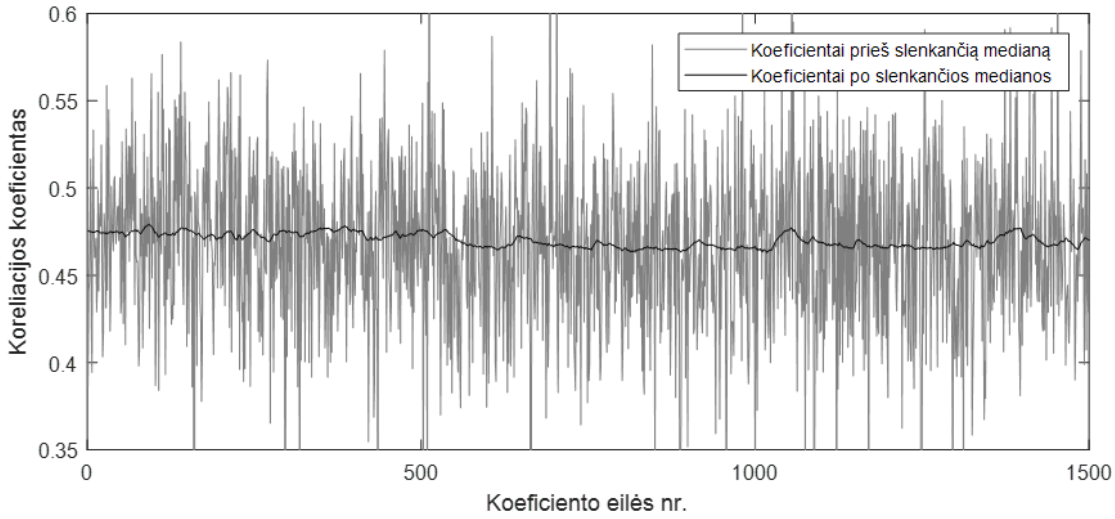
$$SM\rho_i = \begin{cases} SM\rho_{i-1} + \Delta, & \rho_i > SM\rho_{i-1} \\ SM\rho_{i-1}, & \rho_i = SM\rho_{i-1} \\ SM\rho_{i-1} - \Delta, & \rho_i < SM\rho_{i-1} \end{cases} \quad (2.3.1)$$

Čia $SM\rho_i$ – slenkančia mediana apdorota i-oji koreliacijos koeficiento vertė, Δ - slenkančios medianos delta vertė (laisvai pasirenkama), ρ_i – i-tasis apskaičiuotas koreliacijos koeficientas.

Iš formulės (2.3.1) matyti, jog nors po šios operacijos gaunamos reikšmės neatitinka medianos apibrėžimo, bet aproksimuoja tikrąją medianos vertę. Siekiant iliustruoti slenkančios medianos veikimą, buvo atliktas paprastas jos modeliavimas, naudojant tūkstančio pseudo-atsitiktinių skaičių, kurių amplitudė yra tarp 0 ir 200 seką, kai delta yra 1, o slenkančios medianos pradinė reikšmė lygi 0. Taip pat, slenkanti mediana išbandyti su realiais duomenimis, apdorojant koreliacijos koeficientus, gautus iš aukštynkrypčio LTE duomenų srauto. Toliau grafiškai pateikiami rezultatai.



2.9 pav. Slenkančios medianos modeliavimo rezultatai



2.10 pav. Slenkančios medianos poveikis realiems duomenims

Norint derinimo metu įvertinti kaip sistemos parametrų keitimas daro įtaką skaičiavimų rezultatams, reikėtų šiuos rezultatus papildomai apdoroti. Šiame darbe tam pasirinktas paprastas metodas – kiekvieną naują rezultatą lyginti su prieš tai buvusiu ir šiuos skirtumus sumuoti. Ši suma kaskart pakeičiant parametrą, darantį įtaką I/Q disbalansui yra ištrinama. Taip apdorojant duomenis yra aiškiau išskiriami lėtesni, dėsningi, dėl parametrų pakeitimo atsirandantys, koeficiento pokyčiai, taip išskiriant reikšmės kitimo tendenciją. Šio apdorojimo matematinė išraiška pateikta (2.3.1.1) formulėje.

$$DK = \sum_{i=1}^{M} SM\rho_i - SM\rho_{i+1} \qquad (2.3.1.1)$$

Čia DK – derinimo koeficientas (tolimesniam derinimui naudojamas įvertis), $SM\rho_i$– slenkančia mediana apdorotas i-tasis koreliacijos koeficientas, M – laisvai keičiamas parametras, nurodantis sumos narių kiekį.

2.4. Derinimą valdantis modulis

Derinimą valdančio modulio principas yra ganėtinai paprastas – kompensacinis disbalansas pirma padidinamas maža amplitude, tada sumažinamas. Po kiekvieno pakeitimo fiksuojama derinimo koeficiento (DK) vertė. Atlikus abu pakeitimus gauti DK yra palyginami ir nustatoma nauja kompensacinio disbalanso vertė.

Nors labiau išderintoje sistemoje dažniausiai gaunamas didesnis DK (2.11 pav., 2.12 pav.), tačiau smarkiai nesiskiriančio disbalanso sistemos pasižymi ženkliai persiklojančiais DK diapazonais (2.13 pav.). Siekiant sumažinti šio persiklojimo įtaką, prieš tai aprašyta modulio veikimo eiga pakartojama kelis kartus, kaskart inkrementuojant ar dekrementuojant vidinį kintamąjį K, priklausomai nuo to ar kompensacinio disbalanso padidinimas davė geresnį ar prastesnį rezultatą, atitinkamai. Pasiekus norimą iteracijų skaičių sprendžiama ar keisti pradinę kompensacinio disbalanso vertę pagal tai ar kintamojo K vertė viršijo nusistatytą slenkstinę vertę *thr*. Atlikus sprendimą procesas kartojamas iš naujo. Šis valdymo algoritmas grafiškai pavaizduotas 2.14 pav.

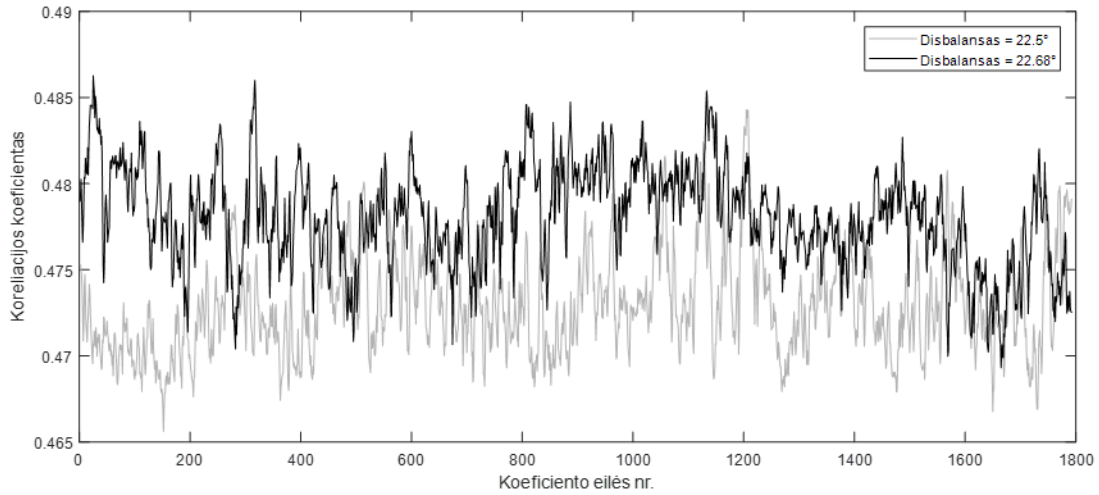
Algoritme pateikiamų simbolių paaiškinimas:

- ε_k – kompensacinis disbalansas (pvz. I kanalo stiprinimas amplitudinio disbalanso atveju, ar papildomas fazių skirtumas tarp kanalų fazinio disbalanso atveju).
- ε_{kp} – pradinė kompensacinio disbalanso vertė.
- K – vidinis kintamasis, saugantis derinimo proceso rezultatus.
- thr – slenkstinė vertę, kurią viršijus atliekamas ε_{kp} pakeitimas.
- DK – derinimo koeficientas.
- Δ – kompensacinio disbalanso keitimo žingsnis.

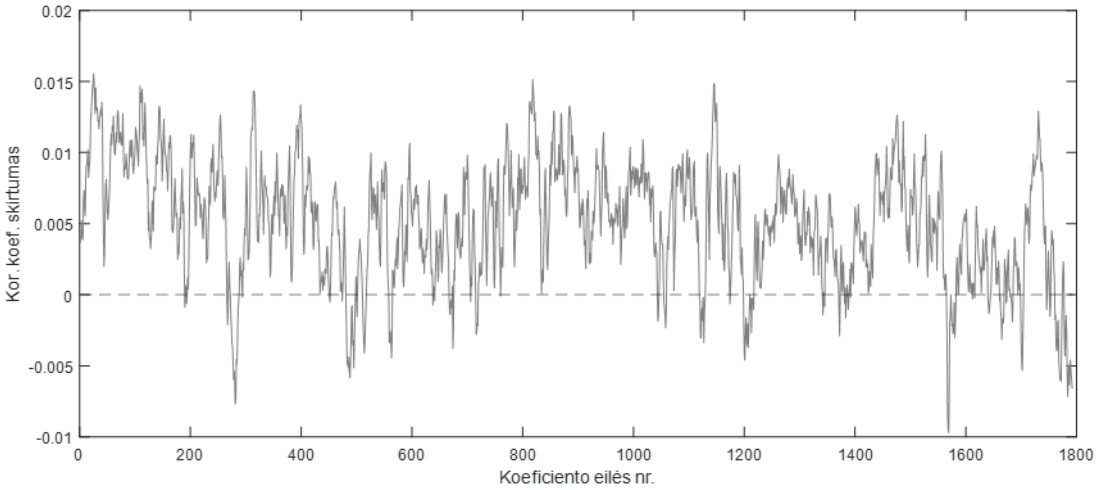
Toliau pateikiama sistemos pilno balanso sąlyga.

$$\varepsilon_k + \varepsilon = 0 \tag{2.4.1}$$

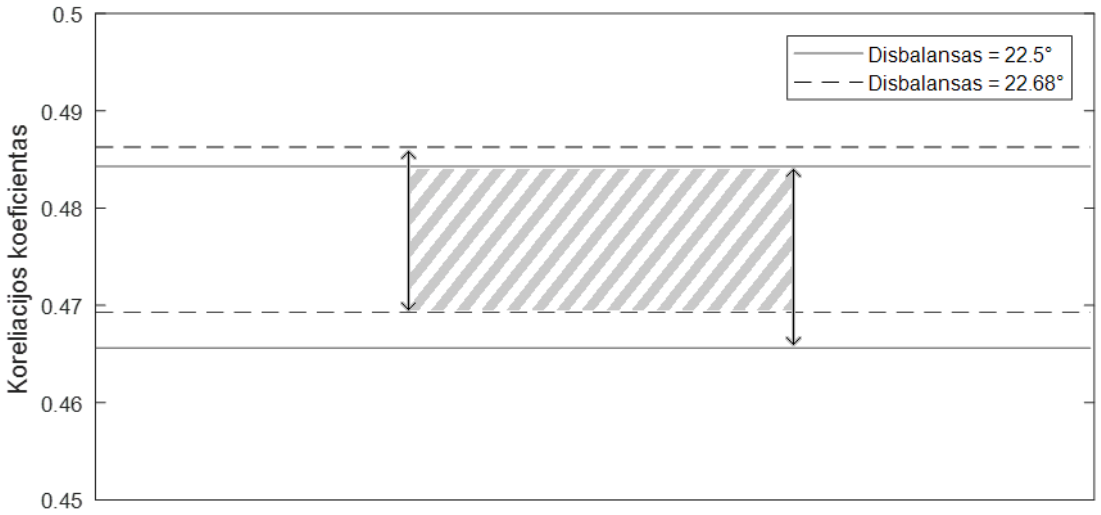
Čia ε_k – kompensacinis disbalansas, ε – tiriamos sistemos disbalansas.



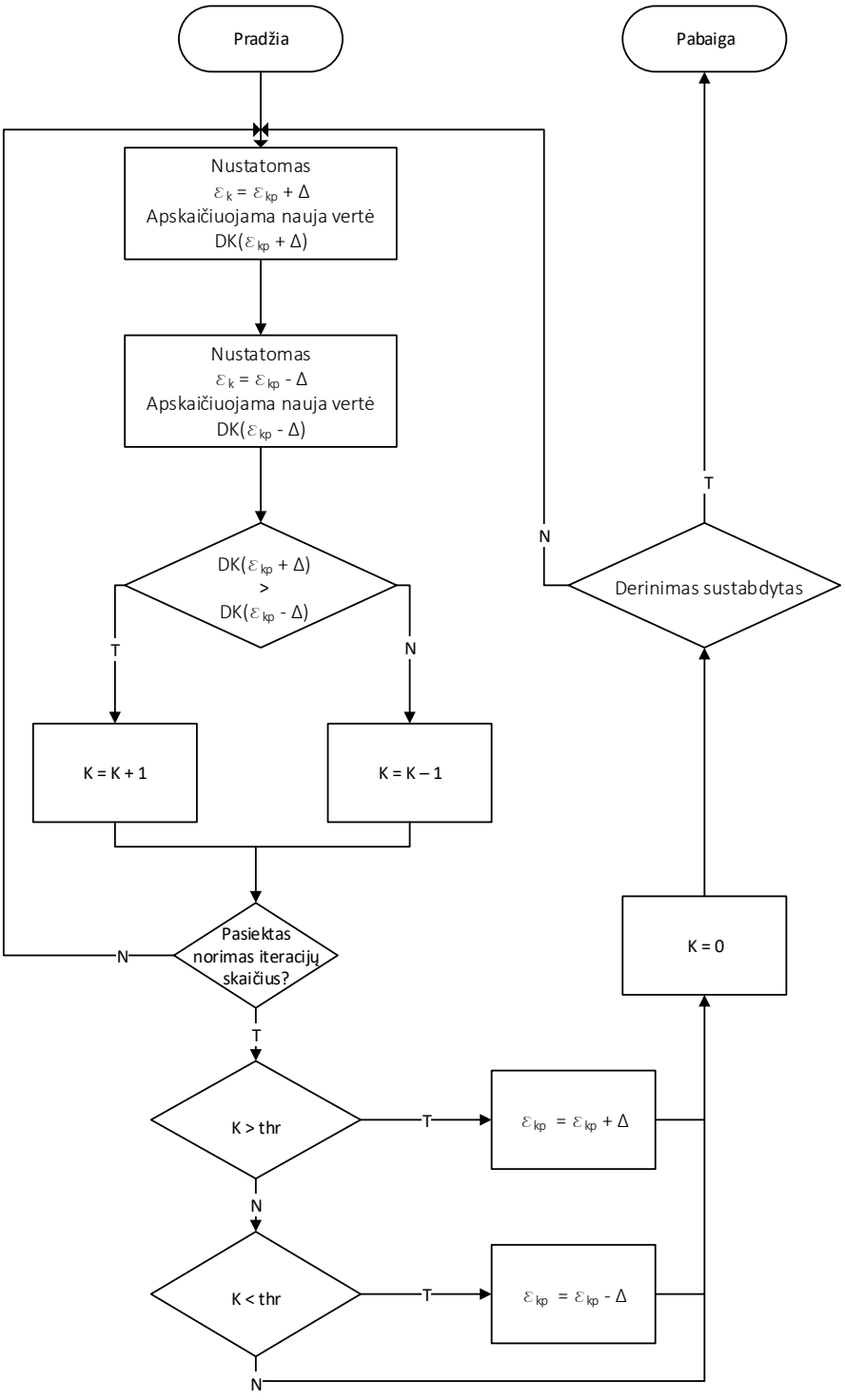
2.11 pav. Momentinės koreliacijos koeficiento vertės kitimas laike, esant artimoms fazinio disbalanso vertėms (duomenys gauti priimant aukštynkryptį LTE duomenų srautą)



2.12 pav. Skirtumas tarp kreivių, pavaizduotų 2.11 pav.



2.13 pav. Persidengimas tarp 2.11 pav. pavaizduotų kreivių verčių diapazonų



2.14 pav. Derinimą valdančio modulio veikimo algoritmas

2.5. Nepageidaujamos nuolatinės dedamosios pašalinimas

Pagrindinis šio metodo privalumas – galimybė derinimui naudoti realius, o ne išskirtinai derinimui sukurtus duomenis sukuria ir vieną jo trūkumų. Nežinant kokios amplitudės nuolatinė dedamoji turi būti priimtuose duomenyse nėra galimybės įvertinti ir pašalinti nepageidaujamą nuolatinės dedamosios dedamąją, aprašytą 1.1.2 skyrelyje.

3. Sukurto metodo tyrimas

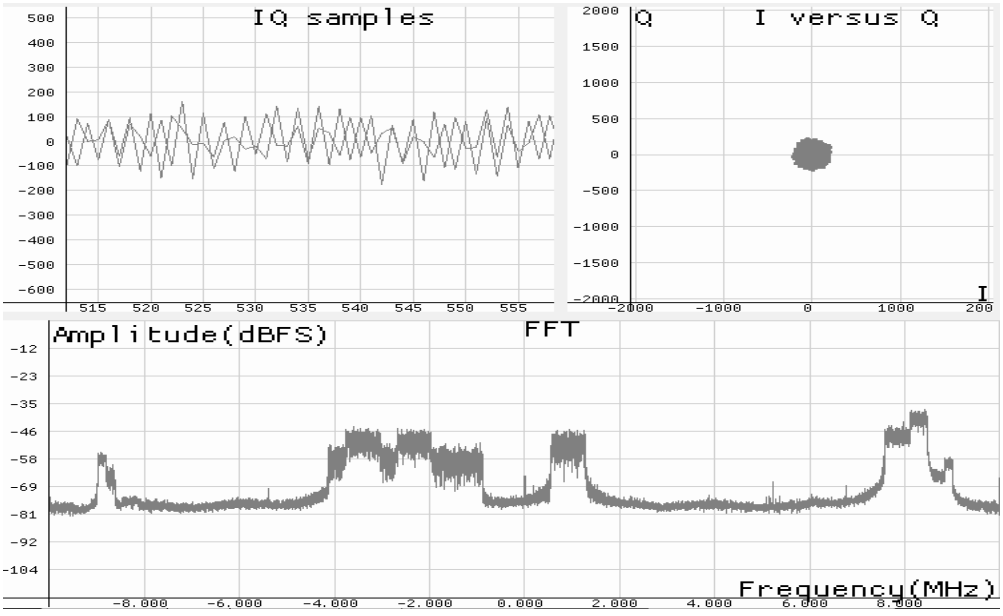
Praeitame skyriuje aprašytas metodas buvo įgyvendintas Lime Microsystems plokštėje LimeSDR-USB (įgyvendinimas aprašytas 3.1 poskyryje). Šiame skyriuje pateikiami metodo bandymo panaudojant tikrus duomenis rezultatai.

Prieš atliekant tyrimą išskirti šie parametrai, galintys daryti įtaką derinimo rezultatams:
1. Tiriamos duomenų sekos ilgis.
2. Slenkančios medianos delta vertė.
3. Sumuojamų koeficientų kiekis.
4. Derinimo metu atliekamų iteracijų skaičius.
5. Kiekvienos iteracijos metu atliekamo parametro keitimo žingsnis.
6. Derinimo sprendimo priėmimo slenkstis.

Parametras 1 plačiau aprašytas 2.1 ir 2.3 poskyriuose. Parametrai 2 ir 3 plačiau aprašyti 2.3.1 skyrelyje. Parametrai 4, 5 ir 6 plačiau aprašyti 2.4 poskyryje.

Tyrimams pagal nutylėjimą naudojami šie parametrai:
1. Centrinis dažnis – 1775MHz (LTE aukštynkrypčio ryšio juostos, aktyvios tyrimo vietoje, centrinis dažnis).
2. Juostos plotis – 20MHz.
3. Tiriamos duomenų sekos ilgis – 256 imtys.
4. Slenkančios medianos delta vertė – 1 kvantas.
5. Sumuojamų koeficientų kiekis – 512.
6. Derinimo metu atliekamų iteracijų skaičius – 64.
7. Kiekvienos iteracijos metu atliekamo parametro keitimo žingsnis – 2 kvantai (fazių disbalanso atveju ≈ 0.088 laipsnio)
8. Derinimo sprendimo priėmimo slenkstis – 16.

Toliau pateikiamas pavyzdys sistemos, suderintos šiam dažniui, priimtų 16384 imčių. Šis signalas pasižymi staigiais pokyčiais. Daugiau pavyzdžių pateikta 1 priede.



3.1 pav. SDR sistemos priimtų aukštynkrypčio LTE ryšio duomenų pavyzdys

Šiame skyriuje pateiktų, konvergavimo proceso eigą vaizduojančių, grafikų kūrimui duomenys yra surenkami panaudojant Intel-Altera loginio analizatoriaus IP šerdį. Ši šerdis prijungiama prie 2.4 poskyryje aprašyto derinimo modulio. Kaskart moduliui pasiekus nustatytą iteracijų skaičių loginis analizatorius užfiksuoja koks parametro keitimo sprendimas yra priimamas. Kiekvienas šių sprendimų grafikuose įvardijamas kaip derinimo ciklas. Turint taip užfiksuotą sprendimų priėmimo istoriją, pradinę parametro vertę, bei žingsnį, kuriuo parametras keičiamas, nesudėtingai sukuriamas vektorius, atspindintis konvergavimo proceso eigą.

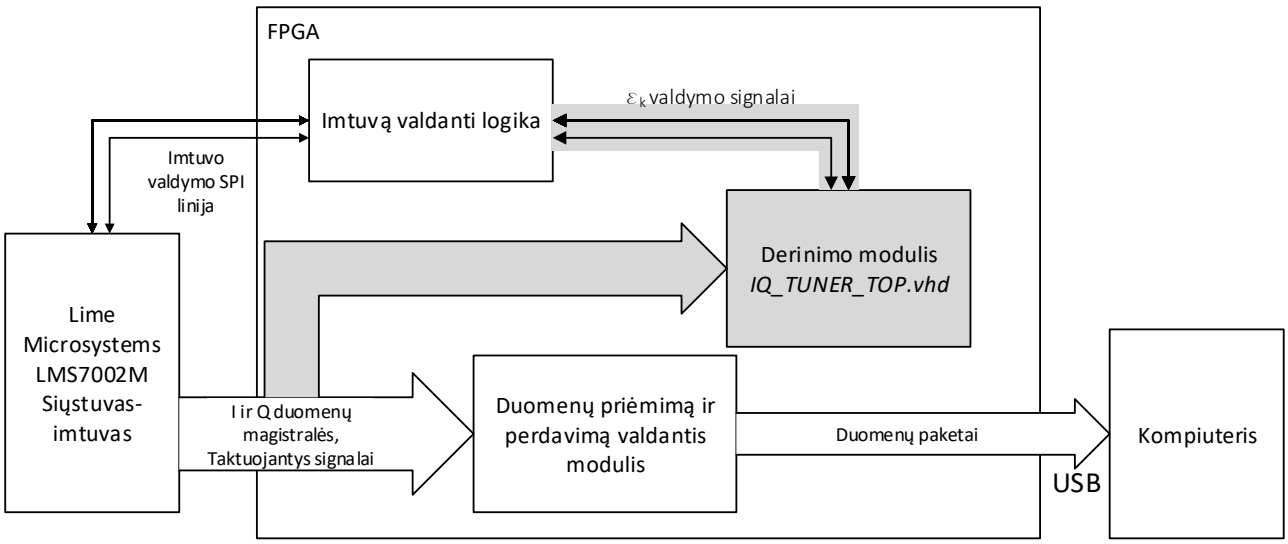
Derinimo procesas skirstomas į dvi dalis – grubų derinimą ir tikslų derinimą. Šios dalys apibrėžiamos taip:

- Grubus derinimas – derinimo dalis, kuomet fazinis disbalansas yra didesnis nei 1°.
- Tikslus derinimas – derinimo dalis, kuomet disbalanso vertės standartinė deviacija nekinta (realiuose duomenyse laikoma, jog tikslus derinimas prasidėjo, kai momentinė standartinė deviacija, skaičiuojama iš 1000 imčių nors vieną ciklą nekinta).

Rezultatuose pateiktas disbalansas apskaičiuotas pagal skirtumą tarp momentinės ε_k ir ε_k vertės, kurią, kaip disbalansą pilnai pašalinančią, nustatė LimeSDR-USB valdymui skirtos programinės aparatūros derinimo funkcija.

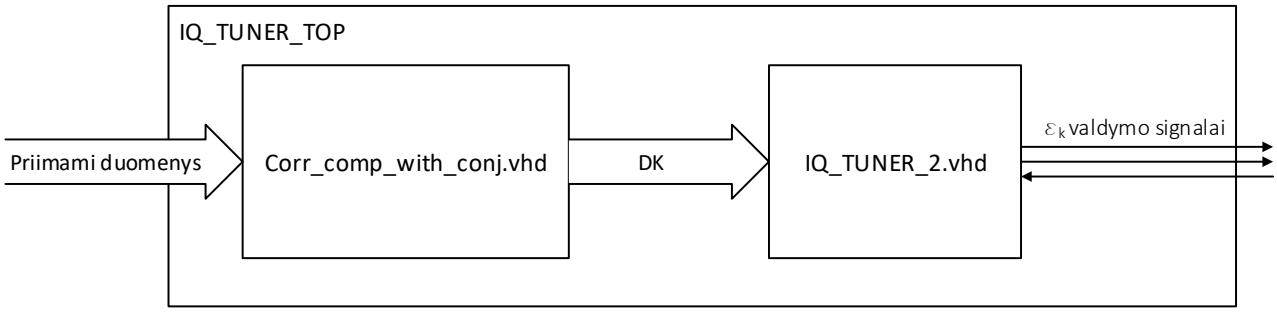
3.1. Derinimo metodo įgyvendinimas

Derinimo metodo įgyvendinimo LimeSDR-USB sistemos FPGA komponentėje schema grafiškai pateikta 3.2 pav. (pilka spalva pažymėti prie sistemos pridėti elementai), jame esančio modulio IQ_TUNER_TOP schema pateikta 3.3 pav..



3.2 pav. Derinimo metodo įgyvendinimo schema

Iš schemos matyti, jog derinimo metodas įtaką sistemai daro tik savo kuriamais valdymo signalais. Valdymo signalai pavaizduoti dvipusiai, kadangi derinimo moduliui suteikiamas grįžtamasis ryšys apie kompensacinio disbalanso ε_k pakeitimą.



3.3 pav. IQ_TUNER_TOP modulio schema

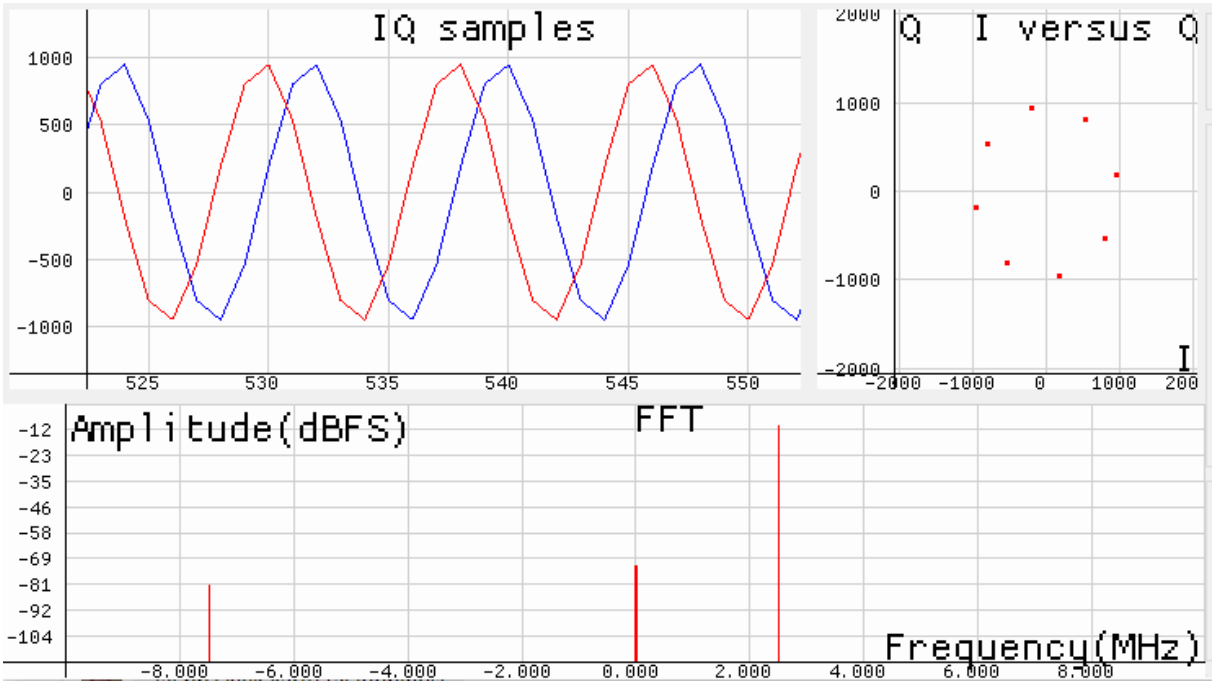
Schemoje pateikiami moduliai Corr_comp_with_conj ir IQ_TUNER_2 atitinka struktūras aprašytas 2.3 poskyryje(įskaitant 2.3.1 skyrelį) ir 2.4 poskyryje, atitinkamai.

3.2. Skirtumai tarp amplitudinio ir fazinio disbalansų derinimo

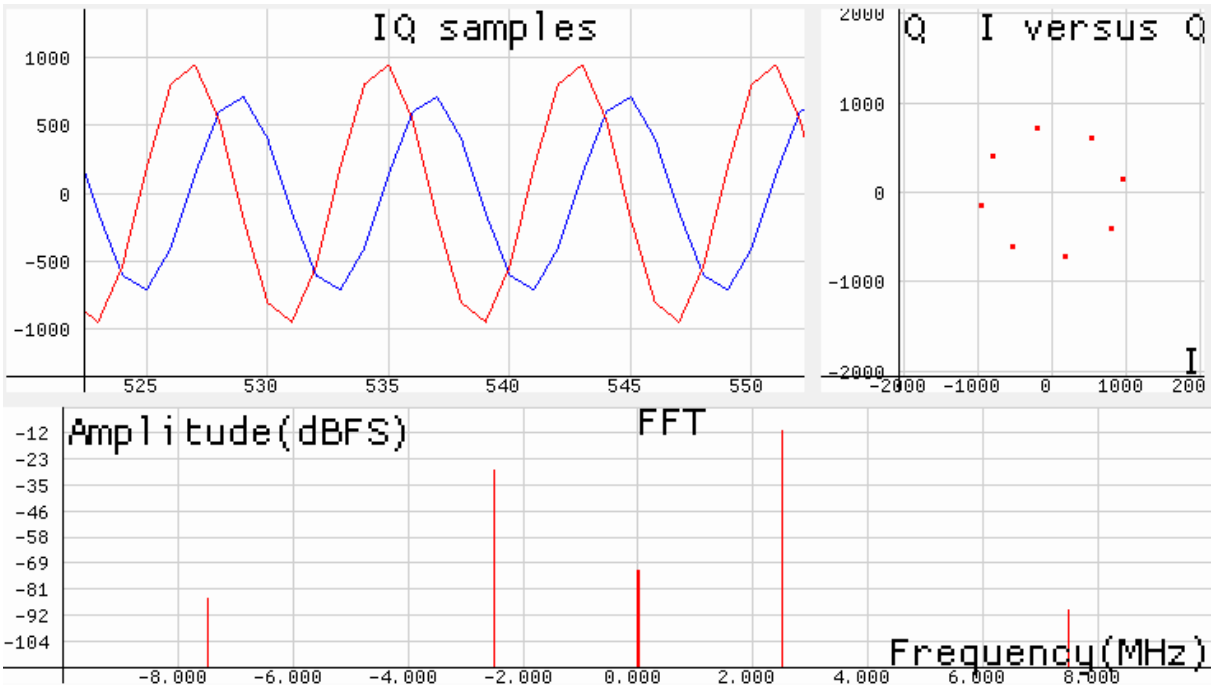
Esminis skirtumas tarp fazinio ir amplitudinio disbalansų derinimų yra tai, kokie parametrai yra prieinami šiam derinimui atlikti. Šio tyrimo metu naudojama aparatūra faziniam disbalansui kompensuoti siūlo vieną koeficientą – papildomo fazinio skirtumo tarp I ir Q dedamųjų nustatymą, o amplitudiniam disbalansui pateikia du koeficientus – abejų dedamųjų stiprintuvų nustatymus.

Kadangi amplitudinis disbalansas atsiranda tuomet, kai abejų dedamųjų stiprinimai yra skirtingi, amplitudinio disbalanso derinimą galima atlikti vieną dedamąją laikant pastovią ir koreguojant kitą. Tuomet tuos pačius modulius galima naudoti abejų disbalansų derinimui.

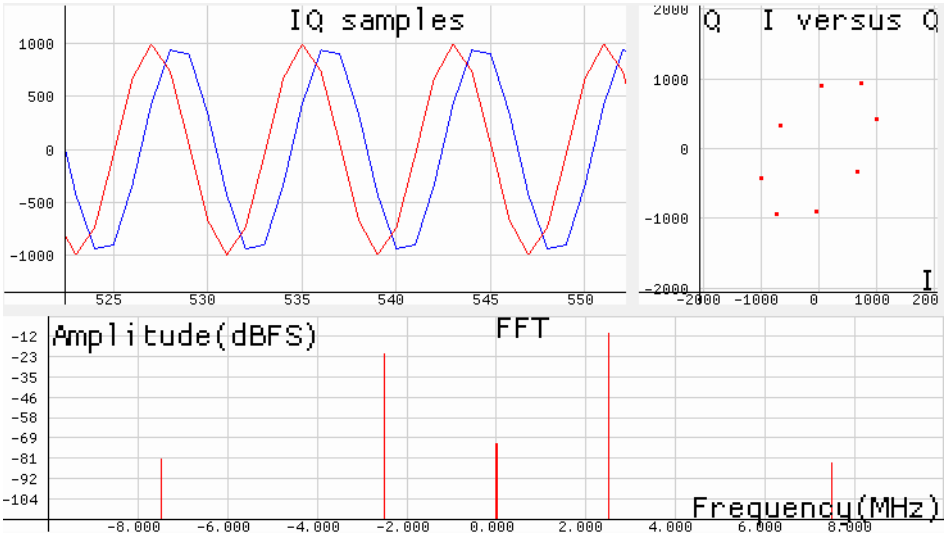
Siekiant ištirti kaip skirsis šių disbalansų konvergavimo procesas, nuspręsta juos abu atlikti naudojant tuos pačius derinimo parametrus (šie parametrai aprašyti 3 skyriaus pradžioje). Kadangi šio bandymo tikslas yra palyginti abu procesus, o ne įvertinti patį metodą, derinimui bus naudojamas pastovus testinis signalas, LimeSDR-USB plokštėje esančio siųstuvo/imtuvo generuojamas norint atlikti derinimą[20]. Testinis signalas pateiktas 3.4 pav., 3.5 pav.ir 3.6 pav., o derinimų rezultatai 3.7 pav. ir 3.8 pav. Pradinės derinimo sąlygos : fazinis disbalansas = 22,5 laipsniai, amplitudinis disbalansas = 25 %.



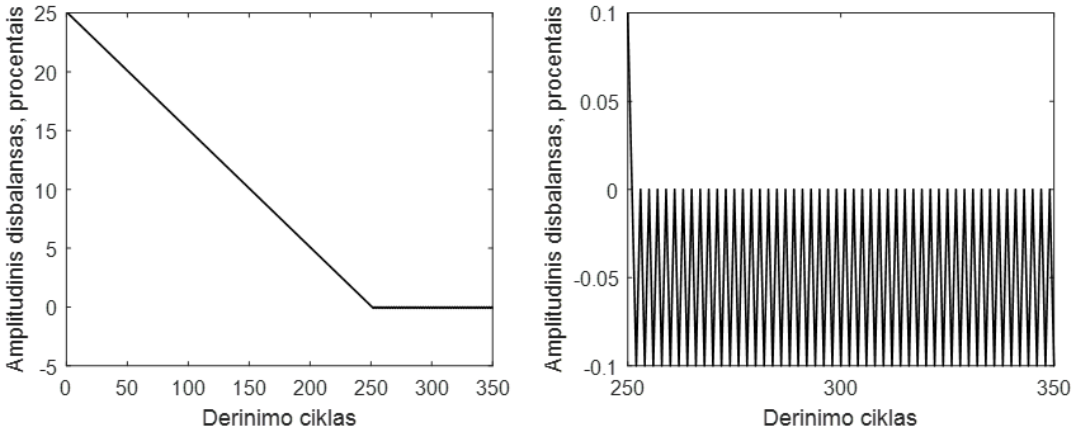
3.4 pav. Amplitudinio ir fazinio disbalansų derinimui naudojamas signalas prieš pradinį išderinimą



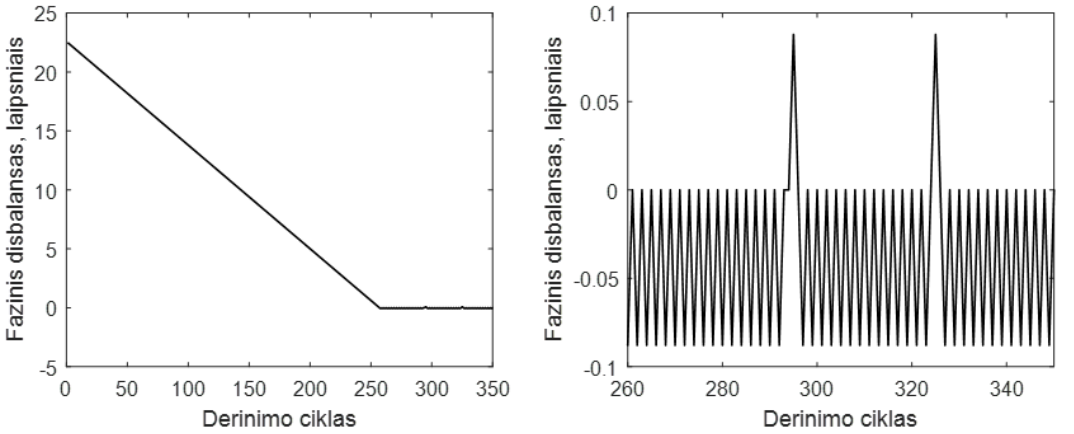
3.5 pav. Amplitudinio disbalanso derinimui naudojamas signalas



3.6 pav. Fazinio disbalanso derinimui naudojamas signalas



3.7 pav. Amplitudinio disbalanso konvergavimo eiga

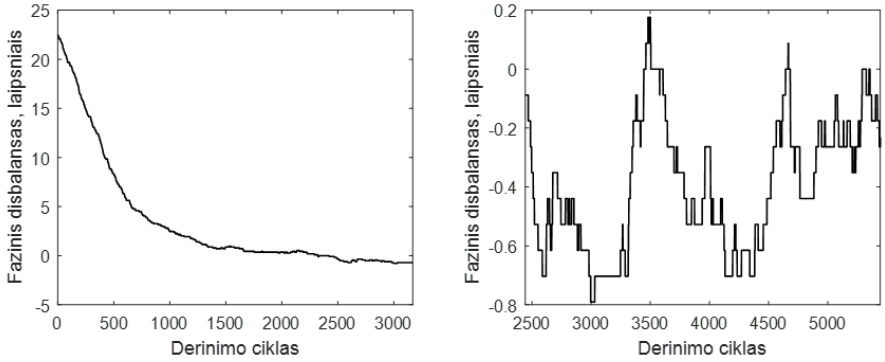


3.8 pav. Fazinio disbalanso konvergavimo eiga

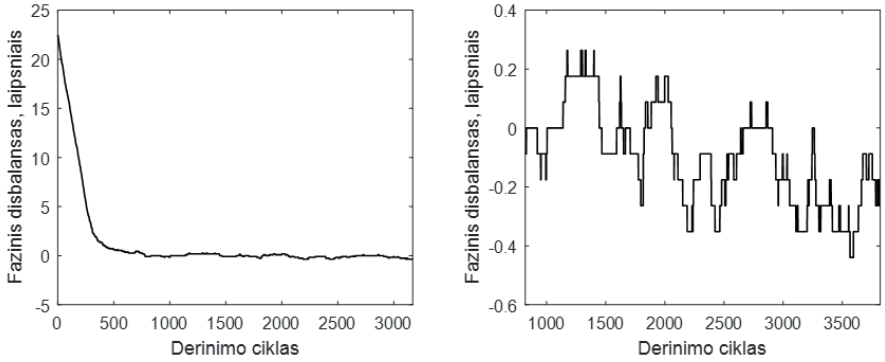
Palyginus abiejų disbalansų derinimo rezultatus matyti, jog reikšmingų skirtumų tarp abiejų procesų nėra. Dėl šios priežasties tolimesniuose poskyriuose bus vaizduojamas tik fazinio disbalanso derinimas.

3.3. Tiriamos sekos ilgio įtaka konvergavimo procesui

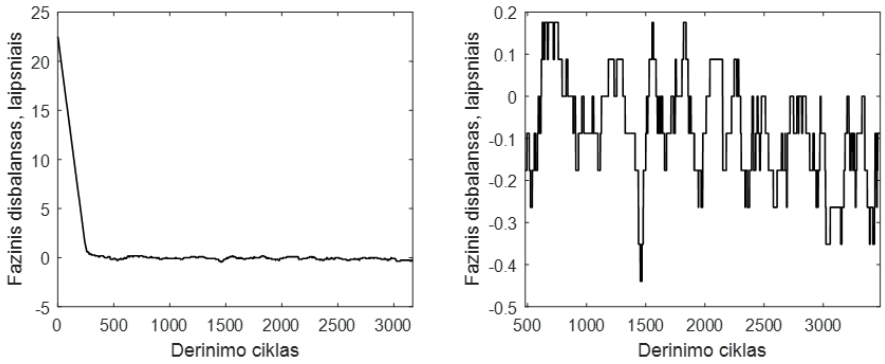
Norint ištirti tiriamos sekos ilgio įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 64, 256 ir 1024. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.9 pav. Konvergavimo eiga, kai tiriamo signalo ilgis yra 64 imtys



3.10 pav. Konvergavimo eiga, kai tiriamo signalo ilgis yra 256 imtys

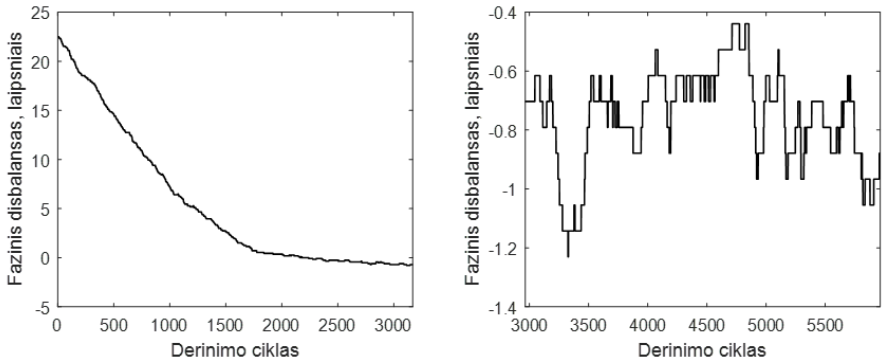


3.11 pav. Konvergavimo eiga, kai tiriamo signalo ilgis yra 1024 imtys

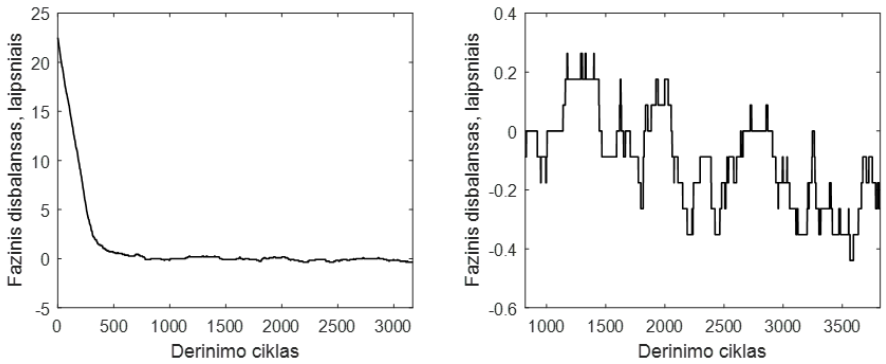
Iš rezultatų matyti, jog sekos ilgio keitimas neturi ryškios įtakos konvergavimo procesui po grubaus derinimo atlikimo. Vienintelis matomas skirtumas tarp šių konvergavimo procesų – grubiam derinimui reikalingas derinimo ciklų skaičius. Verta pastebėti, jog didinant tiriamos sekos ilgį, dėl didesnio vienam ciklui reikalingo duomenų kiekio, vienas ciklas atitinka ilgesnį laiko periodą.

3.4. Slenkančios medianos delta vertės įtaka konvergavimo procesui

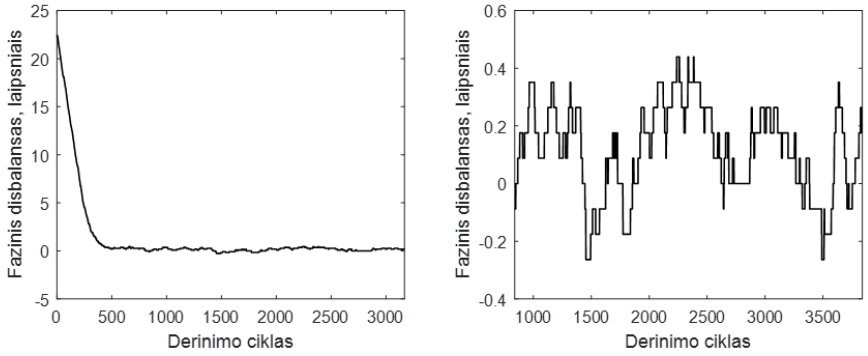
Norint ištirti slenkančios medianos delta vertės įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 0,25, 1 ir 4. Mažesnė už 1 kvantą delta vertė pasiekiama slenkančios medianos modulyje vidinę vertę praplėtus papildomais bitais. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.12 pav. Konvergavimo eiga, kai medianos delta vertė yra 0.25 kvanto



3.13 pav. Konvergavimo eiga, kai medianos delta vertė yra 1 kvantas

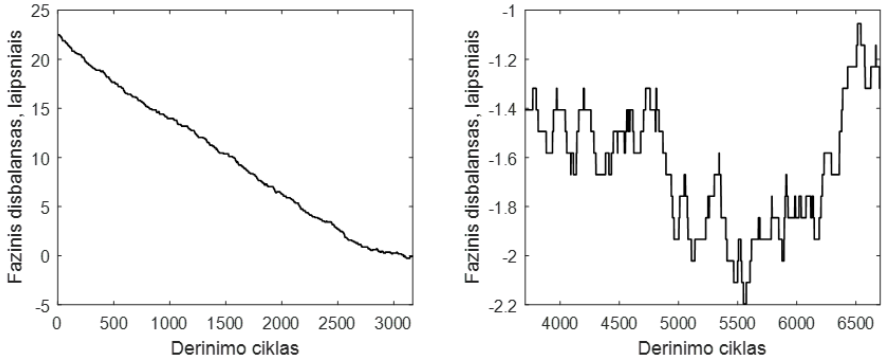


3.14 pav. Konvergavimo eiga, kai medianos delta vertė yra 4 kvantai

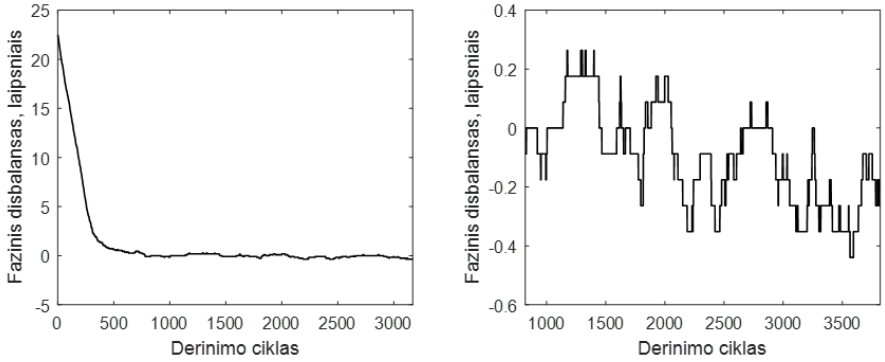
Iš rezultatų matyti, jog naudojant per mažą slenkančios medianos vertę ženkliai išauga derinimo ciklų skaičius, reikalingas grubiam derinimui atlikti, taip pat matyti, jog mažos delta vertės atveju, tikslaus derinimo metu rezultatai nuo pilno fazinio balanso yra nukrypę per apytiksliai 0,7 laipsnio. Galima to priežastis – dėl per mažos delta vertės sistema tampa pernelyg inertiška ir nesugeba pakankamai greitai reaguoti į kintančias koreliacijos koeficiento reikšmes.

3.5. Sumuojamų koeficientų kiekio įtaka konvergavimo procesui

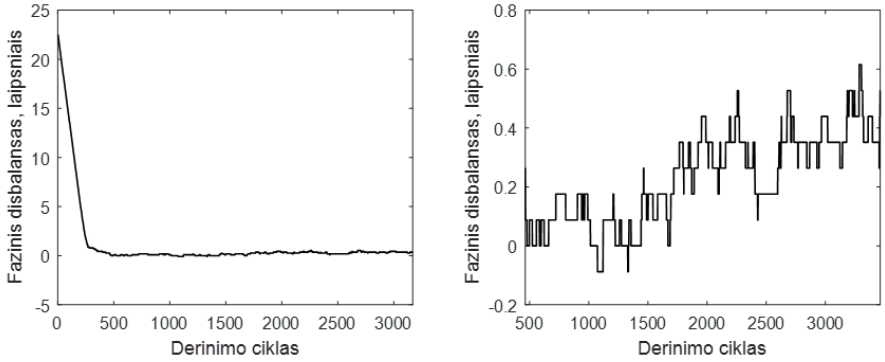
Norint ištirti sumuojamų koeficientų kiekio (M parametras formulėje (2.3.1.1)) įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 128, 512 ir 2048. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.15 pav. Konvergavimo eiga, kai sumuojami 128 koeficientai



3.16 pav. Konvergavimo eiga, kai sumuojama 512 koeficientų

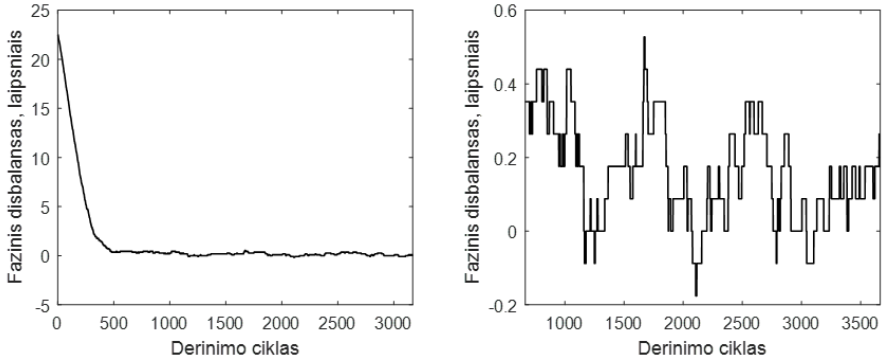


3.17 pav. Konvergavimo eiga, kai sumuojami 2048 koeficientai

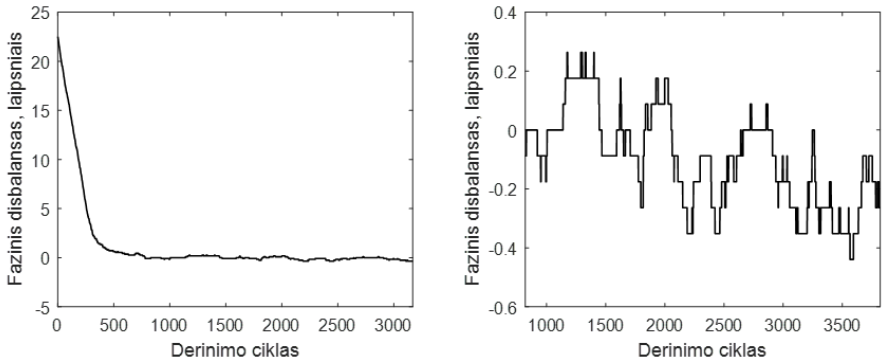
Iš rezultatų matyti, jog sumuojamų koeficientų kiekio didinimas ar mažinimas daro tokio pačio pobūdžio įtaką konvergavimo eigai, kaip ir slenkančios medianos delta vertės keitimas. Tai taip pat susiję su medianos inertiškumu, kadangi, esant inertiškesniems sumuojamiems koeficientams, reikalingas didesnis jų kiekis siekiant užfiksuoti kryptingus koeficiento pokyčius.

3.6. Derinimo metu atliekamų iteracijų skaičiaus įtaka konvergavimo procesui

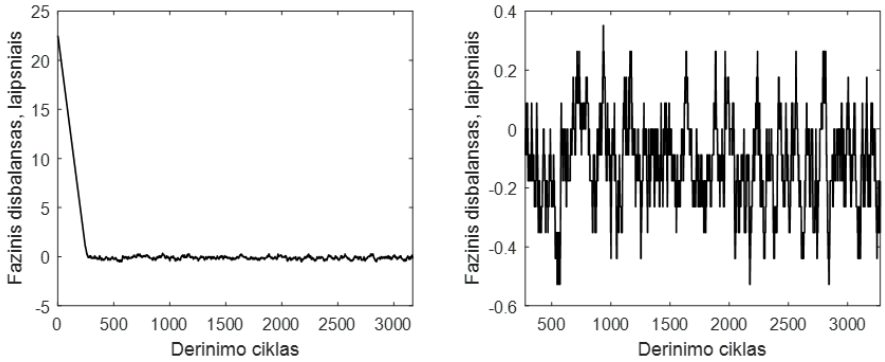
Norint ištirti derinimo metu atliekamų iteracijų skaičiaus įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 32, 64 ir 256. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.18 pav. Konvergavimo eiga, kai derinimo ciklą sudaro 32 iteracijos



3.19 pav. Konvergavimo eiga, kai derinimo ciklą sudaro 64 iteracijos

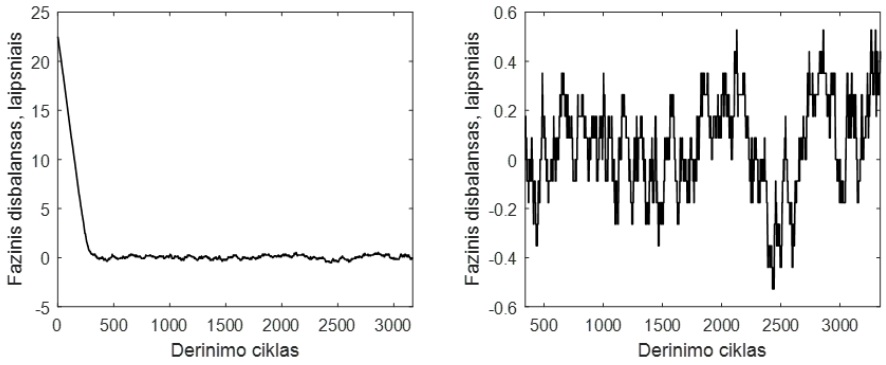


3.20 pav. Konvergavimo eiga, kai derinimo ciklą sudaro 256 iteracijos

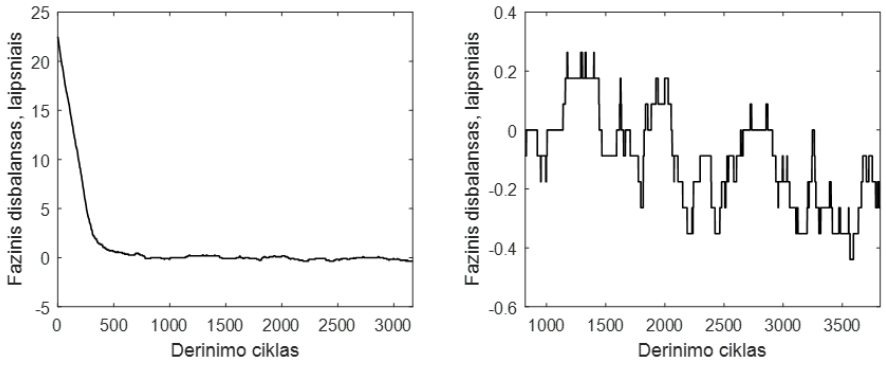
Vertinant šio tyrimo rezultatus verta atsižvelgti į santykį tarp derinimo sprendimo priėmimo slenkstinės vertės (visais šiais atvejais lygi 15) ir iteracijų skaičiaus. Šis santykis naudojant 32, 64 ir 256 iteracijas yra 46,9 %, 23,4 % ir 5,7 %, atitinkamai. Ryškus skirtumas matomas tarp dviejų mažesnių iteracijų skaičių ir didesniojo. Naudojant didesnį iteracijų skaičių gaunamas mažesnis santykinis sprendimo priėmimo slenkstis, kas padažnina parametro keitimą.

3.7. Sprendimo priėmimo slenksčio įtaka konvergavimo procesui

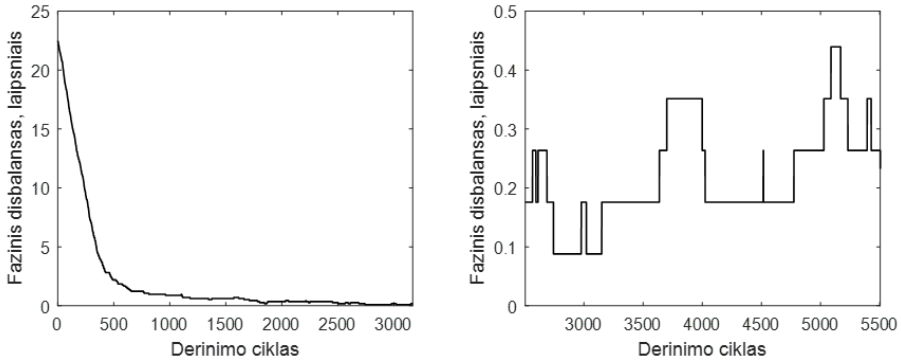
Norint ištirti sprendimo priėmimo slenksčio (*thr*, aprašytas 2.4 poskyryje) algoritme įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 10, 15 ir 20. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.21 pav. Konvergavimo eiga, kai sprendimo priėmimo slenkstis yra 10



3.22 pav. Konvergavimo eiga, kai sprendimo priėmimo slenkstis yra 15

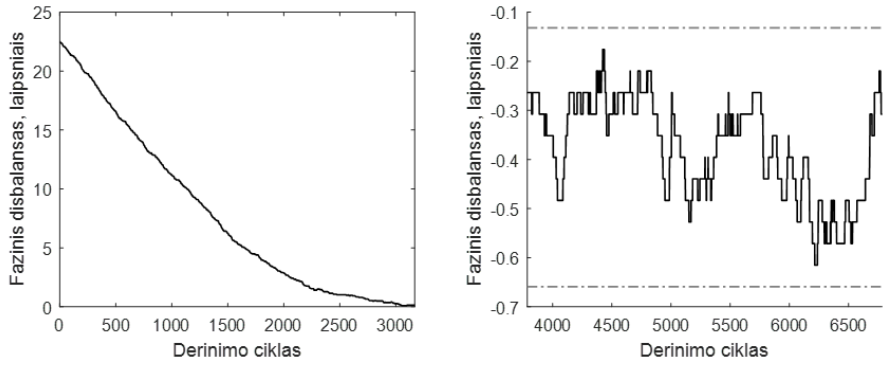


3.23 pav. Konvergavimo eiga, kai sprendimo priėmimo slenkstis yra 20

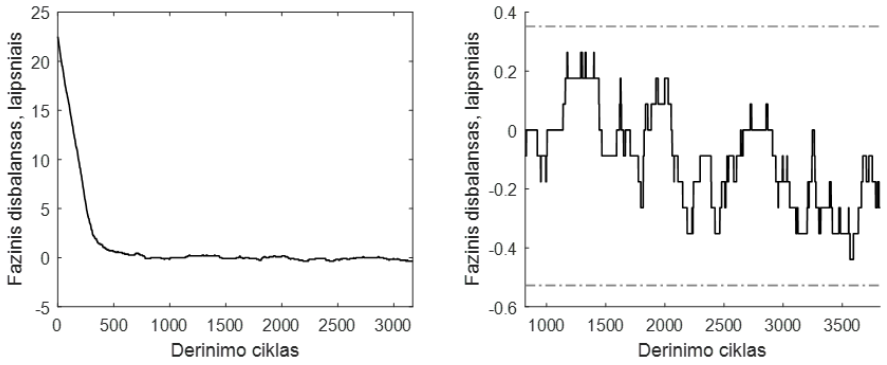
Keičiant sprendimo priėmimo slenkstį matoma panaši įtaka konvergavimo procesui, kaip ir keičiant iteracijų skaičių, kadangi abu parametrai daro įtaką slenksčio ir iteracijų skaičiaus santykiui. Šiuo atveju slenksčiai 10, 15 ir 20 atitinka santykius 15,6 %, 23,4 % ir 31,2 %. Matyti, jog esant mažesniam santykiui parametras keičiamas dažniau.

3.8. Parametro keitimo žingsnio įtaka konvergavimo procesui

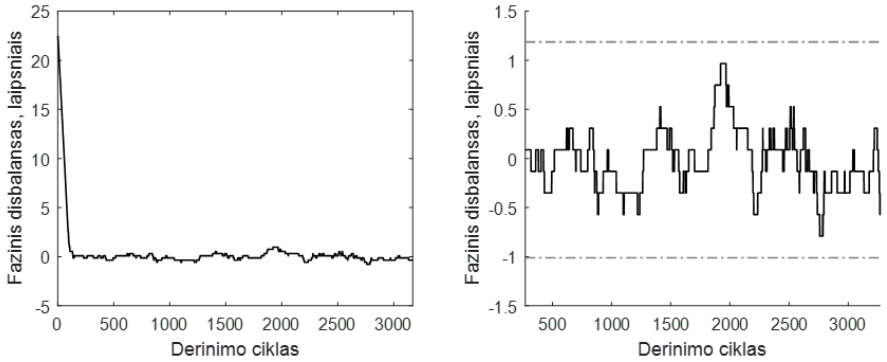
Norint ištirti parametro keitimo žingsnio (Δ, aprašytas 2.4 poskyryje) įtaką konvergavimo procesui atliekami bandymai naudojant tris skirtingas šio parametro reikšmes : 1, 2 ir 5. Pradinė fazinio disbalanso vertė – 22,5 laipsniai.



3.24 pav. Konvergavimo eiga, kai parametro keitimo žingsnis yra 1 kvantas (≈0.044 laipsnio)



3.25 pav. Konvergavimo eiga, kai parametro keitimo žingsnis yra 2 kvantai (≈0.088 laipsnio)



3.26 pav. Konvergavimo eiga, kai parametro keitimo žingsnis yra 5 kvantai (≈0.22 laipsnio)

Iš tyrimo rezultatų matyti, jog naudojant didesnį parametro keitimo žingsnį greičiau atliekamas grubus derinimas. Lyginant didesnį žingsnį naudojančio konvergavimo eigą su mažesnį žingsnį naudojančiu derinimu matyti, jog jis pasižymi staigesniais ir smarkesniais svyravimais. Verta turėti omenyje tai, jog derinamas parametras keičiamas ne tik derinimo sprendimo priėmimo momentais, bet ir viso derinimo metu. Žemiausios ir aukščiausios parametro vertės konvergavimo antroje pusėje grafikuose pažymėtos pilka punktyrine linija.

3.9. Apibendrinimas

Šiame poskyryje apibendrinami praeitų poskyrių tyrimų rezultatai, išskiriant šiuos aspektus.
Bendri:

- Loginių resursų sąnaudos.

Susiję su grubiu derinimu[1] (atitinka paveiksluose esančius kairius grafikus):

- Derinimo ciklų skaičius, reikalingas disbalansą sumažinti iki 1 laipsnio.
- Kompensuota derinimo trukmė. Pavyzdžiui, naudojant 4 kartus didesnį tiriamos sekos ilgį, skaičiavimui reikia 4 kartus daugiau duomenų, taigi vienas derinimo ciklas trunka 4 kartus ilgiau ir derinimo trukmė ciklais padauginama iš 4.

Susiję su tiksliu derinimu[1] (atitinka paveiksluose esančius dešinius grafikus):

- Vidutinė fazinio disbalanso vertė tikslaus derinimo metu.
- Standartinė fazinio disbalanso deviacija tikslaus derinimo metu.
- Skirtumas tarp fazinio disbalanso minimalios ir maksimalios reikšmių.

3.1 lentelė. Tyrimų rezultatų apibendrinimas

Parametras[2]	**Vertė**	**Loginių resursų sąnaudos**		**Derinimo trukmė, ciklais**		ε_θ **vidurkis, laipsniais**	ε_θ **standartinė deviacija, laipsniais**	ε_θ**verčių diapazonas, laipsniais**
		ALUT	**Atminties bitai**	**Nekomp.**	**komp.**			
Sekos Ilgis	1024	10147	24528	252	1008	-0,075	0,126	0,615
	256	9338	6096	420	420	-0,097	0,163	0,703
	64	9118	1488	1332	333	-0,395	0,225	0,967
Slenkančios medianos delta	0,25	9338	6096	1721		-0,751	0,164	0,791
	1			420		-0,097	0,163	0,7031
	4			372		0,116	0,159	0,7031
Sumuojamų koeficientų kiekis	128			2710	677	-1,642	0,253	1,143
	512			420	420	-0,097	0,163	0,703
	2048			268	1072	0,227	0,153	0,703
Atliekamų iteracijų skaičius	32			420	210	0,159	0,137	0,703
	64			420	420	-0,098	0,163	0,703
	256			247	988	-0,116	0,151	0,879
Sprendimo priėmimo slenkstis	10			273		0,053	0,195	1,055
	15			420		-0,097	0,163	0,703
	20			814		0,219	0,087	0,352
Parametro keitimo žingsnis	1			2558		-0,376	0,101	0,440
	2			420		-0,097	0,163	0,703
	5			104		-0,021	0,306	1,758

[1] Grubaus ir tikslaus derinimo apibrėžimai pateikti 3-čiame skyriuje.

[2] Kitų parametrų, pagal nutylėjimą nustatytos, vertės nurodytos 3-čiame skyriuje

Išvados

Šiame darbe aprašytas naujas Zero-IF architektūros imtuvų / siųstuvų IQ disbalanso mažinimo metodas, iš kitų literatūros šaltiniuose aprašytų metodų išsiskiriantis galimybe būti naudojamas prietaiso eksploatacijos metu.

Ištyrus sukurtą derinimo metodą matyti, jog jis vienodai veikia derinant tiek fazinį, tiek amplitudinį disbalansus. Esant ryškiam faziniam IQ disbalansui (didesniam nei 1°) derinimo metodas sėkmingai sugeba šį disbalansą sumažinti iki apie 0.05-0.1 laipsnio, kas atitinka apie 1-2 tyrimui naudotos aparatūros fazės keitimo kvantus. Pasiekus žemą disbalanso vertę išryškėja metodo sukuriama disbalanso atsitiktinė dedamoji – momentinė fazinio disbalanso vertė nėra pastovi ir pasižymi standartine deviacija, lygia apie 0.15 laipsnio. Ši problema galimai galėtų būti išspręsta, ar sumažinta kartu su šiuo metodu naudojant sudėtingesnį valdymo algoritmą, kuris galėtų aptikti minėtą svyravimą ir nustatyti jo centrinę vertę. Vertinant metodo parametrų įtaką konvergavimo trukmei, bei derinimo rezultatams, pastebėta, jog pasirenkant parametrus, kurie derinimo trukmę prailgina, dažnai gaunama mažesnė disbalanso standartinė deviacija tikslaus derinimo metu.

Metodas įgyvendintas LimeSDR-USB plokštėje sunaudojo apie 0,5 % atminties resursų ir apie 25 % loginių resursų, iš kurių apie 90 % sunaudojo dalybos ir cordic elementai. Reikšmingos loginių elementų sąnaudos apriboja šio metodo patrauklumą, tačiau itin mažos atminties resursų sąnaudos šį metodą daro patrauklų situacijose, kuomet derinimas reikalingas sistemoje naudojančioje daug atminties resursų, bet turinčią pakankamai neišnaudotų loginių elementų.

Literatūros sąrašas

1. 1. ELLINGER, F.Sud. Transceiver Architectures. In *Radio Frequency Integrated Circuits and Technologies* [interaktyvus]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 37–56. [žiūrėta 2018-12-18]. ISBN 978-3-540-35790-2Prieiga per internetą: <https://doi.org/10.1007/978-3-540-35790-2_2>.

2. 2. CHEN, S.-J. - HSIEH, Y.-H. TRANSCEIVER ARCHITECTURE DESIGN. In CHEN, S.-J. - HSIEH, Y.-H.Sud. *IQ CALIBRATION TECHNIQUES FOR CMOS RADIO TRANSCEIVERS* [interaktyvus]. Dordrecht: Springer Netherlands, 2006. p. 11–42. [žiūrėta 2018-12-18]. ISBN 978-1-4020-5083-1Prieiga per internetą: <https://doi.org/10.1007/1-4020-5083-6_2>.

3. 3. CHIUEH, T.-D. ir kt. *Baseband Receiver Design for Wireless MIMO-OFDM Communications* [interaktyvus]. . New York, SINGAPORE: John Wiley & Sons, Incorporated, 2012. ISBN 978-1-118-18820-0.

4. 4. SCHUCHERT, A. ir kt. A novel IQ imbalance compensation scheme for the reception of OFDM signals. In *IEEE Transactions on Consumer Electronics* . 2001. Vol. 47, no. 3, p. 313–318. .

5. 5. ZARRINKOUB, H. *Wiley Desktop Editions: Understanding Lte with Matlab: from Mathematical Modeling to Simulation and Prototyping*. . New York: John Wiley & Sons, Incorporated, 2014. ISBN 978-1-118-44341-5.

6. 6. ARSLAN, U. ir kt. Performance Evaluation and Real-Time Implementation of Subspace, Adaptive, and DFT Algorithms for Multi-tone Detection. In . 1996. .

7. 7. LTE Physical Layer Overview. In [interaktyvus]. [žiūrėta 2019-05-20]. Prieiga per internetą: <http://rfmw.em.keysight.com/wireless/helpfiles/89600B/WebHelp/subsystems/lte/content/lte_overview.htm>.

8. 8. 4G - LTE/LTE-A. In [interaktyvus]. [žiūrėta 2019-05-20]. Prieiga per internetą: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2014_2/rafaelreis/ofdma_scfdma.html>.

9. 9. HSU, C. - SHEEN, W. Joint Calibration of Transmitter and Receiver Impairments in Direct-Conversion Radio Architecture. In *IEEE Transactions on Wireless Communications* . 2012. Vol. 11, no. 2, p. 832–841. .

10. 10. HSU, C. - SHEEN, W. A New Self-Calibration Method for Transmitter and Receiver Radio Impairments in Direct-Conversion Architecture. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010* . 2010. p. 1–6. .

11. 11. DEBAILLIE, B. ir kt. Calibration of Direct-Conversion Transceivers. In *IEEE Journal of Selected Topics in Signal Processing* . 2009. Vol. 3, no. 3, p. 488–498. .

12. 12. ZHUANG, Y. - WAN, Y. Joint estimation of carrier frequency offset and in-phase/quadrature-phase imbalances for orthogonal frequency division multiplexing systems. In *Computers & Electrical Engineering* . 2015. Vol. 48, p. 1–11. .

13. 13. FEIGIN, J. - BRADY, D. Joint Transmitter/Receiver I/Q Imbalance Compensation for Direct Conversion OFDM in Packet-Switched Multipath Environments. In *IEEE Transactions on Signal Processing* . 2009. Vol. 57, no. 11, p. 4588–4593. .

14. 14. ZOU, Y. ir kt. Pilot-Based Compensation of Frequency-Selective I/Q Imbalances in Direct-Conversion OFDM Transmitters. In *2008 IEEE 68th Vehicular Technology Conference* . 2008. p. 1–5. .

15. 15. TARIGHAT, A. - SAYED, A.H. Joint compensation of transmitter and receiver impairments in OFDM systems. In *IEEE Transactions on Wireless Communications* . 2007. Vol. 6, no. 1, p. 240–247. .

16. 16. GUANBIN XING ir kt. Frequency offset and I/Q imbalance compensation for direct-conversion receivers. In *IEEE Transactions on Wireless Communications* . 2005. Vol. 4, no. 2, p. 673–680. .

17. 17. GYE-TAE GIL ir kt. Joint ML estimation of carrier frequency, channel, I/Q mismatch, and DC offset in communication receivers. In *IEEE Transactions on Vehicular Technology* . 2005. Vol. 54, no. 1, p. 338–349. .

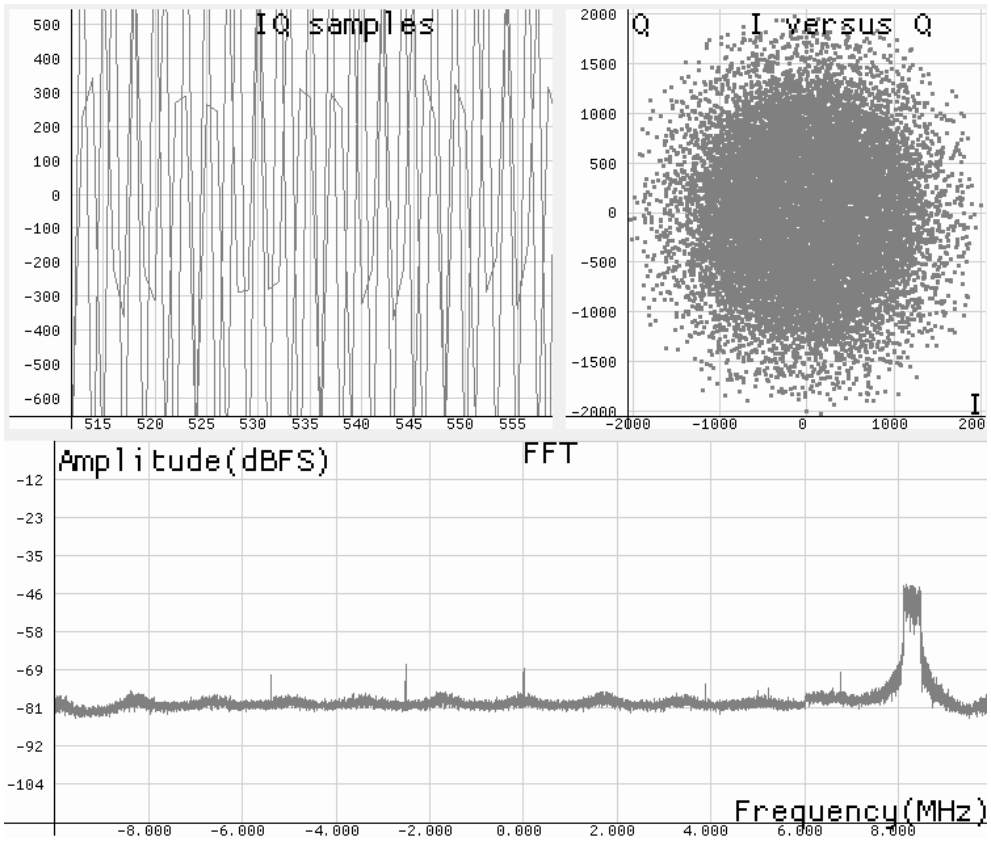
18. 18. VALKAMA, M. ir kt. Advanced methods for I/Q imbalance compensation in communication receivers. In *IEEE Transactions on Signal Processing* . 2001. Vol. 49, no. 10, p. 2335–2344. .

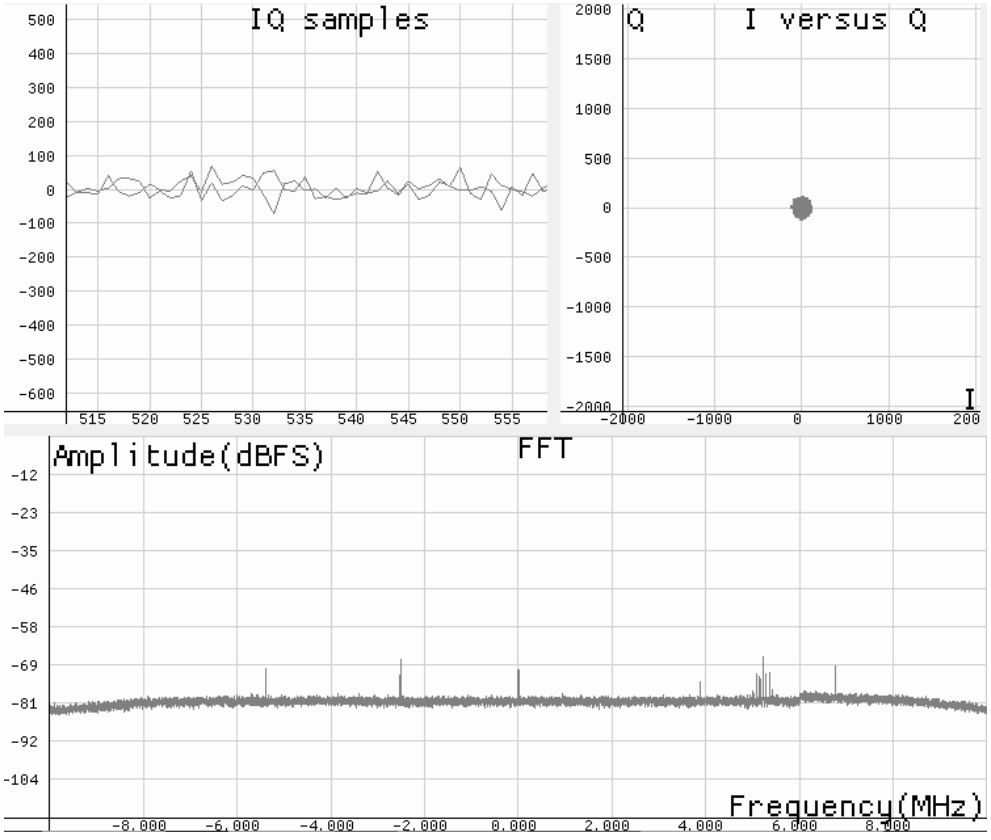
19. 19. Davies: Computer Vision, 5th edition, online materials Matlab Application 7 1 CVL Matlab12bc 28 February 2019, 14:54 © E. R. Davies 2019 Median-based background subtraction. In [interaktyvus]. [žiūrėta 2020-05-13]. Prieiga per internetą: <https://www.elsevier.com/__data/assets/pdf_file/0006/861630/MATLAB_Application_7.pdf>.

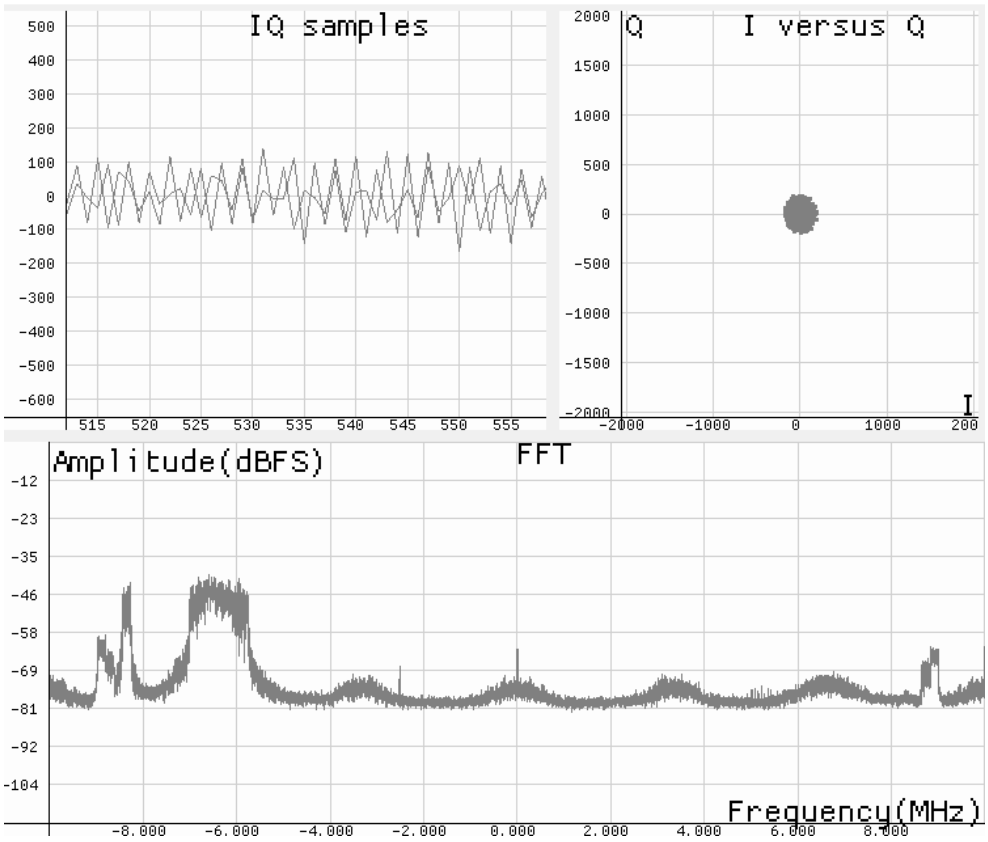
20. 20. LIME MICROSYSTEMS LIMITED RF and Analog Measurement Results -. In [interaktyvus]. [žiūrėta 2020-05-27]. Prieiga per internetą: <https://limemicro.com/app/uploads/2015/08/LMS7002M_Measurements-v1_05.pdf>.

Priedai

1 priedas. Daugiau aukštynkrypčio LTE ryšio duomenų pavyzdžių






IQ samples
500
400
300
200
100
0
-100
-200
-300
-400
-500
-600
515 520 525 530 535 540 545 550 555
Q
I versus Q
2000
1500
1000
500
0
-500
-1000
-1500
-2000
-2000 -1000 0 1000 200
I
Amplitude(dBFS)
FFT
-12
-23
-35
-46
-58
-69
-81
-92
-104
-8.000 -6.000 -4.000 -2.000 0.000 2.000 4.000 6.000 8.000
Frequency(MHz)

2 priedas. IQ_TUNER_TOP.vhd modulio kodas

```
--
************************************************************************************
************************************--
--! @file IQ_TUNER_TOP.vhd
--!
************************************************************************************
************************************--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity IQ_TUNER_TOP is
   port (
   CPU_CLK      : in std_logic;
   CALC_RESET_N : in std_logic;
   TUNE_RESET_N : in std_logic;
   DATA_CLK     : in std_logic;
   DATA_VALID   : in std_logic;

   DATA_REAL    : in std_logic_vector(11 downto 0);
   DATA_IMAG    : in std_logic_vector(11 downto 0);

   MEDIAN_DELTA : in std_logic_vector(15 downto 0);
   COUNT_LIM    : in std_logic_vector(15 downto 0);

   SHIFT_DONE   : in std_logic;
   SHIFT_REQ    : out std_logic_vector(1 downto 0)

   );
end entity IQ_TUNER_TOP;

architecture rtl of IQ_TUNER_TOP is

signal calc_res : std_logic_vector(18 downto 0);
signal calc_done: std_logic;

signal shift_done_int : std_logic;
signal shift_up_req   : std_logic;
signal shift_dn_req   : std_logic;

begin

    sync : process(all)
        variable shift_done_sync_chain : std_logic_vector(2 downto 0);
        variable shift_up_req_sync_chain : std_logic_vector(2 downto 0);
        variable shift_dn_req_sync_chain : std_logic_vector(2 downto 0);
    begin
        if rising_edge(DATA_CLK) then
            shift_done_sync_chain := shift_done_sync_chain(1 downto 0) &
SHIFT_DONE;
            shift_done_int <= shift_done_sync_chain(2) or
shift_done_sync_chain(1) or shift_done_sync_chain(0);

        end if;

        if rising_edge(CPU_CLK) then
            shift_up_req_sync_chain := shift_up_req_sync_chain(1 downto 0) &
shift_up_req;
            shift_dn_req_sync_chain := shift_dn_req_sync_chain(1 downto 0) &
shift_dn_req;

            SHIFT_REQ(0) <= shift_up_req_sync_chain(2) or
shift_up_req_sync_chain(1) or shift_up_req_sync_chain(0);
```

```
            SHIFT_REQ(1) <= shift_dn_req_sync_chain(2) or
shift_dn_req_sync_chain(1) or shift_dn_req_sync_chain(0);

        end if;
    end process;


   comp_Corr_comp_with_conj : entity work.Corr_comp_with_conj
      generic map (
                  G_DATA_LEN_POW2  => 8,
                  G_DATA_WIDTH     => 12,
                  G_AVERAGING_POW2 => 21
   )
      port map (
                iq_calibration_output_export => shift_done_int,
                count_lim                    => COUNT_LIM,
                MEDIAN_DELTA                 => MEDIAN_DELTA,
                VALUE_THR                    => (others => '0'),
                CLK                          => DATA_CLK,
                RESET_N                      => CALC_RESET_N,
                DATA_IN_REAL                 => DATA_REAL,
                DATA_IN_IMAG                 => DATA_IMAG,
                DATA_VALID                   => DATA_VALID,
                OUT_REAL                     => calc_res,
                OUT_IMAG                     => open,
                OUT_VALID                    => calc_done
   );


    comp_IQ_TUNER : entity work.IQ_TUNER_2
      port map (
                CLK              => DATA_CLK,
                RESET_N          => TUNE_RESET_N,
                PH_SHIFT_UP_REQ  => shift_up_req,
                PH_SHIFT_DWN_REQ => shift_dn_req,
                PH_SHIFT_DONE    => shift_done_int,
                CALC_DONE        => calc_done,
                CALC_RES         => calc_res
   );

end architecture rtl;
```

3 priedas. Corr_comp_with_conj.vhd kodas

```
--
**********************************************************************************
************************************--
--! @file  Corr_comp_with_conj.vhd
--! @brief computes the correlation coefficient between the input signal and
it's conjugate
--!
**********************************************************************************
************************************--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity Corr_comp_with_conj is
   generic (
   G_DATA_LEN_POW2 : natural := 12;
   G_DATA_WIDTH    : natural := 12;
    G_AVERAGING_POW2: natural := 12
   );
   port (
    iq_calibration_output_export : in std_logic;
    count_lim               : in std_logic_vector(15 downto 0);
    MEDIAN_DELTA    : in std_logic_vector(15 downto 0);
    VALUE_THR        : in std_logic_vector(13 downto 0);

   CLK             : in std_logic;
   RESET_N         : in std_logic;

   DATA_IN_REAL    : in std_logic_vector(G_DATA_WIDTH-1 downto 0);
   DATA_IN_IMAG    : in std_logic_vector(G_DATA_WIDTH-1 downto 0);
   DATA_VALID      : in std_logic;

   OUT_REAL        : out std_logic_vector(18 downto 0);
   OUT_IMAG        : out std_logic_vector(((G_DATA_WIDTH+1)*4)-1 downto 0);
   OUT_VALID       : out std_logic

   );
end entity Corr_comp_with_conj;

architecture rtl of Corr_comp_with_conj is

     constant G_DELAY_LEN : natural := 2**G_DATA_LEN_POW2;
     signal Raw_data_delayed_real : std_logic_vector(G_DATA_WIDTH-1 downto 0);
     signal Raw_data_delayed_imag : std_logic_vector(G_DATA_WIDTH-1 downto 0);
     signal Raw_data_mean_real    : std_logic_vector(G_DATA_WIDTH-1 downto 0);
     signal Raw_data_mean_imag    : std_logic_vector(G_DATA_WIDTH-1 downto 0);

     signal Data_no_mean_real     : std_logic_vector(G_DATA_WIDTH downto 0);
     signal Data_no_mean_imag     : std_logic_vector(G_DATA_WIDTH downto 0);
     signal Data_no_mean_imag_inv : std_logic_vector(G_DATA_WIDTH downto 0);

     signal After_square_real     : std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);
     signal After_square_imag     : std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);

     signal After_mult_real       : std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);
     signal After_mult_imag       : std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);
```

```
     signal After_mult_real_mean  : std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);

     signal After_square_real_mean: std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);
     signal After_square_imag_mean: std_logic_vector(((G_DATA_WIDTH+1)*2)-1
downto 0);
     signal Covariance_real_padded: std_logic_vector(((G_DATA_WIDTH+1)*4)-1
downto 0);
     signal Covariance_imag_padded: std_logic_vector(((G_DATA_WIDTH+1)*4)-1
downto 0);

     signal Delayed_data_valid    : std_logic;

     signal Second_means_done     : std_logic;

     signal data_out_imag_reg     : std_logic_vector(((G_DATA_WIDTH+1)*4)-1
downto 0);
     signal data_out_real_reg     : std_logic_vector(((G_DATA_WIDTH+1)*4)-1
downto 0);
     signal data_out_valid_reg    : std_logic_vector(0 downto 0);

     signal cordic_amplitude      : std_logic_vector(18 downto 0);
     signal avg_cor_kof           : std_Logic_vector(18 downto 0);
     signal avg_cor_kof2               : std_logic_vector(18 downto 0);
     signal avg_cor_kof2_del           : std_logic_vector(18 downto 0);
     signal avg_cor_kof_dif            : std_logic_vector(19 downto 0);
     signal abs_kof_dif                     : std_logic_vector(18 downto 0);
     signal dif_sum                         : std_logic_vector(18 downto 0) :=
(others => '0');

     signal dif_up                          : unsigned(13 downto 0) := (others
=> '0');
     signal dif_down                             : unsigned(13 downto 0) :=
(others => '0');

     signal kof_dif_sign                    : std_logic;
     signal avg_rdy               : std_logic;


     signal min_val_0                 : unsigned(18 downto 0);
     signal max_val_0                 : unsigned(18 downto 0);
     signal min_val_1                 : unsigned(18 downto 0);
     signal max_val_1                 : unsigned(18 downto 0);
     signal mid_val_1                  : unsigned(18 downto 0);
     signal spread_0                  : unsigned(18 downto 0);
     signal spread_1                  : unsigned(18 downto 0);
     signal calc_len                  : unsigned(15 downto 0) := (others =>
'0');
     signal result_counter          : unsigned(31 downto 0);

     signal placeholder           : std_Logic;


    component Data_delay is
       generic (
        G_DELAY_LEN : natural := 2;
        G_DATA_WIDTH: natural := 12
        );
       port (
        CLK            : in  std_logic;
        DATA_IN        : in  std_logic_vector(G_DATA_WIDTH-1 downto 0);
        DATA_VALID     : in std_logic;
```

```
    DATA_OUT       : out std_logic_vector(G_DATA_WIDTH-1 downto 0)
   );
end component Data_delay;

component Mean_calculator is
   generic (
    G_DATA_LEN_POW2 : natural := 1;
    G_DATA_WIDTH    : natural := 12
    );
   port (
    CLK              : in std_logic;
    RESET_N          : in std_Logic;
    DATA_VALID       : in std_logic;
    DATA_IN          : in std_logic_vector(G_DATA_WIDTH-1 downto 0);

    DATA_OUT         : out std_logic_vector(G_DATA_WIDTH-1 downto 0);
    DATA_OUT_UPDATED : out std_logic

   );
end component Mean_calculator;

component altmult_complex
generic (
    intended_device_family:string := "unused";
    implementation_style:string := "AUTO";
    pipeline:natural := 1;
    representation_a:string := "SIGNED";
    representation_b:string := "SIGNED";
    width_a:natural := G_DATA_WIDTH+1;
    width_b:natural := G_DATA_WIDTH+1;
    width_result:natural := (G_DATA_WIDTH+1)*2;
    lpm_hint:string := "UNUSED";
    lpm_type:string := "altmult_complex");
port(
    aclr:in std_logic := '0';
    clock:in std_logic := '0';
    complex:in std_logic := '1';
    dataa_imag:in std_logic_vector(width_a-1 downto 0);
    dataa_real:in std_logic_vector(width_a-1 downto 0);
    datab_imag:in std_logic_vector(width_b-1 downto 0);
    datab_real:in std_logic_vector(width_b-1 downto 0);
    ena:in std_logic := '1';
    result_imag:out std_logic_vector(width_result-1 downto 0);
    result_real:out std_logic_vector(width_result-1 downto 0));
end component;

component LPM_DIVIDE
 generic (
    LPM_WIDTHN           : natural := (G_DATA_WIDTH+1)*4;
    LPM_WIDTHD           : natural := (G_DATA_WIDTH+1)*2;
    LPM_NREPRESENTATION  : string := "SIGNED";
    LPM_DREPRESENTATION  : string := "SIGNED";
    LPM_PIPELINE         : natural := 1;
    LPM_TYPE             : string := "L_DIVIDE";
    LPM_HINT             : string := "UNUSED");
port (
    NUMER       : in std_logic_vector(LPM_WIDTHN-1 downto 0);
    DENOM       : in std_logic_vector(LPM_WIDTHD-1 downto 0);
    ACLR        : in std_logic := '0';
    CLOCK       : in std_logic := '0';
    CLKEN       : in std_logic := '1';
    QUOTIENT    : out std_logic_vector(LPM_WIDTHN-1 downto 0);
    REMAIN      : out std_logic_vector(LPM_WIDTHD-1 downto 0));
end component;
```

```
begin

-- ------------------------------------------------------------------------------
------------------
--  Delay incoming data to make sure it comes out at the same time as a
corresponding mean value
-- ------------------------------------------------------------------------------
------------------


   raw_data_delay_real : Data_delay
      generic map (
                  G_DELAY_LEN  => G_DELAY_LEN,
                  G_DATA_WIDTH => G_DATA_WIDTH
   )
      port map (
                  CLK        => CLK,
                  DATA_IN    => DATA_IN_REAL,
                  DATA_VALID => DATA_VALID,
                  DATA_OUT   => Raw_data_delayed_real
   );

   raw_data_delay_imag : Data_delay
      generic map (
                  G_DELAY_LEN  => G_DELAY_LEN,
                  G_DATA_WIDTH => G_DATA_WIDTH
   )
      port map (
                  CLK        => CLK,
                  DATA_IN    => DATA_IN_IMAG,
                  DATA_VALID => DATA_VALID,
                  DATA_OUT   => Raw_data_delayed_imag
   );

-- ------------------------------------------------------------------------------
------------------
--  Calculate the mean of the incoming values
-- ------------------------------------------------------------------------------
------------------

   raw_data_mean_calculator_real : Mean_calculator
      generic map (
                  G_DATA_LEN_POW2 => G_DATA_LEN_POW2,
                  G_DATA_WIDTH    => G_DATA_WIDTH
   )
      port map (
                  CLK              => CLK,
                  RESET_N          => RESET_N,
                  DATA_VALID       => DATA_VALID,
                  DATA_IN          => DATA_IN_REAL,
                  DATA_OUT         => Raw_data_mean_real,
                  DATA_OUT_UPDATED => open
   );

    raw_data_mean_calculator_imag : Mean_calculator
      generic map (
                  G_DATA_LEN_POW2 => G_DATA_LEN_POW2,
                  G_DATA_WIDTH    => G_DATA_WIDTH
   )
      port map (
                  CLK              => CLK,
```

```
                RESET_N          => RESET_N,
                DATA_VALID       => DATA_VALID,
                DATA_IN          => DATA_IN_IMAG,
                DATA_OUT         => Raw_data_mean_imag,
                DATA_OUT_UPDATED => open
   );

-- ------------------------------------------------------------------------
------------------
--  Calculate difference between input data and mean of the data
-- ------------------------------------------------------------------------
------------------

    Data_no_mean_real     <=
std_logic_vector(signed(Raw_data_delayed_real(Raw_data_delayed_real'LEFT) &
Raw_data_delayed_real) - signed(Raw_data_mean_real(Raw_data_mean_real'LEFT) &
Raw_data_mean_real));
    Data_no_mean_imag     <=
std_logic_vector(signed(Raw_data_delayed_imag(Raw_data_delayed_imag'LEFT) &
Raw_data_delayed_imag) - signed(Raw_data_mean_imag(Raw_data_mean_imag'LEFT) &
Raw_data_mean_imag));
    Data_no_mean_imag_inv <= std_logic_vector(-signed(Data_no_mean_imag));

-- ------------------------------------------------------------------------
------------------
--  Square the no_mean signal
-- ------------------------------------------------------------------------
------------------

Data_square : altmult_complex
port map(
    clock                       => CLK,
    dataa_imag                  =>Data_no_mean_imag,
    dataa_real                  =>Data_no_mean_real,
    datab_imag                  =>Data_no_mean_imag,
    datab_real                  =>Data_no_mean_real,
    result_imag                 =>After_square_imag,
    result_real                 =>After_square_real
);

-- ------------------------------------------------------------------------
------------------
--  Multiply the no_mean signal with it's conjugate
-- ------------------------------------------------------------------------
------------------

Conj_mult : altmult_complex
port map(
    clock                       => CLK,
    dataa_imag                  =>Data_no_mean_imag,
    dataa_real                  =>Data_no_mean_real,
    datab_imag                  =>Data_no_mean_imag_inv,
    datab_real                  =>Data_no_mean_real,
    result_imag                 =>open,
    result_real                 =>After_mult_real
);

-- ------------------------------------------------------------------------
------------------
--  Generate delayed data_valid signal
-- ------------------------------------------------------------------------
------------------

    gen_del_dat_val : process(all)
```

```
        variable delay_line : std_logic_vector(G_DELAY_LEN downto 0);
    begin
        if(rising_edge(clk)) then
            if(reset_n) then
                delay_line(delay_line'LEFT downto 1) :=
delay_line(delay_line'LEFT-1 downto 0);
                delay_line(0) := DATA_VALID;
            else
                delay_line := (others => '0');
            end if;
        end if;
        Delayed_data_valid <= delay_line(G_DELAY_LEN);
    end process;

-- ------------------------------------------------------------------------
------------------
--  Average results of squaring
-- ------------------------------------------------------------------------
------------------


    after_square_mean_calculator_real : Mean_calculator
      generic map (
                  G_DATA_LEN_POW2 => G_DATA_LEN_POW2,
                  G_DATA_WIDTH    => G_DATA_WIDTH*2+2
   )
      port map (
                CLK              => CLK,
                RESET_N          => RESET_N,
                DATA_VALID       => Delayed_data_valid,
                DATA_IN          => After_square_real,
                DATA_OUT         => After_square_real_mean,
                DATA_OUT_UPDATED => open
   );



    after_square_mean_calculator_imag : Mean_calculator
      generic map (
                  G_DATA_LEN_POW2 => G_DATA_LEN_POW2,
                  G_DATA_WIDTH    => G_DATA_WIDTH*2+2
   )
      port map (
                CLK              => CLK,
                RESET_N          => RESET_N,
                DATA_VALID       => Delayed_data_valid,
                DATA_IN          => After_square_imag,
                DATA_OUT         => After_square_imag_mean,
                DATA_OUT_UPDATED => open
   );

-- ------------------------------------------------------------------------
------------------
--  Average results of multiplication
-- ------------------------------------------------------------------------
------------------

    after_mult_mean_calculator_imag : Mean_calculator
      generic map (
                  G_DATA_LEN_POW2 => G_DATA_LEN_POW2,
                  G_DATA_WIDTH    => G_DATA_WIDTH*2+2
   )
      port map (
                CLK              => CLK,
```

```
                RESET_N          => RESET_N,
                DATA_VALID       => Delayed_data_valid,
                DATA_IN          => After_mult_real,
                DATA_OUT         => After_mult_real_mean,
                DATA_OUT_UPDATED => Second_means_done
   );

-- ------------------------------------------------------------------------
------------------
-- Divide covariance by variances
-- ------------------------------------------------------------------------
------------------
    Covariance_real_padded(Covariance_real_padded'LEFT downto
After_square_real_mean'LEFT+1) <= After_square_real_mean;
    Covariance_real_padded(After_square_real_mean'LEFT downto 0)             <=
(others => '0') when After_square_real_mean(G_DATA_WIDTH*2-1) = '0' else (others
=> '1');
    Covariance_imag_padded(Covariance_real_padded'LEFT downto
After_square_imag_mean'LEFT+1) <= After_square_imag_mean;
    Covariance_imag_padded(After_square_imag_mean'LEFT downto 0)             <=
(others => '0') when After_square_imag_mean(G_DATA_WIDTH*2-1) = '0' else (others
=> '1');

    corr_coeff_calc_real : LPM_DIVIDE
      port map(
        NUMER       => Covariance_real_padded,
        DENOM       => After_mult_real_mean,
        CLOCK       => CLK,
        QUOTIENT    => data_out_real_reg,
        REMAIN      => open
    );

    corr_coeff_calc_imag : LPM_DIVIDE
      port map(
        NUMER       => Covariance_imag_padded,
        DENOM       => After_mult_real_mean,
        CLOCK       => CLK,
        QUOTIENT    => data_out_imag_reg,
        REMAIN      => open
    );

    registers : process(all)
    begin
        if(rising_edge(CLK)) then
            data_out_valid_reg(0) <= Second_means_done;
        end if;
    end process;


----------------------------------------


    absolute_val_calc : entity work.cordic
        port map(
        clk =>CLK,
        areset=>'0',
        en =>data_out_valid_reg,
        x  =>data_out_imag_reg(29 downto 0),
        y  =>data_out_real_reg(29 downto 0),
        q  => open,
        r  => cordic_amplitude);
```

```
    avg_cor_kof_calc : entity work.moving_median
      generic map (
                G_DATA_WIDTH     => 19,--14,
                G_FRACTIONAL     => true,
                G_FRACTION_DEPTH => 4
   )
      port map (
                CLK              => CLK,
                INPUT_VAL        => data_out_valid_reg(0),
                INPUT            => cordic_amplitude,
                OUTPUT           => avg_cor_kof2,
                DELTA            => MEDIAN_DELTA(13 downto 0)
   );

   kof_delay : Data_delay
      generic map (
                   G_DELAY_LEN  => 1,
                   G_DATA_WIDTH => 19
   )
      port map (
                CLK        => CLK,
                DATA_IN    => avg_cor_kof2,
                DATA_VALID => data_out_valid_reg(0),
                DATA_OUT   => avg_cor_kof2_del
   );


    process(all)
    begin
        if rising_edge(CLK) then
            avg_cor_kof_dif <= std_logic_vector(signed('0' & avg_cor_kof2) -
signed('0' & avg_cor_kof2_del));
            if iq_calibration_output_export = '1' then
                calc_len <= (others => '0');
                dif_sum <= (others => '0');
                dif_up  <= (others => '0');
                dif_down<= (others => '0');
            elsif data_out_valid_reg(0)='1' and calc_len < 16x"FFFF" then
                if signed(avg_cor_kof_dif) > 0 then dif_up   <= dif_up + 1  ;
end if;
                if signed(avg_cor_kof_dif) < 0 then dif_down <= dif_down + 1;
end if;
                calc_len <= calc_len + 1;
                dif_sum <= std_logic_vector(signed(dif_sum )+
signed(avg_cor_kof_dif(13 downto 0)));
            end if;
        end if;
    end process;


    OUT_VALID <= '1' when calc_len = unsigned(count_lim) else '0';--'1' when
unsigned(abs_kof_dif) > unsigned(VALUE_THR) else '0';
    OUT_REAL  <= dif_sum;



end architecture rtl;
```

4 priedas. IQ_TUNER_2.vhd kodas

```
--*****************************************************************************
--! @file IQ_TUNER_2
--! @brief File Description
--! *****************************************************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity IQ_TUNER_2 is
   port (
        CLK              : in std_logic;
        RESET_N          : in std_logic;

        PH_SHIFT_UP_REQ  : out std_logic;
        PH_SHIFT_DWN_REQ : out std_logic;
        PH_SHIFT_DONE    : in  std_logic;

        CALC_DONE        : in  std_logic;
        CALC_RES         : in  std_logic_vector(18 downto 0)

   );
end entity IQ_TUNER_2;

architecture rtl of IQ_TUNER_2 is

type T_STATE_TYPE is (reset_state, idle, phase_up, phase_up_wait,
phase_up_wait2,
                            phase_return0, phase_return0_wait,
                            phase_down, phase_down_wait, phase_down_wait2,
                            phase_return1, phase_return1_wait,
                            phase_process, phase_decide);
signal current_state,next_state : T_STATE_TYPE;

signal phase_up_counter_inc: std_logic := '0';
signal phase_up_counter_dec: std_logic := '0';
signal phase_up_counter_rst: std_logic := '0';
signal phase_shift_counter : signed(5 downto 0) := (others => '0');
signal phase_up_counter    : signed(phase_shift_counter'LEFT+2 downto 0) :=
(others => '0');

signal phase_up_res_save   : std_logic;
signal phase_dwn_res_save  : std_logic;
signal phase_up_calc_res   : signed(18 downto 0) := (others => '0');
signal phase_down_calc_res : signed(18 downto 0) := (others => '0');



attribute noprune : boolean;
attribute noprune of phase_up_counter : signal is true;

begin

state_switcher : process(all)
begin
    if RESET_N = '0' then
        current_state <= reset_state;
    elsif rising_edge(CLK) then
        current_state <= next_state;
    end if;
end process;

fsm : process(all)
```

```
begin
    next_state           <= current_state;
    phase_up_counter_inc <= '0';
    phase_up_counter_dec <= '0';
    phase_up_counter_rst <= '0';
    PH_SHIFT_UP_REQ  <= '0';
    PH_SHIFT_DWN_REQ <= '0';

    phase_up_res_save <= '0';
    phase_dwn_res_save<= '0';


    case current_state is
        when reset_state     =>
            phase_up_counter_rst <= '1';
            next_state           <=idle;

        when idle            =>
            if PH_SHIFT_DONE = '0' then
                next_state       <= phase_up;
            end if;

          ----------------PHASE UP STATES BEGIN
        when phase_up        =>
            PH_SHIFT_UP_REQ  <= '1';
            if PH_SHIFT_DONE then
                next_state <= phase_up_wait;
            end if;

        when phase_up_wait   =>
            if PH_SHIFT_DONE = '0' then
                next_state <= phase_up_wait2;
            end if;

        when phase_up_wait2   =>
            if CALC_DONE then
                phase_up_res_save <= '1';
                next_state        <= phase_return0;
            end if;
          ----------------PHASE UP STATES END
          when phase_return0 =>
              PH_SHIFT_DWN_REQ <= '1';
              if PH_SHIFT_DONE then
                    next_state <= phase_return0_wait;
              end if;

          when phase_return0_wait =>
              if PH_SHIFT_DONE = '0' then
                    next_state <= phase_down;
              end if;
          ----------------PHASE DOWN STATES BEGIN
        when phase_down      =>
            PH_SHIFT_DWN_REQ <= '1';
            if PH_SHIFT_DONE then
                next_state <= phase_down_wait;
            end if;

        when phase_down_wait =>
            if PH_SHIFT_DONE = '0' then
                next_state <= phase_down_wait2;
            end if;

        when phase_down_wait2 =>
                if CALC_DONE then
```

```
                phase_dwn_res_save  <= '1';
                next_state          <= phase_return1;
            end if;
        ----------------PHASE DOWN STATES END
          when phase_return1 =>
              PH_SHIFT_UP_REQ <= '1';
              if PH_SHIFT_DONE then
                    next_state <= phase_return1_wait;
              end if;

          when phase_return1_wait =>
              if PH_SHIFT_DONE = '0' then
                    next_state <= phase_process;
              end if;


        when phase_process   =>
                --- if phase up made things worse (higher coeff)
            if phase_down_calc_res >= phase_up_calc_res then
                phase_up_counter_dec <= '1';
            else
                phase_up_counter_inc <= '1';
            end if;

            if phase_shift_counter = "1111" then
                next_state <= phase_decide;
            else
                next_state <= idle;
            end if;

        when phase_decide    =>
            if phase_up_counter > 15 then

                     PH_SHIFT_DWN_REQ <= '1';
                if PH_SHIFT_DONE then
                    phase_up_counter_rst <= '1';
                    next_state <= idle;

                end if;
            elsif phase_up_counter < -15 then
                PH_SHIFT_UP_REQ <= '1';
                if PH_SHIFT_DONE then
                    phase_up_counter_rst <= '1';
                    next_state <= idle;

                end if;
               else
                    phase_up_counter_rst <= '1';
                    next_state <= idle;
            end if;

        when others =>
            next_state <= reset_state;

    end case;
end process;

phase_shift_counter_inc : process(all)
begin
    if(rising_edge(CLK)) then


        if phase_up_res_save then
            phase_up_calc_res   <= signed(CALC_RES);
```

```
        else
            phase_up_calc_res   <= phase_up_calc_res;
        end if;
        if phase_dwn_res_save then
            phase_down_calc_res <= signed(CALC_RES);
        else
            phase_down_calc_res <= phase_down_calc_res;
        end if;


        if phase_up_counter_rst then
            phase_up_counter <= (others => '0');
        elsif phase_up_counter_inc then
            phase_up_counter <= phase_up_counter + 1;
        elsif phase_up_counter_dec then
            phase_up_counter <= phase_up_counter - 1;
        else
            phase_up_counter <= phase_up_counter;
        end if;


        if next_state = phase_process then
            phase_shift_counter <= phase_shift_counter + 1;
        elsif next_state = phase_decide then
            phase_shift_counter <= (others => '0');
        end if;
    end if;
end process;


end architecture rtl;
```

5 priedas. Mean_calculator.vhd kodas

```
--************************************************************************
--! @file   Mean_calculator.vhd
--! @brief  Calculates the mean of given data; data length is calculated by 2^n,
where n is G_DATA_LEN_POW2
--! ************************************************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Mean_calculator is
   generic (
    G_DATA_LEN_POW2 : natural := 1;
    G_DATA_WIDTH    : natural := 12
    );
   port (
    CLK              : in std_logic;
    RESET_N          : in std_Logic;
    DATA_VALID       : in std_logic;
    DATA_IN          : in std_logic_vector(G_DATA_WIDTH-1 downto 0);

    DATA_OUT         : out std_logic_vector(G_DATA_WIDTH-1 downto 0);
    DATA_OUT_UPDATED : out std_logic

   );
end entity Mean_calculator;

architecture rtl of Mean_calculator is

constant counter_max  : unsigned(G_DATA_LEN_POW2-1 downto 0) := (others => '1');
signal counter        : unsigned(G_DATA_LEN_POW2-1 downto 0);

signal sum            : signed  (G_DATA_WIDTH + G_DATA_LEN_POW2 - 1 downto 0);
signal internal_value : signed  (G_DATA_WIDTH + G_DATA_LEN_POW2 - 1 downto 0);


begin

sum <= internal_value + to_signed(to_integer(signed(DATA_IN)),G_DATA_WIDTH +
G_DATA_LEN_POW2);

process(all)
begin
    if(rising_edge(CLK)) then
        if(RESET_N) then
            if(DATA_VALID) then
                if(counter = counter_max) then
                    DATA_OUT_UPDATED <= '1';
                    DATA_OUT         <= std_logic(sum(sum'LEFT)) &
std_logic_vector(sum(G_DATA_WIDTH + G_DATA_LEN_POW2 - 2 downto
G_DATA_LEN_POW2));
                    internal_value   <= (others => '0');
                    counter          <= (others => '0');
                else
                    DATA_OUT_UPDATED <= '0';
                    internal_value   <= sum;
                    counter          <= counter + 1;
                end if;
            end if;
        else
            DATA_OUT_UPDATED <= '0';
            DATA_OUT         <= (others => '0');
            internal_value   <= (others => '0');
```

```
            counter          <= (others => '0');
        end if;
    end if;
end process;


end architecture rtl;
```

6 priedas. moving_median.vhd kodas

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity moving_median is

    generic
    (
        G_DATA_WIDTH : natural := 8;
        G_FRACTIONAL : boolean := false;
        G_FRACTION_DEPTH : natural := 1
    );

    port
    (
        CLK     : in  std_logic;
        INPUT      : in  std_logic_vector   ((G_DATA_WIDTH-1) downto 0);
        INPUT_VAL: in  std_Logic;

        OUTPUT   : out std_logic_vector ((G_DATA_WIDTH-1) downto 0);
        DELTA       : in  std_logic_vector   (13 downto 0)
    );

end entity;

architecture rtl of moving_median is

signal internal_Val : unsigned(G_DATA_WIDTH-1 downto 0);
signal uns_input : unsigned((G_DATA_WIDTH-1) downto 0);
signal uns_delta : unsigned(DELTA'LEFT downto 0);

signal f_internal_val : unsigned(G_DATA_WIDTH-1 + G_FRACTION_DEPTH downto 0);

signal up_count : unsigned(31 downto 0);
signal down_count : unsigned(31 downto 0);
signal master_count : unsigned(31 downto 0);

begin

uns_delta <= unsigned(DELTA);
uns_input <= unsigned(INPUT);

master_count <= up_count or down_count;

FRACTIONAL : if G_FRACTIONAL=true  generate

internal_Val <= f_internal_val(f_internal_val'LEFT downto G_FRACTION_DEPTH);

process(all)
    begin
        if(rising_edge(CLK)) then
            if(INPUT_VAL) then
                if(uns_input > internal_Val) then
                    f_internal_val <= f_internal_val + uns_delta;
                          up_count <= up_count + 1;
                          down_count <= (others => '0');
                elsif(uns_input < internal_Val) then
                    f_internal_val <= f_internal_val - uns_delta;
                          up_count <= (others => '0');
                          down_count <= down_count + 1;
                end if;
```

```
            end if;
        end if;

    end process;
    OUTPUT <= std_logic_vector(internal_val);

end generate;


NOT_FRACTIONAL : if G_FRACTIONAL=false  generate

    process(all)
    begin

        if(rising_edge(CLK)) then
            if(INPUT_VAL) then
                if(uns_input > internal_Val) then
                    internal_val <= internal_val + uns_delta;
                elsif(uns_input < internal_Val) then
                    internal_val <= internal_val - uns_delta;
                end if;
            end if;
        end if;

    end process;
    OUTPUT <= std_logic_vector(internal_val);

end generate;

end rtl;
```

7 priedas. Data_delay.vh kodas

```
--*****************************************************************************
--! @file  Data_delay.vhd
--! @brief Delays data;
--!        G_DELAY_LEN must be at least 2
--!*****************************************************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Data_delay is
   generic (
    G_DELAY_LEN : natural := 2;
    G_DATA_WIDTH: natural := 12
    );
   port (
    CLK            : in  std_logic;
    DATA_IN        : in  std_logic_vector(G_DATA_WIDTH-1 downto 0);
    DATA_VALID     : in std_logic;
    DATA_OUT       : out std_logic_vector(G_DATA_WIDTH-1 downto 0)
   );
end entity Data_delay;

architecture rtl of Data_delay is

type T_DELAY_TYPE is array(G_DELAY_LEN - 1 downto 0) of
std_logic_vector(G_DATA_WIDTH-1 downto 0);
signal delay_signal : T_DELAY_TYPE;

begin

process(all)
begin
    if(rising_edge(CLK)) then
        if(DATA_VALID) then
            delay_signal(G_DELAY_LEN-1 downto 1) <= delay_signal(G_DELAY_LEN-2
downto 0);
            delay_signal(0)                      <= DATA_IN;
        end if;
    end if;
end process;

DATA_OUT <= delay_signal(G_DELAY_LEN-1);

end architecture rtl;
```