



TIETO- JA SÄHKÖTEKNIKAN TIEDEKUNTA
ELEKTRONIIKAN JA TIETOLIIKENNETEKNIIKAN TUTKINTO-OHJELMA

KANDIDAATINTYÖ

GPIB-ohjelmoitavan jännitelähteen käyttö puolijohteen I-V -mittauksiin

Tekijä Santeri Paavola

Ohjaaja Timo Rahkonen

Toukokuu 2026

Paavola S. (2026) GPIB-ohjelmoitavan jännitelähteen käyttö puolijohteen I-V -mittauksiin. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 27 s.

TIIVISTELMÄ

Puolijohteiden virta-jännitekäyriä mitataan yleensä ns. curve tracer -laitteella, joka pystyy pyyhkäisemään jännitteitä ja mittaamaan virrat. Pienikokoisten komponenttien itseislämpöämisen mittaamisen takia mittaussnopeuden täytyy olla nopea, yhteen mitaustapahtumaan saa kulua mikrosekunteja, maksimissaan millisekunteja. Tässä työssä kokeiltiin voiko suurikokoisen diskreettipuolijohteen itseislämpöämistä mitata käyttäen GPIB-ohjattua jännitelähdettä. Hitaan pyyhkäisyn ja nopeasti suoritettun pyyhkäyksen välinen ero I-V-käyrässä oli selvästi havaittavissa.

Avainsanat: semiconductor I-V characterization, self-heating effects, GPIB, pulsed I-V measurements

Paavola S. (2026) Application of a GPIB-programmable power supply for semiconductor I-V measurements. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering, Bachelor's Thesis, 27 p.

ABSTRACT

Semiconductors current–voltage characteristic are conventionally measured with a curve tracer, which is able to sweep voltages and measure the current. Due to the self heating of the component under measurement, the measurement speed has to be fast. Maximum time for each measurement event should be in the microseconds and in it's in the maximum milliseconds. This bachelor's work tested whether the self-heating of a large discrete semiconductor could be measured using a GPIB-controlled power supply. The difference between a slow sweep and fast sweep on the I-V curve was clearly noticeable.

Keywords: semiconductor I-V characterization, self-heating effects, GPIB, pulsed I-V measurements

SISÄLLYSLUETTELO

TIIVISTELMÄ

ABSTRACT

SISÄLLYSLUETTELO

ALKULAUSE

LYHENTEIDEN JA MERKKIEN SELITYKSET

1	Johdanto	7
2	GPIB	8
2.1	GPIB-komentojen syntaksi	8
2.2	Ohjaus tietokoneella.....	9
2.2.1	Ohjaimen asetukset	10
2.3	Muisti	11
2.4	Synkronointi.....	11
2.4.1	Rekisterit synkronoinnissa	12
3	Lämpötilan vaikutus diodissa.....	14
3.1	Diodin virtayhtälö	14
3.2	Sarjaresistanssin vaikutus diodin virtajännitekuvaajaan	15
4	Ohjelmisto	16
4.1	Matalan tason metodit.....	16
4.1.1	readString	17
4.1.2	isSrqBitSet	17
4.1.3	writeCmd.....	18
4.1.4	queryCmd.....	18
4.1.5	openDevice.....	19
4.2	Mittausdatan ottavat metodit.....	19
4.2.1	getCoolMeasurements.....	19
4.2.2	getHotMeasurements.....	20
4.2.3	psuWriteAndLog.....	21
5	Tulokset	22
5.1	UG4D ja 1N4007	22
6	Pohdinta.....	24
6.1	Jäähtymisen viive.....	24
6.2	Ohjelmisto.....	25
7	Yhteenveto	26
8	LÄHTEET	27

ALKULAUSE

Tämä kandidaatintyö oli erittäin mieluisa toteuttaa, sillä ohjelmoinnin ja fyysisen "raudan" kombinaatiosta syntyy aina jotain mielenkiintoista. Työn teoria ja toteutus antoi hyvän mahdollisuuden perehtyä mittalaitteiden automatisaatioon, jota tulen varmasti tarvitsemaan jo myös omissa projekteissani. Isot kiitokset Timo Rahkoselle kandidaatintyön ohjauksesta.

Oulussa 30.04.2026

Santeri Paavola

LYHENTEIDEN JA MERKKIEN SELITYKSET

t	aika
q	alkeisvaraus
k_B	Boltzmannin vakio
D_p ja D_n	diffuusiokerroin
L_p ja L_n	diffuusiopituus
I_D	diodivirta
I_0	diodin saturaativirta
n_p	enemmistövarauksenkuljettajien tasapainokonsentraatio
E_G	energiarako
n	ideaalisuustekijä
k	lämmönsiirtymiskerroin
T	lämpötila
A	pinta-ala
X_{TI}	suunnitteluparametri
p_n	vähemmistövarauksenkuljettajien tasapainokonsentraatio

1 Johdanto

Puolijohdekomponenttien pääasiallinen rakenneosana on pn-liitos, jonka sähköiset ominaisuudet vaikuttavat laajasti käytetyn komponentin toimintaan ja sen käyttökohteisiin. Eräs pn-liitokseen vaikuttava tekijä on liitoksen lämpötila, joka on osatekijä pn-liitoksen kynnysjännitteeseen ja komponentin kykyyn johtaa virtaa. Kun pn-liitos kuluttaa tehoa, liitos lämpenee ja siten muokkaa komponentin toimintaa. Komponentin fyysinen koko vaikuttaa aikaan, joka sillä kestää saavuttaa sille ominainen terminen tasapaino. Esimerkiksi piirilevyyn juotetulla diskreettikomponentilla tähän voi mennä minuutteja, kun taas pienikokoisimmilla transistoreilla siihen voi mennä vain mikrosekunteja.

Yleisesti ottaen yllä mainitut mittaukset suoritetaan *curve tracer*:lla, jolla voidaan analysoida puolijohdekomponenttien ominaisuuksia pyyhkäisemällä jännitteitä ja mittaamalla komponenttien virtoja. Laitteella voidaan siten saada mitattavan komponentin virtajännitekuvaaja. Curve tracerilla voidaan myös yleensä toteuttaa pulssitettuja mittasarjoja, joissa mitattavan komponentin ylle asetetaan tietty jännite ja sen läpi menevä virta mitataan *nopeasti*, jotta komponentin itselämpeäminen ei pääsisi muuttamaan tuloksia [1]. Kuitenkin kyseisen laitteen puutteen vuoksi tässä kandidaatintyössä tutkittiin, olisiko Hewlett Packard E3631A DC virtalähde tarpeeksi nopea itselämpeämisen havaitsemiseen diodeissa.

Työssä käytetty virtalähde, kuten myös monet muut vanhemmat virtalähteet, eivät välttämättä tue tarpeeksi edistyneitä automaatio-ominaisuuksia, puhumattakaan liitännöistä, joilla laitetta voitaisiin ohjata nykyaikaiselta tietokoneelta. Siitä huolimatta osa uusista mittalaitteista sekä yhä useampi vanhemmista mittalaitteista tukevat *General Purpose Interface Bus* -väylää laitteiden automatisaatiossa. Työssä virtalähdettä ohjelmoitiin ajamaan diodia, jonka virta- ja jänniteluemat tallennettiin muistiin käyttäen samaa virtalähdettä. Virtalähteen ohjelmointi toteutettiin *Standard Commands for Programmable Instruments* -komennoilla hyödyntäen virtalähteen tukemaa GPIB-väylää. Koska mittasarjojen ohjaus oli kätevä toteuttaa tietokoneelta käsin, työssä valittiin Prologix GPIB-USB -ohjain virtalähteen ja tietokoneen väliseen kommunikaatioon. Koko systeemiä ohjattiin tietokoneelta graafisesta käyttöliittymästä Linux-käyttöympäristössä.

2 GPIB

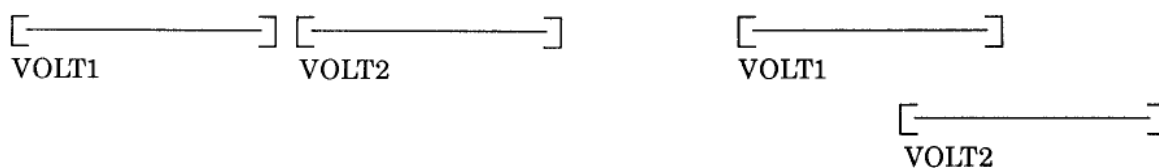
IEEE-488, eli General Purpose Interface Bus (GPIB) on standardoitu digitaalinen käyttöliittymä, jolla pystyy ohjaamaan GPIB-väylään yhdistettyjä laitteita. Hewlett-Packard kehitti GPIB:n (tuolloin vielä nimellä HP-IB) vuonna 1972 muun muassa mittainstrumenttien, massamuistien ja niiden ohjainten yhdistämiseksi yhdeksi systeemiksi, jota käyttäjä pystyisi ohjaamaan yhtenä kokonaisuutena. Liitäntää käytetään yleisesti mittalaitteiden kuten virtalähteiden tai oskiloskooppien ohjaamiseen ja niiden toiminnan automatisaatioon. Vaikka teknologia on suhteellisen vanhaa, monet yhä käytössä olevat laitteet mahdollistavat liitännän hyödyntämisen.

GPIB käyttää 8-bittistä rinnakkaismuotoista väylää ja suurin nopeus IEEE 488.1 standardoitulle versiolle on 1 Mbps, sekä uudemmalla standardilla (IEEE 488.2) 8 Mbps. Molempien versioiden GPIB-laitteita pystyy kuitenkin käyttämään samassa järjestelmässä, kunhan 15 laitteen raja ei ylitä.

2.1 GPIB-komentojen syntaksi

Alkuperäinen IEEE 488.1 GPIB -standardi käsittelee liitäntää ainoastaan laitetasolla, eikä anna määritelmiä eri mittalaitteiden komennoille. Kuitenkin standardin uudempi versio IEEE 488.2 käsittelee yleisimpiä komentoja laitteille. Laitekohtaiset, esimerkiksi lähtöjännitteen tasoa säättävät komennot, käyttävät SCPI-komentoja (Standard Commands for Programmable Instruments). Esimerkiksi virtalähteen tapauksessa komento `VOLTage 0.0` käskää virtalähteen asettaa lähtöjännite nolnaan volttiin. Komentoja voi myös antaa lyhyessä muodossa, jolloin pienillä kirjaimilla kirjoitetut kirjaimet voidaan jättää komennosta kokonaan pois, esimerkiksi `VOLT 0.0`. Komennot, joihin virtalähteen halutaan vastaavan, sisältävät kysymysmerkin komennon lopussa, esimerkiksi `MEAS:CURR?` mittaa virran arvon. Käskyjä voi myös ketjuttaa puolipisteiden avulla, mikä helpottaa useampien komentojen lähettämistä yhdellä kirjoituskerralla, esimerkiksi `VOLT 0.0;OUTP OFF`, mikä käskää virtalähdettä asettamaan lähtöjännitteen nolnaan volttiin ja pois päältä. IEEE 488.2 -komennot eroavat yleisesti käytettävistä SCPI-komennoista, mikä ilmenee muun muassa asteriski-merkin käytöstä komennon edessä. Esimerkiksi `*IDN?` on puhdas IEEE-488.2 -standardin komento, joka käskää laitetta kertomaan laitteen valmistajan.

GPIB-laitteet voivat toteuttaa IEEE 488.2 -standardin mukaisesti komentoja sarja- tai rinnakkaismuodossa, minkä jokainen laitevalmistaja voi määritellä itse. Tästä syystä esimerkiksi yllä oleva puolipisteellä ketjutettu komento saattaisi valmistajasta riippuen lähteä asettamaan lähtöä pois päältä jo silloin, kun jännitettä käsittelevä komento ei olisi vielä valmistunut. Mikäli käytettävä GPIB-laite toteuttaa komentoja rinnakkaismuotoisina, voidaan komennot suorittaa täysin erillään toisistaan, lisäämällä niiden väliin `*WAI`-lipun. Lipulla voidaan käskää laite odottamaan, kunnes sitä edeltävät komennot ovat suoritettu, esimerkiksi komennon `VOLT 1.0;*WAI;MEAS:CURR?` tulisi odottaa, kunnes lähtöjännite on asetettu ja vasta sitten lähteä suorittamaan virran mittausta [2].



Kuva 1. IEEE 488.2 12.2.1. Kaksi GPIB komentosarjaa, vas. sekventiaalinen. [2]



Kuva 2. *WAI-lipun käyttö eri GPIB komentosarjoissa. [2]

Hewlett Packard E3631A ei hyödynnä *WAI -lippua kuin ainoastaan *Trigger*-komennoissa [3], esimerkiksi `TRIG:SOUR BUS;*TRG;*WAI;*TRG;*WAI`. Tällöin virtalähde voidaan määrittellä asettamaan ennalta määritellyt virta- ja jännitemaksimit lähtöön, virtalähteen saadessa trigauskomento GPIB-väylältä. Kuitenkaan trigauskomentoja ei käytetty työssä, vaan lähes kaikki lähtöön vaikuttavat parametrit asetettiin korkeamman tason `APPLY <channel>, <voltage>, <current>` -komennoilla, mikä vastaa samaa kuin `INSTRUMENT:SELECT <channel>; VOLTAGE <voltage>; CURRENT <current>`.

2.2 Ohjaus tietokoneella

GPIB-kommunikaation ohjaus tietokoneelta käsin toteutettiin Prologix GPIB-USB-ohjaimella. Ohjain esittäytyy tietokoneelle USB-sarjalaitteena, jota voidaan ajaa suoraan muun muassa terminaaliemulaattorin avulla. Ohjain tarvitsee toimiakseen FTDI-ajurin, joka on saatavilla sekä Linuxilla että Windowsilla. Ajuri kuuluu useampien Linux-jakeluiden ytimiin nimellä *ftdi_sio*, joka automaattisesti luo kytkettyjen USB-sarjalaitteiden laitetiedostot [4]. Valittu ohjain myös yksinkertaistaa ohjelmointia, sillä yleiset sarjalaitteiden ominaisuudet, kuten baudinopeuden tai databittien määrän, voi jättää huomioimatta [5].

Jotta Prologix-ohjaimen pystyy ottamaan yhteyden tietokoneelta, pitää löytää sen oikea laitetiedosto yhteyden muodostamista varten. Terminaalikomento `ls -l /dev/ttyUSB*` listaa kaikki USB-sarjalaitteet terminaaliin numeroituna nolasta ylöspäin. Jos tietokoneeseen on liitetty useampi USB-sarjalaitte, komennossa oleva asetus `-l` kertoo ajan, jolloin USB-laite liitettiin tietokoneeseen. Tällöin käyttäjä voi päätellä oikean laitetiedoston.

Työssä käytetyllä Linux Fedora 43 -jakelulla ennen Prologix-ohjaimen käyttöä istunnon käyttäjälle täytyi antaa täydet oikeudet sarjaporttien käyttöön. Tämä suoriutui terminaalikomennolla `sudo usermod -a -G dialout $USER`, jossa asetus `-a` (`-append`) `-G` (`-groups`) lisää ryhmään `dialout` argumentin `$USER` (aktiivisen istunnon käyttäjä). Tämän jälkeen käyttöjärjestelmä käynnistettiin uudelleen ja oikeudet tarkistettiin kirjoittamalla terminaaliin `grep dialout /etc/group`, joka tulosti terminaaliin aktiivisen käyttäjän käyttäjänimen, tarkoittaen toimivia oikeuksia laitetiedoston käyttämiseksi.

Prologix-ohjaimen oikea toiminta tarkistettiin käyttämällä sarjakommunikaatio-ohjelmistoa *minicom*. Minicom on tekstipohjainen, terminaalissa käytettävä ohjelma, jolla pystyy lähettämään ja vastaanottamaan merkkijonoja sarjalaitteiden kanssa. Aluksi minicom asennettiin käytettävälle Fedora-jakelulle komennolla `sudo dnf install minicom.x86_64`. Yhteys USB-väylään litettyyn Prologix-ohjaimen saatiin komennolla `minicom -D /dev/ttyUSB0`, jonka ainoa argumentti `-D` avaa annetun asetuksen `/dev/ttyUSB0` laitetiedoston ohjelman käytettäväksi. Ennen komentojen kirjoittamista minicom konfiguroitiin tulostamaan terminaaliin painetut kirjaimet pikanäppäinyhdistelmällä `Ctrl+A, E`, jonka jälkeen ohjaimelle voitiin lähettää komentoja.

2.2.1 Ohjaimen asetukset

GPIB-protokollassa on käytössä kolmenlaisia laitetyppejä: kontrolleri (controller), kuuntelija (listener) ja puhuja (talker). Kontrollereita voi olla aktiivisena vain yksi per järjestelmä ja ne ohjaavat tiedonsiirtoa väylällä antamalla käskyjä kuuntelijoille ja puhujille. Eri kuuntelijat voivat vastaanottaa samaa dataa väylältä kontrollerin käskiessä niitä. Puhujat puolestaan voivat lähettää dataa väylälle yksi kerrallaan, kontrollerin käskiessä niitä [6]. Prologix-ohjain pystyy esittämään itsensä joko kontrollerina tai laitemoodissa (device mode), jossa se voi olla joko puhuja tai kuuntelija [5].

Jotta GPIB-väylän datankulkua voitaisiin ohjata tietokoneelta, ohjain on asetettava GPIB-väylän kontrolleriksi. Ohjain erottelee omat komennot muista GPIB-komennoista kahden plusmerkin avulla. Kuitenkin ennen muita komentoja, aluksi työssä testattiin itse ohjaimen toimivuus Prologix-ohjaimen omalla komennolla `++ver`, johon se vastasi onnistuneesti tulostamalla terminaaliin *Prologix GPIB-USB Controller version 6.107*. Tämän jälkeen ohjain asetettiin kontrolleriksi komennolla `++mode 1`.

GPIB-kontrolleri pystyy ohjaamaan eri laitteita niiden uniikkien osoitteiden avulla. GPIB-laitteet ja kontrolleri erottelevat eri laitteille kuuluvat komennot uniikkien osoitteiden avulla. Tällöin kontrolleri voi lähettää tietyt komennot ainoastaan osalle systeemin laitteista. Tässä työssä käytettiin vain kontrolleria ja virtalähdettä, mutta oikea osoite on silti asetettava oikein. Ohjaimen osoite asetettiin komennolla `++addr <N>`, jossa luvun N on oltava sama kuin virtalähteelle asetettu osoite. Virtalähteen osoitteen voi vaihtaa suoraan sen fyysisestä käyttöliittymästä.

Prologix-ohjain tallentaa sille annettuja asetuksia ajoittain sen omaan EEPROM-muistiin. Kuitenkin ohjaimen manuaali suosittelee automaattisen tallentamisen asettamista pois päältä EEPROM:n keston kannalta [5]. Lisäksi työssä automaattitallennuksista ei ollut hyötyä, jolloin ne voitiin asettaa pois komennolla `++savecfg 0`.

Ohjaimelle asetettu `++read_tmo_ms <N>` viive määrittää Prologix-ohjaimen GPIB-laitteelta vastaanottaman datan maksimaalista kirjainten välistä viivettä. Mikäli viive on liian lyhyt, voi lukuyritys, kuten `++read eoi`, tapahtua liian nopeasti. Tällöin data jää saamatta, ja GPIB-komentojen suoritus saattaa sekoittua järjestelmässä. Työssä käytetty Hewlett Packard E3631A lisäsi jokaiseen sen lähettämään viestiin End Of Instruction -merkin, minkä vuoksi `read_tmo_ms` viiveellä ei ollut merkitystä, ja se asetettiin ohjaimen sallimaan maksimiarvoon 3000 ms.

Taulukko 1. Muita työssä käytettyjä GPIB-ohjainasetuksia

<code>++auto 0</code>	Ohjain ei lue GPIB-laitteelta automaattisesti jokaisen sille lähetetyn komennon jälkeen.
<code>++eoi 1</code>	Asettaa jokaisen GPIB-komennon perään End Of Instruction (EOI) -merkin, joka yleensä vaaditaan GPIB-laitteilta.
<code>++eos 3</code>	Ohjain ei lisää CR/NL-merkkejä GPIB-komennon perään.

Työssä käytetty Hewlett Packard E3631A käyttää IEEE 488.2 -versiota [3]. Virtalähteen ja adapterin välinen toiminta varmistettiin lähettämällä virtalähteelle GPIB 488.2 -standardin mukainen komento `*IDN?`, johon virtalähteen tulisi vastata kertomalla mm. sen malli. Kun toiminta oli tarkistettu, voitiin virtalähdettä ohjata virtalähteelle ominaisilla SCPI-komennoilla. Virtalähteelle voidaan lähettää komento `SYSTEM:VERSION?`, joka kertoo sen SCPI version. Työssä käytetyssä virtalähteessä tämä oli versio 1995.0, mutta SCPI-versiolla ei ollut ohjelmoinnin kannalta juuri mitään merkitystä, sillä kaikki tarvittavat SCPI-komennot olivat saatavilla virtalähteen manuaalissa.

2.3 Muisti

Rajoittava tekijä, etenkin vanhemmilla GPIB-laitteilla, on niiden muistin rajallinen koko. HP E3631A virtalähteen komentojen tulopuskurin koko on noin 100 ASCII-merkkiä [3]. Noin sata ASCII merkkiä on kohtuullisen paljon, mutta lähetettäessä pitkiä komentosarjoja tai paljon eri komentoja nopeasti lukupuskuri voi ylivuotaa. Kokoa voidaan mitigoida muun muassa lähettämällä virtalähteelle kappaleessa 2.1 esitettyjä lyhempiä muotoja eri komennoille.

Hewlett-Packard E3631A virtalähde pystyy tallentamaan ainoastaan yhden mittaustuloksen sen lukupuskuriin kerrallaan. Tästä johtuen virta- ja jännitemittaukset virtalähteeltä täytyy suorittaa erillisillä komennoilla, jolloin jälkimmäinen niistä laahaa ensimmäisen jäljessä. Lisäksi on mahdollista kohdata ongelmia mittauksissa esimerkiksi tapauksissa, joissa otetaan:

1. Virran mittauskomento
2. Virtalähteen vastauksen lukeminen
3. Jännitteen mittauskomento

Edellä olevista komennoista virtalähde saattaa antaa kohdassa 3 virhekoodin *410 Query INTERRUPTED*. Virhe johtuu siitä, että tulos ensimmäisestä mittauskomennosta hetkellä 1 ei ehdi valmistua ennen lukuyritystä hetkellä 2, jolloin hetkellä 3 mittauspyynnöt menevät päällekkäin, jota Hewlett-Packard E3631A ei tue. Muun muassa eräs ad hoc -ratkaisu virhekoodin 410 Query Interrupted korjaamiseksi on asettaa Prologix-ohjaimen kirjaisinten välinen `++read_tmo_ms 3000` -viive maksimaaliseen arvoon, jolloin (ainakin teoriassa) työssä käytetty virtalähde pystyisi toteuttamaan täynnä olevan lukupuskurin verran komentoja ilman virhekoodia 410. Kuitenkin parempi ratkaisu on synkronoida komentojen suorittamisen tila tietokoneen ja virtalähteen välillä, jolloin kaikki komentosarjat voidaan toteuttaa siten, että laitteiden tila on käyttäjälle / ohjelmistolle koko ajan tiedossa.

2.4 Synkronointi

Työssä käytetyn Hewlett-Packard E3631A virtalähteen synkronointi hoidettiin hyödyntäen virtalähteen *SCPI Status System*:iä, jonka käyttö perustuu IEEE-488.2 -komentojen pohjalle. IEEE-488 service request -signaali (SRQ) on tapa ilmoittaa GPIB-väylälle mittalaitteen valmiudesta, jolloin systeemin aktiivinen kontrolleri voi pollata kaikki laitteet erikseen, kunnes SRQ-bitin asettanut laite löydetään. Tässä tapauksessa pollaus oli kohtuullisen triviaalia systeemin koon vuoksi. Vastaavasti SRQ-bitin asettaminen pois päältä suoritettiin riippuen siitä, mikä yllä mainituista rekistereistä oli sen asettanut päälle.

Prologix-ohjain pystyy lukemaan dataa GPIB-väylältä joko komennolla `++read` tai hyödyntäen IEEE-488 *service request* -signaalia (SRQ) komennolla `++srq`. Käytettäessä ohjaimen toimintoa `++read eos`, voitaisiin tietokoneen puolella varmistua siitä, että virtalähde on prosessoanut esimerkiksi jännitteen asettavan komennon ennen muiden komentojen suorittamista vastaavanlaisesti: `VOLT <N>;*WAI;MEAS:VOLT?`, joka luettaisiin virtalähteellä komennolla `++read eos`. Kuitenkin hyöty SRQ-signaalissa on, ettei virtalähteen tarvitse palauttaa mitään lukupuskuriinsa, jotta komentojen valmistuminen virtalähteen puolella voitaisiin havaita Prologix-ohjaimelta, jolloin yllämainittu komentosarja voitaisiin lukea sarjalla: `VOLT <N>;*WAI;*OPC`, jonka jälkeen käytettäisiin Prologix-ohjaimen komentoa `++srq`. Tällöin ohjainta voitaisiin vain pollata tietokoneelta niin kauan, kunnes SRQ-bitti olisi asetettu aktiiviseksi.

2.4.1 Rekisterit synkronoinnissa

Hewlett-Packard E3631A virtalähteen *Status Byte Summary* -rekisteri sisältää virtalähteen muiden rekisterien tilat. Kyseisen rekisterin voi määrittellä aktivoimaan GPIB-väylän SRQ-bitti aktiiviseksi, mikäli käyttäjän konfiguroima bitti rekisterissä asettuu binääriarvoon 1. Nämä SRQ:n aktivoimat bitit voidaan määrittellä rekisteriin komennolla *SRE <N> (*Status Byte enable register*). Bittejä voidaan aktivoida myös useampia yhdellä kerralla, esimerkiksi aktivoitaessa MAV- ja ESB-bitit komennolla *SRE 48.

Taulukko 2. Bit Definitions - Status Byte Summary Register [3]

Bit	Decimal Value	Definition
0-2 Not Used	0	Always set to 0.
3 QUES	8	One or more bits are set in the questionable status register (bits must be “enabled” in the enable register).
4 MAV	16	Data is available in the power supply output buffer.
5 ESB	32	One or more bits are set in the standard event register (bits must be “enabled” in the enable register).
6 RQS	64	The power supply is requesting service (serial poll).
7 Not Used	0	Always set to 0.

Työssä hyödynnettiin MAV-bittiä (*Message Available bit*) sekä ESB-bittiä (*Standard Event Summary bit*). MAV-bitti raportoi kaikki virtalähteen lukurekisteriin tallennetut arvot, eli SRQ-signaali asettuu GPIB-väylällä binääriarvoon 1, mikäli virtalähde palauttaa sen lukurekisteriin jonkin tuloksen. MAV-bittiä voidaan hyödyntää esimerkiksi, kun halutaan tietää mittauksen tila käytettäessä komentoa MEAS:VOLT?. MAV-bitin voi nollata joko komennolla *CLS tai lukemalla arvo virtalähteen lukurekisteristä, mikä tarkoittaa, että virtalähteeltä voidaan yhdellä lukuoperaatiolla ottaa data talteen sekä viimeistellä synkronointi.

Taulukko 3. Bit Definitions - Standard Event Register [3]

Bit	Decimal Value	Definition
0 OPC	1	Operation Complete. All commands prior to and including an *OPC command have been executed.
1 Not Used	0	Always set to 0.
2 QYE	4	Query Error. The power supply tried to read the output buffer but it was empty, or a new command line was received before a previous query had been read, or both the input and output buffers are full.
3 DDE	8	Device Error. A self-test or calibration error occurred.
4 EXE	16	Execution Error. An execution error occurred.
5 CME	32	Command Error. A command syntax error occurred.
6 Not Used	0	Always set to 0.
7 PON	128	Power On. Power has been turned off and on since the last time the event register was read or cleared.

Status Byte Summary -rekisterin ESB-bitti asettuu päälle *Standard Event* -rekisteriin asetettujen bittien mukaisesti. Standard Event -rekisterin bitit konfiguroidaan samalla tavalla kuin Status Byte Summary -rekisteri, esimerkiksi *ESE 1. Työssä hyödynnetty Standard Event -rekisterin kohta oli OPC-bitti, joka aktivoituu, kun virtalähde suorittaa *OPC -komennon. OPC-bitin voi konfiguroida aktiiviseksi Standard Event -rekisteriin bittimaskilla *ESE 1 ja OPC-bitin tulos voidaan resetoida rekisterissä komennoilla *CLS tai *ESR?

Käytännössä OPC-bitin käyttö tarkoittaa sitä, että jokaisen virtalähteen komennon perään pitää ketjuttaa komento ;*OPC. Tällä tavoin jokaisen komennon valmistuminen voidaan varmistaa virtalähteeltä pollaamalla Prologix-ohjaimen ++srq -toimintoa. Kun SRQ-signaali havaitaan, virtalähteeltä nollataan SRQ bitit ja siirrytään seuraavan komennon suorittamiseen. Vaikka OPC-bittiä voitaisiin käyttää kaikkiin virtalähteen komentoihin, MAV-bitin käyttö on hyödyllistä, sillä sen voi nollata automaattisesti luettaessa virtalähteen tulopuskuri tyhjäksi. Rekisterit tulee myös alustaa oikein, mikä hoituu systeemin käynnistämisen alussa komentosarjalla: *ESE 1;*SRE 48.

3 Lämpötilan vaikutus diodissa

Diodin I-V kuvaaja, tai virtajännitekuvaaja, kertoo diodin läpi menevän virran sen yli olevan jännitteen funktiona. Myötäsuuntaisella diodilla tyypillinen kuvaaja alkaa matalalla virralla, kunnes tietty myötöjännite saavutetaan ja diodi alkaa johtamaan eksponentiaalisesti. Estojännitteellä diodi johtaa hieman vuotovirtaa, kunnes se saavuttaa läpilyöntijännitteen, jolloin diodin läpi kulkeva virta kasvaa merkittävästi.

Eräs keino estimoida diodin liitoksen lämpötilaa voidaan hyödyntää diodin lämpöresistanssia ja siinä kuluva tehoa. Lämpöresistanssit (Yhtälö 1, Yhtälö 2) ovat yksiköissä [$^{\circ}\text{C}/\text{W}$], joiden arvot voidaan saada joko mittaamalla tai useammin suoraan komponentin datalehdessä. Läpivi-entikomponenteille voidaan käyttää yhtälöä 1 lämpötilan laskennassa.

$$T = R_{thJA}P_D + T_A \quad (1)$$

$$T = (R_{thJC} + R_{thJH} + R_{thHA})P_D + T_A \quad (2)$$

3.1 Diodin virtayhtälö

Biasoimalla pn-liitosta ulkoisella (myöntäsuuntaisella) jännitteellä saadaan aikaan potentiaalin muutos ja siten pn-liitos johtavaksi. Shockley-yhtälö kertoo pn-liitoksen yli menevän virran. Yhtälössä oleva V_D tarkoittaa diodin yli olevan jännitteen suuruutta, mikä aiheuttaa diodilla sen yli menevän virran eksponentiaalisen kasvun.

$$I_D = I_0(T) \left[\exp\left(\frac{qV_D}{k_B T}\right) - 1 \right], \quad (3)$$

$$I_0(T) = qA \left(\frac{D_p}{L_p} p_n + \frac{D_n}{L_n} n_p \right), \quad (4)$$

Vaikka diodivirtayhtälö saattaa 3 näyttää siltä, että lämpötilan T kasvaessa virta vähenisi, mutta käytännössä saturaatiovirta $I_0(T)$ dominoi lämpötilan kasvaessa ja aiheuttaa liitospotentiaalin V_D laskun sekä diodivirran I_D kasvun. Esimerkiksi Simulation Program with Integrated Circuit Emphasis (SPICE) piirisimulaattori mallintaa lämpötilan vaikutusta saturaatiovirtaan vastaavalla funktiolla [7]:

$$I_0(T) = I_{0,T_{nom}} \left(\frac{T}{T_{nom}} \right)^{\frac{xTI}{n}} \exp \left[\frac{qE_G}{nk_B} \left(\frac{1}{T_{nom}} - \frac{1}{T} \right) \right], \quad (5)$$

3.2 Sarjaresistanssin vaikutus diodin virtajännitekuvaajaan

Diodin virtajännitekuvaajasta (Kuva 3) nähdään, että jännitteen kasvattaminen myötäsunnaisessa diodissa (noin 0.7 V) aiheuttaa suuren muutoksen diodin yli kulkevaan virtaan. Tästä johtuen diodin virtajännitekuvaajaa mittattaessa voidaan erittäin herkästi tuhota diodi liiallisen tehonkulutuksen vuoksi. Eräs tapa estää diodin ylikuormittuminen on käyttää sarjaresistanssia, jolla rajoitetaan piirissä kulkeva maksimivirta arvoon $I_{max} = (V_{max} - V_D) / R$, jossa R on käytetyn vastuksen resistanssi. Sarjaresistanssin käyttö kuitenkin muokkaa saatua virtajännitekuvaajaa, linearisoimalla sitä. Olennaisesti myös esimerkiksi ohuiden mittajohtojen käyttö kasvattaa sarjaresistanssia yhtälön 6 mukaisesti.

$$R = \frac{\rho L}{A} \quad (6)$$

Diodeissa kuitenkin on myös sisäistä resistanssia (Yhtälö 7), mikä tulisi ottaa huomioon virtayhtälössä 3, mikäli sen vaikutus olisi merkittävä. Virtayhtälön ratkaiseminen sarjaresistanssin kanssa on hieman vaikeampi, mutta siihen voidaan käyttää muun muassa Lambertin W-funktiota.

$$I_D = \frac{V - V_D}{R} \quad (7)$$

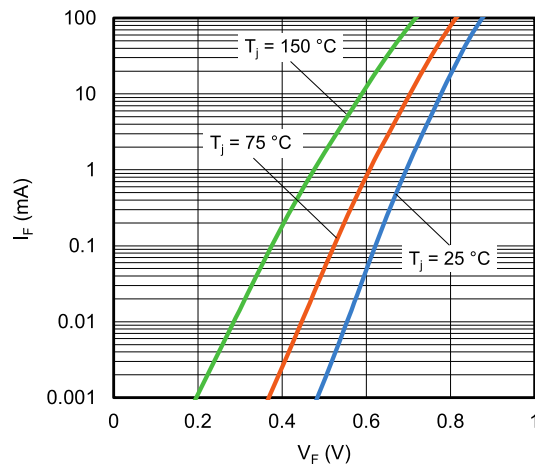


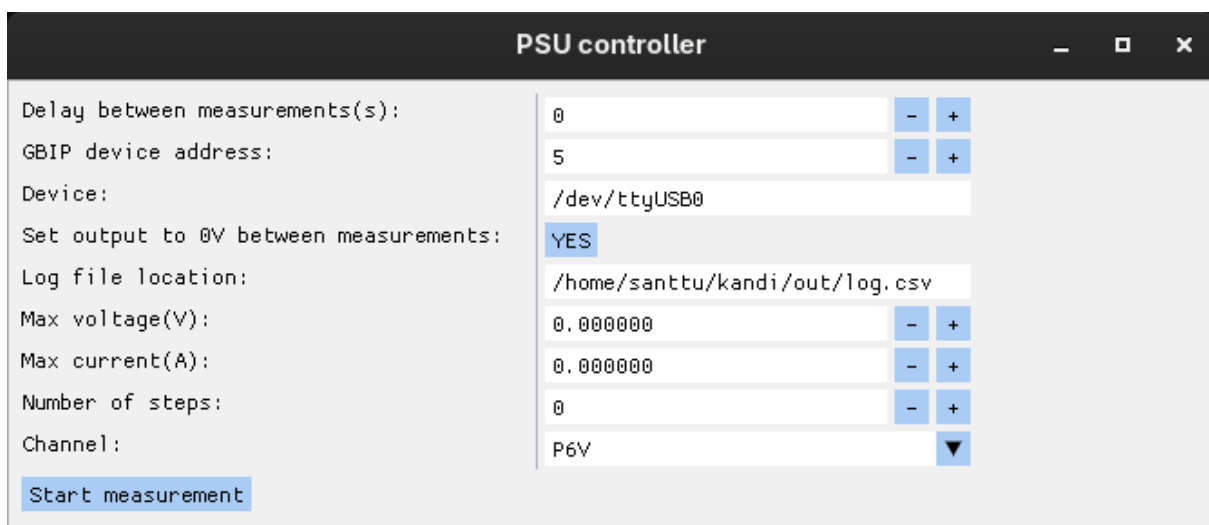
Fig. 2 - Typical Forward Current I_F vs. Forward Voltage V_F

Kuva 3. Vishay Semiconductors, 1N4728A to 1N4761A [8]

4 Ohjelmisto

Virtalähdettä ja Prologix-ohjainta ohjattiin tietokoneelta graafisen käyttöliittymän avulla, joka toteutettiin C++-ohjelmointikielellä. Käyttöliittymää ajettiin Linuxilla, mutta kaikki käytetyt kirjastot toimivat kuitenkin myös Windowsilla, joten ohjelmiston porttaaminen Windowsille olisi mahdollista ilman kirjastojen vaihtamista.

Ohjelmiston suunnittelu jaettiin kahtia käyttöliittymään ja mittauksen tekeviin luokkiin. Käyttöliittymä toteutettiin *Dear ImGui* -kirjastolla. Koska Dear ImGui on *immediate mode GUI*, itse mittausprosessi käynnistettiin standardikirjaston `std::async`-metodilla, jotta käyttöliittymä ei lukittuisi mittauksen ajaksi. Käyttöliittymästä käyttäjä pystyi asettamaan virtalähteen sekä Prologixin parametrejä. Varsinaisen mittausprosessin suorittavalle puoliskolle ohjelmistosta tehtiin kaksi eri luokkaa, joista ensimmäinen erikoistui Prologix-ohjaimen syntaksiin ja tarjosi muulle ohjelmistolle yksinkertaiset read/write-metodit, kun taas toisella luokalla toteutettiin korkeamman tason mittaussekvenssien järjestelyä.



Kuva 4. Käyttöliittymä

4.1 Matalan tason metodit

Työssä tehtiin oma luokka `class GPIBDevice` niille metodeille, jotka vaativat joko Prologix-ohjaimen omia komentoja (kappale 2.2.1) tai jotka vaativat sarjaväylälle tyypillisiä argumentteja. Luokan tarkoitus oli tarjota muille sen periville olioille joukko erilaisia toiminnallisuuksia, joiden pohjalta pystyttiin rakentamaan komentosarjoja virtalähteelle. Toisin sanoen, `GPIBDevice`-luokan perivä luokka pystyi käsittelemään virtalähdettä pelkästään SCPI-komentojen syntaksilla, ilman huolta eri laitteiden synkronoinnista GPIB-väylällä. Sarjakommunikaation ohjaaminen toteutettiin kompositiolla, antamalla `GPIBDevice`-luokalle luokan `class serialib`-luokan attribuutti `m_serial`. Prologix-ohjaimen käyttämä osoite GPIB-väylällä asetettiin luokan attribuutiksi.

4.1.1 readString

Metodia käytettiin datan lukemiseen GPIB-väylältä. Metodilla voitiin lukea enintään neljä kilotavua pitkä merkkijono, jonka pituus johtui Prologix-ohjaimen FTDI-piirin muistin maksimikoosta [5]. Muistin koko oli riittävä, sillä virtalähteen mittaustulosten pituudet olivat maksimissaan kymmeniä ASCII-merkkejä.

Työssä Prologix-ohjaimelle asetettu `++read_tmo_ms` -viive asetettiin sen maksimiarvoon (3000 ms), mutta viivettä ei käytetty sarjaväylällä, vaikka serialib-kirjasto mahdollistaisi samanlaisen viiveen käytön lukuoperaatioissa, kuten esitelty kappaleessa 2.2.1. Syynä tähän oli se, että lukuoperaatioiden toteutus aika piteni yli sekunnilla. Tämä johtui siitä, että sarjakommunikaatioon käytetty Serialib-kirjasto loi oman ajastimen jokaiselle luettavalle ASCII-merkille, hidastaen ohjelmistoa huomattavasti.

- **Palauttaa:** onnistuneesti luettaessa palauttaa enintään 4096 ASCII-merkkiä pitkän merkkijonon tai lukuyrityksen epäonnistuessa tyhjän merkkijonon.

```

1  std::string GPIBDevice::readString()
2  {
3      m_serial.writeString("++read eoi\n");
4
5      const auto bufSize = 4096u;
6      std::string buf(bufSize, '\0');
7
8      const auto readLen = m_serial.readString(buf.data(), '\n', bufSize - 1u);
9      if (readLen <= 0)
10     {
11         return std::string();
12     }
13
14     buf.resize(readLen - 1u);
15     return buf;
16 }
```

4.1.2 isSrqBitSet

Metodia käytettiin SRQ-bitin (Service Request) arvon tarkistamiseksi GPIB-väylällä. Metodi käski Prologix-ohjainta palauttaa SRQ-bitin arvon komennolla `++srq`, jonka jälkeen se luki bitin arvon sarjaväylältä. Bitin arvo on aina joko binääriarvo 0 tai 1.

- **Palauttaa:** totuusarvo tosi/epätosi riippuen SRQ-bitin arvosta väylällä.

```

1  bool GPIBDevice::isSrqBitSet()
2  {
3      constexpr auto srqReadSize = 4u; // bit + \n\r\0
4      std::string buf(srqReadSize, '\0');
5      m_serial.writeString("++srq\n");
6      m_serial.readString(buf.data(), '\n', srqReadSize - 1u);
7      return buf[0] != '\0';
8  }
```

4.1.3 *writeCmd*

Metodilla lähetettiin GPIB-väylälle komennot, jotka eivät palauttaneet tietokoneelle luettavaksi osoitettua dataa Hewlett-Packard E3631A virtalähteeltä. Metodi ketjutti lähetettävän komennon perään *OPC-komennon, jonka jälkeen GPIB-väylän SRQ-bittiä pollattiin, kunnes virtalähde asetti sen päälle. Tämän jälkeen virtalähteelle lähetettiin SRQ-bitin nollaava komento *CLS, jonka jälkeen SRQ-bittiä taas pollattiin, kunnes se asettui pois päältä. Jälkimmäinen pollaus toteutettiin synkronoinnin vuoksi, jottei seuraavaksi suoritettava komento olisi havainnut sitä edeltävän komennon SRQ-bitin arvoa, mikäli *CLS ei olisi ehtinyt nollaamaan sitä ajoissa.

- **Parametrit:** `const std::string& cmd`, lähetettävä komento.

```

1 void GPIBDevice::writeCmd(const std::string& cmd)
2 {
3     constexpr auto srqPollDelayMs = 10u;
4
5     m_serial.writeString((cmd + ";*WAI;*OPC" + '\n').c_str());
6     while (!isSrqBitSet())
7     {
8         std::this_thread::sleep_for(std::chrono::milliseconds(srqPollDelayMs));
9     }
10
11    m_serial.writeString("*CLS\n");
12    while (isSrqBitSet())
13    {
14        std::this_thread::sleep_for(std::chrono::milliseconds(srqPollDelayMs));
15    }
16 }

```

4.1.4 *queryCmd*

Metodia käytettiin komennon kirjoittamiseen ja virtalähteen vastauksen lukemiseen kyseisestä komennosta. Virtalähde konfiguroitiin asettamaan SRQ-bitti päälle MAV-bitin arvosta (kappale 2.4). SRQ:n arvoa pollattiin GPIB-väylällä, kunnes se asettui päälle. Bittiä ei tarvinnut nollata erikseen, vaan virtalähteen lukurekisterin tyhjäksi lukeminen asetti SRQ-bitin pois päältä.

- **Parametrit:** `const std::string& cmd`, virtalähteelle lähetettävä komento.
- **Palauttaa:** onnistuneesti luettaessa palauttaa enintään 4096 ASCII-merkkiä pitkän merkkijonon, tai lukuryityksen epäonnistuessa tyhjän merkkijonon.

```

1 std::string GPIBDevice::queryCmd(const std::string& cmd)
2 {
3     constexpr auto srqPollDelayMs = 10u;
4     m_serial.writeString((cmd + '\n').c_str());
5     while (!isSrqBitSet())
6     {
7         std::this_thread::sleep_for(std::chrono::milliseconds(srqPollDelayMs));
8     }
9     return readString();
10 }

```

4.1.5 openDevice

GPIODevice-luokan metodi, jota kutsuttiin ohjelman alussa Prologix-ohjaimen alustamista varten käyttöliittymästä. Viimeisen writeCmd()-kutsun vuoksi virtalähteen tuli olla jo yhdistetty ohjaimen ennen openDevice:n kutsumista. Prologix-ohjaimelle asetettujen asetusten arvoista lisää kappaleessa 2.2.1, virtalähteen bittimaskeista kappaleessa 2.4 sekä baudinopeuden valinnasta kappaleessa 2.2.

- **Parametrit:** const std::string& device, sarjalaitteen laitetiedosto. Esimerkiksi kappaleessa 2.2 esitetty /dev/ttyUSB0.

```

1 void GPIODevice::openDevice(const std::string& device)
2 {
3     m_serial.openDevice(device.c_str(), 115200u); // Dummy baudrate
4     m_serial.writeString(++savecfg 0\n");
5     m_serial.writeString(++mode 1\n");
6     m_serial.writeString(++auto 0\n");
7     m_serial.writeString(++eoi 1\n");
8     m_serial.writeString(++eos 3\n");
9     m_serial.writeString(++read_tmo_ms 3000\n"); // Max delay
10    m_serial.writeString(std::format(++addr {} \n", m_addr).c_str());
11
12    writeCmd("*ESE 1;*SRE 48"); // OPC=2^0=1 and (MAV=2^4=16 + ESB=2^5=32) = 48
13 }

```

4.2 Mittausdatan ottavat metodit

Virtalähteen ohjausta varten tehtiin oma luokka class PowerSupply. Luokka suunniteltiin toteutettavaksi siten, että sen sisältäviä toiminnallisuuksia voitaisiin mahdollisesti laajentaa tarpeen vaatiessa myös muihin mittaus toimintojen automatisaatioon. Luokka peri matalamman tason luokan GPIODevice (kappale 4.1), jolloin virtalähteen omaa luokkaa voitaisiin ohjata pelkillä SCPI-komennoilla ohjelmiston käyttöliittymästä, oli se sitten graafinen käyttöliittymä tai komentoliittymä.

Luokka class PowerSupply sisälsi kolme eri virtalähteelle ominaista attribuuttia: maksimijännite, maksimivirta ja käytettävä kanava virtalähteessä. Loppuja eri mittauksiin tarvittavia parametrejä voidaan käyttää tämän luokan metodien argumentteina. Eri mittaus tulosten saamiseksi PowerSupply -luokalle toteutettiin kaksi eri metodia mittausten saamiseksi, joista käyttäjä voisi valita toisen ohjelmiston käyttöliittymästä kullekin mittaussarjalle.

4.2.1 getCoolMeasurements

Metodi asetti määritetyn jännitteen virtalähteelle ja kytki sen lähdön päälle. Samassa ketjutuksessa komennossa virtalähteeltä pyydettiin myös virtamittaus, jotta välttyttäisiin ylimääräiseltä kirjoituskerralta ja tulos saataisiin virtalähteeltä ulos mahdollisimman nopeasti. Metodilla otettiin myös erillinen jännitemittaus, jonka jälkeen virtalähteen lähtö asetettiin pois päältä. Lopuksi säikeen annettiin odottaa mittausten välinen viive standardikirjaston std::this_thread::sleep_for-kutsulla.

- **Parametrit:**

- `const uint32_t voltageStep`, asetettavan jänniteaskeleen arvo.
- `const uint32_t delaySeconds`, viive eri mittausten välillä sekunteina.

- **Palauttaa:** mitattu virta ja jännite desimaalilukuna `std::tuple`-formaattissa: `<jännite, virta>`.

```

1  std::tuple<std::string, std::string> PowerSupply::getCoolMeasurements(
2      const double voltageStep,
3      const uint32_t sleepSeconds)
4  {
5      const auto current = queryCmd(
6          std::format("VOLT {:.6f};OUTP ON;*WAI;MEAS:Curr?", voltageStep));
7      const auto voltage = queryCmd("MEAS:VOLT?");
8
9      writeCmd("OUTP OFF");
10     std::this_thread::sleep_for(std::chrono::seconds(sleepSeconds));
11
12     return std::make_tuple(voltage, current);
13 }

```

4.2.2 *getHotMeasurements*

Metodi asetti virtalähteelle tietyn jännitteen ja kytki virtalähteen lähdön päälle. Tämän jälkeen jännitettä pidettiin `sleepSeconds`-välinen aika päällä, jonka jälkeen virtalähde käskettiin palauttaa tulokset käytetystä virrasta ja jännitteestä. Huomioitavaa metodissa ja sen käytössä oli lisäksi se, että virtalähteen lähtöä ei koskaan asetettu mittaussekvenssin aikana pois päältä. Tästä johtuen liian pitkät mittausten väliset viiveet saattoivat herkästi tuhota mitattavana olevan diodin. Itse metodi oli samankaltainen kuin `getCoolMeasurements`, mutta viiveen ajastus toteutettiin siten, että diodi ehti lämmetä uudella jänniteaskeleella mittausten välisen ajan.

```

1  std::tuple<std::string, std::string> PowerSupply::getHotMeasurements(
2      const double voltageStep,
3      const uint32_t sleepSeconds)
4  {
5      const auto gpibCmd = std::format("VOLT {:.6f};OUTP ON", voltageStep);
6      writeCmd(gpibCmd);
7      std::this_thread::sleep_for(std::chrono::seconds(sleepSeconds));
8
9      const auto current = queryCmd("MEAS:Curr?");
10     const auto voltage = queryCmd("MEAS:VOLT?");
11     return std::make_tuple(voltage, current);
12 }

```

4.2.3 *psuWriteAndLog*

Metodilla kutsuttiin joko `getCoolMeasurements` tai `getHotMeasurements` -metodia sen mukaan, haluttiinko virtamittaus suorittaa välittömästi jännitteen asettamisen jälkeen vai annettiin diodin lämmetä `delaySeconds`-ajan verran.

- **Parametrit:**

- `const std::string& logCsvPath`, CSV-tiedoston sijainti tietokoneella.
- `const uint32_t delaySeconds`, viive eri mittausten välillä sekunteina.
- `const bool isCooling`, lippu, jonka avulla virtalähteen lähtö asetettiin pois päältä eri mittausten välissä.
- `const uint32_t steps`, eri jänniteaskeleiden määrä mittauksessa.

```

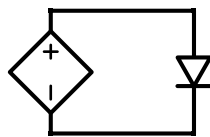
1 void PowerSupply::psuWriteAndLog(const std::string& logCsvPath,
2                                 const uint32_t      delaySeconds,
3                                 const bool          isCooling,
4                                 const uint32_t      steps)
5 {
6     std::ofstream csv(logCsvPath);
7     csv << "Voltage(V),Current(A)\n";
8
9     writeCmd(std::format("APPL {},0, {:.3f}", m_channel, m_maxCurrent));
10
11     const auto dV = m_maxVoltage / steps;
12     for (auto i = 1u; i <= steps; ++i)
13     {
14         const auto voltage = dV * i;
15
16         const auto [measVoltage, measCurrent] =
17             isCooling ? getCoolMeasurements(voltage, delaySeconds)
18                       : getHotMeasurements(voltage, delaySeconds);
19
20         csv << measVoltage << ',' << measCurrent << '\n';
21     }
22     writeCmd("OUTP OFF;VOLT 0");
23 }

```

5 Tulokset

Hewlett Packard E3631A mahdollisti maksimivirran määrittelyn GPIB:n kautta, jolloin diodien kanssa ei jouduttu käyttämään etuvastuksia. Testit toteutettiin mittaamalla myötäsuuntaisia diodeja (Kuva 5). Kun lämmön vaikutus haluttiin minimoida, asetettiin diodi tiettyyn jännitteeseen, jonka jälkeen merkittiin ylös käyttöjännite ja -virta. Sen jälkeen virtalähteen jännitelähtö asetettiin pois päältä ja diodin annettiin jäähtyä jonkin aikaa. Samaa mittausprosessia toistettiin, kunnes maksimijännite oli saavutettu. Kun taas haluttiin korostaa lämpötilan vaikutusta, diodille asetettiin jännite ja odotettiin tietty aika. Tämän jälkeen otettiin ylös käyttöjännite ja -virta, sekä siirryttiin korkeammalle jänniteaskeleelle, toistaen samaa toimenpidettä maksimijännitteeseen asti.

Mittausten jännite- ja virtamittaukset jouduttiin ottamaan erillisillä GPIB-komennoilla (kappale 2.3), minkä vuoksi jälkimmäinen mittaus laahaa hieman ensimmäisen mittauksen jäljessä. Työssä käytetyllä virtalähteellä jännite-/virtamittauksen suorittamisessa kestää maksimissaan noin sata millisekuntia [3], jolloin jälkimmäinen mittaus voi olla ainakin tämän verran ensimmäistä jäljessä. Kuitenkin mikäli virtalähteen maksimivirtaa ei saavuteta, voidaan lähtöjännite olettaa olevan aikainvariantti, jonka vuoksi vähemmän aikakriittinen mittaus oli käyttöjännitteen lukeminen. Siitä syystä virtamittaus suoritettiin työssä ensimmäisenä, kuten kappaleen 4.2.1 metodeista näkee. Jos mittauksissa saavutettiin virtalähteelle maksimi ominainen/ohjelmoitu virta IV-mittausten välissä, niin silloin maksimijännitteen tulokseen ei voinut luottaa. Tämä johtuu siitä, että diodin yli kulkeva virta saattoi kasvaa virta- ja jännitemittauksen välissä niin paljon, että virtalähde lähti laskemaan lähtöjännitettä pitääkseen maksimivirran aisoissa.

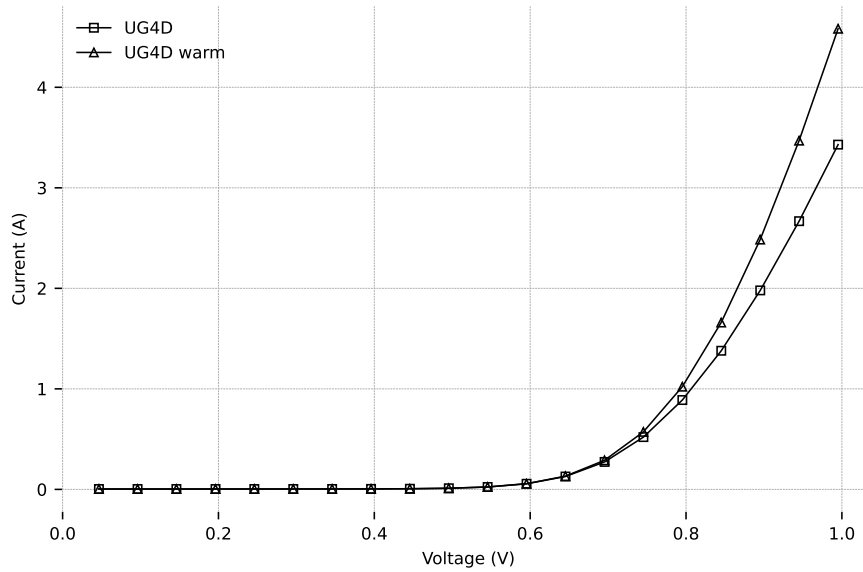


Kuva 5. Mittajärjestely

5.1 UG4D ja 1N4007

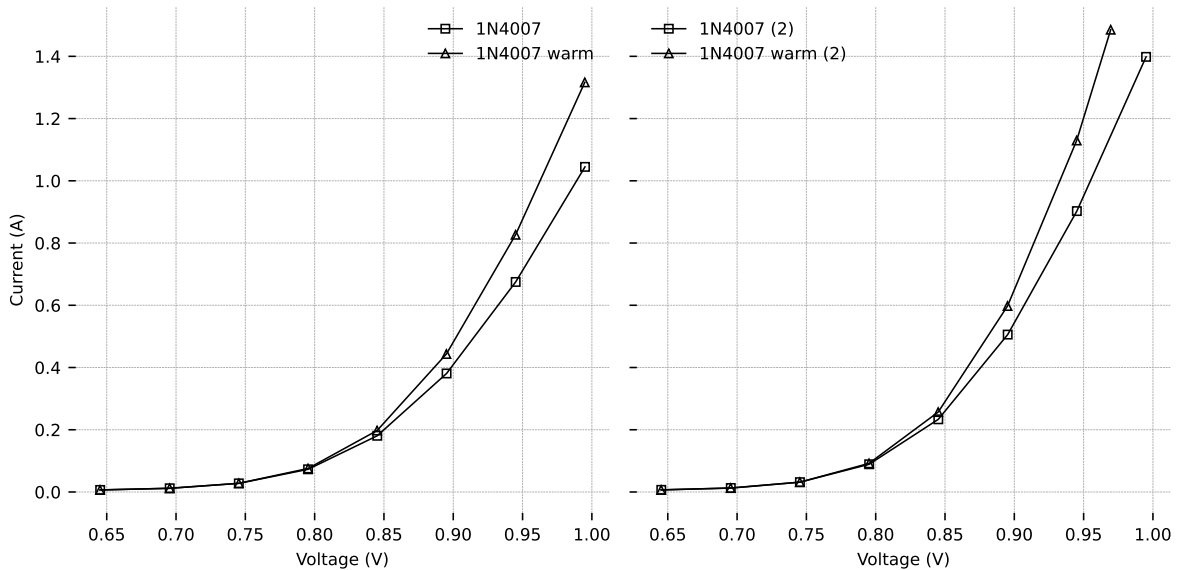
Ensimmäinen testi suoritettiin käyttämällä UG4D diodilla. Maksimivirraksi asetettiin 5 A, maksimijännitteeksi 1 V ja mittapisteiden lukumääräksi 20 kappaletta tasavälein. Diodin lämpenemisen vaikutus mitattiin antamalla diodin jäähtyä kahdeksan sekunnin ajan jokaisessa mittauspisteessä. Vastaavasti diodin lämpenemisen vaikutus testattiin samoilla parametreillä, mutta diodin annettiin lämmetä kolme sekuntia jokaisessa mittauspisteessä.

Tarpeeksi pienellä jännitteellä kuvaajasta 6 huomataan, että molemmat tapaukset ovat lähes samanlaisia johtuen siitä, että diodi ei kerkeä ja/tai pysty generoimaan tarpeeksi lämpöä kolmen sekunnin asetus- ja mittausvälillä. Kun noin 0.75 V kynnyisjännite saavutettiin, diodi alkoi lämmetä tarpeeksi jännitteen asetuksen ja mittausten ottamisen välissä. Noin yhden voltin tasolla kappaleessa 3 kuvattu lämpötilan vaikutus pn-liitoksessa on jo kohtuullisen suuri.



Kuva 6. UG4D virtajännitekuvaaja

Käytettäessä fyysisesti pienempää 1N4007 lämpötilan vaikutus oli myös havaittavissa virran kasvulla, käyttöjännitteen pysyessä samana (Kuva 7). Maksimivirraksi asetettiin 1.5 A ja maksimijännitteeksi 1 V, muuten mittauksen parametrit olivat samat kuin UG4D:n tapauksessa. Kuvaajista huomataan, että vaikka diodeilla oli eroja toisiinsa, mutta lämpötilan vaikutus niissä oli samankaltainen.



Kuva 7. 1N4007 virtajännitekuvaaja

6 Pohdinta

Tulokset näyttivät, että HP E3631A oli tarpeeksi nopea suorittamaan fyysisesti suurempikokoisten diskreettipuolijohteiden virtajännitekuvaajien mittaukset ja komponentin itselämpeämisen voitiin havaita kuvaajista. Kriittisin aika mitattaessa puolijohdetta on aika jännitteen asettamisesta siihen asti, kunnes virtamittaus saadaan valmiiksi. Rajoitteita tähän tuo olennaisesti virtalähteen kyky nostaa käyttöjännite haluttuun arvoon, joka työssä käytetyllä virtalähteellä oli maksimissaan 11 millisekuntia nousta prosentin tarkkuudella maksimiarvoonsa +6V lähdöllä käytettäessä täysin resistiivistä kuormaa [3]. Tämä ei välttämättä ole suurin rajoite, sillä kuten työssä käytetyillä diskreettikomponenteilla, maksimijännite mittauksissa oli noin yhden voltin luokkaa, jolloin virtalähteen manuaalin ilmoittama 11 ms maksimi voi olla matalampi. Eniten mittausnopeuteen todennäköisesti vaikuttaa virtalähteen nopeus prosessoida itse GPIB-komennot, joista tärkein on mittauskomennon prosessointinopeus. Työssä käytetyllä virtalähteellä tämä viive oli sata millisekuntia [3].

Mittausarjojen ottamiseen vaikuttaa kokonaisaika, jolloin virtalähteen lähtöjännite on vielä päällä (eli hetki jännitteen asettamisesta siihen, kunnes lähtö asetetaan pois päältä). Se vaikuttaa lähinnä aikaan, joka pitää odottaa eri mittahetkien välissä diodin jäähtymistä odotellessa. Kuitenkin se on hyvä pitää mielessä, mikäli mitattava komponentti saattaisi tuhoutua siinä ajassa liiallisen virran ja/tai lämmön takia. Käytetyn virtalähteen muunnin lupasi 6V lähdölle 13 ms laskea jännite kuudesta voltista noltaan (ilman kuormaa jopa 200 ms). Tämän lisäksi huomioon tulee ottaa VOLT 0.0 -komennon 50 ms prosessointiviive [3].

Mikäli mitattavan komponentin annetaan jäähtyä takaisin sen ympäristön lämpötilaan, kriittisin aika mittatuloksissa on aika jännitteen asettamisesta siihen asti, kunnes virtamittaus saadaan valmiiksi. Toisin sanoen, vaikka pulssisuhdetta muokataan pidentämällä mittausten välistä jäähtymiseen kuluva aikaa, rajoittavaksi tekijäksi jää virtalähteen jännitteen asetuksen ja virtamittauksen välinen aika. Tässä työssä käytetylle virtalähteelle todennäköisesti suurin viivettä aiheuttava tekijä mittausten välissä oli MEASure? -komennon prosessointiaika, joka oli maksimissaan noin sata millisekuntia [3].

6.1 Jäähtymisen viive

Mikäli mittausten välissä jäähtyvälle diodille jouduttaisiin tekemään useita mittauksia, voisi olla järkevää approksimoida diodin jäähtymiseen kuluva aikaa. Eri mittauspisteet voitaisiin asettaa siten, että ensimmäiset jäähtymiseen kuluvat ajat olisivat huomattavasti lyhyempiä. Jäähtymiseen kuluvat ajat voitaisiin sitten kertoa tietyllä vakiolla, kun diodin myötäjännite saavutettaisiin. Tämän voisi hoitaa joko ennalta määritetyllä jänniteaskeleella tai asettamalla raja luetulle virralle, jonka ylittyessä jäähtymisaikaa nostettaisiin.

$$T(t) = T_{env} + [T(0) - T_{env}] e^{-kt} \quad (8)$$

Eräs tapa approksimoida jäähtymisaikaa voisi hoitua muun muassa Newtonin jäähtymislain mukaan. Yhtälöä 8 voitaisiin hyödyntää ratkaisemalla parametri k komponentin jäähtyessä tietyllä aikavälillä, jonka jälkeen yhtälöä hyödyntämällä päätettäisiin järkevät aikavälit mitattavan komponentin jäähtymiselle eri mittaushetkien välissä. Kuitenkin kyseinen vakio on yleisesti ottaen riippuvainen komponentin massasta, mistä johtuen pienten piilevillä olevien puolijohteiden lämmönvaihtelujen arviointi voi osoittautua hankalaksi.

6.2 Ohjelmisto

GPIB-laitteen synkronointi tietokoneen kanssa aiheuttaa pieniä viiveitä mittauksen ottamisessa (esimerkiksi *OPC -bitin prosessointi). Jos ohjelmistolta haluttaisiin absoluuttista nopeutta ilman minkäänlaista tietokoneen ja GPIB-laitteen synkronointia, voitaisiin SRQ-bittien pollaukset jättää kokonaan pois. Tällöin olisi kuitenkin hyvä, että komentoja ei suoritettaisi liian tiheään tahtiin, jotta virtalähteen tulorekisteri ei täyttyisi. Toinen mahdollisuus olisi myös muuttaa kappaleen 4.1 pollauksen viivettä *srqPollDelayMs* käytettävän laitteiston sallimaan minimiin.

C++:n käyttö työssä ei hidasta ohjelmiston kulkua juuri ollenkaan, sillä objektien luonti aikakriittisimmällä hetkellä koostui lähinnä neljän tavun kokoisen `std::string`-objektin allokoinnista GPIB-väylän SRQ-bitin tarkistamista varten muutamien millisekuntien välein. Moderneilla tietokoneilla tämän ei tulisi olla pullonkaula yllämainittujen mittausten suorittamisessa, vaan suurin viive löytyy todennäköisimmin komentojen prosessointinopeudesta virtalähteen puolella. Kuitenkin, kuten kappaleessa 4.1.1 käsitellyllä aikakatkaisun toteutuksella, ohjelmisto olisi voitu saada tukkoon luomalla liikaa raskaita objekteja.

Ohjelmiston kirjastot toimivat Linux- ja Windows-alustoilla, joten ohjelmiston porttaaminen eri käyttöjärjestelmälle tulisi onnistua kohtuullisen helposti. Porttausta voi helpottaa yhä enemmän muun muassa käyttämällä ohjelmistoa suoraan komentokäyttöliittymän kautta, jolloin kaikki DearImgui-kirjastot ja GUI:n toteutus voidaan unohtaa täysin. Käytännössä kaikki tarpeellinen ohjelmistossa on GPIBDevice-luokassa, mikä mahdollistaa ohjelmiston käytön myös eri GPIB-laitteilla, kunhan käytössä on sama Prologix-adapteri.

Mikäli vastaavalla ohjelmistolla suoritettaisiin pitkäkestoisia mittasarjoja, ohjelmistoon olisi suotavaa toteuttaa tarkempia vianhallintamekanismeja. Esimerkiksi otetun mittaustuloksen syntaksi voitaisiin tarkistaa ja mikäli tulos ei ole järkevä, kyseinen mittauspiste mitattaisiin uudelleen. Myös epästabiilin USB-liitoksen tapauksessa yhteys Prologix-ohjaimen saattaa katketa, mikä olisi hyvä tunnistaa ohjelmistossa ja mahdollisuuksien mukaan yhdistää ohjain uudelleen ilman suurempia häiriöitä jo otettuihin mittauksiin. Tämän voisi toteuttaa mm. kysymällä ohjaimelta jokin yksinkertainen komento ennen tietyn mittapisteen ottamista ja jos ohjain ei vastaa, ohjelmisto yhdistäisi ohjaimen uudelleen esimerkiksi C++ -standardikirjaston `<filesystem>` avulla.

7 Yhteenveto

Kandidaatintyössä ohjelmoitiin Hewlett Packard E3631A virtalähdettä hyödyntäen IEEE-488-standardia (GPIB) diodin itselämpenemisen havaitsemiseksi. Työn pääaiheena oli arvioida virtalähteen ohjelmointinopeuden soveltuvuutta, jotta fyysisesti suurempikokoisen diodin itselämpenemisen vaikutus näkyisi sen virtajännitekuvaajasta. Virtalähdettä ohjattiin tietokoneen kautta, jota varten käytettiin Prologix USB-GPIB -adapteria. Laitteiston yhteensopivuuden manuaalisen tarkistamisen jälkeen työssä tehtiin C++-ohjelmisto, jolla virtalähdettä voitiin ohjata automaattisesti graafisen käyttöliittymän kautta synkronoidusti.

Virtalähteen komentosarjojen suorittamisen tila tarkistettiin tietokoneella hyödyntäen IEEE 488.2 *service request* (SRQ) -toimintoa. Tietokoneen ja virtalähteen synkronoimiseksi virtalähteen rekisterit asetettiin kertomaan SRQ-bitin arvo GPIB-väylälle komentojen valmistuessa, mikä voitiin havaita tietokoneelta.

Tietokoneella pyörivälle ohjelmistolle GPIB-väylän synkronointiin ja Prologix-ohjaimen asetusten alustamiseen tehtiin oma luokka, jolla muu ohjelmisto pystyi lähettämään käyttäjän määrittelemiä komentosarjoja virtalähteelle. Itse mittasarjojen ottamiseen toteutettiin oma luokka, jota käyttäjä pystyi ohjaamaan suoraan ohjelmiston käyttöliittymästä mittausten parametrien asettamiseksi ja mittaussekvenssin aloittamiseksi.

Diodin itselämpeämisen vaikutus mitattiin asettamalla diodille tietty jänniteaskel ja odottamalla tietty aika, jonka aikana diodi ehtisi lämmetä. Tämän jälkeen mitattiin diodin yli menevä virta ja jännite. Toimenpidettä toistettiin ohjelmoituun maksimijännitteeseen asti tasavälein. Kun haluttiin minimoida diodin itselämpeämisen vaikutus sen virtajännitekuvaajassa, mittaukset suoritettiin melkein samoin tavoin, paitsi että virta- ja jännitemittaukset otettiin välittömästi jännitteen asettamisen jälkeen. Lisäksi virtalähteen lähtöjännite asetettiin mittausten välissä pois päältä, jotta diodi pystyisi jäähtymään eri jänniteaskeleiden välissä. Saaduista tuloksista voitiin päätellä, että Hewlett Packard E3631A -virtalähde oli tarpeeksi nopea havaitsemaan fyysisesti suurempikokoisten diodien itselämpeneminen.

8 LÄHTEET

- [1] Keysight Technologies. High-power device testing for today's wide bandgap technologies. keysight.com/us/en/products/semiconductors/power-device-analyzer-curve-tracer.html. Viitattu 02.03.2026.
- [2] Institute of Electrical and Electronics. *IEEE Std 488.2-1992*. Viitattu 21.02.2026.
- [3] Keysight Technologies. *E3631A Triple Output DC Power Supply User's Guide*. Viitattu 21.02.2026.
- [4] FTDI Chip. D2XX for Linux. ftdichip.com/Driver/D2XX/Linux/ReadMe.txt. Viitattu 12.1.2026.
- [5] PROLOGIX. *GPIB-USB CONTROLLER USER MANUAL VERSION 6.91*. Viitattu 21.02.2026.
- [6] National Instruments. GPIB Messages. ni.com/en/support/documentation/supplemental/06/gpib-messages.html. Viitattu 21.02.2026.
- [7] Analog Devices. *The SPICE Diode Model*. Viitattu 21.02.2026.
- [8] Vishay Semiconductors. *IN4728A to IN4761A*. Viitattu 21.02.2026.