TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Elektroenergeetika ja mehhatroonika instituut

VÕRGUSAGEDUSEGA VAHELDUVVOOLU SIGNAALI SUURANDMETE REAALAJALINE MÕÕTMINE JA ANALÜÜS

REAL-TIME MEASUREMENT AND ANALYSIS FOR THE BIG DATA OF UTILITY FREQUENCY ALTERNATING CURRENT

BAKALAUREUSETÖÖ

Üliõpilane: Toomas Erik Anijärv

Üliõpilaskood: 179491EAAB

Juhendaja: Noman Shabbir, doktorant

Tallinn, 2020

(Tiitellehe pöördel)

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 201…..

Autor: .....
/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 201…..

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 201… .

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

LÕPUTÖÖ LÜHIKOKKUVÕTE

Autor: Toomas Erik Anijärv *Lõputöö liik:* Bakalaureusetöö

Töö pealkiri: Võrgusagedusega vahelduvvoolu signaali suurandmete reaalajaline mõõtmine ja analüüs

Kuupäev: 20.05.2020 55 *lk*

Ülikool: Tallinna Tehnikaülikool

Teaduskond: Inseneriteaduskond

Instituut: Elektroenergeetika ja mehhatroonika instituut

Töö juhendaja(d): Noman Shabbir

Töö konsultant (konsultandid): Lauri Kütt

Sisu kirjeldus:

Tänasel päeval kõik meie tehnoloogia põhineb laitmatul informatsiooni kättesaadavusel. Andmevahetus toimub suurte andmehulkade kaupa ning väga palju just reaalajas. Selles uurimuses luuakse süsteem, mille eesmärk on laitmatu täpsusega lugeda võrgusagedusega vahelduvvoolu signaali ning analüüsida selle karakteristikuid, sest elektrivõrk on alus inimkonna arenenud infrastruktuurile.

Töö sihiks on uurida erinevaid meetodeid võrgusagedusega vahelduvvoolu suurandmete analüüsiks. Soovitud eesmärgi täitmiseks luuakse elektrooniline andmehõive süsteem, mis võimaldab katsetada ja mõõta vahelduvvoolu signaali. Süsteemi loomisel kasutatakse erinevaid komponente nagu näiteks toiteallikas, analoog-digitaalmuundur, mikrokontroller ja personaalarvuti ning kirjutatakse süsteemi toimimiseks programm. Saadud tulemusi analüüsitakse signaalitöötluses tuntud meetoditega.

See töö on eeldus suuremale projektile, mille visioon on luua süsteem, mis tuvastab ära automaatselt, kui toimub võrgus suuremad sageduskõikumised ja uue seadme ühendamisel antakse teada, et uus seade on ühendatud elektriringi.

Märksõnad: elektroonika, signaalitöötlus, suurandmed, vahelduvvool, elektrivõrk, analoog-digitaalmuundamine, reaalajaline analüüs.

ABSTRACT

Author: Toomas Erik Anijärv *Type of the work:* Bachelor Thesis

Title: Real-time measurement and analysis for the big data of utility frequency alternating current

Date: 20.05.2020 55 *pages*

University: Tallinn University of Technology

School: School of Engineering

Department: Department of Electrical Power Engineering and Mechatronics

Supervisor(s) of the thesis: Noman Shabbir

Consultant(s): Lauri Kütt

Abstract:

In the modern world, all our technology depends on flawless accessibility of information. Data is exchanged in real-time in big data chunks. In this thesis, the main objective is to create a system which can read utility frequency alternating current signal with high precision and analyze its characteristics. That is because electrical grid is the foundation of the whole infrastructure.

This study aims to research different methods of analyzing big data of utility frequency alternating current signal. To reach this goal, a data acquisition system is created for measuring the signal using electronic components like power supply, analog-digital converter, microcontroller and personal computer and by writing a program algorithm to run it. Received results are used for an analysis by using different signal processing methods.

This thesis is a precondition to a larger project with a vision of building a system which automatically recognizes frequency changes and connections with new devices in the electrical circuit.

Keywords: electronics, signal processing, big data, data acquisition, alternating current, electrical grid, analog-digital converting, real-time analysis.

LÕPUTÖÖ ÜLESANNE

Lõputöö teema:	**Võrgusagedusega vahelduvvoolu signaali suurandmete reaalajaline mõõtmine ja analüüs**
Lõputöö teema inglise keeles:	**Real-time measurement and analysis for the big data of utility frequency alternating current**
Üliõpilane:	**Toomas Erik Anijärv, 179491EAAB**
Eriala:	**Elektroenergeetika ja mehhatroonika, EAAB**
Lõputöö liik:	**bakalaureusetöö**
Lõputöö juhendaja:	**Noman Shabbir**
Lõputöö ülesande kehtivusaeg:	**20.05.2020**
Lõputöö esitamise tähtaeg:	**20.05.2020**

_____	_____	_____
Üliõpilane (allkiri)	Juhendaja (allkiri)	Õppekava juht (allkiri)

1. Teema põhjendus

Tänasel päeval kõik meie tehnoloogia põhineb laitmatul informatsiooni kättesaadavusel. Andmevahetus toimub suurte andmehulkade kaupa ning väga palju just reaalajas. Selles uurimuses luuakse süsteem, mille eesmärk on laitmatu täpsusega lugeda võrgusagedusega vahelduvvoolu signaali ning analüüsida selle karakteristikuid, sest elektrivõrk on alus inimkonna arenenud infrastruktuurile.

2. Töö eesmärk

Töö eesmärgiks on uurida erinevaid meetodeid võrgusagedusega vahelduvvoolu suurandmete analüüsiks.

3. Lahendamisele kuuluvate küsimuste loetelu:

Kuidas luua toimiv süsteem, mis mõõdaks pidevsuurustena signaali pinge väärtuseid ning muundaks need numbrilisteks väärtusteks kasutades analoog-digitaalmuundamise põhitõdesid?

Milliseid meetodeid on kõige efektiivsem rakendada suurandmete kujul oleval vahelduvvoolu signaalil?

4. Lähteandmed

Uurimuse käigus luuakse andmehõive süsteem, mis võimaldab andmeid lugeda vahelduvvooluallikast, ning neid kasutatakse analüüsi meetodite uurimisel. Teoreetiliste teadmiste laiendamiseks kasutatakse võrgupõhist informatsiooni.

5. Uurimismeetodid

Töö soovitud eesmärkide täitmiseks luuakse elektrooniline andmehõive süsteem, mis võimaldab katsetada ja mõõta vahelduvvoolu signaali. Süsteemi loomisel kasutatakse erinevaid komponente nagu näiteks toiteallikas, analoog-digitaalmuundur, mikrokontroller ja personaalarvuti ning süsteemi programmi kirjutamisel kasutatakse Python programmeerimiskeelt. Saadud tulemusi analüüsitakse signaalitöötluse meetoditega (Fourier' teisendus ja *wavelet* funktsioon). Meetodite rakendamine toimub Matlab tarkvara keskkonnas.

6. Graafiline osa

Töös esineb 24 joonist, 4 tabelit ning 4 võrrandit. Samuti esineb töö sisus ka lõikeid programmikoodidest. Lisadesse on lisatud kõik neli täielikku programmikoodi ning kaks fotot elektroonilisest süsteemist.

7. Töö struktuur

Esimene peatükk „Andmehõive süsteemi arenduskäik" koosneb alapeatükkidest „Analoog-digitaalmuundamine", „Esmane katsesüsteem", „Edasiarendatud süsteem" ja „Järeldus".
Teine peatükk „Andmetöötlus ja -analüüs" koosneb alapeatükkidest „Muutuva signaali genereerimine", „Analüüsi meetodid ja protsessi tutvustus", „Meetodite rakendamine" ja „Signaali analüüsi järeldused".

8. Kasutatud kirjanduse allikad

Kasutatud kirjandus koosneb 17st võrgupõhisest allikast ning kahest raamatust. Peamiselt on allikate hulgas elektrooniliste komponentide andmelehed ja teaduslikud artiklid.

9. Lõputöö konsultandid

Töö käigus oli peale juhendaja, Noman Shabbiri, abiks ka kaasjuhendaja/konsultant Lauri Kütt.

10. Töö etapid ja ajakava

Andmehõive süsteemi katsesüsteemi loomine (20. märts 2020)
Andmehõive lõpliku edasiarendatud süsteemi loomine (17. aprill 2020)
Signaali analüüsi meetodite uurimine (24. aprill 2020)
Analüüsi meetodite rakendamine ja tulemuste järeldused (8. mai 2020)
Töö protsessi ja analüüsi meetodite tulemuste kirjeldamine ja vormistamine (19. mai 2020)

SISUKORD

EESSÕNA

Antud lõputöö valmimine sai teoks tänu Tallinna Tehnikaülikooli Elektroenergeetika ja Mehhatroonika instituudile. Autor tänab ka eduka lõputöö protsessi käigus abiks olnud kahte juhendajat – Noman Shabbir ja Lauri Kütt.

Kõik kasutatud materjalid ning töövahendid olid kättesaadavad ülikoolist. Töö tegemise ajal oli Eesti Vabariigis kehtestatud riigisisene eriolukord, kuid kõik sellega seotud probleemid leidsid lahenduse just tänu ülikooli heale eriolukorraga toimimise strateegiatele.

LÜHENDITE JA TÄHISTE LOETELU

ADC	*Analog-digital converter*, analoog-digitaalmuundur
LSB	*Least significant bit*, binaarses süsteemis kõige madalam bait
RPI	*Raspberry Pi*, mikroarvuti/mikrokontroller
CSV	*Comma-separated values,* komadega eraldatud andmete failiformaat
I2C	*Inter-Integrated Circuit,* jadasiin
SPI	*Serial Peripheral Interface*, jadasiin
GPIO	*General-purpose input/output*, *pin* ehk ühendusnõel
ATX	*Advanced Technology eXtended*, toiteploki konfiguratsioon
MISO	*Master input slave output*, ülemseadme andmete sisend
MOSI	*Master output slave input*, ülemseadme andmete väljund
SCLK	*Serial Clock*, ülemseadme genereeritud taktsignaal
GND	*Ground*, ühendus maaga
RMS	*Root mean square*, keskmise pinge ruutjuure tulem
AM	*Amplitude modulation*, signaali parameetrite muutmine madala sagedusega signaali rütmis
FFT	*Fast Fourier Transform*, Fourier' teisenduse diskreetne teisendus

SISSEJUHATUS

Tänasel päeval kõik meie tehnoloogia põhineb laitmatul informatsiooni kättesaadavusel. Andmevahetus toimub suurte andmehulkade kaupa ning väga palju just reaalajas. Suurandmete töötlemisel peab olema kogu süsteem läbimõeldud, et ei tekiks andmekadusid ja andmed oleksid õiged.

Meie infrastruktuuri aluseks on elektrivõrk, mis annab tervele tehnoloogilisele võimekusele vundamendi. Ilma selleta poleks meil kättesaadavust mujal maailmas toimuvale informatsioonile ning andmevahetus poleks lihtsalt võimalik.

Selles uurimuses luuakse süsteem, mille eesmärk on just elektrivõrgust tuleneva vahelduvvoolu signaali uurimine. Lõputöö käigus ehitatakse elektrooniline süsteem koos programmiga, mis võimaldab võrgusagedusega vahelduvvoolu signaali lugeda ning võimalikult täpselt andmed reaalajaliselt salvestada. Elektroonilise süsteemi kõige fundamentaalsem komponent on analoog-digitaalmuundur. Selle abil teostatakse pidevsuurustena pinge ja voolu mõõtmine ning mõõtetulemus väljastatakse mikrokontrollerisse numbrilise väärtusena. Andmete analüüsimiseks proovitakse mitmeid meetodeid, mis on tuntud signaalianalüüsi valdkonnas. Analüüsi eesmärk on leida anomaaliaid signaalis, mis avalduvad moduleeritud sageduse, müra ja koormuse muutumisel.

Selles lõputöös on peamine siht luua elektrooniline süsteem, programm ning uurida välja parimad meetodid, millega rohkete andmetega signaali analüüsida. See töö on eeldus suuremale projektile, mille kallal töötatakse Tallinna Tehnikaülikooli Elektroenergeetika ja Mehhatroonika instituudis. Terve projekti visioon on luua süsteem, mis tuvastab ära automaatselt, kui toimub võrgus suuremad sageduskõikumised ja uue seadme ühendamisel antakse teada, et uus seade on ühendatud ahelasse.

Selles uurimuses alustatakse elektroonilise süsteemi ehitusega, luuakse programm ning analüüsitakse andmeid. Elektrooniline süsteem on ühendatud vahelduvvoolu väljundisse ning see loeb sissetulevaid andmeid kirjutatud programmi abil. Seejärel analüüsitakse andmeid ja katsetatakse erinevaid meetodeid, mis toimivad kõige efektiivsemalt võrgusagedusega signaali puhul.

1. ANDMEHÕIVE SÜSTEEMI ARENDUSKÄIK

Vahelduvvoolusuuruste analüüsiks numbrilisel kujul oli vaja kujundada ja üles ehitada spetsiifiliselt selle ülesande jaoks kohandatud mõõte- ja andmetöötlussüsteem. Selle süsteemi koostamise esimeses etapis oli sihiks kontseptsioonplatvorm, mis võimaldaks koguda mõõteandmeid kasutades valmis mooduleid. Platvormi abil oli eesmärgiks kontrollida analoog-digitaalmuundamise võimekust ja esmast andmetöötlus-suutlikkust.

Platvormi loomisel ja testimisel olid järgmised etapid:

1) Esialgse katseplatvormi ülesseadmine ja testimine;
2) Edasijõudnud arendusega platvormi koostamine ja testimine.

Platvormi arendusprotsessi alguses oli eesmärgiks analoog-digitaalmuundamise kontseptsiooni ja liidestamise katsetamine. Mõõdetavaks ja analüüsitavaks suuruseks oli alalispinge.

1.1 Analoog-digitaalmuundamine

Pinge, voolutugevuse, temperatuuri ja paljude teiste füüsiliste suuruste mõõtmisel tõlgendatakse nende reaalseid väärtuseid tänasel päeval üldjuhul digitaalselt. See tähendab, et mõõdetakse pidevsuurused, kuid mõõtetulemused väljastatakse numbrilise väärtusena. Selleks rakendatavat protsessi tuntakse kui analoog-digitaalmuundamist ning selle jaoks kasutatakse elektroonikas analoog-digitaalmuundureid.

1.1.1 Analoog-digitaalmuundamise protsessi kirjeldus

Üldiselt loeb andur või sensor mõne füüsilise suuruse reaalse väärtuse analoogsuurusena ning edastab selle ADC-le (ik *analog-digital converter*), mis omakorda muudab selle digitaalsuuruseks, et see oleks loetav näiteks arvutile või mõnele mikrokontrollerile. ADC tööd juhib temale määratud ülemseade ehk näiteks mikrokontroller. Suuruste mõõtmise ja muundamise protsessi kirjeldavad kolm peamist etappi – *sampling* ehk mõõtetulemuste võtmine, *quantization* ehk kvantifitseerimine ja *coding/encoding* ehk kodeerimine [1].

Võenduse *(*ik *sampling)* raames defineeritakse aeg (*sampling time*), mil mõõtetulemuste võtmine käib, ehk kui tihti loetakse sisse uus väärtus. Perioodi, mis määrab ajavahemikku kahe mõõtetulemuse vahepeal, nimetatakse diskreetimissageduseks f_s ja -perioodiks T_s (ik *sampling rate/frequency*). Sagedus ja periood on omavahel pöördvõrdelises seoses (Võrrand 1).

$$T_s = \frac{1}{f_s}$$

Võrrand 1. Diskreetimissageduse ja -perioodi pöördvõrdeline seos.

Diskreetimissageduse valikul tuleb arvestada ka Nyquisti teoreemi tingimusega, et mõõtetulemuste võtmise sagedus peaks olema vähemalt kaks korda suurem, kui mõõdetava signaali suurim sagedus. See tekitab olukorra, mida nimetatakse *over-sampling*uks ehk võetakse rohkem mõõtetulemusi, kui reaalselt neid füüsiliselt minimaalselt olukorra kirjeldamiseks vaja läheb. Sel juhul on loetud signaal maksimaalselt täpne [2].

Teine etapp ehk kvantifitseerimine jaotab digitaalselt tulemuste suurused gruppidesse. Kvantifitseerimine määrab tulemuste suuruste täpsuse – mida kõrgem kvantifitseerimise aste, seda täpsemalt vastavad loetud väärtused reaalsete füüsiliste suurustega. Tüüpiliselt kahendsüsteemi arvudega näidatud täpsusega süsteemis on kvantifitseerimise täpsus määratud LSB (ik *least significant bit*) valemiga (Võrrand 2).

$$LSB = \frac{U_{ref}}{2^N},$$

kus N-kahendbaitide arv.

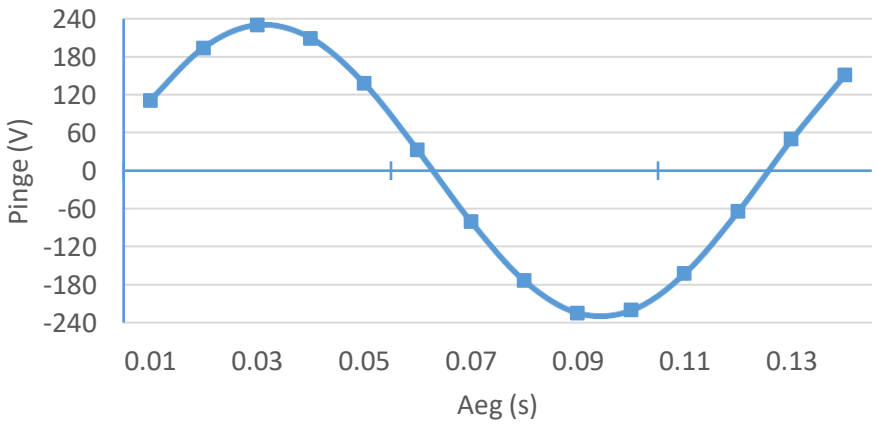
Võrrand 2. LSB valem kvantifitseerimise täpsuse määramiseks [3].

Viimaks kodeeritakse saadud tulemused arvutile või mikrokontrollerile loetavale kujule. Võetakse arvesse mõõtetulemuste võtmise sagedus ning kvantifitseerimise täpsus ja kodeeritakse need tüüpiliselt kahendsüsteemi ehk binaarsele kujule [1].

Analoog-digitaalmuundamise käigus määratud suurused esitatakse diskreetaja süsteemis, mis tähendab, et igal ajahetkel *t* võendatud pidevsuuruse väärtusele vastab diskreetsuurus järjekorranumbriga *n*. Tüüpiliselt on diskreetimise ajasamm püsiv. Selliselt saab vastavusse seada omavahel nii reaalse aja kui ka diskreetse aja.

1.1.2 Protsessi näide

Kogu kolme etapi kirjelduseks võib võtta näiteks suvalise vahelduvvoolu signaali (Joonis 1).

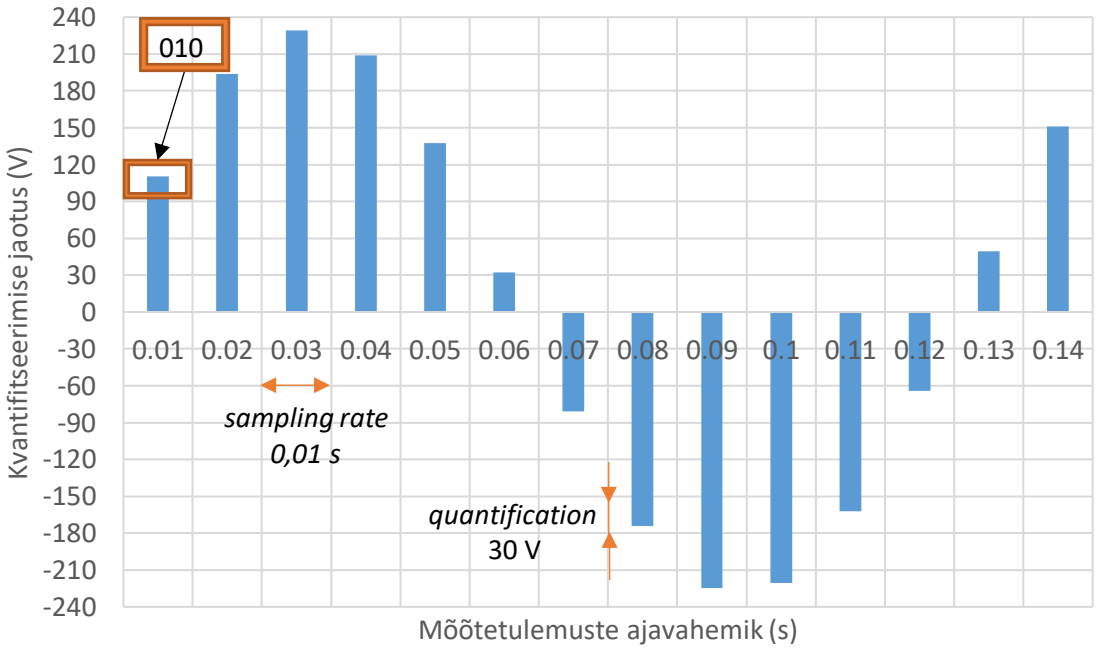


Joonis 1. Vahelduvvoolu signaal.

Kui soovida sellist füüsilist signaali muundada digitaalsele kujule, siis esiteks tuleb valida sobiv mõõtetulemuste võtmise sagedus. Selle jaoks arvestatakse kõigepealt Nyquisti teoreemi põhitõde ehk valitakse *sampling frequency* väärtuseks reaalsest sagedusest kaks korda suurem väärtus – oletades, et signaali sagedus on 50 Hz, siis diskreetimissagedus on 100 Hz. See tähendab, et mõõtetulemusi võetakse iga 10 ms tagant.

Teiseks tuleks kvantifitseerimise juures arvestada, mis täpsusega pinge väärtuseid soovitakse sisse lugeda. Praeguse näite puhul jaotatakse lihtsuse mõttes väärtused kaheksaks grupiks ehk mõõtmistulemuste täpsus on 30 V ehk ligikaudu 13 %.

Viimaks tõlgendatakse iga ajahetke ja mõõtetulemuse grupi alusel tulemused binaarsele kujule ehk näiteks esimene tulemus ajahetkel 10 ms on antud näite juures 120 V, sest neljas jaotis tähistab suurust 120 V. Arvestades kaheksa võimalikku taset kahendsüsteemi arvuna, saadakse kodeeringu tulemuseks 3-baidine tulemus – „010“ (Joonis 2).



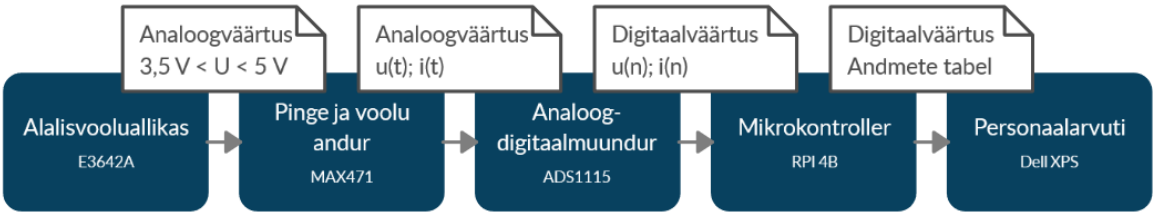
Joonis 2. Vahelduvvoolu signaali digitaalsele kujule muundamine.

1.2 Esmane katsesüsteem

Andmete kogumise baasüsteemiks oli valitud RPI (Raspberry Pi). See platvorm on piisavalt paindlik ja sisaldab erinevaid võimalusi, et liidestada omavahel andmete sisendseadmed (ADC), andmete töötlemisalgoritmid (Pythoni programmeerimiskeeles) ja andmete salvestusmeedia (SD-kaart) kui ka kohaliku võrgu ühendamise toe (WiFi, Ethernet) [4].

Esmase süsteemi ülesandeks oli RPI arendusplatvormi ülesseadmine alalissuuruste andmete lugemiseks. Sisendiks kasutati andurit, millest saadud analoogsignaal suunati analoog-digitaalmuundurisse. ADC väljund ühendati läbi I2C (*Inter-Integrated Circuit*) liidese RPI platvormiga.

Funktsionaalsus oli sarnane plaanitavale lõplikule edasiarendatud süsteemile – pinge ja voolu mõõtmine ning saadud väärtuste salvestamine. Eesmärk oli luua lahendus, mis mõõdab lihtsas ahelas pinge ja voolu väärtused ja jäädvustab saadud tulemused tabelina CSV (*comma-separated values*) faili. Lihtsustatud protsess on välja toodud joonisel Joonis 3.



Joonis 3. Katsesüsteemi protsess.

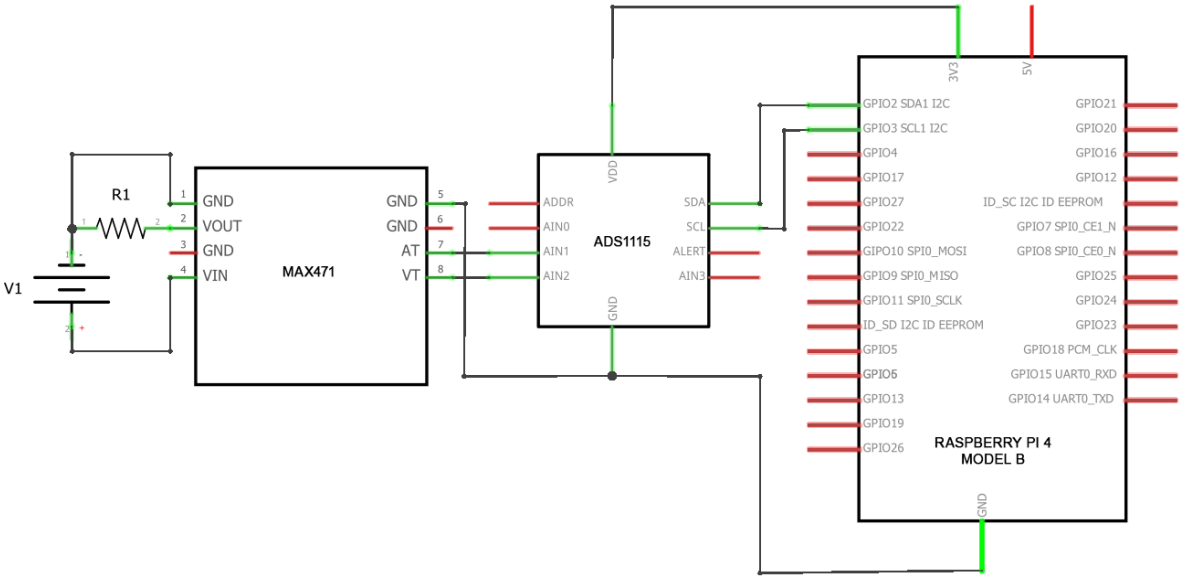
1.2.1 Elektriskeem

Baassüsteemi ning analoog-digitaalmuundamise testimiseks kasutati esialgu lihtsat elektriskeemi, mis koosnes alalisvoolu toiteplokist, pinge andurist, analoog-digitaalmuundurist ja RPI-st.

Mõõtetulemuste mõõtmiseks kasutati toiteallikat Agilent E3642A, mis valiti, sest sellega on lihtne muuta väljastavat pinge suurust. Siseneva alalisvoolu monitooringuks kasutati MAX471 andurit, mida kasutatakse selle täpsuse tõttu tihti akude ja toiteliinide pinge jälgmiseks [5]. ADC valikuks osutus Texas Instruments'i ADS1115 moodulil, sest selle implementeerimine RPI-ga on lihtne ning sellel on I2C protokolli valmisolek, mida selles testsüsteemis kasutatakse mikrokontrolleriga suhtlemiseks [6]. I2C on veel seetõttu hea, et seda on võimalik kasutada ka mikrokontrolleril, millel pole I2C liidese tuge, ning seda on kerge üles seada [7]. Hilisemaks edasiarendatud süsteemiks arvestati, et läheb vaja ka SPI (*Serial Peripheral Interface*) protokolli ja Raspberry Pi 4B mudelil on selle liidese tugi olemas. Samuti on RPI üks võimekamaid mikroarvuteid ning Linuxi operatsioonisüsteemi ja hea dokumentatsiooni olemasolu tõttu on sellel suurel hulgal kasutusalasid.

Toiteplokiga väljastatakse pinge vahemikus 3,5-5 V, mis valiti andurile sobiliku pinge vahemiku tõttu [5]. Andur võtab sisendväärtusena pinge ja voolu suuruse ja edastab selle analoog-

digitaalmuundurile. See omakorda muudab pinge analoogväärtuse numbriliseks väärtuseks ja annab selle edasi RPI-le kasutades I2C protokolli. Joonis 4 kuvab mainitud katsesüsteemi elektriskeemi.



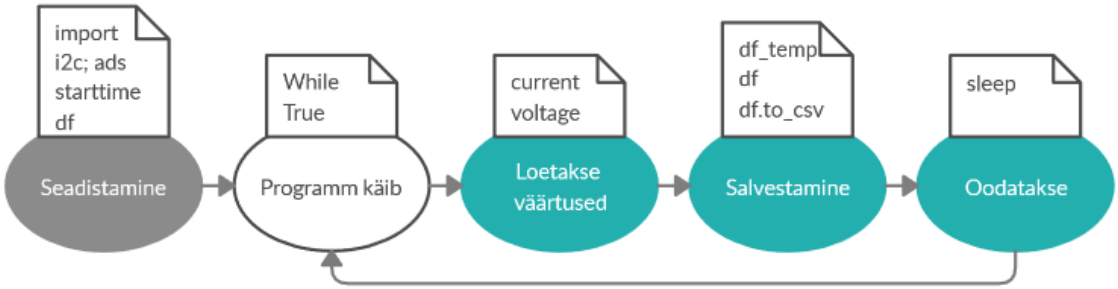
Joonis 4. Katsesüsteemi elektriskeem.

1.2.2 Programm

Programmi eesmärk oli võimaldada andmevahetust mikrokontrolleri ja analoog-digitaalmuunduriga. Programmeerimiskeeles valiti Python, sest sellel oli tugi kõikidele süsteemi toimimiseks vajalikele teekidele ning sellel on hea võrgupõhine dokumentatsioon mikrokontrollerite programmeerimiseks.

Esiteks tuli RPI süsteemikonfiguratsioonis sätestada vaikimisi ka I2C protokoll, et RPI GPIO-d (*general-purpose input/output*) oleksid seadistatud vastavale kommunikatsioonirežiimile. Programmikoodi funktsionaalsuse tagamiseks pidi kasutama mitmeid teeke. Nende hulgas olid teegid *board* ja *busio*, mis sisaldavad RPI peamisi funktsioone, *adafruit* teegid, mis annavad toe töötamiseks varasemalt mainitud ADC-ga ning *time*, *pandas*, *datetime* teegid, mis on andmete salvestamiseks.

Programm toimis nii kaua, kuni see manuaalselt sulgeti. Esialgu loeti voolu ja pinge väärtused I2C sisendist. Seejärel need lisati andmete tabelisse (ik *dataframe*) ning salvestati CSV faili. Enne uut protsessi kordust oodati 100 ms. Joonis 5 kirjeldab programmi algoritmi.



Joonis 5. Katsesüsteemi programmi algoritm.

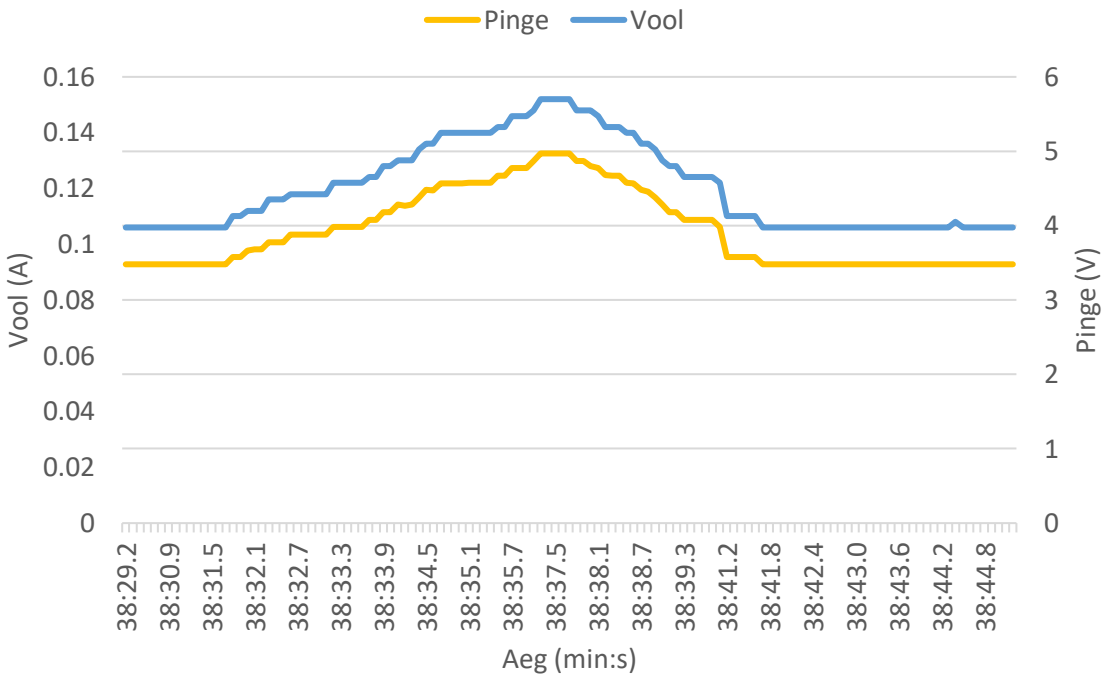
```
i2c = busio.I2C(board.SCL, board.SDA)
ads = ADS.ADS1115(i2c)
starttime = time.time()
df = pd.DataFrame()

while True:
    current = AnalogIn(ads, ADS.P1).voltage
    voltage = AnalogIn(ads, ADS.P2).voltage*5

    df_temp = pd.DataFrame({'aeg': [dt.datetime.now()], 'pinge':
[voltage], 'vool': [current]})
    df = df.append(df_temp)

    df.to_csv(r'/home/pi/Desktop/voltagedata/sensordata.csv',
index=False)

    time.sleep(0.1000 - ((time.time() - starttime) % 0.1000))
```

1.2.3 Tulemused

Süsteemi kontrollimiseks muudeti potentsiomeetriga väljastava pinge väärtust vahemikus 3,5-5 V. Salvestatud CSV failist on toodud 1-sekundiline ajavahemik tulemustest tabelis (Tabel 1) ning graafikul on kuvatud kõik tulemused (Joonis 6).

Aeg (min:s)	Pinge (V)	Vool (A)
38:38.4	4.670143	0.142004
38:38.5	4.58014	0.140004
38:38.6	4.570139	0.140004
38:38.7	4.480137	0.136004
38:38.8	4.460136	0.136004
38:38.9	4.380134	0.134004
38:39.0	4.280131	0.130004
38:39.1	4.180128	0.128004
38:39.2	4.180128	0.128004
38:39.3	4.080125	0.124004
38:39.4	4.080125	0.124004

Tabel 1. Testsüsteemi tulemused.



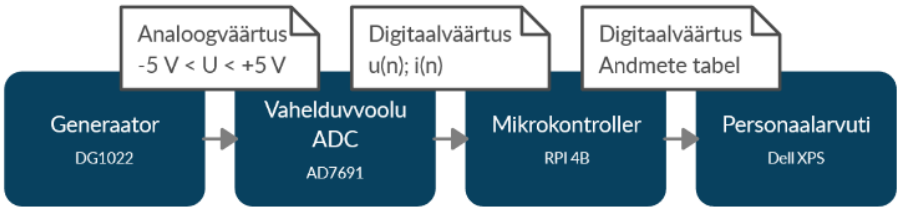
Joonis 6. Katsesüsteemi salvestatud tulemuste pinge ja voolu muutus ajas.

Saadud tulemusi võrreldi toiteallikast väljastatud suurustega ja need tulemused vastasid omavahel ehk järeldati süsteemi toimimist. Selgus, et RPI sobib kasutamiseks mikrokontrollerina ka tulevas edasiarendatud süsteemis ning analoog-digitaalmuundamine toimis vigadeta.

1.3 Edasiarendatud süsteem

Edasiarendatud süsteemi eesmärk oli luua andmehõive platvorm, mis võimaldaks lugeda vahelduvvoolu pinge väärtuseid ja need salvestada. Selles süsteemis kasutati SPI protokolli, sest selle andmete vahetamine on kiirem ning see on mürale vähem vastuvõtlik kui I2C protokoll [8]. SPI kiiruse ja müra taluvus olid olulised, sest vahelduvvoolus toimub ajas rohkem muutuseid, kui eelnevas süsteemis mõõdetud alalisvoolus.

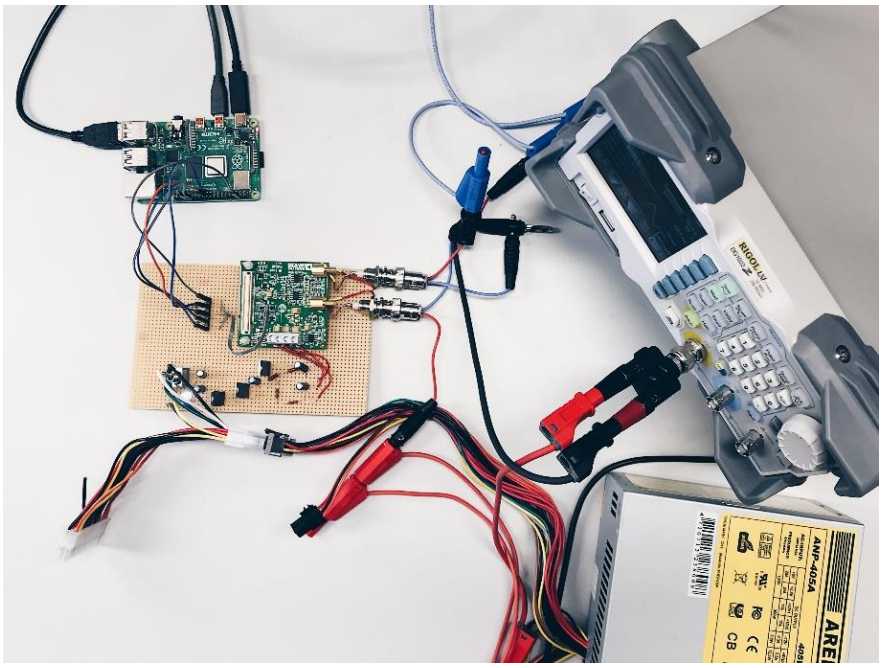
Töö protsess nägi välja järgnevalt. Analoog-digitaalmuundur loeb generaatorist vahelduvvoolu pinge pidevväärtused vahemikus -5 kuni +5 V. Seejärel ADC muudab analoogväärtuse digitaalväärtuseks ning saadab tulemused RPI-le SPI protokolli abil. Kui digitaalväärtused jõuavad RPI-le, töötleb programm tulemused andmete tabeliks ja need saadetakse arvutile CSV failina. Mainitud protsessi kirjeldab Joonis 7.



Joonis 7. Edasiarendatud süsteemi protsess.

1.3.1 Elektriskeem

Edasiarendatud süsteemi elektriskeemis kasutati vahelduvvoolu generaatorit, analoog-digitaalmuundurit koos selle toiteallikaga ja mikrokontrollerit (Joonis 8).



Joonis 8. Foto elektroonilisest süsteemist.

Generaatoriks valiti Rigol DG1022, sest selle graafiline kasutajaliides võimaldab väljastava signaali karakteristikuid kergelt muuta. Katsete käigus mõõdeti enamasti vahelduvvoolu signaali sagedusega 0,1 Hz ja pinge amplituudväärtustega +5 V ja -5 V.

Vahelduvvoolu analoog-digitaalmuundurina kasutati Analog Devices'i AD7691 diferentsiaalset muundurit, sest sellel on SPI liides ja pinge mõõtmise vahemik on +5 kuni -5 V [9]. ADC toiteallikaks valiti ATX (*Advanced Technology eXtended*) toiteplokk, sest AD7691 kasutamiseks on vajalikud sisendpinged +8 V, +5 V ja -2 V. ATX-i väljundist võeti +12 V ja -5 V, mis muudeti nõutud pinge väärtusteks kasutades erinevaid kondensaatoreid (kaks 0,1 mF ja kolm 47 µF mahtuvusega),

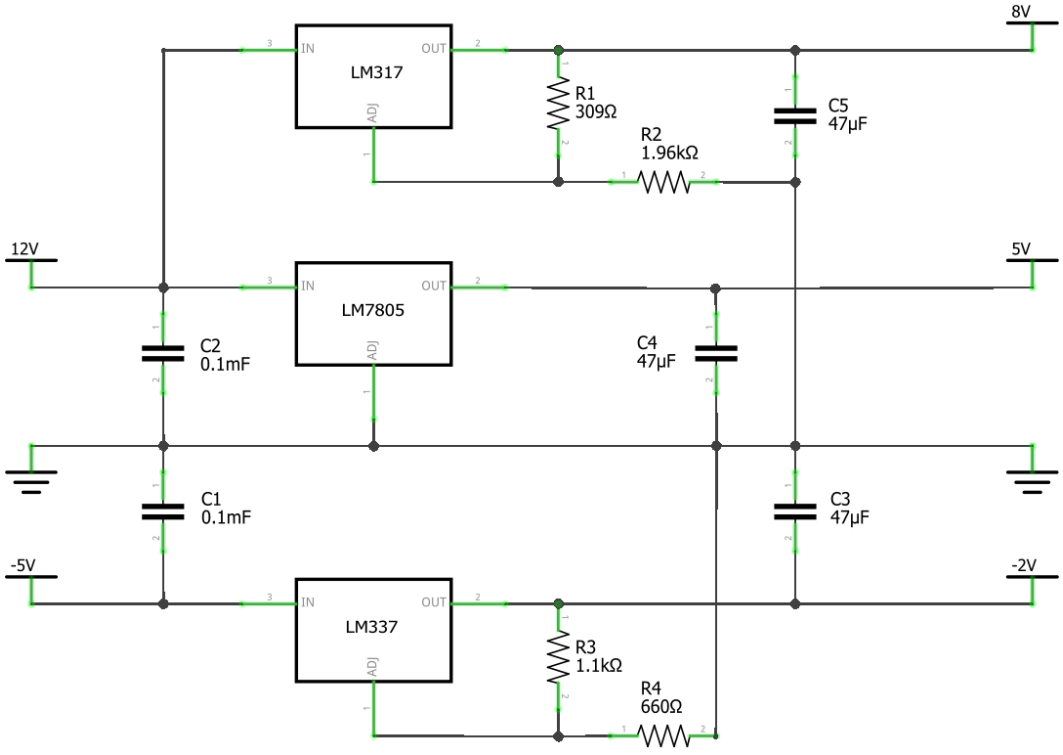
takisteid (väärtustega 309 Ω, 1,96 kΩ, 1,1 kΩ ja 660 Ω) ja regulaatoreid (LM317, LM7805 ja LM337). Pingete muundamiseks kasutatud regulaatorid valiti eesmärgiga luua võimalikult optimaalne ja vähestest komponentidest koosnev elektriskeem ADC toiteplokiga ühendamiseks. LM317 regulaatoriga vajaliku väljundpinge U_{OUT} leidmiseks kasutati selle regulaatori andmelehel antud valemit (Võrrand 3). LM337 regulaatori abil soovitud väljundpinge saavutamiseks tuli kasutada takisti R_2 leidmiseks ühte valitud takistit ning soovitud väljundpinge väärtust (Võrrand 4). LM7805 regulaator takisteid pinge muundamiseks ei vajanud. ADC toiteplokiga ühendamiseks vajalikku elektriskeemi näeb skeemilt Joonis 9.

$$U_{OUT} = 1{,}25\ V \cdot \left(1 + \frac{R_2}{R_1}\right)$$

Võrrand 3. LM317 regulaatori väljundpinge valem [10].

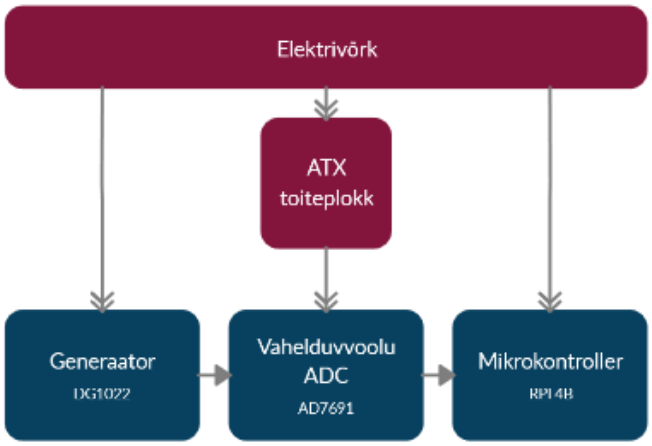
$$R_2 = R_1 \cdot \left(\frac{U_{OUT}}{-1{,}25} - 1\right)$$

Võrrand 4. LM337 regulaatori väljundpinge valem [11].



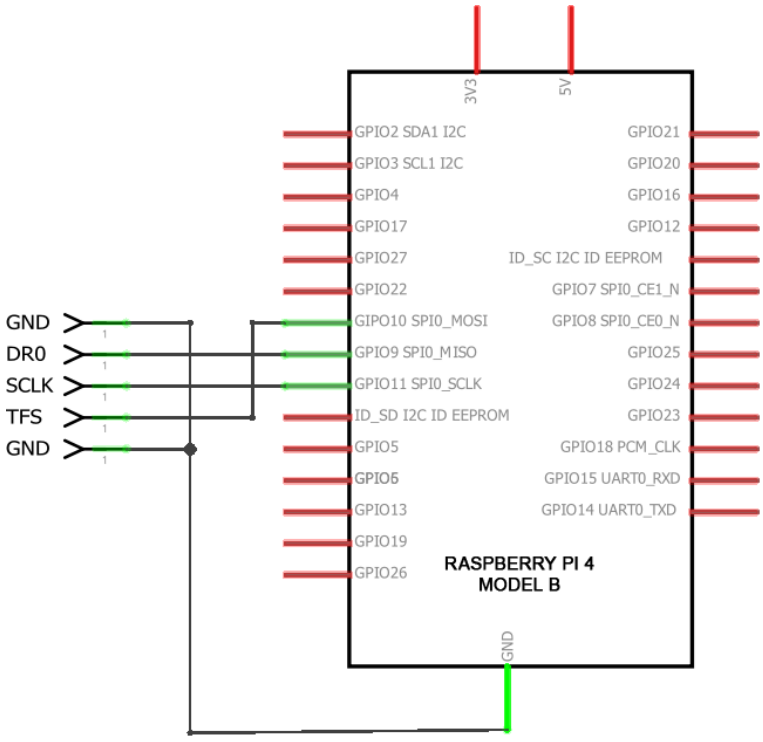
Joonis 9. Analoog-digitaalmuunduri ühendus ATX-tüüpi toiteplokiga.

Elektrivõrguga otseühenduses on DG1022 generaator, mille pinge väärtust mõõdetakse, ning Raspberry Pi 4B mikrokontroller. AD7691 analoog-digitaalmuunduri toitmiseks kasutatud ATX-tüüpi toiteplokk on ühendatud omakorda otse elektrivõrku (Joonis 10).



Joonis 10. Edasiarendatud süsteemi komponentide elektrienergiaga varustamine.

Viimane element elektriskeemis oli mikrokontroller, mis oli ühendatud analoog-digitaalmuunduriga. Nende kahe komponendi kommunikatsiooniviisiks oli SPI protokoll, mis tähendab, et RPI GPIO-idest kasutati SPI-le orienteeritud sisendeid ja väljundeid – GPIO9 MISO (ik *master input slave output*) ehk andmete sisend, GPIO10 MOSI (ik *master output slave input*) ehk andmete väljund ja GPIO11 SCLK (ik *serial clock*) ehk seeriakell/genereeritud taktsignaal [4]. ADC poole pealt on ühendatud kaks GND (ik *ground*), SCLK, TFS ja DR0 ühendusnõelad (Joonis 11). TFS ning DR0 on signaali saamise ja saatmise ühendusnõelad.



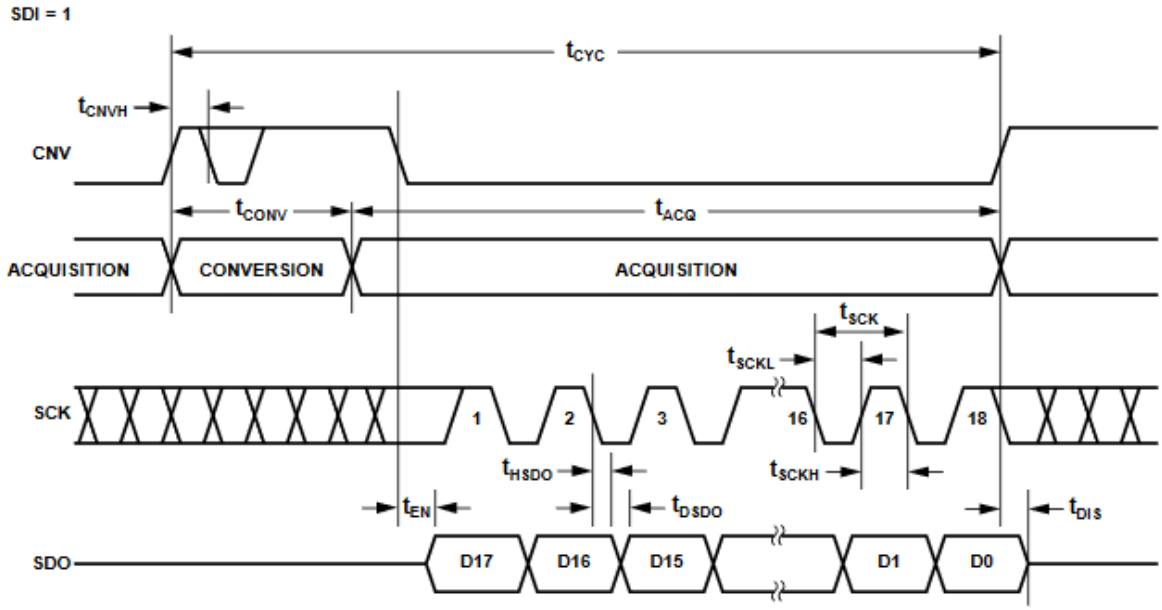
Joonis 11. Raspberry Pi ühendus AD7691 analoog-digitaalmuunduriga.

1.3.2 Programm

Programmi kirjutamiseks kasutati jätkuvalt Python programmeerimiskeelt ning SPI protokolli abil suhtlemiseks tuli RPI süsteemikonfiguratsioonis sätestada vaikimisi kommunikatsioonirežiimiks SPI, et Raspberry Pi GPIO-d oleksid seadistatud seda protokolli kasutama.

RPI SPI liidese kasutamiseks kasutati RPI ametlikku GPIO teeki ja *spidev* teeki. Signaali õigeaegseks jäädvustamiseks rakendati *time* ja *threading* teekide funktsioone ning andmete jäädvustamiseks *pandas*, *datetime* ja *math* teeke. Programmis defineeriti GPIO-ide asukohad ja nende tüübid, SPI protokolli seaded ja maksimaalne andmevahetussagedus. GPIO-ide asukohad mikrokontrolleril nimetati nende füüsiliste asukohtade järgi.

Programmi esimene funktsioon oli vahelduvvoolu signaali digitaalsete väärtuste sisse lugemine. Selle loomisel tuli arvesse võtta AD7691 andmelehel olevat ajajoont, mis kuvas graafiliselt tingimused mikrokontrolleri programmile muunduriga andmete vahetuseks (Joonis 12).



Joonis 12. AD7691 andmevahetuse ajajoon [9].

Signaali lugemise funktsiooni eesmärk oli läbi kommunikatsiooni SPI protokolli abil analoog-digitaalmuunduriga saada baitidena pinge väärtus ning see muuta täisarvuliseks arvuks ja kuueteistkümnendarvuks. Seejärel lisati saadud väärtus andmete tabelisse koos hetke aja väärtusega. Andmelehel olevat ajajoon arvesse võttes sätestati funktsioonis kõigepealt TFS-i väärtus madalaks, oodati 5 µs, loeti kolm korda 8 baiti muutujasse ning seejärel sätestati TFS tagasi kõrgeks. Selle tulemusena saadi listina baitide väärtused ja need arvutati ümber täis- ja kuueteistkümnendarvuks. Kuueteistkümnendarvu kujul signaali väärtus oli vajalik, et kontrollida tulemuste vastavust ADC andmelehel olevate suurustega ning veenduda nende korrektsuses

(Joonis 13). Täisarv korrutati andmelehelt võetud LSB väärtusega, milleks on 38,15 µV, ja saadi pinge numbriline väärtus voltides [9]. Viimaks loodi veel andmete jäädvustamiseks tabel, milles olid reaalajalised pinge väärtused, ja lisatabel, mis sisaldasid iga viie sekundi taguseid pinge keskmiseid ja RMS (*root mean square*) väärtuseid. Reaalajalisi andmeid kasutati arvestamaks ADC analoog-digitaalmuundamise kvantifitseerimise täpsust.

Description	Analog Input V_{REF} = 5 V	Digital Output Code (Hex)
FSR – 1 LSB	+4.999962 V	0x1FFFF[1]
Midscale + 1 LSB	+38.15 µV	0x00001
Midscale	0 V	0x00000
Midscale – 1 LSB	–38.15 µV	0x3FFFF
–FSR + 1 LSB	–4.999962 V	0x20001
–FSR	–5 V	0x20000[2]

[1] This is also the code for an overranged analog input (V_{IN+} – V_{IN-} above V_{REF} – V_{GND}).
[2] This is also the code for an underranged analog input (V_{IN+} – V_{IN-} below V_{GND}).

Joonis 13. AD7691 sisendi pinge väärtused ja väljundi koodid [9].

```
df = pd.DataFrame()
df_temp = pd.DataFrame()
df_meta = pd.DataFrame()

def signal_read():
    GPIO.output(TFS, GPIO.LOW)
    time.sleep(5e-6)
    bytes_read = spi.readbytes(3)
    GPIO.output(TFS, GPIO.HIGH)

    bytes_int = int((bytes_read[2] + 256*bytes_read[1] +
65536*bytes_read[0])/64)
    bytes_hex = hex(bytes_int)

    voltage = 3.815e-5 * bytes_int
    if voltage > 5: voltage = voltage - 10

    global df_temp

    df_temp = pd.DataFrame({'aeg': [dt.datetime.now()],
                            'hex': [bytes_hex],
                            'int': [bytes_int],
                            'pinge': [voltage]})
    df_temp = df_temp.set_index('aeg')

    global df_meta

    try:
        df_meta = df.drop(['hex', 'int'], axis=1).resample('5S').mean()
        df_meta = df_meta.rename(columns={'pinge': 'Vavg'})
        df_meta['Vrms'] = df_meta['Vavg'].mul(m.pi/(2*m.sqrt(2)))
    except:
        pass
```

Edasiarendatud süsteemi vahelduvvoolu pinge väärtuste täpseks mõõtmiseks defineeriti *threading* meetodi abil taimeri objekt *Counter*, mis käivitas signaali lugemise funktsiooni täpselt iga 1 ms

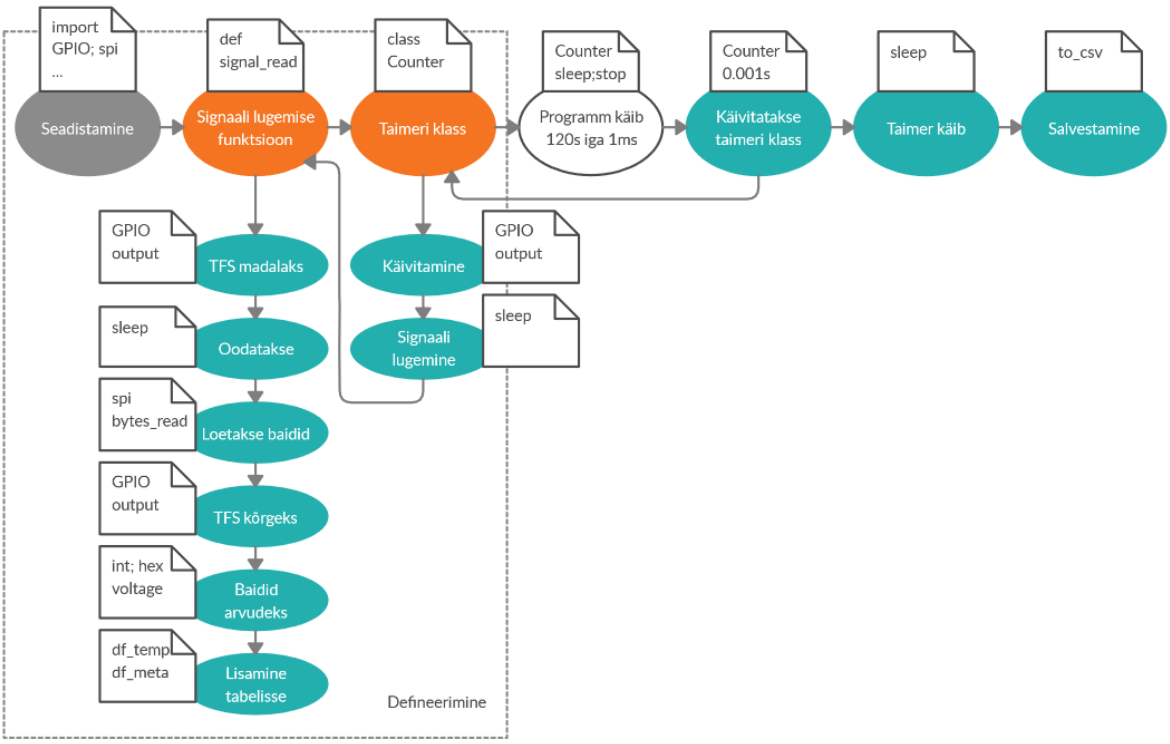
tagant. Selle objekti olulisus tulenes sellest, et *time* teegis olevat ootamise funktsiooni kasutades sõltus programmi efektiivsus liiga palju mikrokontrolleri operatsioonisüsteemist, kus selle programmi prioriteetsuse tase oli liiga madal, et 1 ms täpsusega funktsiooni käivitada [12].

```
class Counter():
    def __init__(self, increment):
        self.next_t = time.time()
        self.i=0
        self.done=False
        self.increment = increment
        self.run()

    def run(self):
        global df

        signal_read()

        if self.i > 1:
            df = df.append(df_temp)

        self.next_t+=self.increment
        self.i+=1

        if not self.done:
            threading.Timer(self.next_t - time.time(),
self.run).start()

    def stop(self):
        self.done=True

a=Counter(increment = 0.001)
time.sleep(120)
a.stop()
```

Peale 120 sekundi möödumist salvestati saadud reaalajalised väärtused ühte CSV faili ning teise iga 5 sekundi tagused pinge keskmised ja RMS väärtused.

```
df.to_csv(r'/home/pi/Desktop/voltagedata/rawdata.csv', index=True)
df_meta.to_csv(r'/home/pi/Desktop/voltagedata/metadata.csv',
index=True)
```

Programmi algoritmi alguses seadistati esialgu vajalikud teegid ning GPIO ja SPI seaded. Peale seadete sätestamist defineeriti signaali lugemise funktsioon ja taimeri objekt. Seejärel käivitati taimeri objekt 120 sekundiks, mis käivitas signaali lugemise funktsiooni iga 1 ms tagant. Peale taimeri tööperioodi lõppu salvestati andmed CSV faili (Joonis 14).



Joonis 14. Edasiarendatud süsteemi programmi algoritm.

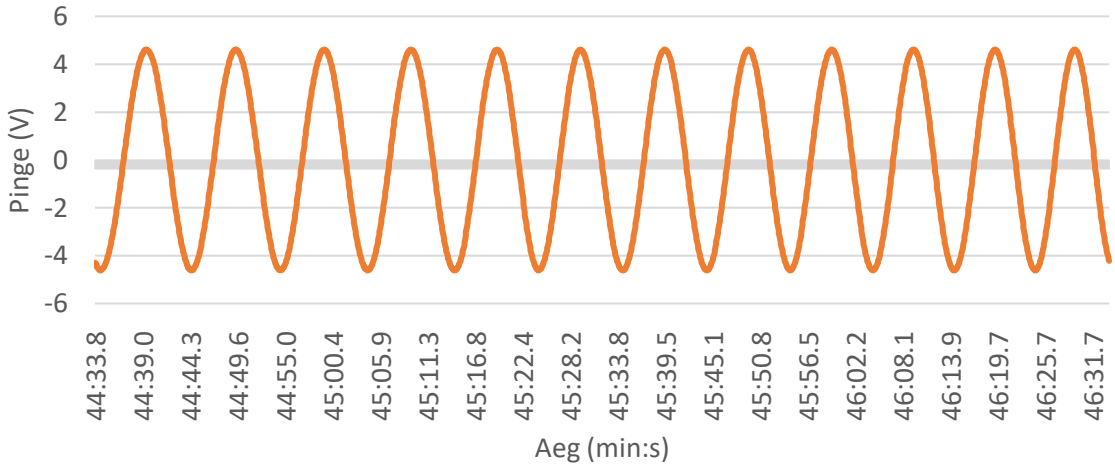
1.3.3 Tulemused

Edasiarendatud süsteemi elektriskeemi ja programmi eesmärgi saavutamiseks pidi katsesüsteemist erinevalt looma spetsiaalse taimeri objekti, mis võimaldaks soovitud täpsusega andmeid lugeda ja salvestada. Katsetamise käigus kasutati andmete õigsuses veendumiseks ostsilloskoopi, et kontrollida RPI ja AD7691 andmevahetust. Salvestatud CSV failidest on lisatud lõiked tabelisse Tabel 2 ning terve signaali reaalajalised andmed on nähtavad graafikul Joonis 15.

Aeg (min:s)	Pinge (HEX)	Pinge (INT)	Pinge (V)
44:33.8	0x24869	149609	-4.29242
44:33.9	0x24459	148569	-4.33209
44:34.0	0x24066	147558	-4.37066
44:34.0	0x237ce	145358	-4.45459
44:34.1	0x233b3	144307	-4.49469
44:34.1	0x230c2	143554	-4.52341
44:34.2	0x22dd3	142803	-4.55207
44:34.2	0x22bea	142314	-4.57072
44:34.3	0x22a32	141874	-4.58751
44:34.3	0x228b7	141495	-4.60197

Aeg (min:s)	Pinge keskmi. (V)	Pinge RMS (V)
44:30.0	-4.49783	-4.99583
44:35.0	1.055934	1.172848
44:40.0	-1.05137	-1.16778
44:45.0	1.027372	1.141124
44:50.0	-0.9827	-1.09151
44:55.0	1.012611	1.124728
45:00.0	-1.09432	-1.21548
45:05.0	1.03409	1.148585
45:10.0	-1.03812	-1.15306
45:15.0	1.047092	1.163027

Tabel 2. Reaalajalised tulemused ja 5-sekundilise intervalliga tulemused kombineeritud ühisesse tabelisse.



Joonis 15. Edasiarendatud süsteemi salvestatud tulemuste pinge muutumine ajas.

1.4 Järeldus

Eeldatud elektriskeemi ja programmi tegemise aeg võttis rohkem aega kui arvestatud oli, kuid saadud tulemused vastasid järgneva andmetöötluse etapi nõuetele ning on korrektsed. Analoog-digitaalmuundamise teoreetilisi tõdemusi arvestades toimisid mõlema süsteemi ADC-d vastavalt vajadustele. SPI protokolli kasutamine sobis, sest nähtavat müra signaali tulemustega kaasa ei tulnud ning programm jõudis andmed salvestada piisavalt kiirel kiirusel.

2. ANDMETÖÖTLUS JA -ANALÜÜS

Signaali töötlemise eesmärgiks pandi uurida esialgu erinevaid meetodeid, mis võimaldaksid suurte andmehulkade korral kiirelt ja efektiivselt leida anomaaliaid ja signaali vajadusel korrastada.

Kaks peamist meetodit, millega analüüsi protseduure läbi viidi, olid *wavelet* funktsioonid ja Fourier' teisendused. Kõige optimaalsema tööriistana võeti kasutusele Matlabi tarkvara, sest see toetab mõlema meetodi kasutamist ja omab head dokumentatsiooni signaalitöötluse valdkonnas.

2.1 Muutuva signaali genereerimine

Elektroonika süsteemi loomisel kasutati vahelduvvoolu mõõtmiseks siinussignaali, et oleks kerge elektroonikasüsteemis ja programmis vigu leida. Reaalses elus pole siiski signaal alati nii puhas – leidub müra, sagedus ei pruugi olla konstantne ning pinge väärtused kõiguvad. Seetõttu otsustati genereerida siinussignaal kahekordse võrgusagedusega ning lisada sellele modulatsioon, mis muudaks signaali ebaühtlasemaks.

2.1.1 Muutused programmi algoritmis

Esialgne elektroonika süsteem ja programm testiti vahelduvvoolu signaaliga, mille sagedus oli 0,1 Hz. Kasutades kõrgemat sagedust, tekkis programmil raskusi andmete piisavalt kiire lugemisega ehk mõõtetulemuste mõõtmise sagedus ei püsinud piisavalt kõrgel, et saada piisavalt tulemusi terviku signaali lugemiseks. Seega tuli programmi algoritmi muuta ning eemaldada osad protsessid, mis osutusid ajamahukaks.

Signaali lugemise funktsioonist eemaldati *Pandas* teegi andmete tabelite kasutamine ja asendati need listide kasutamisega, sest listidesse andmete lisamine võtab vähem aega. Pandas on efektiivne suurandmete töötlemise tööriist, kuid antud reaalajale keskenduval süsteemil kasutas see liiga palju ajaressurssi.

```
datelist = []
voltlist = []

def signal_read():
    GPIO.output(TFS, GPIO.LOW)
    time.sleep(5e-6)
    bytes_read = spi.readbytes(3)
    GPIO.output(TFS, GPIO.HIGH)

    bytes_int = int((bytes_read[2] + 256*bytes_read[1] +
65536*bytes_read[0])/64)
```

```
    voltage = 3.815e-5 * bytes_int
    if voltage > 5: voltage = voltage - 10

    return voltage
```

Samuti tuli teha korrektuure taimeri objektis, kus funktsiooni *run* sisust eemaldati väärtuste andmete tabelisse lisamine. Eemaldati ka täielikult *datetime* teegi kasutamine, sest selle teegi ajalised tulemused varieerusid taimeris kasutatud *time* teegi omadest mitme millisekundi võrra.

```
def run(self):
    global datelist
    global voltlist

    voltlist.append(signal_read())
    datelist.append(self.next_t)

    self.next_t+=self.increment
    self.i+=1

    if not self.done:
        threading.Timer( self.next_t - time.time(), self.run).start()
```

Saadud listid ja ajalised tulemused kanti alles peale ajanõudlikke protsesse andmete tabelisse, et seda oleks programmi lõpus kerge salvestada CSV faili.

```
df = pd.DataFrame(list(zip(datelist, voltlist)), columns=['aeg',
'pinge'])
```

2.1.2 Signaali karakteristikud

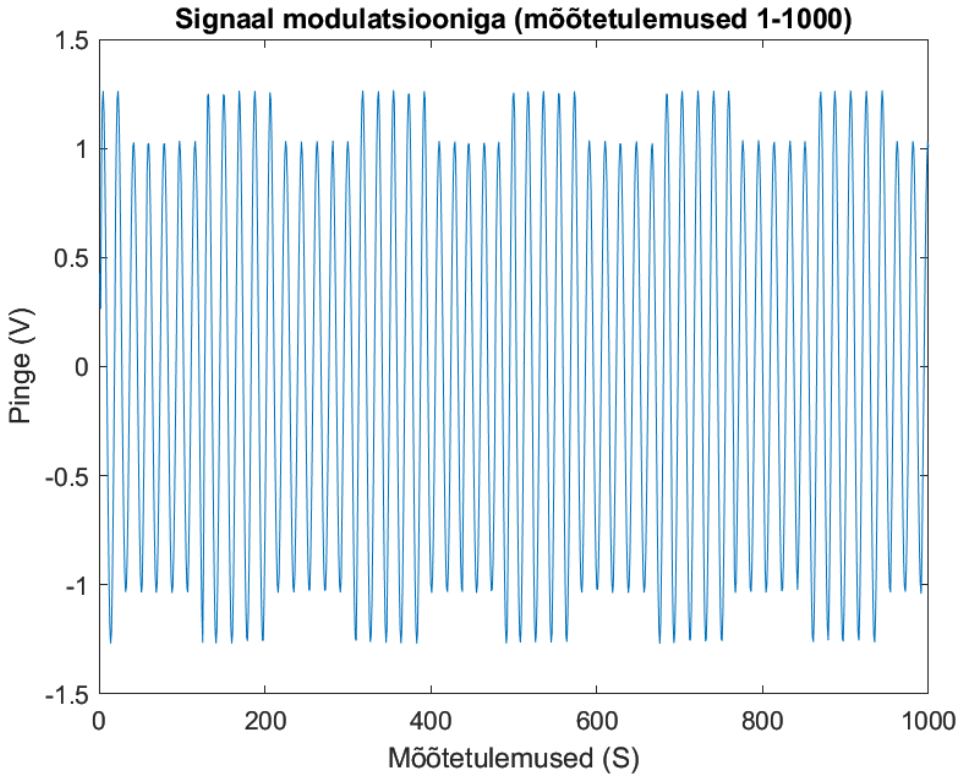
Analüüsi jaoks kasutati siinussignaali, millele oli lisatud ka modulatsioon. Peasagedus signaalil oli 100 Hz ning pinge amplituud 5 V *peak-to-peak* (-2,5 V kuni +2,5 V). Modulatsiooniks kasutati AM-i (ik *amplitude modulation*), mille sagedus oli 10 Hz, amplituudi sügavus 10 % originaalsest signaalist ning kuju ristkülikukujuline [13].

Selline kõrge sagedus koos AM modulatsiooniga sai valitud, et olla kindel, kas süsteem saaks hakkama ka tüüpilise elektrivõrgu (50-60 Hz) väärtuste mõõtmise, töötlemise ja analüütikaga. Korrigeeritud programmikoodiga mõõdeti moduleeritud signaali väärtused ja saadi kokku ligi 30 000 pinge väärtust 30 sekundi vahemikus keskmise diskreetimisperioodiga (ik *sample rate*) 0,0005 sekundit.

Tabel 3 kuvab kümmet esimest loetud signaali pinge väärtust lõigatuna välja CSV failist ja Joonis 16 esimest 1000 signaali pinge mõõtmistulemust graafikule kujutatuna.

Aeg (s)	Pinge (V)
27.1005	0.426517
27.1010	0.264723
27.1015	0.923192
27.1020	1.198902
27.1025	1.263147
27.1030	1.16491
27.1035	0.927236
27.1040	0.587167
27.1045	0.193001
27.1050	-0.24321

Tabel 3. AM modulatsiooniga vahelduvvoolu signaali CSV faili lõige.



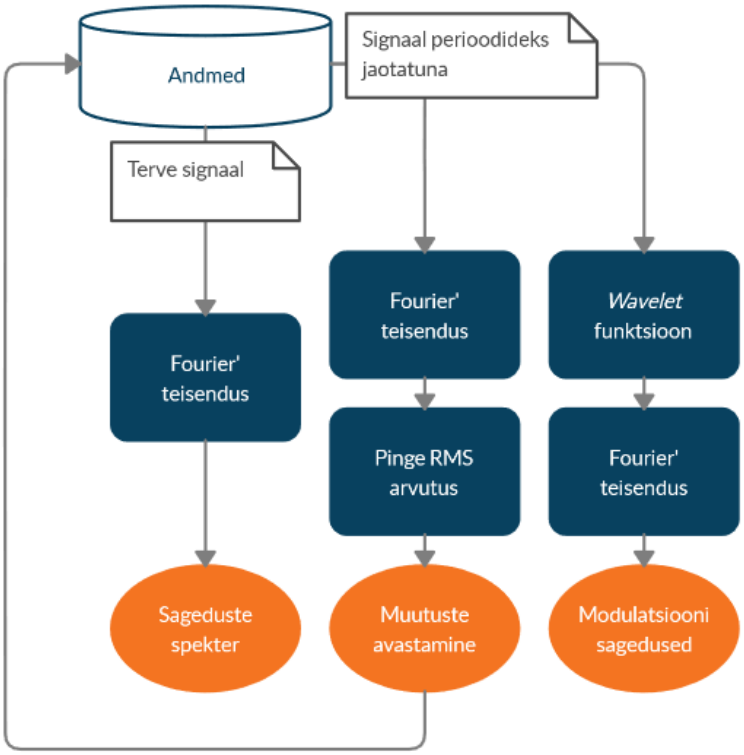
Joonis 16. AM modulatsiooniga vahelduvvoolu signaal.

2.2 Analüüsi meetodid ja protsessi tutvustus

Antud uurimuse käigus kasutati signaali analüüsiks *wavelet* funktsioone ja Fourier' teisendusi. Mõlema peamised kasutusalad on just signaalide töötlemine ja analüüs – näiteks maavärinate magnituudide mõõtmine ja helide mürast puhastamine.

Wavelet funktsioonide ja Fourier' teisenduste kasutamiseks tuli installeerida Matlabis kaks lisapakki – *Wavelet Toolbox* ja *Signal Processing Toolbox*. Meetodeid kasutati kolmeks erinevaks analüüsi

protseduuriks: terve signaali sageduste spektri leidmiseks, signaali teatud perioodide lõikes toimunud muutuste avastamiseks ja perioodi modulatsioonsignaali sageduste leidmiseks (Joonis 17).



Joonis 17. Analüüsi meetodite protsesside kirjeldus.

2.2.1 Fourier' teisendus

Fourier' teisendus (ik *Fourier transform*) on kõige levinum meetod signaalitöötluses ning selle implementeerimiseks Matlabis on vaja kasutada ainult ühte käsku.

```
fft_values = fft(signal_array);
```

Selles töös kasutatakse Fourier' teisenduse alaliiki FFT (ik *Fast Fourier Transform*), mis on arvutuslikult kiirem variant diskreetsest Fourier' teisendusest. Matemaatiliselt terve Fourier' teisenduse definitsioon selle uurimuse huviorbiidile ei satu, kuid Võrrand 5 kuvab FFT definitsiooni valemina.

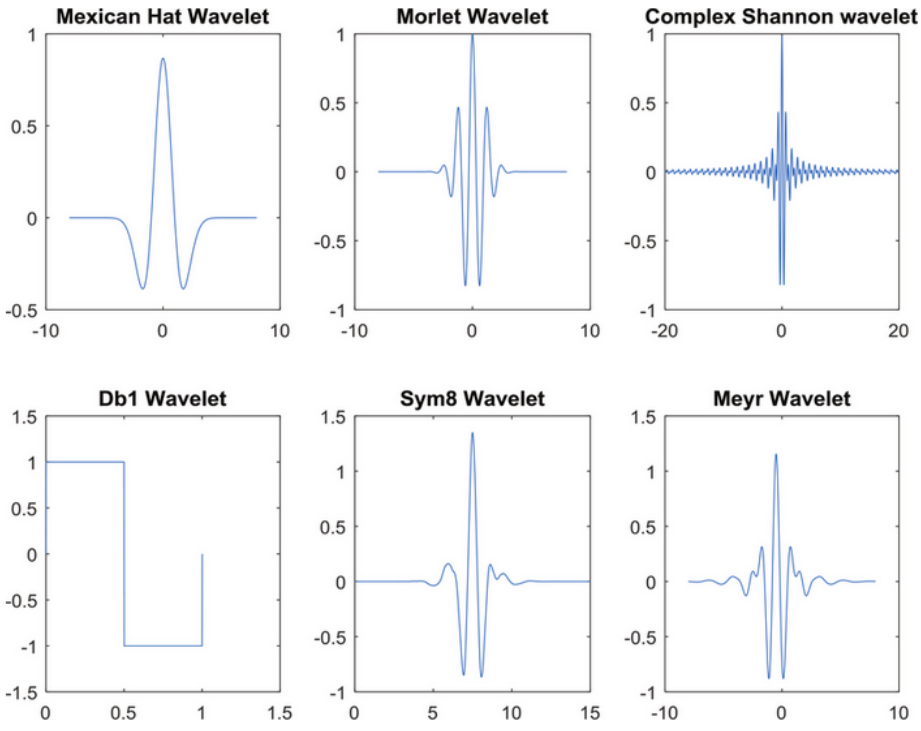
$$X_k = \sum_{n=0}^{N-1} x_n e^{-2j\pi kn/N} \ , kus\ k = 0, \dots, N-1$$

Võrrand 5. FFT definitsioon [14].

Üks peamisi kasutusi sellel teisendusel on signaalide sageduste spektri leidmine. Näiteks kui muusika lindistuses on tahtmata sattunud sisse mõni kõrge sagedusega heli signaal, siis kasutades Fourier' teisendust, on võimalik luua sageduste spekter, mis kuvab graafikuna erinevad sagedused, mis selles lindistuses esinevad, ning nende magnituudi ehk signaali amplituudi kordajad. Sel juhul on graafikult näha, et kui teatud kõrge sageduse juures amplituudi kordaja väärtus on suur, siis see kõrge sagedus põhjustab sobimatut heli.

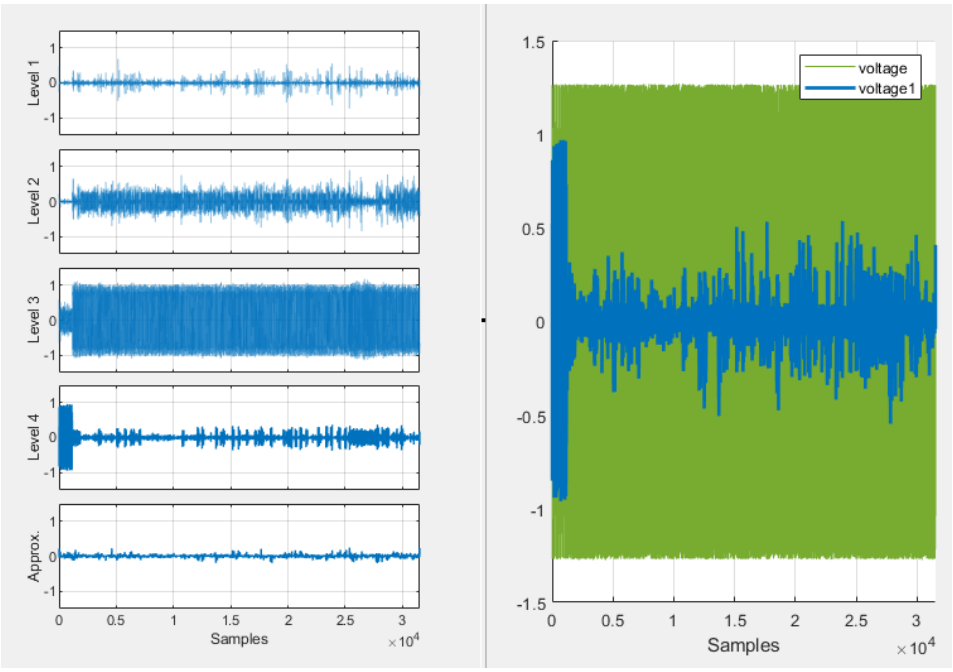
2.2.2 Lainekeste ehk *wavelet* funktsioonid

Wavelet funktsioon on signaalitöötlusmeetod, milline võimaldab leida erinevaid sarnasusi teatud etteantud kujudega. Fourier' teisendus ei anna oma rakendamisel aja piirkondasid, millal teatud sagedused esinevad, kuid *wavelet*i puhul kasutatakse kindlaid lained (Joonis 18), millega filtreeritakse terve signaal ja vastavalt leitakse nii sageduse väärtus kui ka aja piirkond kui mil sageduskomponent oli teatud suurusega [15].



Joonis 18. *Wavelet* lained [16].

Matlabis on selle integreerimiseks lihtsaim viis kasutada *Signal Multiresolution Analyzer* rakendust (Joonis 19), mille abil saab valida sobiva *wavelet*i laine ning mitmeks erinevaks sagedushulgaks signaali demonteerida soovitakse. Rakenduses on visuaalselt toodud välja erinevad alamsignaalid koos sageduspiirkondade ja nende osakaaluga tervest signaalist [17].



Joonis 19. *Wavelet* funktsiooni kasutamine *Signal Multiresolution Analyzer* tööriistaga.

2.2.3 RMS pinge arvutamine

Pinge RMS väärtuse vajalikkus esineb selles, et vahelduvvool pole koguaeg konstantne nagu alalisvool ning see näitab vahelduvvoolu umbkaudse vastavuse alalisvooluga. RMS leitakse pinge ruutkeskmisest (Võrrand 6).

$$U_{RMS} = \sqrt{\frac{U_1^2 + U_2^2 + \ldots + U_n^2}{N}}$$

Võrrand 6. Pinge RMS leidmise valem [18].

2.3 Meetodite rakendamine

Antud töös kasutatakse Fourier' teisenduse alaliiki FFT, mille peamiseks ülesandeks on jaotada terve signaal seda sisaldavateks sagedusteks. Peale selle rakendatakse FFT-d koos *wavelet*

funktsiooniga, mille eesmärk on leida perioodiliti esinevad modulatsiooni sagedused ehk kõik sagedused, mis ei ole peasignaali sagedus (100 Hz). Viimaks kasutatakse FFT-d veel koos RMS pinge arvutamisega, kus RMS pinge väärtust ja FFT tulemust võrreldakse, et leida signaalis toimuvaid muutuseid.

2.3.1 Andmete demonteerimine

Matlabi keskkonnas töötamiseks tuli esialgu lugeda sisse CSV fail ning valida sealt ainult analüütikuks vajalikud read. Antud analüüsi meetodite kasutamiseks läks vaja pinge ja aja andmeid.

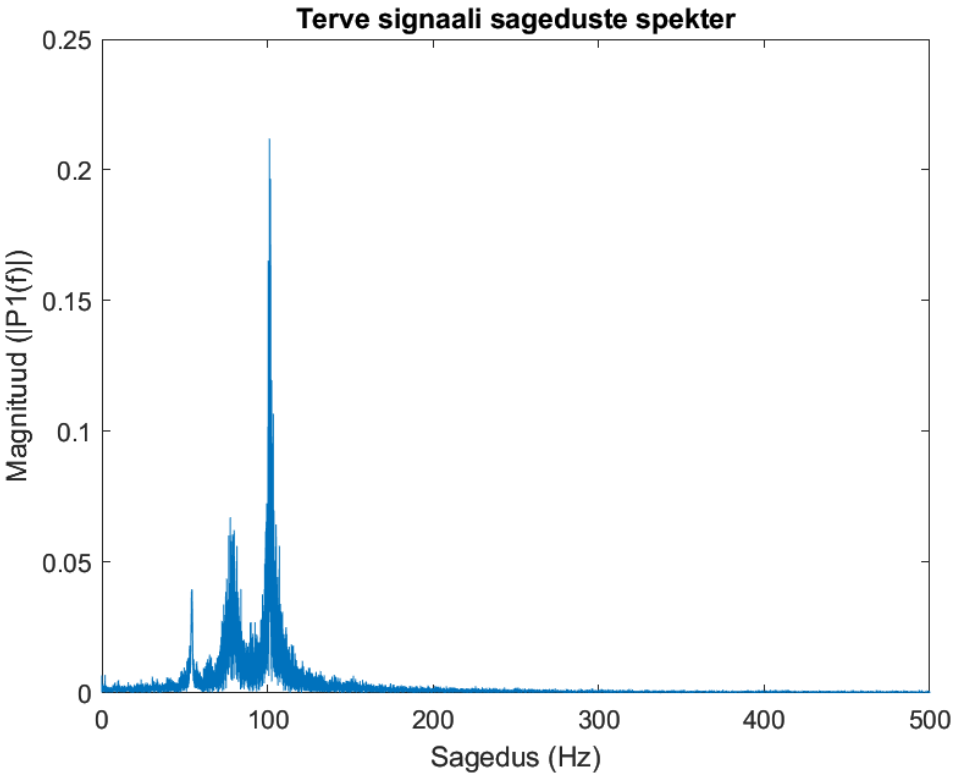
```
rawdata = readtable('data/rawdata.csv', "PreserveVariableNames",false);

voltage = rawdata(:, 3);
voltage = voltage{:, :};
time = rawdata(:, 2);
time = time{:, :};
```

2.3.2 Terve signaali sageduste spekter

Üks eesmärkidest oli leida, millised sagedused esinevad analüüsitavas signaalis. Selle jaoks tuli demonteeritud andmeid kasutada FFT funktsioonis. Esiteks defineeriti ära, mis ajaperioodi tagant andmed sisse loetud on (*sampling period*, T), mis on sagedus (*sampling frequency*, Fs), signaali pikkus (L), ajavektor (t) ja sageduste piirkond (f). Seejärel rakendati pinge väärtustele FFT teisendus ning Nyquisti teoreemi järgi eemaldati pooled väärtused, sest meie teisendusest tulenes topelt arv väärtuseid, millest pooled olid lihtsalt originaalväärtuste peegeldused [19].

```
T = 0.0005;
Fs = 1/T;
L = length(voltage);
t = (0:L-1)*T;
f = Fs*(0:(L/2))/L;

fft_volt = fft(voltage);
P2 = abs(fft_volt)/L;
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
```

Saadud tulemus on visualiseeritud graafikul Joonis 20. Näha on, et peamiselt domineerib sagedus 100 Hz ning modulatsiooni tõttu leidub signaalis ka muid sagedusi.

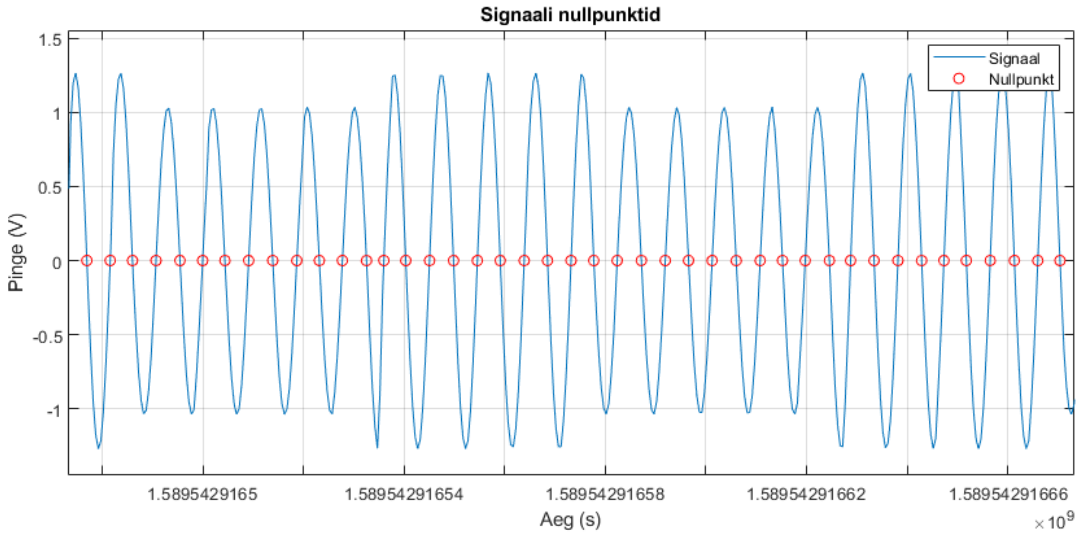


Joonis 20. Terve signaali sageduste spekter.

2.3.3 Signaali perioodideks jaotamine

Terve signaali analüüsil oli näha kogu sageduste esinemise tulemust. See kahjuks aga ei ütle meile peaaegu et midagi. Seejaoks jaotati terve signaal perioodideks. Kasutades *zero-crossing* meetodit, leiti signaali kõik punktid, mis on lähimad nullile [20]. Nullpunktide vahele jäi keskmiselt 20 mõõdetud pinge väärtust ehk järeldati, et ühe perioodi pikkus on 20 pinge väärtust (Joonis 21). Nullpunktide leidmise skripti loomisel kasutati internetist leitud algoritmi [21].

```
UpZCi = @(v) find(v(1:end-1) <= 0 & v(2:end) > 0);
DownZCi = @(v) find(v(1:end-1) >= 0 & v(2:end) < 0);
ZeroX = @(x0,y0,x1,y1) x0 - (y0.*(x0 - x1))./(y0 - y1);

ZXi = sort([UpZCi(voltage'),DownZCi(voltage')]);
ZX = ZeroX(time(ZXi),voltage(ZXi),time(ZXi+1),voltage(ZXi+1));

if voltage(end)==0
    ZX(end+1) = t(end);
end
```



Joonis 21. Signaali nullpunktid (ajatelje ühik on UNIX süsteemi järgi arvutuslikel eesmärkidel).

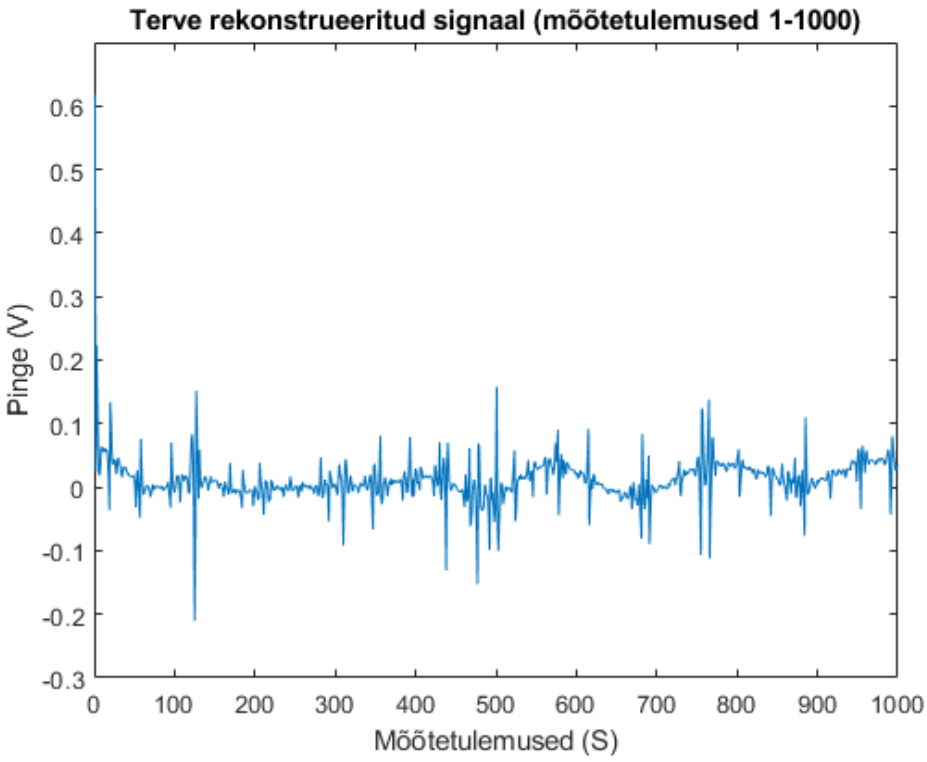
2.3.4 Müra sageduste uurimine

Mürasignaali leidmiseks tuli esiteks eemaldada peamine signaali kordussagedus – 100 Hz. Selle saavutamiseks kasutati *wavelet* funktsiooni Matlabi rakenduses *Signal Multiresolution Analyzer*. Signaalile rakendati *symlet* laine, mis on tuntud kasutamiseks perioodiliste signaalide puhul [22]. Signaal jaotati 14-ks erineva ülesehitusega signaaliks ja eemaldati neist neli suurema osakaaluga signaali, mis olid 100 Hz või selle sageduse lähedal. Eemaldatud signaale koos oma karakteristikutega on näha tabelist Tabel 4.

	Frequencies (cycles/sample)	Relative Energy	Include	Show
Level 1	0.25 - 0.5	0.43%	☑	☑
Level 2	0.121 - 0.259	17.33%	☑	☑
Level 3	0.0603 - 0.129	75.92%	☐	☑
Level 4	0.0302 - 0.0646	5.95%	☐	☑
Level 5	0.0151 - 0.0323	0.24%	☐	☑
Level 6	0.00754 - 0.0162	0.07%	☐	☑
Level 7	0.00377 - 0.00...	0.02%	☑	☑
Level 8	0.00188 - 0.00...	0.01%	☑	☑
Level 9	0.000943 - 0.0...	0.01%	☑	☑
Level 10	0.000471 - 0.0...	0.00%	☑	☑
Level 11	0.000237 - 0.0	0.00%	☑	☑

Tabel 4. *Signal Multiresolution Analyzer*i tulemusel saadud alamsignaalid.

Eemaldatud signaalide tulemusel saadi esialgsest signaalist töödeldud versioon ehk rekonstrueeritud signaal, mida on näha graafikul Joonis 22. Antud signaal on peasignaalis esinev müra.

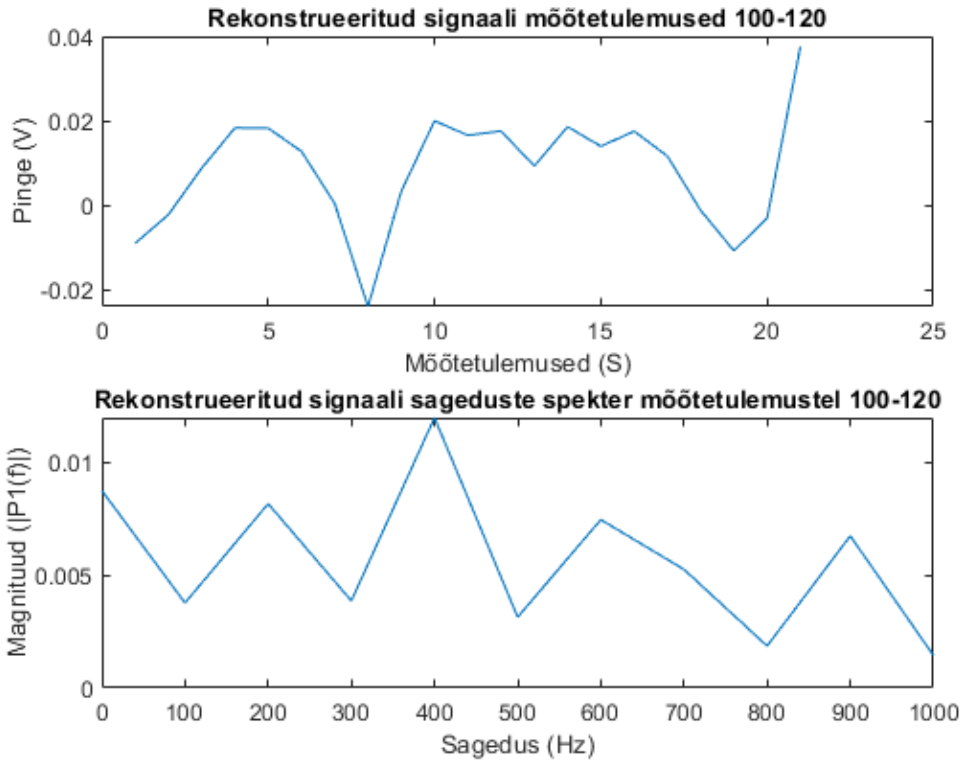


Joonis 22. Rekonstrueeritud mürasignaal.

Kogu rekonstrueeritud signaal jaotati perioodideks, leiti iga perioodi korral selle FFT väärtus ja saadi selle perioodi sageduste spekter. Sellega saavutati hea ülevaade erinevate signaali perioodide jooksul esinevatest sagedustest, mis ei kuulunud peasageduse hulka. FFT rakendamisel tuli seekord signaaliks valida rekonstrueeritud signaali väärtused (*voltage1*) ning kasutati ainult esimest 50 perioodi signaalist.

```
q = 50;
m = 1;
n = 20;

for k = 1:q
    voltage_ind = voltage1(m:n);

    T_ind = 0.0005;
    Fs_ind = 1/T_ind;
    L_ind = 20;
    t_ind = (0:L_ind-1)*T_ind;
    f_ind = Fs_ind*(0:(L_ind/2))/L_ind;

    fft_volt_ind = fft(voltage_ind);
    P2_ind = abs(fft_volt_ind)/L_ind;
    P1_ind = P2_ind(1:L_ind/2+1);
    P1_ind(2:end-1) = 2*P1_ind(2:end-1);

    m=n;
    n=n + 20;
end
```

Peale FFT rakendamist igale perioodile on näha selgelt signaalis esinevaid anomaaliaid. Graafikul (Joonis 23) on näha ühe perioodi sageduste esinemist spektrina. Sel võetud perioodil on suuremad mürasignaali sageduse väärtused 200, 400, 600 ja 900 Hz.



Joonis 23. Rekonstrueeritud mürasignaali sageduste spekter ühel perioodil.

2.3.5 Muutuste ja trendide tuvastamine

Kõikide metoodikate tulemusena oli terve suurema projekti raames leida muutuseid ja signaali karakteristikute trende. Muutuste ja trendide tuvastamiseks leiti kõige efektiivsemaks viisiks võrrelda perioodide pinge keskmiste RMS väärtuseid ja Fourier' teisenduse spektrit.

Katsetamiseks kasutati perioode 2-51, et vältida esimest poolikut perioodi. Defineeriti kolm maatriksi, kuhu sisestati pinge RMS väärtused (*rms_volt*), perioodid (*periods*) ja FFT tulemused (*fft_total*). Tsükkel jäeti sarnaseks, kuid lisati juurde RMS väärtuste arvutamine ning perioodide ja FFT tulemuste lisamine maatriksitesse.

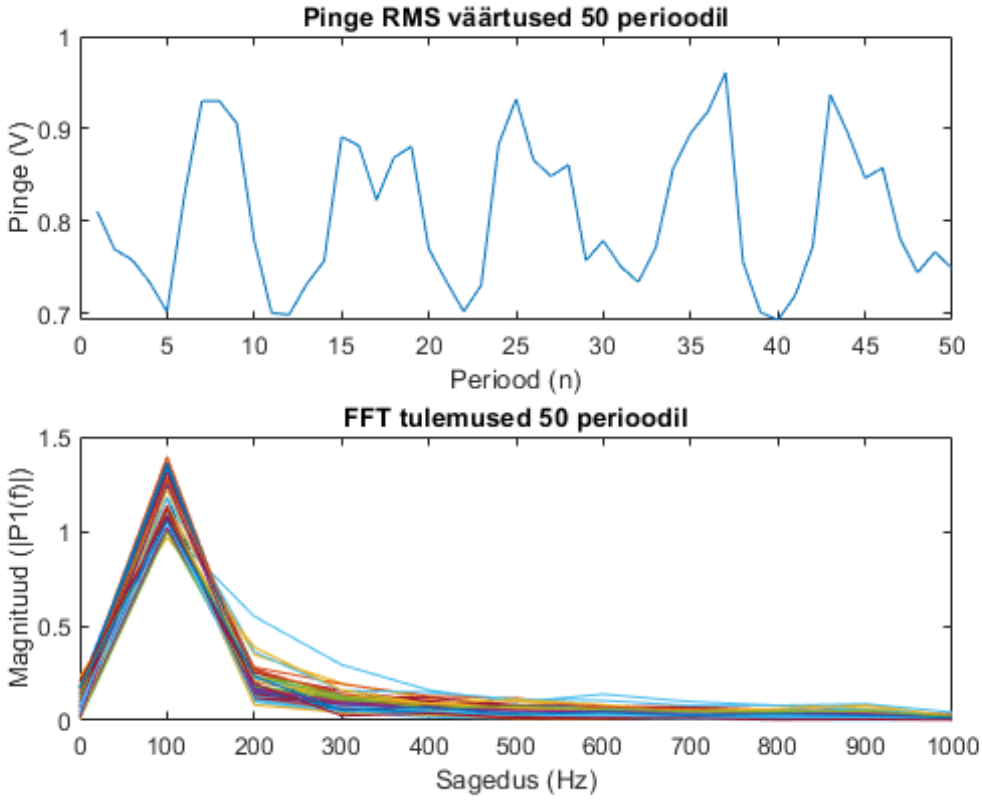
```
q = 50;
m = 20;
n = 40;

rms_volt = zeros(1, q);
periods = zeros(1, q);
fft_total = zeros(q, 21);
```

```
for k = 1:q
    voltage_ind = voltage(m:n);

    periods(k) = k;
    rms_volt_ind = sqrt((sum(voltage_ind.^2))/length(voltage_ind));
    rms_volt(k) = rms_volt_ind;

    T_ind = 0.0005;
    Fs_ind = 1/T_ind;
    L_ind = 20;
    t_ind = (0:L_ind-1)*T_ind;
    f_ind = Fs_ind*(0:(L_ind/2))/L_ind;

    fft_volt_ind = fft(voltage_ind);
    P2_ind = abs(fft_volt_ind)/L_ind;
    P1_ind = P2_ind(1:L_ind/2+1);
    P1_ind(2:end-1) = 2*P1_ind(2:end-1);
    fft_total(k, :) = P1_ind';

    m=n;
    n=n + 20;
end
```

Peale tsükli töökäigu lõppu lisatakse saadud väärtused kahte maatriksisse ja salvestatakse need CSV faili. Sel juhul on võimalik efektiivselt perioodilõikude pinge RMS ja FFT tulemustele ning sageduse andmetele ligi pääseda ka hiljem.

```
maindata = zeros(q, 23);
maindata(1:q, 1) = periods';
maindata(1:q, 2) = rms_volt';
maindata(:, 3:13) = fft_total;

metadata = zeros(5, 11);
metadata(1, 1) = T_ind;
metadata(2, 1) = Fs_ind;
metadata(3, 1) = L_ind;
metadata(4, 1:11) = f_ind;

csvwrite("data\maindata.csv", maindata);
csvwrite("data\metadata.csv", metadata);
```

Nagu Joonis 24 kuvab, siis saab järeldada, et antud 50 perioodil oli signaali sagedus enamasti 100 Hz piires, kuid leidub ka sageduste väärtuseid 200, 300 ja 600 Hz juures. Erinevaid perioodide lõike FFT ja pinge RMS väärtuste võrdlemisel on võimalik saada aimdust, miks võib toimuda teatud muude sageduste esinemist signaalis. Näiteks võib pinge RMS väärtus olla kuskil järsult kukkunud ja vastavalt ka FFT spekter näitab sel juhul mingit teist sagedusväärtust.



Joonis 24. Pinge RMS väärtused ja FFT tulemused.

2.4 Signaali analüüsi järeldused

Terve signaali sageduste spektri ülevaatus on kasulik arusaam analüütikule, sest see annab hea ülevaate, mis sagedused esinevad terves signaalis. Praeguses kasutatavas signaalis teati, et peamine sagedus on 100 Hz, kuid kui signaali sagedus pole teada, siis saab Fourier' teisendusega teada signaali kõik sageduste piirkonnad.

Signaali perioodideks jaotamine on oluline, kui on vaja teada erinevate sageduste või vigade leidmine kindlates ajapiirkondades. Kui on signaali sattunud mõneks ajaks mõni müra põhjustav tegur, kuid soovitakse teada, millal müra esines, siis perioodideks jaotamine ning FFT rakendamine igale perioodile teeb selle protsessi võimalikuks.

Mürasignaali sageduste uurimise peamine eesmärk oli teada saada moonutuste sagedused ja mis ajahetkedel need toimusid. See tuleb eriti kasuks veel siis, kui on vaja signaal moonutustest puhastada. Sel juhul on võimalik viimaks veel *Inverse Fourier Transform* teisendusega luua uus peasignaal, millest on eemaldatud mittesobivad alamsignaalid.

Perioodide vältel pinge keskmiste RMS väärtus muutub ning seda on võimalik seostada FFT sageduste spektriga. Kui spekter kuvab põhisageduse asemel ka muid väärtuseid, siis seda saab seostada ka pinge RMS väärtusega, mis võivad seetõttu sisaldada anomaaliaid ja pinge väärtuse trendi muutust.

KOKKUVÕTE

Käesoleva töö eesmärgiks oli kujundada elektrooniline süsteem programmiga, mis võimaldaks reaalajaliselt salvestada perioodilisi sisendsuuruseid, ning uurida meetodeid signaali töötlemiseks ja analüüsiks.

Esimeses peatükis kirjeldatakse süsteemi arenduskäiku ja kirjeldati teoreetiliselt, kuidas füüsikalisi suuruseid muundatakse digitaalsele kujule kasutades analoog-digitaalmuundurit. Süsteemi ehitamist alustati katsesüsteemist, mille ülesanne oli alalisvoolu pinge ja voolu väärtuseid salvestada mikroarvutisse. Soovitud tulemuste saavutamisel arendati lõplik edasiarendatud süsteem, mis suudaks reaalajaliselt lugeda võrgusagedusega vahelduvvoolu pinge andmeid.

Teine peatükk seletab lahti vahelduvsignaali andmete töötluse ja analüüsi protsessid. Siinussignaalile lisati modulatsioon, järgnevalt uuriti signaali parameetrite tuvastamist ja töötlemist. Lisaks tutvustatakse teises peatükis signaalianalüüsi meetodeid (Fourier' teisendus ja *wavelet* funktsioonid) ning lõpuks rakendatakse need genereeritud signaalile.

Edasiarendatud elektroonika süsteem koosnes neljast peamisest komponendist – vahelduvvoolu toiteallikas, analoog-digitaalmuundur ja mikroarvuti. Süsteemi katsetamisel selgus, et Raspberry Pi mikroarvuti operatsioonisüsteem ja kirjutatud programm ei suutnud kõrgema sagedusega signaali täpselt lugeda. See probleem lahendati programmi algoritmi optimeerimisega. Süsteemi võimekuse analüüsil järeldati, et sarnasel süsteemil peaks olema reaalajaline operatsioonisüsteem või spetsiifiline draiver, mis lubaks andmete lugemise määrata prioriteetsemaks kui Pythoni keskkonnas töötaval makroprogrammil.

Andmete töötlemiseks valiti Matlabi keskkond, sest sellel on parem tugi ja dokumentatsioon signaali analüüsi metoodikatele. Analüüsi protsessi käiguks genereeriti vahelduvvool sagedusega 100 Hz, millele oli lisatud AM sagedusega modulatsioon. Esiteks andmed demonteeriti ehk suur andmehulk viidi analüüsi metoodikatele sobivale kujule. Teiseks leiti terve signaali sageduste spekter kasutades Fourier' teisendust. Kolmandaks uuriti perioodide kaupa signaalis esinevat müra kasutades kõigepealt *wavelet* funktsioone ja Fourier' teisendust. Viimaks arvutati perioodi pinge RMS väärtus ja võrreldi seda FFT tulemusega, mis võimaldab efektiivselt näha sageduskomponentide muutumist teatud perioodide vältel.

Autori ja instituudi poolt jätkatakse antud arendustööga. Lõputöö eesmärk uurida riistvaralist võimekust ja signaali töötlemise esmaseid metoodikaid on täidetud. Elektroonika süsteemi ja katsetatud analüüsi meetodeid kasutatakse tuleviku projektis, kus eesmärgiks on elektrivõrgus tuvastada sündmusi, nt uute seadmete ühendamist ahelasse.

SUMMARY

The main objective of this thesis was to build an electronic system with a functionality of reading and saving real-time alternating current signal data and research different methods for signal processing and analysis.

The first chapter describes the development of the electronic system by considering the theory of analog-digital converting. The first step of building that system was to create a testing platform which had to read and save direct current values to the microcontroller. After the completion of the first step, an advanced system was built. This platform was able to read real-time utility frequency alternating current signal and save the data to the microcontroller.

The second chapter explains the methods which were used for signal processing and analysis in this research. The frequency of the alternating current signal was set higher and a modulation was added for complexity purposes. The chapter introduces and then applies the most well-known signal processing methods – Fourier transform and wavelet function – for analysis.

The advanced electronic system included four main components – an alternating current power supply, analog-digital converter and a microcomputer. Firstly, during the data acquisition, it was noted that Raspberry Pi's operating system and the written program were too slow to read the inputted utility frequency alternating current signal. This problem was solved by making some adjustments in the program's algorithm. However, it was concluded that a real-time data acquisition system like this current platform should have a real-time operating system or a driver which could prioritize the written Python program script.

The chosen environment for data processing was Matlab because of its better support and documentation for signal analysis methods. Firstly, an alternating current was generated with a frequency of 100 Hz and an added AM modulation. The monitored signal data was then decomposed to a suitable format for the analysis. Secondly, the frequency spectrum of the whole signal was found using Fourier transform. Thirdly, the modulation of the signal was examined periodically with wavelet function and Fourier transform. Finally, signal's RMS voltage and FFT results were compared to find alterations in frequency during some period of time.

The mission of this study to test hardware capabilities and research different signal processing methods for utility frequency signal was successful. This thesis will be continued by the author and the institute to develop a system which automatically recognizes frequency changes and connections with new devices in the electrical circuit.

KASUTATUD KIRJANDUS

[1] Analog Devices, „Analog to Digital Conversion,“ 5 September 2013. [Võrgumaterjal]. Saadaval: https://wiki.analog.com/university/courses/electronics/text/chapter-20.

[2] B. P. Lathi, „Sampling And Pulse Code Modulation,“ *Modern Digital and Analog Communication Systems*, 1983, lk. 251-293.

[3] A. S. Nastase, „An ADC and DAC Least Significant Bit (LSB),“ 2010. [Võrgumaterjal]. Saadaval: https://masteringelectronicsdesign.com/an-adc-and-dac-least-significant-bit-lsb/.

[4] Raspberry Pi Trading, „Raspberry Pi 4 Model B Datasheet,“ Juuni 2019. [Võrgumaterjal]. Saadaval: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf.

[5] Maxim Integrated Products, „MAX471/MAX472 Datasheet,“ Detsember 1996. [Võrgumaterjal]. Saadaval: https://pdfserv.maximintegrated.com/en/ds/MAX471-MAX472.pdf.

[6] Texas Instruments, „ADS1115 Datasheet,“ Oktoober 2009. [Võrgumaterjal]. Saadaval: https://cdn-shop.adafruit.com/datasheets/ads1115.pdf.

[7] I2C Info, „I2C Info,“ 2020. [Võrgumaterjal]. Saadaval: https://i2c.info/.

[8] M. Burris, „Lifewire,“ 13 November 2019. [Võrgumaterjal]. Saadaval: https://www.lifewire.com/selecting-between-i2c-and-spi-819003.

[9] Analog Devices, „AD7691 Data Sheet,“ 2015. [Võrgumaterjal]. Saadaval: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7691.pdf.

[10] Texas Instruments, „LM317 3-Terminal Adjustable Regulator,“ Aprill 2020. [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/slvs044y/slvs044y.pdf?HQS=slvs044-aaj&ts=1589888297219.

[11] Texas Instruments, „LMx37 3-Terminal Adjustable Regulators,“ Jaanuar 2015. [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/slvs047l/slvs047l.pdf?&ts=1589889076594.

[12] Stack Overflow, „Python periodic timer interrupt,“ 2018. [Võrgumaterjal]. Saadaval: https://stackoverflow.com/questions/52722864/python-periodic-timer-interrupt.

[13] Tutorialspoint, „Amplitude Modulation,“ 2020. [Võrgumaterjal]. Saadaval: https://www.tutorialspoint.com/analog_communication/analog_communication_amplitude_modulation.htm.

[14] C. Maklin, „Fast Fourier Transform,“ 29 Detsember 2019. [Võrgumaterjal]. Saadaval: https://towardsdatascience.com/fast-fourier-transform-937926e591cb.

[15] M. Ryan, „What is Wavelet and How We Use It for Data Science,“ 31 Mai 2019. [Võrgumaterjal]. Saadaval: https://towardsdatascience.com/what-is-wavelet-and-how-we-use-it-for-data-science-d19427699cef.

[16] ResearchGate, „Wavelet Transform for Educational Network Data Traffic Analysis,“ 2020. [Võrgumaterjal]. Saadaval: https://www.researchgate.net/figure/Examples-of-different-types-of-wavelets_fig1_328086476.

[17] The MathWorks, „Signal Multiresolution Analyzer,“ 2020. [Võrgumaterjal]. Saadaval: https://se.mathworks.com/help/wavelet/ref/signalmultiresolutionanalyzer-app.html.

[18] R. Price, „How to Get the RMS in Excel,“ 10 Veebruar 2017. [Võrgumaterjal]. Saadaval: https://www.techwalla.com/articles/how-to-get-the-rms-in-excel.

[19] Electronics Stack Exchange, „Nyquist frequency mirroring,“ 2016. [Võrgumaterjal]. Saadaval: https://electronics.stackexchange.com/questions/202467/nyquist-frequency-mirroring.

[20] The MathWorks, „Zero-Crossing Detection,“ 2020. [Võrgumaterjal]. Saadaval: https://se.mathworks.com/help/simulink/ug/zero-crossing-detection.html.

[21] N. Hunter, „MATLAB Central,“ 12 Aprill 2020. [Võrgumaterjal]. Saadaval: https://se.mathworks.com/matlabcentral/answers/267222-easy-way-of-finding-zero-crossing-of-a-function.

[22] R. Y. Robert X Gao, „Symlet Wavelet,“ *Wavelets: Theory and Applications for Manufacturing*, 2011, lk. 63.

LISAD

```
import board
import busio
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn
import time
import pandas as pd
import datetime as dt

i2c = busio.I2C(board.SCL, board.SDA)
ads = ADS.ADS1115(i2c)
starttime = time.time()
df = pd.DataFrame()

while True:
    current = AnalogIn(ads, ADS.P1).voltage
    voltage = AnalogIn(ads, ADS.P2).voltage*5

    df_temp = pd.DataFrame({'aeg': [dt.datetime.now()], 'pinge':
[voltage], 'vool': [current]})
    df = df.append(df_temp)

    df.to_csv(r'/home/pi/Desktop/voltagedata/sensordata.csv',
index=False)

    time.sleep(0.1000 - ((time.time() - starttime) % 0.1000))
```

```
import RPi.GPIO as GPIO
import time, threading
import spidev
import pandas as pd
import datetime as dt
import math as m
import matplotlib.pyplot as plt

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
TFS = 12
DRO = 23
SCLK = 24
GPIO.setup(TFS, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(DRO, GPIO.IN)
GPIO.setup(SCLK, GPIO.OUT)
spi = spidev.SpiDev()
spi.open(0, 1)
spi.max_speed_hz = 10000

df = pd.DataFrame()
df_temp = pd.DataFrame()
df_meta = pd.DataFrame()

def signal_read():
    GPIO.output(TFS, GPIO.LOW)
    time.sleep(5e-6)
    bytes_read = spi.readbytes(3)
    GPIO.output(TFS, GPIO.HIGH)

    bytes_int = int((bytes_read[2] + 256*bytes_read[1] +
65536*bytes_read[0])/64)
    bytes_hex = hex(bytes_int)

    voltage = 3.815e-5 * bytes_int
    if voltage > 5: voltage = voltage - 10

    global df_temp

    df_temp = pd.DataFrame({'time': [dt.datetime.now()],
                            'hex': [bytes_hex],
                            'int': [bytes_int],
                            'voltage': [voltage]})
    df_temp = df_temp.set_index('time')

    global df_meta

    try:
        df_meta = df.drop(['hex', 'int'], axis=1).resample('5S').mean()
        df_meta = df_meta.rename(columns={'voltage': 'Vavg'})
        df_meta['Vrms'] = df_meta['Vavg'].mul(m.pi/(2*m.sqrt(2)))
    except:
        pass

class Counter():
    def __init__(self, increment):
        self.next_t = time.time()
        self.i=0
        self.done=False
```

```
        self.increment = increment
        self.run()

    def run(self):
        global df

        signal_read()

        if self.i > 1:
            df = df.append(df_temp)

        self.next_t+=self.increment
        self.i+=1

        if not self.done:
            threading.Timer( self.next_t - time.time(),
self.run).start()

    def stop(self):
        self.done=True

a=Counter(increment = 0.001)
time.sleep(120)
a.stop()

df.to_csv(r'/home/pi/Desktop/voltagedata/rawdata.csv', index=True)
df_meta.to_csv(r'/home/pi/Desktop/voltagedata/metadata.csv',
index=True)
```

Lisa 3 Edasiarendatud süsteemi parandustega programmikood

```
import RPi.GPIO as GPIO
import time, threading
import spidev
import pandas as pd
import datetime as dt
import math as m
import matplotlib.pyplot as plt

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
TFS = 12
DRO = 23
SCLK = 24
GPIO.setup(TFS, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(DRO, GPIO.IN)
GPIO.setup(SCLK, GPIO.OUT)
spi = spidev.SpiDev()
spi.open(0, 1)
spi.max_speed_hz = 100000

datelist = []
voltlist = []

def signal_read():
    GPIO.output(TFS, GPIO.LOW)
    time.sleep(5e-6)
    bytes_read = spi.readbytes(3)
    GPIO.output(TFS, GPIO.HIGH)

    bytes_int = int((bytes_read[2] + 256*bytes_read[1] +
65536*bytes_read[0])/64)

    voltage = 3.815e-5 * bytes_int
    if voltage > 5: voltage = voltage - 10

    return voltage

class Counter():
    def __init__(self, increment):
        self.next_t = time.time()
        self.i=0
        self.done=False
        self.increment = increment
        self.run()

    def run(self):
        global datelist
        global voltlist

        voltlist.append(signal_read())
        datelist.append(self.next_t)

        self.next_t+=self.increment
        self.i+=1

        if not self.done:
            threading.Timer( self.next_t - time.time(),
self.run).start()
```

```
    def stop(self):
        self.done=True

a=Counter(increment = 0.0005)
time.sleep(10)
a.stop()

df = pd.DataFrame(list(zip(datelist, voltlist)), columns=['time',
'voltage'])
df.to_csv(r'/home/pi/Desktop/voltagedata/rawdata.csv', index=True)
```

```
% Demonteeri CSV fail Matlabile sobivateks maatriksiteks

clear;
close all;

rawdata = readtable('data/rawdata.csv', "PreserveVariableNames",false);

voltage = rawdata(:, 3);
voltage = voltage{:, :};
time = rawdata(:, 2);
time = time{:, :};

figure;
plot(voltage)
title('Signaal modulatsiooniga')
xlabel('Mõõtetulemused (S)')
ylabel('Pinge (V)')

figure;
plot(voltage(1:1000))
title('Signaal modulatsiooniga (mõõtetulemused 1-1000)')
xlabel('Mõõtetulemused (S)')
ylabel('Pinge (V)')

warning('off','all')

% Leia terve signaali sageduste spekter kasutades FFT teisendust

T = 0.001;
Fs = 1/T;
L = length(voltage);
t = (0:L-1)*T;
f = Fs*(0:(L/2))/L;

fft_volt = fft(voltage);
P2 = abs(fft_volt)/L;
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

figure;
plot(f,P1)
title('Terve signaali sageduste spekter')
xlabel('Sagedus (Hz)')
ylabel('Magnituud (|P1(f)|)')

% Leia signaali nullpunktid kasutades "zero-crossing" meetodit

UpZCi = @(v) find(v(1:end-1) <= 0 & v(2:end) > 0);
DownZCi = @(v) find(v(1:end-1) >= 0 & v(2:end) < 0);
ZeroX = @(x0,y0,x1,y1) x0 - (y0.*(x0 - x1))./(y0 - y1);

ZXi = sort([UpZCi(voltage'),DownZCi(voltage')]);
ZX = ZeroX(time(ZXi),voltage(ZXi),time(ZXi+1),voltage(ZXi+1));

if voltage(end)==0
    ZX(end+1) = t(end);
end

figure;
```

```
plot(time, voltage)
hold on;
plot(ZX,zeros(1,length(ZX)),'ro')
grid on;
legend('Signaal', 'Nullpunkt')
title('Signaali nullpunktid')
xlabel('Aeg (s)')
ylabel('Pinge (V)')

% Kasuta "wavelet" funktsiooni, et eemaldada peasagedus (100 Hz) ja
jäta alles vaid muud sagedused

levelForReconstruction = [true, true, false, false, false, false, true,
true, true, true, true, true, true, true, true];
wt = modwt(voltage, 'sym4', 14);
mra = modwtmra(wt, 'sym4');
voltage1 = sum(mra(levelForReconstruction,:),1);

figure;
plot (voltage1(1:1000))
title("Terve rekonstrueeritud signaal (mõõtetulemused 1-1000)")
ylabel("Pinge (V)")
xlabel("Mõõtetulemused (S)")

% Leia modulatsiooni signaali sageduste spekter 10 perioodi kohta

q = 10;
m = 1;
n = 20;

for k = 1:q
    voltage_ind = voltage1(m:n);

    T_ind = 0.0005;
    Fs_ind = 1/T_ind;
    L_ind = 20;
    t_ind = (0:L_ind-1)*T_ind;
    f_ind = Fs_ind*(0:(L_ind/2))/L_ind;

    fft_volt_ind = fft(voltage_ind);
    P2_ind = abs(fft_volt_ind)/L_ind;
    P1_ind = P2_ind(1:L_ind/2+1);
    P1_ind(2:end-1) = 2*P1_ind(2:end-1);

    figure;
    subplot(2,1,1)
    plot (voltage1(m:n))
    title("Rekonstrueeritud signaali mõõtetulemused " + num2str(m) + "-
" + num2str(n))
    ylabel("Pinge (V)")
    xlabel("Mõõtetulemused (S)")
    subplot(2,1,2)
    plot(f_ind, P1_ind)
    title("Rekonstrueeritud signaali sageduste spekter mõõtetulemustel
" + num2str(m) + "-" + num2str(n))
    xlabel("Sagedus (Hz)")
    ylabel("Magnituud (|P1(f)|)")

    m=n;
    n=n + 20;
end
```

```
% Leia pinge RMS väärtus ja sageduste spekter 50 perioodi kohta

q = 50;
m = 20;
n = 40;

rms_volt = zeros(1, q);
periods = zeros(1, q);
fft_total = zeros(q, 11);

for k = 1:q
    voltage_ind = voltage(m:n);

    periods(k) = k;
    rms_volt_ind = sqrt((sum(voltage_ind.^2))/length(voltage_ind));
    rms_volt(k) = rms_volt_ind;

    T_ind = 0.0005;
    Fs_ind = 1/T_ind;
    L_ind = 20;
    t_ind = (0:L_ind-1)*T_ind;
    f_ind = Fs_ind*(0:(L_ind/2))/L_ind;

    fft_volt_ind = fft(voltage_ind);
    P2_ind = abs(fft_volt_ind)/L_ind;
    P1_ind = P2_ind(1:L_ind/2+1);
    P1_ind(2:end-1) = 2*P1_ind(2:end-1);
    fft_total(k, :) = P1_ind';

    figure;
    subplot(2,1,1)
    plot(voltage(m:n))
    title("Real time voltage from samples " + num2str(m) + "-" +
num2str(n))
    ylabel("Voltage (V)")
    xlabel("Samples (S)")
    subplot(2,1,2)
    plot(f_ind, P1_ind)
    title("Single-sided amplitude spectrum from samples " + num2str(m)
+ "-" + num2str(n))
    xlabel('Frequency (Hz)')
    ylabel('Magnitude (|P1(f)|)')

    m=n;
    n=n + 20;
end

% Loo maatriksid perioodide, pinge RMS ja FFT väärtuste kohta ja
sageduste kohta

maindata = zeros(q, 23);
maindata(1:q, 1) = periods';
maindata(1:q, 2) = rms_volt';
maindata(:, 3:13) = fft_total;

metadata = zeros(5, 11);
metadata(1, 1) = T_ind;
metadata(2, 1) = Fs_ind;
metadata(3, 1) = L_ind;
metadata(4, 1:11) = f_ind;
```

```
csvwrite("data\maindata.csv", maindata);
csvwrite("data\metadata.csv", metadata);

% Loo pinge RMS ja FFT tulemuste graafik

figure;
plot(voltage(1:400))

figure;
subplot(2,1,1)
plot(maindata(:, 1), maindata(:, 2))
title("Pinge RMS väärtused " + num2str(length(periods)) + " perioodil")
ylabel("Pinge (V)")
xlabel("Periood (n)")
subplot(2,1,2)
plot(metadata(4, 1:11), maindata(:, 3:13))
title("FFT tulemused " + num2str(length(periods)) + " perioodil")
xlabel("Sagedus (Hz)")
ylabel("Magnituud (|P1(f)|)")
```

Lisa 5 Elektroonika süsteemi ehitamise fotod