HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Heiko Paal

Entwicklung eines Testplatzes für Induktivitäten und Übertrager

Mittweida, 2022

Fakultät: Ingenieurswissenschaften

BACHELORARBEIT

Entwicklung eines Testplatzes für Induktivitäten und Übertrager

Autor:
Herr

Heiko Paal

Studiengang:
Elektrotechnik - Automation

Seminargruppe:
EA19wA-B

Erstprüfer:
Prof Dr.-Ing. Lutz Rauchfuß

Zweitprüfer:
Dipl.-Ing.(FH) Dirk Prochaska

Einreichung:
Chemnitz, 02.08.2022

Verteidigung/Bewertung:
Chemnitz, 2022

Faculty of Engineering Sciences

BACHELOR THESIS

Development of a test station for inductors and transformers

author:
Mr. Heiko Paal

course of studies:
Electrical Engineering - Automation

seminar group:
EA19wA-B

first examiner:
Prof. Dr.-Ing. Lutz Rauchfuß

second examiner:
Dipl.-Ing.(FH) Dirk Prochaska

submission:
Chemnitz, 02.08.2022

defence / evaluation:
Chemnitz, 2022

Bibliografische Beschreibung:

Paal, Heiko:

Entwicklung eines Testplatzes für Induktivitäten und Übertrager

Development of a test station for inductors and transformers

Verzeichnisse: 4 Seiten, Inhalt: 66 Seiten, Anhänge: 20 Seiten
Mittweida, Hochschule Mittweida, Fakultät Ingenieurswissenschaften, Bachelorarbeit, 2022

Referat:

Ziel dieser Bachelorarbeit ist es, einen Testplatz für Induktivitäten und Übertrager zu entwickeln. Dabei wird anhand des Messprinzips für Hysteresekurven ein Messaufbau mit allen notwendigen Geräten erarbeitet. Danach wird die Automatisierung von häufigen Bedienhandlungen durch Software beschrieben. Anschließend wird eine Bedienung über eine graphische Oberfläche implementiert. Am praktischen Beispiel wird gezeigt, wo die Grenzen des erarbeiteten Messaufbaus liegen, und es werden Hinweise für eine zukünftige Weiterentwicklung gegeben.

Thema der Abschlussarbeit:

„Entwicklung eines Testplatzes für Induktivitäten und Übertrager“

Aufgabenstellung der Bachelorarbeit:

- Ziel: Darstellung der Hysteresekurve auf einem PC
- Aufbau und Funktionsbeschreibung des Messplatzes
- Automatisierung des Prüfablaufes mittels Python-Programm
- Erstellung einer graphischen Oberfläche zur Bedienung und Anzeige des Messplatzes

Mit dem Vorgehen einverstanden

29.06.2022

Gunter Griessbach

Siemens AG
Digital Industries
Factory Automation
Product Creator Chemnitz 1
DI FA SEA SYS CHE1
Leipziger Str. 400
09247 Chemnitz, Deutschland
Tel.: +49 (371) 4810-2310
Mobil: +49 (0172) 3820775
mailto:gunter.griessbach@siemens.com
www.siemens.com

Inhalt

Abbildungsverzeichnis

1 Einleitung

In diesem Kapitel wird beschrieben, warum ein Testplatz für Übertrager notwendig ist. Daraus ergibt sich die Aufgabenstellung und ein Entwurf, wie das Projekt umgesetzt werden kann.

1.1 Notwendigkeit des Testplatzes

Lieferschwierigkeiten von Herstellern elektronischer Bauelemente können negative Auswirkungen auf den eigenen Fertigungsprozess des Unternehmens haben. Daher ist man bestrebt, gleiche oder vergleichbare Produkte von unterschiedlichen Herstellern einzukaufen. Dadurch wird die Gefahr von Versorgungsengpässen minimiert. Allerdings werden oft nicht alle relevanten technische Daten in den Datenblättern hinterlegt, so das man bei Vergleich von Bauelementen eigene Messungen notwendig sind. Bei Übertragern gibt die Hysteresekurve (siehe Abs. 2.1) Aufschlüsse über die magnetischen Eigenschaften des Kerns. Das Ziel dieser Bachelorarbeit ist es, einen Testplatz für Induktivitäten und Übertrager zu entwickeln, mit dem sich Hysteresekurven auf einem PC darstellen lassen.

1.2 Aufgabenstellung

Die Aufgabenstellung „Entwicklung eines Testplatzes für Induktivitäten und Übertrager“ beinhaltet folgende Schwerpunkte:

- Beschreibung des Aufbaus und der Funktionsweise des Messplatzes
- Automatisierung von häufig auftretenden Bedienabläufen und Datenübertragung zwischen Oszilloskop und PC durch ein geeignetes Python-Programm
- eine graphische Oberfläche auf dem PC wertet das geschriebene Programm optisch auf, nimmt Nutzereingaben entgegen und stellt die Hysteresekurve dar

1.3 Umsetzung

Die Umsetzung gliedert sich in mehrere Phasen. Zunächst wird anhand der Messmethode für Hysteresekurven ein Konzept für den hardwareseitigen Aufbau des Messplatzes erarbeitet. Anschließend wird die Datenübertragung zwischen PC und Oszilloskop eruiert. Das bedeutet, das wiederkehrende Bedienhandlungen und die Auswertung von Messdaten durch Software übernommen wird. Im dritten Entwicklungsabschnitt erfolgt die Erzeugung einer graphischen Oberfläche, die Nutzereingaben entgegennimmt und die Hysteresekurve anzeigt. Abschließend werden der Messplatz erprobt und bei Bedarf weitere Handlungsempfehlungen gegeben. Diese Entwicklungsphasen werden in kleinere Abschnitte unterteilt, die dann in den einzelnen Kapiteln dieser Arbeit näher erläutert werden.

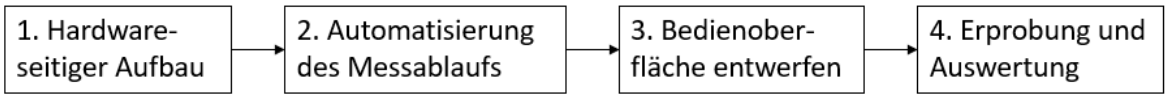


Abbildung 1 Gliederung der Aufgabenstellung in vier Hauptkategorien

2 Hardwareseitiger Aufbau

In diesem Kapitel wird der Aufbau und die Funktionsweise des Testplatzes beschrieben. Nach der Vorstellung des Messprinzips für Übertrager werden sinnvolle Vereinfachungen und deren Umsetzung erläutert.

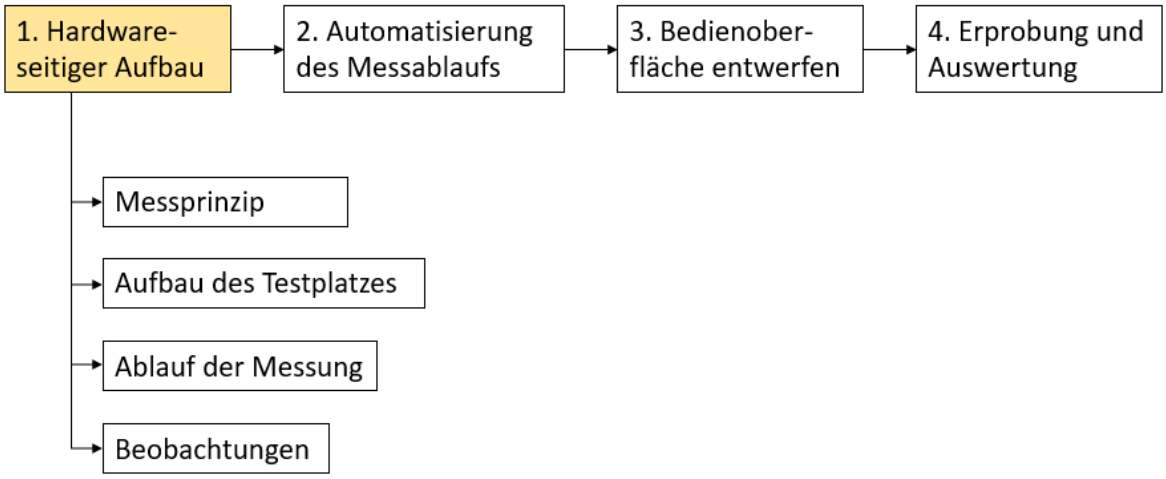


Abbildung 2 Erster Entwicklungsabschnitt

2.1 Theoretische Grundlagen

Eine Spule ist ein elektrisches Bauelement mit folgenden Eigenschaften:

- sie wirkt schnellen Stromänderungen durch eine Selbstinduktionsspannung entgegen
- durch die Änderung des Stromflusses wird ein Magnetfeld aufgebaut / abgebaut

Die Induktivität gibt an, wie hoch die induzierte Spannung und damit der „Widerstand“ gegen die Stromänderung ist. Durch die Wahl eines geeigneten Kernmaterial ist es möglich, die Magnetfeldlinien zu bündeln (so dass diese nicht durch die Luft verlaufen) und die Induktivität zu erhöhen. Bei einem ferromagnetischen Kernwerkstoff sind die Elementarmagnete innerhalb der Weiss’schen Bezirke in die gleiche Richtung ausgerichtet. Die Abgrenzungen der Weiss’schen Bezirke werden Bloch-Wände genannt. Je nach Stärke des äußeren Magnetfeldes richten sich die Elementarmagnete aus und die Bloch-Wände werden verschoben (Abb. 3a). Während dieses Prozesses wird das Magnetfeld verstärkt. Wenn alle Elementarmagnete ausgerichtet sind, ist keine Verstärkung des Magnetfeldes mehr möglich. Dieser Zustand heißt Sättigung (Abb. 3b). Die Induktivität einer Spule sinkt dann auf den Wert, den sie ohne Kernmaterial hätte.

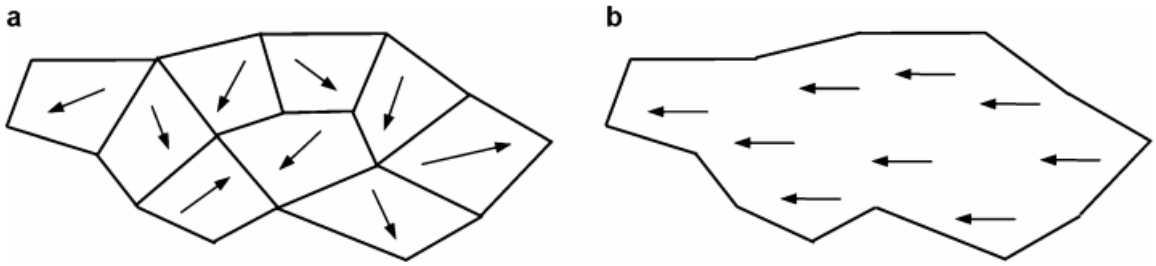


Abbildung 3 Weiss'sche Bezirke, Ausrichtung der Elementarmagnete und Verschiebung der Bloch-Wände (1 S. 269)

Die oben erwähnte Verstärkung der magnetischen Feldstärke H gegenüber Vakuum wird mit der relativen magnetischen Permeabilität μ_r (einheitenlos) angegeben. Diese Permeabilitätszahl ist abhängig vom verwendeten Kernmaterial und von der momentanen Größe der magnetischen Feldstärke. Die Wirkung des Magnetfeldes wird durch die magnetische Flussdichte B angegeben. Es entsteht folgende Beziehung, wobei μ_0 die magnetische Feldkonstante ist:

$$B = \mu_r \cdot \mu_0 \cdot H \tag{1}$$

Der Zusammenhang zwischen B und H ist nicht linear und wird deshalb in einer Magnetisierungskennlinie visuell dargestellt.

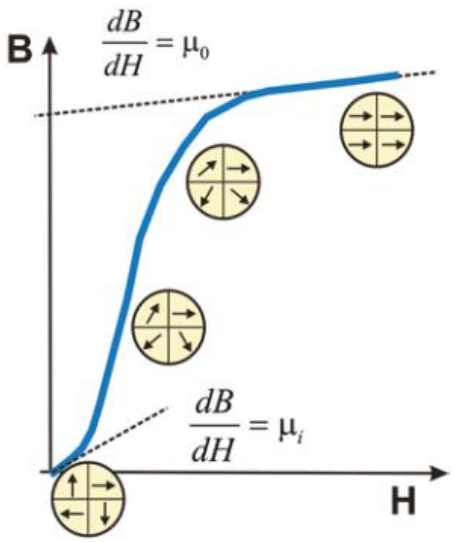


Abbildung 4 Magnetisierungskennlinie mit Ausrichtung der Weiss’schen Bezirke (2 S. 81)

Diese Kennlinie wird nur durchlaufen, wenn das Kernmaterial zum ersten Mal magnetisiert wird (Neukurve). An einer sinusförmigen Wechselspannung betrieben, ergibt sich ein anderer Kurvenverlauf (Abb. 5).

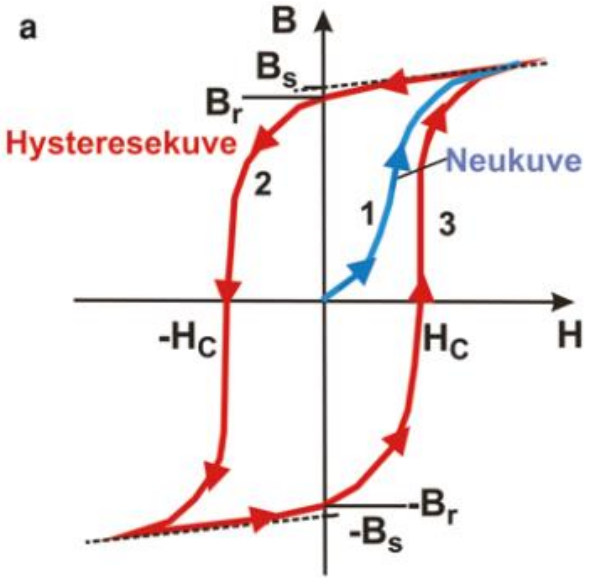


Abbildung 5 Hysteresekurve (2 S. 81)

Die blaue Kurve 1 ist die Neukurve. Vormagnetisiertes Kernmaterial fährt die Kurven 2 und 3 ab. Wenn die magnetische Feldstärke H nicht mehr vorhanden ist, verbleibt ein Restmagnetismus, die Remanenz B_r. Die Weiss'schen Bezirke sind noch etwas ausgerichtet. Das ist der Schnittpunkt der Kennlinie mit der Y-Achse. Ein weiterer signifikanter Punkt ist die Koerzitivfeldstärke H_c, die Höhe der Induktion, die erforderlich ist, um das Kernmaterial vollständig zu entmagnetisieren. Ein dritter Punkt ist die Sättigungsinduktion B_S, eine Erhöhung von H bringt keine wesentliche Erhöhung von B.

Mit dem Testplatz für Induktivitäten und Übertrager soll genau diese Hysteresekurve aufgenommen und dargestellt werden. Außer dem Vergleich von induktiven Bauelementen unterschiedlicher Hersteller kann man feststellen, ob das Kernmaterial hart- oder weichmagnetisch ist und ob ein Luftspalt vorhanden ist. Die eingeschlossene Fläche der Kennlinien 2 und 3 ist ein Maß für die Hystereseverluste.

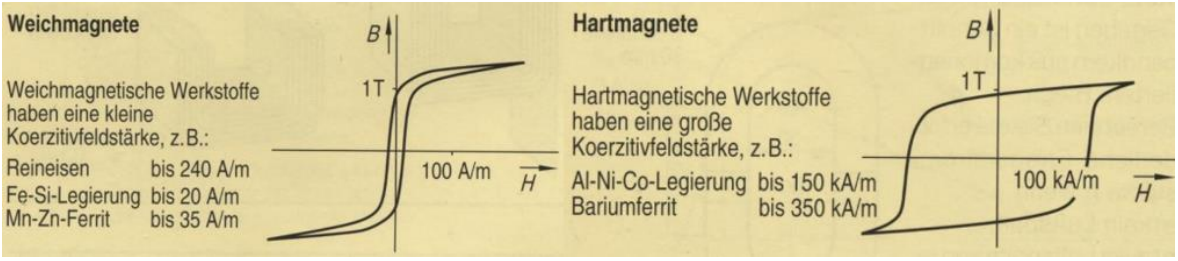


Abbildung 6 Hysteresekurve hart- / weichmagnetisch (3 S. 114)

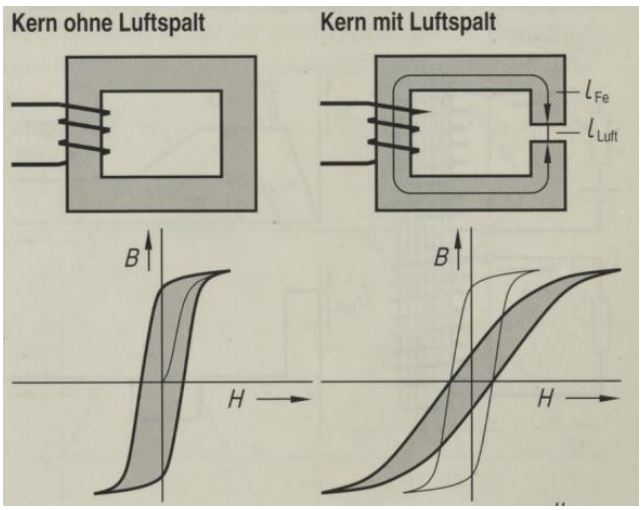


Abbildung 7 Hysteresekurve mit / ohne Luftspalt (3 S. 115)

Die Hysteresekurve verläuft bei Kernen mit Luftspalt flacher und damit über einen größeren Bereich nahezu linear.

2.2 Messprinzip

Das Messprinzip der Hysteresekurve hat sich in den letzten Jahrzehnten nicht wesentlich verändert. Das folgende Bild aus dem Buch „Oszillografen-Messtechnik“ von 1959 zeigt die Messschaltung:

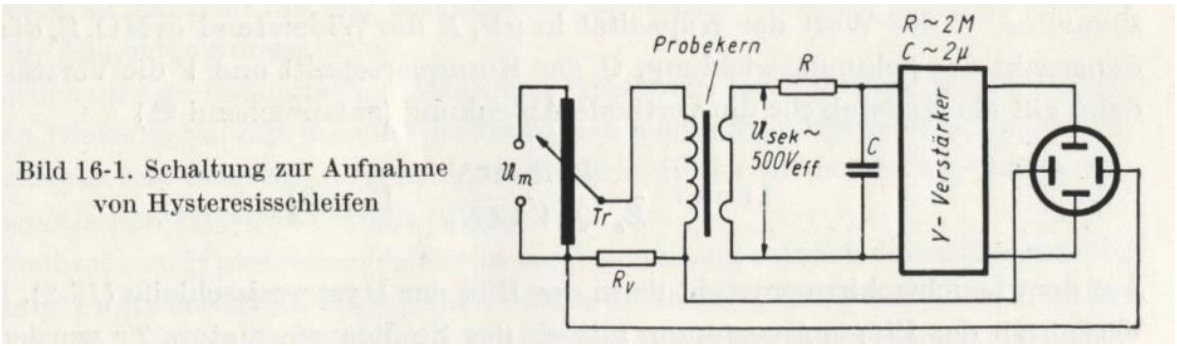


Abbildung 8 Schaltung zur Aufnahme der Hysteresekurve 1959 (4 S. 427)

1993 entfiel in der Darstellung der Y-Verstärker, das Messprinzip war unverändert :

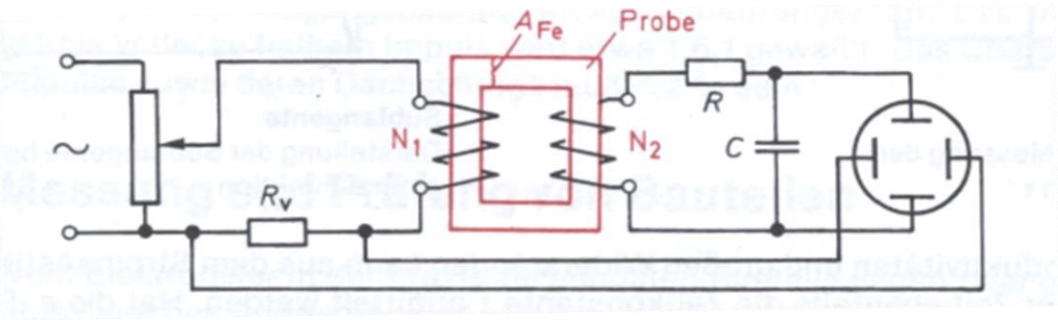


Abbildung 9 Aufnahme der Hysteresekurve 1993 (5 S. 169)

Eine Wechselspannungsquelle verursacht eine zeitliche Änderung des Primärstroms I_{prim}. Der Primärstrom fließt durch die Primärwicklung des Übertragers. Um den Leiter bilden sich geschlossene Feldlinien, die hauptsächlich im Kernmaterial verlaufen. Das Produkt aus Primärstrom und Windungsanzahl ist die magnetische Urspannung Θ (Theta). N ist die Windungszahl.

$$\Theta = I_{prim} \cdot N_{prim} \tag{2}$$

Diese magnetische Urspannung (oder Durchflutung) treibt wiederum den magnetischen Fluss Φ (Phi), dieser ist abhängig vom magnetischen Widerstand R_{magn}.

$$\Phi = \frac{\Theta}{R_{magn}} \tag{3}$$

Die magnetische Feldstärke H ist die magnetische Urspannung bezogen auf die mittlere Länge des magnetischen Kreises:

$$H = I_{prim} \cdot \frac{N_{prim}}{l_{magn}} \tag{4}$$

Eine zeitliche Änderung des magnetischen Flusses Φ bewirkt auf der Sekundärseite eine induzierte Spannung U (Formel 5). Das übliche Minuszeichen wurde weggelassen, da es nur anzeigt, das Strom und Spannung entgegengesetzt sind (Erzeugerzählpfeilsystem):

$$U_{sek} = N_{sek} \cdot \frac{d\Phi}{dt} \tag{5}$$

Durch Einsetzen der Beziehung $\Phi = B \cdot A$ in Formel 5 erhält man (A ist der Querschnitt des Kerns):

$$U_{sek} = N_{sek} \cdot A \cdot \frac{dB}{dt} \tag{6}$$

Durch Integration folgt:

$$B = \frac{1}{N_{sek} \cdot A} \cdot \int U_{sek}\, dt \tag{7}$$

Die Hysteresekurve bildet auf der X-Achse die magnetische Feldstärke H ab. Durch Messung des primärseitigen Stroms kann man die magnetische Feldstärke berechnen (vgl. Formel 4). Der Strom wird durch den Spannungsabfall über R_v ermittelt. Sekundärseitig muss die Spannung integriert werden, um die magnetische Flussdichte B nach Formel 7 berechnen zu können. Für die Integration der Spannung wird das RC-Glied in der Schaltung eingesetzt.

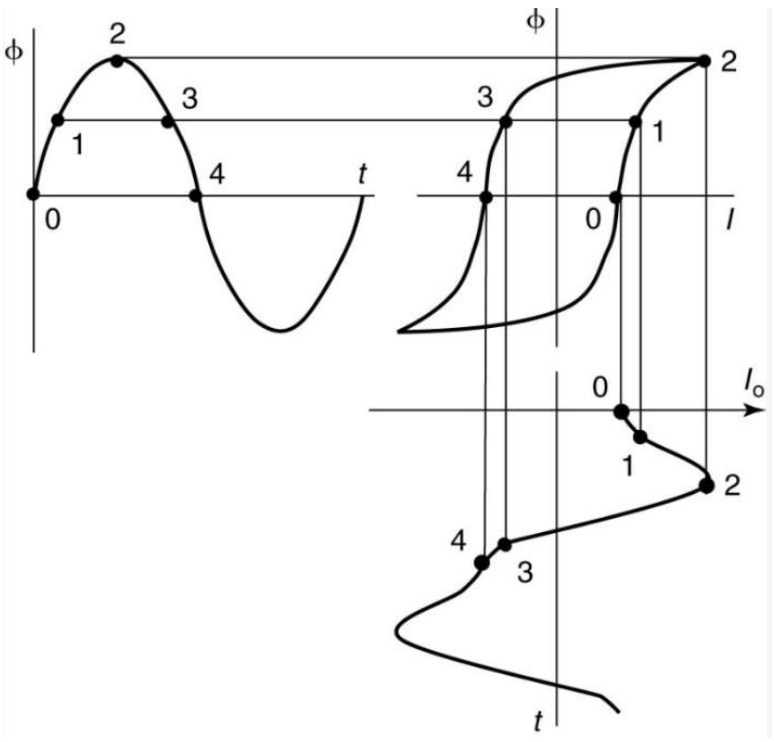


Abbildung 10 Entstehung der Hysteresekurve (6)

Durch den Einsatz eines Digitaloszilloskops kann der Messaufbau vereinfacht werden:

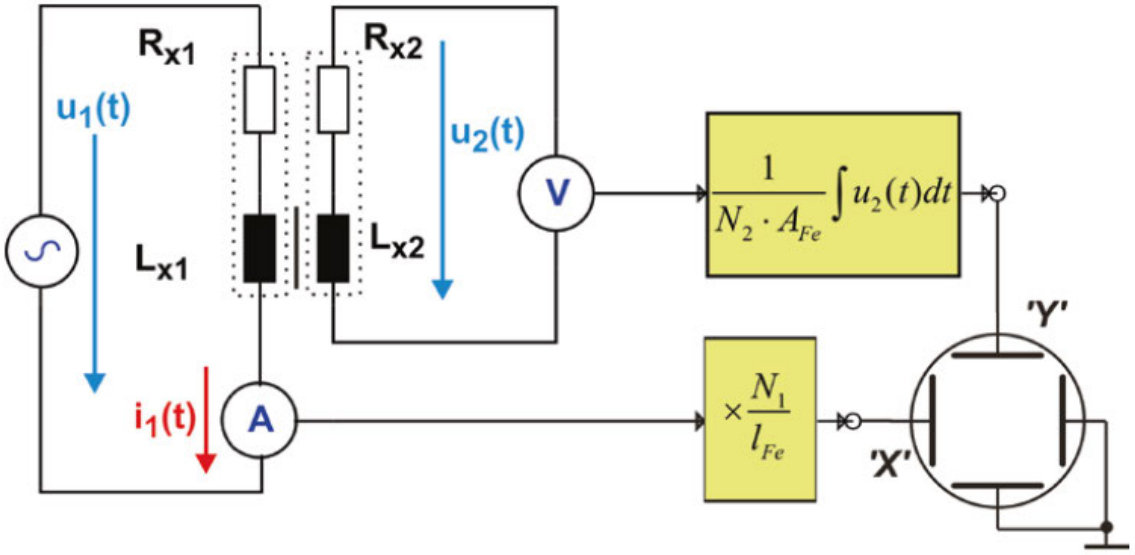


Abbildung 11 Messung der Hysteresekurve mit DSO (2 S. 304)

Der Vorwiderstand entfällt, die Strommessung erfolgt über eine Stromzange. Die Integration der Spannung durch das RC-Glied wird durch eine interne Funktion des Oszilloskops ersetzt. Nach diesem Prinzip soll der Testplatz für Induktivitäten und Übertrager gestaltet werden. Dabei können Übertrager so verwendet werden, wie sie sind, da sowohl Primär- als auch Sekundärwicklung vorhanden sind. Für die Messung von Spulen muss in unserem Fall eine Hilfswicklung (sekundäre Wicklung) aufgebracht werden.

2.3 Aufbau des Testplatzes

Für den Testplatz steht folgendes Testequipment zur Verfügung:

- Oszilloskop MDO34 mit eingebauten Frequenzgenerator
- Signalverstärker Toellner TOE 7607
- Stromtastkopf TCP202 mit TPA-BNC-Adapter
- Spannungstastköpfe TPP0500
- USB-Switch GS108GE
- Laptop HP Probook 650GB, Betriebssystem Windows 10
- Netzwerk- und BNC-Kabel
- USB-Netzwerk-Adapter D-Link DUB-1312

Da die Induktivitäten und Übertrager bei verschiedenen Frequenzen und Strömen geprüft werden sollen, wird die Wechselspannungsquelle aus Abb.11 durch den eingebauten Frequenzgenerator und den nachgeschalteten Signalverstärker gebildet. Der eingebaute Frequenzgenerator des Oszilloskops kann an 50 Ohm Last nur Spannungen bis 2.5 Volt liefern, was einen sehr geringen Strom besonders bei hohen Frequenzen bedeutet. Daher muss ein Signalverstärker angeschlossen werden, der sowohl die Spannung verstärkt als

auch einen größeren Strom liefern kann. Der Verstärker Toellner TOE7607 erfüllt diese Bedingung mit ± 22.5 Volt Ausgangsspannung und ± 300 mA Strom bei rund 10 Ohm Innenwiderstand leider nicht. Zur Messung der im Unternehmen eingesetzten Übertrager sollte der Toellner-Signalverstärker in Zukunft durch das Gerät „Hubert A1110-05-QE“ ersetzt werden. Dieser Signalverstärker kann Spannungen von ± 100 Volt und Ausgangsströme von ± 5 A liefern. Die Primärseite des Übertragers wird entweder über BNC-Kabel oder über normale Leitungen vom Ausgang des Signalverstärkers gespeist. Zur Kontrolle und für eine spätere Erweiterung des Messplatzes gelangt das Signal zum Kanal 2 des Oszilloskops. Das Ausgangssignal des Frequenzgenerators gelangt über ein BNC-Kabel von der Rückseite des MDO34 zum Eingang des Signalverstärkers und zur Kontrolle zusätzlich an Kanal 1 des Oszilloskops. Für eine verwertbare Anzeige aller Kurvenverläufe kann die Auto-Set-Taste genutzt werden. Insbesondere, wenn die Abläufe automatisiert werden sollen, ist ein robustes Signal am Eingang 1 notwendig, da auf diesen Kanal getriggert wird.

Der Primärstrom wird mit dem Stromtastkopf TCP202 über einen TPA-BNC-Adapter mit dem Oszilloskop an Kanal 3 verbunden. Das Oszilloskop erkennt den Tastkopf und stellt die Anzeige in mA oder A ein. Der Nullpunkt des Stromtastkopfes kann über ein Potentiometer am Anschlussgehäuse abgeglichen werden, oder viel komfortabler über ein Menü am Oszilloskop. Der Abgleich ist notwendig, da der Stromtastkopf nur DC-gekoppelt betrieben werden kann und eine Verschiebung des Nullpunktes eine Verschiebung der Hysteresekurve auf der X-Achse zur Folge hat (siehe Abb. 10).

Die Spannung auf der Sekundärseite des Übertragers wird über einen Spannungstastkopf am Kanal 4 des Oszilloskops angezeigt. Kanal 3 und Kanal 4 sind also die entscheidenden Signale zur Darstellung der Hysteresekurve.

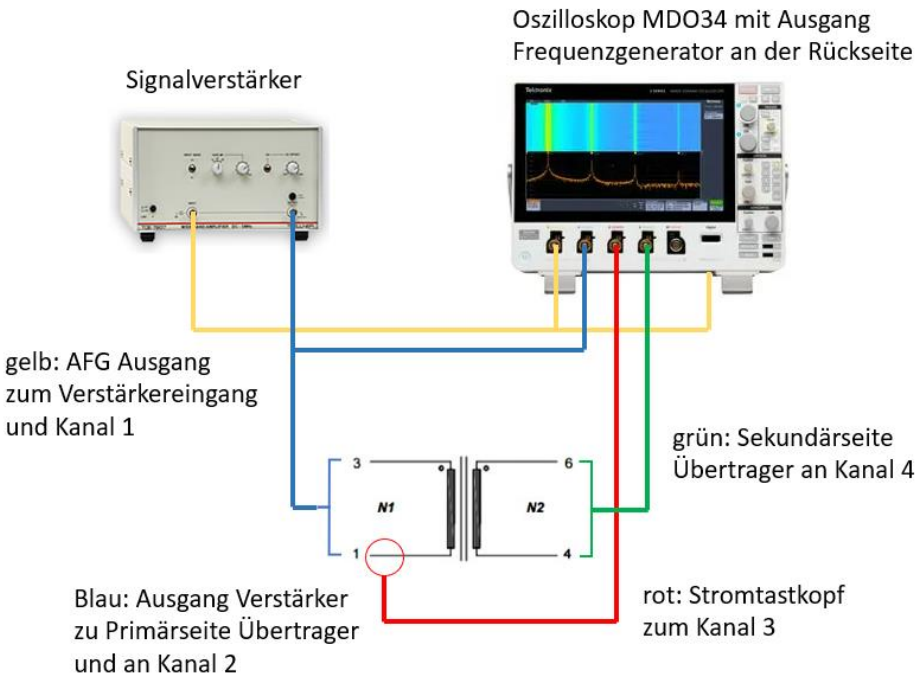


Abbildung 12 Übersicht Mess- und Signalleitungen

Um einen Datentransfer zwischen Oszilloskop und Laptop herstellen zu können, wird über einen USB-Netzwerk-Adapter ein Netzwerkkabel mit dem Netzwerk-Switch verbunden. Der Switch wurde installiert, um zukünftig weiteres Testequipment einbinden zu können. Das Oszilloskop wird ebenfalls über ein Netzwerkkabel mit dem Switch verbunden. Für eine stabile Kommunikation ist die feste Vergabe der IP für PC und Oszilloskop notwendig.

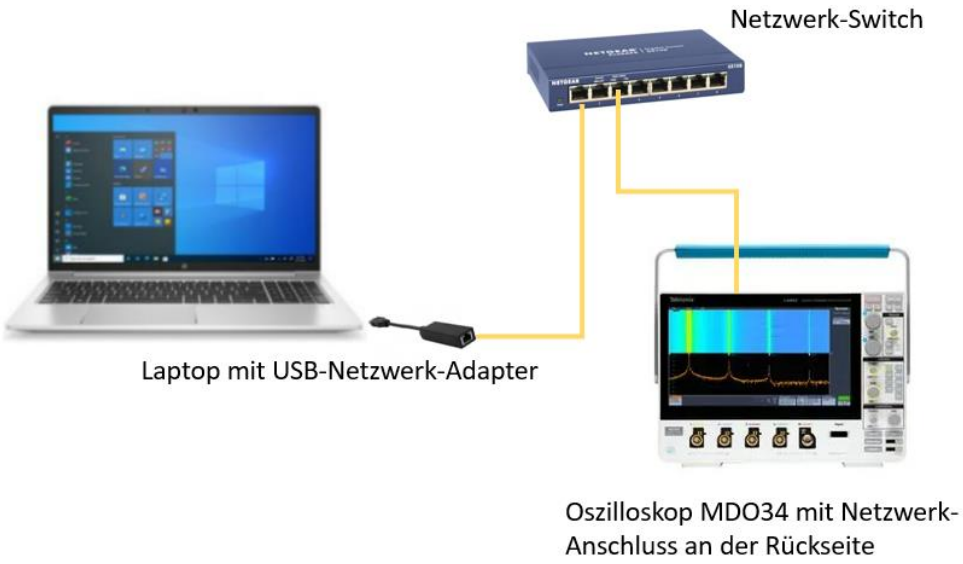


Abbildung 13 Übersicht Netzwerkverbindungen

2.4 Ablauf der Messung

Es wird die manuelle Messung beschrieben. Im nächsten Kapitel wird diese Messung weitgehend automatisiert.

- das Bauelement anschließen (siehe Abb. 12)
- Oszilloskop und Signalverstärker anschalten
- Nullabgleich des Stromtastkopfes
- Frequenzgenerator aktivieren
    - Wellenform, Frequenz, Spannung einstellen
- Auto-Set-Taste drücken oder Anzeige manuell einstellen
    - Anzeige auf mindestens eine Periode
    - für eine Mittelung von mehreren Durchgängen unter Acquisition „Average“ mit 64 Samples wählen
    - Bandbreite der Kanäle auf 20 MHz und Kopplung auf AC stellen
    - auf eine ausreichende vertikale Anzeige bei Kanal 3 (Strom) achten
- Acquisition auf Single / Einzelmessung, damit die Messungen des gleichen Zeitpunktes erfasst werden

Im XY-Betrieb können nur die Kanalpaare CH1/CH2, CH3/CH4, REF1/REF2 oder REF3/REF4 dargestellt werden. Es ist keine Anzeige des MATH-Kanals möglich. Deswegen muss der MATH-Kanal gespeichert werden.

- MATH-Kanal aktivieren und über Advanced INT(CH4) auswählen (Integration des Signals von Kanal 4)
- Kanal 3 auf REF1 speichern
- den MATH-Kanal auf REF2 speichern
- in den XY-Betrieb wechseln und alle Kanäle außer REF1/REF2 deaktivieren

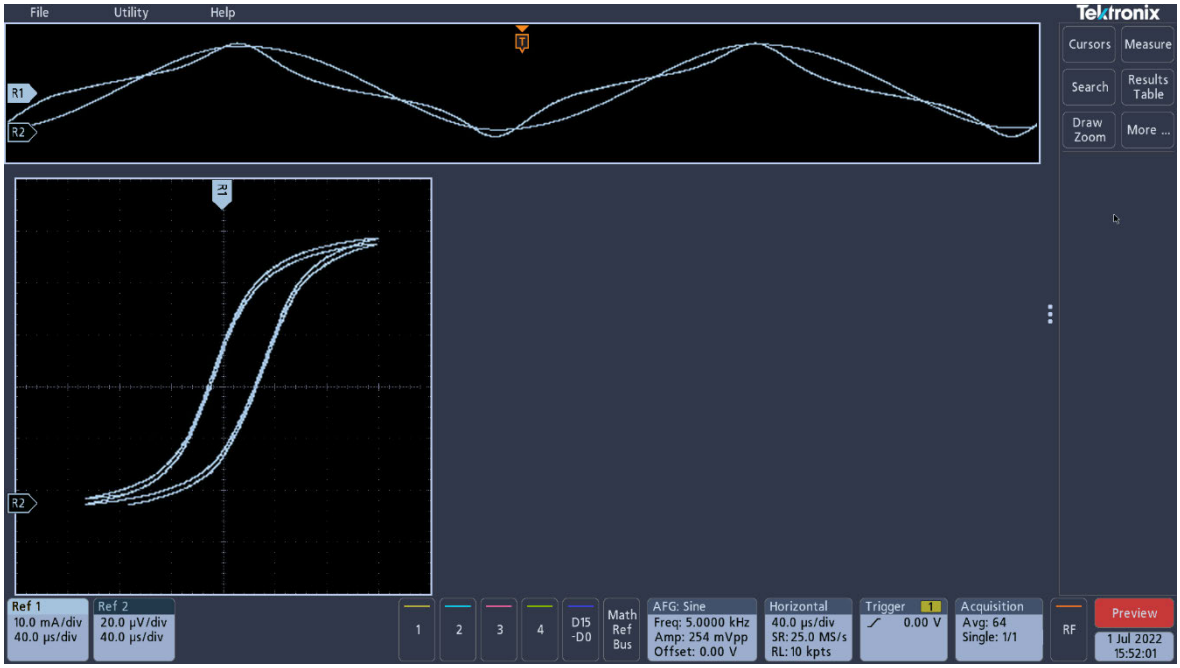


Abbildung 14 Anzeige der Hysteresekurve im XY-Betrieb

Die angezeigte Kurve zeigt den Primärstrom auf der X-Achse und die integrierte Sekundärspannung auf der Y-Achse an.

2.5 Beobachtungen

Es sind sehr umfangreiche Einstellvorgänge notwendig. Insbesondere beim „Herantasten“ an einen aussagekräftigen Kurvenverlauf ist es mühevoll, die beiden Kanäle für die XY-Darstellung immer wieder neu speichern zu müssen. Eine Vollbildanzeige der Kennlinie ist nicht möglich. Die angezeigte Kennlinie ist nicht die BH-Kurve, sondern stellt ∫U in Abhängigkeit des Primärstroms dar. Außerdem zeigt die Messung einen Versatz in Y-Richtung.

Der Aufbau soll die externe Darstellung der Kennlinie auf einem PC ermöglichen. Die dazu notwendigen Abläufe werden im nächsten Kapitel behandelt.

3 Automatisierung des Messablaufs

Im vorherigen Kapitel wurde gezeigt, dass die Bedienung des Oszilloskops viele Einzelschritte erfordert und die dargestellte Hysteresekurve im XY-Betrieb nicht akzeptabel ist. Deswegen werden immer wiederkehrende Bedienhandlungen automatisiert und eine externe Darstellung der Hysteresekurve wird implementiert. Als Programmiersprache soll Python verwendet werden.

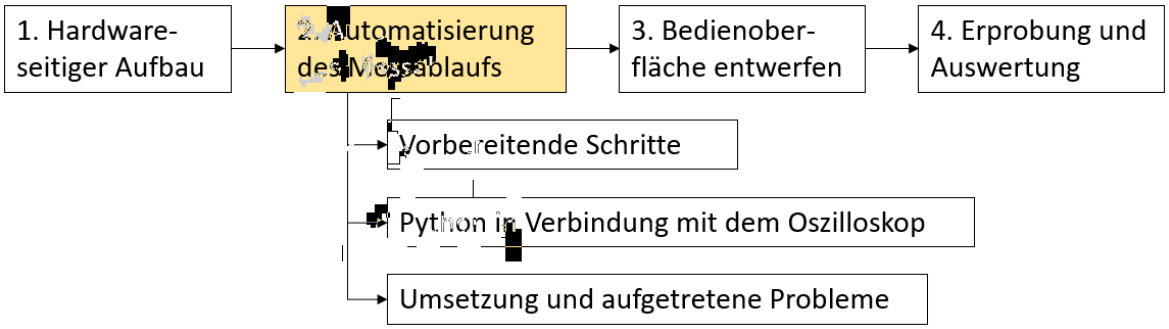


Abbildung 15 Zweiter Entwicklungsabschnitt

In diesem Kapitel wird zunächst die Software vorgestellt, die für die Umsetzung des Projektes notwendig ist. Dann erfolgt eine Übersicht von Python-Funktionen, die zum Verständnis des Programmcodes im Anhang erforderlich sind. Anschließend erfolgt die Programmierung, so dass eine Hysteresekurve auf dem PC angezeigt wird, aber noch ohne graphische Oberfläche. Das Python-Programm wird so erstellt, dass es sich einfach in eine graphische Oberfläche einfügen lässt. Alle aufgetretenen Probleme und deren Lösungsansätze sind ebenfalls dokumentiert.

3.1 Vorbereitende Schritte

Für die Kommunikationen zwischen PC und Oszilloskop sind einige Vorbereitungen, d.h. die Installation von Software, notwendig. In den nächsten Abschnitten sind Informationen über

- notwendige Gerätetreiber
- Befehle zur Kommunikation mit Messgeräten (SCPI)
- Entwicklungsumgebung Thonny

zu finden.

3.1.1 NI-VISA

NI-VISA ist eine Programmierschnittstelle zur Steuerung von Messgeräten über verschiedene Medien wie Ethernet, USB, GPIB und die serielle Schnittstelle (7). Unter (8) kann die notwendige Software heruntergeladen werden, die für das Senden und Empfangen von Befehlen zwischen PC und Oszilloskop erforderlich ist. Momentan ist die Version 21.5 für Windows 10 verfügbar. Während der Installation erscheint eine Abfrage, welche Optionen installiert werden sollen. Die Vorauswahl der zu installierenden Objekte sollte beibehalten werden.

3.1.2 SCPI-Befehle

SCPI (standard commands for programmable instruments) ist ein internationaler Standard für die Steuerung von Testequipment, wie z.B. Signalgeneratoren, Oszilloskopen und Netzteilen (9). Nicht jedes Mess- und Testgerät lässt sich über SCPI-Befehle ansprechen. Die genaue Syntax der Befehle ist von der Implementierung des Herstellers abhängig und kann dem entsprechenden Programmierhandbuch des Gerätes entnommen werden. Im Fall des Tektronix-Oszilloskops MDO34 basieren die Befehle auf dem SCPI-Standard. Für die Anwendung der Befehle ist die Recherche im fast 1000-seitigen „3 Series MDO Oscilloscopes Programmer Manual“ notwendig. Dieses Handbuch ist unter (10) erhältlich.

Es folgt ein einfaches Beispiel für die Syntax der Befehle: Die Kopplung (AC/DC) eines Messwertaufnehmers am Kanal 1 des Oszilloskops soll eingestellt werden. Bei Bedienung direkt am Gerät setzt man die Kopplung auf DC mit ① rechte Maustaste auf den Kanal 1, ② Klick auf DC, anschließend Klick außerhalb des Menüs auf die freie Bildschirmfläche (Menü schließt sich):

Abbildung 16 Kopplung am Oszilloskop einstellen

Die Syntax für einen SCPI-Befehl zur Fernsteuerung findet man im Programmer Manual (10 S. 216):

Syntax `CH<x>:COUPling {AC|DC|DCREJect}`
`CH<x>:COUPling?`

Abbildung 17 Syntax für Kopplung einstellen / auslesen

In <> wird dabei ein notwendiger Parameter hinterlegt, die geschweiften Klammern {} geben eine Auswahl an Parametern vor. Alle Großbuchstaben sind notwendig, die Kleinbuchstaben sind optional für bessere Lesbarkeit der Befehle. Werden die Kleinbuchstaben verwendet, müssen sie vollständig übergeben werden. Der SCPI-Befehl für das Setzen des Kanal 1 auf DC-Kopplung wäre: **CH1:COUPling DC** oder **CH1:COUP DC**. Soll die Kopplung des Kanals abgefragt werden, sendet man **CH1:COUPling?** oder **CH1:COUP?** an das Gerät und liest die Antwort.

Für das Senden von Befehlen und Rücklesen der Antworten kann man die Software „NI-MAX“ verwenden, die bereits durch das NI-VISA-Paket (siehe Abs. 3.1.1) installiert wurde. Das Programm findet man unter „Start → Programme → National Instruments“. Durch Öffnen von ③ „Mein System → Geräte und Schnittstellen“ und Erweitern des Ordners „Netzwerkgeräte“ werden alle verfügbaren Messgeräte im Netzwerk angezeigt.

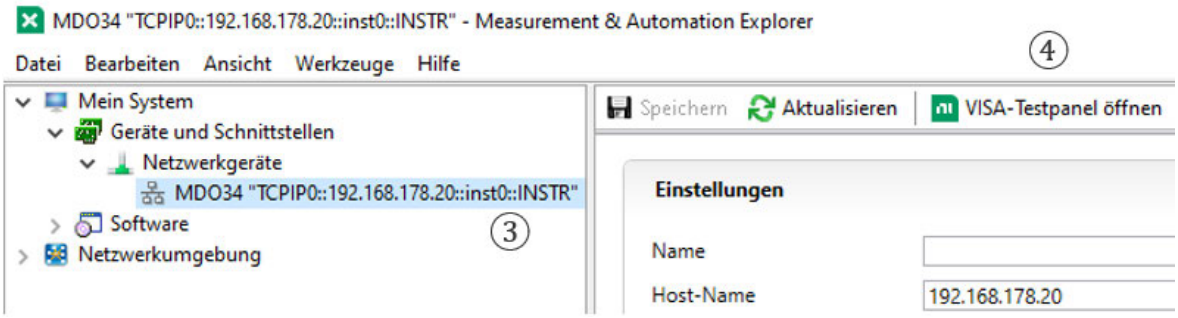


Abbildung 18 Anzeige der verfügbaren Geräte

Durch Klick auf ④ „VISA-Testpanel öffnen“ gelangt der Nutzer zu einem Fenster, in dem unter „Input/Output“ SCPI-Befehle gesendet und Antworten gelesen werden können.

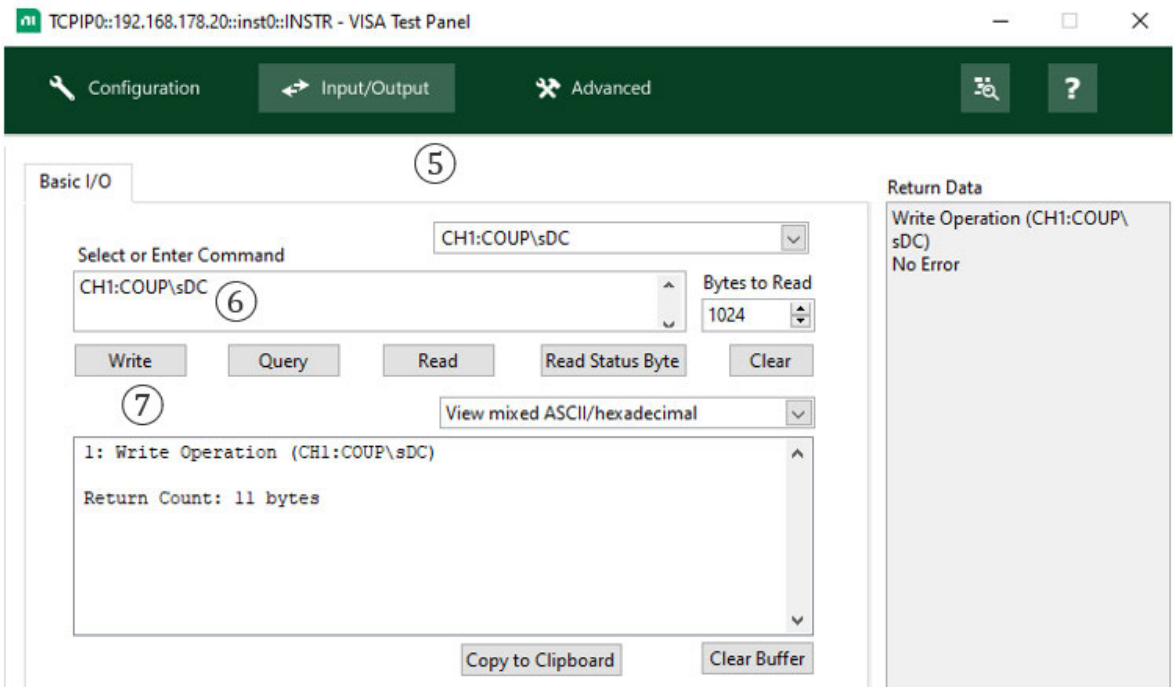


Abbildung 19 Senden von SCPI-Befehlen

Als erstes muss der Tab ⑤ „Input/Output“ aktiviert werden. Dann wird ⑥ der Befehl **CH1:COUP DC** eingegeben, das Leerzeichen wird dabei automatisch zu \s verändert. Mit ⑦ wird der Befehl an das Oszilloskop gesendet. Um die Ausführung des Befehls zu verifizieren, kann mit **CH1:COUP?** den aktuellen Status lesen ⑧ und erhält dabei folgende Antwort (blau markiert):

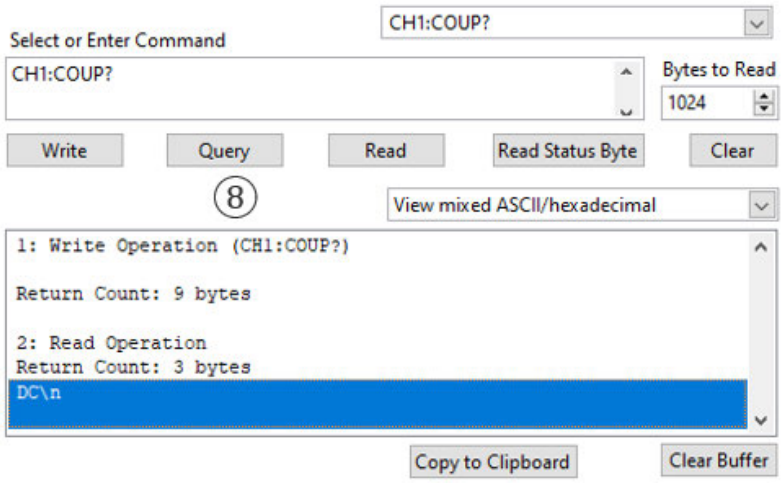


Abbildung 20 SCPI - Auslesen von Werten

Es wurde erfolgreich auf DC-Kopplung umgeschaltet (\n ist das Ende-Zeichen). Der Button „Query“ vereinigt dabei die Write- und Read-Operation.

3.1.3 Entwicklungsumgebung Thonny

Das Senden von Befehlen mit NI-MAX ist nur sinnvoll, um einzelne SCPI-Kommandos zu testen oder nachzuvollziehen, ob überhaupt eine Kommunikationsverbindung zum Oszilloskop besteht. Für eine komfortable Fernsteuerung sendet man Befehlsfolgen mittels einer Programmiersprache. Python hat hier entscheidende Vorteile, es gibt sehr viele Bibliotheken, die dem Anwender eine einfache Programmierung ermöglichen. Eine Übersetzung entfällt, da Python eine Interpreter-Sprache ist (Übersetzung zur Laufzeit in Maschinensprache). Zum Programmieren benötigt man eine Entwicklungsumgebung. Im Folgenden wird dafür die IDE (Integrated Development Environment) Thonny genutzt.

Das Programm Thonny ist einfach und übersichtlich gestaltet und ist als Download unter (11) erhältlich. Thonny beinhaltet Python bereits (momentan in der Version 3.7.9), so dass keine weitere Installation von Python notwendig ist. Nach der Installation sieht die Oberfläche so aus:

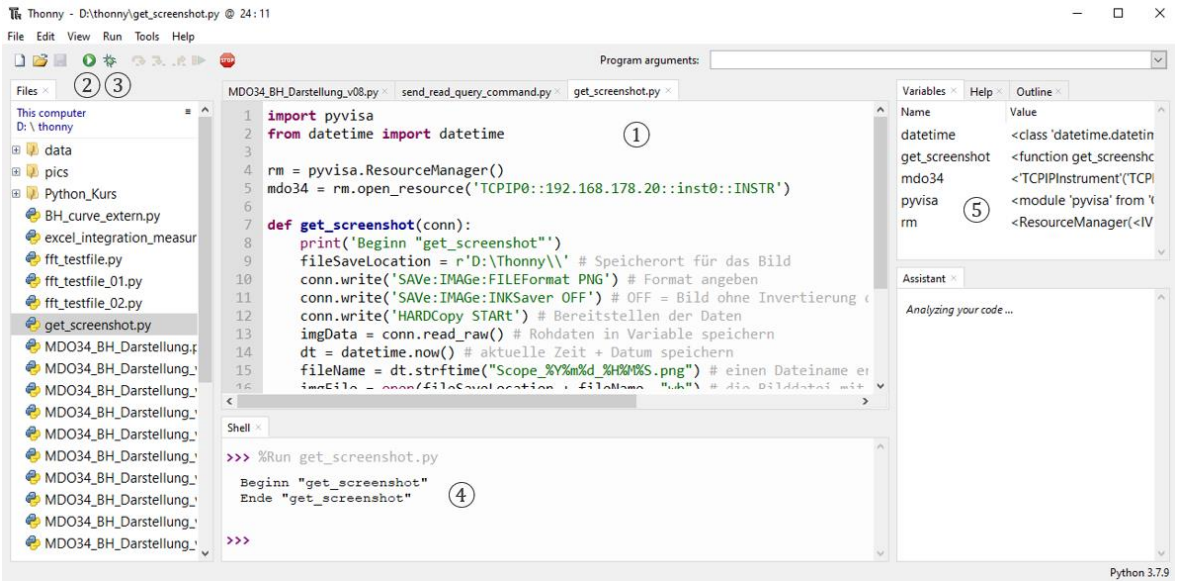


Abbildung 21 Entwicklungsumgebung Thonny

Der Programmcode wird im ① Hauptfenster bearbeitet. Mit dem ② Play-Zeichen wird der geschriebene Programmcode komplett abgearbeitet. Im ③ Debug-Mode kann das Programm auf Fehler untersucht werden. Print-Ausgaben und Fehlermeldungen erscheinen in der ④ Shell. Werte von Variablen sind im ⑤ rechten Fenster sichtbar.

Oft ist es notwendig, weitere Bibliotheken zu installieren. Zum Beispiel ist es für das Senden von SCPI-Befehlen notwendig, die Bibliothek PyVISA-py zu installieren. Dazu wird der Bibliotheksverwalter über „Tools → Manage packages…“ geöffnet, dann gibt man in der Suchleiste „pyvisa-py“ ein drückt anschließend auf „Search on PyPI“. Der Eintrag PyVISA-py wird angeklickt und die Bibliothek installiert. Die korrekte Installation von PyVISA kann anschließend über die System-Shell („Tools → Open System Shell“) mit dem Befehl „pyvisa-info“ überprüft werden. Die System-Shell kann auch für das Ausführen

von fertigen Python-Skripts mit dem Befehl „python <NameDesSkripts>.py“ genutzt werden.

3.2 Python in Verbindung mit dem Oszilloskop

In diesem Abschnitt wird auf Python-spezifische Details eingegangen, die zum Verständnis des Programmcodes im Anhang notwendig sind. Grundlagen von Python bzw. einer anderen Programmiersprache werden vorausgesetzt. Dieser Abschnitt kann auch als Nachschlagewerk bei der Entwicklung eigener Programme für Testequipment verwendet werden.

3.2.1 Bibliotheken

Der Grundumfang von Python kann durch Bibliotheken erweitert werden. Dadurch können unter anderem Berechnungen vereinfacht und die Bearbeitungsgeschwindigkeit erhöht werden. Bibliotheken müssen in der Python-Installation vorhanden sein, die Installation erfolgt wie in Abs 3.1.3 beschrieben. Für die Nutzung im Programmcode müssen diese dann importiert werden.

```
1 import pyvisa
2 import numpy as np
3 from scipy import integrate
```

Abbildung 22 Import von Bibliotheken

In Zeile 1 wird pyvisa importiert und im Programmcode auch mit pyvisa angesprochen (siehe Abs. 3.2.3). In Zeile 2 wird numpy importiert und mit dem Alias np versehen. Im Programmcode wird dann immer np anstatt numpy verwendet. Bei bestimmten Bibliotheken hat sich der Import mit einem Alias als Standard durchgesetzt. In Zeile 3 wird nur ein Teil einer Bibliothek importiert, da diese sehr umfangreich ist. Wegen der Übersichtlichkeit erfolgt der Bibliotheksimport immer am Anfang des Quellcodes.

3.2.2 Variablen definieren

Der zweite Teil des Programmcodes umfasst Variablen. Somit kann ein häufig vorkommender Ausdruck zentral an einer Stelle geändert werden. Für die Wiedererkennung im Quellcode kann man Variablen immer großschreiben. Im Hinblick auf die Entwicklung der graphischen Oberfläche wird der Variablenteil genutzt, um Werte für Checkboxen, Eingabefelder usw. vorzugeben.

```
21 # Adresse für connect()
22 ADDRESS_MDO34 = 'TCPIP0::192.168.178.20::inst0::INSTR'
23 # Alias für Kanäle
24 CH_AFG = 'CH1'
25 CH_U_PRIM = 'CH2'
26 CH_CURRENT = 'CH3'
27 CH_U_SEC = 'CH4'
```

Abbildung 23 Beispiel für Variablen

In Zeile 22 wird die Instrumenten-Adresse für das Oszilloskop fest vorgegeben (siehe Abs. 3.2.3). In den Zeilen 24 bis 27 erlauben die Variablen eine nachträgliche Änderung der Anschlüsse der Messsonden an die einzelnen Kanäle des Oszilloskops (siehe Abs. 2.3 und 2.4).

3.2.3 Kommunikation mit Testequipment

Die Kommunikation zwischen PC und Oszilloskop soll über Ethernet stattfinden. Dazu ist am MDO34 eine feste IP notwendig. Im Netzwerk befindet sich momentan nur PC, Netzwerk-Switch und Oszilloskop (siehe Abs. 2.3). Die Kommunikation kann, wie in Abs. 3.1.2 beschrieben, getestet werden. Informationen zur Herstellung der Verbindung und der Fehlersuche findet man in der PyVISA-Dokumentation online unter (12). Durch die Verwendung einer festen IP reduziert sich der Programmcode auf zwei Zeilen:

```
51 # rm gibt den Speicherort des Visa-Treibers zurück
52 rm = pyvisa.ResourceManager()
53 # eine Verbindung aufbauen und zurückgeben
54 conn = rm.open_resource(ADDRESS_MDO34)
```

Abbildung 24 Herstellung der Kommunikation mit dem MDO34

Möchte man Befehle senden oder Daten erhalten, greift man auf die Verbindung wie folgt zu (vergleiche Abs. 3.1.2):

```
1 # Daten schreiben
2 conn.write('CH1:COUP DC')
3
4 # Daten lesen, Klammern bleiben leer!
5 value_read = conn.read()
6
7 # Daten lesen und schreiben
8 value_query = conn.query('CH1:COUP?')
```

Abbildung 25 Daten schreiben und lesen mit Python

In Zeile 2 wird Kanal 1 auf DC-Kopplung gesetzt. Zeile 8 liest die aktuelle Kopplung in die Variable „value_query“ ein. Da Query die Kombination aus Write und Read ist, benötigt man den Programmcode aus Zeile 5 nicht.

3.2.4 Unterprogramme / Funktionen

Um das Programm leserlich zu gestalten und auch eine Fehlersuche zu vereinfachen, kann man häufige genutzte Abläufe in Unterprogramme auslagern. Es können mehrere Parameter übergeben werden und es ist auch möglich, mehrere Rückgabewerte an die aufrufende Funktion zu übertragen. Bei der Erstellung eines Unterprogramms orientiert man sich an den Bedienhandlungen, die der Nutzer an der Bedienoberfläche vorgenommen hätte. Als Beispiel wird das Aktivieren des Funktionsgenerators (AFG – Arbitrary Function Generator) mit bestimmten Einstellungen verwendet:

```
92  # Arbitrary Frequenz Generator (AFG)
93  # AFG anschalten
94  def set_AFG(conn, waveform, frequency, offset, amplitude):
95      conn.write('AFG:OUTP:LOA:IMPED HIGHZ') # HighZ da höhere Spannung mgl.
96      conn.write('AFG:FUNC {}'.format(waveform)) # Waveform
97      conn.write('AFG:FREQ {}'.format(frequency)) # Frequenz in Hz
98      conn.write('AFG:OFFS {}'.format(offset)) # Offset in V
99      conn.write('AFG:AMPL {}'.format(amplitude)) # Amplitude in V
100     conn.write('AFG:OUTP:STATE ON') # AFG an
```

Abbildung 26 AFG einstellen und aktivieren

Es ist sowohl die Übergabe der Kommunikationsverbindung („conn“) notwendig als auch die einzustellende Wellenform, Frequenz, Amplitude und Offsetspannung. Alle notwendigen SCPI-Befehle sind dem Programmer-Manual (10) zu entnehmen.

```
312     # AFG anschalten
313     print('AFG (Sinus, keine Offset) anschalten\t')
314     freq = input('Bitte Frequenz in kHz eingeben: ')
315     freq = float(freq) * 1000 # Faktor 1000 da Übermittlung in Hz
316     vpp = input('Bitte Spannung nach Verstärker in Vpp eingeben: ')
317     vpp = float(vpp)
318     vpp = vpp/SA_RATIO
319     set_AFG(mdo34, 'SINE', freq, 0.0, vpp)
```

Abbildung 27 Unterprogramm aufrufen

Der Aufruf erfolgt in Zeile 319, nachdem vom Nutzer die Frequenz und sowie die Spitzen-Spitzen-Spannung am Ausgang des Signalgenerators abgefragt wurden. Da der Signalgenerator einen Spannungsverstärkungsfaktor aufweist, muss die eingestellte Spannung des Frequenzgenerators um diesen Faktor niedriger sein. Der Faktor SA_RATIO wurde in den Variablen (siehe Abs. 3.2.2) hinterlegt.

3.2.5 Fehlerbehandlung

Die Behandlung von aufgetretenen Fehlern kann in 2 Kategorien eingeteilt werden:

- Ausnahmebehandlung mit Python
- Auslesen von Informationen, die im Fehlerspeicher des Oszilloskops hinterlegt sind

Ausnahmebehandlung mit Python

Das Try-Except-Konzept von Python ermöglicht, Programmcode versuchsweise auszuführen und bei einem aufgetretenen Fehler entsprechend zu reagieren. Umgesetzt wird die Ausnahmebehandlung beim Aufbau einer Verbindung zum Oszilloskop. Wenn es einen Time-Out-Error gibt, kann das verschiedene Fehlerursachen haben (Oszilloskop nicht angeschaltet, Netzwerkverbindung funktioniert nicht, etc.). Das Programm gibt dann einen Hinweis aus, bei erfolgreicher Verbindung wird normal im Programmablauf fortgefahren.

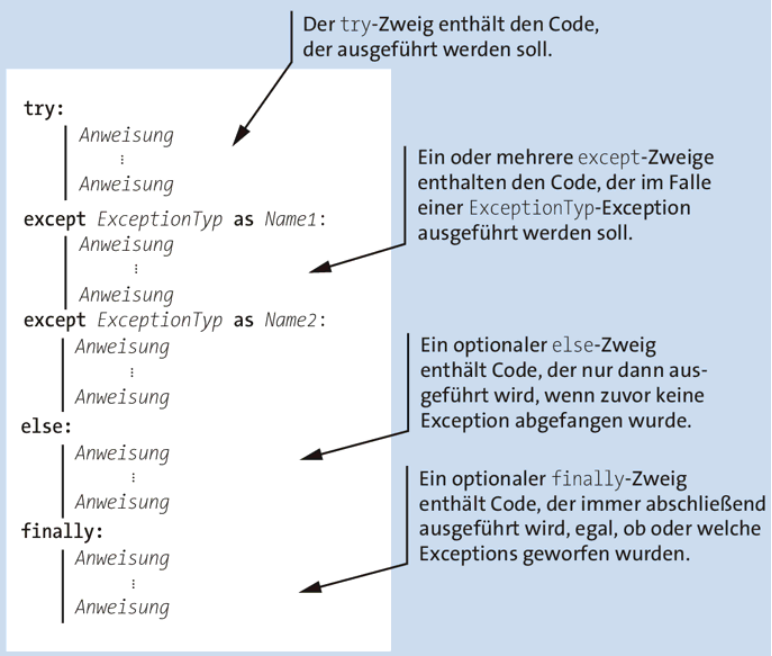


Abbildung 28 Das vollständige Try-Except-Konzept von Python (13 S. 389)

Die zweite Möglichkeit besteht im Auslesen von Fehlern, die innerhalb des Oszilloskops aufgetreten sind:

Status- und Eventhandler des Oszilloskops

Es gibt verschiedene Ereignisse, die innerhalb des Oszilloskops auftreten können. Diese Events sind bestimmten Kategorien zugeordnet und in der nachfolgenden Übersicht zu finden:

Bit	Function	
7 (MSB)	PON	**Power On.** Shows that the oscilloscope was powered on. On completion, the diagnostic self tests also set this bit.
6	URQ	**User Request.** Indicates that an application event has occurred. *See note.
5	CME	**Command Error.** Shows that an error occurred while the oscilloscope was parsing a command or query.
4	EXE	**Execution Error.** Shows that an error executing a command or query.
3	DDE	**Device Error.** Shows that a device error occurred.
2	QYE	**Query Error.** Either an attempt was made to read the Output Queue when no data was present or pending, or that data in the Output Queue was lost.
1	RQC	**Request Control.** This is not used.
0 (LSB)	OPC	**Operation Complete.** Shows that the operation is complete. This bit is set when all pending operations complete following an *OPC command.

Abbildung 29 Ereignisse innerhalb des Oszilloskops (10 S. 852)

Ein Ereignis passiert als erstes das Register DESER. Das Register filtert die eingehenden Events mit einer bitweisen UND-Verknüpfung. Die Standardeinstellung ist, dass alle Bits auf 1 gesetzt sind, somit werden alle Events zum Register SESR durchgereicht. Dort wird dann das entsprechende Bit gesetzt und eine genaue Fehlerbeschreibung in der Event-Queue hinterlegt. Alle Fehlerbeschreibungen sind im Programmierhandbuch (10) ab S.864 zu finden. Damit endet die Fehlerbehandlung für Ethernet- und USB-Verbindungen.

Die anderen Register sind nur für GPIB-Verbindungen relevant, da dort Service Requests ausgelöst werden können (14). Das Register ESER dient als Filter, um das Bit 5 ESB im Register SBR zu setzen (oder auch nicht). Das Bit 4 MAV wird gesetzt, wenn sich Daten in der Output-Queue befinden (angefragte Daten können vom PC gelesen werden). Das Register SRER ist ein Filter, um das Bit 6 im Register SBR in Abhängigkeit von MAV und ESB zu setzen.

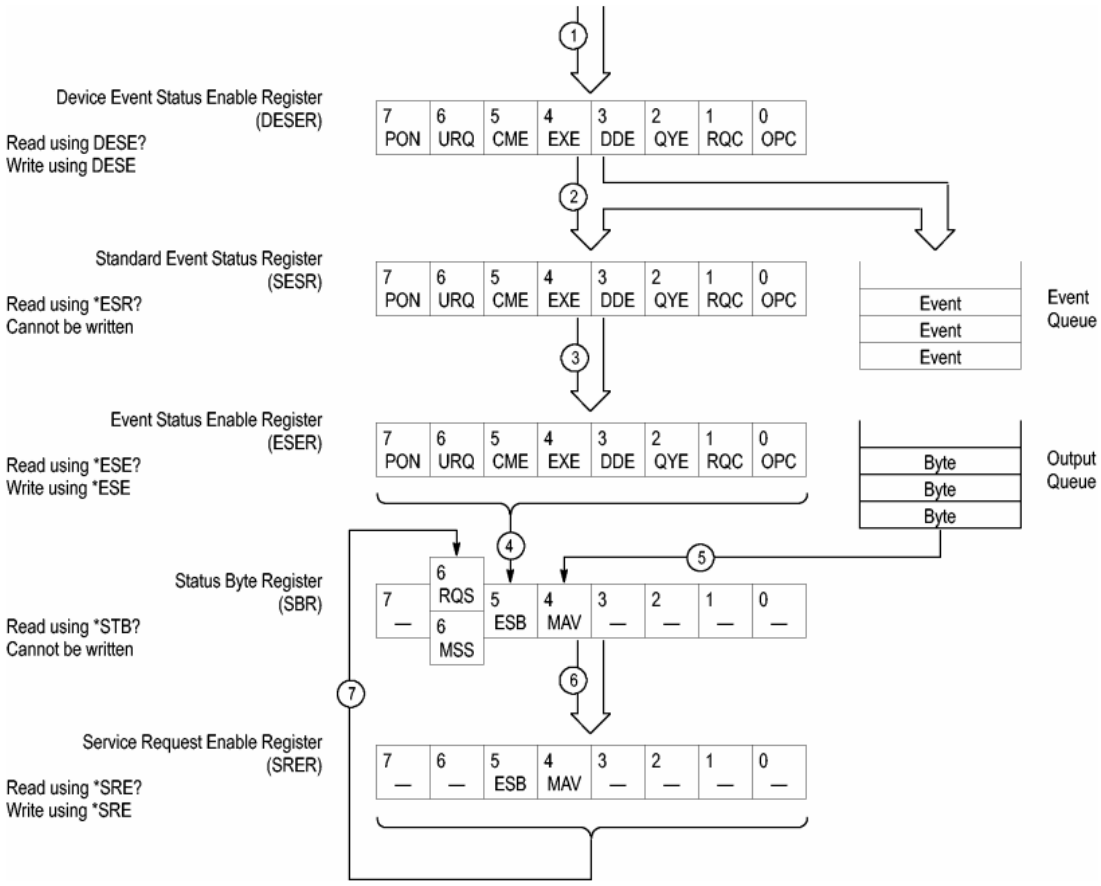


Abbildung 30 Ablauf der Fehlerspeicherung im MDO34 (10 S. 856)

Die Fehlerbehandlung innerhalb eines Python-Skripts umfasst daher das Löschen von bisher aufgetretenen Fehlern, also das Register SESR und die Eventqueue, mit dem Befehl *CLS. Dann erfolgt die Ausführung von Anweisungen. Danach wird der Inhalt der Register SESR und die Eventqueue mit den Befehlen *ESR? und ALLEV? ermittelt und ausgewertet. Dieses Verfahren ist beim Einlesen von Messwerten (Cycle Mean, Min, Max, etc.) wichtig, um die Plausibilität zu prüfen.

3.2.6 Warten auf das Ende einer Operation

In bestimmten Fällen muss eine Operation erst beendet werden, bevor eine nächste Operation starten kann. Beispielsweise muss Auto Set (automatische zeit- und amplitudenrichtige Darstellung von Signalverläufen) erst beendet sein, bevor man Messwerte eingelesen werden. Dafür gibt es zwei verschiedene Möglichkeiten:

- dem Oszilloskop ausreichend Zeit für die Beendigung des Vorgangs einräumen
- das Oszilloskop teilt das Ende der Operation mit

Option 1: Zeit für Ausführung einräumen

Dafür ist der Import der Bibliothek time erforderlich („import time“). Danach wird mit dem Befehl time.sleep(<Pausenzeit in Sekunden>) für die angegebene Zeit der Programmablauf pausiert.

Option 2: Oszilloskop gibt Wert nach Beendigung der Operation zurück

Mit dem Befehl *OPC? gibt das Oszilloskop den Wert 1 zurück, wenn alle bestimmte Operationen ausgeführt wurden. Laut Programmierhandbuch (10 S. 858) sind nur einige Operationen fähig, das OPC-complete Bit zu senden. Die Umsetzung in Python-Code ist sehr einfach, man liest das OPC-Bit einfach in eine Variable ein [r = conn.query('*OPC?')]. Die Variable braucht nicht ausgewertet werden, da die Programmzeile erst abgearbeitet wird, wenn das Oszilloskop die Operation beendet hat.

3.2.7 Erhalten von Waveform-Daten

Möchte man die Daten eines Signals importieren, kann man sich am Beispiel „Waveform Transfer, Example 1“ des Programmierhandbuchs (10 S. 921 ff.) orientieren. Programmierbeispiele sind auch auf der Tektronix-GitHub-Webseite verfügbar (15). Beim Datenimport von mehreren Kanälen ist es sinnvoll, den Betriebsmodus auf „Single/Seq“ zu setzen, um Daten innerhalb desselben Abtastzeitraums zu erhalten. Es gibt verschiedene Einstellmöglichkeiten, wie die zu übertragenden Daten formatiert sind. Diese sind im Programmierhandbuch tabellarisch zusammengefasst (10 S. 269) und werden auch detailliert beschrieben (10 S. 268). Im Python-Programmcode (siehe Anhang, Teil 2, Funktion „get_waveform_data_one_period“) wurde „SRIBINARY“ gewählt, da die Dateilänge gegenüber dem ASCII-Format kürzer ist. Außerdem kann die Anzahl Bytes pro Messpunkt festgelegt werden, für 1 Bit gibt es nur 25 Digits Auflösung pro vertikalen Teiler, bei 2 Bit sind es 6400 Digits/Teiler. Hier wurde zugunsten der höheren Auflösung entschieden.

Nach Auswahl des Kanals wird festgelegt, in welchem Bereich Daten importiert werden. Das bedeutet die Angabe des Start- und Endpunktes. Der Bereich, der abgerufen wird, bezieht sich auf den internen Speicher des Oszilloskops, es ist insbesondere bei hohen Speichertiefen und hohen Frequenzen nicht immer der am Bildschirm angezeigte Ausschnitt. Die binären Daten werden mit dem Befehl CURVE? abgerufen, dabei können noch Optionen übergeben werden, die eingehend auf der PyVISA Dokumentation erläutert werden (16). Zu diesem Zeitpunkt hat man vertikale Daten vorliegen, aber ohne Bezug auf die Höhe der Spannung und dem zeitlichen Abstand der einzelnen Datenpunkte. Aus diesem Grund müssen noch weitere Informationen vom Oszilloskop angefordert werden.

Für die vertikalen Daten (Höhe der Spannung / des Stroms) benötigt man noch:

- Offset in Volt
- Position in div
- Faktor, wieviel Volt oder Ampere ein Digit der Auflösung (wurde mit 2 Bit auf 6400 Digits/Teiler festgelegt) entspricht

Nach Umwandlung der Rohdaten in eine Fließkommazahl werden die Spannungswerte mit den drei oben genannten Daten berechnet (10 S. 108). Nachdem die vertikalen Daten fertig berechnet wurden, werden weitere Daten für die horizontale Achse (Zeitdaten) benötigt. Dafür ist die Abfrage eines Startpunktes und eines horizontalen Abstandes zwischen zwei Datenpunkten notwendig. Mit diesen Daten wird dann ein Array zur Speicherung der horizontale Zeitdaten gebildet (10 S. 107).

3.2.8 Die Bibliothek numpy

Die Bibliothek numpy stellt Funktionen für numerische Berechnungen bereit (17 S. 93 ff.). Nachfolgend sind einige wichtige Funktionen aufgeführt, weitere Informationen sind in der Numpy-Dokumentation erhältlich (18).

Array mit np.linspace erzeugen

Syntax: np.linspace(<Start>, <Stop>, <Schrittweite>, endpoint={False | True})

„Endpoint“ gibt an, ob „Stop“ der letzte Datenpunkt ist (wenn endpoint=True). Sonst ist der letzte Datenpunkt „Stop – Schrittweite“. Es wird ein numpy-Array erzeugt, eine auf die Bibliothek angepasste Version eines Arrays.

Arrays speichern

Mit np.savetxt(<Dateiname>, <Array>, delimiter=‘ ‘) wird ein Array in eine Textdatei gespeichert. Damit können bei Verwendung des Kommas als Zeichentrenner csv-Dateien erzeugt werden.

Weitere wichtige numpy-Funktionen

np.mean(<array>)	arithmetischen Mittelwert berechnen
np.size(<array>)	Größe eines Arrays ausgeben
np.abs(<array>)	gibt für jedes Element den Abstand zum Nullpunkt an, gilt auch für komplexe Zahlen
np.argmin(<array>)	gibt den Index des Minimums im Array zurück
np.argmax(<array>)	gibt den Index des Maximums im Array zurück
np.trapz(<Y-Achse>,<X-Achse>)	integriert nach der Trapezregel
np.concatenate(<Array_1>,<Array_2>)	fügt Arrays aneinander

Abbildung 31 Übersicht wichtiger numpy-Funktionen

3.2.9 Trapezintegration

Die Funktion np.trapz() aus Abb. 31 gibt eine Fließkommazahl zurück, es wird die gesamte Fläche unter dem übergebenen Array berechnet. Möchte man das Integral von Messdaten optisch darstellen, zum Beispiel weil eine gemessene Spannung integriert werden soll, benötigt man eine Integration von einem Element zum nächsten Element. Tektronix löst die interne Integration mit einer Berechnung nach der Formel:

$$y(n) = scale \sum_{i=1}^{n} \frac{x(i) + x(i-1)}{2} T$$

Abbildung 32 Interne Berechnung der Integration (19 S. 349)

Eine Integration kann mit Python durch die Funktion „scipy.integrate.cumulative_trapezoid(<Y-Array>, <X-Array>, initial=0) realisiert werden. Bei dem Y-Array werden die vertikalen Spannungsdaten übergeben, das X-Array beinhaltet die horizontalen Zeitdaten. Mit „initial=0“ gibt man einen ersten Datenpunkt vor, fehlt diese Angabe, hat das erzeugte Array einen Datenpunkt weniger als das Y-Array. Das kann beim Plotten von Daten (siehe Abs 3.2.11) Probleme verursachen. Die Integration erfolgt nach der „composite trapezoidal rule“ (20).

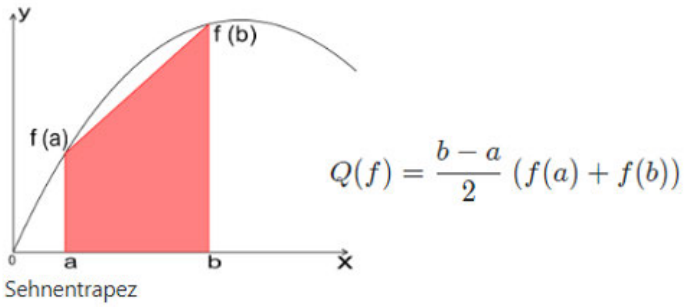


Abbildung 33 Sehnentrapezregel (21)

Da beide Formeln gleich sind, liefert die Integration mit Python dasselbe Ergebnis wie die Integrationsmethode des Oszilloskops.

3.2.10 FFT

Zeitlich abhängige periodische Signale können durch die Fourier-Transformation in sinusförmige Frequenzanteile zerlegt werden. Für abgetastete Signale heißt dieses Verfahren DFT (Diskrete Fourier Transformation). FFT (Fast Fourier Transformation) ist eine durch bestimmte Rechenalgorithmen optimierte DFT und wird vorzugsweise bei der elektronischen Datenverarbeitung implementiert. Die FFT mit Python besteht im kürzesten Fall incl. Rücktransformation aus nur 4 Zeilen Programmcode (17 S. 323 f.) und ist daher rela-

tiv einfach umzusetzen. Die FFT kann eingesetzt werden, um höherfrequente Anteile eines verrauschten Signals auszufiltern oder den Gleichanteil zu eliminieren.

Im Programmcode (siehe Anhang, Teil 2, Funktion „apply_fft_filter“) ist zunächst der Import einiger Bibliotheken (fft, ifft, fftfreq) aus „scipy.fft“ notwendig, anschließend findet mit fft(<Waveform_Zeitbereich>) die Transformation vom Zeitbereich in den Frequenzbereich statt. Die dabei entstehenden Daten bestehen aus einem Real- und einem Imaginärteil und enthalten noch keine Informationen bezüglich der Frequenz. Die Zuordnung der einzelnen Frequenzen zum Bildbereich erhält man mit „fftfreq(<Anzahl_Daten>, <Abtastzeit>)“. An dieser Stelle kann dann die Filterung von Daten erfolgen, zum Beispiel ist der Gleichanteil immer beim Index 0. Durch einen Vergleich der Frequenz mit einer vorgegebenen Grenzfrequenz kann Rauschen entfernt werden. Nach erfolgter Bereinigung im Frequenzbereich wird das Signal mit „ifft(<Waveform_Frequenzbereich>)“ in den Zeitbereich überführt.

3.2.11 Diagramme darstellen / Matplotlib

Für die Darstellung von Diagrammen eignet sich die Bibliothek Matplotlib. Der einfachste Plot ist mit drei Zeilen Code realisiert:

```
1 import matplotlib.pyplot as plt
2 # Definition der X-Daten und Y-Daten
3 plt.plot(<X-Daten>,<Y-Daten>)
4 plt.show()
```

Abbildung 34 Diagramm plotten

Um das Diagramm optisch ansprechender zu gestalten, kann man zusätzlich Titel anzeigen, die Achsen beschriften, ein Gitternetz anzeigen und die Größe des Anzeigefensters vorgeben. Weitere Informationen gibt es in der Dokumentation zu Matplotlib unter (22).

plt.savefig(<Dateiname.Endung>)	Speichern als png, pdf, jpg, eps, svg
plt.grid(True)	Gitternetz anzeigen
plt.title("Titel")	Titel anzeigen
plt.xlabel("X-Achse")	die X-Achse beschriften
plt.ylabel("Y-Achse")	die Y-Achse beschriften
plt.legend()	eine Legende anzeigen
plt.figure(figsize=(10,6))	eine Fenstergröße vorgeben

Abbildung 35 Matplotlib Optionen

Es ist außerdem möglich, die Farben, Linienstile und Linienbreiten zu gestalten. Dafür gibt es die Parameter color, linestyle (ls) und linewidth (lw), die dem plt.plot()- oder plt.grid()-Befehl übergeben werden können (siehe Abb.35). Es ist auch möglich, in LaTex-Notation (z.B. $\int$ für ein Integralsymbol) griechische Buchstaben und Sonderzeichen anzeigen zu lassen. Umfangreiche Informationen gibt es in der Dokumentation zu Matplotlib unter (22).

Sollen in einem Diagramm mehrere Graphen angezeigt werden, kann man vor dem plt.show()-Befehl einen weiteren plt.plot()-Befehl eingeben. Der Anzeigebereich richtet sich dann nach den größeren Werten, so das unter Umständen ein Datensatz mit sehr kleinen Werten nur als Gerade angezeigt wird. Das kann man durch die Definition einer zweiten Y-Achse bereinigen. Im nachfolgenden Programmcode wird ein Strom- und ein Spannungsverlauf mit 2 getrennten Y-Achsen dargestellt:

```
fig, ax1 = plt.subplots(figsize=(10, 6))
ax1.set_xlabel('time (s)')
ax1.set_ylabel('Strom [A]')
ax1.plot(time_data_current, current_data_filtered, color='orange')
ax2 = ax1.twinx()
ax2.set_ylabel('Spannung [V]')
ax2.plot(time_data_current, voltage_data_filtered, ls='--', color='blue')
plt.show()
```

Abbildung 36 Plot mit zwei Y-Achsen

Während der Anzeige eines Plots, also nach der Anweisung plt.show(), wird der Ablauf des Programmes blockiert, bis das Fenster geschlossen wird.

3.2.12 Speichern von Dateien / Screenshot

Um Arrays zu speichern, sei auf die entsprechende numpy-Funktion in Abs.3.2.8 verwiesen. Für das Speichern anderer Dateien, wie Bilddateien, wird ein Dateiobjekt erzeugt und dann eine Datei mit Schreib-Rechten geöffnet. Dann werden die Daten in das Dateiobjekt geschrieben und anschließend wird dieses geschlossen. Das Schließen darf nicht vergessen werden, da sonst ein Zugriff von anderer Stelle (Öffnen über Explorer) verhindert sein kann (13 S. 84 f.). Für die Generierung eines Zeitstempels im Dateinamen kann die Funktion strftime aus der Bibliothek datetime.datetime genutzt werden (13 S. 241 f.). Das Speichern eines Screenshots des Oszilloskops basiert auf dem Python-Code der Tektronix-GitHub-Seite (15). Nach Festlegen des Dateiformates und der Übermittlung, ob das Bild invertiert oder nicht invertiert ist, wird mit dem Befehl "HARDCopy STARt" der Transfer der Rohdaten gestartet (siehe Anhang, Teil 1, Funktion „get_screenshot“).

3.3 Umsetzung und aufgetretene Probleme

Nachdem die Installation und Bedienung der notwendigen Software erörtert und Informationen zu Python gegeben wurden, wird in diesem Anschnitt der Entwurf der Software beschrieben. Die Hauptaufgabe besteht darin, Bedienhandlungen (siehe Abs. 2.4) automatisch ablaufen zu lassen, Messdaten zum PC zu transferieren und diese Messdaten optisch anzuzeigen. Es soll ein lauffähiges Python-Programm entstehen, was aber noch keine graphische Oberfläche hat. Dieses Programm wird in der Entwicklungsumgebung Thonny geschrieben und aufgerufen. Nutzereingaben erfolgen in der Shell (siehe Abb.20).

Diese Hauptaufgabe wird in Teilaufgaben unterteilt. Das sind Abläufe, die einer bestimmten Bedienhandlung zugeordnet werden können. Dann erfolgt das Schreiben der Funktion, die im Hauptteil aufgerufen wird. Dadurch bleibt der Code übersichtlich und es ist einfacher, Fehler zu finden. Funktioniert die Funktion wie gewünscht, wird die nächste Teilaufgabe bearbeitet, ansonsten wird die Funktion überarbeitet und erneut getestet.

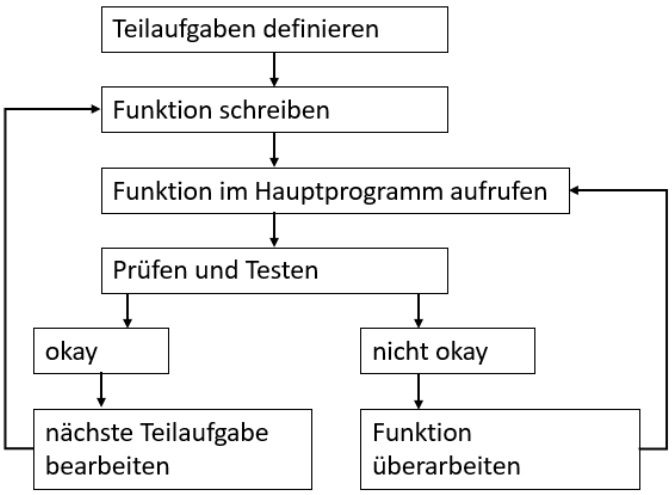


Abbildung 37 Ablauf der Softwareprogrammierung

3.3.1 Definition der Teilaufgaben

Das erste Programm soll die Messdaten für Kanal 3 (Strom primär) und MATH (Kanal 4 – Spannung integriert) importieren. Danach soll ein Diagramm die integrierte Spannung in Abhängigkeit vom Strom anzeigen. Die Erwartungshaltung ist eine Anzeige, die dem XY-Plot in Abb. 14 entspricht. Dazu wurden die Teilaufgaben wie folgt definiert:

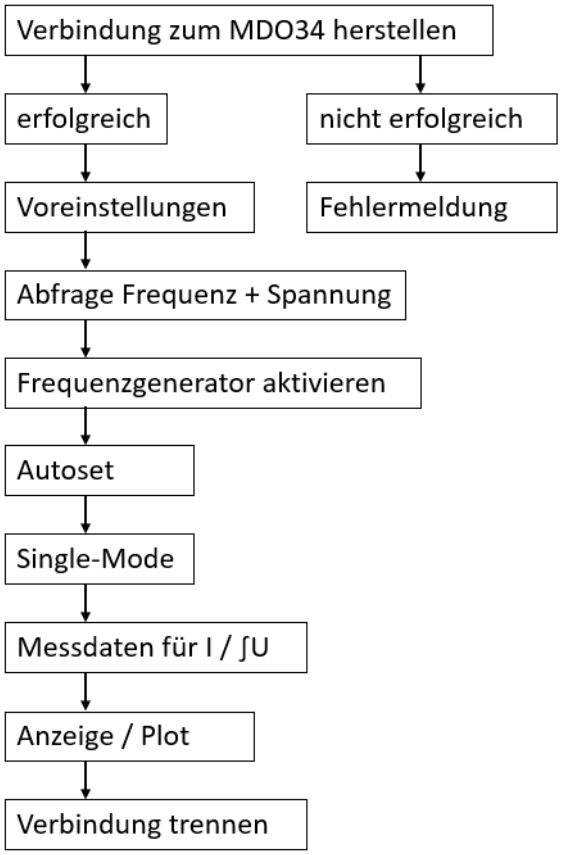


Abbildung 38 Vorläufige Untergliederung in Teilaufgaben

Nach Starten des Programmes wird eine Verbindung zum Oszilloskop hergestellt. Ist diese nicht erfolgreich, zum Beispiel weil die Netzwerkverbindung nicht funktioniert, erhält der Nutzer in der Shell einen Hinweis und das Programm endet. Beim erfolgreichen Verbindungsaufbau werden einige Voreinstellungen vorgenommen, wie zum Beispiel eine auf 10 Sekunden verlängerte Time-Out-Zeit (sonst würde es bei Autoset eine Fehlermeldung geben). Anschließend wird der Nutzer aufgefordert, eine Frequenz und Spannung einzugeben, die beim Aktivieren des Frequenzgenerators an diese Funktion übergeben wird. Danach erfolgt Autoset und die fortlaufende Messung wird gestoppt, damit die Messdaten synchron sind. Nach Import der Messdaten werden diese in einem Diagramm angezeigt und die Verbindung zum Oszilloskop getrennt.

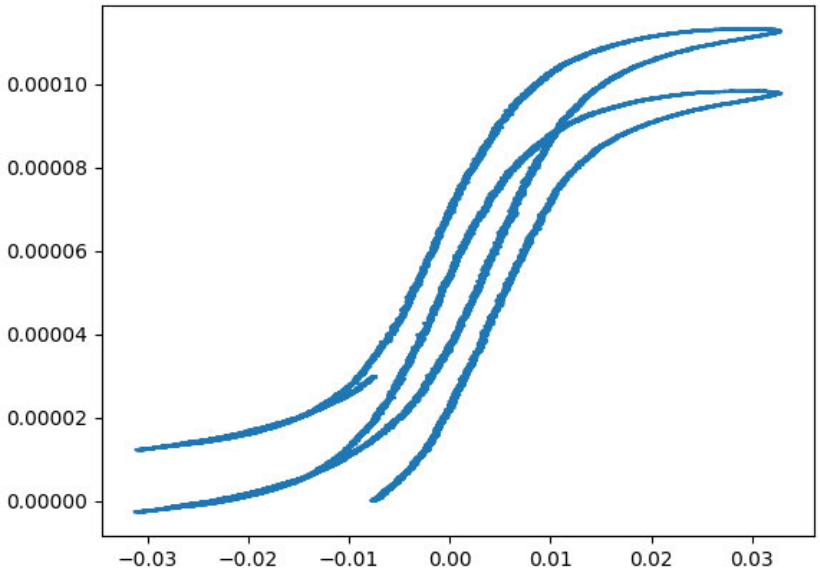


Abbildung 39 Der erste externe Plot

An diesem Plot fällt auf:

- Messdaten verrauscht
- Versatz in Y-Richtung

Der Versatz in Y-Richtung kommt von einen Gleichanteil, der trotz AC-Kopplung vorhanden ist und offensichtlich im Oszilloskop selbst entsteht. Bestätigt wurde das durch eine Cycle-Mean-Messung über das Measurement-Menü des Oszilloskops, bei mehreren Messungen wurde immer ein geringer Gleichanteil angezeigt. Ein Lösungsansatz wäre, den Gleichanteil der Messdaten zu entfernen und dann eine Integration vorzunehmen (23). Dafür wurde zunächst überprüft, ob die interne Integration des Oszilloskops und die über Python realisierte Integration gleiche Ergebnisse liefern (siehe Abs. 3.2.9).

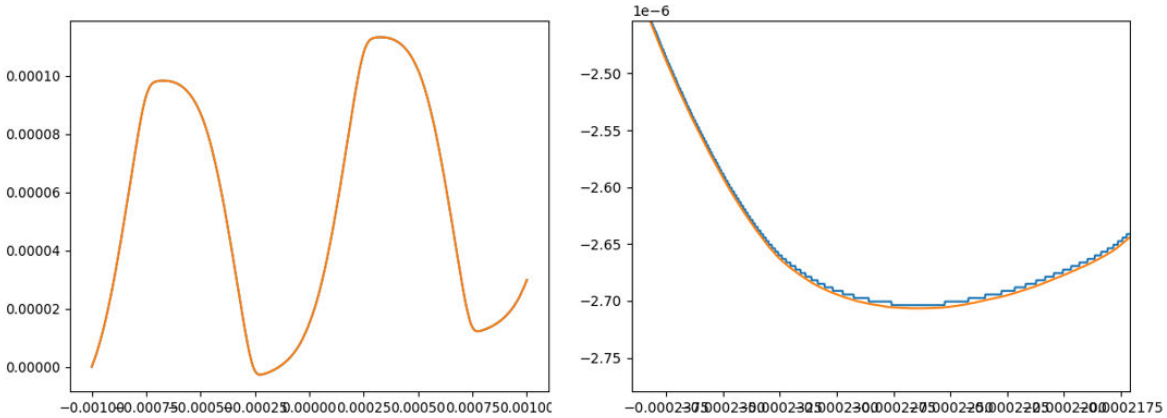


Abbildung 40 Vergleich der Integration von MDO34 und Python

Die blaue Linie zeigt die Integration des Oszilloskops und die orange Linie die Trapezintegration von Python. Beide Linien liegen exakt aufeinander (links) und sind nur beim Hereinzoomen in den Plot zu unterscheiden (rechts). Daher werden in den folgenden Schritten die Messdaten der Kanäle 3 und 4 importiert und die Integration wird in Software durchgeführt.

3.3.2 Entfernung Gleichanteil und Rauschen

Der Gleichanteil führt bei der Integration zu einem Versatz in Y-Richtung. Das Rauschen der Messdaten ist deshalb unangenehm, weil optional vom Unternehmen eine Berechnung der von der Hysteresekurve eingeschlossenen Fläche gewünscht wurde. Diese Fläche repräsentieren die Hystereseverluste. Die Trapezintegration (siehe Abs. 3.2.9) führt zu einer fehlerhaften Berechnung, wenn der zweite Datenpunkt einen kleineren Wert hat als der erste Datenpunkt. Die Fläche wird dann negativ und damit stimmt die gesamte Summierung der Einzelsummen nicht mehr. Das könnte bei verrauschten Messdaten der Fall sein.

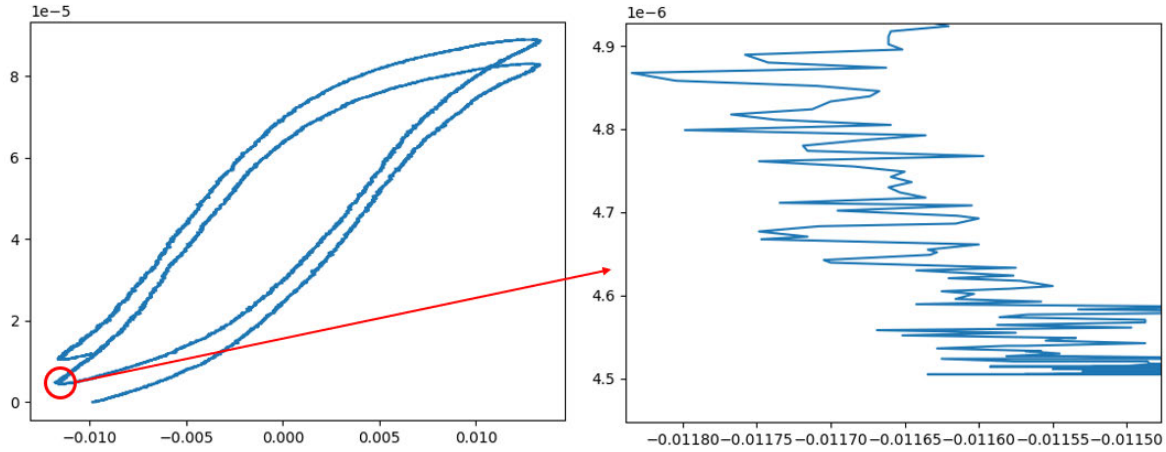


Abbildung 41 Verrauschte Messdaten detailliert dargestellt

Erster Lösungsansatz

Der erste Lösungsansatz zum oben genannten Problem bezog sich ausschließlich auf die Entfernung des Gleichanteils. Der Cycle-Mean-Wert aus dem Measurement-Menü des entsprechenden Kanals wurde ausgelesen und als Offset von den Messdaten abgezogen. Bei einigen Messungen ergab sich statt einer sauberen Hysteresekurve folgende Kennlinie:

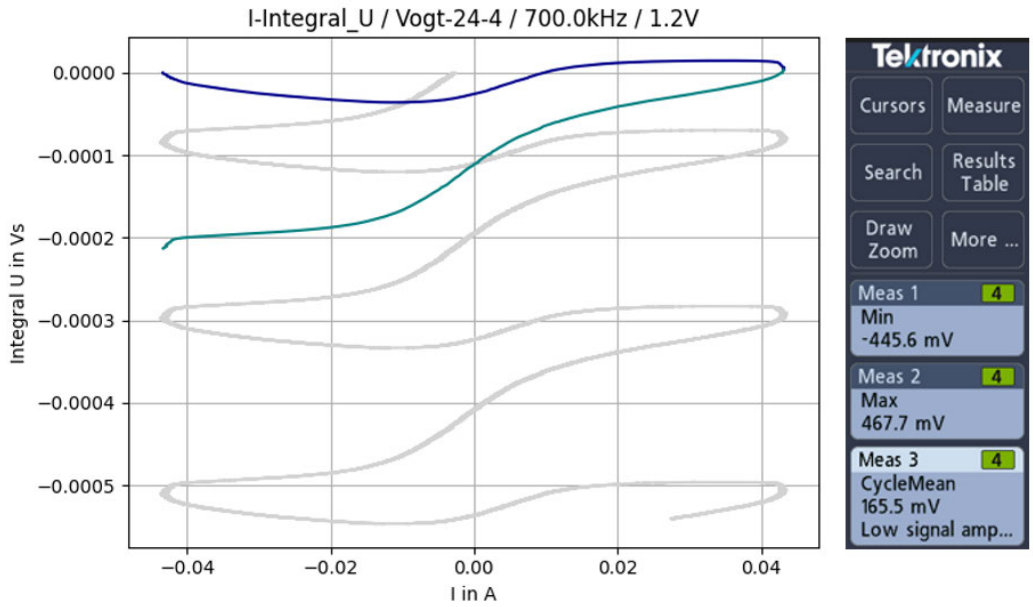


Abbildung 42 Fehlerhafte Hysteresekurve

Als Ursache wurde eine unzuverlässige Cycle-Mean-Messung gefunden. Wie aus Abb. 41 ersichtlich, gibt das Oszilloskop eine Meldung „Low Signal Amplitude“ aus. Diese Meldung tritt weiterhin auf, wenn die vertikale Darstellung des Signals vergrößert wird. Auch bei korrekter Berechnung erscheint manchmal diese Fehlermeldung. Es ist zwar möglich, die Plausibilität der Messung auszulesen (siehe Abs. 3.2.5 „Status- und Eventhandler des Oszilloskops“), aber für den Fehlerfall fehlt eine alternative Strategie der Berechnung des notwendigen Offsets. Deshalb wurde nach einer anderen Möglichkeit der Entfernung des Gleichanteils gesucht.

Zweiter Lösungsansatz

Der Gleichanteil kann durch Überführung der Daten in den Frequenzbereich (FFT) gefunden und eliminiert werden. Außerdem können die höherfrequenten Anteile, also das Rauschen, entfernt werden. Dann erfolgt die Rücktransformation in den Zeitbereich. Zunächst mag dieses Verfahren sehr umfangreich und rechenintensiv erscheinen, die Umsetzung ist in Python aber mit wenigen Zeilen Code erledigt (siehe Abs. 3.2.10 „FFT“).

Bei der Umsetzung des Lösungsansatzes wird genau eine Periode der Messdaten importiert (siehe Anhang, Teil 2, Funktion „get_datapoints_one_period“). Anschließend erfolgt die Entfernung des Gleichanteils und der hochfrequenten Anteile (siehe Anhang, Teil 2, Funktion „apply_fft_filter“). Das Entfernen des Rauschens erfolgt ab einer Grenzfrequenz, die das Vielfache der Grundfrequenz ist. Dieser Faktor kann über eine Variable verändert werden.

Die anschließende Erprobung zeigt folgende Kurvenverläufe mit dem Bauteil Vogt 24-4 (grau sind ungefilterte Messdaten, blau und grün sind gefilterte Messdaten):

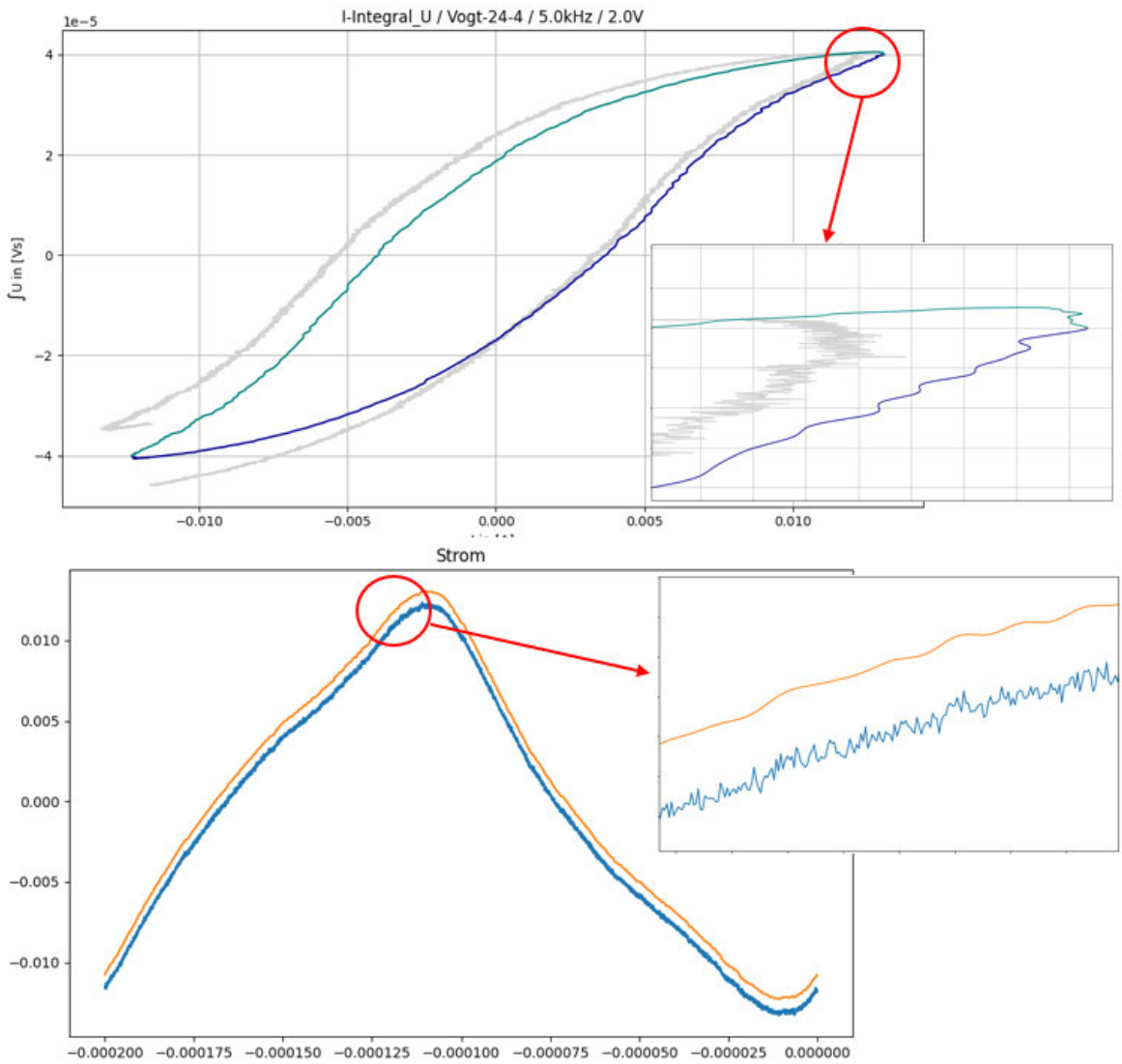


Abbildung 43 Messung bei 5kHz / 2Volt / Grenzfrequenz=1MHz

Die Grenzfrequenz wurde mit einem Faktor der 200-fachen Grundfrequenz gewählt, die gefilterte Hysteresekurve wird unsauber dargestellt. Es können Fehler bei der Berechnung der eingeschlossenen Fläche auftreten, wenn der zweite Datenpunkt auf der X-Achse weiter links als der erste Datenpunkt liegt (siehe obere Detailaufnahme in Abb. 43). Der Versatz in den ungefilterten Daten (grau) entsteht durch die Integration des Gleichanteils.

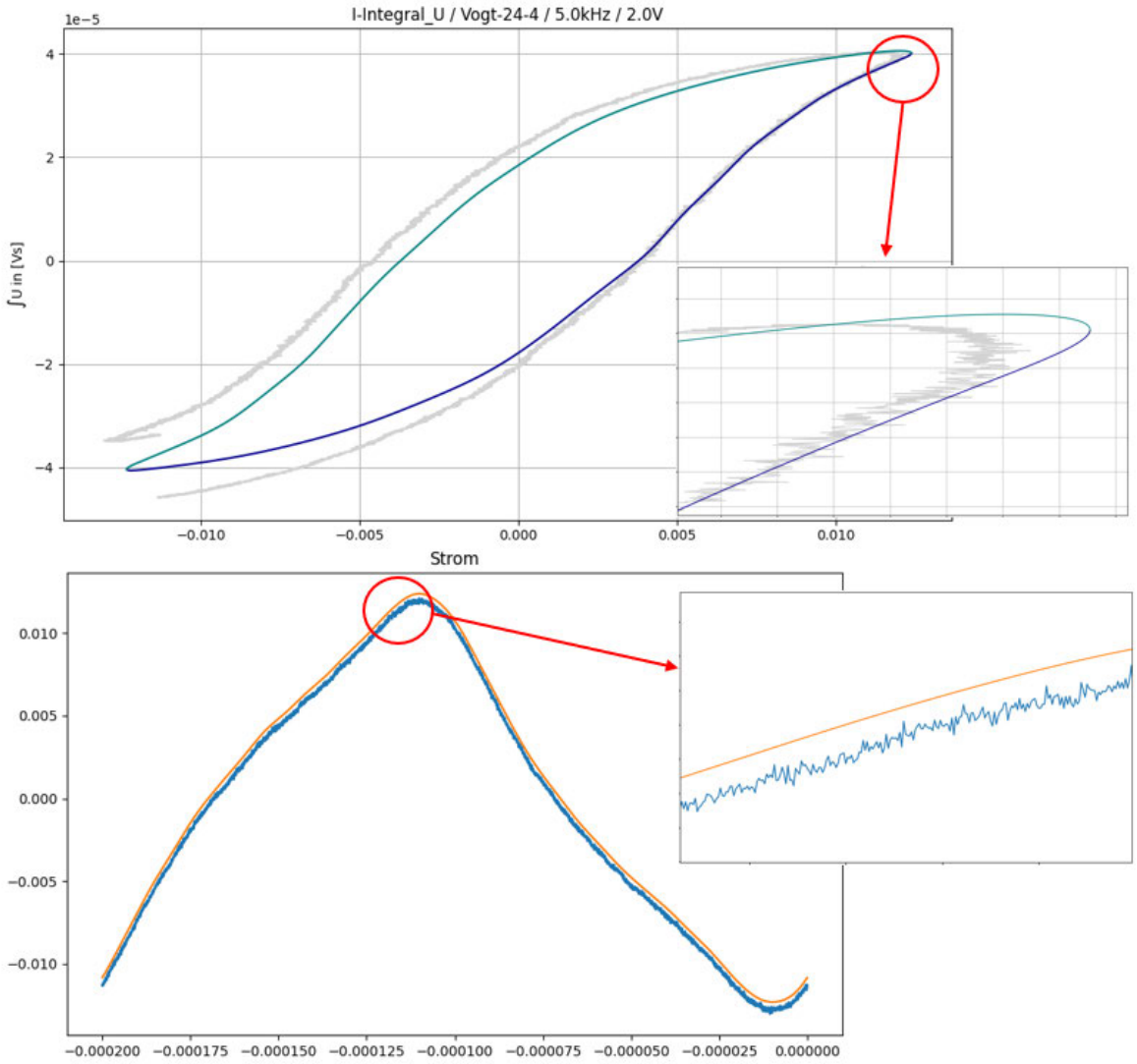


Abbildung 44 Messung bei 5kHz / 2Volt / Grenzfrequenz=100kHz

Im Beispiel der Messung mit Faktor 20 wird eine optimal gefilterte Hysteresekurve angezeigt, auch die Filterung der Stromdaten ist gut. Daher würde die Flächenberechnung ein zuverlässiges Ergebnis liefern.

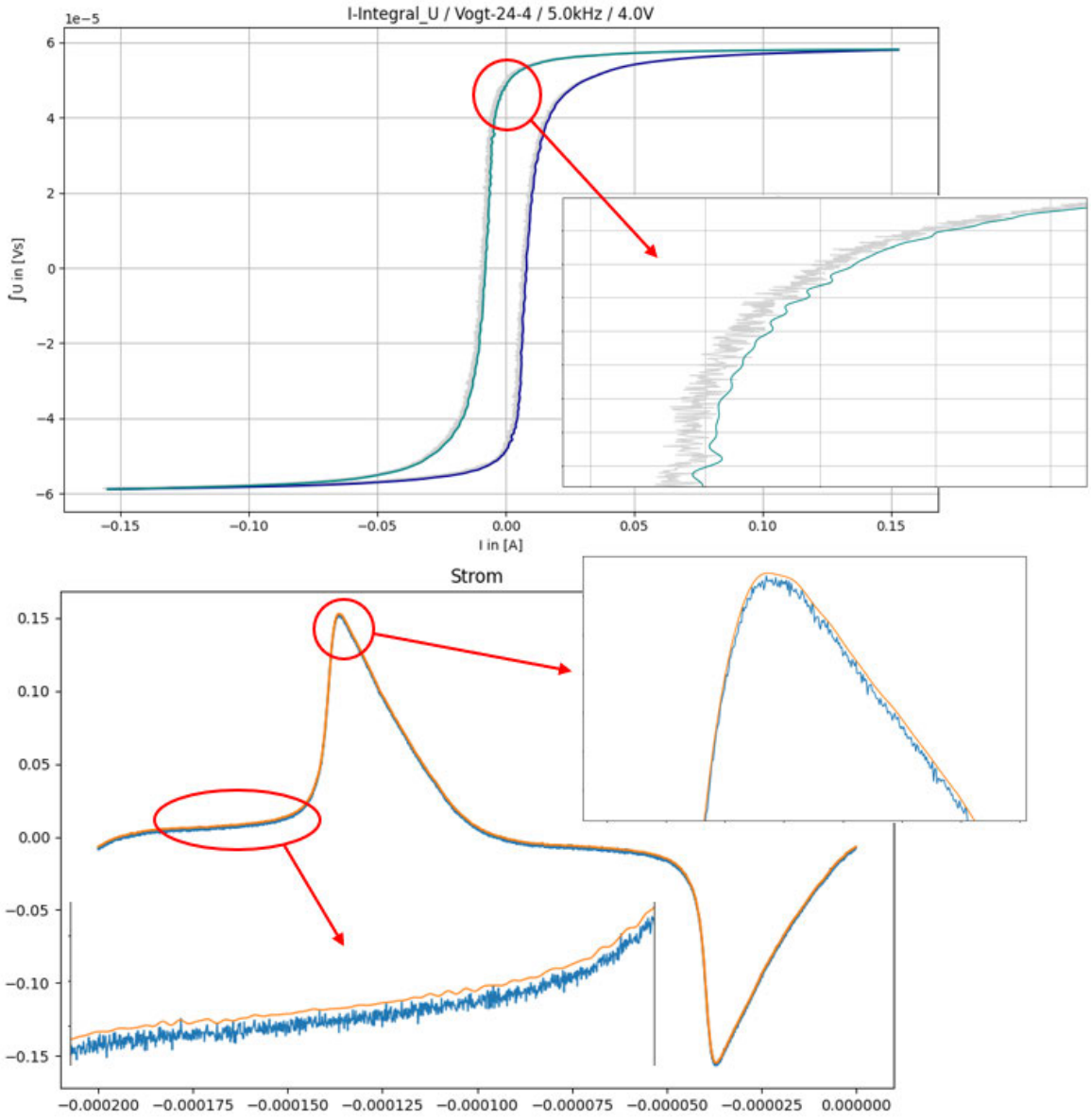


Abbildung 45 Messung bei 5kHz / 4Volt / Grenzfrequenz=1MHz

Bei dieser Messung wurde die Spannung auf 4 Volt erhöht, was einen höheren Stromfluss zur Folge hat. Der Stromfluss steigt stark an und der Übertrager geht in die Sättigung. Mit der hohen Grenzfrequenz ist die Filterung der Stromkurve an der Stelle des Maximums gut, aber im linearen Bereich nicht optimal. Dort passt sich die gefilterte Kurve zu stark den Schwankungen der Messdaten an. Dadurch wird auch die Hysteresekurve nicht optimal gefiltert dargestellt.

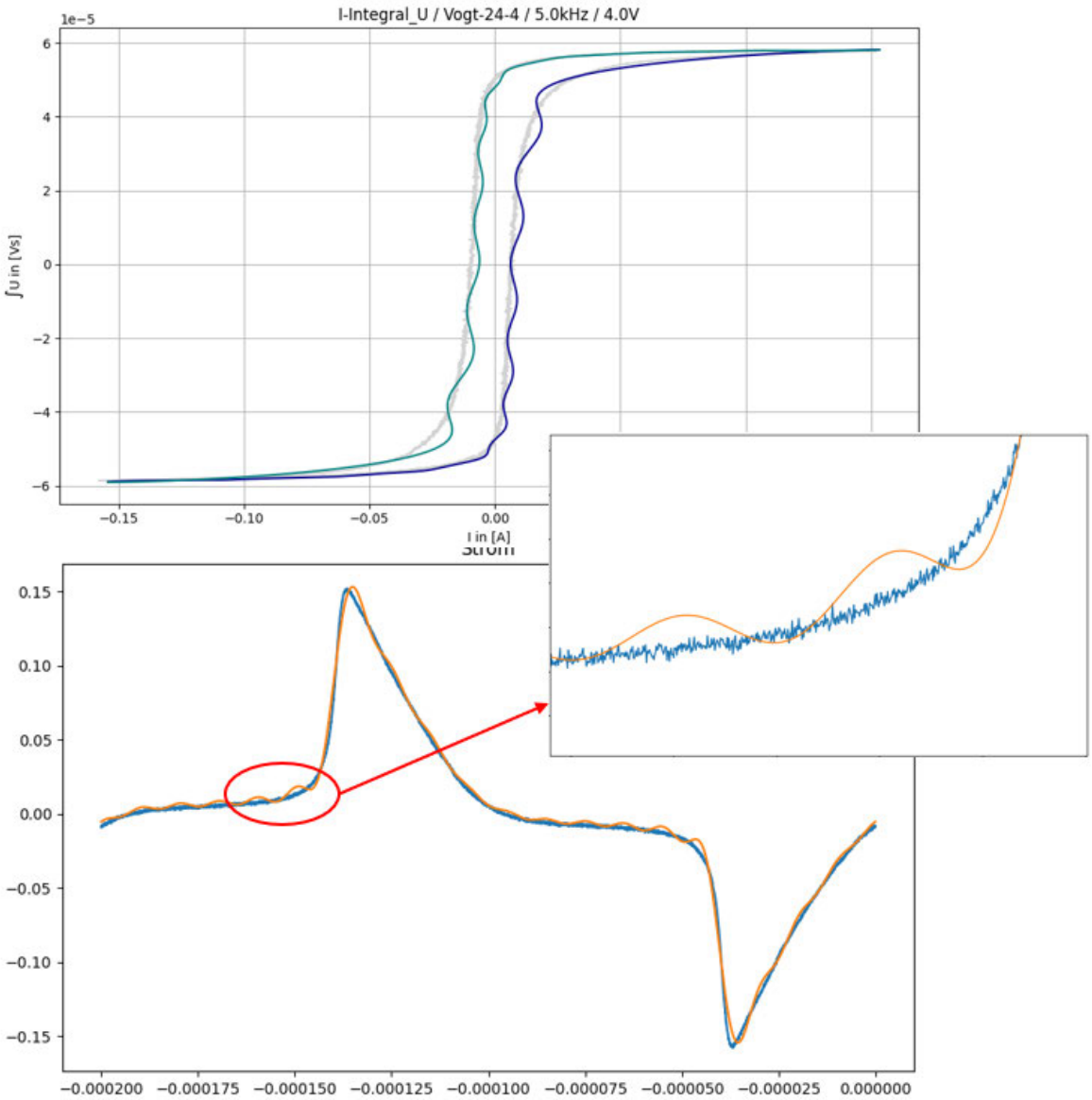


Abbildung 46 Messung bei 5kHz / 4Volt / Grenzfrequenz=100kHz

Die niedrige Grenzfrequenz reicht nicht aus, um den Stromfluss so wiederzugeben, wie er in Wirklichkeit ist. Daher hat die Filterung mit dieser Grenzfrequenz einen negativen Einfluss auf die angezeigte Hysteresekurve.

Zusammenfassung der Versuche

Mit der Filterung der Messdaten durch FFT lässt sich der Gleichanteil, der zum vertikalen Versatz der Hysteresekurve führte, sehr gut entfernen. Die Anfangs- und Endpunkte zeigen keinen Versatz mehr. Für die Entfernung der Rauschanteile ist es aber schwierig, eine optimale Grenzfrequenz für alle Einsatzfälle zu finden. Das bedeutet für die Entwicklung der graphischen Oberfläche im nächsten Kapitel, das dem Bediener die Möglichkeit gegeben werden sollte, zu entscheiden, ob er die Messdaten filtern möchte und welche Grenzfrequenz dabei eingesetzt wird. Die Berechnung des Flächeninhalts der Hysterese-

kurve kann bei bestimmten Messbedingungen unzuverlässig sein und wird deswegen vorerst nicht implementiert werden.

3.3.3 Umrechnung zur BH-Kennlinie

Um aus den gegebenen Stromdaten und integrierten Spannungsdaten eine BH-Kennlinie darstellen zu können, benötigt man einige Parameter des Übertragers:

- Windungszahl Primärseite
- Windungszahl Sekundärseite
- Querschnitt des Kerns
- Länge des Magnetkreises

Die Umrechnung erfolgt über die Beziehung (siehe Abs 2.2):

$$H = I_{prim} \cdot \frac{N_{prim}}{l_{magn}} \tag{8}$$

$$B = \frac{1}{N_{sek} \cdot A} \cdot \int U_{sek}\, dt \tag{9}$$

3.3.4 Endgültige Umsetzung des Python-Programms und Test

Nachdem die einzelnen Verbesserungen in das Programm eingeflossen sind, ergibt sich folgende Untergliederung:

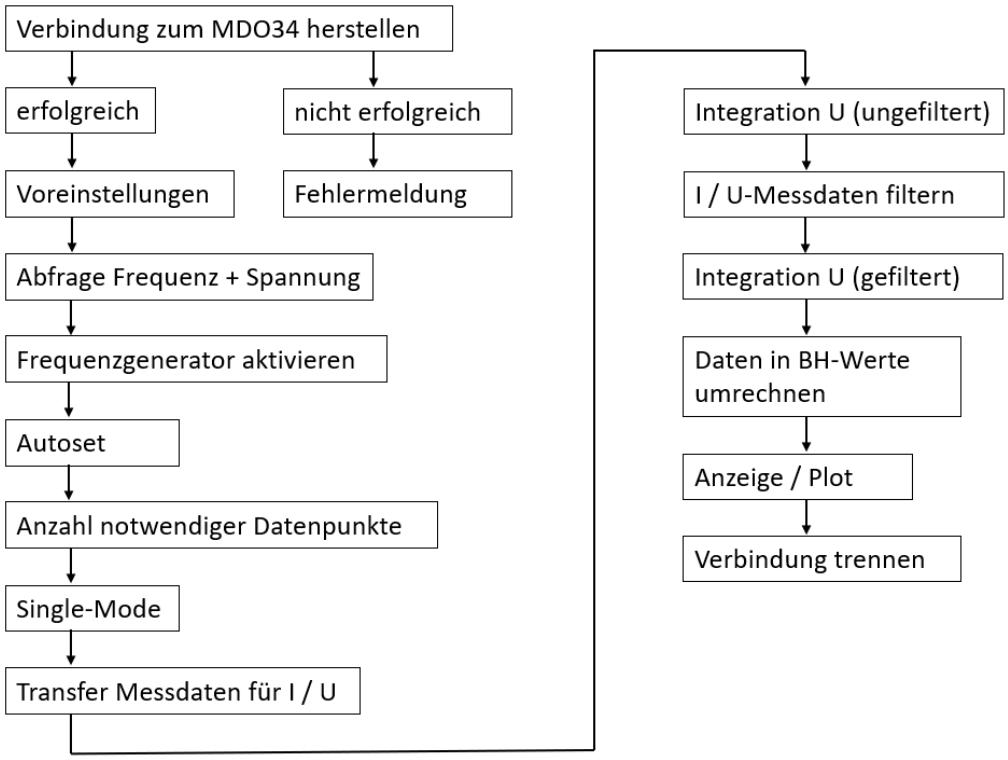


Abbildung 47 Endgültige Untergliederung der Teilaufgaben / Funktionen

Das fertige Programm ist im Anhang Teil 2 zu finden. Es lässt sich über die Entwicklungsumgebung Thonny starten. Bei der Entwicklung wurde darauf geachtet, Optionen als Variablen in Vorbereitung auf eine graphische Oberfläche zu hinterlegen. Diese Optionen können den Zustand 0 (False) oder 1 (True) haben und werden im Programm über eine if-Bedingung abgefragt. Es können aber auch andere Parameter wie ein Bauteilname oder geometrische Angaben sein. In der graphischen Oberfläche wird das als Checkbox, Eingabefeld oder andere Varianten implementiert werden.

```
27 # Variablen in Vorbereitung auf graphische Oberfläche
28 USE_CUTOFF_FREQ = 0 # gefilterte Daten -> Grenzfrequenz anwenden
29 SHOW_FILTERED_DATA = 1 # Anzeige der gefilterten Daten
30 SHOW_RAW_DATA = 1 # Anzeige der Rohdaten
31 AUTOZERO_CURRENT_PROBE = 1 # AutoZero ausführen, 1 = JA, 0 = NEIN
32 COMPONENT = 'Vogt-24-4' # Bezeichnung für Bauteil
33 SAVE_DATA = 1 # Daten speichern
34 CUTOFF_FREQUENCY = 20 # Grenzfrequenz als Faktor der Grundfrequenz (für FFT, 20...500 empfohlen)
35 SHOW_BH = 1 # 1 = BH-Kurve anzeigen, sonst I/intU
36 # BH Parameter
37 LM = 13.96E-3 # Länge magnetisch in m
38 LG = 0 # Länge Luftspalt in m
39 N_PRIM = 14 # Primärwindungen
40 N_SEC = 19 # Sekundärwindungen
41 A = 5.495E-6 # Fläche in mm**2
```

Abbildung 48 Optionen in Vorbereitung auf die graphische Oberfläche

Der Test mit dem Bauteil Vogt 24-4 erfolgte bei verschiedenen Frequenzen, Spannungen und Optionen. Die Prüfung verlief bei allen getesteten Kombinationen zur Zufriedenheit und somit kann zur Entwicklung der graphischen Oberfläche übergegangen werden.

4 Entwurf einer graphischen Oberfläche

Die Anzeige der Hysteresekurve und die Interaktion mit dem Nutzer soll visuell ansprechender gestaltet werden. Dazu wird eine graphische Oberfläche entworfen. Nach einer Übersicht über das entsprechende Designertool wird die Umsetzung erläutert.

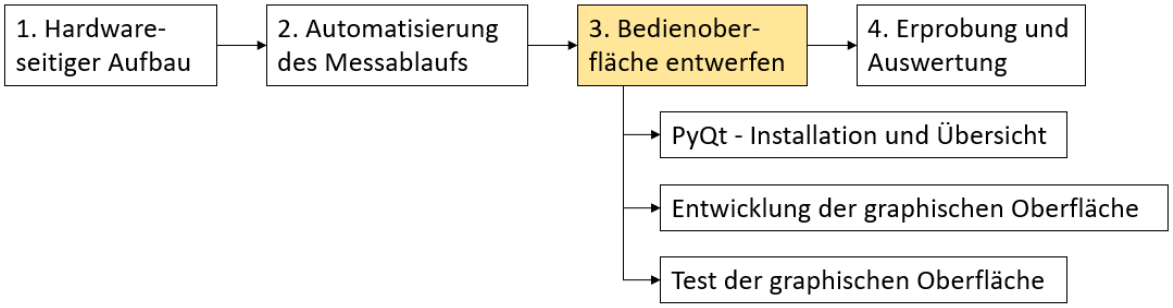


Abbildung 49 Dritter Entwicklungsabschnitt

4.1 PyQt - Installation und Übersicht

Eine graphische Nutzeroberfläche besteht aus einer Reihe von Steuerelementen, die in einer bestimmten Anordnung innerhalb eines Anzeigefensters platziert werden. Die Erstellung dieser GUI (graphical user interface) kann für einfache Anwendung durch Programmcode erfolgen. Für umfangreichere Oberflächen nutzt man Designtools, in denen das gewünschte Aussehen und Verhalten erstellt wird und das anschließend der Programmcode durch das Designtool erstellt wird. In Verbindung mit Python kann dafür PyQt 5 verwendet werden (24). Nachfolgend wird die Installation beschrieben und eine Übersicht über das Programm gegeben.

4.1.1 Installation PyQt5

PyQt5 kann als Paket über Thonny installiert werden (siehe Abs 3.1.3). Dazu öffnet man über den Reiter „Tools“ den Paketmanager und gibt im Suchfenster „PyQt“ ein. Danach wählt man PyQt5 aus und installiert dieses Paket.

4.1.2 Übersicht PyQt-Designer

Das Programmer „Designer.exe“ lässt sich am einfachsten über die Windows-Suche in der Taskleiste finden. Es gibt keinen Eintrag im Startmenü, wenn die Installation wie in Abs. 4.1.1 beschrieben, erfolgte. Nach dem Öffnen des Qt Designers erscheint ein Dialogfeld für ein neues Formular. Dort wählt man „Main Window“ aus.

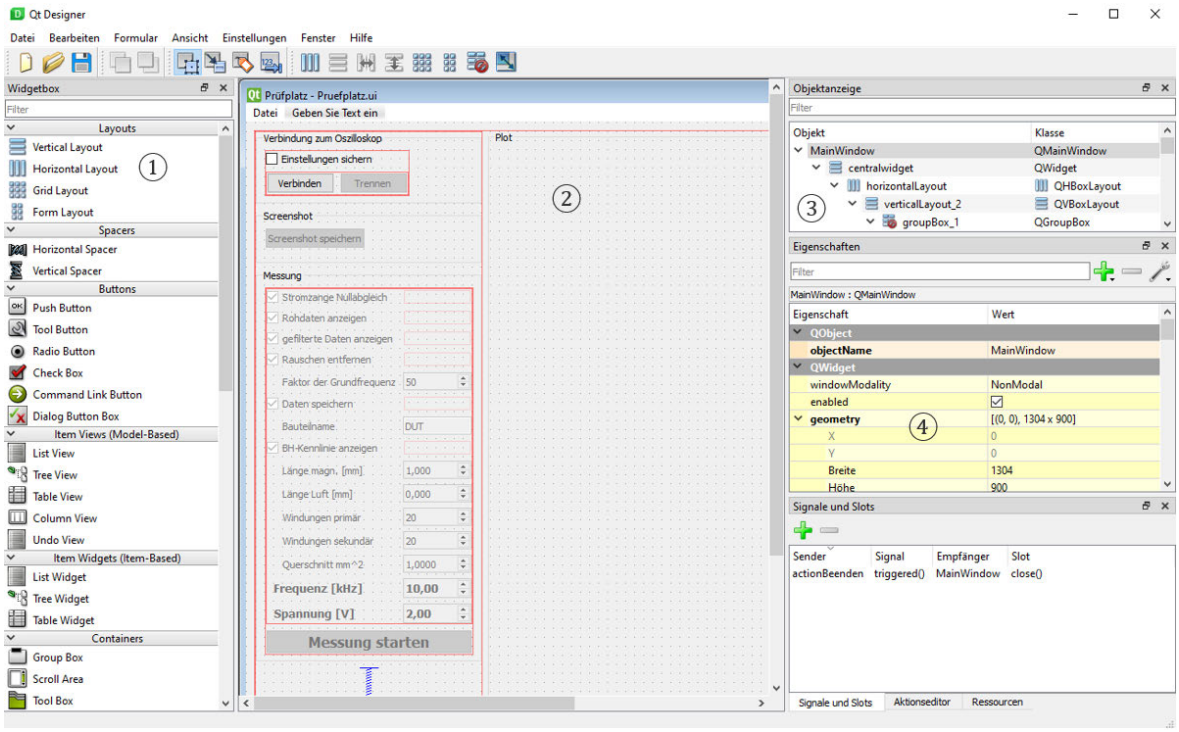


Abbildung 50 Qt-Designer, Übersicht

In der Widgetbox ① sind verschiedene Steuerelemente und Layouts verfügbar. Häufig verwendete Steuerelemente sind die Check Box, das Label und die (Double-)Spin Box. Layouts dienen zur Strukturierung der Oberfläche ②. Die Objektanzeige ③ zeigt über- und untergeordnete Layout-Elemente an. Im Eigenschaften-Fenster ④ können die Parameter der Widgets verändert werden. Das erstellte Bild kann mit der Tastenkombination Strg + R angeschaut und getestet werden.

Layouts, Spacer und Container

Diese dienen zur Strukturierung der Oberfläche in einzelne Segmente. Layouts bestimmen, wie Buttons und andere Widget zueinander angeordnet sind (horizontal, vertikal oder in tabellarischer Anordnung). Ein Widget ist jedes in der Widgetbox vorhandene Element. In Container kann man logisch zueinander gehörende Widgets platzieren.

Wird die Größe des Hauptfensters geändert, kann das eine Verschiebung der Widgets zur Folge haben. Dieses Verhalten kann man beeinflussen, indem man in der sizePolicy im Eigenschaften-Fenster geeignete Einstellungen auswählt. Durch Verwendung eines Spacer werden Widgets räumlich zusammengehalten, anstatt den gesamten verfügbaren Platz innerhalb der vorgesehenen Fläche einzunehmen.

Buttons, Input Widgets und Display Widgets

In dieser Kategorie sind die Bedien- und Anzeigeelemente zu finden. Diese werden wie die anderen Widgets mit Drag & Drop in das Hauptfenster oder in die Objektanzeige gezogen.

Eigenschaften von Widgets

Im Eigenschaften-Fenster können die Widgets angepasst werden. „Enabled“ legt fest, ob ein Widget aktiv oder ausgegraut ist. Ist ein Push Button inaktiv, ist keine Bedienhandlung möglich. Die „sizePolicy“ legt das Verhalten bei Größenänderung fest, „Expanding“ legt fest, dass ein Widget den maximal verfügbaren Platz einnimmt. Die anderen Widgets müssen dann auf „Minimum“ oder „Fixed“ eingestellt werden. So erreicht man bei Größenänderung des Hauptfensters, das sich nur ein Bereich ändert, Bedienelemente aber erhalten bleiben. Für (Double-)Spin-Boxen können der minimale und maximale Wert, die Schrittweite und der Wert bei erstmaligem Aufruf des Fensters festgelegt werden. Eine Spin-Box dient zur Einstellung ganzzahliger Werte, die Double-Spin-Box ist für Kommazahlen vorgesehen.

4.1.3 Verbindung der graphischen Oberfläche mit Python

Der erstellte Dialog wird gespeichert (Dateiendung .ui) und soll über ein Python Programm genutzt werden. Dafür gibt es zwei Möglichkeiten:

- mit dem Programm pyuic5 wird die ui-Datei in eine Python-Datei überführt
- die ui-Datei wird von Python eingebunden

Weitere Informationen zur ersten Möglichkeit sind unter (13 S. 842 f.) verfügbar. Die zweite Variante gestattet eine nachträgliche Änderung der graphischen Oberfläche. Zum Einbinden in den Python-Code ist der Import einiger Bibliotheken und das Erstellen einer Klasse notwendig (25 S. 356 ff.):

```
1  # Importiere die notwendigen Elemente für die GUI
2  import PyQt5
3  import PyQt5.uic
4  import PyQt5.QtWidgets
5
6  # Definiere eine Klasse, die von QMainWindow abgeleitet wird
7  class MainWindow(PyQt5.QtWidgets.QMainWindow):
8
9      def __init__(self):
10         super().__init__() # QMainWindow initialisieren
11         PyQt5.uic.loadUi('Pruefplatz.ui', self) # User Interface laden (aus .ui-Datei)
12
13 # Erzeuge eine QApplication und das Hauptfenster.
14 app = PyQt5.QtWidgets.QApplication([])
15 window = MainWindow()
16
17 # Zeige das Fenster und starte die QApplication.
18 window.show()
19 app.exec_()
```

Abbildung 51 Minimalprogramm zum Aufruf einer ui-Datei

Hinweis: Wurde im Qt-Designer statt des Main-Window ein Dialog ausgewählt, muss das Programm zum Aufruf des Dialogs etwas anders geschrieben werden. Ein Beispiel dazu ist im Python-3-Handbuch (13 S. 843) zu finden.

4.1.4 Signal-Slot-Prinzip

Wird ein Programm mit graphischer Benutzeroberfläche geschrieben, wird das Prinzip der ereignisgesteuerten Programmierung verwendet. Dieses Programm läuft nicht sequenziell von oben nach unten ab, wie das in dem Python Programm aus Kapitel 3 der Fall ist. Der Ablauf wird vom Benutzer durch Bedienhandlungen gesteuert. Beim Auftreten bestimmter Ereignisse werden die dafür vorgesehenen Codeabschnitte ausgeführt (13 S. 844 ff.).

Ein Widget sendet ein Signal, wenn ein bestimmtes Ereignis eingetreten ist. Das kann ein Klick auf einen Button sein oder die Änderung eines Wertes einer Spin Box. Welche Signale gesendet werden, kann man in der PyQt-Dokumentation (24) recherchieren, in Fachbüchern nachschlagen (13 S. 847 ff.) oder über den Qt-Designer herausfinden. Dazu drückt man den Button „Signale und Slots bearbeiten“ und zieht dann vom entsprechenden Widget mit gedrückter Maustaste weg. Es erscheint folgendes Fenster:

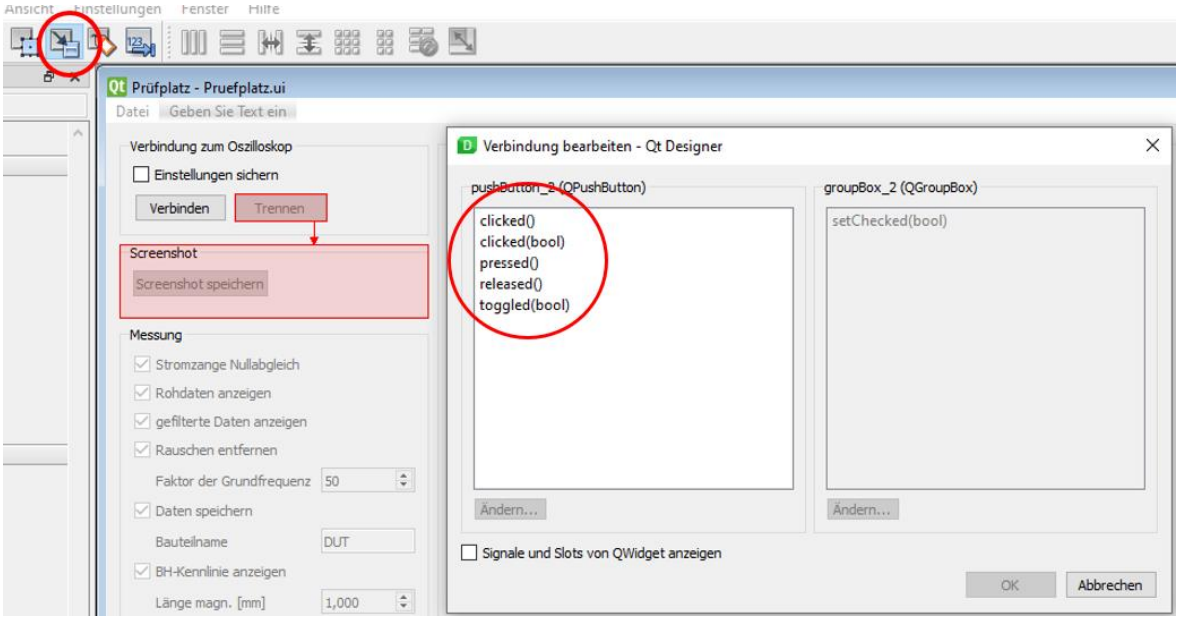


Abbildung 52 Signale im Qt-Designer

Signale werden im Beispiel (Abb. 52) des Push Buttons unter anderem gesendet, wenn dieser gedrückt, losgelassen oder angeklickt wurde (drücken und loslassen).

Für die Verarbeitung der Signale werden im Programmcode Slots eingerichtet. Das ist eine Funktion, die immer dann aufgerufen wird, wenn ein bestimmtes Signal gesendet wurde. Es können verschiedene Signale mit einem Slot verbunden werden.

```
44          # Slots einrichten
45          self.pushButton_1.clicked.connect(self.Connect)

53      # Funktionen
54      def Connect(self):
55          # versuche Verbindungsaufbau
56          try:
57              # rm gibt den Speicherort des Visa-Treibers zurück
58              rm = pyvisa.ResourceManager()
59              # eine Verbindung aufbauen und zurückgeben
60              global conn
61              conn = rm.open_resource(ADDRESS_MD034)
```

Abbildung 53 Signale und Slots

In der Abb. 53 wird ein Klick-Ereignis (ein Signal) mit einem Slot verbunden. Dieser Slot ist die Funktion „Connect“. Immer wenn der Push Button 1 geklickt wird, wird der Code dieser Funktion ausgeführt.

4.1.5 Plot anzeigen

Es ist im Qt-Designer nicht direkt möglich, einen Plot in der graphischen Oberfläche anzeigen zu lassen. In (25 S. 356 ff.) wird das Vorgehen für externe Hinzufügen eines Plots erläutert. Wichtig ist, dass nicht das Modul „matplotlib.pyplot“ verwendet wird.

```
1  # Importiere Matplotlib-Bibliotheken
2  import matplotlib as mpl
3  import matplotlib.backends.backend_qt5agg
4  import matplotlib.figure
5
6  # Definiere eine Klasse, die von QMainWindow abgeleitet wird
7  class MainWindow(PyQt5.QtWidgets.QMainWindow):
8
9      def __init__(self):
10         super().__init__() # QMainWindow initialisieren
11         PyQt5.uic.loadUi('Pruefplatz.ui', self) # User Interface laden (aus .ui-Datei)
12         self.fig = mpl.figure.Figure() # ein figure-Objekt erzeugen
13         mpl.backends.backend_qt5agg.FigureCanvasQTAgg(self.fig) # Qt-Zeichenfläche erzeugen
14         self.fig.set_tight_layout(True) # tight layout passt Plot in vorhandene Fläche ein
15         self.groupBox_5.layout().addWidget(self.fig.canvas) # Zeichenfläche (canvas) in GUI einfügen
16 #-----------------------------------------------------------------------------------------------
17     def showPlot(self):
18         # Plot vorbereiten
19         self.ax = self.fig.add_subplot(1, 1, 1)
20         self.ax.set_xlabel('I [A]')
21         self.ax.set_ylabel('$\int$U [m]')
22         #self.ax.set_aspect('equal')
23         self.ax.grid()
24         # einen Plot erzeugen (ohne Messwerte)
25         self.plot, = self.ax.plot([2,4], [1,3], color='gray')
26         self.fig.canvas.draw()
```

Abbildung 54 Einen Plot in der graphischen Oberfläche anzeigen

In den Zeilen 2-4 werden die notwendigen Bibliotheken für den Plot importiert. Die Zeilen 9-11 sind aus Abb. 50 bekannt. Die Erzeugung und das Einfügen in die graphische Oberfläche wird in den Zeilen 12-15 umgesetzt. Dabei muss in Zeile 15 „groupBox_5“ mit dem Bezeichner der Group-Box ersetzt werden, die für die Anzeige zuständig ist. Die Funktion „showPlot“ (Zeile 17-26) richtet den Plot ein und zeigt diesen an.

Hinweis: Im Qt-Designer muss die Group-Box, in der das Diagramm geplottet wird, wie folgt bearbeitet werden (25 S. 354):

- einen Push Button in die Group Box ziehen
- mit rechter Maustaste im leeren Bereich der Group Box klicken und unter Layout den Eintrag „Objekte senkrecht anordnen“ auswählen
- den Push Button wieder löschen

4.2 Entwicklung der graphischen Oberfläche

Nachdem die Installation, Bedienung und Besonderheiten von PyQt beschrieben wurden, wird in diesem Abschnitt die Entwicklung der graphischen Oberfläche und die Einbindung in den Python-Code beschrieben.

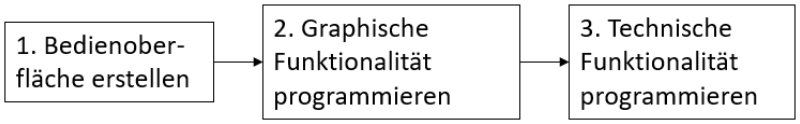


Abbildung 55 Entwicklungsablauf graphische Oberfläche

4.2.1 Bedienoberfläche erstellen

An die graphische Oberfläche werden folgende Anforderungen gestellt:

- Anzeige des XY-Diagramms im rechten Teil, Fenstergröße skalierbar
- Bedienbereich links, Funktionalität auch bei Verkleinern des Fensters gegeben
- Unterteilung des Bedienbereiches
    - Verbindung zum Oszilloskop herstellen und trennen
    - einen Screenshot aufnehmen
    - Parameter eingeben und Messung starten
- die Parameter sind:
    - Stromzange Nullabgleich ja/nein
    - Rohdaten anzeigen ja/nein
    - gefilterte Daten anzeigen ja/nein
    - Rauschen entfernen ja/nein, Eingabe des Faktors für die Grenzfrequenz
    - Daten speichern ja/nein, Eingabe Bauteilname
    - BH-Kennlinie anzeigen ja/nein
        - wenn nein, dann ∫U / I anzeigen
        - wenn ja, dann Eingabe technischer Parameter zur Berechnung
    - Frequenz und Spannung für Frequenzgenerator
- es werden nur Bedienelemente freigegeben, die notwendig/sinnvoll für die Anwendung sind

In der Umsetzung wird geplant, wo und wie die Bedienelemente angeordnet werden. Zunächst wird ein Layoutplan erstellt, angewendet, und dann die Bedienelemente hinzugefügt.

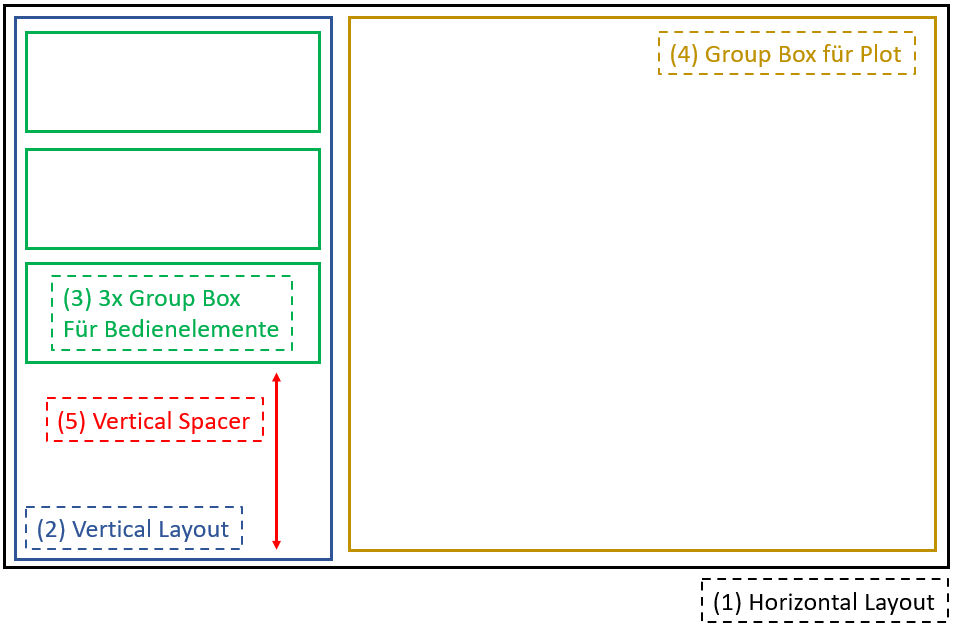


Abbildung 56 Anordnung der Layouts

Die Anordnung der Layouts erfolgt in der Reihenfolge der Abb. 56. Die Beziehung der Layouts untereinander ist in der Objektanzeige sichtbar:

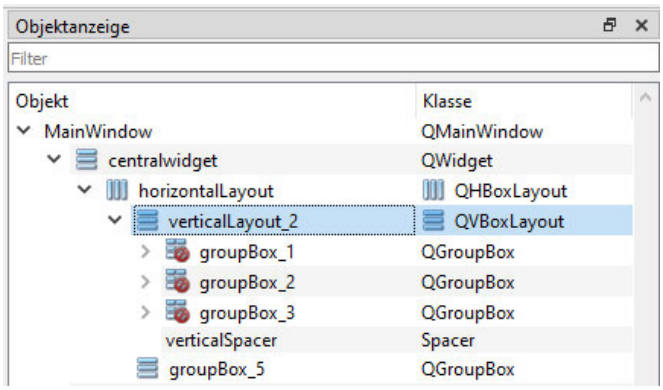


Abbildung 57 Layout-Übersicht in der Objektanzeige

Den Groupboxen wird eine Mindestgröße zugewiesen und die Groupbox für den Plot wird auf „Expanding“ gesetzt. Das stellt die Funktionalität der Bedienelemente bei Größenänderung des Fensters sicher. Anschließend werden die notwendigen Bedienelemente hinzugefügt. Dazu müssen die in Abb. 48 sichtbaren Variablen in Widgets umgesetzt werden. Es werden noch einige Push Buttons für Start der Messung, Verbinden, Trennen und den Screenshot implementiert. Abb. 57 zeigt das fertige Fenster und beschreibt, welche Widgets verwendet wurden. Alle Widgets in der Group Box „Messung“ wurden über das Grid Layout in tabellarischer Form platziert.

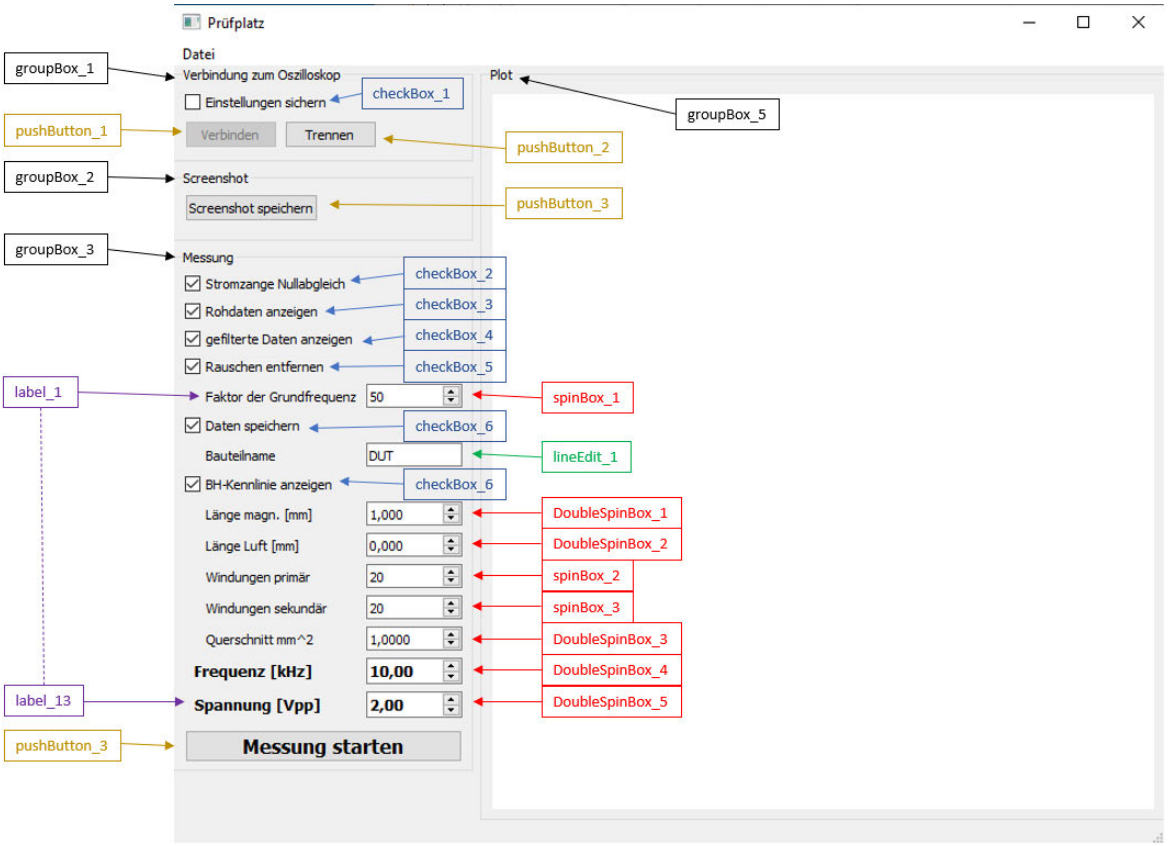


Abbildung 58 Zuordnung Widgets in der graphischen Oberfläche

Über die Eigenschaft „enabled“ kann festgelegt werden, ob ein Widget aktiviert oder ausgegraut ist. Da zunächst eine Verbindung mit dem Oszilloskop hergestellt werden muss, werden alle Widgets außer „Einstellungen Sichern“ und „Verbinden“ deaktiviert. Nachdem die Verbindung hergestellt wurde, werden alle anderen Widgets aktiviert – diese Funktionalität wird von der Software bereitgestellt.

4.2.2 Programmierung der graphischen Funktionalität

Widgets sollen dann aktiviert werden, wenn eine Nutzung sinnvoll ist. Zum Beispiel ist es nicht notwendig, einen Bauteilnamen einzugeben, wenn keine Messwerte gespeichert werden (der Dateiname beginnt mit dem Bauteilname). Ist „Daten speichern“ nicht aktiviert, dann soll auch „Bauteilname“ und das dazugehörige Line-Edit-Feld deaktiviert werden. Das wird über das Signal-Slot-Prinzip (siehe Abs. 4.1.4) für die Check-Boxen „Rauschen entfernen“, „Daten speichern“ und „BH-Kennlinie“ gelöst.

Die Aktivierung aller Widgets, nachdem die „Verbinden“-Taste gedrückt wurde, ist von einer erfolgreich hergestellten Verbindung abhängig. Daher wird dieser Teil nach Implementierung der technischen Funktionalität bearbeitet.

4.2.3 Programmierung der technischen Funktionalität

Ziel ist es, die Funktionalität des in Kapitel 3 erstellten Python-Skripts auf die graphische Oberfläche zu übertragen. Die Programmierung gliedert sich in drei Abschnitte:

- Festlegung, welche Funktionen einem Push Button zugeordnet wird
- Ersatz von Variablen, die den Wert „True“ oder „False“ hatten, mit Abfrage über den Zustand einer Check Box
- Ersatz von Variablen, die einen alphanumerischen oder numerischen Wert haben, mit den Werten aus einer (Double-)Spin-Box oder dem Line-Edit-Feld

Zunächst erfolgt die Planung, welche Funktionen den 4 Push-Buttons zugeordnet wird:

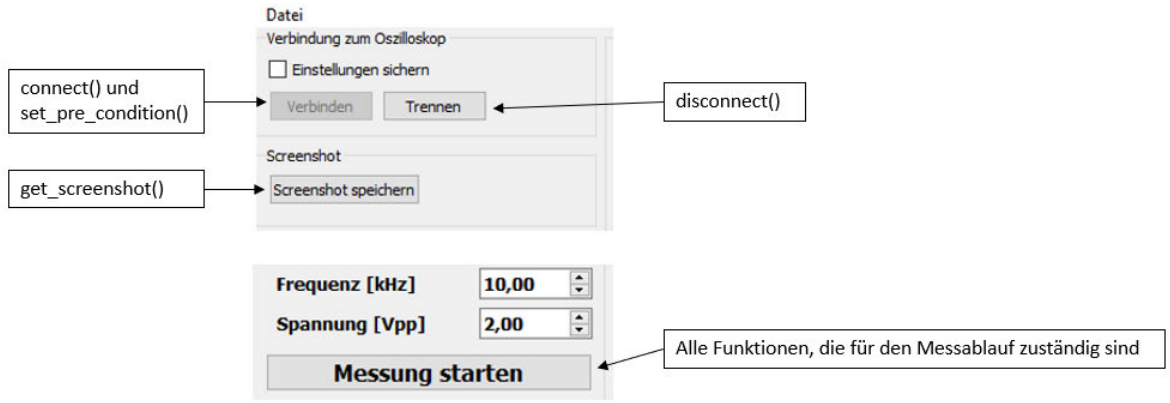


Abbildung 59 Zuordnung der Funktionen

Nach dem Signal-Slot-Prinzip (siehe Abs. 4.1.4) werden bei Betätigung der Buttons „Verbinden“, „Trennen“ und „Screenshot speichern“ die in Abb.57 gezeigten Funktionen aufgerufen. Für den Messablauf („Messung starten“) wird jedoch die Funktion „startMeasurement“ aufgerufen, die dann alle Funktionen für den Messablauf enthält sowie deren Aufruf. Dadurch ist im Quelltext eine verbesserte Übersichtlichkeit gegeben.

Der Python-Code aus Kapitel 3 wurde durch den Einsatz von Variablen schon auf die graphische Oberfläche vorbereitet. Die Abfrage der Variablen wird durch die Abfrage des Zustandes der Check Box ersetzt:

```
27  # Variablen in Vorbereitung auf graphische Oberfläche
28  USE_CUTOFF_FREQ = 1 # gefilterte Daten -> Grenzfrequenz anwenden
```

```
230     if(USE_CUTOFF_FREQ):
231         F_g = wf_fft * (np.abs(fk) < cutoff_frequency)
```

Python-Code aus Kapitel 3

```
350             if(self.checkBox_5.isChecked()==True):
351                 F_g = wf_fft * (np.abs(fk) < cutoff_frequency)
```

Code für graphische Oberfläche

Abbildung 60 Check Box ersetzt Variable

Im dritten und letzten Abschnitt werden in Variablen hinterlegte Parameter durch die Abfrage der Spin-Boxen ersetzt. Das erfolgt analog dem Vorgehen aus Abb. 58.

4.3 Test der graphischen Oberfläche

Durch die schrittweise und strukturierte Erstellung der graphischen Oberfläche und des Programmes konnten Fehler im Ansatz erkannt und behoben werden. Nach Fertigstellung wurde das Programm umfangreich mit und ohne Aktivierung der einzelnen Check-Boxen erfolgreich getestet. Ist die Verbindung zum Oszilloskop fehlgeschlagen, erscheint in der Statusbar (unten links im Fenster) ein Hinweis. Mit der Check Box „Einstellungen sichern“ können sämtliche Nutzereinstellungen auf dem internen Speicher 10 des Oszilloskops gesichert werden (muss vor dem Druck auf „Verbinden“ aktiviert sein). Ist die Check Box bei Druck auf „Trennen“ aktiviert, werden diese Einstellungen wieder hergestellt. Viele Widgets wurden mit einem Tooltip versehen.

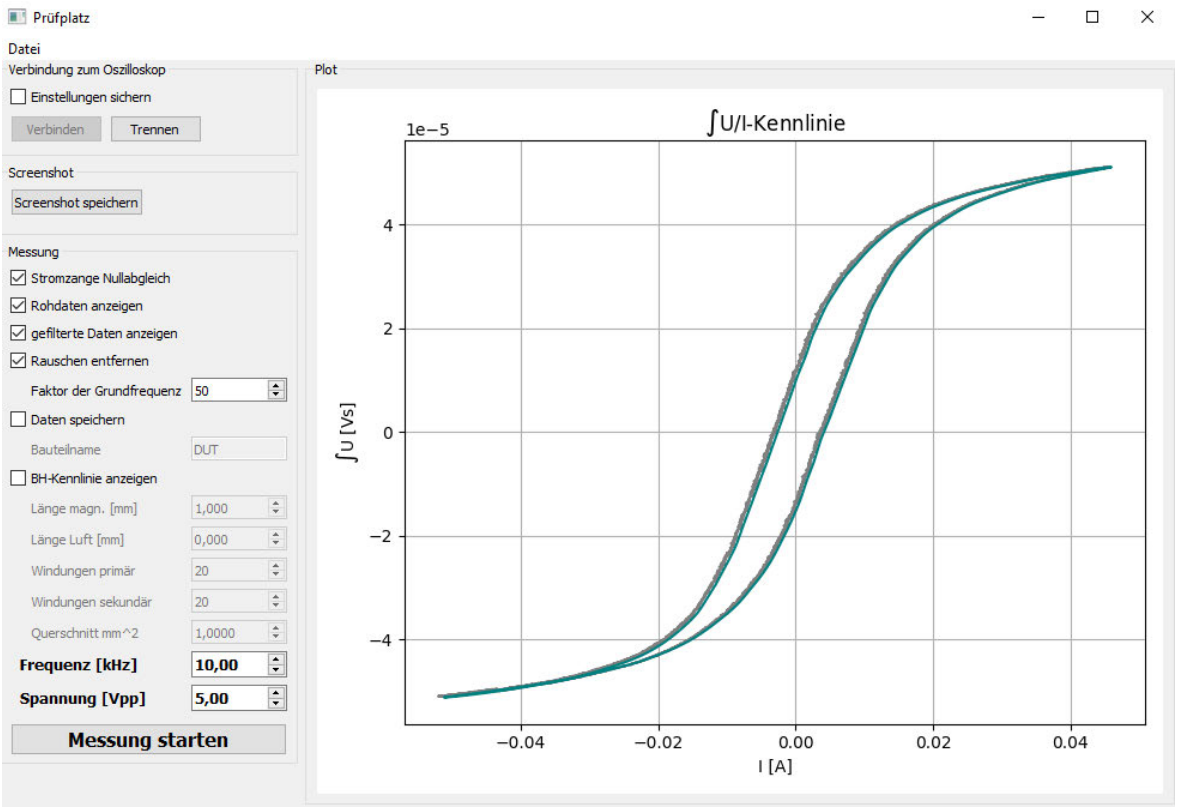


Abbildung 61 Graphische Oberfläche nach einer Messung

Durch die Entwicklung des Programms in Thonny wurde das Programm darüber aufgerufen. Für die spätere Nutzung ist ein Aufruf über den Desktop wünschenswert. Die dafür notwendige Batchdatei besteht aus der Pfadangabe zur ausführbaren Python.exe-Datei und dem vollständigen Dateipfad des Python-Skriptes. Außerdem wird die Wiederholung der Befehle unterdrückt und das Shell-Fenster schließt sich nach dem Schließen des Pythons-Skriptes.

Pruefplatz.bat

```
1  @echo off
2  "C:\Program Files (x86)\Thonny\python.exe" "D:\thonny\Pruefplatz.py"
3  exit
```

Abbildung 62 Start eines Python-Skripts über eine Batchdatei

5 Erprobung und Auswertung

In diesem Kapitel wird die Aufnahme der Hysteresekurve mit dem erstellten Programm getestet. Die gesamte Entwicklung wurde in 3 Hauptabschnitte geteilt:

- Aufbau der Hardware
- Entwicklung eines Python-Skripts für die gewünschte Funktionalität und in Vorbereitung auf die graphische Oberfläche
- Entwicklung der graphischen Oberfläche und Nutzung des erstellten Skriptes

Diese drei Abschnitte wurden in kleinere Aufgaben unterteilt und es wurden jeweils Tests nach Fertigstellung der Aufgaben durchgeführt. Bis zur Fertigstellung der graphischen Oberfläche wurde immer der Signalverstärker Toellner 7607 und das Bauelement Vogt_24_4 verwendet. Bei diesen Übertragern beginnt die Sättigung schon ab ungefähr 30 mA (siehe Abb. 61). Viele Übertrager erfordern aber höhere Ströme, um in die Sättigung zu kommen. Daher sollte der Toellner-Verstärker durch ein leistungsfähigeres Gerät ersetzt werden. Das Unternehmen hatte für einen begrenzten Zeitraum ein Leihgerät Hubert A1110-40-QE erhalten, um feststellen zu können, ob dieses Gerät den Anforderungen an dieses Projekt entspricht.

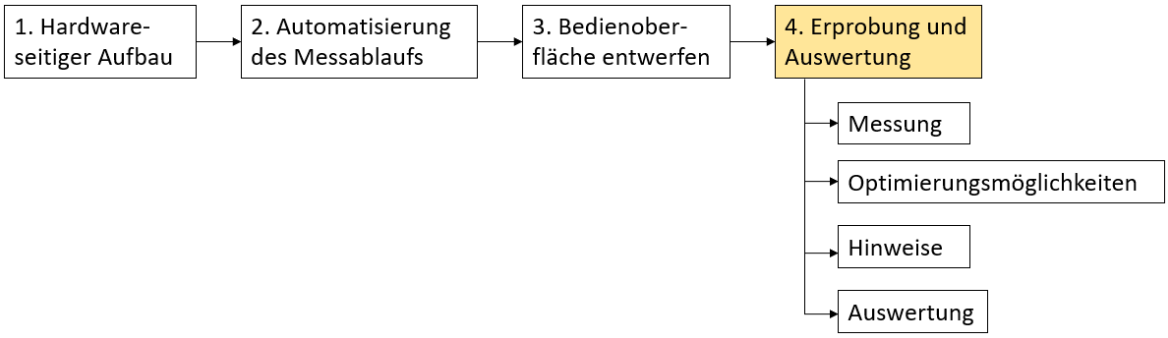


Abbildung 63 Letzter Entwicklungsabschnitt

5.1 Messung mit Toellner Signalverstärker

Der Signalverstärker Toellner TOE7607 hat folgende Eigenschaften (26):

- Frequenzbereich 0 bis 5 MHz
- Ausgangsspannung 0 bis 40 V_{pp} an 50 Ω
- Ausgangsimpedanz ca. 10 Ω
- Ausgangstrom 0,8 A_{pp}

Die Peak-Peak-Angaben bedeuten einen Effektivwert von rund 14 V und einen Ausgangsstrom von rund 280 mA an einer Last von 50 Ω. Damit lassen sich nahezu alle im Unternehmen eingesetzten Übertrager nicht in die Sättigung treiben. Dem Unternehmen wurde ein Leihgerät Hubert A1110-40-QE zur Verfügung gestellt, was einen höheren Ausgangsstrom und -spannung liefern kann (siehe Abs. 5.2).

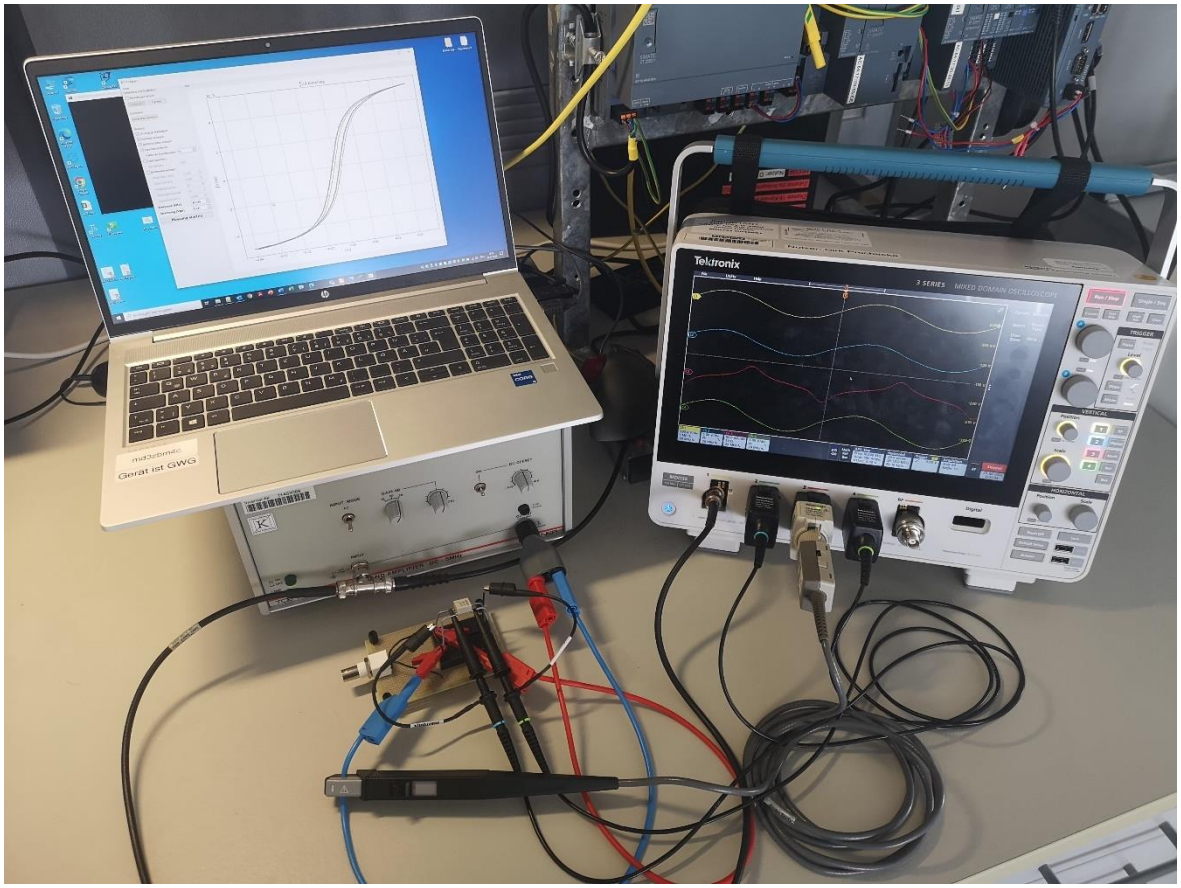


Abbildung 64 Messaufbau komplett

Der komplette Messaufbau umfasst das Oszilloskop mit Frequenzgenerator, den Signalverstärker, den PC mit dem Programm und allen notwendigen Tastköpfen und Verbindungen.

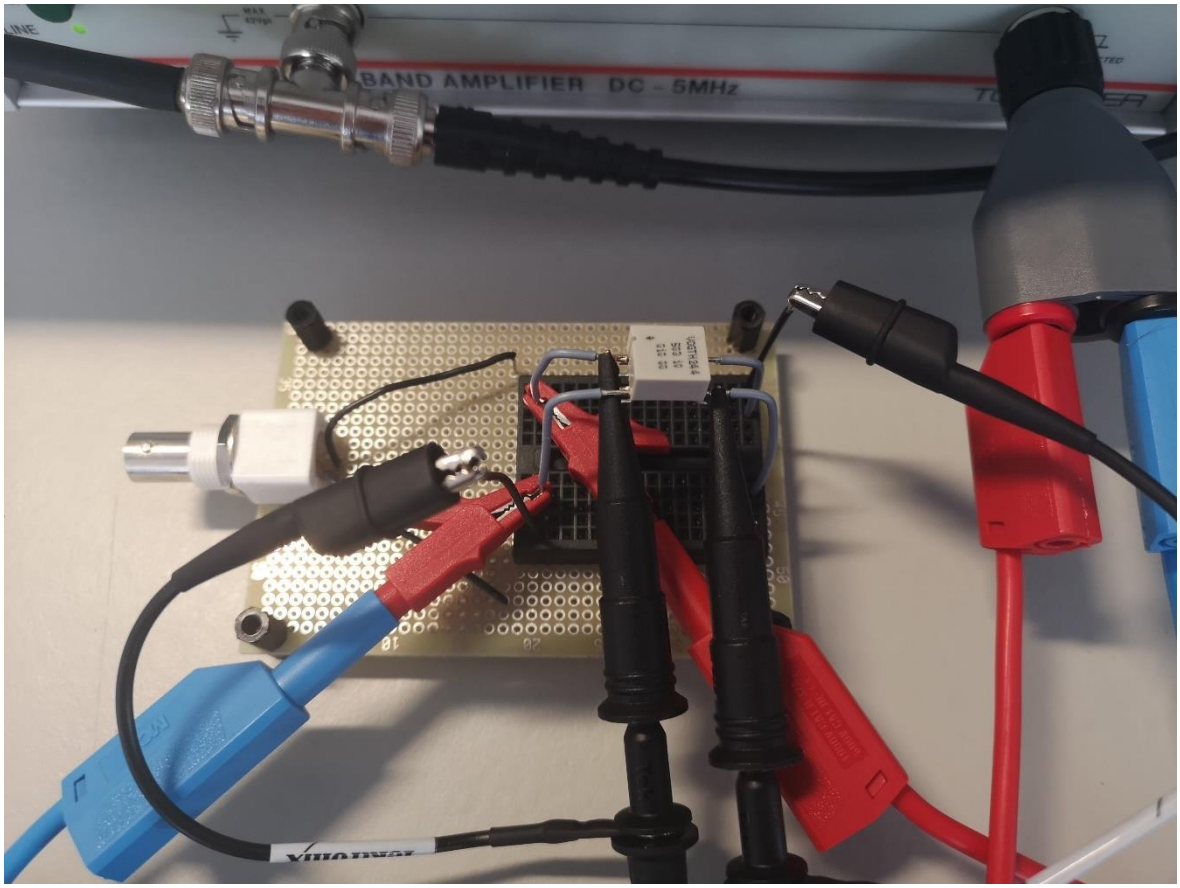


Abbildung 65 Messaufbau, Anschluss Übertrager

Die Detailansicht zeigt den Übertrager im Messaufbau. Die Primärseite (links) wird vom Signalverstärker gespeist.

Messungen mit dem Toellner-Verstärker wurden an dem Bauteil Vogt_24_4 bei verschiedenen Frequenzen durchgeführt (Datenblatt siehe Anlage Teil 4). Der Primärstrom und die integrierte Sekundärspannung (Messdaten, Rohdaten) werden in den Diagrammen grau dargestellt. Nach Entfernung des Gleichanteils und der höherfrequenten Anteile wird der Primärstrom und die integrierte Spannung mit der grünen Kennlinie sichtbar.

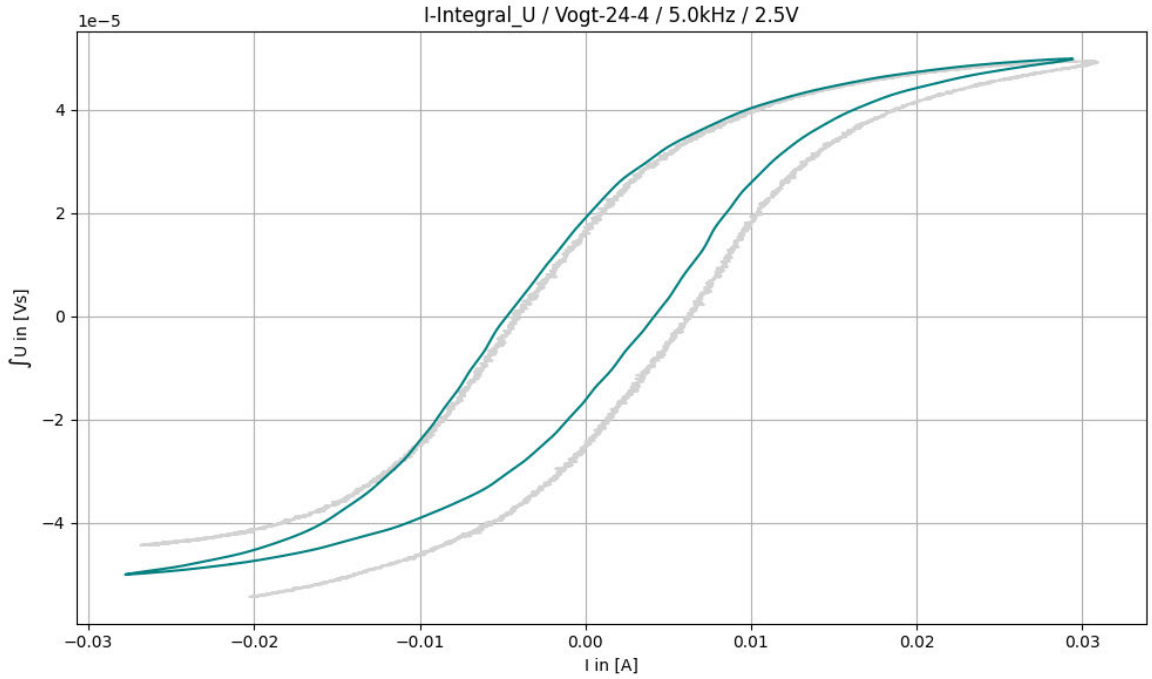


Abbildung 66 Hysteresekurve 5kHz 2.5V

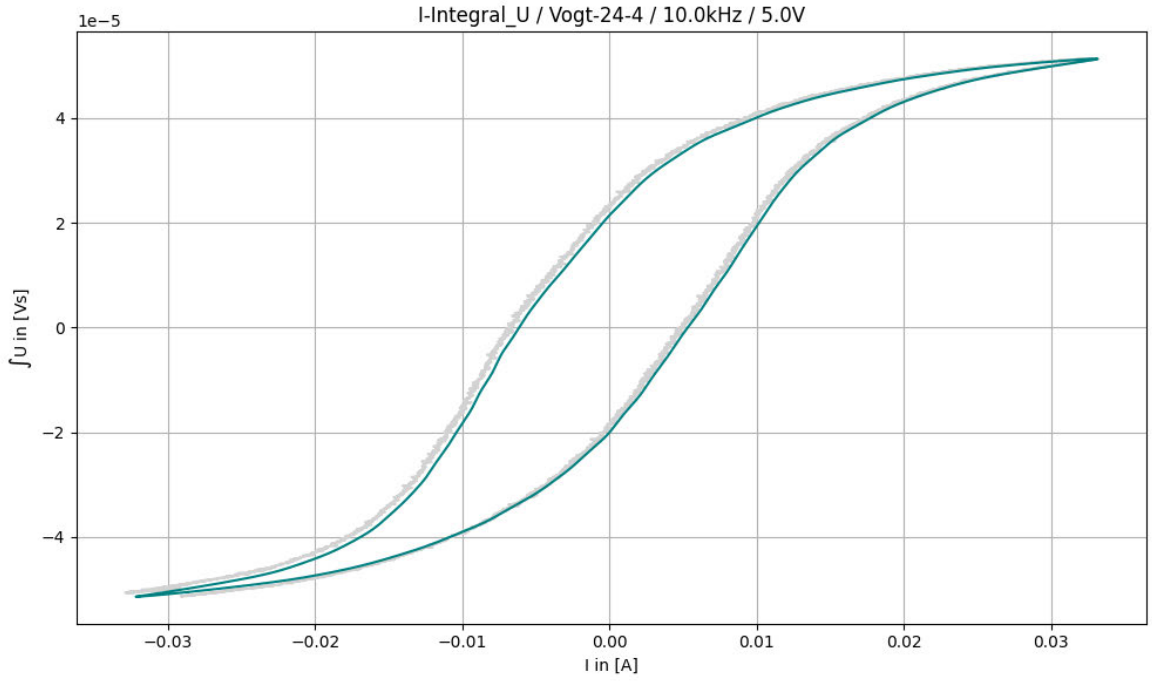


Abbildung 67 Hysteresekurve 10kHz 5V

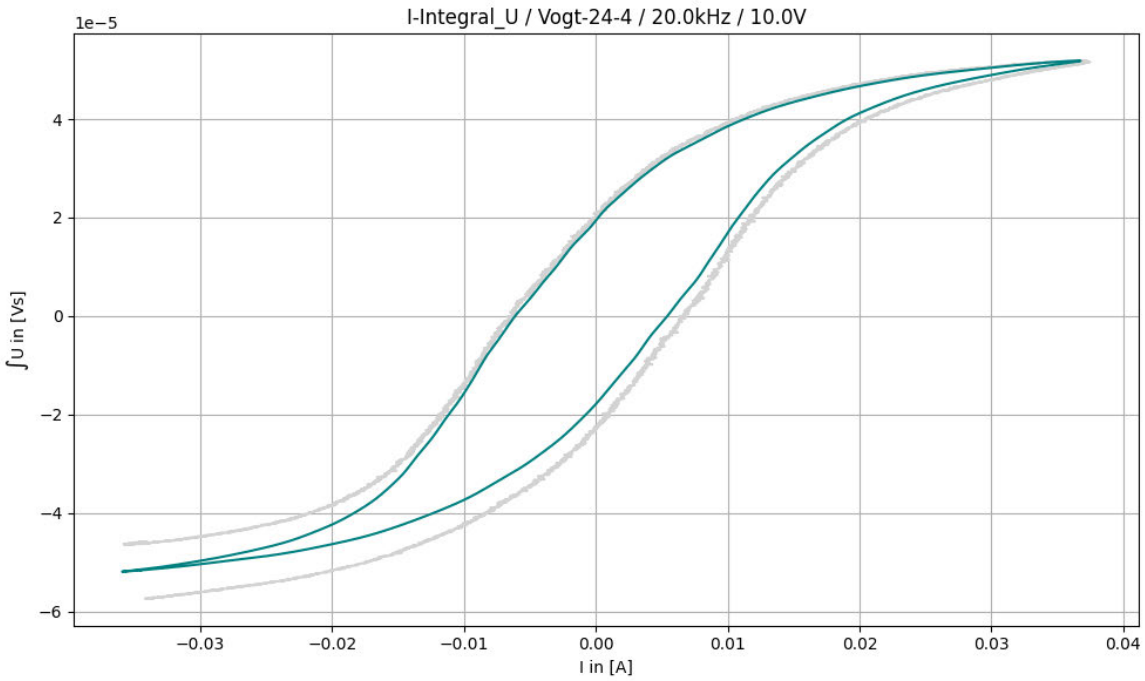


Abbildung 68 Hysteresekurve 20kHz 10V

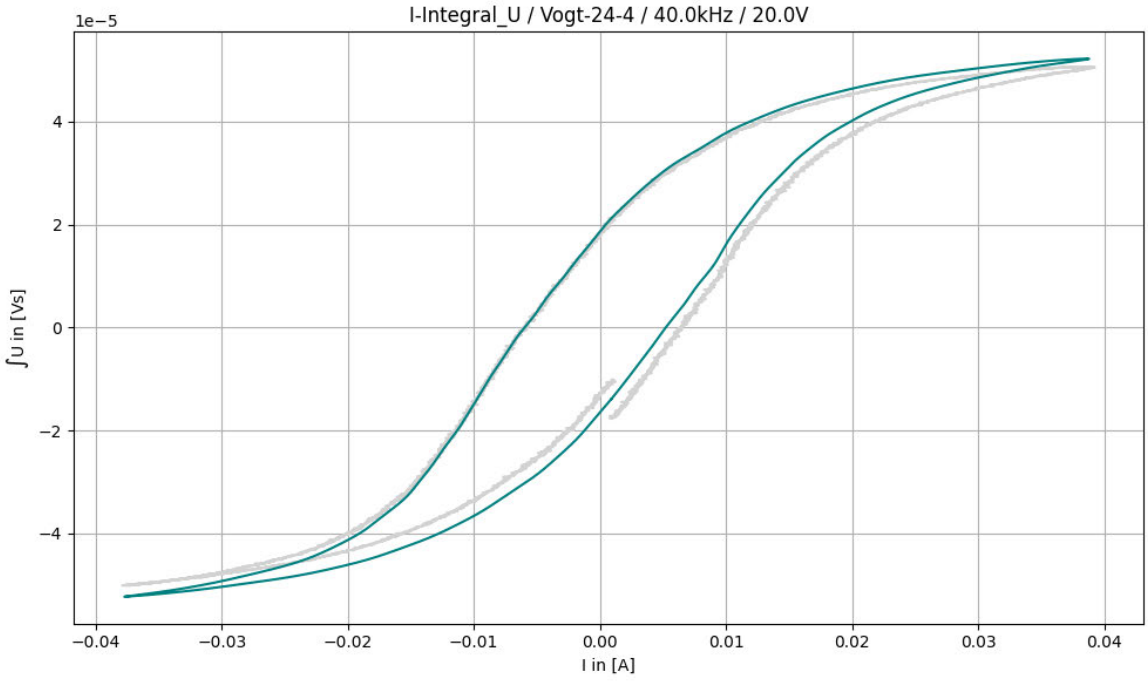


Abbildung 69 Hysteresekurve 40kHz 20V

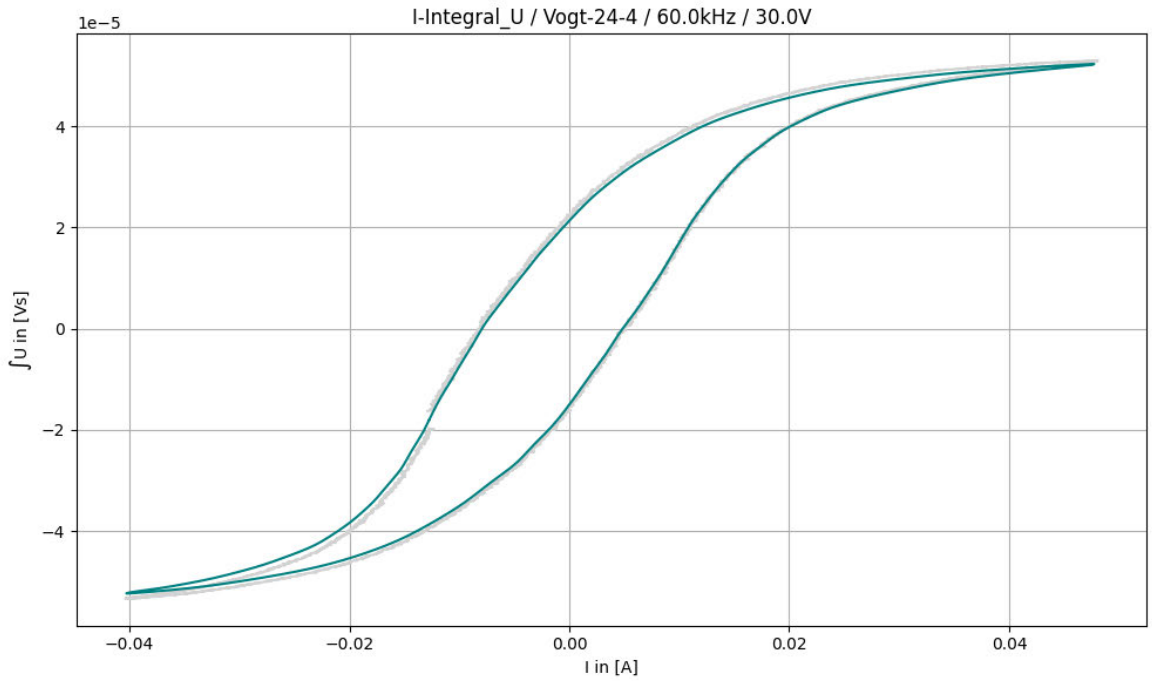


Abbildung 70 Hysteresekurve 60kHz 30V

Die Messungen begannen bei 5kHz, die Spannung wurde so gewählt, dass eine ausgebildete Hysteresekurve sichtbar wird. Es wurde immer die Frequenz und Spannung verdoppelt. Gut sichtbar ist außerdem der Versatz, der durch den Gleichspannungsanteil in den Rohdaten entsteht. Dieser wird im Frequenzbereich bei der FFT entfernt.

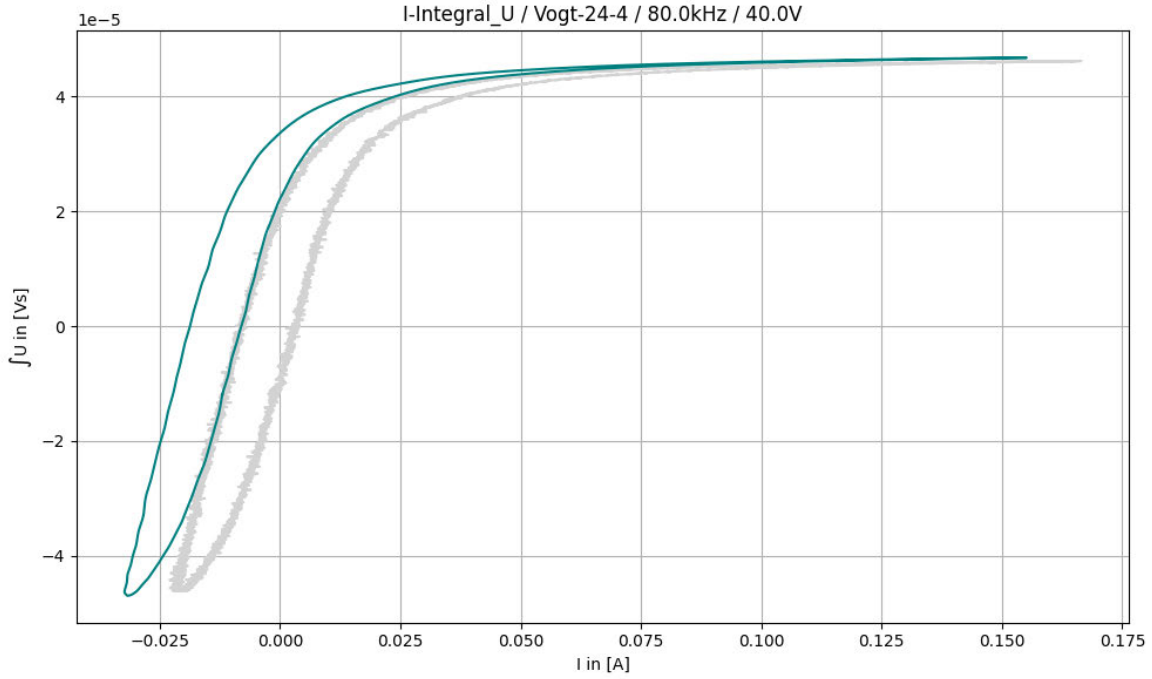


Abbildung 71 Hysteresekurve 80kHz 40V

In Abbildung 71 kann der Signalverstärker auf der negativen Seite die Spannung nicht mehr bereitstellen. Der Spannungsverlauf wird abgeschnitten (clipping).

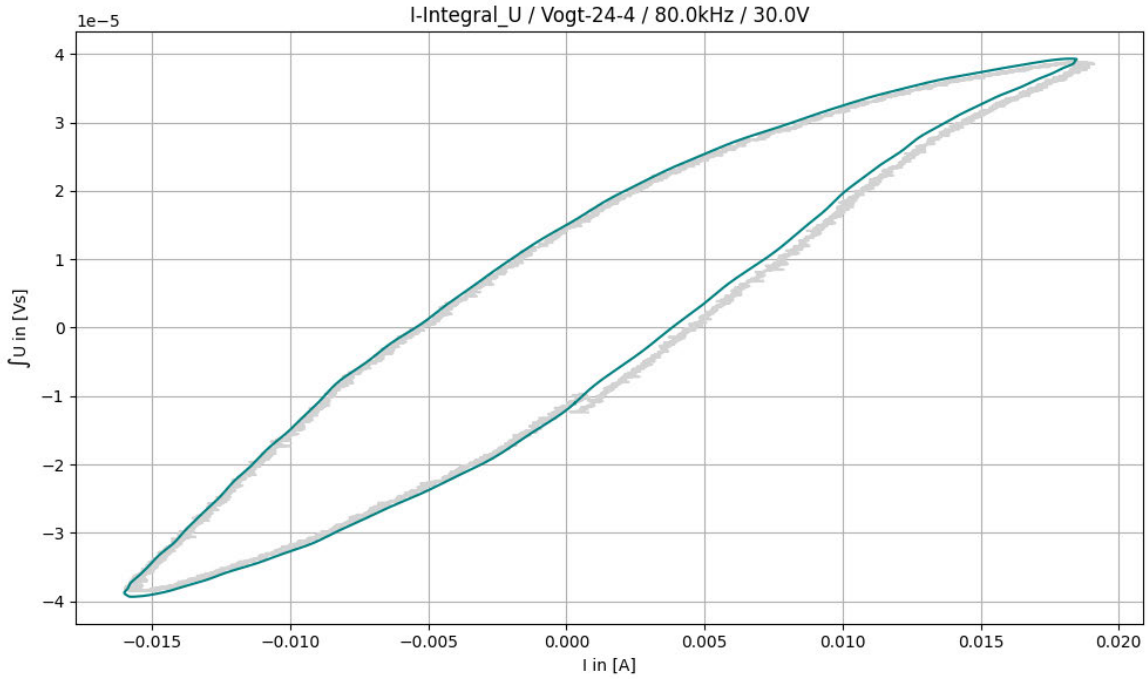


Abbildung 72 Hysteresekurve 80kHz 30V

Daraufhin wurde die Spannung reduziert mit der Folge, dass die Hysteresekurve nicht mehr ausgeprägt ist. Eine weitere Frequenzerhöhung macht keinen Sinn, da die Spannung nicht weiter erhöht werden kann.

Als nächster Schritt folgt die Anzeige als BH-Kennlinie, dafür müssen Kerndaten bekannt sein. Das Datenblatt gibt für den Kern die Maße 5.84x3.05x2.5 an. Mit dem Messschieber wurde 5.84 mm als Außendurchmesser ermittelt, der Innendurchmesser ist 2.5 mm und die Höhe ist 3.84 mm. Daher ergibt sich eine Fläche (rechteckförmig mit abgerundeten Ecken) von 5.0935mm^2. Die mittlere magnetische Länge ist 13.1mm. Die Windungsanzahl ist im Datenblatt nur als Verhältnis angegeben, konnte unter dem Mikroskop mit 14 Windungen primärseitig und 19 Windungen sekundärseitig ermittelt werden. Nach Eingabe dieser Daten ergibt sich folgende Hysterese Kennlinie:

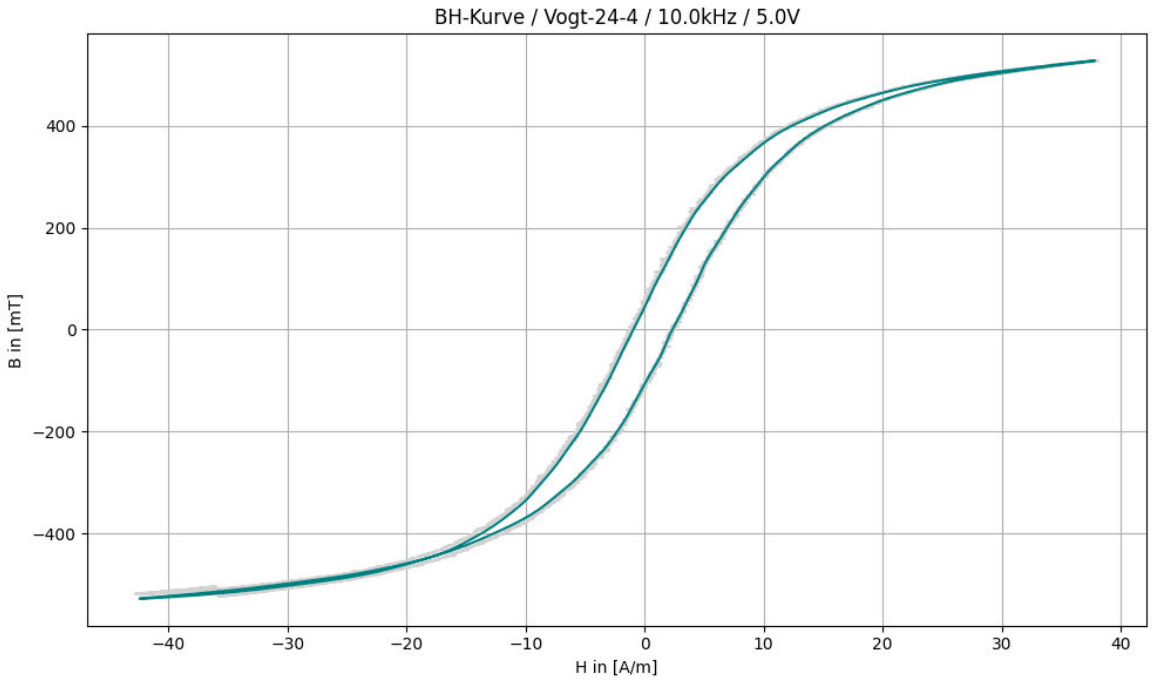


Abbildung 73- BH-Kennlinie

Das Datenblatt zum verwendeten Kernmaterial Fi360 zeigt folgende Magnetisierungskurve (27):

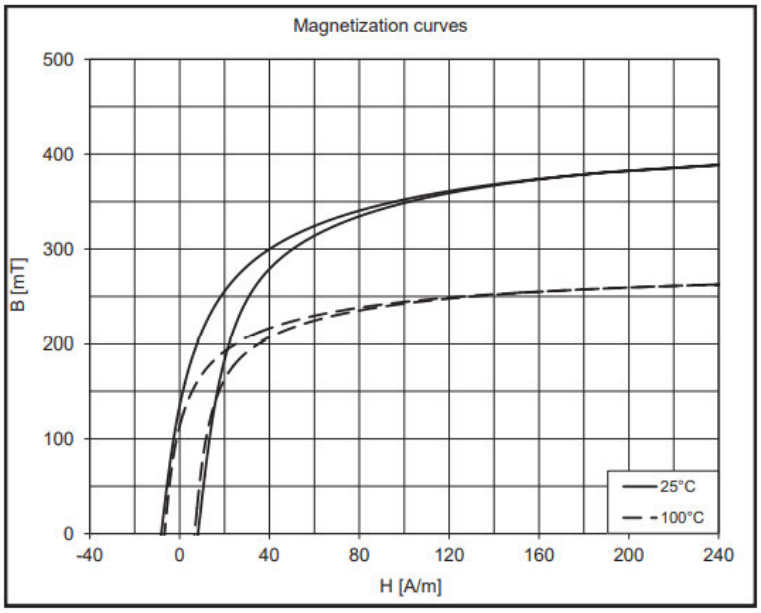


Abbildung 74 Magnetisierungskurve Fi360

Die ermittelte maximale magnetische Flussdichte B_{max} ist mit rund 500mT höher als die im Datenblatt angegebene Flussdichte von 380mT. Noch größere Abweichungen gibt es bei der magnetischen Feldstärke (wo beide Kennlinien zusammenkommen) von 25 A/m zu 100 A/m. Da das Unternehmen aber vorrangig Vergleichsmessungen von Bauteilen

durchführen möchte anstatt Verifizierung der Datenblattangaben, wurde die Ursache dieser Abweichungen nicht weiter erforscht.

5.2 Messung mit Hubert Signalverstärker

Das Leihgerät Hubert A1110-40-QE hat folgende Eigenschaften (28):

- Frequenzbereich 0 bis 200 kHz
- Ausgangsspannung 0 bis ±75 V_{peak}
- Ausgangsimpedanz gegen 0 Ω
- Ausgangstrom ±40 A_{peak}

Für die zu messenden Übertrager ist dieses Gerät ausreichend dimensioniert. Bei der Aufnahme einer Hysteresekurve im Vergleich zum Toellner-Gerät war die Hysteresekurve einseitig verschoben. Daher wurde zuerst das Offset des Frequenzgenerators angepasst, was keine Verbesserung nach sich zog. Anschließend wurden Vergleichsmessungen zum Toellner-Verstärker gemacht und diese zeigten bei einer induktiven Last an einem Rechtecksignal einen einseitig verschobenen Strom.

- Kanal 1 (gelb): Spannung Ausgang Frequenzgenerator, Eingang Signalverstärker
- Kanal 2 (blau): Spannung Ausgang Signalverstärker, Primärseite Übertrager
- Kanal 3 (rot): Strom Ausgang Signalverstärker, Primärseite Übertrager
- induktive Last L=1.5mH, R_{DC}=370mΩ, f=10kHz, Sollspannung 1V_{pp} (Faktor 10)

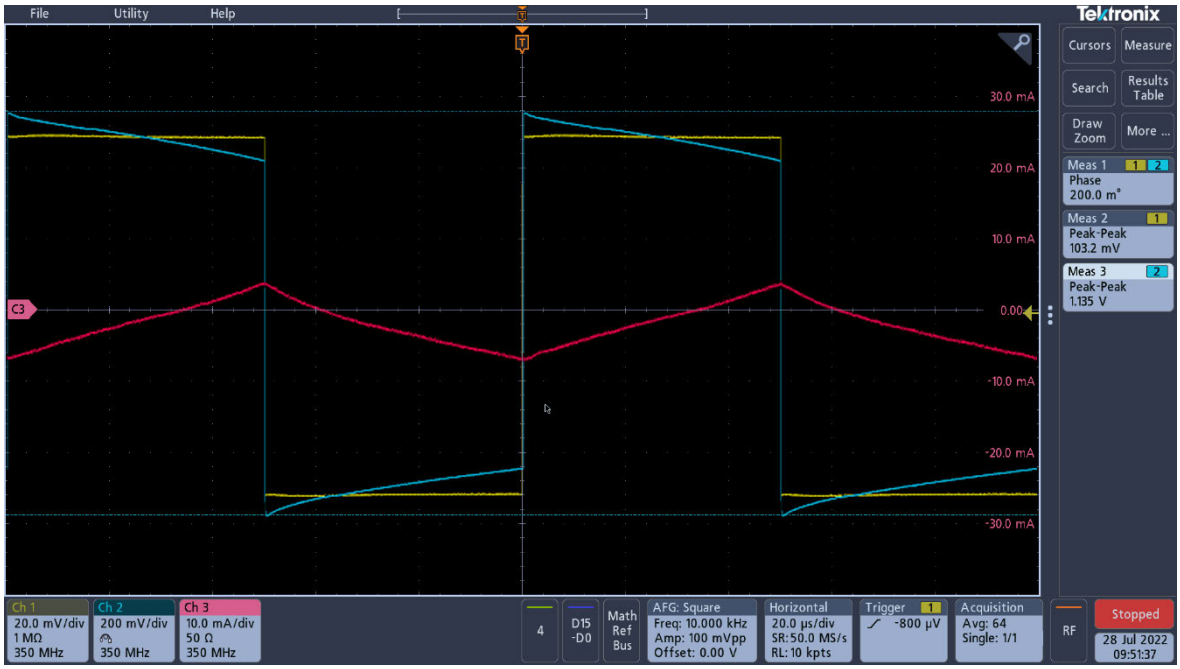


Abbildung 75 Stromverhalten induktive Last, Toellner

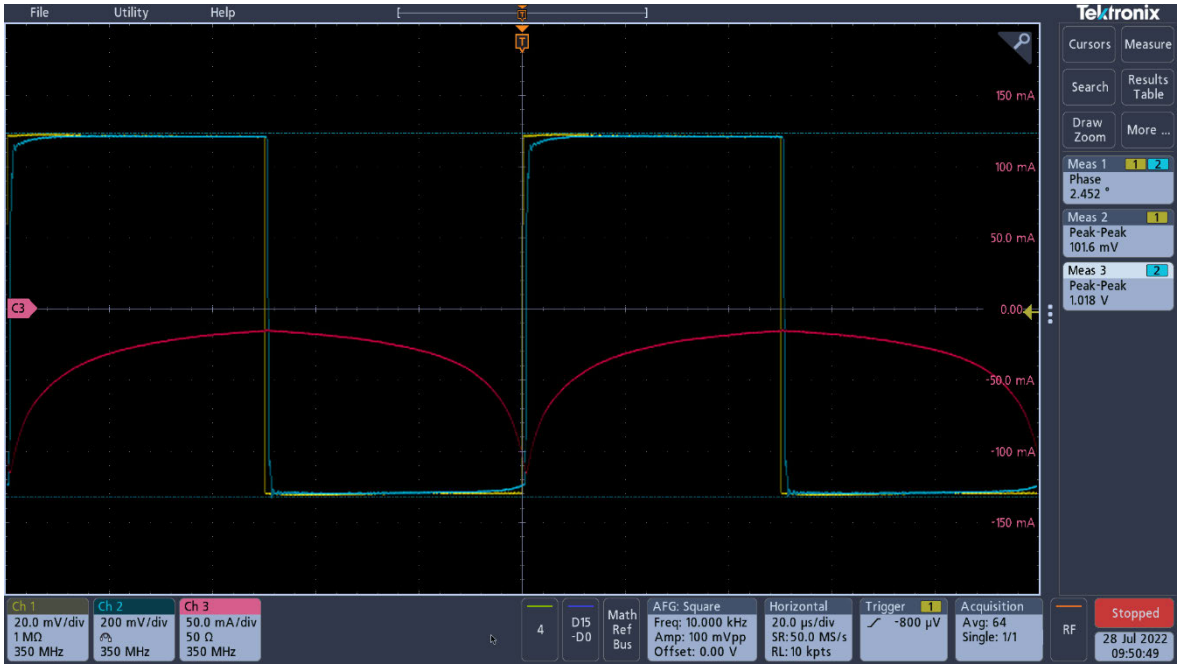


Abbildung 76 Stromverhalten induktive Last, Hubert

Der Stromverlauf beim Toellner-Verstärker ist nahezu symmetrisch, während der Stromverlauf beim Hubert-Verstärker nur negative Amplituden aufweist (Marke „C3“ ist 0 Ampere).

Eine weitere Messung zeigt, dass ab einem bestimmten Strom der Hubert-Verstärker ungewollte Schwingungen zeigt.

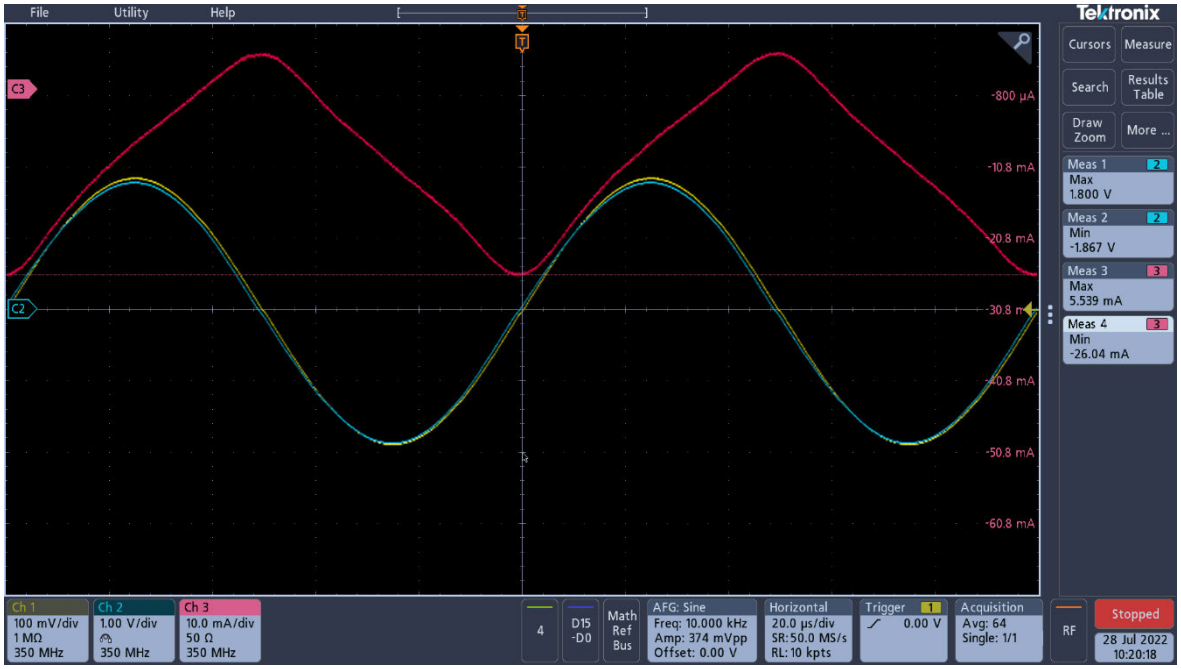


Abbildung 77 Stromverlauf ohne Schwingungen, Hubert

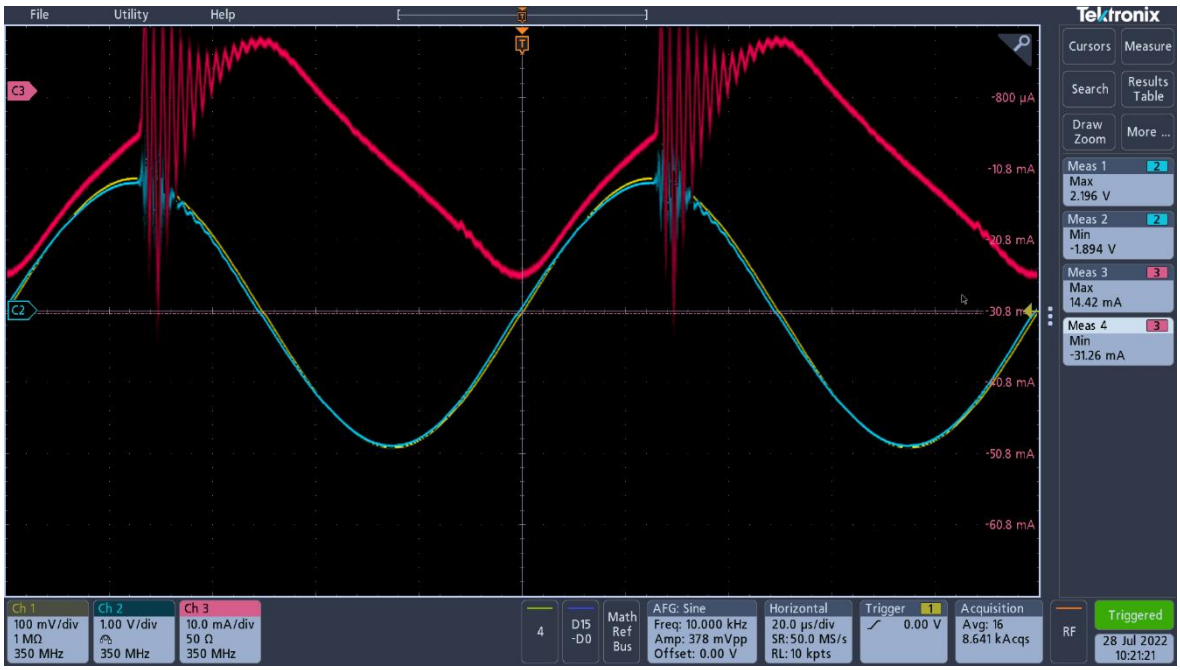


Abbildung 78 Stromverlauf mit Schwingungen, Hubert

Die beiden Abbildungen zeigen den Stromverlauf, wenn aufgrund der Spannungserhöhung primärseitig an der induktiven Last der Laststrom ansteigt. Es sind Schwingungen sichtbar, die beim Toellner-Verstärker nie sichtbar waren. Um Probleme mit dem Stromtastkopf auszuschließen, wurde diese deaktiviert und der primärseitige Strom über einen Messwiderstand von 1Ω gemessen.

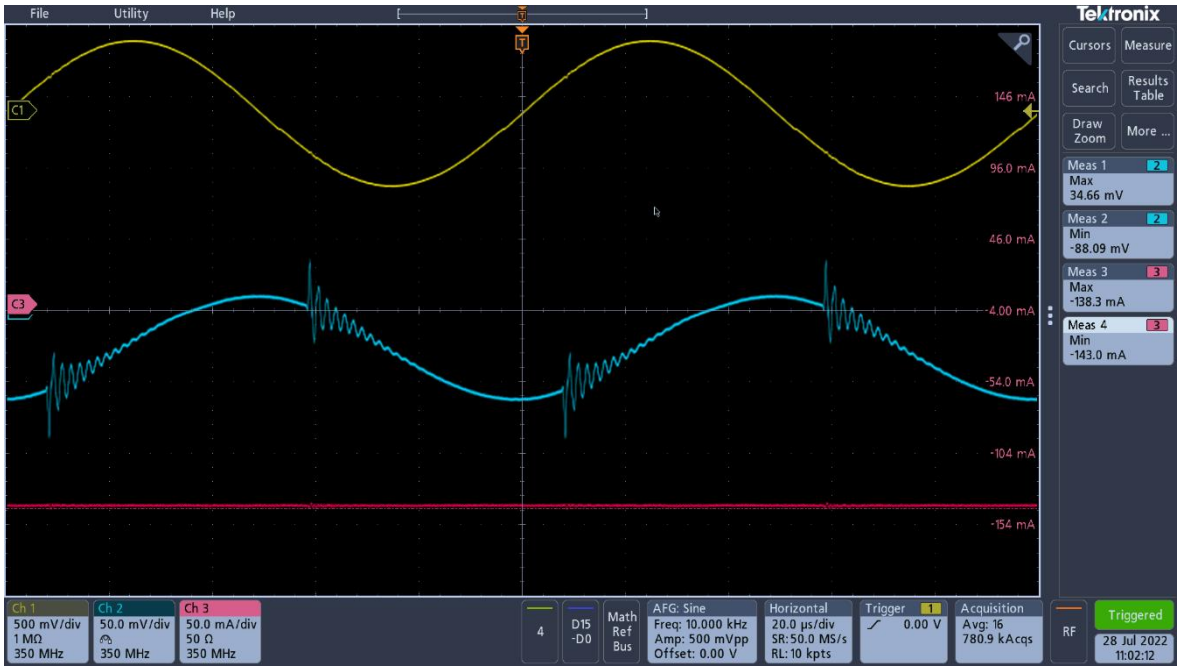


Abbildung 79 Schwingung über Messwiderstand, Hubert

Es wurden noch verschiedene Messungen durchgeführt, um Probleme beim Messaufbau auszuschließen:

- Verstärker galvanisch getrennt
- Oszilloskop galvanisch getrennt
- Verwendung eines Handheld-Oszilloskop mit galvanisch getrennten Eingängen
- verschiedene Offset-Einstellungen
- Messungen mit kapazitiver Last

Keine der Änderungen am Messaufbau brachte Verbesserungen. Insbesondere die Messung mit kapazitiver Last (C=220nF) zeigt ein deutliches Schwingverhalten:

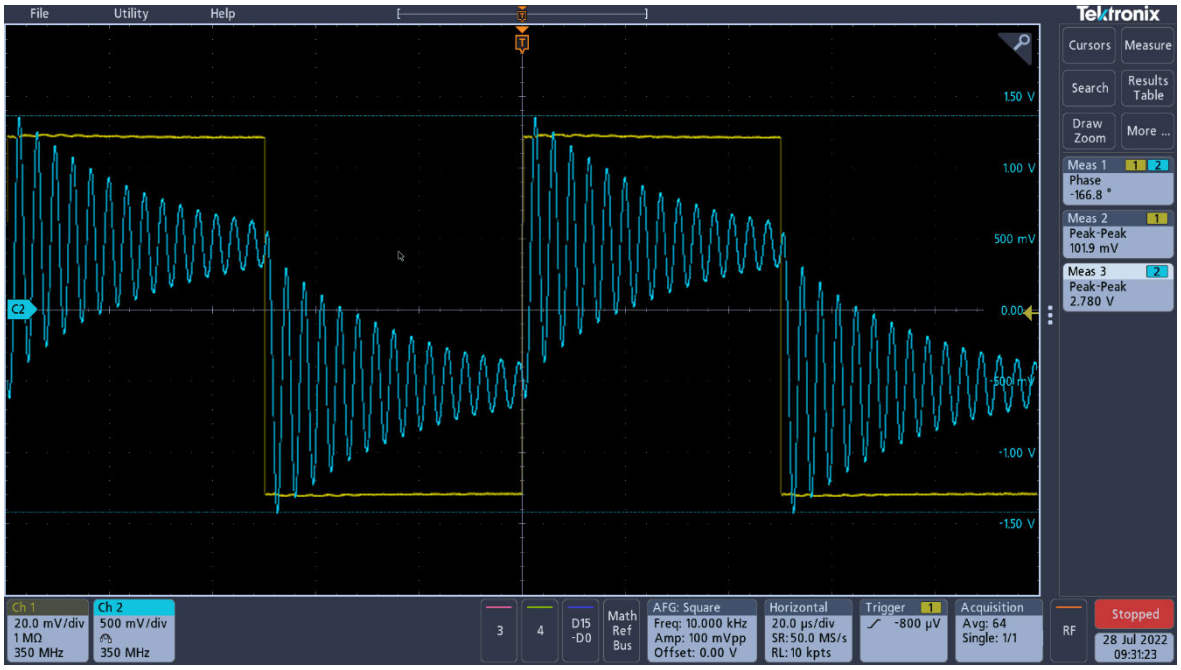


Abbildung 80 Kapazitive Last, Hubert

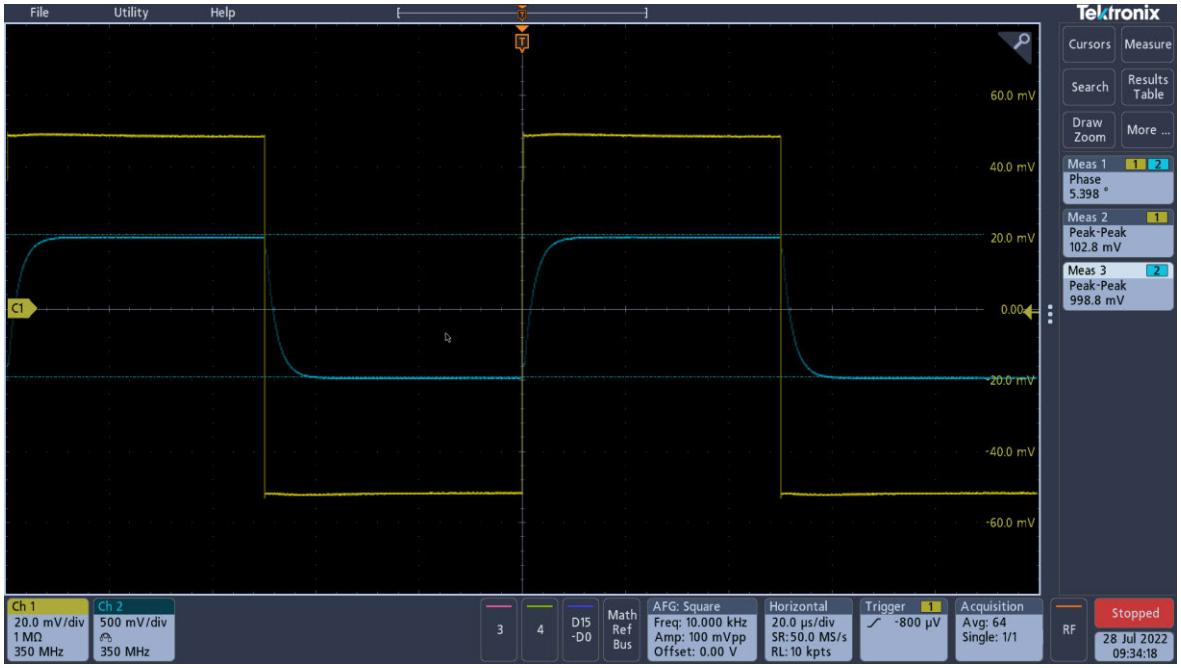


Abbildung 81 Kapazitive Last, Toellner

Während es beim Hubert-Verstärker zu Schwingungen kommt, zeigt der Toellner-Verstärker das Verhalten eines RC-Gliedes. Der Hersteller wurde kontaktiert und bestätigte ein ungewolltes Verhalten des Gerätes. Da sich das Leihgerät als defekt erwiesen hat, bestehen berechtigte Zweifel, ob ein Verstärker dieses Unternehmens überhaupt die gewünschten Anforderungen erfüllt. Das bedeutet, das zurzeit nur der Toellner-Verstärker verwendet werden kann und somit keine Übertrager am Testplatz gemessen werden können, deren Sättigungsstrom höher als 200 mA ist.

5.3 Optimierungsmöglichkeiten

5.3.1 Berechnung des Flächeninhaltes der Hysteresekurve

In Abs. 3.3.2 wurde auf die optionale Berechnung des Flächeninhaltes der Hysteresekurve eingegangen. Aufgrund der Schwierigkeiten mit der Flächeninhaltsberechnung bei verrauschten Signalen anhand der Trapezmethode wurde diese Berechnung nicht implementiert. Wenn eine berechnete Fläche ausgegeben wird, dann sollte der Nutzer sich auf diese Angabe verlassen können. Hier könnte eine Curve-Fitting-Funktion untersucht werden. Das bedeutet, der gegebene Verlauf der Hysteresekurve muss in den fallenden und den steigenden Teil aufgeteilt werden. Anschließend wird für beide Teile eine Annäherung an eine mathematische Funktion (zum Beispiel an eine tanh-Funktion) erzeugt und der Flächeninhalt ermittelt.

5.3.2 Toolbar in den Plot aufnehmen

Wird der Python-Skript aus Kapitel 3 ausgeführt, erscheint ein Plot, der eine Toolbar beinhaltet. Diese erlaubt es, verschiedene Einstellungen am Plot vorzunehmen, die Graphik zu speichern als auch bestimmte Teile der Ausgabe vergrößert darzustellen (Zoom-Funktion). Wird das Programm aus Kapitel 4 mit der graphischen Oberfläche ausgeführt, gibt es diese Möglichkeit nicht. Es könnte untersucht werden, ob eine Möglichkeit der Implementierung besteht. Das unterschiedliche Verhalten liegt daran, das im Python-Skript die Bibliothek „matplotlib“ verwendet wird, während bei der graphischen Oberfläche eine andere Bibliothek („matplotlib.backends.backend_qt5agg“) genutzt werden muss.

Abbildung 82 Toolbar im Plot

5.4 Hinweise

5.4.1 Measurement-Funktion des Oszilloskops nicht zuverlässig

Die Measurement-Funktion des Oszilloskops zeigt oft „Low Signal Amplitude“ an (siehe Abs. 3.2.2). Das Problem lässt sich nicht durch eine Änderung des Teilers lösen. Wird die Meldung angezeigt, können die Messwerte fehlerhaft sein. Eine Abfrage der Meldung ist möglich. Werden bei Programmerweiterungen Measurement-Werte benötigt, sollte darauf geachtet werden, dass diese unter allen Umständen zuverlässig sind.

5.4.2 Einsatz eines Signalverstärkers mit einer anderen Verstärkung

Bei dem Toellner-Verstärker ist die Verstärkung einstellbar, wurde jedoch durch Vergleich des Ein- und Ausgangs im unbelasteten Zustand auf einen Faktor 10 eingestellt. Auch der Hubert-Verstärker (das Leihgerät) hatte eine Verstärkung von 10. Bei Einsatz eines Signalverstärkers mit einem anderen Verstärkungsfaktor muss dieser im Programmcode implementiert werden. In beiden Varianten findet man diesen Faktor in den Variablen unter „SA-RATIO“.

```
28  # Verstärkungsfaktor Signalverstärker
29  SA_RATIO = 10
```

Abbildung 83 Änderung des Verstärkungsfaktors im Quellcode

5.5 Auswertung

Notwendig ist der Kauf oder die Entwicklung eines geeigneten Signalverstärkers. Erst dann können die im Unternehmen eingesetzten Übertrager einer Messung der Hysteresekurve unterzogen werden. Die dafür notwendige Software, die den Messablauf automatisiert, wurde erfolgreich entwickelt und soweit es möglich war, erprobt.

Literatur

1. **Stiny, Leonhard.** *Passive elektronische Bauelemente.* Haag a.d. Amper : Springer Vieweg, 2015. ISBN 978-3-658-24732-4.

2. **Zacharias, Peter.** *Magnetische Bauelemente.* Kassel : Springer Vieweg, 2020. ISBN 978-3-658-24741-6.

3. **Bieneck, Wolfgang.** *Elektro T Grundlagen der Elektrotechnik.* Stuttgart : Holland + Josenhans, 2010. ISBN 978-3-7782-4900-0.

4. **Czech, J.** *Oszillografen-Messtechnik.* Berlin-Borsigwalde : Verlag für Radio-Foto-Kinotechnik GmbH, 1959.

5. **Benz, Wilhelm.** *Messtechnik Messgeräte Messmethoden Messübungen.* Neusäß : Kieser Verlag GmbH, 1993. ISBN 3-8242-2005-9.

6. **Forum, Engineering.** Engineering Forum. [Online] [Zitat vom: 19. Juli 2022.] https://www.eng-tips.com/viewthread.cfm?qid=472050.

7. **NI-VISA, Überblick.** NI-VISA Überblick. [Online] [Zitat vom: 13. Juli 2022.] https://www.ni.com/de-de/support/documentation/supplemental/06/ni-visa-overview.html.

8. **NI-VISA, Download.** NI-VISA Download. [Online] [Zitat vom: 13. Juli 2022.] https://www.ni.com/de-de/support/downloads/drivers/download.ni-visa.html#442805.

9. **HAMEG.** *SCPI Programmierhandbuch für HM1000x.* s.l. : HAMEG. http://www.pewa.de/DATENBLATT/DBL_HM_HM2005_DATENBLATT1_DEUTSCH.pdf.

10. **Tektronix, Programmer Manual 3-Series.** *3 Series MDO Oscilloscopes Programmer Manual.* Beaverton, USA : Tektronix, Inc., 2022. https://download.tek.com/manual/3-MDO-Oscilloscope-Programmer-Manual-077149800.pdf.

11. **IDE-Thonny.** IDE Thonny. [Online] [Zitat vom: 2022. Juni 2022.] https://thonny.org/.

12. **PyVISA_Kommunikation.** PyVISA_Kommunikation. [Online] [Zitat vom: 14. Juli 2022.] https://pyvisa.readthedocs.io/en/latest/introduction/communication.html.

13. **Johannes Ernesti, Peter Kaiser.** *Python 3 Das umfassende Handbuch.* Bonn : Rheinwerk Verlag, 2015. ISBN 978-3-8362-3633-1.

14. **GPIB.** GPIB. [Online] [Zitat vom: 14. Juli 2022.] https://documentation.help/NI-488.2/gpib6hir.html.

15. **GitHub.** GitHub, Tektronix . [Online] [Zitat vom: 15. Juli 2022.] https://github.com/tektronix/Programmatic-Control-Examples/tree/master/Examples/Oscilloscopes/TekSeriesScopes_HighSpeedDigitizers.

16. **PyVISA_Reading_Values.** PyVISA_Reading_Values. [Online] [Zitat vom: 15. Juli 2022.] https://pyvisa.readthedocs.io/en/latest/introduction/rvalues.html.

17. **Steinkamp, Veit.** *Der Python-Kurs für Ingenieure und Naturwissenschaftler.* Bonn : Rheinwerk Technik, 2021. ISBN 978-3-8362-7316-9.

18. **Numpy_Reference.** Numpy_Reference. [Online] [Zitat vom: 15. Juli 2022.] https://numpy.org/doc/stable/reference/routines.io.html.

19. **Tektronix, Bedienungsanleitung.** *Tektronix_MDO34_Bedienungsanleitung.* Beaverton : Tektronix, 2022. https://download.tek.com/manual/3SeriesMDO-Printable-Help-DE-DE-077158800.pdf.

20. **Scipy_Integration_Trapez.** Scipy_Integration_Trapez. [Online] [Zitat vom: 15. Juli 2022.] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.cumulative_trapezoid.html.

21. **mathepedia-Trapezregel.** mathepedia-Trapezregel. [Online] [Zitat vom: 21. Juli 2022.] https://mathepedia.de/Trapezregel.html.

22. **Matplotlib_Reference.** Matplotlib_Reference. [Online] [Zitat vom: 19. Juli 2022.] https://matplotlib.org/stable/api/pyplot_summary.html.

23. **meetechniek.info.** meetechniek.info. [Online] [Zitat vom: 20. Juli 2022.] https://meettechniek.info/passive/magnetic-hysteresis.html.

24. **PyQt5.** PyQt5. [Online] [Zitat vom: 24. Juli 2022.] https://pypi.org/project/PyQt5/.

25. **Natt, Oliver.** *Physik mit Python.* Nürnberg : Springer Spektrum, 2020. ISBN 978-3-662-61273-6.

26. **Toellner.** Toellner_Datenblatt. [Online] [Zitat vom: 29. Juli 2022.] https://www.toellner.de/datenblaetter/de_7607.pdf.

27. **Sumida.** Sumida_Fi360. [Online] [Zitat vom: 29. Juli 2022.] http://www.qsi.co.kr/wp-content/uploads/2019/03/Fi360_2018.pdf?ckattempt=1.

28. **Hubert.** Hubert_Verstärker. [Online] [Zitat vom: 29. Juli 2022.] https://www.drhubert.de/126.html.

Anlagen

Anlagen, Teil 1

In Anlage 1 ist der Programmcode für das Speichern eines Screenshots als png-Datei hinterlegt

Programmcode „get_screenshot.py“

```
import pyvisa
from datetime import datetime

rm = pyvisa.ResourceManager()
mdo34 = rm.open_resource('TCPIP0::192.168.178.20::inst0::INSTR')

def get_screenshot(conn):
    print('Beginn "get screenshot"')
    fileSaveLocation = r'D:\thonny\pics\\' # Speicherort für das Bild
    conn.write('SAVe:IMAGe:FILEFormat PNG') # Format angeben
    conn.write('SAVe:IMAGe:INKSaver OFF') # OFF = Bild ohne
                                          # Invertierung der Farben
    conn.write('HARDCopy STARt') # Bereitstellen der Daten
    imgData = conn.read raw() # Rohdaten in Variable speichern
    dt = datetime.now() # aktuelle Zeit + Datum speichern
    fileName = dt.strftime("Scope_%Y%m%d_%H%M%S.png") # einen
    # Dateiname erstellen mit Einbindung der Zeit/des Datums
    imgFile = open(fileSaveLocation + fileName, "wb") # die
    # Bilddatei mit Schreibrechten öffnen
    imgFile.write(imgData) # die Rohdaten in die Datei speichern
    imgFile.close() # die Bilddatei schließen
    print('Ende "get_screenshot"\n')

get_screenshot(mdo34)

mdo34.close()
rm.close()
```

Anlagen, Teil 2

In Anlage 2 ist der vollständige Python-Skript aus Kapitel 3 hinterlegt

Programmcode „MDO34_BH_Darstellung.py“

```
# Dieses Python Programm stellt eine Verbindung mit dem
# Oszilloskop MDO34 her und startet einen Messablauf mit
# einer vom Nutzer festgelegten Frequenz und Spannung.
# unter Variablen sind verschiedene Optionen wählbar
# (Vorbereitung auf graphische Oberfläche)

import pyvisa
import time
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from scipy.fft import fft,ifft,fftfreq # für FFT

# Variablen
# Adresse für connect()
ADDRESS_MDO34 = 'TCPIP0::192.168.178.20::inst0::INSTR'
# Alias für Kanäle
CH_AFG = 'CH1'
CH_U_PRIM = 'CH2'
CH_CURRENT = 'CH3'
CH_U_SEC = 'CH4'
# Verstärkungsfaktor Signalverstärker
SA_RATIO = 10
# für Datentransfer
HORIZONTAL_DATAPOINTS = 1E4
# Variablen in Vorbereitung auf graphische Oberfläche
USE_CUTOFF_FREQ = 1 # gefilterte Daten -> Grenzfrequenz anwenden
SHOW_FILTERED_DATA = 1 # Anzeige der gefilterten Daten
SHOW_RAW_DATA = 1 # Anzeige der Rohdaten
AUTOZERO_CURRENT_PROBE = 1 # AutoZero ausführen, 1 = JA, 0 = NEIN
COMPONENT = 'Vogt-24-4' # Bezeichnung für Bauteil
SAVE_DATA = 1 # Daten speichern
CUTOFF_FREQUENCY = 50 # Grenzfrequenz als Faktor der Grundfrequenz
# (für FFT, 20...500 empfohlen)
SHOW_BH = 1 # 1 = BH-Kurve anzeigen, sonst I/intU
# BH Parameter
LM = 13.96 # Länge magnetisch in mm
LG = 0 # Länge Luftspalt in mm
N_PRIM = 14 # Primärwindungen
N_SEC = 19 # Sekundärwindungen
AREA = 6 # Fläche in mm**2

# ----------------------------------
# Unterprogramme
# Verbindungsaufbau und -abbau, Voreinstellungen,
# Sichern von Einstellungen
def connect():
```

```
    # versuche Verbindungsaufbau
    try:
        # rm gibt den Speicherort des Visa-Treibers zurück
        rm = pyvisa.ResourceManager()
        # eine Verbindung aufbauen und zurückgeben
        conn = rm.open resource(ADDRESS_MDO34)
    # wenn Oszilloskop offline ist
    except pyvisa.errors.VisaIOError as error_msg:
        print('Keine Verbindung möglich: {}'.format(error_msg))
        return 0
    # bei anderen Fehlern
    except:
        print('Unbekannter Fehler aufgetreten')
        return 0
    # wenn Verbindungsaufbau möglich war
    else:
        # Voreinstellungen, wie längeres Timeout,
        # durch Unterprogramm setzen
        set pre condition(conn)
        # Oszilloskop Einstellungen auf internen Speicher Nr.10 sichern
        # warten bis Befehl ausgeführt wurde
#         conn.write('*SAV 10')
#         r = conn.query('*OPC?')
        # Oszilloskop auf Werkseinstellungen setzen, auf Beenden warten
        conn.write('*RST')
        r = conn.query('*OPC?')
        # Verbindungsparameter zurückgeben
        return conn
def set_pre_condition(conn):
    # Zeit (in ms) verlängern wegen Timeout Fehler (Standard ist 2000ms)
    conn.timeout = 10000
    conn.encoding = 'latin_1'
    conn.read_termination = '\n'
    conn.write termination = None
    conn.write('HEAD 0')

def disconnect(conn):
    # Oszilloskop Einstellungen vom internen Speicher wiederherstellen
    # warten bis Befehl ausgeführt wurde
#     conn.write('*RCL 10')
#     r = conn.query('*OPC?')
    # Verbindung zum MDO34 beenden
    conn.close()

# Arbitrary Frequenz Generator (AFG)
# AFG anschalten
def set_AFG(conn, waveform, frequency, offset, amplitude):
    conn.write('AFG:OUTP:LOA:IMPED HIGHZ') # HighZ da höhere Spannung
mgl.
    conn.write('AFG:FUNC {}'.format(waveform)) # Waveform
    conn.write('AFG:FREQ {}'.format(frequency)) # Frequenz in Hz
    conn.write('AFG:OFFS {}'.format(offset)) # Offset in V
    conn.write('AFG:AMPL {}'.format(amplitude)) # Amplitude in V
    conn.write('AFG:OUTP:STATE ON') # AFG an
# AutoSet
def autoset(conn):
    # nur Kanal 1-4 darstellen
    conn.write('SEL:CH1 ON')
    conn.write('SEL:CH2 ON')
    conn.write('SEL:CH3 ON')
    conn.write('SEL:CH4 ON')
```

```
    conn.write('SEL:REF1 OFF')
    conn.write('SEL:REF2 OFF')
    conn.write('SEL:REF3 OFF')
    conn.write('SEL:REF4 OFF')
    conn.write('SEL:MATH OFF')
    # auf AC-Kopplung schalten, Stromzange (Kanal 3)
    # kann nur DC-gekoppelt werden
    conn.write('CH1:COUPling AC')
    conn.write('CH2:COUPling AC')
    conn.write('CH4:COUPling AC')
    # horizontale Auflösung, sonst Probleme beim Datentransfer
    conn.write('HORizontal:RECOrdlength {}'.format \
    (HORIZONTAL DATAPOINTS))
    # Stromzange AutoZero, wenn gewünscht
    if(AUTOZERO CURRENT PROBE):
        autozero current probe(conn, CH CURRENT)
    # die AutoSet Prozedur + warten auf Ende
    conn.write('AUTOS EXEC')
    r = conn.query('*OPC?')
    # für eine saubere Darstellung Average mit 64
    #Samples einstellen
    conn.write('ACQ:MOD AVE')
    conn.write('ACQ:NUMAV 64')
    time.sleep(1) # 1 Sekunde warten damit 64 Erfassungen vorhanden sind
    # AutoSet setzt auf volle Bandbreite, BW auf 20Mhz einstellen:
    conn.write('CH1:BANDWIDTH 20E6')
    conn.write('CH2:BANDWIDTH 20E6')
    conn.write('CH3:BANDWIDTH 20E6')
    conn.write('CH4:BANDWIDTH 20E6')
    # Anpassung der Amplitude des Strom-Kanals
    # (bei niedrigen Aplituden manchmal auf 1A)
    # wenn vertikale Skalierung auf 1 eingestellt ist, dann evtl.
    # Anpassung div nötig
    vertical scale = float(conn.query('{}:SCAle?'.format \
    (CH_CURRENT)))
    # Peak2Peak über 2 div
    if(vertical scale == 1.0):
        conn.write('MEASU:IMM:SOU {}'.format(CH_CURRENT)) # Auswahl Kanal
        conn.write('MEASU:IMM:TYP PK2Pk') # Peak2Peak messen
        curr p2p = float(conn.query('MEASU:IMM:VAL?'))
        # wenn peak2peak kleiner 50mA ist, dann 10mA - div einstellen
        if(curr_p2p < 0.2):
            conn.write('{}:SCA 0.01'.format(CH CURRENT))
    # Setzen der horizontalen Einstellung auf ca. 2 Perioden
    horizontal_scale = conn.query('HOR:SCA?') # Autoset zeigt ca 4
    # Perioden an
    horizontal_scale = float(horizontal_scale) # String zu Float
    # konvertieren
    horizontal scale = horizontal scale/2 # ca 2 Perioden anzeigen
    conn.write('HOR:SCA {}'.format(horizontal_scale))

# ACQ auf Stop (Einzelmessung) schalten und eine neue Messung starten
def set_ACQ_to_single(conn):
    conn.write('ACQ:STOPA SEQ') # auf Single (Stop) setzen
    conn.write('ACQ:STATE ON') # einzelnes Ereignis aufnehmen
    r = conn.query('*OPC?')

# Stromzange AutoZero
def autozero_current_probe(conn, channel):
    status_AFG = conn.query('AFG:OUTP:STATE?') # ist AFG an
    # oder aus? für Wiederherstellung am Ende
```

```
    conn.write('AFG:OUTP:STATE OFF') # AFG aus
    time.sleep(3) # Signal soll nicht mehr anliegen
    scale before = conn.query('{}:SCA?'.format(channel)) # vertical
    # scale speichern
    conn.write('{}:SCA 0.01'.format(channel)) # kleinstes Auflösung
    # einstellen für opt. Kontrolle
    conn.write('{}:PRO:AUTOZ EXEC'.format(channel)) # Autozero
    # ausführen
    r = conn.query('*opc?') # auf das Ende der Ausführung warten
    time.sleep(3) # Pause
    conn.write('{}:SCA {}'.format (channel, scale_before))
    conn.write('AFG:OUTP:STATE {}'.format(status_AFG)) # AFG auf
    # ursprl. Zustand

# Anzahl Datenpunkte für eine Periode erfassen
def get datapoints per period(conn, channel):
    conn.write('DATa:source {}'.format(channel)) # Auswahl Kanal
    horizontal_record_length = \
    float(conn.query('HORizontal:RECOrdlength?')) # Anzahl Datenpunkte
    horizontal_scale = float(conn.query('HORizontal:SCAle?'))
    # Horizontaler Teiler in s/div
    entire time = horizontal scale * 10 # 10 Teiler (divs) vorhanden
    # Berechnung der Datenpunkte pro Periode
    datapoints per period = \
    int(horizontal record length / (entire_time * freq))
    return datapoints_per_period

# data transfer
def get_waveform_data_one_period(conn, channel, datapoints):
    conn.write('DATa:ENCdg SRIBINARY') # [2-246] Datenformat einstellen
    conn.write('WFMOutpre:byt n 2') # 2 byte per sample,
    # 65536 datapoints [2-246]
    conn.write('DATa:source {}'.format(channel)) # channel
    conn.write('DATa:STARt 0') # first sample
    conn.write('DATa:STOP {}'.format(datapoints)) # last sample
    wave_data = \
    conn.query binary values('CURVe?', datatype='h', container=np.array)
    # binäre Daten speichern
    r = conn.query('*opc?')
    # Konvertierung von String zu float notwendig
    vertical_scale = float(conn.query('WFMOutpre:YMUlt?')) # V oder A pro
    # digit // vertical_scale:6400 bei 2 bit [2-806]
    vertical offset = float(conn.query('WFMOutpre:YZEro?')) # Offset in
    # A oder V
    vertical_position = float(conn.query('WFMOutpre:YOFf?')) # gibt
    # Positions-Offset in div zurück
    wave_data = np.array(wave_data, dtype='double') # Konvertierung
    # zu double
    wave data = \
    (wave_data - vertical_position) * vertical_scale + vertical_offset
    # Umrechnung in V / A
    # Zeitskala
    time_start = float(conn.query('WFMOutpre:XZEro?'))
    time_scale = float(conn.query('WFMOutpre:XINcr?'))
    time stop = time_start + time_scale * datapoints
    time_data = \
    np.linspace(time_start, time_stop, datapoints, endpoint=False)
    # speichern als file
    if(SAVE_DATA):
        dt = datetime.now()
        file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
```

```
        np.savetxt('D:/thonny/data/{} {} wave data raw {}.txt'.format \
        (COMPONENT, channel, file_time), wave_data, delimiter=',')
        np.savetxt('D:/thonny/data/{} {} time data raw {}.txt'.format \
        (COMPONENT, channel, file_time), time_data, delimiter=',')
    # Übergabe der Parameter
    return time_data, wave_data

# Daten integrieren
def get integrated data(time data, wave data):
    # Integration der Daten nach Trapez-Methode
    integrated_data = integrate.cumulative_trapezoid \
    (wave data, time data, initial=0)
    # cycle mean als Offset anwenden, damit BH-Kurve mittig
    # angeordnet ist
    int offset = np.mean(integrated data)
    integrated data = integrated_data - int_offset
    # Daten speichern
    if(SAVE_DATA):
        dt = datetime.now()
        file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
        np.savetxt('D:/thonny/data/{}_wave_data_integrated_{}.txt'.format
\
        (COMPONENT, file_time), integrated_data, delimiter=',')
    return time_data, integrated_data

# FFT - Filter
def apply_fft_filter(time_data, wave_data, frequency):
    cutoff frequency = CUTOFF_FREQUENCY * frequency # Grenzfrequenz
    # berechnen
    wf_fft = fft(wave_data) # Transformation in den Frequenzbereich
    # Abtastfrequenz berechnen
    N = np.size(time_data) # Anzahl Stützstellen / Datenpunkte
    Ta = (time_data[100] - time_data[0])/100 # Abtastzeit
    # gemittelt aus 100 Datenpunkten
    fk = fftfreq(N,Ta) # fk ordnet der U_fft-X-Achse die
    # Frequenzen zu
    # Signal im Frequenzbereich filtern, np.abs(fk)
    # Betragsbildung aus Re- und Im-Teil
    if(USE_CUTOFF_FREQ):
        F g = wf fft * (np.abs(fk) < cutoff frequency)
        # (np.abs(fk) < fg) ergibt True(1) oder False(0)
    else:
        F g = wf fft
    F_g[0] = 0 # DC-Anteil auf 0 setzen
    # Rücktransformation in den Zeitbereich
    wf g = ifft(F g)
    # Daten speichern
    if(SAVE_DATA):
        dt = datetime.now()
        file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
        np.savetxt('D:/thonny/data/{}_wave_data_filtered_{}.txt'.format \
        (COMPONENT, file time), wf g.real, delimiter=',')
    # Rückgabe der Zeitdaten (unverändert) und des Realteils im
    # Zeitbereich (Imaginärteil = 0)
    return time_data, wf_g.real

def calculate_BH_values(BH, waveform):
    if(BH == 'B'):
        # 1E09 für Umrechnung von mm^2 in m^2 und von T in mT
        waveform = waveform * 1E09 / (AREA * N_SEC)
        return waveform
```

```
    if(BH == 'H'):
        # 1000 für Umrechnung mm in m
        waveform = waveform * N_PRIM * 1000 / (LG+LM)
        return waveform
    return 0

#-----------------------------------------
# Hauptprogramm, ruft oft Unterprogramme auf
# während des Tests ist das Speichern und Wiederherstellen
# deaktiviert
mdo34 = connect()
if(mdo34 == 0):
    print('Verbindungsprobleme zum Oszilloskop')
else:
    # AFG anschalten
    freq = input('Bitte Frequenz in kHz eingeben: ')
    freq = float(freq) * 1000 # Faktor 1000 da Übermittlung in Hz
    vpp = input('Bitte Spannung nach Verstärker in Vpp eingeben: ')
    vpp = float(vpp)
    vpp = vpp/SA_RATIO
    set_AFG(mdo34, 'SINE', freq, 0.0, vpp)
    # AutoSet aufrufen
    autoset(mdo34)
    # Anzahl der Datenpunkte für eine Periode erfassen
    datapoints one period = get_datapoints_per_period(mdo34, CH_U_SEC)
    # I/U-Daten abrufen
    set_ACQ_to_single(mdo34)
    # Stromdaten, Spannungsdaten
    curr_td_raw, curr_wf_raw = get_waveform_data_one_period(mdo34, \
    CH_CURRENT, datapoints_one_period)
    volt td raw, volt wf raw = get waveform_data_one_period(mdo34, \
    CH_U_SEC, datapoints_one_period)

    # Rohdaten Spannungs-Integration für Vergleich
    # (enthalten Rauschen)
    intU_td_raw, intU_wf_raw = get_integrated_data(volt_td_raw, \
    volt_wf_raw)

    # Daten filtern
    curr td fil, curr wf_fil = apply_fft_filter(curr_td_raw, \
    curr_wf_raw, freq)
    volt_td_fil, volt_wf_fil = apply_fft_filter(volt_td_raw, \
    volt wf raw, freq)
    # gefilterte Daten Integration
    intU_td_fil, intU_wf_fil = get_integrated_data(volt_td_fil, \
    volt_wf_fil)

    # für BH-Kennlinie Werte umrechnen
    if(SHOW BH):
        curr_wf_raw = calculate_BH_values('H', curr_wf_raw)
        intU_wf_raw = calculate_BH_values('B', intU_wf_raw)
        curr wf fil = calculate BH values('H', curr wf fil)
        intU_wf_fil = calculate_BH_values('B', intU_wf_fil)

    # Daten plotten
    plt.figure(figsize=(10,6))
    if(SHOW_RAW_DATA):
        plt.plot(curr wf raw, intU_wf_raw, color='lightgray')
    if(SHOW_FILTERED_DATA):
        plt.plot(curr_wf_fil, intU_wf_fil, color='teal')
    if(SHOW_BH):
```

```
        plt.xlabel('H in [A/m]')
        plt.ylabel('B in [mT]')
        plt.title('BH-Kurve / {} / {}kHz / {}V'.format \
        (COMPONENT, freq/1000, vpp*SA_RATIO))
    else:
        plt.xlabel('I in [A]')
        plt.ylabel(r'$\int$U in [Vs]')
        plt.title('I-Integral_U / {} / {}kHz / {}V'.format \
        (COMPONENT, freq/1000, vpp*SA_RATIO))
    plt.grid(True)
    plt.tight_layout()
    plt.show()


    disconnect(mdo34)
print('\nEnde\n')
```

Anlagen, Teil 3

In Anlage 3 ist der Programmcode der graphischen Oberfläche hinterlegt.

Programmcode „Pruefplatz.py“

```
# Importiere Matplotlib-Bibliotheken
import matplotlib as mpl
import matplotlib.backends.backend_qt5agg
import matplotlib.figure

# Importiere die notwendigen Elemente für die GUI
import PyQt5
import PyQt5.uic
import PyQt5.QtWidgets

# Importiere sonstige Module
import pyvisa
import time
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from scipy.fft import fft,ifft,fftfreq # für FFT

# Variablen
# Adresse für connect()
ADDRESS_MDO34 = 'TCPIP0::192.168.178.20::inst0::INSTR'
# Alias für Kanäle
CH_AFG = 'CH1'
CH_U_PRIM = 'CH2'
CH_CURRENT = 'CH3'
CH_U_SEC = 'CH4'
# Verstärkungsfaktor Signalverstärker
SA_RATIO = 10
# für Datentransfer
HORIZONTAL_DATAPOINTS = 10000

# Definiere eine Klasse, die von QMainWindow abgeleitet wird
class MainWindow(PyQt5.QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__() # QMainWindow initialisieren
        PyQt5.uic.loadUi(r'D:\thonny\Pruefplatz.ui', self) # User
        # Interface laden (aus .ui-Datei)
        self.fig = mpl.figure.Figure() # ein figure-Objekt erzeugen
        mpl.backends.backend_qt5agg.FigureCanvasQTAgg(self.fig) # Qt
        # Zeichenfläche erzeugen
        self.fig.set_tight_layout(True) # tight layout passt Plot in
        # vorhandene Fläche ein
        self.groupBox_5.layout().addWidget(self.fig.canvas) # Zeichen-
```

```
        # fläche (canvas) in GUI einfügen

        # Slots einrichten
        self.pushButton_1.clicked.connect(self.Connect)
        self.pushButton_2.clicked.connect(self.Disconnect)
        self.checkBox_5.stateChanged.connect(self.OnNoise)
        self.checkBox_6.stateChanged.connect(self.OnSaveData)
        self.checkBox_7.stateChanged.connect(self.OnShowBH)
        self.pushButton_3.clicked.connect(self.getScreenshot)
        self.pushButton_4.clicked.connect(self.startMeasurement)

    # Funktionen
    def Connect(self):
        # versuche Verbindungsaufbau
        try:
            # rm gibt den Speicherort des Visa-Treibers zurück
            rm = pyvisa.ResourceManager()
            # eine Verbindung aufbauen und zurückgeben
            global conn
            conn = rm.open_resource(ADDRESS_MDO34)
        # wenn Oszilloskop offline ist
        except:
            self.statusbar.showMessage \
            ('Verbindung konnte nicht hergestellt werden.', 3000)
            return
        # wenn Verbindungsaufbau möglich war
        else:
            # Voreinstellungen, z.B. längeres Timeout, setzen
            conn.timeout = 10000
            conn.encoding = 'latin_1'
            conn.read_termination = '\n'
            conn.write_termination = None
            conn.write('HEAD 0')
            # Oszilloskop Einstellungen auf internen
            # Speicher Nr.10 sichern
            # wenn CheckBox aktiviert ist
            if(self.checkBox_1.isChecked()==True):
                conn.write('*SAV 10')
                r = conn.query('*OPC?')
            # Oszilloskop auf Werkseinstellungen setzen
            # auf Beenden warten
            conn.write('*RST')
            r = conn.query('*OPC?')
            # GUI Buttons freigeben / Sperren
            self.pushButton_2.setEnabled(True)
            # "Trennen" aktivieren
            self.pushButton_1.setEnabled(False)
            # "Verbinden" ausgrauen
            self.groupBox_2.setEnabled(True)
            # Feld Screenshot aktivieren
            self.groupBox_3.setEnabled(True)
            # Feld Messung aktivieren
            self.checkBox_6.setChecked(False)
            # Daten speichern deaktivieren
            self.checkBox_7.setChecked(False)
            # BH-Kennlinie anzeigen deaktivieren

    def Disconnect(self):
        # Einstellungen sichern, wenn aktiviert
        if(self.checkBox_1.isChecked()==True):
            conn.write('*RCL 10')
```

```
            conn.query('*OPC?')
        conn.close() # Verbidnung trennen
        # Buttons sperren / freigeben
        self.pushButton_2.setEnabled(False)
        # "Trennen" ausgrauen
        self.pushButton_1.setEnabled(True)
        # "Verbinden" aktivieren
        self.groupBox_2.setEnabled(False)
        # Feld Screenshot deaktivieren
        self.groupBox_3.setEnabled(False)
        # Feld Messung deaktivieren

    def OnNoise(self):
        # wenn "Rauschen entfernen" aktiviert wird, Felder aktivieren
        if(self.checkBox_5.isChecked()==True):
            self.label_1.setEnabled(True)
            self.spinBox_1.setEnabled(True)
        # wenn "Rauschen entfernen" deaktiviert wird, Felder ausgrauen
        else:
            self.label_1.setEnabled(False)
            self.spinBox_1.setEnabled(False)

    def OnSaveData(self):
        # wenn "Daten speichern" aktiviert wird, Felder aktivieren
        if(self.checkBox_6.isChecked()==True):
            self.label_2.setEnabled(True)
            self.lineEdit_1.setEnabled(True)
        # wenn "Daten speichern" deaktiviert wird, Felder ausgrauen
        else:
            self.label_2.setEnabled(False)
            self.lineEdit_1.setEnabled(False)

    def OnShowBH(self):
        # wenn "BH Kennlinie anzeigen" aktiviert wird, Felder aktivieren
        if(self.checkBox_7.isChecked()==True):
            self.label_3.setEnabled(True)
            self.label_4.setEnabled(True)
            self.label_5.setEnabled(True)
            self.label_6.setEnabled(True)
            self.label_7.setEnabled(True)
            self.doubleSpinBox_1.setEnabled(True)
            self.doubleSpinBox_2.setEnabled(True)
            self.doubleSpinBox_3.setEnabled(True)
            self.spinBox_2.setEnabled(True)
            self.spinBox_3.setEnabled(True)
        # wenn "BH Kennlinie anzeige" deaktiviert wird, Felder ausgrauen
        else:
            self.label_3.setEnabled(False)
            self.label_4.setEnabled(False)
            self.label_5.setEnabled(False)
            self.label_6.setEnabled(False)
            self.label_7.setEnabled(False)
            self.doubleSpinBox_1.setEnabled(False)
            self.doubleSpinBox_2.setEnabled(False)
            self.doubleSpinBox_3.setEnabled(False)
            self.spinBox_2.setEnabled(False)
            self.spinBox_3.setEnabled(False)

    # Screenshot speichern
    def getScreenshot(self):
        fileSaveLocation = r'D:\thonny\pics\\' # Speicherort für das Bild
```

```
        conn.write('SAVe:IMAGe:FILEFormat PNG') # Format angeben
        conn.write('SAVe:IMAGe:INKSaver OFF') # OFF = Bild ohne
        # Invertierung der Farben
        conn.write('HARDCopy STARt') # Bereitstellen der Daten
        imgData = conn.read_raw() # Rohdaten in Variable speichern
        dt = datetime.now() # aktuelle Zeit + Datum speichern
        fileName = dt.strftime("Screenshot MDO34  %Y%m%d %H%M%S.png")
        # einen Dateiname erstellen mit Einbindung der Zeit/des Datums
        imgFile = open(fileSaveLocation + fileName, "wb") # die
        # Bilddatei mit Schreibrechten öffnen
        imgFile.write(imgData) # die Rohdaten in die Datei speichern
        imgFile.close() # die Bilddatei schließen

    def startMeasurement(self):
        # importierte Funktionen
        # AFG anschalten
        def set_AFG():
            frequency = self.doubleSpinBox_4.value() # Frequenz aus
            # DoubleSpinBox
            frequency = 1000 * frequency # Umrechnung in kHz
            amplitude = self.doubleSpinBox_5.value() # Spannung aus
            # DoubleSpinBox
            amplitude = amplitude/SA_RATIO # Umrechnung mit
            # Verstärkungsfaktor Signalverstärker
            conn.write('AFG:OUTP:LOA:IMPED HIGHZ') # HighZ da
            # höhere Spannung mgl.
            conn.write('AFG:FUNC {}'.format('SINE')) # Waveform
            conn.write('AFG:FREQ {}'.format(frequency)) # Frequenz in Hz
            conn.write('AFG:OFFS {}'.format('0.0')) # Offset in V
            conn.write('AFG:AMPL {}'.format(amplitude)) # Amplitude in V
            conn.write('AFG:OUTP:STATE ON') # AFG an

        # AutoSet
        def autoset():
            # nur Kanal 1-4 darstellen
            conn.write('SEL:CH1 ON')
            conn.write('SEL:CH2 ON')
            conn.write('SEL:CH3 ON')
            conn.write('SEL:CH4 ON')
            conn.write('SEL:REF1 OFF')
            conn.write('SEL:REF2 OFF')
            conn.write('SEL:REF3 OFF')
            conn.write('SEL:REF4 OFF')
            conn.write('SEL:MATH OFF')
            # auf AC-Kopplung schalten, Stromzange (Kanal 3)
            # kann nur DC-gekoppelt werden
            conn.write('CH1:COUPling AC')
            conn.write('CH2:COUPling AC')
            conn.write('CH4:COUPling AC')
            # horizontale Auflösung, sonst Probleme beim Datentransfer
            conn.write('HORizontal:RECOrdlength {}'.format \
            (HORIZONTAL DATAPOINTS))
            # Stromzange AutoZero, wenn gewünscht
            if(self.checkBox_2.isChecked()==True):
                autozero current probe()
            # die AutoSet Prozedur + warten auf Ende
            conn.write('AUTOS EXEC')
            r = conn.query('*OPC?')
            # für eine saubere Darstellung Average mit 64 Samples
            # einstellen
            conn.write('ACQ:MOD AVE')
```

```
        conn.write('ACQ:NUMAV 64')
        time.sleep(1) # 1 Sekunde warten damit 64 Erfassungen
        # vorhanden sind
        # AutoSet setzt auf volle Bandbreite, BW auf 20Mhz
        # einstellen:
        conn.write('CH1:BANDWIDTH 20E6')
        conn.write('CH2:BANDWIDTH 20E6')
        conn.write('CH3:BANDWIDTH 20E6')
        conn.write('CH4:BANDWIDTH 20E6')
        # Anpassung der Amplitude des Strom-Kanals (bei
        # niedrigen Aplituden manchmal auf 1A)
        # wenn vertikale Skalierung auf 1 eingestellt ist,
        # dann evtl. Anpassung div nötig
        vertical_scale = float(conn.query('{}:SCAle?'.format \
        (CH CURRENT)))
        # Peak2Peak über 2 div
        if(vertical_scale == 1.0):
            conn.write('MEASU:IMM:SOU {}'.format(CH_CURRENT))
            # Auswahl Kanal
            conn.write('MEASU:IMM:TYP PK2Pk') # Peak2Peak messen
            curr_p2p = float(conn.query('MEASU:IMM:VAL?'))
            # wenn peak2peak kleiner 50mA ist, dann 10mA - div
            # einstellen
            if(curr p2p < 0.2):
                conn.write('{}:SCA 0.01'.format(CH CURRENT))
        # Setzen der horizontalen Einstellung auf ca. 2 Perioden
        horizontal_scale = conn.query('HOR:SCA?') # Autoset zeigt
        # ca 4 Perioden an
        horizontal_scale = float(horizontal_scale) # String zu Float
        # konvertieren
        horizontal_scale = horizontal_scale/2 # Wert halbieren - ca 2
        # Perioden
        conn.write('HOR:SCA {}'.format(horizontal_scale))

    # Stromzange AutoZero
    def autozero_current_probe():
        status AFG = conn.query('AFG:OUTP:STATE?') # ist AFG an oder
        # aus? für Wiederherstellung am Ende
        conn.write('AFG:OUTP:STATE OFF') # AFG aus
        time.sleep(3) # Signal soll nicht mehr anliegen
        scale_before = conn.query('{}:SCA?'.format(CH_CURRENT))
        # vertical scale speichern
        conn.write('{}:SCA 0.01'.format(CH CURRENT)) # kleinstes
        # Auflösung einstellen für opt. Kontrolle
        conn.write('{}:PRO:AUTOZ EXEC'.format(CH_CURRENT)) # Autozero
        # ausführen
        r = conn.query('*opc?') # auf das Ende der Ausführung warten
        time.sleep(3) # Pause
        conn.write('{}:SCA {}'.format (CH CURRENT, scale before))
        conn.write('AFG:OUTP:STATE {}'.format(status_AFG)) # AFG
        # auf ursprl. Zustand

    # Anzahl Datenpunkte für eine Periode erfassen
    def get_datapoints_per_period():
        conn.write('DATa:source {}'.format(CH_U_SEC)) # Auswahl Kanal
        horizontal_record_length = float \
        (conn.query('HORizontal:RECOrdlength?')) # Anzahl Datenpunkte
        horizontal scale = float(conn.query('HORizontal:SCAle?'))
        # Horizontaler Teiler in s/div
        entire_time = horizontal_scale * 10
        # 10 Teiler (divs) vorhanden
```

```
        # Berechnung der Datenpunkte pro Periode
        frequency = self.doubleSpinBox_4.value() # Frequenz aus
        # DoubleSpinBox
        frequency = frequency * 1000 # Umrechnung kHz in Hz
        datapoints_per_period = \
        int(horizontal record length / (entire_time * frequency))
        return datapoints_per_period

    # ACQ auf Stop (Einzelmessung) schalten
    # und eine neue Messung starten
    def set_ACQ_to_single():
        conn.write('ACQ:STOPA SEQ') # auf Single (Stop) setzen
        conn.write('ACQ:STATE ON') # einzelnes Ereignis aufnehmen
        r = conn.query('*OPC?')

    # data transfer
    def get_waveform_data_one_period(channel, datapoints):
        conn.write('DATa:ENCdg SRIBINARY') # [2-246] Datenformat
        # einstellen
        conn.write('WFMOutpre:byt_n 2') # 2 byte per sample,
        # 65536 datapoints [2-246]
        conn.write('DATa:source {}'.format(channel)) # channel
        conn.write('DATa:STARt 0') # first sample
        conn.write('DATa:STOP {}'.format(datapoints)) # last sample
        wave data = conn.query binary values \
        ('CURVe?', datatype='h', container=np.array)
        # binäre Daten speichern
        r = conn.query('*opc?')
        # Konvertierung von String zu float notwendig
        vertical_scale = float(conn.query('WFMOutpre:YMUlt?'))
        # V oder A pro digit
        # vertical_scale:6400 bei 2 bit [2-806]
        vertical_offset = float(conn.query('WFMOutpre:YZEro?'))
        # Offset in A oder V
        vertical_position = float(conn.query('WFMOutpre:YOFf?'))
        # gibt Positions-Offset in div zurück
        wave data = np.array(wave_data, dtype='double')
        # Konvertierung zu double
        wave_data = (wave_data - vertical_position) * \
        vertical scale + vertical_offset
        # Umrechnung in V / A
        # Zeitskala
        time start = float(conn.query('WFMOutpre:XZEro?'))
        time_scale = float(conn.query('WFMOutpre:XINcr?'))
        time_stop = time_start + time_scale * datapoints
        time data = \
        np.linspace(time_start, time_stop, datapoints, \
        endpoint=False)
        # speichern als file
        if(self.checkBox_6.isChecked()==True):
            component = self.lineEdit_1.text()
            dt = datetime.now()
            file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
            np.savetxt \
            ('D:/thonny/data/{} {} wave data raw {}.txt' \
            .format(component, channel, file_time), \
            wave_data, delimiter=',')
            np.savetxt \
            ('D:/thonny/data/{}_{}_time_data_raw_{}.txt' \
            .format(component, channel, file_time), \
            time_data, delimiter=',')
```

```
        # Übergabe der Parameter
        return time_data, wave_data

    # Daten integrieren
    def get_integrated_data(time_data, wave_data, name):
        # Integration der Daten nach Trapez-Methode
        integrated data = \
        integrate.cumulative_trapezoid \
        (wave data, time data, initial=0)
        # cycle mean als Offset anwenden, damit BH-Kurve mittig
        # angeordnet ist
        int offset = np.mean(integrated data)
        integrated data = integrated_data - int_offset
        # Daten speichern
        if(self.checkBox 6.isChecked()==True):
            component = self.lineEdit_1.text()
            dt = datetime.now()
            file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
            np.savetxt('D:/thonny/data/{}_CH4_wave_data \
            _{}_integrated_{}.txt' \
            .format(component, name, file_time), \
            integrated data, delimiter=',')
        return time_data, integrated_data

    # FFT - Filter
    def apply_fft_filter(time_data, wave_data, name):
        frequency = self.doubleSpinBox_4.value()
        # Frequenz aus DoubleSpinBox
        frequency = 1000 * frequency # Umrechnung in kHz
        cutoff_frequency = self.spinBox_1.value()
        # Faktor der Grenzfrequenz ggü.
        # Grundfrequenz
        cutoff_frequency = cutoff_frequency * frequency
        # Grenzfrequenz berechnen
        wf_fft = fft(wave_data) # Transformation in den
        # Frequenzbereich
        # Abtastfrequenz berechnen
        N = np.size(time_data)
        # Anzahl Stützstellen / Datenpunkte
        Ta = (time data[100] - time data[0])/100
        # Abtastzeit gemittelt aus 100 Datenpunkten
        fk = fftfreq(N,Ta) # fk ordnet der U_fft-X-Achse
        # die Frequenzen zu
        # Signal im Frequenzbereich filtern, np.abs(fk)
        # Betragsbildung aus
        # Re- und Im-Teil
        if(self.checkBox_5.isChecked()==True):
            F_g = wf_fft * (np.abs(fk) < cutoff_frequency)
            # (np.abs(fk) < fg) ergibt True(1) oder False(0)
        else:
            F_g = wf_fft
        F g[0] = 0 # DC-Anteil auf 0 setzen
        # Rücktransformation in den Zeitbereich
        wf_g = ifft(F_g)
        # Daten speichern
        if(self.checkBox_6.isChecked()==True):
            component = self.lineEdit_1.text()
            dt = datetime.now()
            file_time = dt.strftime("%Y%m%d_%H%M%S.%f")
            np.savetxt('D:/thonny/data/{}_{}_wave_data_ \
            filtered_{}.txt'.format \
```

```
            (component, name, file time), wf g.real, delimiter=',')
        # Rückgabe der Zeitdaten (unverändert) und des
        # Realteils im Zeitbereich
        # (Imaginärteil = 0)
        return time_data, wf_g.real

    def calculate BH values(BH, waveform):
        if(BH == 'B'):
            area = self.doubleSpinBox 3.value()
            n sec = self.spinBox 3.value()
            # 1E09 für Umrechnung von mm^2 in m^2 und von T in mT
            waveform = waveform * 1E09 / (area * n_sec)
            return waveform
        if(BH == 'H'):
            n prim = self.spinBox 2.value()
            lg = self.doubleSpinBox 2.value()
            lm = self.doubleSpinBox_1.value()
            # 1000 für Umrechnung mm in m
            waveform = waveform * n_prim * 1000 / (lg+lm)
            return waveform
        return 0

    # Aufruf der Funktionen
    set AFG() # Frequenzgenerator starten
    autoset() # AutoSet
    datapoints_one_period = get_datapoints_per_period()
    # Datenpunkte pro Periode auslesen
    set ACQ to single() # auf Einzelmessung schalten
    # Rohdaten Strom und Spannung erhalten
    curr_td_raw, curr_wf_raw = get_waveform_data_one_period \
    (CH CURRENT, datapoints one period)
    volt_td_raw, volt_wf_raw = get_waveform_data_one_period \
    (CH_U_SEC, datapoints_one_period)
    # Rohdaten Spannungs-Integration für Vergleich
    # (enthalten Rauschen)
    intU_td_raw, intU_wf_raw = \
    get integrated data(volt_td_raw, volt_wf_raw, "raw")
    # Daten filtern
    curr_td_fil, curr_wf_fil = \
    apply fft filter(curr td raw, curr_wf_raw, 'CH3')
    volt_td_fil, volt_wf_fil = \
    apply_fft_filter(volt_td_raw, volt_wf_raw, 'CH4')
    # gefilterte Daten Spannungs-Integration
    intU_td_fil, intU_wf_fil = \
    get_integrated_data(volt_td_fil, volt_wf_fil, 'filtered')
    # für BH-Kennlinie Werte umrechnen
    if(self.checkBox_7.isChecked()==True):
        curr_wf_raw = calculate_BH_values('H', curr_wf_raw)
        intU wf raw = calculate BH values('B', intU wf raw)
        curr_wf_fil = calculate_BH_values('H', curr_wf_fil)
        intU_wf_fil = calculate_BH_values('B', intU_wf_fil)
    # Plotten der Daten
    self.fig.clear()
    self.ax = self.fig.add_subplot(1, 1, 1)
    # wenn "Rohdaten anzeigen" aktiviert ist
    if(self.checkBox_3.isChecked()==True):
        self.plot, = 1\
        self.ax.plot(curr wf raw, intU wf raw, color='gray')
    # wenn "gefilterte Daten anzeigen" aktiviert ist
    if(self.checkBox_4.isChecked()==True):
        self.plot, = \
```

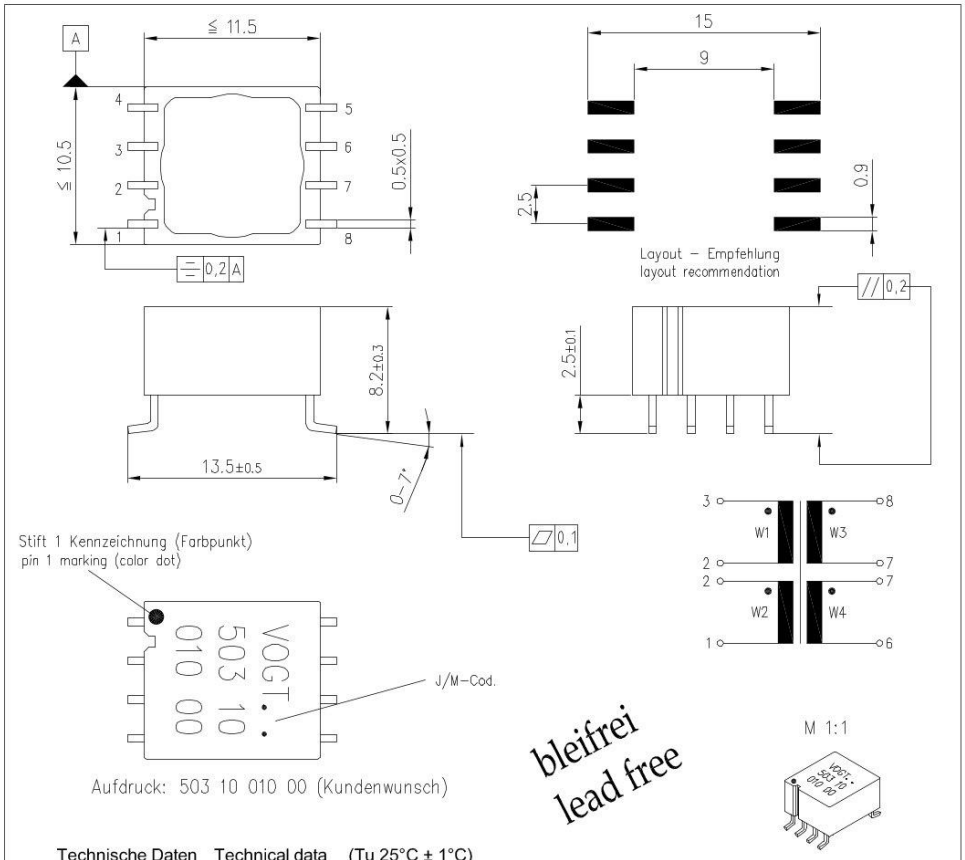
```
            self.ax.plot(curr_wf_fil, intU_wf_fil, color='teal')
        # wenn "BH-Kennlinie anzeigen" aktiviert ist
        if(self.checkBox_7.isChecked()==True):
            self.ax.set_xlabel('H [A/m]')
            self.ax.set_ylabel('B [mT]')
            self.ax.set_title('BH-Kennlinie')
        else:
            self.ax.set_xlabel('I [A]')
            self.ax.set_ylabel(r'$\int$U [Vs]')
            self.ax.set_title('I-$\int$U-Kennlinie')
        # Gitter anzeigen
        self.ax.grid()
        # Plot zeichnen
        self.fig.canvas.draw()

# Erzeuge eine QApplication und das Hauptfenster.
app = PyQt5.QtWidgets.QApplication([])
window = MainWindow()

# Zeige das Fenster und starte die QApplication.
window.show()
app.exec_()
```

Anlagen, Teil 4

Datenblatt Übertrager Vogt_24_4



Technische Daten Technical data (Tu 25°C ± 1°C)

Gewicht weight 1,7 g	Kern Core R 5,84x3,05x2,5	Material material Fi 360 / equivalent	Luftspalt air gap 0 mm
Betriebstemperatur Operating temperature −40°C...+125°C		Lagertemperatur Storage temperature −30°C...+70°C	Klimakategorie climatic category IEC 68-1 40/125/56
Arbeitsbereich Operating range	$U_{Bmin.}$= $U_{Bmax.}$=	Maximale Leistung Maximum output	Frequenz Frequency

Aufbau nach Norm
Design according to standard

L	W1 + W2	1,5 mH ±50%	f = 10 kHz, U_0 ≦ 50 mV
R_{DC}	W1+W2	370 mΩ ±15%	
U_P	W1/W2 – W3/W4	1,5 kV_{eff}	f = 50 Hz, t = 2 s
ü	W1+W2 : W3+W4	1 : 1,33	

VOGT electronic

Benennung description: DC/DC Converter K10

Index	Änderung changes	Datum	Name

Projektionsmethode 1	Werkstoff material	Maßstab scale 3:1
CAD-Zeichnung! Manuelle Änderung nicht zulässig.		
Dimensions in mm		

Gez.	29.03.06	HÖLLMÜLLER
Gepr.	29.03.06	WANDL C.
Gepr.	31.05.06	KORNEXL H.
Status: Freigegeben		

Teile-Nr. part no.: 503 10 912 00

NL 060 327 21 01 (06)

Blatt 01 | V0 | Freimaßtol. DIN

Technische Daten Technical data

(Tu 25°C ± 1°C)

	Wicklung / Stifte winding / pins	Messwert value	Toleranz tolerance	Messschärfe Inspection level	Messbedingungen / Geräte test conditions / devices
L_N	W1 + W2	1,5 mH	± 50 %	100%	f = 10 kHz U_0 ≦ 50 mV
L_N					
L_N					
L_{Streu} $L_{leakage}$					
Q					
Z					
R_{HF}					
R_{DC}	W1 + W2 W3 + W4	370 mΩ 510 mΩ	± 15 %	Typ	
Leerlaufübersetzung und Verpolung turns ratio	W1 W2 W3 W4 15 15 20 20			100%	
Kappel – C coupling –capacity					
THD					
Hochspannung HV	W1/W2 – W3/W4	1,5 kV_{eff}		100 %	50 Hz 2 sec.

VOGT electronic

Benennung description: test parameter

Index	Änderung changes	Datum	Name

	Datum	Name
Gez.	15.05.06	FASSER A.
Gepr.	15.05.06	BAUER F.
Gepr.	31.05.06	KORNEXL H.
Status:	Freigegeben	

Teile-Nr. part no.: 503 10 912 00

Blatt 04 | VO | Freimaßtol. DIN | (06)

Projektionsmethode 1 | Werkstoff material | Maßstab scale | CAD-Zeichnung! Manuelle Änderung nicht zulässig. | Dimensions in mm

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Chemnitz, den 02. August 2022

Heiko Paal