TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Elektroenergeetika ja mehhatroonika instituut

# NUTIKAS MÕÕTESEADE ELEKTRIVÕRGU PARAMEETRITE LOGIMISEKS

# SMART ENERGY MONITOR FOR LOGGING ELECTRICAL PARAMETERS

BAKALAUREUSETÖÖ

Üliõpilane: Robert Seredenko

Üliõpilaskood: 193997EAAB

Juhendaja: Indrek Roasto, vanemlektor

Tallinn 2022

*(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.

"....." ..... 20.....

Autor: .....
/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 20.....

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 20..... .

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

**Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]**

Mina, Robert Seredenko

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Nutikas mõõteseade elektrivõrgu parameetrite logimiseks,

mille juhendaja on Indrek Roasto,

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2022

---

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.*

# LÕPUTÖÖ LÜHIKOKKUVÕTE

*Autor:* Robert Seredenko *Lõputöö liik:* Bakalaureusetöö
*Töö pealkiri*: Nutikas mõõteseade elektrivõrgu parameetrite logimiseks

*Kuupäev:* 18.05.2022 98 *lk*

*Ülikool:* Tallinna Tehnikaülikool

*Teaduskond:* Inseneriteaduskond

*Instituut:* Elektroenergeetika ja mehhatroonika instituut

*Töö juhendaja(d):* Indrek Roasto

*Töö konsultant (konsultandid):* -

*Sisu kirjeldus:*

Antud töö eesmärgiks on projekteerida ja ehitada nutikas energiamõõtur, mis võimaldab mõõta pinget, voolu, võimsust, võrgusagedust, võimsustegurit, tarbitud elektrienergiat, vahelduvpinge ja -voolu kuju, harmoonikutest põhjustatud moonutusi, salvestada kogutud andmed andmebaasi ning kuvada andmed kasutajale.

Eesmärgi täitmiseks teostati turuülevaade olemasolevate seadmetele, uuriti elektriparameetrite mõõtmist, seejärel ehitati ja kirjutati programmikood prototüübile ning teostati katsed prototüübi pinge, voolu, võimsuse ja voolu moonutusteguri täpsuse leidmiseks.

*Märksõnad:* Energiamõõtur, FFT, IoT, ESP32, projekteerimine, veebileht, mõõtmine

# ABSTRACT

*Author:* Robert Seredenko *Type of the work:* Bachelor Thesis
*Title*: Smart Energy Monitor for Logging Electrical Parameters

*Date:* 18.05.2022 98 *pages*

*University:* Tallinn University of Technology

*School*: School of Engineering

*Department:* Department of Electrical Power Engineering and Mechatronics

*Supervisor(s) of the thesis:* Senior Lecturer Indrek Roasto

*Consultant(s):* -

*Abstract:*

The aim of this work was to design and build a smart energy meter that can do voltage, current, power, energy, frequency, power factor, voltage and current waveform measurement, measure total harmonic distortion, save measured data to external server and show measured data to user.

To achieve goals of this work, author did market analysis of existing devices and researched how electrical parameters are measured and based on two previous points specifications for device were chosen. Prototype was designed and built according to specifications and measurement accuracy of voltage, current, active power and current total harmonic distortion was tested.

*Keywords:* Energy monitor, FFT, IoT, ESP32, design, website, measurement

# LÕPUTÖÖ ÜLESANNE

| | |
|---|---|
| Lõputöö teema: | **Nutikas mõõteseade elektrivõrgu parameetrite logimiseks** |
| Lõputöö teema inglise keeles: | **Smart Energy Monitor for Logging Electrical Parameters** |
| Üliõpilane: | **Robert Seredenko, 193997EAAB** |
| Eriala: | **Elektroenergeetika ja mehhatroonika** |
| Lõputöö liik: | **bakalaureusetöö** |
| Lõputöö juhendaja: | **Indrek Roasto** |
| Lõputöö kaasjuhendaja: (ettevõte, amet ja kontakt) | |
| Lõputöö ülesande kehtivusaeg: | **2022/2023 2022/2023 Sügis** |
| Lõputöö esitamise tähtaeg: | **18.05.22** |

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

_____
Kaasjuhendaja (allkiri)

**1. Teema põhjendus**

Hetkel on eriti aktuaalne elektritarbimise jälgimine, kuna elektrihind tõusis märgatavalt. Seega otsustasin uurida, millised seadmed minu majapidamises kõige rohkem elektrienergiat tarbivad. Kuna elektrienergia tarbimist mõõdetakse vattmeetriga ja minul ei olnud kodust vattmeetrit võtta, otsustasin turul olemasolevaid seadmeid uurida. Uurimise käigus selgus, et need põhiliselt jagunevad kolme segmenti:

1. Kallid terviklahendused (mõeldud tööstussektorile)
2. Seadmed, mis ei logi mõõdetuid andmeid
3. Telefonirakendusega targa kodu seadmed

Terviklahendused, mis mõõdavad elektrienergia parameetreid ja kogutud andmed saadavad andmebaasi on põhiliselt suunatud tööstussektorile. Sellised valmislahendused maksavad rohkem kui 500 eurot, mis on kallis tavatarbija jaoks. Nt openZmeter, MYeBOX.

Turul on müügil palju seadmeid, mis suudavad mõõta elektrienergia parameetreid, kuid need kuvavad andmeid kasutajale ainult läbi sisseehitatud kuvari. Seadmes andmete logimist ei toimu ning logimise funktsiooni lisamine on keeruline. Seadmete hind jääb enamjaolt alla 25 euro.

Kolmandas segmendis seadmed enamasti logivad ainult võimsust ning andmed kuvatakse läbi telefonirakenduse. Kogutud andmeid on võimalik vaadata, aga keeruline kuidagi lisaks töödelda nt tarbitud elektrienergiat ei saa korrutada elektrienergia hinnaga, et leida rahaline kulu. Teistest erineb see segment selle poolest, et on võimalik läbi rakenduse ühendatud seade sisse/välja lülitada. Seadmete hind jääb enamjaolt alla 25 euro.

Alla 100 euro eest ei ole hetkel turul sellist toodet, mis lisaks tavalistele elektrienergia parameetritele suudaks mõõta ka pinge ja voolu siinuse kuju, harmoonikutest põhjustatud moonutusi ning võimaldaks salvestada kogutud andmed andmebaasi.

**2. Töö eesmärk**

Töö eesmärgiks on projekteerida nutikas energiamõõtur, mis suudab mõõta pinget, voolu, võimsust, võrgusagedust, võimsustegurit, tarbitud elektrienergiat, vahelduvpinge ja -voolu kuju, harmoonikutest põhjustatud moonutusi, salvestada kogutud andmed andmebaasi ning kuvada andmed kasutajale.

**3. Lahendamisele kuuluvate küsimuste loetelu:**

Turuülevaade mõõteseadmetest elektrivõrgu parameetrite jälgimiseks ja logimiseks
Elektrivõrgu parameetrite mõõtmine
Nutika mõõteseadme projekteerimine
Mõõteseadme kalibreerimine ja mõõtevea leidmine

**4. Lähteandmed**

Inspiratsiooni allikas:

https://www.analog.com/media/en/technical-documentation/user-guides/ev-ade9153ashieldz-ug-1253.pdf

Erinevate mõõtekivide andmelehed:
https://www.analog.com/media/en/technical-documentation/data-sheets/ade9153a.pdf
https://www.st.com/content/st_com/en/products/data-converters/metering-ics/stpm32.html
https://www.analog.com/en/products/ade9000.html#product-documentation

**5. Uurimismeetodid**

Kirjanduse kogumine ja analüüs
Prototüübi skeemi väljatöötamine ja trükkplaadi disain
Prototüübi ehitamine ja katsetamine
Prototüübi kalibreerimine
Tulemuste analüüs

**6. Graafiline osa**

Graafiline osa on nii põhiosas kui ka lisades. Olulisemad joonised on seadme põhimõtteline joonis ja elektriskeem, trükkplaadi joonised. Olulisemad tabelid on tehniliste andmete tabel, tabel mõõtetulemustega enne ja pärast kalibreerimist.

**7. Töö struktuur**

Sissejuhatus
1 Turuanalüüs
2 Ülevaade elektrivõrgu parameetrite mõõtmisest
3 Prototüübi projekteerimine ja ehitus
4 Prototüübi tarkvara
5 Täpsuse katsetamine ja spetsifikatsioonide saavutamine
Kokkuvõte
Kasutatud kirjandus

**8. Kasutatud kirjanduse allikad**

Raamatud – Jaan Järvik „Üldelektrotehnika", Nihal Kularatna „Digital and Analogue Instrumentation"
Andmelehed – ADE9153, ESP32

**9. Lõputöö konsultandid**

-

## 10. Töö etapid ja ajakava

Kirjanduse läbitöötamine ning teoreetilise osa kirjutamine (6.12.2021)

Esialgse prototüübi riistvara disaini valmimine (20.12.2021)

Esialgse programmikoodi valmimine (20.01.2022)

Täiustatud prototüübi riistvara disaini valmimine (24.02.2022)

Täiendatud programmikoodi valmimine (25.03.2022)

Seadme kalibreerimine (20.04.2022)

Lõputöö sisuosa valmimine (03.05.2022)

Juhendajale läbilugemiseks saatmine (03.05.2022)

Lõputöö viimased parandused ning juhendajale läbilugemiseks saatmine (09.05.2022)

Töö lõplik versioon valmis (16.05.2022)

# SISUKORD

# EESSÕNA

Idee arendada nutikas energiamõõtur tuli enda initsiatiivil, kuna olin huvitatud elektriparameetrite ja -kvaliteedi mõõtmistest kodus, kuid turul olevad elektrikvaliteeti mõõtvad seadmed on suhteliselt kallid. Seega otsustasin ise projekteerida ja ehitada sarnase funktsionaalsusega, kuid vähem täpse ja odavama seadme.

Sooviks tänada enda juhendajat Indrek Roasto abi eest lõputöö kirjutamisel.

# LÜHENDITE JA TÄHISTE LOETELU

ADC – analoog-digitaalmuundur

DFT – diskreetne Fourier’ teisendus

FFT – kiire Fourier’ teisendus

HTML – Veebilehtede kuvamiseks kasutatav keel (ingl k *HyperText Markup Language)*

IC – integraallülitus (ingl k *integrated circuit*)

IoT – asjade internet (ingl k *Internet of Things*)

JS – JavaScript programmeerimiskeel

JSON – universaalne andmeedastusformaat (ingl k *JavaScript Object Notation*)

LDO – lineaarne regulaator (ingl k *Low dropout regulator*)

OTA – traadita ülekanne (ingl k *Over the Air*)

PCB – trükkplaat (ingl k *Printed Circuitboard*)

RTOS – reaalaja operatsioonisüsteem (ingl k *Real-time operating system*)

SPI – jadaliidese standard (ingl k *Serial Peripheral Interface*)

THD – moonutustegur (ingl k *Total Harmonic Distortion*)

# SISSEJUHATUS

Alates 2021 aasta sügisest kasvas märgatavalt börsi elektrihind. Kuna hinnatõus oli rohkem kui 40% võrreldes eelmise aastaga [1], muutus majapidamises elektrienergia tarbimise jälgimine eriti aktuaalseks. Seega otsustasin uurida, kui palju elektrienergiat seadmed minu majapidamises tarbivad.

Elektrienergia mõõtmiseks kasutatakse elektrienergia mõõtureid. Need on seadmed, mis suudavad mõõta ühendatud koormuse poolt tarbitavat võimsust. Kuni digitaalsete energiaarvestite leiutamiseni olid eelmisel sajandil peamiselt kasutuses induktsioonarvestid, mis tuginesid pöördmagnetväljal ja loenduril. Induktsioonarvestid olid kõrge töökindlusega, kuid kohmakad ja madala täpsusega. Digitaalsete sardsüsteemide arenemisega oli peagi võimalik arvestid täielikult digitaliseerida, mis võimaldas märgatavalt suurendada arvestite funktsionaalsust. Tänapäeva digitaalsed energiaarvestid võimaldavad mõõta ja salvestada lisaks tarbitud elektrienergiale ka muid parameetreid nagu pinge, vool, võimsustegur ning võimaldavad kogutuid andmeid kaugelt lugeda [2].

Eelnevalt mainitud elektrienergia mõõturid olid peamiselt mõeldud elektrifirmadele elektrienergia tarbimise jälgimiseks, kuid viimase aastakümne jooksul on turule jõudnud ka tavatarbijale disainitud ja taskukohased nutikad energiamõõturid. Võrreldes kommertslike energiaarvestitega, mõõdavad need vähem elektrivõrgu parameetreid ja on madalama täpsusega, kuid see-eest kasutajasõbralikuma kasutajaliidesega ja madalama hinnaga.

Kui tavakasutajale oleks piisanud elektrienergia mõõtmiseks harilikust energiamõõturist, olen ma elektroenergeetika ja mehhatroonika eriala tudeng ning otsustasin enda bakalaurusetöö eesmärgina projekteerida ja ehitada nutikas ühefaasiline mõõteseade, mis suudab mõõta pinget, voolu, võimsust, võrgusagedust, võimsustegurit, tarbitud elektrienergiat, vahelduvpinge ja -voolu kuju, harmoonikutest põhjustatud moonutusi, võimalusega salvestada kogutud ning kuvada kasutajale.

Lõputöö esimeses peatükis teen ülevaate turul olemasolevatest seadmetest, nende hinnaklassidest ja funktsionaalsusest. Teises peatükis selgitan, kuidas ja milliseid andureid kasutades toimub elektrivõrgu parameetrite mõõtmine digitaalsetes energiamõõturites. Kolmandas peatükis sean projekteeritavale seadmele spetsifikatsioonid, kirjeldan lähemalt komponentide valimist ning riistvara ja trükkplaatide disainimist. Neljandas peatükis selgitan seadme tarkvara osi ja nende ülesandeid. Viiendas peatükis mõõdan seadme täpsust, leian mõõtmiste põhjal mõõtevead ning uurin kolmandas peatükis seatud spetsifikatsioonide saavutamist.

# 1 TURUÜLEVAADE OLEMASOLEVATEST ELEKTRIENERGIA MÕÕTURITEST

Esialgu saab jagada elektrienergia mõõturid kahte kategooriasse, ühefaasilised ja kolmefaasilised elektrienergia mõõturid. Kuna minu töö eesmärgiks on projekteerida ja ehitada ühefaasiline mõõteseade, ei vaatle ma kolmefaasilisi elektrienergia mõõtureid. Edasine elektrienergia mõõturite liigitamine on palju keerulisem, kuna tooteid turul on palju, kõikidel seadmetel on erinevad funktsionaalsused, sihtrühmad ning hinnaklassid. Enda uurimise põhjal jagasin turul olevad elektrienergia mõõturid nelja rühma:

1. Kommertslikud elektrienergia mõõturid
2. Lihtsad kuvariga elektrienergia mõõturid
3. Telefonirakendusega targa kodu funktsionaalsusega elektrienergia mõõturid
4. Professionaalsed elektritarbimise analüsaatorid koos elektrikvaliteedi analüüsimisega

Esimesse kategooriasse kuuluvad elektrienergia arvestid, mida kasutatakse tarbitava elektrienergia arvestamiseks ning selle põhjal esitatakse arve tarbijale (näide Joonis 1.1). Need arvestid omavad kõrget täpsust (mõõteviga alla 1%), mõõdavad kõiki põhilisi parameetreid (vool, pinge, sagedus, võimsus, võimsustegur jne), omavad mitut tariifi ja võimaldavad mõõdetuid andmeid kaugelt lugeda. Arvestite hind sõltub individuaalse seadme funktsionaalsusest ja maksimaalsest voolust, kuid jäävad hinnavahemikku 25 – 200 eurot.


Joonis 1.1 Vasakul Noark DIN-liistule paigaldatav elektrienergia arvesti [3] ja paremal Brennenstuhl elektrienergia mõõtur [4]

Teises segmendis on lihtsad elektrienergia mõõturid, mis on eelkõige mõeldud tavakasutajale. Need on võimalikult lihtsa ehituse ja madala hinnaga. Kuigi seadmed selles segmendis mõõdavad põhilisi elektrivõrgu parameetreid, kuvatakse mõõdetud

andmeid ainult sisseehitatud kuvarile (näide Joonis 1.1) ning harilikult salvestatakse mällu ainult tarbitud energia ja maksimaalne võimsus. Selle segmendi mõõturite hind jääb vahemikku 15 – 30 eurot.

Kolmandasse segmenti kuuluvad kõik targa kodu funktsionaalsusega elektrienergia mõõturid, alates tarkadest pistikupesadest kuni elektrikilpi paigaldatavateni energiamõõtjateni (vaata Joonis 1.2). Põhiliseks erinevuseks teistest segmentidest on seadmete traadita andmeside Wi-Fi, Bluetooth, ZigBee või Z-Wave teel, lihtne integreerimine olemasolevatele seadmetele, kinnine ökosüsteem ja võimalus seadmeid jälgida ning juhtida läbi telefonirakenduse. Levinud on andmete kogumine ning töötlemine pilves, mis tihti tähendab, et kuigi erinevad tootjad võivad kasutada sarnast riistvara, ei ole erinevate tootjate seadmed tarkvara tõttu võimelised koos töötama. Kodu automatiseerimine ja *IoT* on üheks oluliseks müügipunktiks selle kategooria seadmetel. Hinnavahemik on 15-150 euro vahel.

Neljandas kategoorias asuvad professionaalsed ja kõrge täpsusega elektritarbimise analüsaatorid nagu Fluke 1770 seeria analüsaatorid (vaata Joonis 1.2). Lisaks eelmainitud elektrivõrgu parameetritele on selle kategooria seadmed võimelised kuvama ka näivvõimsust, reaktiivvõimsust, faasinihet, eraldi pinge ja voolu sagedust, ruutkeskmist, keskväärtust, moonutustegurit, kuni 50 järgu harmoonikuid, lainekuju ja lülitusvoolu. Kuna selle segmendi seadmed omavad kõrget täpsust ja palju funktsioone ning mõeldud eelkõige professionaalidele, on ka seadmete hind märgatavalt kõrgem võrreldes eelmiste kategooriatega. Odavamad analüsaatorid maksavad alates 200 eurost kuni mitmetuhande euroni kallimate analüsaatorite puhul.


Joonis 1.2 Vasakul emporia elektrikilpi paigaldatav energiamõõtur [5], keskel Xiaomi tark pistikupesa [6] ja paremal Fluke 1770 seeria elektritarbimise analüsaator [7]

Enda poolt ehitaks energiamõõturi, millel oleks sarnane funktsionaalsus professionaalsetele elektrienergia mõõturitele, kuid madalamas hinnaklassis, madalama täpsusega ning targa kodu traadita funktsionaalsusega. Seadmele esitatavatele spetsifikatsioonidega saab lähemalt tutvuda Tabel 3.1.

# 2 ELEKTRIVÕRGU PARAMEETRID JA NENDE MÕÕTMINE

Minu poolt projekteeritav seade on eelkõige mõeldud kasutamiseks madalpingelise ühefaasiliste koormustega, seega lähtun oma analüüsiga ka sellest vaatepunktist. Iga parameetri juures kirjeldan esialgu lühidalt, mis parameeter endast kujutab ja peale seda põhjalikumalt vastava parameetri erinevatest mõõtmismeetoditest. Kuna kaasaegsed energiamõõturid põhinevad peamiselt digitaalsetel sardsüsteemidel ja mõõdetavad parameetrid on analoogsuurused, tuleb kõik mõõdetavad analoogsignaalid esialgu teisendada pingeks, et seda oleks võimalik teisendada analoog-digitaalmuunduriga (ADC) digitaalsignaaliks [8].

Mõõtmisviisid jagunevad kaheks: otsesed ja kaudsed mõõtmised. Otsesel mõõtmisel saadakse tulemus üldjuhul ühe mõõteriistaga ja mõõtmisega. Kaudsel mõõtmisel arvutatakse otsitav mõõtesuurus kasutades mitmeid mõõtesuurusi ning valemit [9].

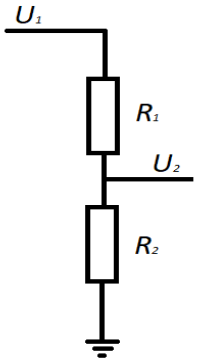
Elektrivõrgu parameetritest saab otseselt mõõta ainult pinget ja voolu, kõiki ülejäänud parameetreid tuletatakse kaudsete mõõtmistega.

## 2.1 Otsesed mõõtmised

### 2.1.1 Pinge mõõtmine

Pinget on võimalik mõõta otse ADC-ga, aga ADC-l esinevad mitmed piirangud. ADC sisendid töötavad ainult kindlas pingevahemikus, tüüpiliselt jääb ADC maksimaalne sisendpinge alla 5V. Seega kõrgemat pinget mõõtes tuleb signaal eelnevalt madaldada enne ADC klemmidele toitmist. Pinge madaldamiseks on kaks võimalust: kasutada pingejagurit või pingetrafot.

**Pingejagur** on mitmest takistist koosnev elektriskeem, kus sobitades takistuste suhet (vaata Joonis 2.1 ja valem 2.1) on võimalik langetada pinget seadmele sobivale tasemele. Pingejagurit saab kasutada nii vahelduvvoolu kui ka alalisvoolu teisendamiseks [2].




Joonis 2.1 Pingejaguri elektriskeem

$$U_2 = U_1 \frac{R_2}{R_1 + R_2} \tag{2.1}$$

kus $U_2$ - soovitud väljundpinge, V,

$U_1$ - sisendpinge, V,

$R_1$, $R_2$ - pingejaguri takistid, Ω.

Madalpinge mõõtmiseks on takistite suhe tavaliselt suurem kui 1000:1, mis tähendab, et kasutatavad takistused erinevad üksteisest 1000 korda. Näiteks kui mõõteseadmel oleks takistus 1 kΩ, siis teine takistus oleks 1000 korda suurem ehk 1 MΩ.

**Pingetrafo** on trafo, kus primaarmähis ühendatakse rööbiti elektrivõrku ning mõõteseade ühendatakse sekundaarmähise külge [2]. Kuna tegemist on trafoga, sobib pingetrafo ainult vahelduvvoolu mõõtmiseks. Sobivat pinget saab kätte muutes primaar- ja sekundaarmähise keerdude arvu. Pingetrafo väljundpinget saab leida järgmise valemiga:

$$U_2 = U_1 \frac{N_2}{N_1} \tag{2.2}$$

kus $U_2$ - väljundpinge, V,

$U_1$ - sisendpinge, V,

$N_1$ - primaarmähise keerdude arv,

$N_2$ - sekundaarmähise keerdude arv.

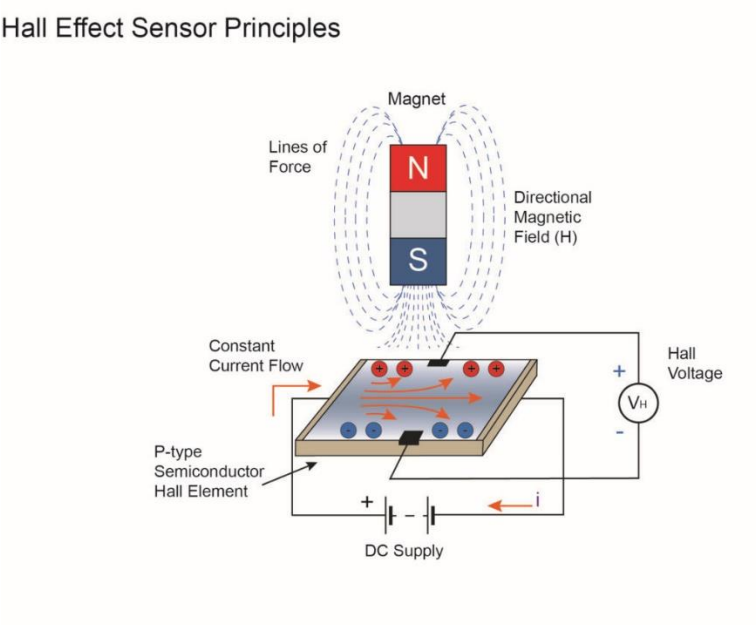
### 2.1.2 Voolu mõõtmine

Voolu energiamõõturites saab mõõta mitme erineva viisiga. Neli levinud viisi on kasutades voolutrafot, šunti, halli efekti andurit või Rogowski vööd. Kuna ADC ei suuda voolu mõõta, teisendatakse kõikide andurite puhul vool proportsionaalselt pingeks, mida saab ADC-ga mõõta.

**Voolutrafo** on trafo, kus primaarmähis on ühendatud jadamisi mõõdetavas ahelas ja sekundaarmähisesse indutseeritakse vool, mis on proportsionaalne vooluga primaarmähises. Mõõdetav vool leitakse korrutades sekundaarmähise vool ülekandeteguriga [2]. Et voolu oleks võimalik mõõta ADC-ga, lisatakse koormav takisti paralleelselt sekundaarmähise klemmidega, mis muundab voolusignaali pingesignaaliks [10]. Voolutrafo on galvaaniliselt isoleeritud ja sobib kasutamiseks ainult vahelduvvooluga.

**Šunt** on täppistakisti madala takistuse ja temperatuurisõltuvusega. Šunt ühendatakse jadamisi koormusega ning pingelang šundil näitabki voolutugevust. Šundi takistus hoitakse võimalikult madal, et hoida kadusid madalana. Sobib nii alalisvoolu kui ka vahelduvvoolu mõõtmiseks, kuid ei ole galvaaniliselt isoleeritud [2].

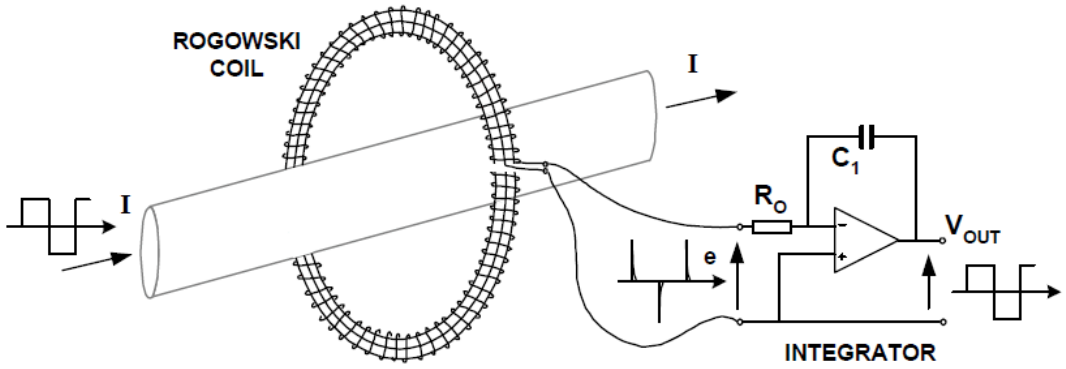
**Halli efektiga** andurid kasutavad voolutugevuse mõõtmiseks Halli efekti. Kui paigutada vooluga ristuvasse magnetvälja suhteliselt õhukese ja laia voolujuhi, mida läbib alalisvool, siis kahe magnetvälja liitumisel mõjub elektronidele jõud ja elektronid koonduvad juhi ühte serva (vaata Joonis 2.2) [2].




Joonis 2.2 Halli efekti illustreeriv joonis [11]

Kasutades seda efekti on võimalik elektrijuhis mõõta voolu või magnetvoo muutust, kuna elektrijuhi vahel tekkiva potentsiaalide vahe amplituud ja märk olenevad magnetvoo ja voolu suunast ning tugevusest [2]. Halli efekti anduritega on võimalik mõõta nii alalisvoolu kui ka vahelduvvoolu ning andur on galvaaniliselt isoleeritud koormusest. Kuna halli efekti andurid vajavad oma tööks rohkem energiat ja välist vooluallikat, teeb see halli anduri kasutamise keerulisemaks ja kallimaks. Seega on halli efekti andurid kasutuses pigem alalisvoolu mõõtmistes [12].

**Rogowski vöö** kujutab endast toroidalusele mähitud õhksüdamikuga pooli ehk olemuselt trafot. Trafo primaarmähiseks on pooli läbiv vooljuht ja sekundaarmähiseks õhksüdamikuga pool (vaata Joonis 2.3). Sekundaarpinge hetkeväärtus on võrdeline primaarvoolu hetkeväärtuse tuletisega. Kuna Rogowski vöö mõõdab voolu muutumist ajas, saab Rogowski vööga mõõta ainult vahelduvvoolu ja tulemuse saamiseks tuleb lisada ka integraator [2].




Joonis 2.3 Rogowski vöö koos integraatoriga [13]

## 2.2 Kaudsed mõõtmised

### 2.2.1 Võimsuse mõõtmine

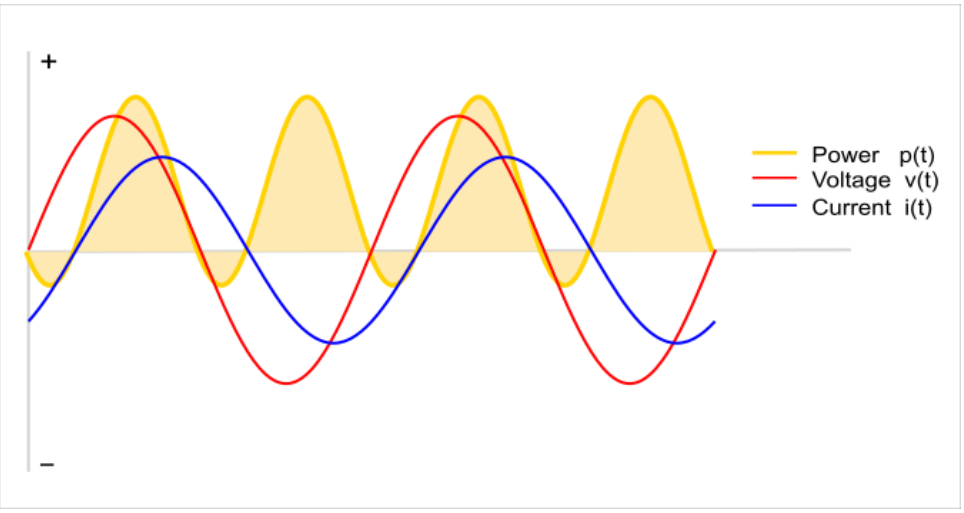
**Vahelduvvoolu võimsus** koosneb kahest komponendist: Aktiivvõimsusest ja reaktiivvõimsusest [2]. Nende kahe suuruse geomeetrilist korrutist nimetatakse näivvõimsuseks. Tavatarbijate energiamõõturites on oluline mõõta eelkõige aktiivvõimust, kuna just tarbitud aktiivvõimsusel esitatakse arve.

**Aktiivvõimsus** on elektrienergia, mida muundatakse tarbijal kasulikuks tööks. Aktiivvõimsuse arvutamiseks on olemas kaks valemit. Üks valem põhineb voolu ja pinge efektiivväärtustel ja teine voolu ning pinge hetkväärtusel [2].

$$P = UIcos\varphi \tag{2.3}$$

kus $P$ - aktiivvõimsus, W,
$U$ - pinge efektiivväärtus, V,
$I$ - voolu efektiivväärtus, A,
$cos\varphi$ - võimsustegur.

Valem 2.3 on küll mugav staatilise elektriskeemis võimsuse arvutamiseks, aga see valem sisaldab võimsustegurit, mis teeb muutuvas süsteemis võimsuse leidmise arvutuslikult mahukaks. Seepärast on elektriarvestites kasutuses teine variant kasutades hetkvõimsust.




Joonis 2.4 Hetkpinge, -voolu ja -võimsuse graafikud induktiivse koormuse korral [14]

Korrutades hetkvoolu hetkpingega saab kätte süsteemi hetkvõimsuse, mis näitab tarbitava võimsuse suurust ning suunda. Kui faasinihe voolu ja pinge vahel $\varphi$ ei võrdu nulliga, tekib teatud ajahetkedel negatiivne hetkvõimsus (vaata Joonis 2.4). See negatiivne hetkvõimsus on osa reaktiivvõimsusest, mis ühe poolperioodi jooksul on kord positiivne kord negatiivne, kuid summaarselt terve perioodi reaktiivvõimsus on 0. Kasutades seda reaktiivvõimsuse omadust on võimalik leida tarbija aktiivvõimsus kasutades ainult hetkpinget ja -voolu.

$$P = \frac{1}{T}\int_0^T u(t)i(t) \tag{2.4}$$

kus $T$ - mõõdetav periood, s,
$u(t)$ - hetkpinge, V,
$i(t)$ - hetkvool, I.

Seades valemis 2.4 mõõdetavaks perioodiks signaali ühe täisperioodi, tuleb aktiivvõimsuse tulemus samasugune kui valemis 2.3. Teist meetodit on ka lihtne implementeerida digitaalses energiamõõturis, kuna tegemist on korrutustehtega ning leitud hetkvõimsus tuleb ainult integreerida mõõdetava perioodi kohta.

**Näivvõimsus** on pinge ja voolu efektiivväärtuste korrutis, mõõdetakse volt-amprites.

$$S = UI \tag{2.5}$$

kus $S$ - näivvõimsus, VA.

### 2.2.2 Elektrienergia mõõtmine

Elektrienergia näitab, kui suurt võimsust on ajas tarbitud. Elektrienergia jaguneb kaheks: aktiivenergiaks ja reaktiivenergiaks ning nende leidmiseks summeeritakse vastav hetkvõimsus üle kindla aja (valem 2.6 ja 2.7).

$$W_a = \frac{1}{T}\int_0^T P \tag{2.6}$$

kus $W_a$ - aktiivenergia, Wh,

$T$ - mõõdetav periood, s.

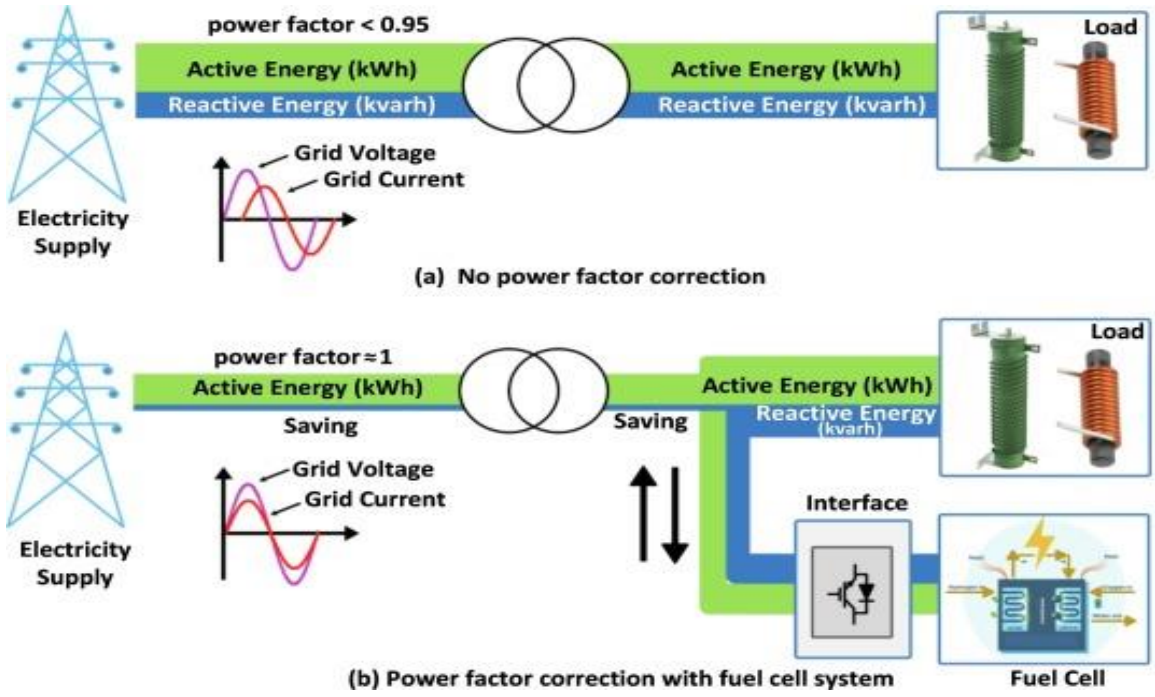
$$W_r = \frac{1}{T}\int_0^T Q \tag{2.7}$$

kus $W_r$ - reaktiivenergia, varh,

$T$ - mõõdetav periood, s,

$Q$ – reaktiivvõimsus, var.

Majapidamistes mõõdetakse ja makstakse harilikult ainult aktiivenergia eest, tööstusettevõtete puhul eraldi ka reaktiivenergia tarbimise eest, kuna reaktiivenergia ei tee kasulikku tööd ning asjata koormab elektrivõrku. Reaktiivenergia tarbimise vähendamiseks (võimsusteguri parandamiseks) paigaldatakse reaktiivenergiat kompenseerivad seadmed, mis kompenseerivad reaktiivenergia kohe tarbija juures ning ei lase reaktiivenergial võrku mõjutada (vaata Joonis 2.5).

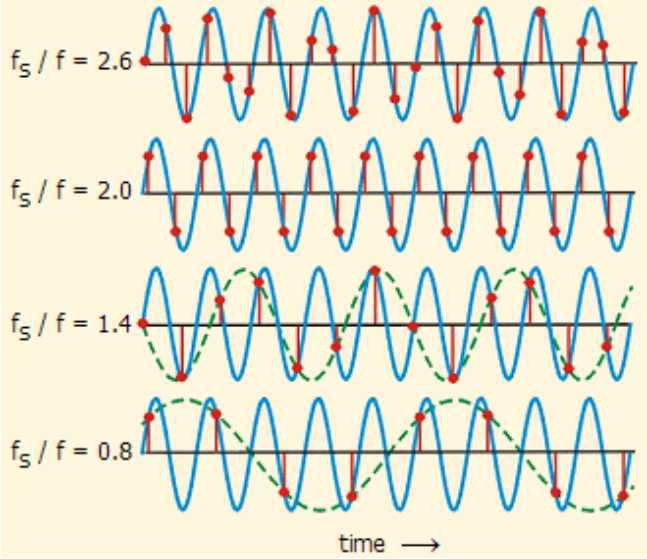



Joonis 2.5 Reaktiivenergia kompenseerimine [15]

Elektriarvestites on ka levinud mitme tariifi kasutamine. Kuna elektrijaamade tehnilis-majanduslikud näitajad on parimad nimiparameetrite juures, üritatakse koormust ühtlustada seades erinevad hinnad tööpäevadele ning nädalavahetustele ja öödele - päevased ja öised tariifid. Harilikult on öised tariifid madalamad kui on seda päevased tariifid [2].

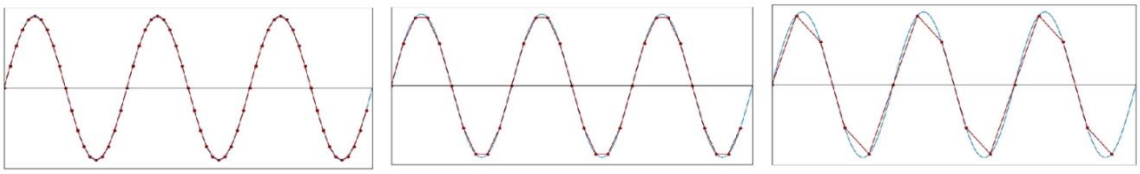
### 2.2.3 Lainekuju mõõtmine

Lainekuju mõõtmisel mõõdetakse vastava parameetri väärtust ning see kuvatakse ajas. Signaali lainekuju mõõtmiseks teisendab ADC analoogväärtuse digitaalseks väärtuseks ning salvestab muundatud väärtusse seadme mällu, seda tegevust nimetatakse ka diskreetimiseks. Kui seadme mälu on täis saanud, kuvatakse mõõdetud väärtuste jada kuvarile (või mõnel muul viisil), peale mida alustatakse uuesti otsast pihta. Lainekuju mõõtmisel tuleb kindlasti meeles pidada, et diskreetimissagedus oleks vähemalt kaks korda kõrgem kui mõõdetava signaali sagedus (Nyquisti-Shannoni teoreem). Kui diskreetimissagedus pole vähemalt kaks korda kõrgem, kaob diskreetimisel osa informatsiooni mõõdetava signaali sageduse kohta (vaata Joonis 2.6) [8].




Joonis 2.6 Näide siinuslaine diskreetimisest erinevatel diskreetimisagedustel [16]

Kuigi kahekordse diskreetimissageduse korral ei kao informatsioon mõõdetava laine kohta, ei sarnane mõõdetud signaal esialgsele signaalile. Parema signaalikuju kuvamiseks tuleb märgatavalt suurendada diskreetimissagedust kuni mõõdetud signaal sarnaneb esialgsele signaalile. Rusikareegel on seada diskreetimissagedus vähemalt kümme korda kõrgemaks mõõdetavast signaalist ja võimalusel veel kõrgemaks, vaata näiteid lainekuju taasesitusest erinevatel diskreetimissagedustel Joonis 2.7.

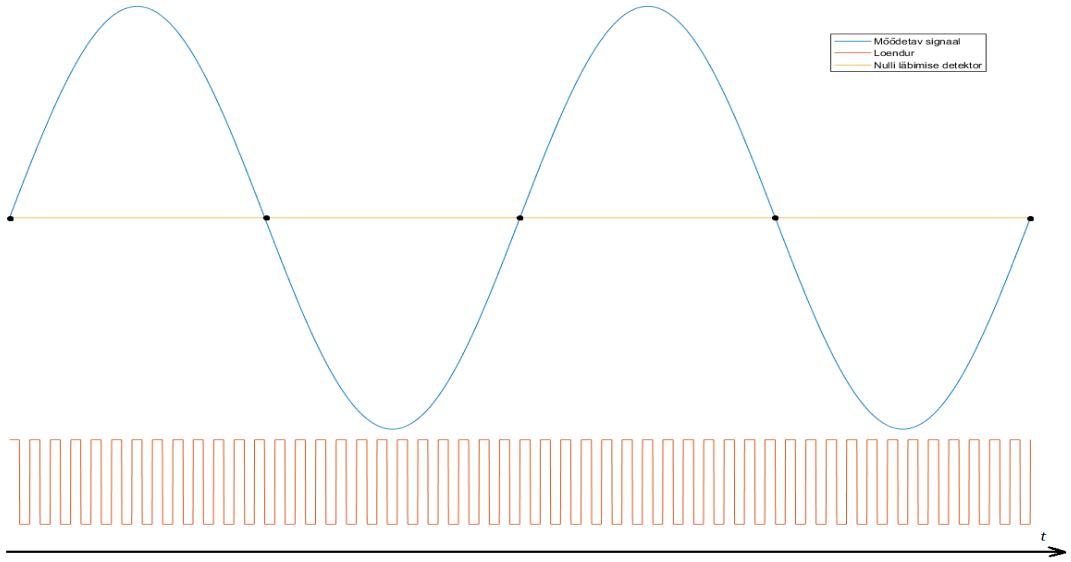


Joonis 2.7 Siinuslaine taasesitus 20, 10 ja 5 kordsel diskreetimissagedusel [17]

Lainekuju mõõtmisel pakub kõige rohkem huvi pinge ja voolu lainekuju, kuna elektrivõrgu pinge ja voolu lainekuju pole puhtalt siinuselised signaalid ning sisaldavad erinevatel sagedustel harmoonikuid ning häiringuid. Ainuke viis harmoonikute ja häiringute analüüsimiseks on läbi lainekuju ning seega on lainekuju mõõtmine üks oluline osa elektrikvaliteedi uurimisel.

### 2.2.4 Sageduse mõõtmine

Sagedus on vahelduvvooluallika pinge või voolu täisperioodide arv ühe sekundi jooksul [2]. Eesti elektrivõrgu nimisageduseks on 50 Hz, mis teeb ühe täisperioodi pikkuseks 20 ms. Sageduse mõõtmiseks on olemas kolm võimalust: kasutades kasvavat fronti, langevat fronti või nulli läbimise detektorit (ingl k *zero crossing detector*). Esimesed kaks on põhiliselt kasutuses digitaalses elektroonikas, kus signaalidel on ainult kaks võimalikku väärtust (kõrge või madal). Madalasageduslike vahelduvpinge sageduse mõõtmiseks sobib palju paremini nulli läbimise detektor, kuna siinuseline lainekuju läbib nulli kaks korda perioodi jooksul. Kasutades referentssagedust (*clock*), ühte loendurit ja nulli läbimise detektorit on võimalik mõõta sagedust ka digitaalsetes süsteemides. Kui nulli läbimise detektor avastab nulli läbimise, hakkab loendur referentssageduse pulsse lugema (vaata Joonis 2.8). Loendur loeb referentssageduse pulsse kuni nulli läbimise detektor avastab järgmise läbimise [18]. Otsitava signaali sageduse saab kätte jagades poolperioodi jooksult loetud pulsside arvu referentssagedusega ja korrutades tulemust kahega. Kindlasti tuleks meeles pidada, et referentssagedus oleks palju kõrgem kui otsitava signaali sagedus, vastasel juhul madala loenduri arvu tõttu võib tekkida suur viga sageduse arvutamisel.




Joonis 2.8 Sageduse mõõtmine kasutades nulli läbimise detektorit

### 2.2.5 Võimsusteguri mõõtmine

Võimsustegur iseloomustab suhet aktiivvõimsuse ja näivvõimsuse vahel. Kuna võimsustegur põhineb koossiinusfunktioonil, siis võimsusteguri väärtus jääb vahemikku 0 – 1. Mida lähemal on võimsusteguri väärtus ühele, seda väiksem on reaktiivvõimsuse osakaal, madalam vool ning madalamad kaod. Võimsusteguri leidmiseks on olemas mitu erinevat võimalust: kasutades aktiivvõimsust ja näivvõimsust (valem 2.8), mõõtes nurka pinge ja voolu vahel ning võttes nurgast koossiinus (valem 2.9) või kasutades aktiivtakistust ja näivtakistust (valem 2.10) [2].

$$PF = \frac{P}{S} \tag{2.8}$$

kus *PF* - võimsustegur.

$$PF = cos\varphi \tag{2.9}$$

kus $\varphi$ - nurk pinge ja voolu vahel, rad.

$$PF = \frac{r}{z} \tag{2.10}$$

kus *r* – aktiivtakistus, Ω,

*z* – näivtakistus, Ω.

Vastavalt eeltoodud võimalustele tuleb välja, et võimsustegurit ei ole võimalik otse mõõta. Lihtsaim viis võimsusteguri leidmiseks on kasutada esimest meetodit jagades aktiivvõimsust näivvõimsusega. Kuna energiamõõturid juba mõõdavad aktiivvõimsust ning näivvõimsust, ei valmista raskusi võimsusteguri leidmine.

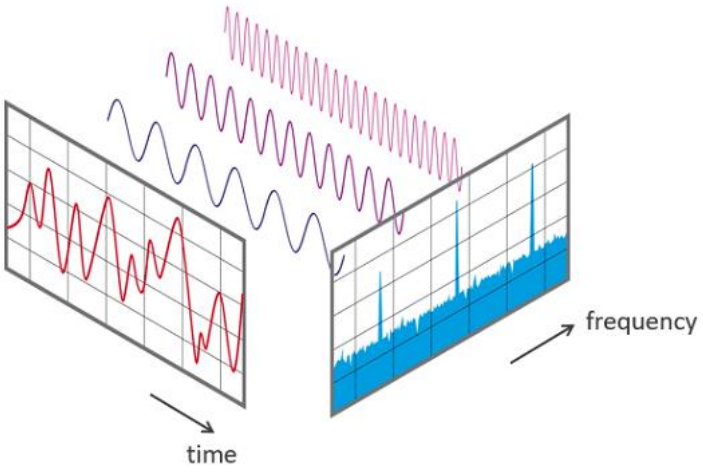
Teine võimalus on leida nurk vahelduvvoolu ja pinge vahel, aga see meetod on palju keerulisem ja arvutuslikult mahukam, kuna lisaks kahe siinussignaali vahelise nurga leidmisele tuleb ka võtta koossiinus leitud nurgast.

Kolmanda meetodi kasutamine pole praktiline energiaarvestites, kuna ühendatud seadmed ja seadmete arv võivad pidevalt muutuda ning takistuse mõõtmine on veel omakorda kaudne mõõtmine, mille tõttu täpsus langeks veelgi rohkem.

### 2.2.6 Diskreetne ja kiire Fourier' teisendus

Fourier' teisendus on matemaatiline manipulatsioon ajasignaali teisendamiseks sageduse signaaliks (vaata Joonis 2.9) [8]. Fourier' teisendust saab jagada kaheks, pidevaks teisenduseks ning diskreetseks teisenduseks. Digitaalsete signaalide

töötlemiseks sobib eelkõige diskreetne Fourier' teisendus (DFT), mida kasutasin ka enda seadmes.




Joonis 2.9 Signaal aja- ja sagedustasandil [19]

$$H\left(e^{j\omega}\right) = \sum_{k=-\infty}^{\infty} h[k]e^{-j\omega} \tag{2.11}$$

kus $H\left(e^{j\omega}\right)$ – sagedustasandi funktsioon,
ω – normaliseeritud muutuja vahemikus [-π,π],
$h[k]$ – ajatasandi funktsioon.

Paraku ei ole valem 2.11 hästi kasutatav arvutisüsteemides pideva muutuja $\omega$ pärast. Selleks, et arvutisüsteemides oleks mugav Fourier' teisendus teha, kasutatakse arvutustes lihtsustatud valemit 2.12. Selles valemis jagatakse ajatasandi signaal *N* pikkuseks jadaks, mis peale teisendust annab välja *N* pikkuse signaali sagedustasandil [8].

$$H\left(e^{j\frac{2\pi n}{N}}\right) = H[n] = \sum_{k=0}^{N-1} h[k]e^{-j\frac{2\pi nk}{N}} \tag{2.12}$$

kus *N* - arvujada pikkus,
$H[n]$ - sagedustasandi funktsioon,
*k*, *n* = 0, 1, … , *N*-1.

Jada pikkus *N* võib olla ükskõik milline naturaalarv, aga kui seada jada pikkuseks $2^a$, kus *a* on naturaalarv, on võimalik märgatavalt optimeerida Fourier' teisenduse tehete arvu $N^2$ tehelt $N \log N$-ni. Sellist teisendust nimetatakse kiireks Fourier teisenduseks (FFT) ja just see meetod on põhiliselt kasutuses arvuti- ja sardsüsteemides [8].

### 2.2.7 Moonutustegur

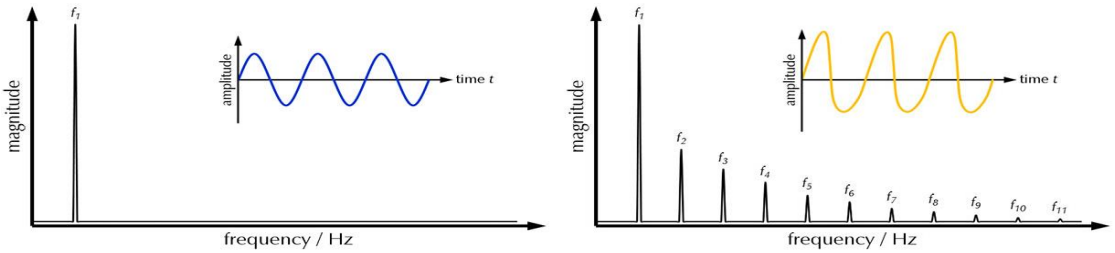
Moonutustegur (THD), näitab protsentides, kui palju on signaal moondunud ideaalsest siinusest (vaata Joonis 2.10) [8]. Enne kui THD saab leida, tuleb viia uuritav signaal ajatasandilt sagedustasandile ning sealt saadud harmoonikute väärtused saab kasutada valemis 2.13 THD leidmiseks. Oluline on meeles pidada, et FFT käigus kõrgeim saavutatav harmooniku järk sõltub otseselt diskreetimissagedusest, milleks Nyquisti-Shannoni teoreemi kohaselt on selleks pool diskreetimissagedusest.

THD väärtus on nullist suurem arv ning osadel juhtudel võib olla suurem kui 100%. Kui THD väärtuseks on 0%, tähendab seda, et tegemist on puhta siinuselise lainega ilma ühegi moonutuseta ning 100% tähendab moonutuste osakaal on võrdne põhisageduse osakaaluga.

$$THD = \frac{\sqrt{\sum_{i=0}^{n} V_{i_{rms}}^2}}{V_{fund_{rms}}} \tag{2.13}$$

kus $V_{i_{rms}}$ - i-nda harmooniku ruutkeskmise väärtus, V,

$V_{fund_{rms}}$ – põhisageduse ruutkeskmine, V.




Joonis 2.10 0% moonutusteguriga signaal vasakul ja >0% moonutusteguriga signaal paremal [20]

# 3 PROTOTÜÜBI PROJEKTEERIMINE JA EHITAMINE

## 3.1 Projekteerimine ning seadme spetsifikatsioon

Nutika energiamõõturi projekteerimisele saab läheneda kahelt poolt, kas seade täielikult nullist disainida või võtta olemasolev seade ning lisada sellele juurde nutikas funktsionaalsus. Järgnevas loetelus toon välja mõlema lähenemise plussid ja miinused.

Enda seadme disainimine plussid:

- Täielik kontroll kõikide seadme osade üle
- Võimalus hiljem teha muudatusi ning lisada uut funktsionaalsust

Miinused:

- Nõuab rohkem aega ja ressursse
- Suure tõenäosusega tuleb luua mitu prototüüpi
- Kogu seadme riistvara ja tarkvara tuleb ise disainida

Nutika funktsionaalsuse lisamine olemasolevale seadmele plussid:

- Tuleb disainida ainult lisatav osa
- Arendus odavam ja kiirem

Miinused:

- Ei ole täielikku kontrolli seadme üle
- Kindel funktsionaalsus, mida pole võimalik muuta, lisada või seadistada
- Palju tehnilisi piiranguid

Kuigi nutika funktsionaalsuse lisamine olemasolevale seadmele kõlab atraktiivselt, piiraks selline lähenemine arendamisvabadust ning seade sõltuks funktsionaalsusest, mis seadme tootja on seadmesse juba implementeerinud. Seadme disainimine kõlas väga huvitava väljakutsena ning põhinedes eelnvalt välja toodud põhjustele otsustasin seadmele täielikult enda disaini luua. Põhinedes esimeses peatükis tehtud turu-uurimisele, seadsin Tabel 3.1 spetsifikatsioonid minu poolt projekteeritud seadmele.

Tabel 3.1 Projekteeritavale energiamõõturile esitatavad spetsifikatsioonid

| | |
|---|---|
| Seadme omahind | alla 50 euro |
| Toetatud mõõtmisfunktsionaalsus | Pinge, vool, võimsus, sagedus, võimsustegur, energia, pinge ning voolu siinuse kuju, FFT, THD |
| Pinge, voolu, võimsuse mõõteviga | ±5% vahemikus 100-230 V |
| Seadme toide | Peab normaaltalitluses töötama sisseehitatud toiteallikaga ning elektrikatkestuse korral vähemalt 30 minutit iseseisvalt |
| Andmete kuvamine | Võimalus kasutajal andmeid kuvada nii juhtmevabalt kui ka otse seadmelt |

## 3.2 Riistvara valik

Otsustasin enda seadme ehitada mikrokontrolleri baasil, kuna kaasaegsed mikrokontrollerid omavad head võimekust ning on piisavalt madala energiatarbega, et kasutada sardsüsteemides. Mikrokontrolleri osas oli valik lihtne, ainuke hetkel turul olev mikrokontroller koos sisseehitatud Wi-Fi toega on Espressif ESP sarja mikrokontrollerid. Katsetamise käigus selgus, et parim mikrokontroller targa mõõturi jaoks oli kahetuumalise protsessoriga ESP32. Esimesed protüübid ehitasin valmis ESP8266 mikrokontrolleriga, kuid sellel esines palju probleeme süsteemi stabiilsusega ning ei piisanud jõudlust korraga FFT ja veebiserveri funktsionaalsuse jaoks.

Järgmisena tuli otsustada, kuidas teostada seadmel mõõtmisi. Arutasin kahe võimaluse vahel, kas teostada pinge ja voolu mõõtmisi kasutades ESP32 sisseehitatud ADC-d ning teha kogu signaalitöötlus mikrokontrolleris või kasutada mõõtmiste jaoks eraldiseisvat mikrokiipi.

Uurides Espressif *ADC API reference* dokumenti [21] tuli välja, et ESP32 ADC ei ole võimeline mõõta pinget alla 100 mV. Kui pinget saaks edukalt pingejaguriga madaldada, siis šundiga voolu mõõta ei saaks, kuna pingelang šundil oleks liiga madal, et sisseehitatud ADC-ga täpset tulemust saada. Voolutrafoga oleks saanud küll voolu mõõta, aga ESP32 ADC ei ole just kõige lineaarsem ja on tundlik välisele mürale. Lisaks sellele tehes signaalitöötlust teisel mikrokiibil, võimaldab see vähendada mikrokontrolleri koormust ning kasutada vabanenud ressurssi muude ülesannete jaoks. Eelnevalt mainitud põhjuste pärast otsustasin mõõtmisteks kasutada eraldi mikrokiipi.

Eraldi mikrokiibina oli mul valida, kas kasutada välist ADC-d või mõõtekivi koos sisseehitatud vahelduvvoolu mõõtmisfunktsionaalsusega. Kuna seadme projekteerimiseks ja ehitamiseks oli aega piiratud, oli otstarbekam enda disainis kasutada juba turul olevat elektrienergiat mõõtvat mõõtekivi, seeläbi lihtsustada riistavara- ja tarkvaraarenduse osa ning saavutada kõrgem seadme täpsus. Selline mõõtekivi on küll kallim lahendus kui ainult ADC kasutamine, kuid prototüübi jaoks aktsepteeritav.

Mõõtmiste jaoks kasutatav mõõtekivi peab olema võimeline mõõtma ühe faasi pinget, voolu, võimsust, võimsustegurit, sagedust, energiat ja pinge-voolu lainekuju. FFT ja THD ei pea olema osa mõõtekivi funktsionaalsusest, kuna seda signaalitöötluse osa saab teostada mikrokontrolleris.

Peale elektroonikakataloogide DigiKey ja Farnelli uurimist, otsustasin enda disainis kasutatavaks mõõtekiviks valida ADE9153A. Üheks selle mõõtekivi suureks eripäraks

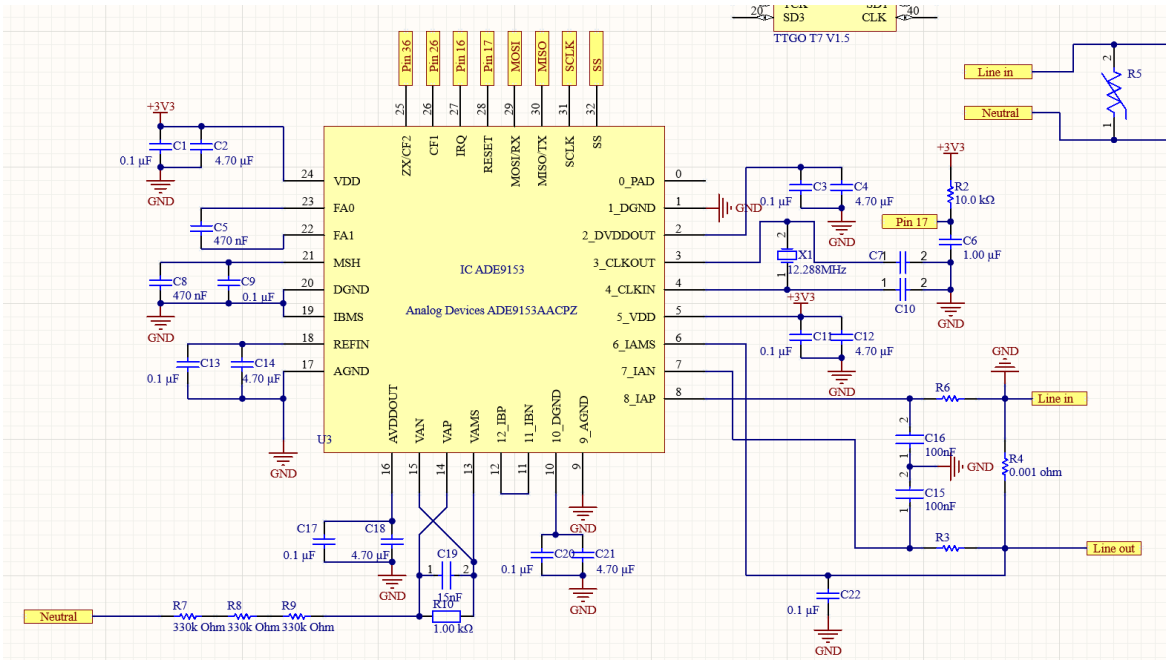
on sisseehitatud isekalibreerimise funktsioon mSure, mis võimaldab saavutada alla 1% mõõtevea kasutamata referentsmõõtevahendeid. See mõõtekivi suudab kõiki spetsifikatsioonides etteantud funktsioone täita, omab head täpsust ning muid kasulikke elektrikvaliteedi mõõtmisfunktsioone. Samas oli mõõtekivi tootjal Analog Device'l olemas teek mõõtekivi juhtimiseks ArduinoIDE keskkonnas, mis tegi tarkvara kirjutamise osa märgatavalt lihtsamaks.

Analog Device'l on olemas ka Arduino laiendusplaat (EV-ADE9153ASHIELDZ [22]) koos ADE9153A mõõtekiviga, aga sellel laiendusplaadil on väljaviigud Arduino UNO, Arduino Zero või ESP8266 jaoks, kuid mitte ESP32 jaoks. Seega ma ei saanud kasutada seda enda seadmes. Lisaks mittesobivale väljaviigule on see laiendusplaat suuruse poolest kohmakas ning ka liiga kallis energiamõõturis kasutamiseks. Ainuke ülejäänud võimalus on disainida enda trükkplaat ja see ise kokku joota.

## 3.3 Elektriskeem ja trükkplaadi disain

Enne kui saab hakata trükkplaati disainima, tuleb esialgu luua seadmele elektriskeem. Enda energiamõõturi riistvara osa saab jaotada kolmeks osaks: seadme toite osa, vahelduvvoolu mõõtmiste osa ja mikrokontroller koos ülejäänud komponentidega. Elektriskeemi ja trükkplaadi disainimiseks kasutasin Altium Designer tarkvara.

Vahelduvvoolu mõõtmise osa koosneb ADE9153A mõõtekivist ja paljudest muudest toetavatest komponentidest (kondensaatorid, takistid, šunt). Kuna mõõtekivi hakkab teostama mõõtmisi, tuleb olla hoolikas selle mikrokiibi ümbruse disainimisel. Mõõtekivi elektriskeem koos kõikide toetavate komponentidega on olemas Joonis 3.1.




Joonis 3.1 ADE9153 mõõtekivi elektriskeem

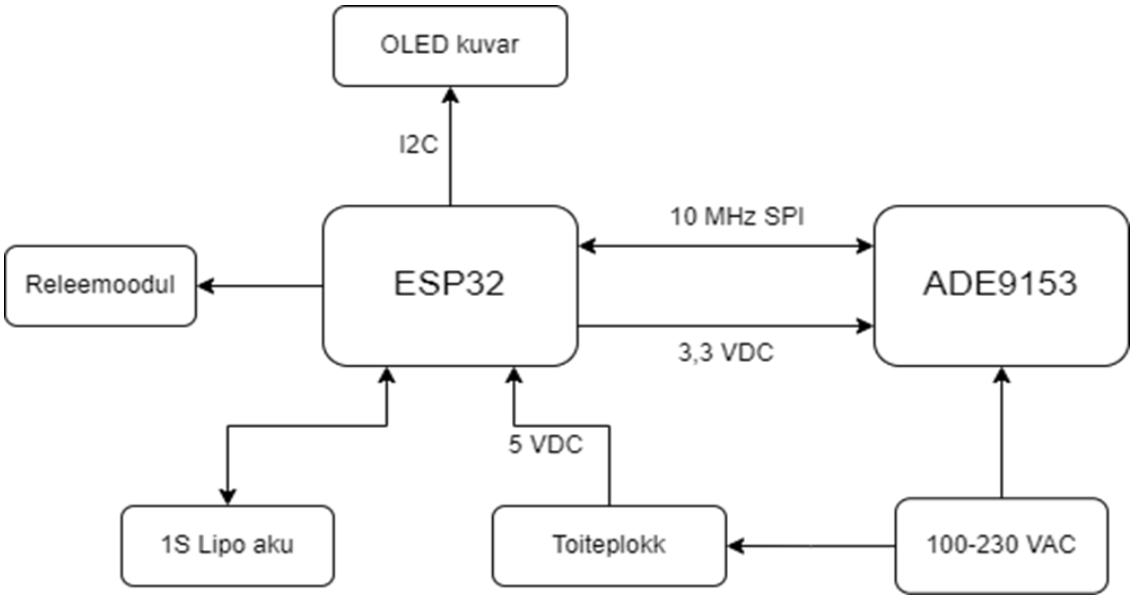
Seadme toite osa koosneb toiteplokist energiamõõturi toitmiseks, sulavkaitsmest ja varistorist seadme lühise või liigpinge kaitsme eest.

Ülejäänud elektriskeemi osa koosneb ESP32 mikrokontrolleri moodulist, välistest pesadest relee ja kuvari jaoks ning kruviterminalidest (vaata Lisa 1).

Energiamõõturis kasutan paralleelselt kahte andmevahetusprotokolli, SPI ja I2C. Mõõtekivi edastab andmed mikrokontrollerile üle SPI taktkiirusega 10MHz ja I2C siini kasutatakse kuvari juhtimiseks taktkiirusel 100kHz. Relee juhtimiseks kasutan ESP32 pin 1.

Kuna ma ei plaani teha energiamõõturit sisemisi ühendusi tavakasutuses kasutajale kättesaadavaks, sidusin mõõtekivi ja ESP32 mikrokiipide maad omavahel kokku ning ei asetanud komponentide vahele optoisolaatoreid, kuna see oleks märgatavalt tõstnud seadme hinda, suurust ja trükkplaadi keerukust.

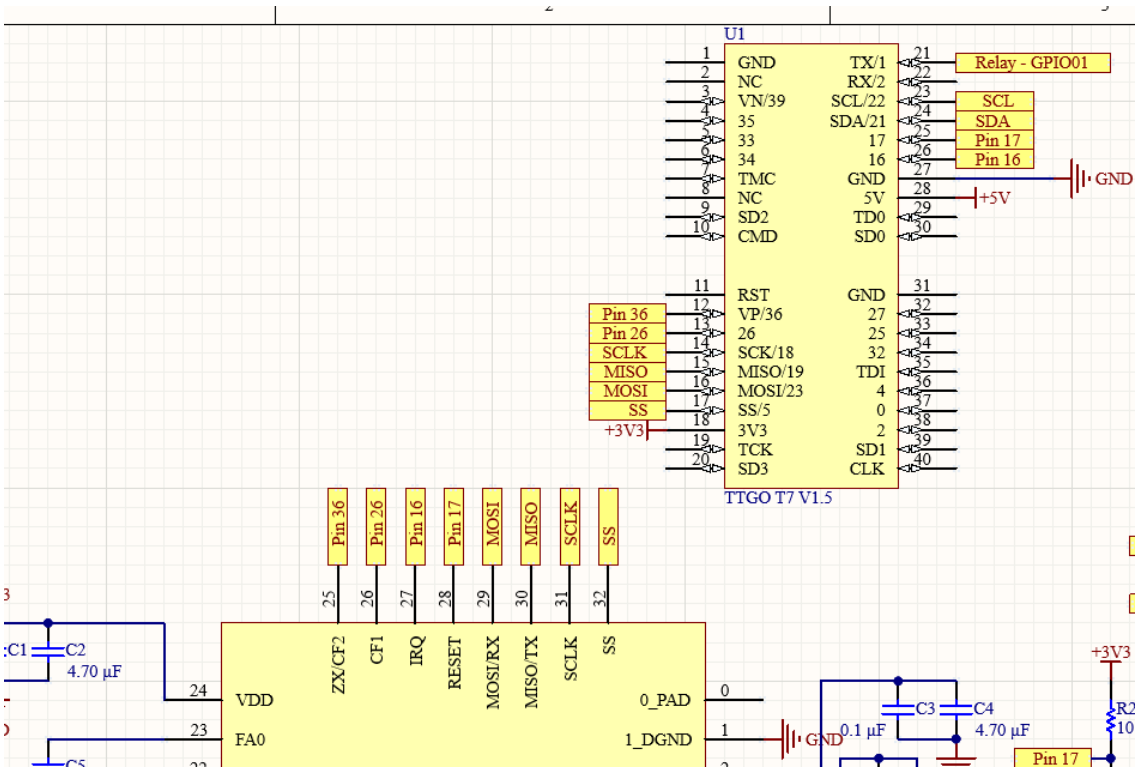
Disainitud energiamõõturi lihtsustatud elektriskeem on leitav Joonis 3.2 ja täielik elektriskeem kättesaadav Lisas 1.




Joonis 3.2 Energiamõõturi lihtsustatud plokkskeem

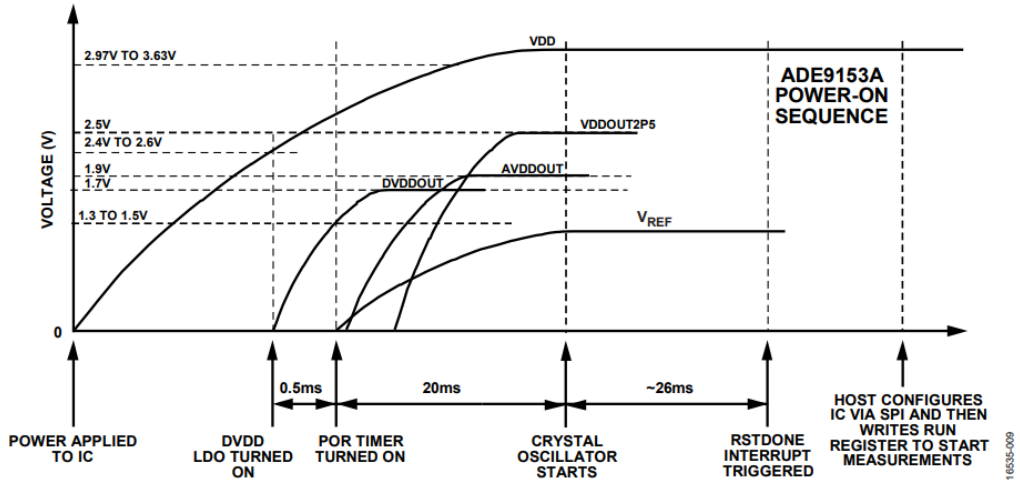
### 3.3.1 Elektriühendused mõõtekivi ja mikrokontrolleri vahel

Elektriskeemis SPI ühenduse loomiseks kasutan ESP32 VSPI-d, mis on ühendatud mõõtekivi SPI pinnide juurde külge. Lisaks mõõtekivi SPI pinnidele on mõõtekivil olemas 4 eraldi väljaviiku erinevate funktsioonide jaoks, milleks on Reset, IRQ, CF1, ZX/CF2. Kõiki eelnimetatud ühendusi saab vaadata lähemalt Joonis 3.3.




Joonis 3.3 Ühendused ESP32 ja ADE9153 mõõtekivi vahel

Reset on mõeldud mõõtekivi algseisundisse lähtestamiseks. Selleks tuleb reset pin hoida madalas seisus vähemalt 10 µs, peale mida lülitatakse mõõtekivi sisemised LDO välja ning käivitatakse uuesti vastavalt Joonis 3.4.




Joonis 3.4 ADE9153A mõõtekivi käivitusprotsess [23]

Sisselülitusel valib mõõtekivi vastavalt SS ja SCLK pinnide väärtustele andmevahetusprotokolli. Kui SS pin on madal ja SCLK pin on kõrge, on aktiivseks protokolliks SPI. Kui pinnide väärtused on vastupidi, on aktiivne protokoll UART. Mõõtekivi andmevahetusprotokolli töörežiimis vahetada ei saa, selleks tuleb mikrokiibile teha reset [24]. Reset pin on ühendatud ESP32 pin 17 külge.

IRQ (katkestus) pinni peamine kasutus on katkestuste saatmine mikrokontrollerile. Katkestuse sündmust on võimalik seadistada mõõtekivi registrite kaudu, näiteks pingelohu korral saab on võimalik saata teavitus mikrokontrollerile [24]. Kuigi see pin pole energiamõõturis kasutusel, tulevikukindluse mõttes ühendasin igaks juhuks mikrokontrolleriga. Ühendatud ESP32 pin 16 külge.

CF1 pin edastab signaaliimpulsse proportsionaalselt tarbitud elektrienergiale, mida on võimalik seadistada kuvama aktiiv-, reaktiiv- ja näivenergiat CFMODE registriga [24]. Ei ole energiamõõturis kasutuses, kuid tulevikukindluse mõttes ühendasin igaks juhuks mikrokontrolleriga. Ühendatud ESP32 pin 26 külge.

Viimane pin on multifunktsionaalne ZX/DREADY/CF2 pin, mis lisaks CF1 pinni funktsionaalsusele võimaldab tuvastada mõõdetava lainekuju nulli läbimist [24]. Ei ole energiamõõturis kasutuses, kuid tulevikukindluse mõttes ühendasin igaks juhuks mikrokontrolleriga. Ühendatud ESP32 pin 25 külge.

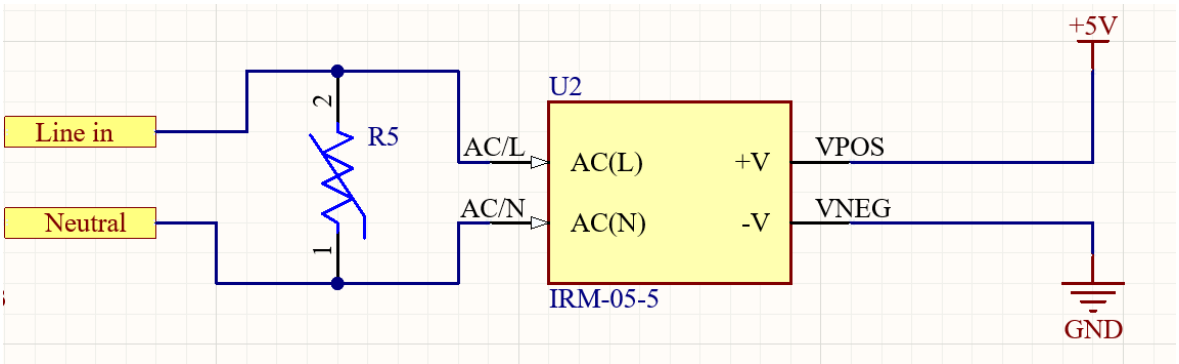
Kuvarina kasutan väikest 1,5 tollist 128x64 piksliga OLED kuvarit, mis töötab I2C andmevahetusprotokollil. SCL on ühendatud ESP32 pin 22 ja SDA ühendatud pin 21 külge.

Koormuse sisse-välja lülitamiseks kasutan valmis releemoodulit, mida saab juhtida kasutades kõrget ja madalat väärtust pinnil. Enda seadmes on releemoodul ühendatud ESP32 pin 1 külge. Siin tuleb välja tuua, et pin 1 on samal ajal ka ESP32 UART-i TX pin. See tähendab, et korraga ei ole võimalik kasutada releed ja ESP32 jadaliidest. Ma kasutasin laialdaselt UART-i seadme disaini katsetamisel ja koodi silumisel, kuid tarkvara lõppversioonis on kogu info kättesaadav läbi ESP32 veebilehe ning sain jätta viimases tarkvaraversioonis jadaliidese välja.

### 3.3.2 Seadme voolutarve ja sobiva toiteploki valimine

Toite osa vastutab 230 V vahelduvvoolu muundamiseks 5 V, kuna nii mikrokontroller kui ka mõõtekivi töötavad ainult madalapingelisel alalisvoolul. ESP32 moodulil asuv lineaarne muundur muundab 5 V 3,3 V-ks, mis on vaja nii ESP32 kui ka mõõtekivi jaoks. Lisaks toiteplokile tuleb seadmele implementeerida ka kaitsekomponendid nagu liigpingekaitse ning sulavkaitse lühise vastu.

Toiteploki valimisel pidin olema väga hoolikas, kuna energiamõõturi puhul tuleb paar piirangut ette. Energiamõõtur hakkab suhtlema kasutajaga üle Wi-Fi, seega toiteplokk võiks tekitada võimalikult vähe elektromagnetilisi häireid. Mul pole eelnevat koguemust toiteplokkide ehitamisega, seetõttu otsustasin kasutada valmisehitatud ja elektromagnetiliste häiringute kvalifitseeritud toiteplokki. Nii saan vähendada ka trükkplaadi kompleksust ja muuta lõppseade kompaktsemaks.




Joonis 3.5 Energiamõõturi toite elektriskeem ilma sulavkaitseta

Toiteploki väljundpinge valimisel üllatusena esines ka mitu väljakutset. Esialgu plaanisin toiteploki väljundpingeks valida 3,3 V, kuna kõik komponendid trükkplaadil töötavad selle pinge juures ning tundus mõistlik kasutada võimalikult vähe vahepealseid muundureid. Esimesel kolmel trükkplaadi versioonil kasutasingi 3,3 V

väljundpinget ja seade töötas probleemideta. Kui tagavaratoite spetsifikatsiooni täitmiseks lisasin 1S lipo aku, sain teada, et ESP32 moodulisse sisseehitatud aku laadimiskontroller laeb akut ainult 5 V sisendpinge juures. 3,3 V juures laadimiskontroller ei tööta ja isegi kui töötaks, suudaks see laadida Lipo aku kuni 3,3 V-ni. See tähendaks, et aku oleks kogu aeg peaaegu tühi ning juhul kui toide ära kaob, ei suudaks tagavaratoidet spetsifikatsioonides esitatud ajaks pakkuda. Seepärast pidin toiteploki väljundpingeks valima 5 V.

Peamised kaks tarbijat energiamõõturis on Lipo aku laadija (500 mA 5 V juures) ja ESP32 mikrokontroller (200 mA 3,3 V juures), mõõtekivi voolutarbimine on võrdlemisi madal (10 mA 3,3 V juures) ning relee tarbib elektrit ainult koormust väljas hoides (100 mA 3,3 V juures). Eraldi LDO-d ma trükkplaadile lisama ei hakanud, kuna ESP32 moodulil juba asub LDO, mis muundab 5 V 3,3 V-ks ning sellel LDO-l on piisavalt varu, et kõik 3,3 V vajavad seadmed toita. Kokku on ESP32 mooduli LDO võimeline toitma 600 mA 3,3 V juures, kust ESP32 kasutab ~200 mA, mõõtekivi ~10 mA, relee ~100 mA, OLED kuvar ~20 mA , mis teeb kokku 330 mA. Peale pikaajalist seadme testimist, ühegi süsteemi osas toitega probleeme ei esinenud.

Toiteplokiks valisin IRM-05-5, mis suudab 230 V juures väljundpingel 5 V toita 1 A. Seadme maksimaalne tarbimine jääb 5 V juures on ~600 mA, mis on ainult Lipo aku laadimise ajal ning normaalseisundis on tarbimine pigem ~200 mA. Seadme töökindluse mõttes otsustasin jätta võimsama toiteploki. Teine põhjus selle toiteploki valikuks oli komponendi kerge kättesaadavus. See oli olemas kõikides suuremates elektroonikakataloogides ning laoseis oli ka hea, kokku üle 10000 tk, mis on oluline praeguses elektroonikakomponentide laialdase puuduse ajal. Toiteploki andmelehe põhjal valisin ka vajalikud kaitsekomponendid nagu sulavkaitse ja varistori.

1S Lipo aku suuruse valisin vastavalt spetsifikatsioonis esitatud 30 minuti nõudele ja tarbitavaks võimsuseks võtsin seadme normaalseisundis tarbimise (~1W). Minimaalse aku mahtuvuse leidmiseks kasutasin järgmist arvutust:

$$\frac{1\,W \cdot 0{,}5\,h}{3{,}7\,V} = 0{,}135\,Ah = 135\,mAh$$

Vastavalt ülaltoodud arvutusele tuli minimaalseks aku mahtuvuseks 135 mAh, aga kuna mul oli käepärast ainult 450 mAh aku, siis prototüübis kasutasin seda.

### 3.3.3 Trükkplaadi disain

Trükkplaadi disainimisel kasutasin peamiselt abiks ADE9153A mõõtekivi andmelehte [24], ADE9153A laiendusmooduli elektriskeemi [22] ja AN-1562 dokumenti [25]. Nendes kolmes dokumendis on olemas peaaegu kogu vajalik informatsioon trükkplaadi

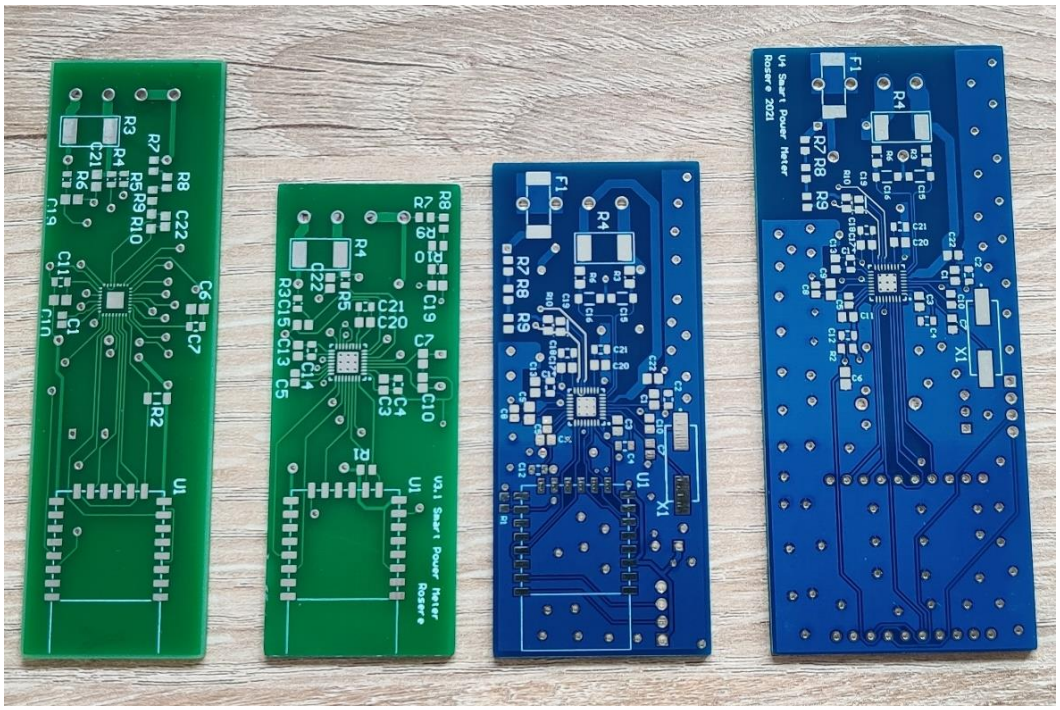
disainimiseks nagu ADE9153A näidisskeemid, soovitatud komponentide asetused trükkplaadil ja valemid komponentide valikuks. Lisaks mõõtekivi näidisskeemis olevatele komponentidele lisasin omalt poolt trükkplaadile ESP32 mikrokontrolleri, toiteploki, väljaviigud relee ja displei jaoks ning sulavkaitse koos varistoriga.

Erilist tähelepanu pidin osutama mõõtekivile ja teistele analoogmõõtmistes osalevatele komponentidele, kuna ebakorrektne komponentide asetus võib märgatavalt mõjutada mõõtmiste täpsust. Komponentide valimine ja asetamine vastavalt AN-1562 dokumendile [25] tegi küll trükkplaadi suuremaks, aga see-eest sain olla kindlam seadme korrektses töötamises.

Kuna energiamõõtur töötab elektrivõrgust tuleval vahelduvvoolul, pidin komponentide asetamisel ja radade loomisel jätma piisava vahe faasi ja neutraali vahel. Kui need kaks rada üksteisele liiga lähedale asetada, on oht, et võib toimuda elektriline läbilöök ning seade saab laialdaselt kahjustada. Lisaks tuleb võimalikult kaugele hoida nii faasi kui ka neutraali rajad madalpinge osast, et vältida nii läbilööki kui ka elektromagnetilisi häiringuid madalpinge osas.

Energiamõõturi disainis kasutan voolu mõõtmiseks šunti, seega peavad voolu kandvad rajad olema piisvalt laiad, et saaksid ilma liiga suure kuumenemiseta nimivoolul töötada. Rajade laiuste leidmiseks kasutasin veebis kättesaadavat kalkulaatorit [26].

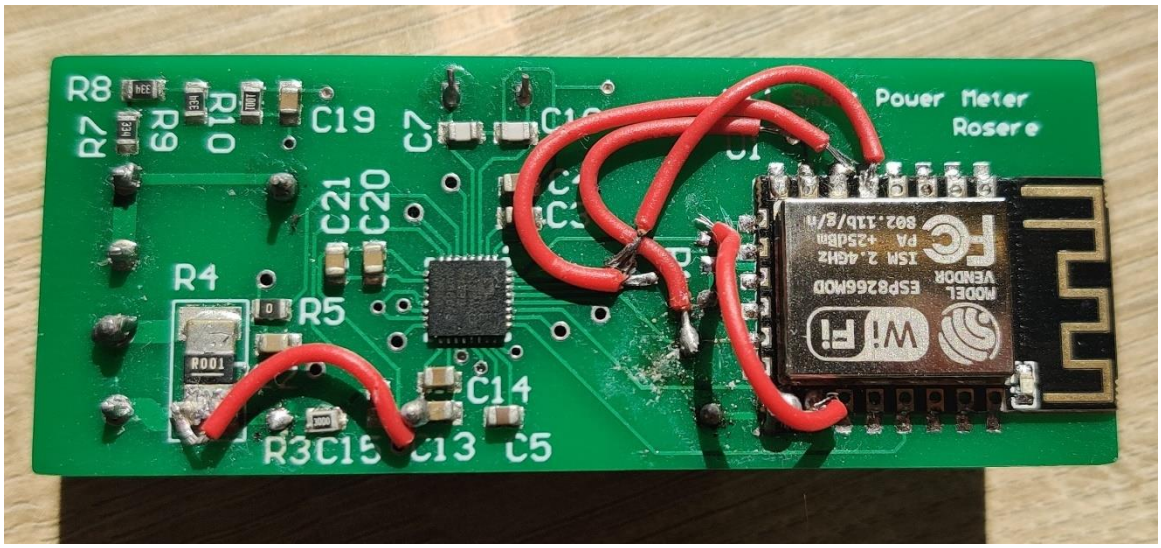
Kokku on trükkplaadil 39 komponenti (komponentide nimekiri on kättesaadav Lisas 2), neist 6 põhinevad läbivauk ning kõik ülejäänud SMD tehnoloogial. Kuigi komponentide arv trükkplaadil pole madal, peale mitut trükkplaadi versiooni ning mitmeid parandusi suutsin mahutada kõik komponendid ühe trükkplaadi poolele ning jätta trükkplaat kahekihiliseks. Kokku pidin disainima 4 erinevat trükkplaadi versiooni, kuna igas versioonis esines mitmeid suuri vigu. Trükkplaatide versioone saab näha Joonis 3.6.



Joonis 3.6 Neli trükkplaadi versiooni, versioonid vasakult paremale

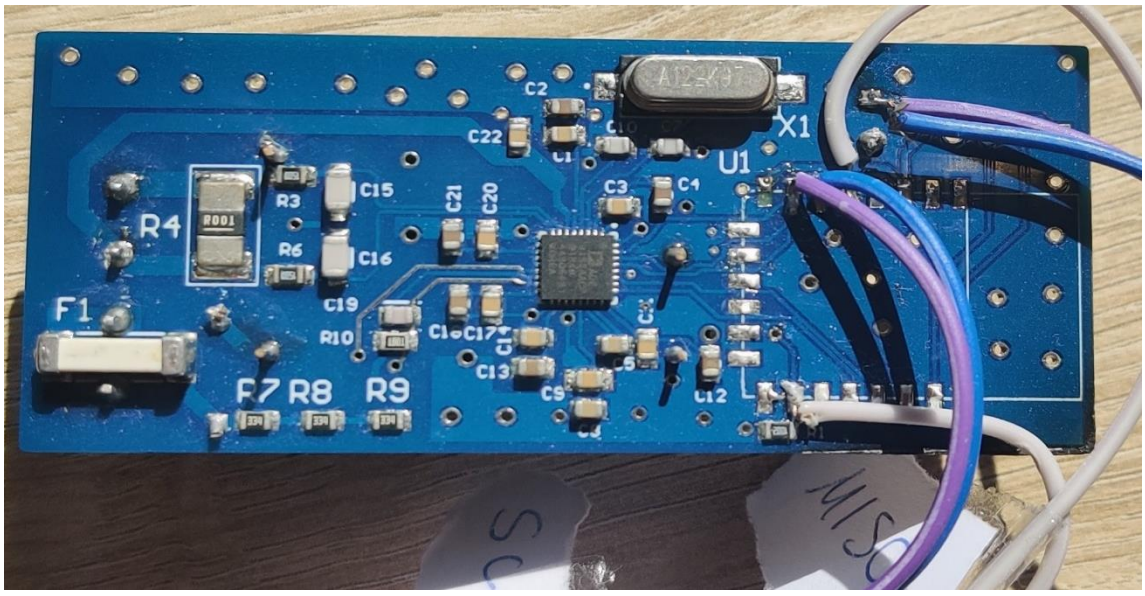
Esimese trükkplaadi ei suutnud ma üldse kokku joota, kuna mõõtekivi jalajälje tegin ise ja keeruka väljaviigu (32 pin LFCSP) tõttu ei suutnud jootekolviga trükkplaadile joota.

Teises versioonis kasutasin internetis võetud jalajälje ADE9153A jaoks, tegin disaini kompaktsemaks, kuid pärast trükkplaadi kokku jootmist ei saanud mikrokontroller ühendust mõõtekiviga. Tuli välja, et ühendasin mõõtekivi SPI pinnid vale mikrokontrolleri SPI külge. Selle asemel, et ühendada välise SPI külge, ühendasin mikrokontrolleri sisemise SPI mälu külge. Selleks et komponendid raisku ei läheks, lõikasin SPI rajad läbi ja jootsin käsitsi juhtmed õigete pinnide külge (vaata Joonis 3.7). Peale seda läks SPI andevahetus tööle, kuid voolumõõtmised ei toiminud, kuna olin unustanud ühendada faasi mõõtekivi maaga, mis on vajalik voolumõõtmiste jaoks. See trükkplaadi versioon oli kõikidest versioonidest kõige väiksem, kuna komponendid asetsesid mõlemal trükkplaadi poolel ja komponentide asetused ei olnud vastavuses AN-1562 dokumendiga [25].




Joonis 3.7 Käsitsi parandatud teine trükkplaadi versioon

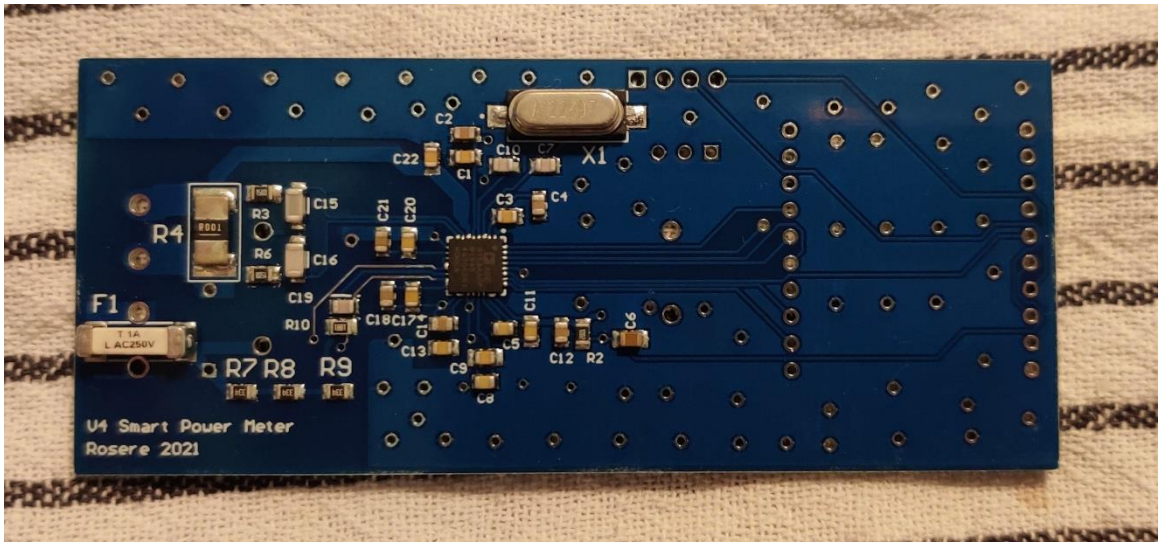
Kolmandas versioonis viisin võimalikult palju komponente ühele trükkplaadi poolele, lisasin väljaviigu I2C kuvari jaoks, ühendasin ADE9153 õige SPI külge ja ühendasin faasi mikrokontrolleri maaga. Lisaks viisin komponentide asetused vastavusse AN-1562 dokumendile, mis suurendas trükkplaadi mõõtmeid, kuid mitte palju. Valmisjootes ning seade käima pannes ei suutnud uuesti SPI ühenduse mikrokiipide vahel käima saada. Kuigi SPI rajad olid õige SPI külge ühendatud, olin valesti ühendanud MOSI, SCLK ja SS rajad, mille tõttu jootsin mikrokontrolleri plaadilt lahti ja jootsin trükkplaadile külge juhtmed (vaata Joonis 3.8). Nii sain katsetada mitmeid erinevaid mikrokontrollerid ja ei olnud ainult ühe mikrokontrolleriga piiratud. Katsetamise käigus sain teada, et ESP8266 võimekusest ei piisa minu seadme jaoks ning mul tuleb mikrokontroller vahetada välja ESP32 vastu.




Joonis 3.8 Käsitsi parandatud kolmas trükkplaadi versioon

Neljandas ja viimases trükkplaadi versioonis asendasin ESP8266 mikrokontrolleri ESP32-ga ja parandasin SPI ühendused. ESP32 mooduli suuruse tõttu on viimases trükkplaadi versioonis trükkplaadi mõõtmeteks 92 mm * 39 mm, mis on märgarvalt suurem võrreldes eelmistega. Seega otsustasin vaba ruumi ära kasutada ja lisasin ka väljaviigud välisele releemoodulile. Tagavaratoite spetsifikatsiooni täitmiseks vahetasin välja sellel trükkplaadi versioonil ka toiteploki 5 V väljundipingega. Sellel versioonil asuvad ka kõik SMD komponendid ühel plaadipoolel, mis teeb seadme jootmise veelgi kiiremaks.

Alates teisest trükkplaadi versioonist kasutan trükkplaadi valmisjootmiseks jootepastat ning šablooni, kuna ADE9153A mõõtekivi on 32-pin LFCSP väljaviiguga, mille jootmine jootekolviga on väga keeruline. Jootepasta ja šablooni kasutamine tegi jootmisprotsessi märgatavalt kiiremaks ning kergemaks, kuna käsitsi 33 SMD komponendi jootmine on aeganõudev ning nõuab häid jootmisoskusi. Jootepastat kasutades piisab komponentide asetamisest trükkplaadile ning rakendades tervele trükkplaadile temperatuuri vähemalt 260 °C muutub jootepasta vedelaks ning joodab korraga kõik asetatud komponendid õigesse kohta kinni (Joonis 3.9). Kuna mõõtekivi komponendi väljaviigud on väiksed ning jootepastat kannan trükkplaadile käsitsi, võib jootmise käigus tekkida jootesillad väljaviikude vahele. Enne seadme sisselülitamist tuleb need kindalasti käsitsi jootekolviga eemaldada, vastasel juhul ei pruugi seade korrektselt töötada või mõõtekivi saab sisselülitamisel kahjustada.




Joonis 3.9 Neljas trükkplaadi versioon valmisjoodetud SMD komponentidega

# 4 ENERGIAMÕÕTURI TARKVARA

Nutika energiamõõturi tarkvara jaguneb kaheks põhiliseks osaks: mikrokontrolleri ja veebilehe programmikood. Veebilehe kood jaguneb omakorda veel kaheks: kasutajapoolseks veebileheks ning veebilehe skriptiks. Järgmistes alapeatükkides kirjeldan lähemalt, millistest alamosades programmiosad koosnevad ning ka milliste ülesannete eest kumbki koodiosa vastutab.

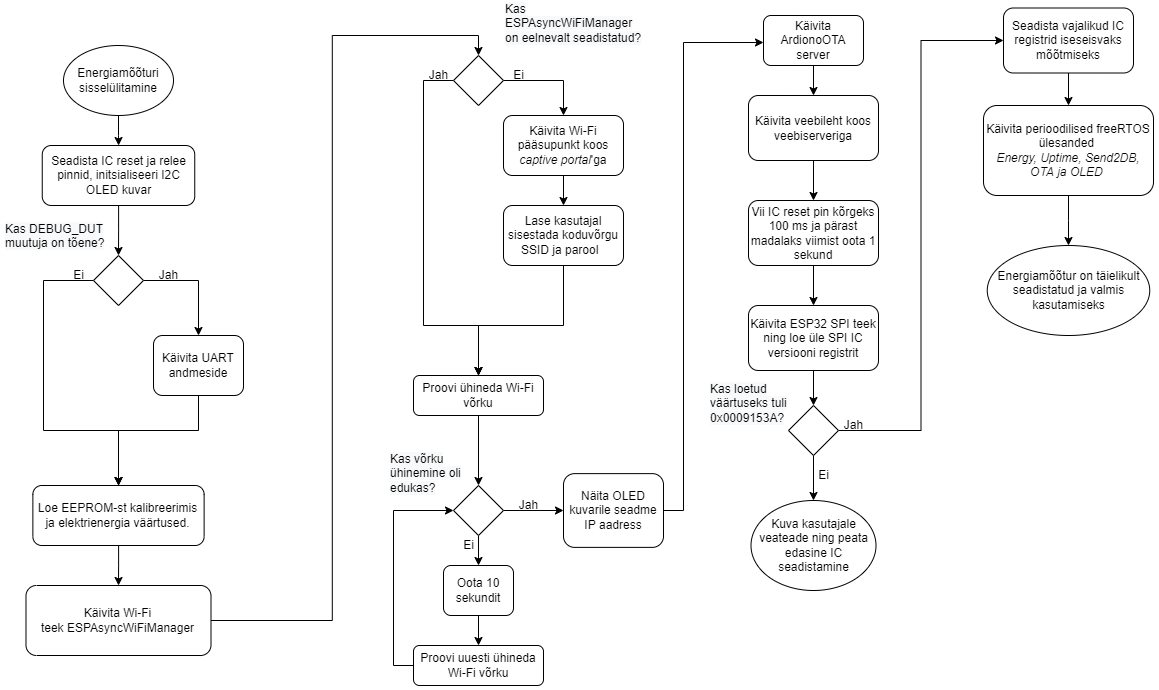
## 4.1 Mikrokontrolleri programmikood

Mikrokontrolleri kood on kõige mahukam ja olulisem osa kahest koodiosast. See vastutab andmevahetusega mõõtekiviga, saadud andmete töötlemisega (FFT ja THD), andmete kuvamisega OLED kuvarile, relee juhtimisega ja andmete salvestamisega. Lisaks kõikidele eelnimetatud ülesannetele seadistab see koodiosa üles ka veebilehe ning vahendab andmeid mikrokontrolleri ja veebilehe vahel.

Mikrokontrolleri programmi kirjutamisel kasutasin programmeerimiskeelena C++, kuna see on hetkel kõige levinud programmeerimiskeel mikrokontrollerite jaoks ning olen ise peamiselt seda programmeerimiskeelt varem kasutanud. Kuna ESP32 omab mikrokontrolleri jaoks kaks üpriski võimsat tuuma, võimaldas see laialdaselt kasutada RTOS (*Real-Time Operating System*) funktsionaalsust. See oli suur eripära võrreldes enda eelmiste projektidega, kus kogu kood asub ühes *while* tsüklis.

ESP32 kasutab freeRTOS operatsioonisüsteemi, mis oli välja töötatud just piiratud ressurssidega seadmetele nagu mikrokontrollerid. Kuigi freeRTOS ei oma täielikku RTOS süsteemi funktsionaalsust, sellest piisab et reaalajasüsteemi imiteerida. Enda programmikoodis käib kogu seadme funktsioneerimine läbi freeRTOS'i perioodiliste ja mitteperioodiliste ülesannete (*task*) kaudu, harilik *while* tsükkel on tühi. Selline lähenemine võimaldab freeRTOS-l endal valida täidetav ülesanne, mida hakatakse vastavalt ülesannetele antud prioriteedile täitma. Nii ei pea kõik funktsioonid ühte *while* tsüklisse panema ning saab ka valida funktsioonide täitmise prioriteete. Lisaks enda loodud freeRTOS ülesannetele, loovad mitmed teegid enda toimimiseks sisemisi ülesandeid nagu näiteks Wi-Fi ja veebiserveri ülesanded. Need ülesanded käivitatakse automaatselt teekide initsialiseerimisel ning ei suhtle otse enda töö käigus ei energiamõõturi programmikoodiga. Enne kui kasutaja saab energiamõõturit kasutama hakata, tuleb teostada seadme kõikide põhisüsteemide seadistamine, mis toimub Joonis 4.1 järgi. Esialgu seadistatakse mõõtekivi reset ja relee väljundi pinnid, initsialiseeritakse I2C kuvar ja kui programmikoodis on silumise muutuja tõene, käivitab mikrokontroller ka UART andmeside arvutiga. Kuna relee väljund ja UART TX

signaal on ühendatud sama pinni külge, ei ole võimalik mõlemat funktsiooni korraga kasutada. See pole probleem, kuna lõplikus versioonis pole UART kasutuses ning saan mureta kasutada pinni relee jaoks. Edasi loeb mikrokontroller EEPROM mälust kalibreerimise väärtused ning ka salvestatud elektrienergia väärtused. Kui seade läheb esimest korda käima, siis üldjuhul pole ei kalibreerimise ega tarbitud elektrienergia faile olemas ning sellisel juhul loob programm ise tühjad failid.




Joonis 4.1 Energiamõõturi käivitamise algoritm

Kui kõik eelnevad funktsioonid edukalt käima lähevad, käivitab ESP32 Wi-Fi teegi nimega *ESPAsyncWiFiManager* [27]. See teek lisab tavalisele Wi-Fi funktsionaalsusele ka *captive portal*'i, mis seadme esimesel sisselülitusel võimaldab kasutajal ühendada mikrokontrolleriga ning sisestada enda koduvõrgu SSID ja parooli, et energiamõõtur saaks kasutaja koduvõrgule ligi. Selline lahendus on palju parem harilikust Wi-Fi lahendusest, kuna nii ei pea kindla võrgu SSID ja parooli väärtused lisama programmikoodi, kogu koodi uuesti kompileerima ning seadmesse üles laadima. Võib-olla prototüübi jaoks sobiks lihtne lahendus, kuid *captive portal* lahendus on ilmselgelt paindlikum ja kasutajasõbralikum ning otsustasin selle lisada enda seadmesse.

Kui Wi-Fi ühendus on edukalt loodud, järgmise sammuna käivitatakse ja seadistatakse OTA (*Over the Air*) server juhtmevaba uuenduste jaoks, milleks kasutan teeki nimega ArudinoOTA [28]. Selle asemel, et uut programmikoodi mikrokontrollerisse laadida üle UART-i, võimaldab ArduinoOTA teostada programmikoodi uuendust üle Wi-Fi. Ainuke piirang OTA kasutamisel on see, et esimene kord tuleb programmikood üles laadida läbi UART-i, kuna üldjuhul pole sisse ostetud mikrokontrolleritel sobivat

programmikoodi peal. Peale esimest üleslaadimist ja Wi-Fi võrku ühendamist on võimalik seadmele juhtmevabalt ligi pääseda ning järgnevaid uuendusi laadida üles üle võrgu. Lisaks paremale kasutajasõbralikkusele on uuendamine läbi ArduinoOTA ka kiirem kui üle UART-i. Kui energiamõõtur täielikult kokku panna, pole ohutuse huvides UART enam kätte saadav ja ainuke viis tarkvara uuendamiseks ongi üle Wi-Fi.

Peale OTA serveri üles seadmist paneb mikrokontroller käima veebilehe. Selleks, et veebileht saaks suhelda mikrokontrolleriga, tuleb seadistada suhtlus mikrokontrolleri ja veebiserveri vahel. Kui kasutajalt tuleb päring, saadab veebiserver päringu edasi mikrokontrollerile, mis vastavalt programmikoodile täidab etteantud käsud. Näiteks kui kasutaja avab veebibrauseris seadme IP-aadressi, saab veebiserver päringu kätte ning suunab selle edasi mikrokontrollerile aadressil „/". Kui mikrokontroller saab päringu kätte, saadab mikrokontroller päringule vastu mällu salvesatud veebilehe „/index.html". Iga unikaalse päringu jaoks tuleb mikrokontrolleri koodi programmeerida vastus või funktsioon, mida mikrokontroller peaks päringu saamisel täitma.

Selline suhtlus kasutaja ja mikrokontrolleri vahel läbi veebiserveri võib käia mõlemat pidi. Sarnaselt nagu kasutaja veebibrauseris saatis päringud veebiserverile, saadab ka mikrokontroller mõõdetuid andmed veebilehele edasi kasutades päringuid, kuid nüüd lisab mikrokontroller päringule kaasa ka JSON formaadis mõõtetulemused. Mis on JSON formaat ja miks ma sellist lahendust kasutasin, saab rohkem teada punktis 4.2.2.

Viimase sammuna katsetab mikrokontroller SPI andmesidet, initsialiseerib mõõtekivi ja käivitab mõõtekivi mõõtmise režiimi. Selleks hoiab mikrokontroller mõõtekivi reset pinni madalana 100 ms ja peale kõrgeks viimist katsetab mikrokontroller SPI andmevahetust, lugedes mõõtekivi versiooni registrit. Kui loetud registri väärtuseks tuleb 0x0009153A, siis andmeside on õigesti seadistatud ning võib minna edasi initsialiseerimisega. Kui loetud tulemuseks on mõni muu väärtus, siis järelikult on SPI andmesiinid valesti ühendatud või mikrokontroller pole laiendusplaadile paigaldatud. Sellise olukorra puhul kuvab programmikood kasutajale veateate ning ei lase mõõtekivi edasi initsialiseerida. See funktsioon oli väga kasulik esimeste trükkplaatide versioonide silumisel, kuna nendel esines mitmeid vigu SPI ühendustes. Järgmise sammuna kirjutab mikrokontroller mõõtekivi registritesse erinevad seadeväärtused, et mõõtekivi saaks hakata mõõtmisi tegema. Registrid ja sinna kirjutatavad väärtused sain mõõtekivi andmelehest ja laiendusplaadi teegist. Eraldi ühtegi registri väärtusi muutma ei pidanud, kuna laiendusplaat oli ka seadistatud mõõtma vahelduvvoolu nimisagedusel 50 Hz ja nimipingel 230 V. Peale seadistamist teostab mõõtekivi edaspidiseid mõõtmisi iseseivalt ja ainuke muudetav register on kõrgpääsfiltri register, mille välja lülitamisel võimaldab energiamõõturil mõõta ka alalisvoolu.

Sellega on energiamõõturi põhisüsteemid seadistatud ning valmis järgmiseks sammuks.

### 4.1.1 Energiamõõturi operatsioonisüsteemi ülesanded

Peale energiamõõturi põhisüsteemide seadistamist käivitab mikrokontrolleri operatsioonisüsteem FreeRTOS viis paralleelset protsessi: *Energy, Uptime, Send2DB, OTA ja OLED*. Iga ülesanne vastutab enda osa eest, mida kinda perioodi tagant ja prioriteediga hakatakse täitma (vaata Tabel 4.1). Mida kõrgem on prioriteedi number, seda tähtsam on ülesanne ning kiiremini täitma hakatakse.

Tabel 4.1 RTOS ülesannete nimetused, prioriteedid ja täitmisintervallid

| RTOS ülesanne | Prioriteet | Täitmisintervall |
|---|---|---|
| *Energy* | 1 | 60 s |
| *Uptime* | 1 | 30 s |
| *Send2DB* | 1 | 5 s |
| *OTA* | 1 | 0,1 s |
| *OLED* | 1 | 5 s |
| *Calibrate* | 1 | täidetakse üks kord |
| *Waveform* | 5 | 15 s |

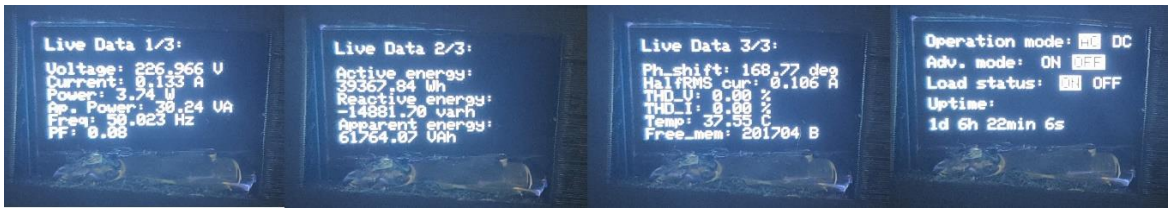
*Energy* ülesanne loeb iga 30 sekundi tagant mikrokontrolleris salvestatud energia muutujate väärtused ja salvestab need EEPROM-i. Niimoodi ei kao seadme väljalülitusel andmed tarbitud elektrienergia kohta.

*Uptime* ülesanne on lühike ülesanne, mis kaks korda minutis saadab veebilehele millisekundite arvu seadme sisselülitusest. Nii on võimalik veebilehele kuvada, kaua on seade järjest töötanud.

*Send2DB* ülesanne loeb iga viie sekundi tagant mõõtekiviga mõõdetud elektrivõrgu parameetreid ning saadab need edasi nii veebilehele kuvamiseks kui ka soovi korral välisesse andmebaasi. Elektrivõrgu parameetrite kätte saamiseks mõõtekivist, tuleb üle SPI lugeda vastava elektrivõrgu parameetri registri väärtus ja teisendada mõõdetud väärtus õigeks ühikuks. Näiteks pinge ruutkeskmise saamiseks tuleb lugeda registrit nimega AVRMS aadressil 0x202 ning registrist saadud arv korrutada kalibreerimiskonstantiga, et tulemuseks saada voldid. Iga parameeter tuleb läbi korrutada unikaalse konstantiga, et saadud tulemus näitaks õiget väärtust. Oma koodis kasutasin esialgu ADE9153A laiendusplaadi teegi konstante, kuid need sõltuvad trükkplaadi disainist ja andmelehest saadud arvudel ning täpse mõõtetulemuse saavutamiseks tuleb igale trükkplaadi versioonile leida uued konstandid. Konstantide leidmist arutan lähemalt alapeatükis 5.1.

*OTA* ülesanne on vajalik ArduinoOTA serveri tööl hoidmiseks ning uuenduste vastu võtmiseks. Põhimõtteliselt oleks saanud kasutada mõnda teist ülesannet, kuid otsustasin luua eraldi ülesande, kuna kõik muud perioodilised ülesanded on perioodiga üle viie sekundi ning katsetamise käigus tekitas nii pikk vahe probleeme uuenduste vastu võtmisel.

*OLED* ülesanne tegeleb kuvarile andmete kuvamisega kasutades Adafruit SSD1306 [29] ja GFX [30] teeke. Kuna kuvar on üpris väike (128x64 pikslit), ei mahu kuvarile korraga kõik andmed ning seega pidin jaotama andmed nelja lehe vahel. Esimesel lehel on mõõdetud pinge, vool, võimsus, näivvõimsus, sagedus ja võimsustegur. Teisel lehel on aktiiv-, reaktiiv- ja näivenergia. Kolmandal lehel faasinihe, poolperioodi vool, voolu ja pinge THD, temperatuur ja vaba mälu kogus. Viimasel lehel saab näha aktiivset mõõtmisrežiimi (AC või DC), lisafunktsioonide staatus (sees või väljas), koormus sisse- või väljalülitatud ning seadme tööaega (vaata Joonis 4.2).




Joonis 4.2 OLED kuvari neli lehte andmetega

Lisaks viiele pidevalt toimivale perioodilisele ülesandele on programmikoodis olemas veel kaks mitteperioodilist freeRTOS ülesannet: *Calibrate* ja *Waveform*.

*Calibrate* ülesanne on kasutaja päringu peale käivitatud mitteperioodiline freeRTOS ülesanne, mis käivitab mõõtekivisse sisseehitatud isekalibreerimise funktsiooni, mis leiab uued voolu ja pinge kalibreerimiskonstandid. Nende väärtuste põhjal saab mikrokontroller välja arvutada, kui palju peab pinge ja voolu väärtusi kompenseerima täpse tulemuse saavutamiseks. Parima isekalibreerimise tulemuse saavutamiseks tuleks seade elektrivõrgust kalibreerimise ajaks lahti ühendada. Loomulikult saab ka elektrivõrku ühendatud seadmele teha isekalibreerimist, kuid sellisel juhul tuleb pingekanali isekalibreerimise mõõtemääramatus märgatavalt kõrgem. Voolukanali määramatust elektrivõrgus olemine peaaegu ei mõjuta, kuna isekalibreerimise käigus saab ühendatud koormused lahti ühendada ning nii mõõdetavas kanalis muutusi ei esine. Seadme katsetamisel sain pingekanali mõõtemääramatuseks elektrivõrku ühendatuna 5-10 korda kõrgema (keskmiselt 1-2,5%) kui seda eraldi toite puhul (keskmiselt 0,25%). Voolukanali isekalibreerimisel oli määramatuseks mõlemal juhul 0,2%.

Viimane ja mahult suurim RTOS ülesanne on *Waveform* ülesanne. Nagu ka *Calibrate* ülesande puhul, läheb see ülesanne käima ainult kasutaja päringu peale, kuid erineb selle poolest, et see ülesanne jääb töötama kuni kasutaja annab päringu *Waveform* ülesande lõpetada. Nagu ka ülesande nimi ütleb, tegeleb see ülesanne pinge ja voolu lainekuju mõõtmisega ning teostab ka mõõdetud andmepunktide põhjal pingele ja voolule FFT-d (kasutades Robin Scheibler'i FFT teeki [31]) ning leiab THD väärtuse. ADE9153 mõõtekivi andmelehest lugesin, et mõõtekivi uuendab töödeldud lainekuju registreid sagedusel 4000 Hz, seega seadsin mikrokontrolleri samal sagedusel ka lainekuju väärtusi pärima ning mällu salvestama. Kui andmepunktide puhver on täis loetud, teisendab mikrokontroller mõõtekivilt mõõdetud väärtused vastavalt voltideks või ampriteks kasutades eelnevalt mõõdetud kalibreerimisväärtusi. Teisendatud lainekuju väärtused saadetakse ka kohe veebilehele kasutajale kuvamiseks.

### 4.1.2 FFT seadistamine ja mikrokontrollerist tulenevad piirangud

FFT tegemiseks on vaja lisaks diskreetimissagedusele ka andmepunktide jada pikkust *N*. Kui diskreetimissagedusest sõltus kõrgeim mõõdetav harmooniku järk, siis andmepunktide jada pikkus määras FFT lahutusvõime, s.t. kui suur sageduse vahemik jääb kahe punkti vahele. Mida kõrgem on diskreetimissagedus ja andmepunktide jada pikkus, seda parem on FFT tulemus, kuid ka suuremat mälu ja arvutuslikku ressursi läheb vaja. Energiamõõturi puhul on piiravaks faktoriks mikrokontrolleri võimekus ja täpsemalt andmete edastamine veebilehele. ESP32 suudab edukalt teha FFT andmepunktide jadale pikkusega $N = 4096$, kuid sealjuures kasutades üle poole mikrokontrolleri muutmälust. Probleem tuleb ette FFT tulemuste saatmisega veebilehele. Niipea kui edastatud andmete kogus läks üle $N = 512$ andmepunkti, pole veebilehel piisavalt mälu andmete vastuvõtmiseks ja osa andmetest ei jõua veebilehele kohale, kuid programm töötab edasi. Kui tõsta andmepunktide kogust $N = 2048$ punktini, ei jõua mikrokontroller kõiki andmeid vastu võtta ja jookseb täielikult kokku. Nende piirangute tõttu valisin FFT andmepuntide arvuks $N = 1024$, kuid veebilehele andmete kuvamiseks saadan ainult esimesed 512 andmepunkti. Nii saavutasin kõrgema FFT lahutusvõime, kuid see-eest kaotasin kõrgemate sageduste harmoonikud. Punktis 4.2.2 arutan võimalikust lahendusest, kuidas 512 andepunkti piirangust ümber saada.

Kui FFT on tehtud, on võimalik suhteliselt lihtsalt välja arvutada pingele ja voolule ka THD väärtus. THD leidmiseks tuleb leida esialgu põhisagedus, mida saab leida käies läbi kõik sagedustasandi punktid ning valida kõige kõrgema amplituudiga sagedus. Kuigi mõnedes olukordades ei pruugi kõrgeim amplituud tähendada alati põhisagedust,

sobib selline lihtsustus elektrivõrgu THD arvutamiseks, kuna põhisagedus on teada ja ajas ei muutu. Leitud põhisageduse põhjal saab mikrokontroller harmoonikute otsimist märgatavalt lihtsustada, leides oodatavad harmoonikute sageduste väärtused ning arvutades ruutkeskmine kõikide harmoonikute amplituudidest. Edasi leiab mikrokontroller THD väärtuse valemi 2.13 põhjal. Siin tuleb välja tuua üks oluline märkus, et mõõtmiste ebatäpsuste tõttu võib arvutatud harmooniku sagedus natuke erineda tegelikust sagedusest. Sellise olukord tuleb kindlasti arvesse võtta ning seetõttu lisasin harmooniku amplituudi leidmise funktsioonile juurde ka ümbruse amplituudi võrdlemise. Mikrokontroller võrdleb 3 amplituudväärtust enne ja pärast arvutatud sagedust ning valib sealt suurima väärtuse. Nii saan olla kindel, et mikrokontroller sai õige harmooniku väärtuse kätte ning arvutatud THD väärtus ei tule vale.

## 4.2 Veebilehe programmikood

Nagu eelnevalt juba mainisin, veebilehe programmikood jaguneb veel omakorda kaheks eraldi osaks, kasutajapoolseks veebileheks ja veebilehe skriptideks. Mõlemal on omad kindlad täidetavad ülesanded, mis kokku moodustavad ühe terviku.

Oluline on siin välja tuua, et kuigi veebilehe programmikood laetakse mikrokontrolleri mälust ja tavatalitluses töötab mikrokontrolleri ressurssidel, pole veebilehe programmikoodil turvalisuse kaalutlustel otsest juurdepääsu mikrokontrolleri põhikoodis olevatele muutujatele ja üleüldse mikrokontrolleri mälule. Veebileht on seega peaaegu täielukult isoleeritud ülejäänud seadmest, mis on hea seadme turvalisuse jaoks, kuid märgatavalt raskendab suhtlust kahe programmiosa vahel.

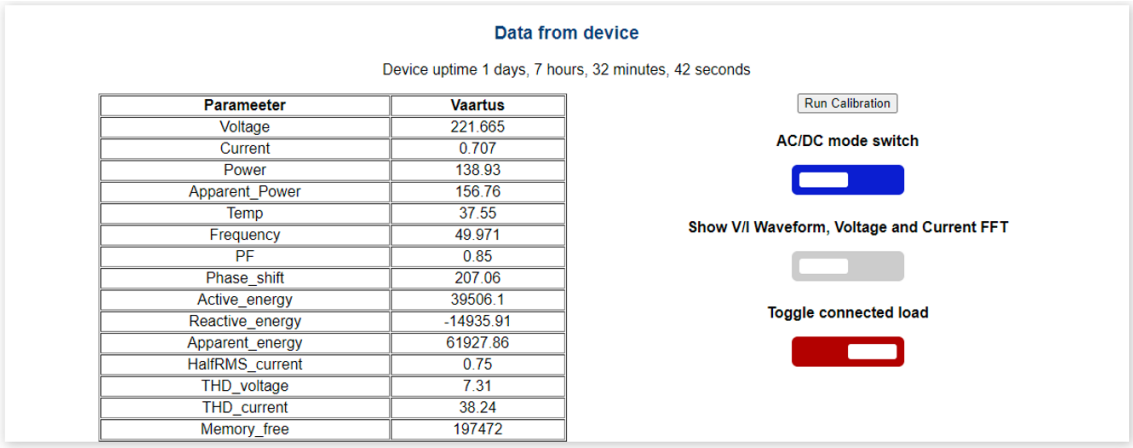
### 4.2.1 Kasutajapoolne veebileht

Kasutajapoolne veebileht vastutab põhiliselt kasutajaliidese eest. See võimaldab mikrokontrolleri poolt edastatud ja veebiserveri poolt vastuvõetud andmeid kuvada kasutajale ja lubab veebilehele integreeritud nuppudega juhtmevabalt mikrokontrollerit juhtida. Kuigi veebileht pole mahu poolest suur ega keerukas, pole see vähem oluline kui muud programmiosad. Just läbi veebilehe hakkab käima põhiline suhtlus seadme ja kasutaja vahel ning veebilehe disain määrab, kui kerge ja murevaba on seadme kasutamine.

Veebilehtede loomiseks ja kuvamiseks kasutatakse põhiliselt programmeerimiskeelt nimega HTML (*HyperText Markup Language*), mis võimaldab veebibrauseritel teisendada failis olevad elemendid ja tekst kasutajale nähtavaks veebileheks. Kuna mul ei olnud erivajadusi veebilehe loomiseks, otsustasin teha veebilehe HTML keeles.

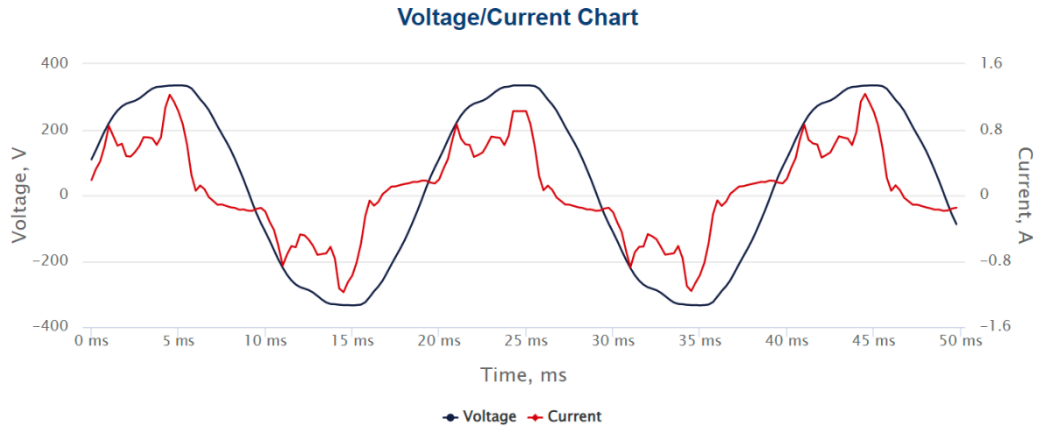
Veebilehe esimeseks elemendiks on <head> element, mis sisaldab endas veebilehe pealkirja, veebilehe kuvamise sätteid ja viiteid veebilehel kasutatavatele failidele ja skriptidele. See element on vajalik eelkõige veebilehe õigeks kuvamiseks veebibrauseris ja pole kasutajale nähtav. Teiseks elemendiks on <body> element, kus toimub veebilehe nähtava osa loomine kasutades , <table> ja <button> elemente. Nende elementide kombineerimisel jagasin veebilehe neljaks osaks.

Esimeses osas asub tabel mõõdetud elektriparameetritega koos seadet juhtivate nuppudega (vaata Joonis 4.3). Kasutades nelja nuppu on võimalik käivitada isekalibreerimine, vahetada mõõtmisrežiim vahelduvvoolult alalisvoolule ja vastupidi, lülitada sisse lainekuju mõõtmise ja FFT ning lülitada sisse-välja ühendatud koormus.

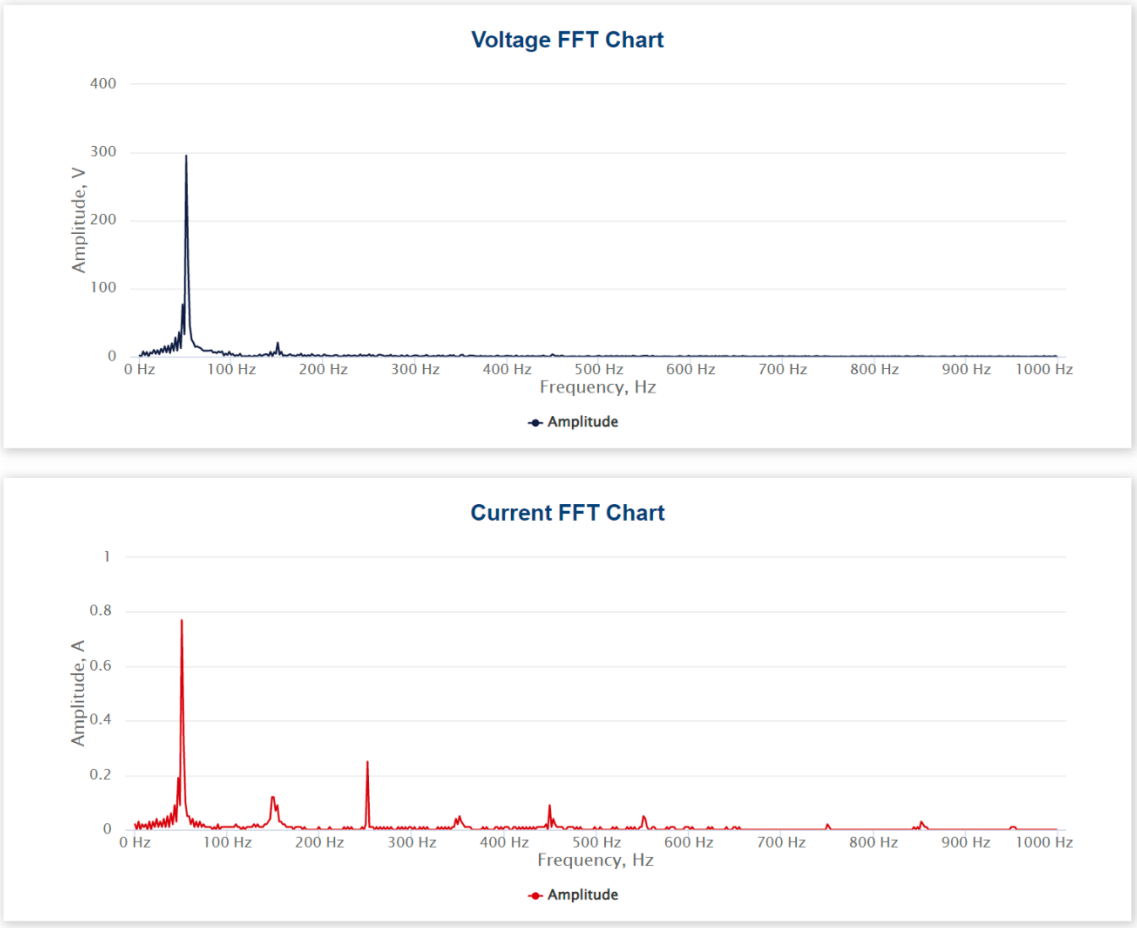



Joonis 4.3 HTML element tabeliga mõõdetud andmega ja nuppudega juhtimiseks

Ülejäänud kolmes osas asuvad graafikud pinge-voolu lainekuju (vaata Joonis 4.4) ning pinge ja voolu FFT-ga (vaata Joonis 4.5). Need graafikud on veebilehe laadimisel tühjad ning graafikud joonistatakse ainult siis, kui vastav nupp on vajutatud esimeses osas ning mikrokontroller on saatnud andmed kuvamiseks veebilehe skriptile.




Joonis 4.4 Veebilehe graafik pinge ja voolu lainekujuga




Joonis 4.5 Veebilehe graafikud pinge ja voolu FFT-ga

### 4.2.2 Veebilehe skript

Veebilehe skripti põhiliseks ülesandeks on mikrokontrollerile andmete saatmine ning vastu võtmine ja mikrokontrollerilt saadud andmete põhjal voolu ja pinge lainekuju ning ka FFT graafiku joonestamine. Lisaks sellele loob skript tabeli elektrienergia parameetritega ja haldab nuppe seadme juhtimiseks nagu koormuse sisse-välja lülitamine, kalibreerimine ja AC/DC mõõterežiimi valik.

Veebilehe skript põhineb programmeerimiskeelel nimega JavaScript, mis võimaldab dünaamiliselt luua, uuendada ja kustutada veebilehe elemente ning andmeid. Tänu JS-le ei pea kasutaja uute andmete saamiseks iga kord veebilehte uuendama ega nuppu veebilehel vajutama. Veel suureks eeliseks on laialdane teekide olemasolu erinevate funktsioonide täitmiseks, nagu näiteks graafikute joonestamine ning andmebaasidega suhtlemiseks.

Veebilehe skripti jaguneb koodi poolelt kaheks osaks: Andmete vastu võtmine, päringute saatmine mikrokontrollerile.

Esimeses skripti osas toimub mikrokontrollerilt JSON formaadis andmete vastu võtmine, töötlemine ja veebilehel andmete uuendamine. JavaScript Object Notation on lihtne universaalne andmeedastusformaat, mida on kerge erinevate programmeerimiskeelte vahel vahendada. Lisaks sellele on JSON lihtsasti genereeritav ning masinate poolt loetav, mis teeb JSON-i hea andmevahetusformaadi andmete saatmiseks serveri ja veebilehe vahel [32]. Eelmainitud põhjuste pärast kasutan JSON't andmete saatmiseks mikrokontrollerilt veebilehele.

Mikrokontroller saadab koos mõõdetud andmetega välja ka sõnumi, kus on kirjas, millise parameetri andmed olid JSON-na saadetud. Ilma selle sõnumita ei osaks skript aru saada, mida nende andmetega peale hakata. Näiteks tabelisse lähevad ainult elektrienergia parameetrite andmed , sest lainekuju punktidel pole tabelis midagi teha. Seega vastavalt saadetud sõnumile teeb veebiskript andmetega ühe järgmistest tegevustest:

- Uuenda tabel uute andemetega
- Uuenda pinge lainekuju
- Uuenda voolu lainekuju
- Uuenda pinge FFT graafik
- Uuenda voolu FFT graafik

Ma otsustasin teha iga lainekuju uuendamise eraldi sõnumina, kuna mikrokontrollerist tuleneva piirangu tõttu ei ole võimalik saata korraga üle 512 JSON formaadis andmepunkti edasi veebilehele. Niipea kui andmepunktide arv oli suurem, muutus veebileht ebastabiilseks ning graafikute kuvamine enam ei töötanud korrektselt. Lisaks eelmainitule oli eraldi saadetuid andmepakke märgatavalt lihtsam skriptis töödelda. Üheks viisiks ületada 512 andmepunkti limiit on saata ühe graafiku andmed mitme paketina s.t. kuvada esialgu esimesed 512 andmepunkti ja seejärel saata näiteks veel 512 andmepunkti. Selline lahendus oleks suhteliselt lihtne lisada olemasolevale koodile, kuid vajaks põhjalikku katsetamist tagamaks selle lahenduse stabiilsust. Eelmainitud põhjuste pärast otsustasin hetkel piirduda 512 andmepunktiga ning uurida eelmainitud lahendust energiamõõturi järgmises versioonis.

Graafikute joonestamiseks kasutasin teeki nimega Highcharts [33], mis tegi graafikute loomise ning vormistamise märgatavalt lihtsamaks. Minu poolt oli vaja ainult seadistada ja nimetada graafikute teljed ning lisada andmepunktid graafikutele. Kogu ülejäänud töö teeb Highcharts teek ära. Selle teegi poolt loodavad graafikud on interaktiivsed ning võimaldavad kasutajal iga andmepunkti väärtust kuvada.

Teises skripti osas võtab JS kasutajalt vastu erinevaid päringuid ning teisendab need mikrokontrollerile vastuvõetavasse formaati. Vastavalt kasutaja tegevusele saadab skript päringu kindlale veebiserveri aadressile. Nendeks kolmeks aadressiks on „/", „/calib" ja „/update". Esimese kahe aadressi puhul ei lisa skript päringule midagi juurde, kuna esimese päringu korral soovib kasutaja laadida veebilehe ning teise puhul käivitatakse seadme isekalibreerimine. Kolmandat aadressi kasutab veebilehe skript laialdaselt nuppude vajutuse edastamist mikrokontrollerile. Kasutaja vajutusel nupule käivitab skript funktsiooni, mis loeb vajutatud nupu nime ning staatust (kas nupp aktiivne või mitte) ja saadab järgneva päringu:

„/update?output=element.id&state=element.checked", kus *element.id* on vajuatud nupu nimi ja *element.checked* on nupu saatus. Võimalikud *element.id* väärtused on *Waveb*, *Relayb* ja *Modeb* ja *element.chekced* on 1 või 0. Näiteks relee väljalülitamisel on saadetud päringuks /update?output=Relayb&state=0.

# 5 ENERGIAMÕÕTURI TÄPSUSE LEIDMINE JA SEATUD SPETSIFIKATSIOONIDE SAAVUTAMINE

## 5.1 Energiamõõturi täpsuse leidmine

### 5.1.1 Energiamõõturi kalibreerimine

Enne kui saab täiesti uut energiamõõturi versiooni kasutusele võtta, tuleb teostada esialgne ADE9153 registrite kalibreerimine. Kui esimest korda lugeda mõõtekivi elektriparameetrite registreid, tuleb tulemuseks arv, mis selgelt ei näita õiget väärtust. Näiteks 230 V asemel tuleb pinge efektiivväärtuse registri tulemuseks väärtus 17150100. Kui on teada mõõdetava parameetri väärtus, siis on võimaik leida konstant, mis võimaldaks teisendada mõõtekivi registri tulemus õigeks ühikuks kasutades valemit 5.1.

$$c = \frac{val}{reg} \tag{5.1}$$

kus $c$ – kontsant registri tulemuse teisendamiseks, $\frac{V}{\text{üh}}, \frac{A}{\text{üh}}, \frac{W}{\text{üh}}$,

*val* – mõõdetava parameetri teadaolev väärtus, V, A või W,

*reg* – mõõtekivist loetud registri väärtus, üh.

ADE9153 mõõtekivi puhul on vaja kalibreerida 3 kalibreerimisregistrit: pinge, vool, ja võimsus. Ülejäänud väärtused suudab mõõtekivi leida ise nende registrite põhjal. Siin on oluline välja tuua, et need konstandid on unikaalsed iga trükkplaadi versiooni jaoks ning tuleb igale versioonile uuesti leida täpse mõõtmistulemuse saavutamiseks. Kui konstandid on juba korra leitud, on võimalik kasutada mõõtekivi isekalibreerimise funktsiooni teise samasuguse seadme kalibreerimiseks. Kuna minu poolt ehitatud energiamõõtur oma ehituselt sarnane Analog Device'i ADE9153 mõõtekivi laiendusplaadile, võtsin lähtepunktis laiendusplaadi konstandid.

Konstantide ja mõõturi mõõtevea leidmiseks otsustasin läbi viia neli katset, mille käigus mõõtsin ja võrdlesin energiamõõturi täpsust enne ja pärast registrite kalibreerimist nelja erineva vahelduvpinge juures (100V, 150V, 200V, 230V). Kaks katset tegin ilma lisatud koormuseta ning kaks 650W koormusega. Erineva vahelduvpinge genereerimiseks kasutasin Keysight AC/AC AC6802A toitplokki, mis võimaldab luua vahelduvvoolu vahemikus 0 – 310 VAC. Pinge ja voolu mõõtmiseks kasutasin Keysight 34461A multimeetrit ja võimsust mõõtsin toiteploki kuvarilt.

Selleks, et katsete tulemuste põhjal leida mõõteviga, kasutasin mõõtevea leidmiseks valemit 5.2. Leitud mõõtevea põhjal on võimalik hinnata energiamõõturi täpsust ning palju tuleb konstante muuta õige mõõtetulemuse saavutamiseks.

$$\Delta x = \frac{x_{seade} - x_{referents}}{x_{referents}} \cdot 100\% \qquad (5.2)$$

kus $\Delta x$ –mõõteviga,% ,

$x_{seade}$ – energiamõõturilt mõõdetud väärtus, V, A või W,

$x_{referents}$ – referentsseadmelt mõõdetud väärtus, V, A või W.

Peale esimese kahe katse läbiviimist ja tulemusi analüüsides sain teada, et isegi kasutades ADE9153 laiendusplaadi lähtekonstante jäi energiamõõturi parameetrite täpsus vahemikku ±5% (vaata Tabel 5.1). Kõige suurem mõõteviga oli võimsuse, milleks oli 3,87% pingel 200V. See tähendab, et spetsifikatsioonides seatud täpsus oli saavutatud ning paremat täpsust pole vaja saavutada. Sellegipoolest on selline mõõteviga küllaltki suur ning on lihtsalt parandatav konstantide muutmisega.

Tabel 5.1 Energiamõõturi mõõtevead enne kalibreerimist

| Seadepinge, V | Pinge erinevus, % | Voolu erinevus, % | Võimsuse erinevus, % |
|---|---|---|---|
| 100 | 1,93 | 1,96 | 3,21 |
| 150 | 1,96 | 2,43 | 3,62 |
| 200 | 1,98 | 2,34 | 3,87 |
| 230 | 1,99 | 2,28 | 3,68 |

Sättides kolme kontsanti ja viies läbi ülejäänud kaks katset oli tulemus märgatavalt parem. Nii voolu kui ka pinge väärtused ei erinenud rohkem kui 0,1% (vaata Tabel 5.2) ning suurim erinevus oli uuesti võimsusel, milleks oli -0,66% pingel 100V. Selline tulemus tuli mulle suure üllatusena ning edasise uurimisena tuleks teostada võimsuse kalibreerimine täpsema mõõteriistaga, kuna toiteploki võimsuse mõõtemääramatus ei võimalda korrektselt hinnata energiamõõturi mõõteviga.

Tabel 5.2 Energiamõõturi mõõtevead pärast kalibreerimist

| Seadepinge, V | Pinge erinevus, % | Voolu erinevus, % | Võimsuse erinevus, % |
|---|---|---|---|
| 100 | -0,08 | 0,03 | -0,66 |
| 150 | -0,03 | 0,02 | 0,59 |
| 200 | 0,01 | 0,00 | 0,11 |
| 230 | 0,00 | 0,00 | 0,12 |

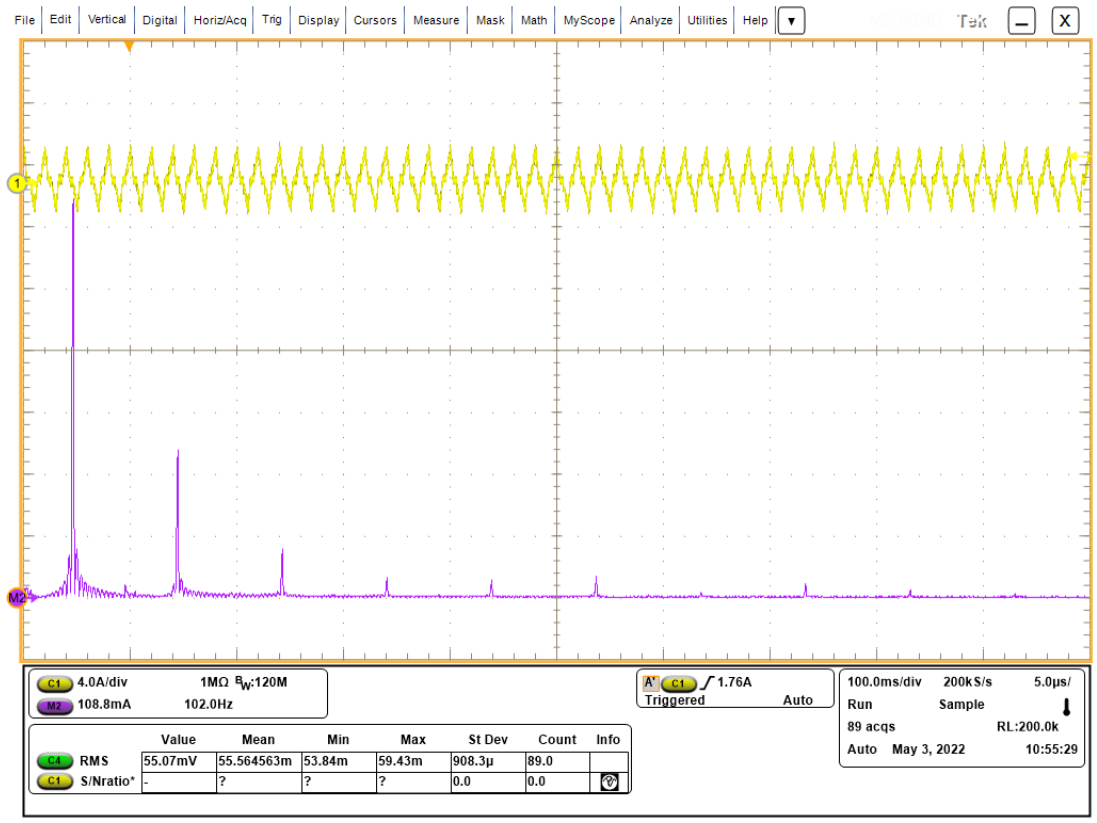
### 5.1.2 Voolu moonutusteguri täpsus

Voolu moonutusteguri täpsuse leidmiseks viisin läbi veel kaks katset, kus ostsiloskoobiga mõõtsin ühendatud koormuse voolu lainekuju ning seadsin ostsiloskoobi Fourier' teisendust teostama. Fourier' teisenduse graafikult panin kirja harmoonikute amplituudväärtused, et valemi 2.13 abil leida voolu moonutustegur. Ühe katse tegin koormusena sülearvuti laadijaga ja teise katse elektritrelliga.

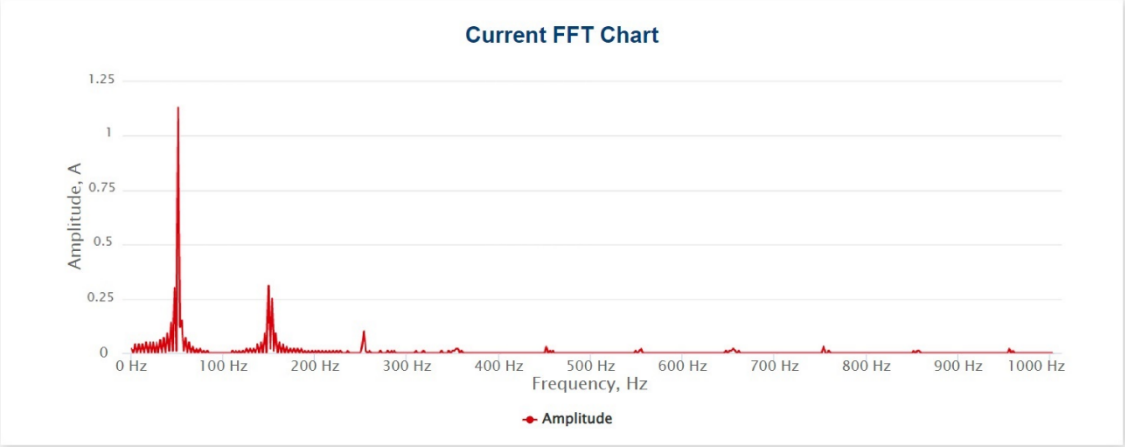
Ostsiloskoobilt saadud Fourier' teisenduse graafiku ja THD väärtust võrdlemisel energiamõõturi tulemusega tuli välja, et energiamõõturi tulemuse suurusjärk on õige (vaata Tabel 5.3), kuid madala FFT punktide arvu tõttu pole harmoonikud Fourier' teisenduse graafikul nii kõrged kui ostsiloskoobi graafikul (vaata Joonis 5.1 ja Joonis 5.2). Seega tuleb energiamõõturi arvutatud moonutusteguri väärtus märksa madalam kui on see ostsiloskoobil.

Tabel 5.3 Voolu moonutusteguri katsete tulemused

| Mõõdetav koormus | Seadmega leitud THD väärtus, % | Ostsiloskoobiga leitud THD väärtus, % |
|---|---|---|
| Sülearvuti laadija | 32,5 | 40,1 |
| Elektritrell | 10,8 | 12,7 |




Joonis 5.1 Ostsiloskoobiga saadud sülearvuti laadija Fourier' teisenduse graafik




Joonis 5.2 Energiamõõturiga saadud sülearvuti laadija Fourier' teisenduse graafik

Kuna energiamõõturi THD tulemus märgatavalt erines ostsiloskoobi tulemusest, oleks vaja veel uurida võimalikke põhjusi, miks tulemus niivõrd erinev tuli ning kuidas oleks võimalik THD tulemust parandada

## 5.2 Seatud spetsifikatsioonide saavutamine

Alapeatükis 3.1 seadsin spetsifikatsioonid projekteeritavale energiamõõturile ja peale seadme valmisehitamist ning katsete läbiviimist, on võimalik vaadata spetsifikatsioonide täitmist.

Seadme omahinna arvutamiseks liitsin kokku kõik energiamõõturis kasutatud komponentide hinnad kokku, mille tulemuseks sain energiamõõturi omahinnaks 31,71€ (vaata Lisa 2). Kolm kallimat komponenti olid AC/DC toiteplokk, ESP32 mikrokontroller ja ADE9153 mõõtekivi, mis moodustasid ~78% seadme koguhinnast. Vastavalt Lisas 2 toodud tabelile, oli seadme omahinna spetsifikatsioon saavutatud.

Vastavalt Joonis 4.3, Joonis 4.4 ja Joonis 4.5 on näha, et kõik spetsifikatsioonis määratud mõõtmisfunktsionaalsus on seadmes olemas ning on ka võimalus andmetele juhtmevabalt ligi pääseda. Lisaks kasutades OLED kuvarit on võimalik ka andmeid otse seadmelt kuvada. Sellega on andmete kuvamise spetsifikatsioon täidetud.

Seadme toite spetsifikatsioon normaaltalitluses saavutasin kasutades AC/DC toiteplokki ning iseseisev töötamine kasutades 1S 450mAh Lipo akut.

Spetsifikatsioonis seatud täpsus oli saavutatud peale katsete läbiviimist ning mõõtevigade arvutamist (vaata Tabel 5.1 ja Tabel 5.2). Spetsifikatsiooni täpsuse suutis seade saavutada isegi enne kalibreerimist ning peale kalibreerimist oli seadme mõõteviga märgatavalt madalam.

# KOKKUVÕTE

Käesoleva bakalaurusetöö käigus projekteerisin, ehitasin ja kirjutasin tarkvara nutikale energiamõõturile, millega on võimalik mõõta pinget, voolu, võimsust, sagedust, võimsustegurit, energiat, pinge ning voolu lainekuju, FFT ja moonutustegurit.

Töö esimeses osas analüüsisin turul olemasolevaid seadmeid, et saada ülevaade turu hetkeseisundist ning seada projekteeritavale seadmele konkurentsivõimelised spetsifikatsioonid. Edasi uurisin, kuidas toimub digitaalsetes sardsüsteemides elektrivõrgu parameetrite mõõtmine.

Järgmise sammuna seadsin eelmise osa põhjal projekteeritavale seadmele spetsifikatsioonid ning selle põhjal valisin riistvara komponendid nutikale energiamõõturile. Vahelduvvoolu mõõtmiste teostamiseks valisin ADE9153 mõõtekivi ja mikrokontrolleriks ESP32.

Peale põhiliste komponentide valikut, disainisin nende kahe komponendi ümber trükkplaadi koos kõikide muude toetavate komponentidega. Kokku pidin tegema neli trükkplaadi versiooni, kuna igal trükkplaadi versioonil esinesid mitmed suured vead, mis polnud kergelt parandatavad. Viimasel neljandal trükkplaadi versioonil vigu ei esinenud ning spetsifikatsioonides seatud vajalik funktsionaalsus olemas.

Neljandas peatükis kirjeldasin lähemalt energiamõõturi tarkvara osa, kus tarkvara jaguneb kaheks osaks: mikrokontrolleri ja veebilehe programmikood. Mikrokontrolleri osa vastutab andmevahetusega mõõtekiviga, mõõdetud andmete töötlemisega (FFT ja THD), andmete kuvamisega kuvarile, andmete salvestamisega ja saatmisega veebilehele. Veebilehe peamiseks ülesandeks on juhtmevabalt kasutajale andmete kuvamine ning seadme juhtimine.

Viimase sammuna viisin läbi mitmed katsed seadme pinge, voolu, võimsuse ja voolu moonutusteguri täpsuse mõõtmiseks. Pinge, voolu ja võimsuse katsete mõõtetulemused tulid üllatavalt head ning suurim mõõteviga peale seadme kalibreerimist oli 0,66%. Energiamõõturi voolu moonutusteguri katse tulemus oli õiges suurusjärgus, kuid märksa madalam ostsiloskoobiga saadud tulemusega, mistõttu tuleks võimalikke põhjusi edasi uurida.

Töö iga etapi käigus esines palju erinevaid probleeme, kuid nendest kõige rohkem avaldas mõju just trükkplaatidel esinevad vead. Trükkplaatidel vigade leidmiseks kulus palju aega ning nende parandamiseks tuli tellida uus trükkplaadi versioon, mis märgatavalt aeglustas energiamõõturi arendamist. Sellegipoolest sain kõik probleemid lahendatud, töö eesmärgid ning spetsifikatsioonid saavutatud.

# SUMMARY

In course of this bachelor's, I designed, built and wrote software for a Smart Energy Meter, which can measure voltage, current, power consumption, frequency, power factor, consumed energy, voltage and current waveform, FFT and total harmonic distortion.

In the first part of my thesis, I analyzed existing products on the market, to get an idea of current market state and set competitive specifications for my device. After that I researched ways for measuring electrical grid parameters in digital devices.

As next step, based on market analysis, I set specifications and chose hardware components for Smart Energy Meter. Electrical grid parameter measurement is performed by ADE9153 Energy Metering IC and for microcontroller I chose ESP32.

After two main components were chosen, I designed a printed circuit board around two main parts with all other supporting components. In total I had to create four PCB revisions, because of multiple big mistakes on each PCB revision, which were not easily fixable. Fourth and last PCB revision had no mistakes in it and met all set specifications.

In the fourth chapter, I go into details about software part of Smart Energy Meter. Software is divided into two main parts: microcontroller and website part. Microcontroller part is responsible for data transfer between Energy Metering IC, signal processing (FFT and THD), displaying data to screen, saving measured data and sending it to website. Main task of website part is to show measured data to user and let user control Smart Energy Meter.

Lastly, I performed experiments to determine Smart Power Meter measurement accuracy of voltage, current, power and current THD. Voltage, current and power experiment results were surprisingly good and largest measurement error was 0,66% when measuring power. Current THD experiment results were in correct order of magnitude, but lower than result measured by oscilloscope and possible reasons need to be studied further.

During each stage of thesis there were a lot of different issues, but the biggest influence were errors on PCB-s. Troubleshooting faults took quite a bit of time and only way to fix them was to create new PCB revision, which significantly slowed down development. Nonetheless all issues were fixed, thesis goals were completed and specifications achieved.

# KASUTATUD KIRJANDUSE LOETELU


[1] Nord Pool AS, [Võrgumaterjal]. Saadaval: https://www.nordpoolgroup.com/. [Kasutatud 12 03 2022].

[2] J. Järvik, Üldelektrotehnika, Tallinn: TTÜ Kirjastus, 2014.

[3] Noark Electric Europe, „Energy Meter Ex9EMS 1P 2M 100A 1T," [Võrgumaterjal]. Saadaval: https://www.noark-electric.eu/en/products/107291. [Kasutatud 13 03 2022].

[4] Hugo Brennenstuhl GmbH & Co Kommanditgesellschaft, „Primera-Line Wattage and current meter PM 231 E," [Võrgumaterjal]. Saadaval: https://www.brennenstuhl.com/en-DE/products/travel-adapters-adapter-plugs/primera-line-wattage-and-current-meter-pm-231-e. [Kasutatud 13 03 2022].

[5] emporia, „The Gen 2 Vue Energy Monitor," [Võrgumaterjal]. Saadaval: https://www.emporiaenergy.com/how-the-vue-energy-monitor-works. [Kasutatud 13 03 2022].

[6] Xiaomi, „Mi Smart Plug Wi-Fi," [Võrgumaterjal]. Saadaval: https://www.mi.com/us/mj-socket/. [Kasutatud 13 03 2022].

[7] Fluke Corporation, „Fluke 1770 Series Three-Phase Power Quality Analyzers," [Võrgumaterjal]. Saadaval: https://www.fluke.com/en/product/electrical-testing/power-quality/1773-1775-1777. [Kasutatud 13 03 2022].

[8] N. Kularatna, Digital and Analogue Instrumentation testing and measurement, London: The Institution of Electrical Engineers, 2003.

[9] R. Laaneots ja O. Mathiesen, Mõõtmise alused, Tallinn: Tallinna Tehnikaülikooli Kirjastus, 2002.

[10] Maxim Integrated Products, „Using Current Transformers with the 78M661x," Aprill 2010. [Võrgumaterjal]. Saadaval: https://pdfserv.maximintegrated.com/en/an/AN4841.pdf. [Kasutatud 05 03 2022].

[11] Standex Electronics, „Magnetic Sensing Technologies: Reed switches vs. Hall effect switches," [Võrgumaterjal]. Saadaval: https://standexelectronics.com/magnetic-sensing/magnetic-sensing-technologies-reed-switches-vs-hall-effect-switches/. [Kasutatud 14 03 2022].

[12] NK Technologies, „Current Sensing Theory," [Võrgumaterjal]. Saadaval: https://www.nktechnologies.com/engineering-resources/current-sensing-theory/. [Kasutatud 05 03 2022].

[13] Keysight Technologies, „What is a Rogowski Coil Current Probe?," 15 12 2017. [Võrgumaterjal]. Saadaval: https://www.rs-online.com/designspark/what-is-a-rogowski-coil-current-probe. [Kasutatud 13 03 2022].

[14] OpenEnergyMonitor, „An Introduction to AC Power," 14 11 2016. [Võrgumaterjal]. Saadaval: https://github.com/openenergymonitor/learn/blob/master/view/electricity-monitoring/ac-power-theory/introduction.md. [Kasutatud 10 03 2022].

[15] M. Inci, „Active/reactive energy control scheme for grid-connected fuel cell system with local inductive loads," *Energy,* kd. 197, 2020.

[16] C. E. Efstathiou, „Singal Sampling: Nyquist - Shannon Theorem," [Võrgumaterjal]. Saadaval: http://195.134.76.37/applets/AppletNyquist/Appl_Nyquist2.html. [Kasutatud 20 03 2022].

[17] R. Keim, „The Nyquist–Shannon Theorem: Understanding Sampled Systems," 06 05 2020. [Võrgumaterjal]. Saadaval:

https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/. [Kasutatud 20 03 2022].

[18] National Instruments Corporation, „Frequency Measurements: How-To Guide," 05 02 2021. [Võrgumaterjal]. Saadaval: https://www.ni.com/en-us/support/documentation/supplemental/21/frequency-measurements-how-to-guide.html. [Kasutatud 09 03 2021].

[19] NTI Audio, „Fast Fourier Transformation FFT - Basics," [Võrgumaterjal]. Saadaval: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft. [Kasutatud 26 03 2022].

[20] Gamry Instruments, Inc, „Total Harmonic Distortion: Theory and Practice," 2021. [Võrgumaterjal]. Saadaval: https://www.gamry.com/application-notes/EIS/total-harmonic-distortion/. [Kasutatud 26 03 2022].

[21] Espressif Systems, „Analog to Digital Converter," 2022. [Võrgumaterjal]. Saadaval: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html. [Kasutatud 27 03 2022].

[22] Analog Devices, „ADE9153A Arduino Shield Evaluation Board," [Võrgumaterjal]. Saadaval: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-ADE9153A.html. [Kasutatud 29 03 2022].

[23] Analog Devices Inc, „ADE9153A Technical Reference Manual," 2018. [Võrgumaterjal]. Saadaval: https://www.analog.com/media/en/technical-documentation/user-guides/ade9153a-technical-reference-manual-ug-1247.pdf. [Kasutatud 09 04 2022].

[24] Analog Devices Inc, „Energy Metering IC with Autocalibration Datasheet ADE9153A," [Võrgumaterjal]. Saadaval: https://www.analog.com/media/en/technical-documentation/data-sheets/ade9153a.pdf. [Kasutatud 31 03 2022].

[25] Analog Devices, inc, „Layout Considerations when Adding Energy Monitoring to a System Using the ADE9153A," 2018. [Võrgumaterjal]. Saadaval: https://www.analog.com/media/en/technical-documentation/application-notes/AN-1562.pdf. [Kasutatud 03 04 2022].

[26] B. Suppanz, „PCB Trace Width Calculator," 31 01 2006. [Võrgumaterjal]. Saadaval: https://circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/. [Kasutatud 03 04 2022].

[27] „ESPAsyncWiFiManager," [Võrgumaterjal]. Saadaval: https://github.com/alanswx/ESPAsyncWiFiManager. [Kasutatud 25 04 2022].

[28] Arduino, „ArduinoOTA," [Võrgumaterjal]. Saadaval: https://www.arduino.cc/reference/en/libraries/arduinoota/. [Kasutatud 25 04 2022].

[29] Adafruit, „Adafruit_SSD1306," [Võrgumaterjal]. Saadaval: https://github.com/adafruit/Adafruit_SSD1306. [Kasutatud 15 05 2022].

[30] Adafruit, „Adafruit GFX Library," [Võrgumaterjal]. Saadaval: https://github.com/adafruit/Adafruit-GFX-Library. [Kasutatud 15 05 2022].

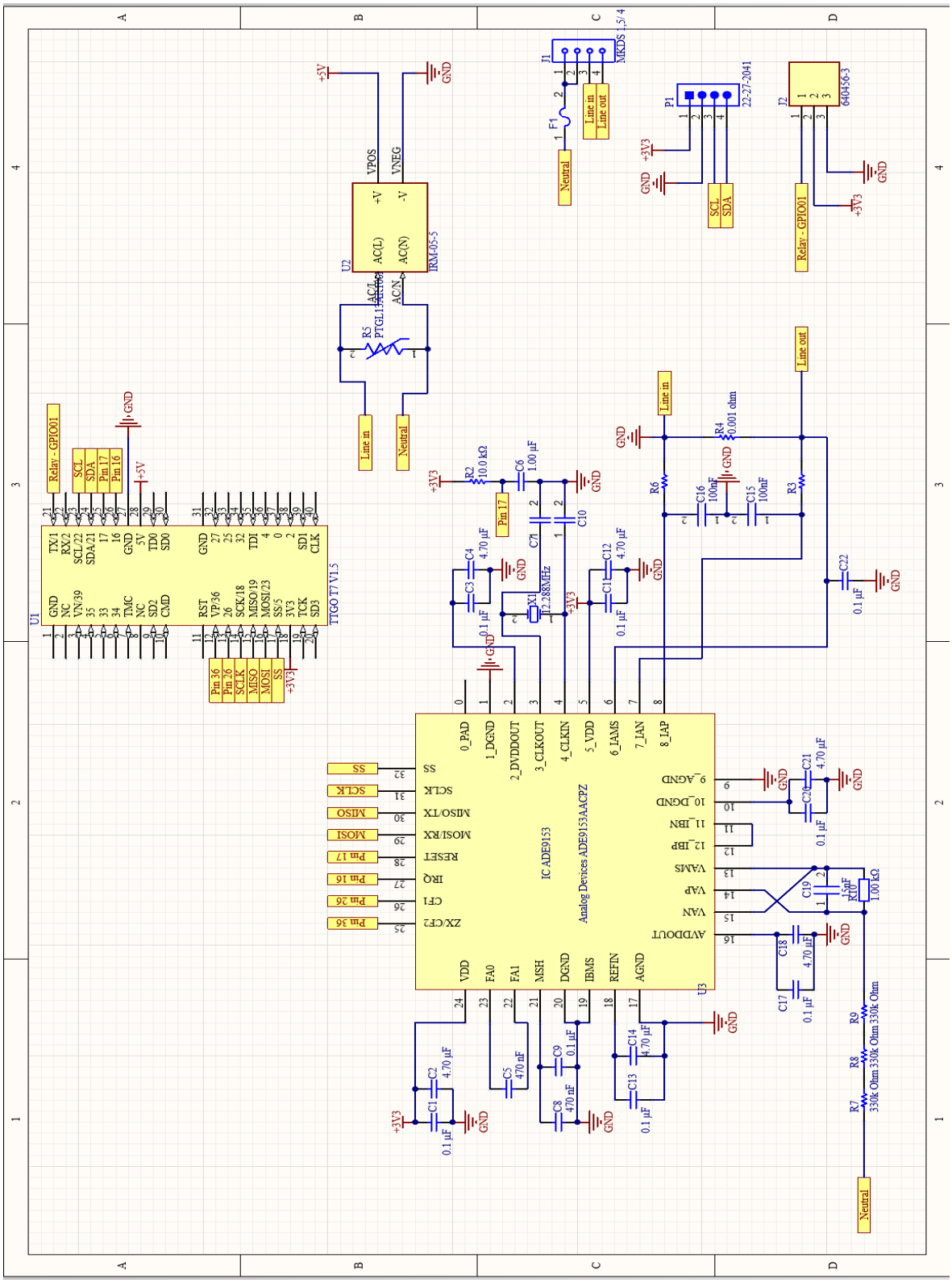
[31] R. Scheibler, „ESP32 FFT," [Võrgumaterjal]. Saadaval: https://github.com/fakufaku/esp32-fft. [Kasutatud 10 05 2022].

[32] Ecma International , „The JSON Data Interchange Syntax," 12 2017. [Võrgumaterjal]. Saadaval: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf. [Kasutatud 23 04 2022].

[33] Highcharts, [Võrgumaterjal]. Saadaval: https://www.highcharts.com/. [Kasutatud 23 04 2022].

# LISAD

Lisa 1 Nutika energiamõõturi terve elektriskeem



Lisa 2 Energiamõõturis kasutatavate komponentide nimekiri koos hindadega

| Komponent | Kogus, tk | Koguhind, € |
|---|---|---|
| 0805 0,1µF SMD kondensaator | 7 | 0,2 |
| 0805 4,7µF SMD kondensaator | 6 | 0,2 |
| 0805 0,47µF SMD kondensaator | 2 | 0,05 |
| 0805 1µF SMD kondensaator | 1 | 0,027 |
| 0805 30pF SMD kondensaator | 2 | 0,11 |
| 1206 0,1µF C0G SMD kondensaator | 2 | 0,4 |
| 0805 15nF SMD kondensaator | 1 | 0,19 |
| Sulavkaitse 1A 250VAC | 1 | 0,56 |
| Kruviterminal 1x4 | 1 | 1,08 |
| 3 pin pesa | 1 | 0,3 |
| 4 pin pesa | 1 | 0,3 |
| 0805 150Ω SMD takisti | 2 | 0,05 |
| 0805 10kΩ SMD takisti | 1 | 0,025 |
| 2512 0,001Ω SMD šunt | 1 | 0,4 |
| 0805 330kΩ SMD takisti | 3 | 0,075 |
| 0805 1kΩ SMD takisti | 1 | 0,025 |
| TTGO T7 ESP32 | 1 | 8,25 |
| IRM-05-5 toiteplokk | 1 | 10,35 |
| Varistor | 1 | 0,53 |
| ADE9153A | 1 | 6,3 |
| 12,288MHz kristall | 1 | 0,285 |
| Trükkplaat | 1 | 1 |
| 3D prinditud ümbris | 1 | 1 |
| Kokku: | 40 | 31,71 |

Lisa 3 Nelja katse töötlemata andmed

| Enne seadme registrite kalibreerimist | | | |
|---|---|---|---|
| Katse 1 | Ilma koormuseta sagedusel 50Hz | DMM Keysight 34461A | AC/AC PSU Keysight AC6802A |

| Seadepinge | Seadmelt mõõdetud pinge | Multimeetriga mõõdetud pinge |
|---|---|---|
| 100VAC | 101,909 | 99,975 |
| 150VAC | 153,272 | 150,322 |
| 200VAC | 204,77 | 200,799 |
| 230VAC | 235,593 | 231,006 |

| Katse 2 | ~650W koormusega sagedusel 50Hz | DMM Keysight 34461A | AC/AC PSU Keysight AC6802A |
|---|---|---|---|

| Seadepinge | Seadmega mõõdetud pinge | Multimeetriga mõõdetud pinge | Seadmega mõõdetud vool | Multimeetriga mõõdetud vool | Seadmega mõõdetud võimsus | Toiteplokilt mõõdetud võimsus |
|---|---|---|---|---|---|---|
| 100VAC | 101,267 | 99,483 | 6,653 | 6,525 | 676 | 655 |
| 150VAC | 152,744 | 149,997 | 4,431 | 4,326 | 673,5 | 650 |
| 200VAC | 204,275 | 200,57 | 3,284 | 3,209 | 662,7 | 638 |
| 230VAC | 235,101 | 230,81 | 2,874 | 2,81 | 661,5 | 638 |

| Peale seadme registrite kalibreerimist | | | |
|---|---|---|---|
| Katse 3 | Ilma koormuseta sagedusel 50Hz | DMM Keysight 34461A | AC/AC PSU Keysight AC6802A |

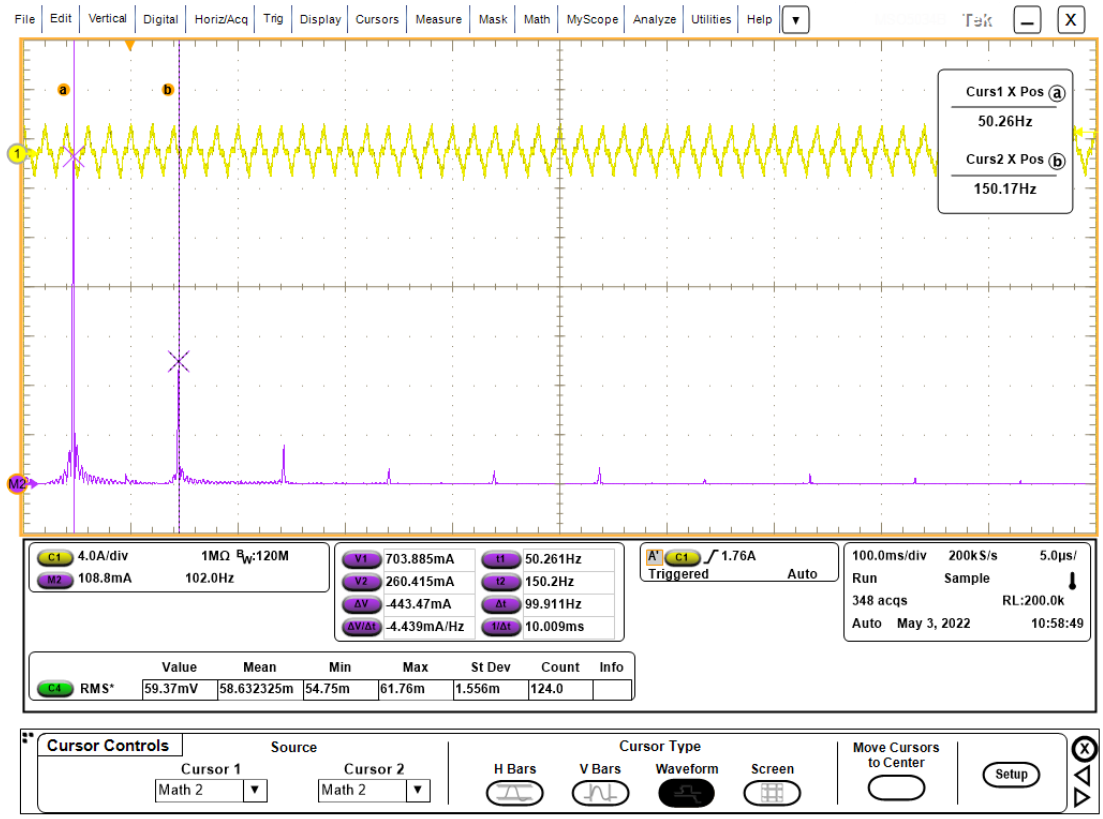
| Seadepinge | Seadmelt mõõdetud pinge | Multimeetriga mõõdetud pinge |
|---|---|---|
| 100VAC | 99,898 | 99,975 |
| 150VAC | 150,283 | 150,325 |
| 200VAC | 200,818 | 200,798 |
| 230VAC | 231,015 | 231,005 |

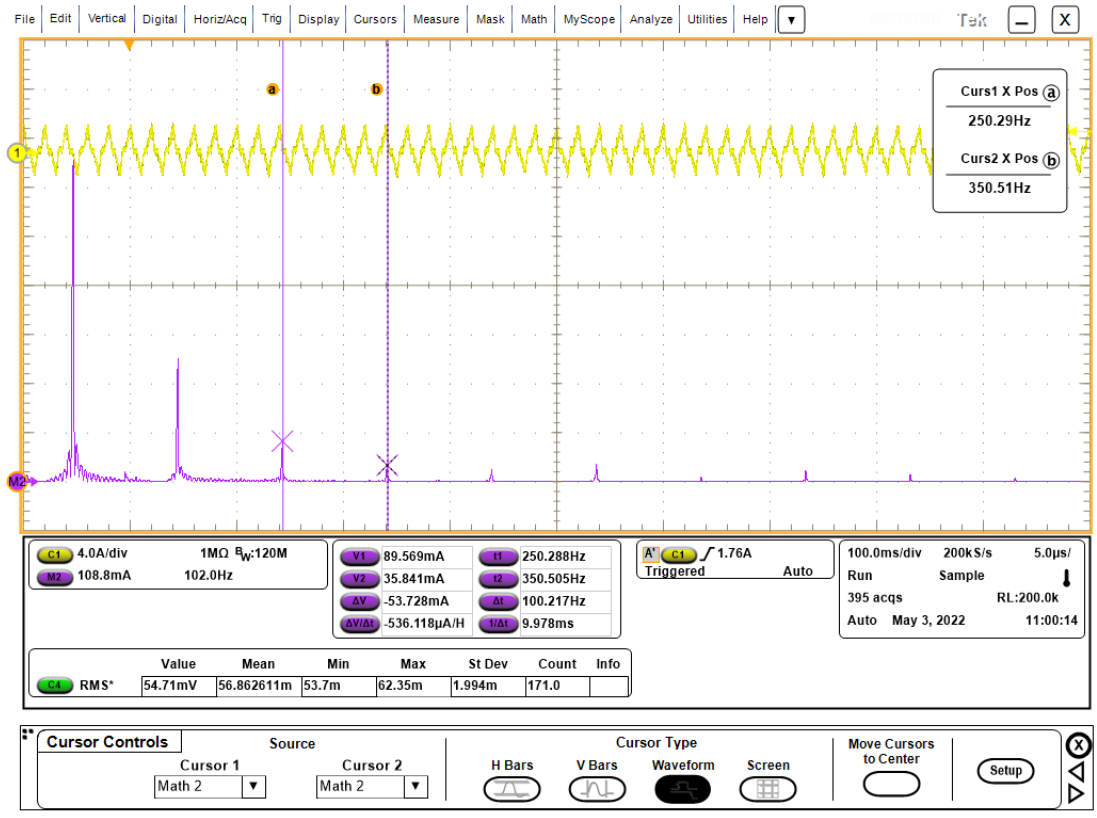
| | | | |
|---|---|---|---|
| Katse 4 | ~650W koormusega sagedusel 50Hz | DMM Keysight 34461A | AC/AC PSU Keysight AC6802A |

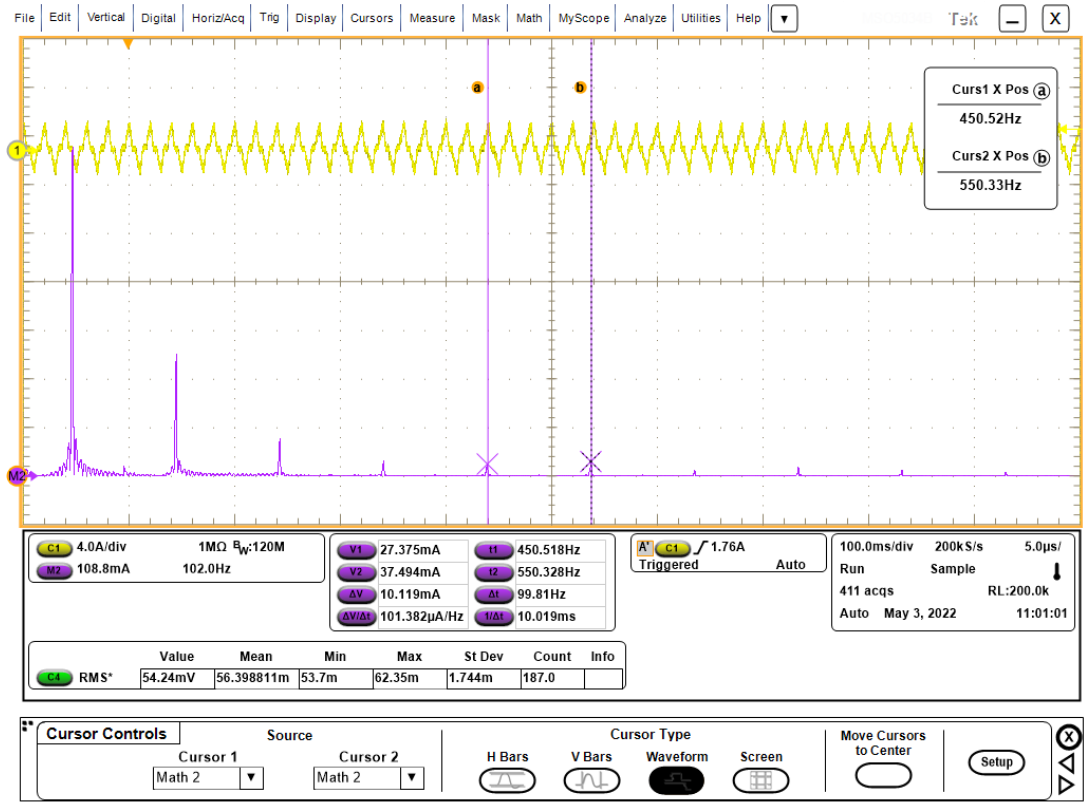
| Seadepinge | Seadmega mõõdetud pinge | Multimeetriga mõõdetud pinge | Seadmega mõõdetud vool | Multimeetriga mõõdetud vool | Seadmega mõõdetud võimsus | Toiteplokilt mõõdetud võimsus |
|---|---|---|---|---|---|---|
| 100VAC | 99,466 | 99,48 | 6,629 | 6,627 | 651,2 | 655,5 |
| 150VAC | 149,984 | 149,995 | 4,372 | 4,371 | 657,54 | 653,7 |
| 200VAC | 200,668 | 200,565 | 3,211 | 3,211 | 642,5 | 641,8 |
| 230VAC | 231,006 | 230,9 | 2,809 | 2,809 | 636,96 | 636,2 |

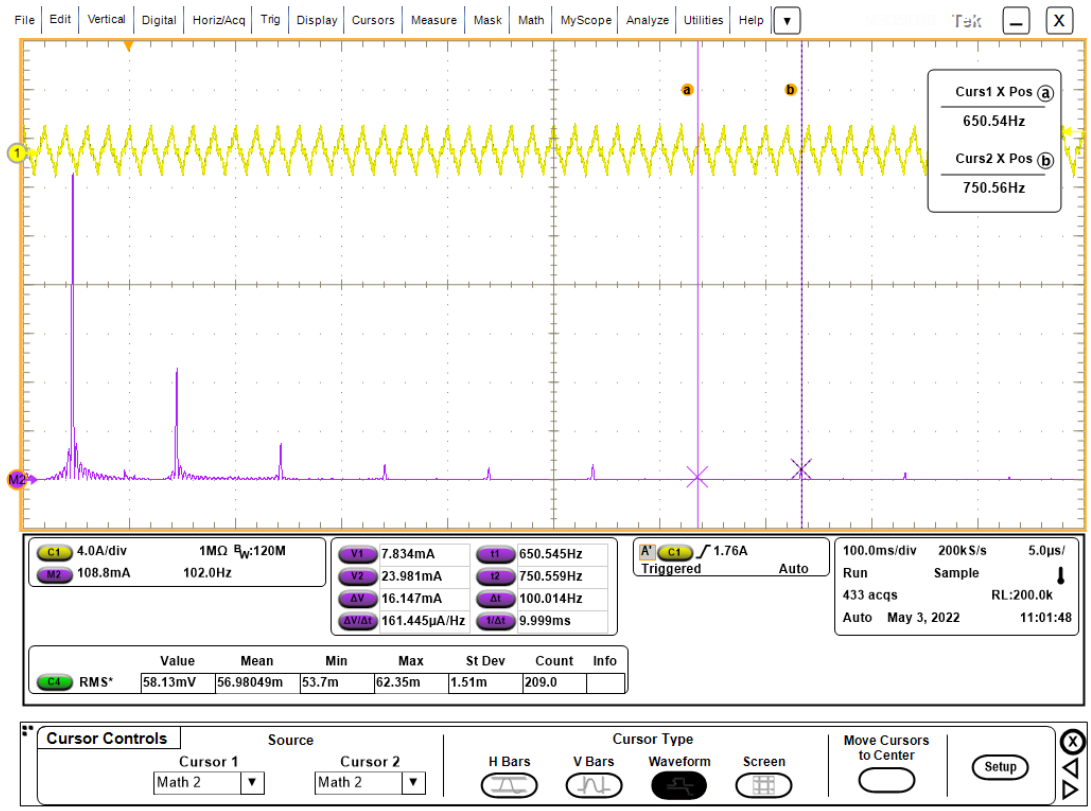
Lisa 4 Voolu moonutusteguri katsete mõõtmised

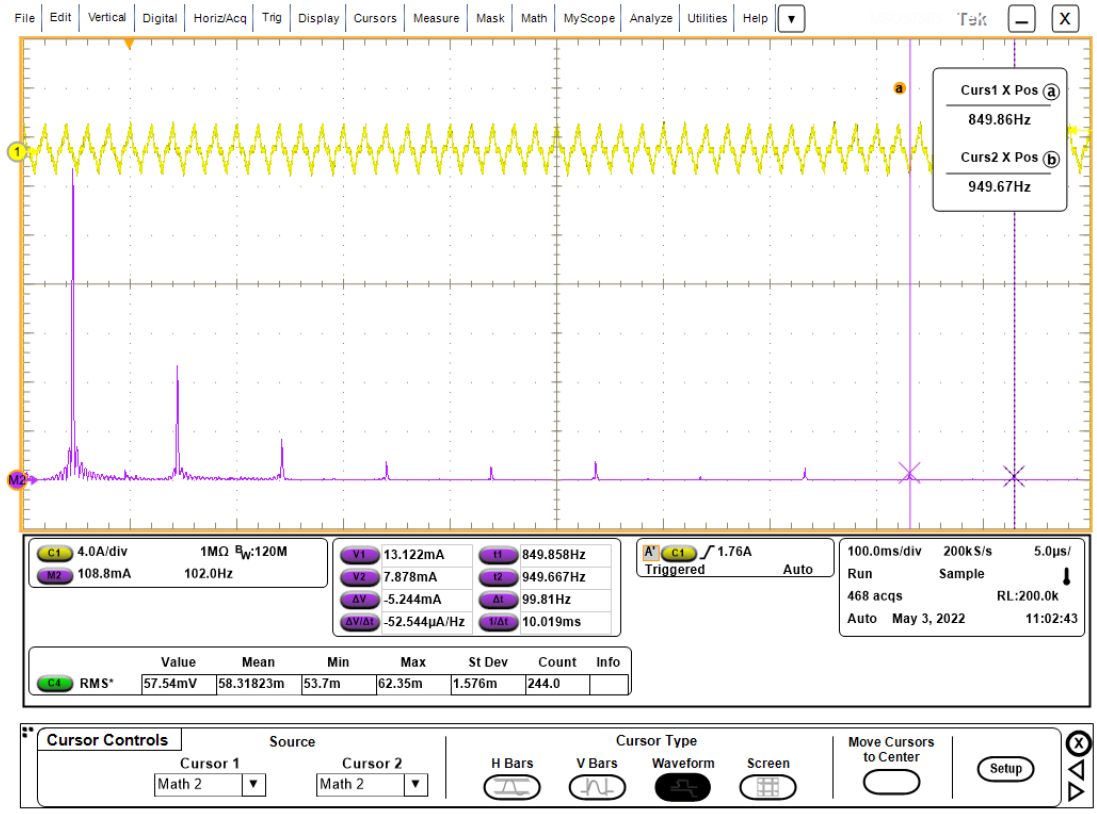
Sülearvuti laadija mõõtetulemused:



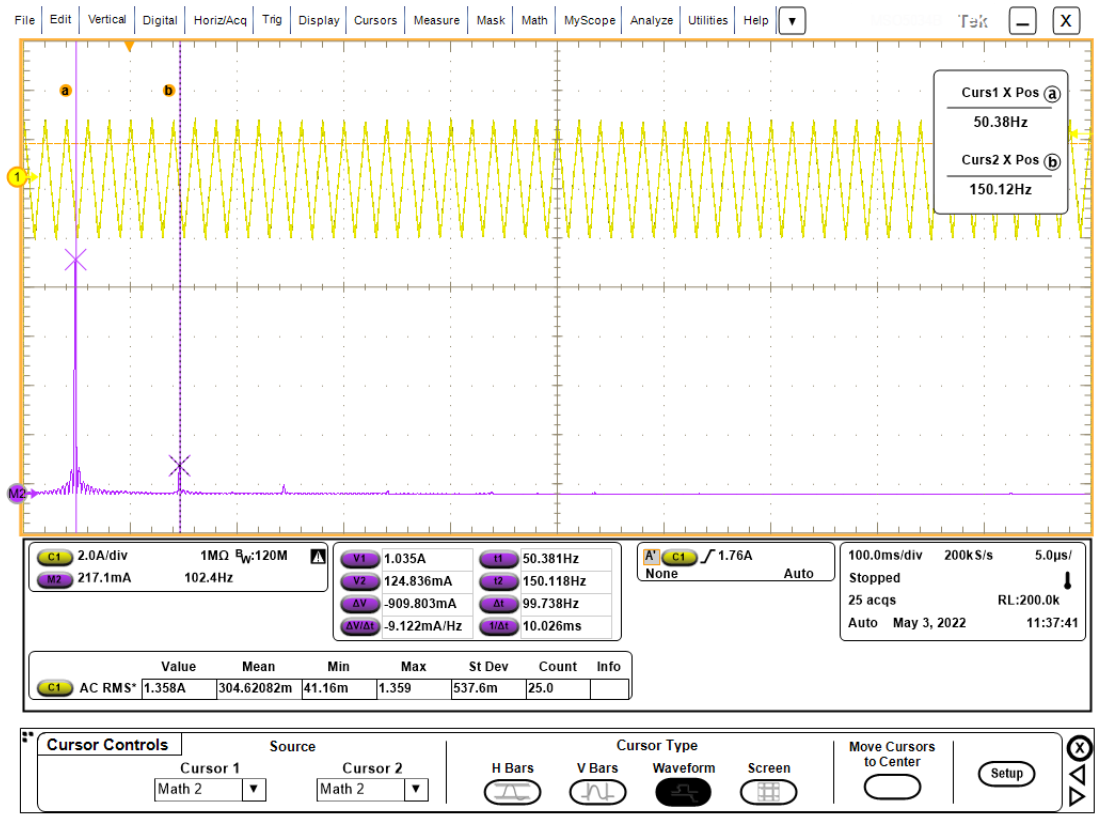



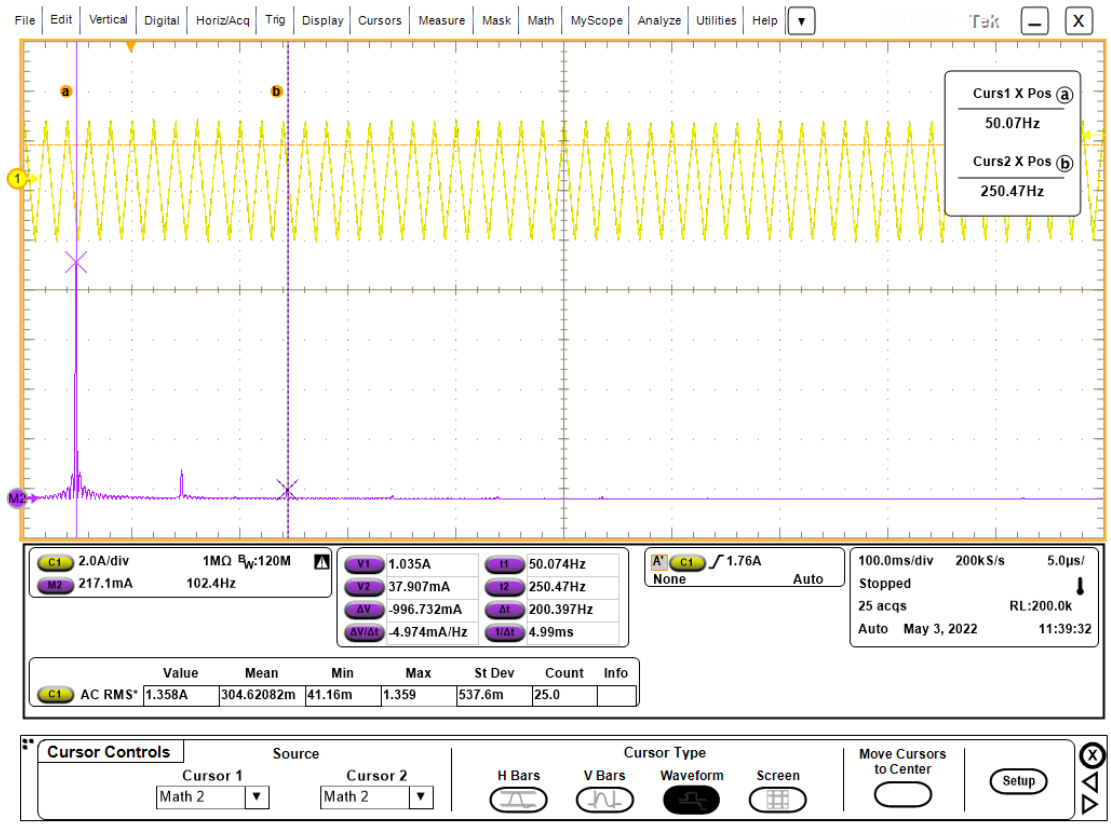

File Edit Vertical Digital Horiz/Acq Trig Display Cursors Measure Mask Math MyScope Analyze Utilities Help
Curs1 X Pos ⓐ
450.52Hz
Curs2 X Pos ⓑ
550.33Hz
C1 4.0A/div 1MΩ BW:120M
M2 108.8mA 102.0Hz
V1 27.375mA t1 450.518Hz
V2 37.494mA t2 550.328Hz
ΔV 10.119mA Δt 99.81Hz
ΔV/Δt 101.382µA/Hz 1/Δt 10.019ms
A' C1 1.76A
Triggered Auto
100.0ms/div 200kS/s 5.0µs/
Run Sample
411 acqs RL:200.0k
Auto May 3, 2022 11:01:01
Value Mean Min Max St Dev Count Info
C4 RMS* 54.24mV 56.398811m 53.7m 62.35m 1.744m 187.0
Cursor Controls
Source
Cursor 1 Math 2
Cursor 2 Math 2
Cursor Type
H Bars V Bars Waveform Screen
Move Cursors to Center
Setup


File Edit Vertical Digital Horiz/Acq Trig Display Cursors Measure Mask Math MyScope Analyze Utilities Help
Curs1 X Pos ⓐ
650.54Hz
Curs2 X Pos ⓑ
750.56Hz
C1 4.0A/div 1MΩ BW:120M
M2 108.8mA 102.0Hz
V1 7.834mA t1 650.545Hz
V2 23.981mA t2 750.559Hz
ΔV 16.147mA Δt 100.014Hz
ΔV/Δt 161.445µA/Hz 1/Δt 9.999ms
A' C1 1.76A
Triggered Auto
100.0ms/div 200kS/s 5.0µs/
Run Sample
433 acqs RL:200.0k
Auto May 3, 2022 11:01:48
Value Mean Min Max St Dev Count Info
C4 RMS* 58.13mV 56.98049m 53.7m 62.35m 1.51m 209.0
Cursor Controls
Source
Cursor 1 Math 2
Cursor 2 Math 2
Cursor Type
H Bars V Bars Waveform Screen
Move Cursors to Center
Setup




Elektritrelli mõõtetulemused:




File Edit Vertical Digital Horiz/Acq Trig Display Cursors Measure Mask Math MyScope Analyze Utilities Help
Curs1 X Pos ⓐ
50.07Hz
Curs2 X Pos ⓑ
250.47Hz
C1 2.0A/div 1MΩ BW:120M
M2 217.1mA 102.4Hz
V1 1.035A t1 50.074Hz
V2 37.907mA t2 250.47Hz
ΔV -996.732mA Δt 200.397Hz
ΔV/Δt -4.974mA/Hz 1/Δt 4.99ms
C1 1.76A
None Auto
100.0ms/div 200kS/s 5.0µs/
Stopped
25 acqs RL:200.0k
Auto May 3, 2022 11:39:32
Value Mean Min Max St Dev Count Info
C1 AC RMS* 1.358A 304.62082m 41.16m 1.359 537.6m 25.0
Cursor Controls
Source
Cursor 1
Math 2
Cursor 2
Math 2
Cursor Type
H Bars
V Bars
Waveform
Screen
Move Cursors to Center
Setup

Lisa 5 Mikrokontrolleri programmikood

```
#ifdef ESP32
  #include <WiFi.h>
  #include <AsyncTCP.h>
  #include <Spiffs.h>
  #include <HTTPClient.h>
  #include <FS.h>
  #include <PubSubClient.h>
  #include <Wire.h>
  #include <Adafruit_GFX.h>
  #include <Adafruit_SSD1306.h>
#endif

#ifdef ESP8266
  #include <ESPAsyncTCP.h>
  #include <ESP8266HTTPClient.h>
  #include <ESP8266WiFi.h>
  #include <Hash.h>
  #include <FS.h>
  #include <LittleFS.h>
#endif

#include <ADE9153A.h>
#include <ADE9153AAPI.h>
#include <Arduino.h>
#include <Wire.h>
#include <ArduinoOTA.h>
#include <WiFiClient.h>
#include <SPI.h>
#include "FFT.h" // include the library
#include <ESPAsyncWebServer.h>
#include <ESPAsyncWiFiManager.h>


#define OTA_PASSDW "admin"
#define DEBUG true // false: OTA_PASSDW enabled; true: OTA_PASSDW disabled

#define DEBUG_DUT false

#define MAX_LENGTH_TELNET_PW 50
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels


AsyncWebServer server2(80);
DNSServer dns;

// Create an Event Source on /events
AsyncEventSource events("/events");

WiFiClient wifiClient;

String apiKey = "cc9a44ab7f1a994adfc7a7a21f29529d";
```

```
/* Basic initializations */
#define SPI_SPEED 10000000     //SPI Speed

/*Esp32 pins CS 5, Mosi 23, Miso 19, Sclk 18; ESP32c3 CS 7, Mosi 6, Miso
5, Sclk 4*/
#ifdef ESP32
  #define CS_PIN 5            //5-esp32, 7-esp32c3
  #define ADE9153A_RESET_PIN 17 //17-esp32, 5-esp32c3
#endif

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

ADE9153AClass ade9153A;

struct EnergyRegs energyVals;  //Energy register values are read and
stored in EnergyRegs structure
struct PowerRegs powerVals;    //Metrology data can be accessed from these
structures
struct RMSRegs rmsVals;
struct PQRegs pqVals;
struct AcalRegs acalVals;
struct Temperature tempVal;
struct HalfRMSRegs halfrmsVals;


void readandwrite(void);
void resetADE9153A(void);
void runLength(unsigned long seconds);
void measureWaveForm(int cycles);
double doFFT(int cycles, int select);
uint32_t SPI_Read_32(uint16_t Address);
void configModeCallback (AsyncWiFiManager *myWiFiManager);
void wifiManagerInit();
void OTAinit();
void calibrate();
void initWebServer();
int trigger(int cycles, float triggerLevel, unsigned long timeout, int
select);
String processor(const String& var);
String outputState(int output);
void readEnergyValues(void);
void callback(char* topic, byte* payload, unsigned int length);
void displayInit();
void updateDisplay();

void advTask( void * parameter );
void saveEnergyTask( void * parameter );
void triggerTask( void * parameter );
void uptimeTask( void * parameter );
void sendDataToDBTask( void * parameter );
void OTATask( void * parameter );
void displayUpdateTask( void * parameter );
void calibTask( void * parameter );

const int blinkInterval = 5000;
const int kordus2 = 1024;
```

```
double THDV = 0;
double THDI = 0;
float realEnergy = 0;
float reactiveEnergy = 0;
float apparentEnergy = 0;
int relay = 1;
const float samplingFrequency = 4000;//3415, 2048
String triggerData;
String triggerData2;
String Message1;
String Message2;
String Message3;
String Message4;
int measMode = 0;
const char* PARAM_INPUT_1 = "output";
const char* PARAM_INPUT_2 = "state";

const char* PARAM_INPUT_TRIGGER_COUNT = "count";
const char* PARAM_INPUT_TRIGGER_LEVEL = "level";
const char* PARAM_INPUT_TRIGGER_TIMEOUT = "timeout";
const char* PARAM_INPUT_TRIGGER_SELECT = "select";

int waveState = 0;
int relayState = 1;
int wifiWTD = 0;


TaskHandle_t Task1, Task2, Task3, Task4, Task5, Task6, Task7, Task8,
Task9;

void setup()
{
  pinMode(ADE9153A_RESET_PIN, OUTPUT);
  pinMode(relay, OUTPUT);
  digitalWrite(ADE9153A_RESET_PIN, HIGH);
  digitalWrite(relay, LOW);

  if (DEBUG_DUT == true) Serial.begin(115200);


  if (!SPIFFS.begin()) {
    Serial.println("An error has occurred while mounting SPIFFS");
  }
  else{
    Serial.println("SPIFFS mounted successfully");
  }
  Serial.println("Booting");

  displayInit();
  readEnergyValues();
  wifiManagerInit();
  OTAinit();
  initWebServer();
  resetADE9153A();
  delay(500);
```

```
  xTaskCreate(saveEnergyTask,"Energy",4096,NULL,1,&Task4);
  delay(100);
  xTaskCreate(uptimeTask,"Uptime",4096,NULL,1,&Task6);
  delay(100);
  xTaskCreate(sendDataToDBTask,"Send2DB",8192,NULL,1,&Task7);
  delay(100);
  xTaskCreate(OTATask,"OTA",4096,NULL,1,&Task8);
  delay(100);
  xTaskCreate(displayUpdateTask,"OLED",4096,NULL,1,&Task9);

}

void loop() {
  yield();
}

void readandwrite()
{
 /* Read and Print WATT Register using ADE9153A Read Library */
  ade9153A.ReadPowerRegs(&powerVals);
  ade9153A.ReadRMSRegs(&rmsVals);
  ade9153A.ReadPQRegs(&pqVals);
  ade9153A.ReadTemperature(&tempVal);
  ade9153A.ReadEnergyRegs(&energyVals);
  ade9153A.ReadHalfRMSRegs(&halfrmsVals);

  if(WiFi.status()== WL_CONNECTED)
  {
    realEnergy = realEnergy + energyVals.ActiveEnergyValue/1000;
    reactiveEnergy = reactiveEnergy +
energyVals.FundReactiveEnergyValue/1000;
    apparentEnergy = apparentEnergy + energyVals.ApparentEnergyValue/1000;

    String serverName1 =
"http://192.168.50.154:8080/input/post?node=esp32v4&fulljson=";
    String Json = "";

    if (measMode == 0)
    {
      Json = "{\"Voltage\":"+ String(rmsVals.VoltageRMSValue/1000,3) +
",\"Current\":"+ String(rmsVals.CurrentRMSValue/1000,3) +
              ",\"Power\":"+ String(powerVals.ActivePowerValue/1000) +
",\"Apparent_Power\":"+ String(powerVals.ApparentPowerValue/1000) +
              ",\"Temp\":"+ String(tempVal.TemperatureVal) +
",\"Frequency\":"+ String(pqVals.FrequencyValue,3) + ",\"PF\":"+
String(pqVals.PowerFactorValue) +
              ",\"Phase_shift\":"+ String(pqVals.AngleValue_AV_AI) +
",\"Active_energy\":"+ String(realEnergy) + ",\"Reactive_energy\":"+
String(reactiveEnergy) +
              ",\"Apparent_energy\":"+ String(apparentEnergy)+
",\"HalfRMS_current\":"+ String(halfrmsVals.HalfCurrentRMSValue/1000) +
",\"THD_voltage\":"+ String(THDV) +
              ",\"THD_current\":"+ String(THDI) + ",\"Memory_free\":"+
String(ESP.getFreeHeap()) + "}";
    }
    else if (measMode == 1)
```

```
    {
      Json = "{\"Voltage\":"+ String(rmsVals.VoltageRMSValue/1000-1.05) +
",\"Current\":"+ String(rmsVals.CurrentRMSValue/1000*0.9434) +
              ",\"Power\":"+ String(powerVals.ActivePowerValue/1000) +
",\"Apparent_Power\":"+ String(powerVals.ApparentPowerValue/1000) +
",\"Temp\":"+ String(tempVal.TemperatureVal) +
              ",\"Frequency\":"+ String(pqVals.FrequencyValue,3) +
",\"PF\":"+ String(pqVals.PowerFactorValue) + ",\"Phase_shift\":" +
String(pqVals.AngleValue_AV_AI) +
              ",\"Active_energy\":"+ String(realEnergy) +
",\"Reactive_energy\":"+ String(reactiveEnergy) + ",\"Apparent_energy\":"+
String(apparentEnergy)+
              ",\"HalfRMS_current\":" +
String(halfrmsVals.HalfCurrentRMSValue/1000) + ",\"THD_voltage\":"+
String(THDV) + ",\"THD_current\":"+ String(THDI) +
              ",\"Memory_free\":"+ String(ESP.getFreeHeap()) + "}";
    }

    events.send(Json.c_str(),"new_data" ,millis());
    serverName1 = serverName1 + Json + "&apikey=" + apiKey;

    HTTPClient http;
    http.begin(wifiClient, serverName1);
    //Serial.println(serverName1);
    int httpResponseCode = http.GET();
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    wifiWTD = 0;

  }
  else {
    Serial.println("WiFi Disconnected");
    if (wifiWTD > 5) ESP.restart();//Wi-Fi watchdog
  }
}

void resetADE9153A(void)
{
 digitalWrite(ADE9153A_RESET_PIN, LOW);
 delay(100);
 digitalWrite(ADE9153A_RESET_PIN, HIGH);
 delay(1000);
 Serial.println("Reset Done");

  delay(1000);

  /*SPI initialization and test*/
  bool commscheck = ade9153A.SPI_Init(SPI_SPEED,CS_PIN); //Initialize SPI
  if (!commscheck) {
    Serial.println("ADE9153A Shield not detected. Plug in Shield and reset
the Arduino");

    while (!commscheck) {     //Hold until arduino is reset
      delay(1000);
      ArduinoOTA.handle();
    }
```

```
  }

  ade9153A.SetupADE9153A(); //Setup ADE9153A according to ADE9153AAPI.h
  /* Read and Print Specific Register using ADE9153A SPI Library */
  Serial.println(String(ade9153A.SPI_Read_32(REG_VERSION_PRODUCT), HEX));
// Version of IC
}

void runLength(unsigned long seconds)
{
  unsigned long startTime = millis();

  while (millis() - startTime < (seconds*1000)){
    delay(blinkInterval);
    ade9153A.ReadAcalRegs(&acalVals);
    Serial.print("AICC: ");
    Serial.println(acalVals.AICC);
    Serial.print("AICERT: ");
    Serial.println(acalVals.AcalAICERTReg);
    Serial.print("AVCC: ");
    Serial.println(acalVals.AVCC);
    Serial.print("AVCERT: ");
    Serial.println(acalVals.AcalAVCERTReg);

  }
  HTTPClient http;

    // Your Domain name with URL path or IP address with path
    String serverName1 =
"http://192.168.50.154:8080/input/post?node=esp32v4&fulljson={\"AICC\":"+
String(acalVals.AICC) + ",\"AICERT\":"+ String(acalVals.AcalAICERTReg) +
                         ",\"AVCC\":"+ String(acalVals.AVCC) +
",\"AVCERT\":"+ String(acalVals.AcalAVCERTReg) + "}&apikey=" + apiKey;

    http.begin(wifiClient, serverName1);

    int httpResponseCode = http.GET();
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);

    // Free resources
    http.end();
}

void measureWaveForm(int cycles)
{
  float arrayV[cycles];
  float arrayI[cycles];

  unsigned int sampling_period_us;
  unsigned long microseconds;
  sampling_period_us = round(1000000*(1.0/samplingFrequency));
  microseconds = micros();

  for(int i=0; i < cycles; i++)
  {
```

```
      arrayV[i] = int32_t (SPI_Read_32(REG_AV_WAV));
      arrayI[i] = int32_t (SPI_Read_32(REG_AI_WAV));
      while(micros() - microseconds < sampling_period_us){
        //empty loop
      }
      microseconds += sampling_period_us;
  }

  for (int i=0; i < cycles; i++)
  {
      arrayI[i] = arrayI[i] / 1194486;
      arrayV[i] = arrayV[i] / 70711;
  }

  // Your Domain name with URL path or IP address with path
  String Vwaveform = "{";
  String Iwaveform = "{";
  for (int i=0; i < cycles; i++)//function for building JSON string to
send data to Emoncms and website
  {
      Vwaveform = Vwaveform + "\"" + String(i) + "\":" +
String(arrayV[i],3);
      Iwaveform = Iwaveform + "\"" + String(i) + "\":" +
String(arrayI[i],3);
      if (i < cycles - 1)//-1 is original solution
      {
        Vwaveform = Vwaveform + ",";
        Iwaveform = Iwaveform + ",";
      }
  }
  Vwaveform = Vwaveform + "}";
  Iwaveform = Iwaveform + "}";

  events.send(Vwaveform.c_str(),"new_voltage" ,millis());
  events.send(Iwaveform.c_str(),"new_current" ,millis());

  //Serial.println(Vwaveform);
  //Serial.println(Iwaveform);
}

uint32_t SPI_Read_32(uint16_t Address)
{
  uint16_t temp_address;
  uint32_t temp_highpacket;
  uint16_t temp_lowpacket;
  uint32_t returnData;

  digitalWrite(CS_PIN, LOW);

  temp_address = (((Address << 4) & 0xFFF0) + 8);
  SPI.transfer16(temp_address);
  temp_highpacket = SPI.transfer16(0);
  temp_lowpacket = SPI.transfer16(0);

  digitalWrite(CS_PIN, HIGH);
```

```
  returnData = temp_highpacket << 16;
  returnData = returnData + temp_lowpacket;

  return returnData;
}

void configModeCallback (AsyncWiFiManager *myWiFiManager) {
  Serial.println("Entered config mode");
  Serial.println(WiFi.softAPIP());
  //if you used auto generated SSID, print it
  Serial.println(myWiFiManager->getConfigPortalSSID());
}

void wifiManagerInit()
{
  //WiFiManager
  //Local intialization. Once its business is done, there is no need to
keep it around
  AsyncWiFiManager wifiManager1(&server2,&dns);

  display.println("Connecting to WiFi,  Please wait ...");
  display.display();

  //set callback that gets called when connecting to previous WiFi fails,
and enters Access Point mode
  wifiManager1.setAPCallback(configModeCallback);

  //fetches ssid and pass and tries to connect
  //if it does not connect it starts an access point with the specified
name
  //here  "AutoConnectAP"
  //and goes into a blocking loop awaiting configuration
  if(!wifiManager1.autoConnect()) {
    Serial.println("failed to connect and hit timeout");
    //reset and try again, or maybe put it to deep sleep
    #ifdef ESP32
      ESP.restart();
    #endif

    #ifdef ESP8266
      ESP.reset();
    #endif
    delay(1000);
  }
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Wi-Fi connected, IP:");
  display.println(WiFi.localIP());
  display.display();
  Serial.println(WiFi.localIP());

  //if you get here you have connected to the WiFi
  Serial.println("connected...yeey :)");
  WiFi.setSleep(false);
  delay(2000);
}
```

```
void OTAinit()
{
  // Port defaults to 8266
  // ArduinoOTA.setPort(8266);

  // Hostname defaults to esp8266-[ChipID]
  // ArduinoOTA.setHostname("myesp8266");

  if (DEBUG == false) {
    // Comment to: No authentication by default
    ArduinoOTA.setPassword(OTA_PASSDW);
    // Password can be set with it's md5 value as well
    // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
    // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
  }

  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS
using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}


void calibrate()
{
    Serial.println("Autocalibrating Current Channel");
    ade9153A.StartAcal_AITurbo();
    runLength(60);
    ade9153A.StopAcal();
    Serial.println("Autocalibrating Voltage Channel");
```

```
    ade9153A.StartAcal_AV();
    runLength(60);
    ade9153A.StopAcal();
    delay(100);

    ade9153A.ReadAcalRegs(&acalVals);
    Serial.print("AICC: ");
    Serial.println(acalVals.AICC);
    Serial.print("AICERT: ");
    Serial.println(acalVals.AcalAICERTReg);
    Serial.print("AVCC: ");
    Serial.println(acalVals.AVCC);
    Serial.print("AVCERT: ");
    Serial.println(acalVals.AcalAVCERTReg);
    long Igain = (-acalVals.AICC-740.7) * 134217728; //622.4 for v3, 740.7
for v4
    long Vgain = (acalVals.AVCC-13289) * 134217728; //13283 for v3, 13289
for v4
    ade9153A.SPI_Write_32(REG_AIGAIN, Igain);
    ade9153A.SPI_Write_32(REG_AVGAIN, Vgain);

    Serial.println("Autocalibration Complete");
    delay(2000);
}

double doFFT(int cycles, int select)
{
  float fftInput[cycles];
  float fftOutput[cycles];

  unsigned int sampling_period_us;
  unsigned long microseconds;
  sampling_period_us = round(1000000*(1.0/samplingFrequency));
  microseconds = micros();

  //float samplesMinusOne = (float(cycles) - 1.0); //Needed for window
functions

  if (select == 0)
  {
    for(int i=0; i < cycles; i++)
    {
      fftInput[i] = int32_t (SPI_Read_32(REG_AV_WAV));
      while(micros() - microseconds < sampling_period_us){
        //empty loop
      }
      microseconds += sampling_period_us;
    }

    for (int i=0; i < cycles; i++)
    {
      //Window functions for FFT
      //float indexMinusOne = float(i);
      //float ratio = (indexMinusOne / samplesMinusOne);
      //float weighingFactor = 0.2810639 - (0.5208972 * cos(TWO_PI *
ratio)) + (0.1980399 * cos(2 * TWO_PI * ratio));
```

```
      //float weighingFactor = 0.54 * (1.0 - cos(TWO_PI * ratio));// hann
      //float weighingFactor = 0.54 * (0.46 - cos(TWO_PI * ratio));//
hamming
      float weighingFactor = 1;
      fftInput[i] = fftInput[i] / 70711 * weighingFactor;
    }
  }
  else if (select == 1)
  {
    for(int i=0; i < cycles; i++)
    {
      fftInput[i] = int32_t (SPI_Read_32(REG_AI_WAV));
      while(micros() - microseconds < sampling_period_us){
        //empty loop
      }
      microseconds += sampling_period_us;
    }
    for (int i=0; i < cycles; i++)
    {
      //window functions for FFT
      //float indexMinusOne = float(i);
      //float ratio = (indexMinusOne / samplesMinusOne);
      //float weighingFactor = 0.2810639 - (0.5208972 * cos(TWO_PI *
ratio)) + (0.1980399 * cos(2 * TWO_PI * ratio)); // flat top
      //float weighingFactor = 0.54 * (1.0 - cos(TWO_PI * ratio)); // hann
      //float weighingFactor = 0.54 * (0.46 - cos(TWO_PI * ratio));//
hamming
      float weighingFactor = 1;

      fftInput[i] = fftInput[i] / 1194486 * weighingFactor;
    }
  }
  float max_magnitude = 0;
  float fundamental_freq = 0;

  fft_config_t *real_fft_plan = fft_init(kordus2, FFT_REAL, FFT_FORWARD,
fftInput, fftOutput);
  fft_execute(real_fft_plan);

  for (int k = 1 ; k < real_fft_plan->size / 2 ; k++)
  {
    /*The real part of a magnitude at a frequency is followed by the
corresponding imaginary part in the output*/
    float mag = sqrt(pow(real_fft_plan->output[2*k],2) +
pow(real_fft_plan->output[2*k+1],2))/1;
    float freq = k*1.0/(kordus2/samplingFrequency);
    if(mag > max_magnitude)
    {
        max_magnitude = mag;
        fundamental_freq = freq;
    }
  }

  int y0 = (fundamental_freq) * (kordus2/samplingFrequency*2);

  double calcTHD = 0;
```

```
  float y1 = abs(fftOutput[y0]);
  for (int i = -3; i < 4; i++)
  {
    float temp = abs(fftOutput[y0+i]);
    if (temp > y1)
    {
      y1 = temp;
    }

  }
  float tegur = samplingFrequency/kordus2;
  float ysum = 0;

  for (int i = 2; i < ((samplingFrequency / 2 ) / (y0*tegur)); i++)
  {
    float yx = abs(fftOutput[y0*i]);
    for (int j = -3; j < 4; j++)
    {
      float temp = abs(fftOutput[y0*i+j]);
      if (temp > yx)
      {
        yx = temp;
      }
    }
    ysum = ysum + yx*yx;
  }

  calcTHD = sqrt(ysum)/y1*100;
  Serial.print("THD: ");
  Serial.print(calcTHD, 2);
  Serial.println(" %");

  String FFTV = "{";
  FFTV = FFTV + "\"" + String(0) + "\":" +
String(abs(fftOutput[0])/cycles)+ ",";
  for (int i=1; i < (cycles >> 1); i=i+1)
  {
    FFTV = FFTV + "\"" + String(i) + "\":" +
String(abs(fftOutput[i])/cycles*2);

    if (i < (cycles >> 1 )- 1)//-1 is original solution
    {
      FFTV = FFTV + ",";
    }
  }
  FFTV = FFTV + "}";
  fft_destroy(real_fft_plan);

  if (select == 0)
  {
    events.send(FFTV.c_str(),"new_VFFT" ,millis());
  }
  else if (select == 1)
  {
    events.send(FFTV.c_str(),"new_IFFT" ,millis());
```

```
  }
  return calcTHD;
}

void initWebServer ()
{
  // Web Server Root URL
  server2.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", "text/html", false, processor);
  });

  server2.serveStatic("/", SPIFFS, "/");

  // Request for the latest sensor readings
  server2.on("/readings", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", "OK!");
  });

  // Request for voltage and current calibration
  server2.on("/calib", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", "OK!");
    xTaskCreate(calibTask,"Calib",2048,NULL,1,&Task2);
  });

  // Send a GET request to
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
  server2.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputMessage1 = "";
    String inputMessage2 = "";

    // GET input1 value on
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
    if (request->hasParam(PARAM_INPUT_1) && request-
>hasParam(PARAM_INPUT_2)) {
      inputMessage1 = request->getParam(PARAM_INPUT_1)->value();
      inputMessage2 = request->getParam(PARAM_INPUT_2)->value();

      if ((inputMessage1 == "Waveb") && (inputMessage2.toInt() == 1) &&
(waveState == 0))
      {
        request->send(200, "text/plain", "Wave task started!");
        xTaskCreate(advTask,"Wave",20000,NULL,5,&Task1);
        waveState = 1;
      }

      else if ((inputMessage1 == "Waveb") && (inputMessage2.toInt() == 0)
&& (waveState == 1))
      {
        request->send(200, "text/plain", "Wave task killed!");
        vTaskDelete(Task1);
        waveState = 0;
      }

      if ((inputMessage1 == "Relayb") && (inputMessage2.toInt() == 1) &&
(relayState == 0))
```

```
      {
        request->send(200, "text/plain", "Load turned on!");
        digitalWrite(relay, LOW);
        relayState = 1;

      }

      else if ((inputMessage1 == "Relayb") && (inputMessage2.toInt() == 0)
&& (relayState == 1))
      {
        request->send(200, "text/plain", "Load turned off!");
        digitalWrite(relay, HIGH);
        relayState = 0;

      }

      else if ((inputMessage1 == "Modeb") && (inputMessage2.toInt() == 0))
      {
        request->send(200, "text/plain", "AC mode!");
        measMode = 0;
        ade9153A.SPI_Write_32(REG_CONFIG0,0x00000000);
      }

      else if ((inputMessage1 == "Modeb") && (inputMessage2.toInt() == 1))
      {
        request->send(200, "text/plain", "DC mode!");
        measMode = 1;
        ade9153A.SPI_Write_32(REG_CONFIG0,0x00000001);
      }
    }

    else if (request->hasParam(PARAM_INPUT_TRIGGER_COUNT) && request-
>hasParam(PARAM_INPUT_TRIGGER_LEVEL) && request-
>hasParam(PARAM_INPUT_TRIGGER_TIMEOUT) && request-
>hasParam(PARAM_INPUT_TRIGGER_SELECT))
    {
      Message1 = request->getParam(PARAM_INPUT_TRIGGER_COUNT)->value();
      Message2 = request->getParam(PARAM_INPUT_TRIGGER_LEVEL)->value();
      Message3 = request->getParam(PARAM_INPUT_TRIGGER_TIMEOUT)->value();
      Message4 = request->getParam(PARAM_INPUT_TRIGGER_SELECT)->value();

      xTaskCreate(triggerTask,"trigger",10000,NULL,6,&Task5);
      request->send(200, "text/plain", "Trigger task started!");
    }

    else {
      inputMessage1 = "No message sent";
      inputMessage2 = "No message sent";
    }

    Serial.print(inputMessage1);
    Serial.print(" - Set to: ");
    Serial.println(inputMessage2);

  });
```

```
  events.onConnect([](AsyncEventSourceClient *client){
    if(client->lastId()){
      Serial.printf("Client reconnected! Last message ID that it got is:
%u\n", client->lastId());
    }
    // send event with message "hello!", id current millis
    // and set reconnect delay to 10 seconds
    client->send("hello!", NULL, millis(), 10000);
  });

  server2.addHandler(&events);

  // Start server
  server2.begin();
}

int trigger(int cycles, float triggerLevel, unsigned long timeout, int
select)
{
  unsigned int sampling_period_us;
  unsigned long microseconds;
  unsigned long milliseconds;
  float triggerBuffer[cycles];
  float triggerBuffer2[cycles];

  sampling_period_us = round(1000000*(1.0/samplingFrequency));
  int32_t triggerCode = 0;
  if (select == 0)
  {
    triggerCode = triggerLevel * 1194486; //for current
  }
  else if (select == 1)
  {
    triggerCode = triggerLevel * 70711; //for voltage
  }
  milliseconds = millis();
  microseconds = micros();

  int y = 0;
  int32_t  readValue = (SPI_Read_32(REG_AI_WAV));
  int32_t  readValue2 = (SPI_Read_32(REG_AV_WAV));
  triggerBuffer[y] = readValue;
  triggerBuffer2[y] = readValue;
  y++;
  delayMicroseconds(200);

  //Prefilling buffer
  while ((readValue < triggerCode && select == 0) || (readValue2 <
triggerCode && select == 1) || y < cycles/2)
  {
    readValue = (SPI_Read_32(REG_AI_WAV));
    readValue2 = (SPI_Read_32(REG_AV_WAV));
    if ( y > cycles/2)
    {
      for (int i = 1; i < y + 1; i++)
      {
```

```
        triggerBuffer[i-1] = triggerBuffer[i];
        triggerBuffer2[i-1] = triggerBuffer2[i];
      }
      triggerBuffer[cycles/2] = readValue;
      triggerBuffer2[cycles/2] = readValue2;
    }
    else
    {
      triggerBuffer[y] = readValue;
      triggerBuffer2[y] = readValue2;
      y++;
    }

    if (millis() - milliseconds > timeout) return 1;

    while(micros() - microseconds < sampling_period_us)
    {
        //empty loop
    }
    microseconds += sampling_period_us;
  }

  for(int i=cycles/2+1; i < cycles; i++)
  {
      triggerBuffer[i] = int32_t (SPI_Read_32(REG_AI_WAV));
      triggerBuffer2[i] = int32_t (SPI_Read_32(REG_AV_WAV));
      while(micros() - microseconds < sampling_period_us)
      {
        //empty loop
      }
      microseconds += sampling_period_us;
  }
  for (int i=0; i < cycles; i++)
  {
      triggerBuffer[i] = triggerBuffer[i] / 1194486;
      triggerBuffer2[i] = triggerBuffer2[i] / 70711;
  }

  triggerData = "{";
  triggerData2 = "{";
  for (int i=0; i < (cycles); i++)
  {
    triggerData = triggerData + "\"" + String(i) + "\":" +
String(triggerBuffer[i]);
    triggerData2 = triggerData2 + "\"" + String(i) + "\":" +
String(triggerBuffer2[i]);
    if (i < (cycles)- 1)
    {
      triggerData = triggerData + ",";
      triggerData2 = triggerData2 + ",";
    }
  }
  triggerData = triggerData + "}";
  triggerData2 = triggerData2 + "}";
  return 0;
```

```
}

void advTask( void * parameter )
{
  for (;;)
  {
    measureWaveForm(200);
    THDV = doFFT(kordus2, 0);
    THDI = doFFT(kordus2, 1);
    vTaskDelay(15000 / portTICK_PERIOD_MS);
  }
}


void saveEnergyTask( void * parameter )
{
  for (;;)
  {
    File file1 = SPIFFS.open("/RealEnergy.txt", FILE_WRITE);
    file1.print(String(realEnergy,3));
    Serial.println(String(realEnergy,3));
    file1.close();

    File file2 = SPIFFS.open("/ImagEnergy.txt", FILE_WRITE);
    file2.print(String(reactiveEnergy,3));
    Serial.println(String(reactiveEnergy,3));
    file2.close();

    File file3 = SPIFFS.open("/ApparentEnergy.txt", FILE_WRITE);
    file3.print(String(apparentEnergy,3));
    Serial.println(String(apparentEnergy,3));
    file3.close();

    vTaskDelay(60000 / portTICK_PERIOD_MS);
  }
}

void triggerTask( void * parameter )
{
  int status = trigger(Message1.toInt(), Message2.toFloat(),
Message3.toInt(), Message4.toInt());
  for (;;)
  {
    if (status == 0)
    {
      events.send(triggerData.c_str(),"new_trigger_current" ,millis());
      events.send(triggerData2.c_str(),"new_trigger_voltage" ,millis());
      Serial.println("Data sent");
    }
    vTaskDelete(Task5);
  }
}

void uptimeTask( void * parameter )
{
  for (;;)
```

```
  {
    events.send(String(millis()/1000).c_str(),"uptime" ,millis());
    vTaskDelay(30000 / portTICK_PERIOD_MS);
  }
}

String processor(const String& var)
{
  if(var == "BUTTONPLACEHOLDER") //lülitite seisundi saamine
  {
    String buttons = "";
    buttons += "<h4>AC/DC mode switch</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"Modeb\" " +
outputState(measMode) + "><span class=\"slider2\"></span></label>";
    buttons += "<h4>Show V/I Waveform, Voltage and Current FFT</h4><label
class=\"switch\"><input type=\"checkbox\"
onchange=\"toggleCheckbox(this)\" id=\"Waveb\" " + outputState(waveState)
+ "><span class=\"slider\"></span></label>";
    buttons += "<h4>Toggle connected load</h4><label
class=\"switch\"><input type=\"checkbox\"
onchange=\"toggleCheckbox(this)\" id=\"Relayb\" " +
outputState(relayState) + "><span class=\"slider\"></span></label>";
    return buttons;
  }
  return String();
}

String outputState(int output)
{
  if(output == 1)
  {
    return "checked";
  }
  else
  {
    return "";
  }
}

void readEnergyValues(void)
{
  File file1 = SPIFFS.open("/RealEnergy.txt");
  String buffer = "";
  if (!file1)
  {
    Serial.println("There was an error opening the file1 for reading");
  }
  while(file1.Available())
  {
    buffer += (char)file1.read();
  }
  Serial.println(buffer);
  realEnergy = buffer.toFloat();
  Serial.println(realEnergy);
  file1.close();
```

```
  File file2 = SPIFFS.open("/ImagEnergy.txt");
  buffer = "";
  if (!file2)
  {
    Serial.println("There was an error opening the file2 for reading");
  }
  while(file2.Available())
  {
    buffer += (char)file2.read();
  }

  Serial.println(buffer);
  reactiveEnergy = buffer.toFloat();
  Serial.println(reactiveEnergy);
  file2.close();

  File file3 = SPIFFS.open("/ApparentEnergy.txt");
  buffer = "";
  if (!file3)
  {
    Serial.println("There was an error opening the file3 for reading");
  }
  while(file3.Available())
  {
    buffer += (char)file3.read();
  }

  Serial.println(buffer);
  apparentEnergy = buffer.toFloat();
  Serial.println(apparentEnergy);
  file3.close();
}

void sendDataToDBTask( void * parameter )
{
  for (;;)
  {
    if (wifiWTD > 5) ESP.restart();//Wi-Fi watchdog,
    wifiWTD++;
    readandwrite();//Uploading data to external server
    vTaskDelay(5000 / portTICK_PERIOD_MS);
  }
}

void OTATask( void * parameter )
{
  for (;;)
  {
    ArduinoOTA.handle();//OTA update task
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}

void displayUpdateTask( void * parameter )
{
  for (;;)
```

```
  {
    updateDisplay(); //Display update
    vTaskDelay(5000 / portTICK_PERIOD_MS);
  }
}

void calibTask( void * parameter )
{
  for (;;)
  {
    calibrate(); //voltage and current channel update
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    vTaskDelete(Task2);
  }
}

void displayInit()
{
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 allocation failed");
    for(;;);
  }
  delay(2000);

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
}


void updateDisplay()
{
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Live Data 1/3:");
  display.setCursor(0,15);

  double test1 = rmsVals.VoltageRMSValue/1000;
  double test2 = rmsVals.CurrentRMSValue/1000;
  double test3 = powerVals.ActivePowerValue/1000;
  double test4 = powerVals.ApparentPowerValue/1000;
  double test5 = pqVals.FrequencyValue;
  double test6 = pqVals.PowerFactorValue;

  int placeX1 = 94;
  int placeY1 = 1;
  int placeX2 = placeX1 +20;
  int placeY2 = placeY1;

  int placeX3 = 70;
  int placeY3 = 14;
  int placeX4 = placeX3 +20;
  int placeY4 = placeY3;

  int placeX5 = 82;
```

```
  int placeY5 = 27;
  int placeX6 = placeX5 +20;
  int placeY6 = placeY5;

  //Uptime
  long timeVar = millis()/1000;
  long secs = timeVar % 60;
  timeVar = (timeVar - secs) / 60;

  long mins = timeVar % 60;
  timeVar = (timeVar - mins) / 60;

  long hrs = timeVar % 24;
  timeVar = (timeVar - hrs) / 24;

  long days = timeVar;

  delay(10);

  display.println("Voltage: "+ String(test1, 3) + " V");
  display.println("Current: "+ String(test2, 3)+ " A");
  delay(10);
  display.println("Power: "+ String(test3)+ " W");
  display.println("Ap. Power: "+ String(test4)+ " VA");
  delay(10);
  display.println("Freq: "+ String(test5, 3)+ " Hz");
  display.println("PF: "+ String(test6));
  delay(10);
  display.display();
  delay(5000);

  display.clearDisplay();
  delay(10);
  display.setCursor(0,0);
  display.println("Live Data 2/3:");
  display.setCursor(0,15);
  delay(10);
  display.println("Active energy: ");
  display.println(String(realEnergy) + " Wh");
  display.println("Reactive energy: ");
  display.println(String(reactiveEnergy) + " varh");
  display.println("Apparent energy: ");
  display.println(String(apparentEnergy) + " VAh");
  delay(5);
  display.display();
  delay(5000);

  display.clearDisplay();
  delay(5);
  display.setCursor(0,0);
  display.println("Live Data 3/3:");
  display.setCursor(0,15);

  display.println("Ph_shift: "+ String(pqVals.AngleValue_AV_AI) + " deg");
  display.println("HalfRMS cur: "+
String(halfrmsVals.HalfCurrentRMSValue/1000, 3)+ " A");
```

```
  delay(10);
  display.println("THD_V: "+ String(THDV)+" %");
  display.println("THD_I: "+ String(THDI)+ " %");
  delay(10);
  display.println("Temp: "+ String(tempVal.TemperatureVal)+ " C");
  display.println("Free_mem: "+ String(ESP.getFreeHeap())+ " B");
  delay(10);
  display.display();
  delay(5000);


  display.clearDisplay();
  delay(5);
  display.setCursor(0,placeY1);
  display.println("Operation mode: ");
  display.setCursor(placeX1,placeY1);
  display.println("AC");
  display.setCursor(placeX2,placeY2);
  display.println("DC");

  if (measMode == 0)
  {
    display.fillRect(placeX1-1, placeY1-1, 13, 9, INVERSE);//AC mode
  }
  else if (measMode == 1)
  {
    display.fillRect(placeX2-1, placeY2-1, 13, 9, INVERSE);//DC mode
  }

  display.setCursor(0,placeY3);
  display.println("Adv. mode: ");
  display.setCursor(placeX3,placeY3);
  display.println("ON");
  display.setCursor(placeX4,placeY3);
  display.println("OFF");

  if (waveState == 1)
  {
    display.fillRect(placeX3-1, placeY3-1, 13, 9, INVERSE); //Waveform +
FFT off
  }
  else if (waveState == 0)
  {
    display.fillRect(placeX4-1, placeY4-1, 20, 9, INVERSE);//Waveform +
FFT on
  }

  display.setCursor(0,placeY5);
  display.println("Load status: ");
  display.setCursor(placeX5,placeY5);
  display.println("ON");
  display.setCursor(placeX6,placeY6);
  display.println("OFF");

  if (relayState == 1)
  {
```

```
    display.fillRect(placeX5-1, placeY5-1, 13, 9, INVERSE); //Waveform +
FFT off
  }
  else if (relayState == 0)
  {
    display.fillRect(placeX6-1, placeY6-1, 20, 9, INVERSE);//Waveform +
FFT on
  }

  display.setCursor(0,placeY5 +13);
  display.println("Uptime:");
  display.setCursor(0,placeY5 +26);
  display.println(String(days)+ "d "+ String(hrs)+ "h "+ String(mins)+
"min "+ String(secs)+ "s");
  delay(5);
  display.display();
}
```

Lisa 6 Kasutajapoolse veebilehe programmikood

```
<!-- Code inspired by: https://randomnerdtutorials.com/esp8266-nodemcu-
plot-readings-charts-multiple/ -->

<!DOCTYPE html>
<html>

  <head>
    <title>ESP32 IOT DASHBOARD</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/png" href="favicon.png">
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="https://code.highcharts.com/highcharts.js"></script>
    <!--<script src="highcharts.js"></script>-->
  </head>

  <body>
    <div class="topnav">
      <h1>Nutikas energiamõõtur</h1>
    </div>
    <div class="content">
      <div class="card-grid">
        <div class="card">
          <p class="card-title">Data from device</p>
          <p id = "uptime"> Device uptime: </p>
            <div>
              <div class='child float-left-child' id="dataTableDiv">
                <table id="dataTable" border="1">
                  <thead>
                    <th>Parameeter</th>
                    <th>Vaartus</th>
                  </thead>
                  <tbody id="insertdataTable">
                  </tbody>
                </table>
              </div>
              <div class='child float-left-child'>
                <input type="button" name="button" value="Run Calibration"
onclick="sendCalibRequest()">
                %BUTTONPLACEHOLDER%
              </div>
            </div>
            <p ></p>
        </div>
        <div class="card">
          <p class="card-title">Voltage/Current Chart</p>
          <div id="chart-VW" class="chart-container"></div>
        </div>
        <div class="card">
          <p class="card-title">Voltage FFT Chart</p>
          <div id="chart-VFFT" class="chart-container"></div>
        </div>
        <div class="card">
          <p class="card-title">Current FFT Chart</p>
          <div id="chart-IFFT" class="chart-container"></div>
```

```
        </div>
        <div class="card">
          <p class="card-title">Trigger Chart, currently only for
current</p>
          <form name="myform" action="" method="get">
            Enter nr of measured points, trigger level and timeout in
textboxes: <br>
            <input type="number" name="inputbox1" value="150" id="nr">
            <input type="number" name="inputbox2" value="0.25"
id="tlevel">
            <input type="number" name="inputbox3" value="10000"
id="timeout">
            <input type="radio" name="choice" value="1" id="radvoltage">
            <label for="radvoltage">Trigger on voltage</label>
            <input type="radio" name="choice" value="0" id="radcurrent">
            <label for="radcurrent">Trigger on current</label>
            <input type="button" name="button" value="Trigger"
onclick="sendTriggerRequest()">
          </form>
          <div id="chart-Trigger" class="chart-container"></div>
        </div>

      </div>

    </div>
    <script src="script.js"></script>
  </body>
</html>
```

Lisa 7 Veebilehe skript

```
// Code inspired by: https://randomnerdtutorials.com/esp8266-nodemcu-plot-readings-charts-multiple/

// Get current sensor readings when the page loads
window.addEventListener('load', getReadings);

// Create voltage and current waveform Chart
var chartVW = new Highcharts.Chart({
  chart: {
    renderTo: 'chart-VW',
    marginRight: 80
  },
  series: [{
      name: 'Voltage',
      type: 'line',
      color: '#101D42'
    },
    {
      name: 'Current',
      type: 'line',
      color: '#d9000b',
      yAxis: 1
    },
  ],
  title: {
    text: undefined
  },
  xAxis: {
    labels: {
      format: '{value} ms'
    },
    minRange: 5,
    title: {
      text: 'Time, ms'
    },
    accessibility: {
      rangeDescription: 'Range: 0 to 42 ms.'
    }
  },
  yAxis: [{
    title: {
      text: 'Voltage, V'
    }
  }, {
    opposite: true,
    title: {
      text: 'Current, A'
    }
  }],
  credits: {
    enabled: false
  }
});
```

```
// Create voltage FFT Chart
var chartVFFT = new Highcharts.Chart({
  chart: {
    renderTo: 'chart-VFFT'
  },
  series: [{
    name: 'Amplitude',
    type: 'line',
    color: '#101D42'/*,
    marker: {
      symbol: 'circle',
      radius: 3,
      fillColor: '#101D42',
    }*/
  }, ],
  title: {
    text: undefined
  },
  xAxis: {
    labels: {
      format: '{value} Hz'
    },
    minRange: 5,
    title: {
      text: 'Frequency, Hz'
    },
    accessibility: {
      rangeDescription: 'Range: 0 to 512 Hz.'
    }
  },
  yAxis: {
    title: {
      text: 'Amplitude, V'
    }
  },
  credits: {
    enabled: false
  }
});

// Create current FFT Chart
var chartIFFT = new Highcharts.Chart({
  chart: {
    renderTo: 'chart-IFFT'
  },
  series: [{
    name: 'Amplitude',
    type: 'line',
    color: '#d9000b'
  }, ],
  title: {
    text: undefined
  },
  xAxis: {
    labels: {
      format: '{value} Hz'
```

```
    },
    minRange: 5,
    title: {
      text: 'Frequency, Hz'
    },
    accessibility: {
      rangeDescription: 'Range: 0 to 512 Hz.'
    }
  },
  yAxis: {
    title: {
      text: 'Amplitude, A'
    }
  },
  credits: {
    enabled: false
  }
});

// Create current FFT Chart
var chartTrigger = new Highcharts.Chart({
  chart: {
    renderTo: 'chart-Trigger'
  },
  series: [{
    name: 'Voltage',
    type: 'line',
    color: '#101D42'
  },
  {
    name: 'Current',
    type: 'line',
    color: '#d9000b',
    yAxis: 1
  },
  ],
  title: {
    text: undefined
  },
  xAxis: {
    labels: {
      format: '{value} ms'
    },
    minRange: 5,
    title: {
      text: 'time, ms'
    },
    accessibility: {
      rangeDescription: 'Range: 0 to 40 ms.'
    }
  },
  yAxis: [{
    title: {
      text: 'Voltage, V'
    }
  }, {
```

```
    opposite: true,
    title: {
      text: 'Current, A'
    }
  }],
  credits: {
    enabled: false
  }
});

//Plot values in the charts
function plotjson(jsonValue, nr) {

  //Clearing charts for new data, 0 and 1 for waveform chart, 2 for
voltage FFT, 3 for current FFT, 4 and 5 for trigger waveform
  if (Number(nr) == 0 || Number(nr) == 1) {
    chartVW.series[nr].setData([]);
  } else if (Number(nr) == 2) {
    chartVFFT.series[0].setData([]);
  } else if (Number(nr) == 3) {
    chartIFFT.series[0].setData([]);
  } else if (Number(nr) == 4) {
    chartTrigger.series[0].setData([]);
  } else if (Number(nr) == 5) {
    chartTrigger.series[1].setData([]);
  }

  var keys = Object.keys(jsonValue);
  console.log(nr);
  //console.log(keys.length);

  for (var i = 0; i < keys.length; i++) {
    const key = keys[i];
    var y = Number(jsonValue[key]);

    if (Number(nr) == 0) {
      chartVW.series[0].addPoint([i * 0.25, y], false, false, true);
    } else if (Number(nr) == 1) {
      chartVW.series[1].addPoint([i * 0.25, y], false, false, true);
    } else if (Number(nr) == 2) {
      chartVFFT.series[0].addPoint([i*1.953125 , y], false, false, true);
    } else if (Number(nr) == 3) {
      chartIFFT.series[0].addPoint([i*1.953125 , y], false, false, true);
    } else if (Number(nr) == 4) {
      chartTrigger.series[0].addPoint([i * 0.25, y], false, false, true);
    } else if (Number(nr) == 5) {
      chartTrigger.series[1].addPoint([i * 0.25, y], false, false, true);
    }

  }

  if (Number(nr) == 0 || Number(nr) == 1) {
    chartVW.redraw();
  } else if (Number(nr) == 2) {
    chartVFFT.redraw();
  } else if (Number(nr) == 3) {
```

```
    chartIFFT.redraw();
  } else if (Number(nr) == 4 || Number(nr) == 5 ) {
    chartTrigger.redraw();
  }

}

// Function to get current readings to the webpage when it loads for the
first time and create table for gathered data
function getReadings() {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      console.log(this.responseText);
    }
  };
  xhr.open("GET", "/readings", true);
  xhr.send();
  tableCreate();
}

if (!!window.EventSource) {
  var source = new EventSource('/events');

  source.addEventListener('open', function(e) {
    console.log("Events Connected");
  }, false);

  source.addEventListener('error', function(e) {
    if (e.target.readyState != EventSource.OPEN) {
      console.log("Events Disconnected");
    }
  }, false);

  source.addEventListener('message', function(e) {
    console.log("message", e.data);
  }, false);

  source.addEventListener('new_voltage', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 0);
  }, false);
  source.addEventListener('new_current', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 1);
  }, false);
  source.addEventListener('new_VFFT', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 2);
  }, false);
  source.addEventListener('new_IFFT', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 3);
  }, false);
  source.addEventListener('new_data', function(e) {
    var myObj = JSON.parse(e.data);
```

```
    tableUpdate(myObj);
  }, false);
  source.addEventListener('new_trigger_current', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 5);
  }, false);
  source.addEventListener('new_trigger_voltage', function(e) {
    var myObj = JSON.parse(e.data);
    plotjson(myObj, 4);
  }, false);
  source.addEventListener('uptime', function(e) {
    var myObj = parseInt(e.data, 10);
    msToTime(myObj);
  }, false);
}

//Create table for data
function tableCreate() {
  var table = document.getElementById('insertdataTable');
  for (var i = 0; i < 15; i++) {
    var x = document.getElementById('insertdataTable').insertRow(i);
    for (var c = 0; c < 2; c++) {
      var z = x.insertCell(c);
    }
    table.rows[i].cells[0].innerHTML = i;
    table.rows[i].cells[1].innerHTML = i;
  }
}

//Update table values
function tableUpdate(jsonValue) {
  var table = document.getElementById('insertdataTable');
  var keys = Object.keys(jsonValue);
  for (var i = 0; i < 15; i++) {
    const key = keys[i];
    var y = Number(jsonValue[key]);
    table.rows[i].cells[0].innerHTML = key;
    table.rows[i].cells[1].innerHTML = y;
  }
}

//Send button command to MCU
function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked)
  {
    xhr.open("GET", "/update?output="+element.id+"&state=1", true);
  }
  else
  {
    xhr.open("GET", "/update?output="+element.id+"&state=0", true);
  }
  xhr.send();
}

//Send waveform capture request
```

```
function sendTriggerRequest () {
  var xhr = new XMLHttpRequest();
  const rbs = document.querySelectorAll('input[name="choice"]');
  let selectedValue;
  for (const rb of rbs) {
    if (rb.checked) {
      selectedValue = rb.value;
      break;
    }
  }
  var showData1= document.getElementById("nr").value;
  var showData2= document.getElementById("tlevel").value;
  var showData3= document.getElementById("timeout").value;

  xhr.open("GET", "/update?count="+showData1+"&level=" + showData2
+"&timeout=" + showData3  + "&select=" + selectedValue, true);
  xhr.send();
}

//Convert ms to seconds, minutes, hours and days
function msToTime(s) {
  var secs = s % 60;
  s = (s - secs) / 60;

  var mins = s % 60;
  s = (s - mins) / 60;

  var hrs = (s) % 24;
  s = (s - hrs) / 24;

  var days = s;
  document.getElementById("uptime").textContent = "Device uptime "+ days +
" days, " + hrs + " hours, " + mins + " minutes, " + secs + " seconds";
}

//Send calibration request
function sendCalibRequest() {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      console.log(this.responseText);
    }
  };
  xhr.open("GET", "/calib", true);
  xhr.send();
}
```