HAW HAMBURG

Bachelorarbeit

Niklas Guntelmann

Effiziente Anregeschaltung für die Elektrochemische Impedanzspektroskopie in großen Fahrzeugbatterien

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Niklas Guntelmann

Effiziente Anregeschaltung für die Elektrochemische Impedanzspektroskopie in großen Fahrzeugbatterien

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Elektro- und Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Pawel Buczek

Eingereicht am: 30. August 2024

Niklas Guntelmann

Thema der Arbeit

Effiziente Anregeschaltung für die Elektrochemische Impedanzspektroskopie in großen Fahrzeugbatterien

Stichworte

Elektrochemische Impedanzspektroskopie, effiziente Anregeschaltung

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines Laboraufbaus, der ein Konzept einer effizienten Anregeschaltung für die Elektrochemische Impedanzspektroskopie in großen Fahrzeugbatterien demonstriert. Dazu wird eine Schaltung entworfen und die nötigen Komponenten beschafft. Die Steuerung der Schaltung wird über einen Mikrocontroller realisiert, für den entsprechende Software programmiert wird. Abschließend soll die Funktionalität der Anregeschaltung mit Messungen erprobt werden.

Niklas Guntelmann

Title of Thesis

Efficient excitation circuit for the electrochemical impedance spectroscopy in large vehicle batteries

Keywords

Electrochemical impedance spectroscopy, efficient excitation circuit

Abstract

This thesis presents the development of a laboratory setup that demonstrates the concept of an efficient excitation circuit for electrochemical impedance spectroscopy in large vehicle batteries. A circuit is designed, components are procured, and a microcontroller is programmed to control the circuit. Finally, the functionality of the excitation circuit is tested through measurements

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Zeit der Anfertigung dieser Arbeit unterstützt haben.

So ermöglichte mir Herr Prof. Karl-Ragmar Riemschneider, diese Arbeit im Rahmen seines Forschungsprojekts zu schreiben. Weiterhin möchte ich mich bei dem Team des Projekts bedanken. So konnten sie mir bei Problemen stets Unterstützung anbieten und meine Fragen beantworten. Insbesondere Tobias Frahm, Dr. Florian Rittweger und Günter Müller möchte ich hier erwähnen. Auch meinen Eltern Maike und Marco Guntelmann, die mich während meines gesamten Studiums und auch dieser Arbeit unterstützt haben, möchte ich danken.

Denn nur mit dieser Unterstützung und Hilfe konnte diese Arbeit erstellt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BP bipolar

EIS Elektrochemische Impedanzspektroskopie

IDLE Ruhezustand

LR links nach rechts

MP monopolar

PySerial PySerial Modul

PyVISA PyVISA Modul

RL rechts nach links

Schaltmodul MOSFET-Schaltmodul HW-532

TMR TMR 6-2412WI der Firma Traco Power

1 Einleitung

Die Elektromobilität ist aktuell ein stark wachsender Sektor, indem die Forderung nach mehr Leistung und Reichweite der Fahrzeuge aufkommt. Dies macht die Entwicklung immer größerer und leistungsfähigerer Batterien notwendig. Dabei kommt auch eine hohe Anzahl neuer Probleme auf. So müssen Fahrzeugbatterien eine lange Lebensdauer von zehn Jahren besitzen und dabei ausreichend Ladezyklen absolvieren können, wobei auch eine ausreichende Leistung bei extremen Temperaturen und die Sicherheit der Batterie bei Unfällen zu gewährleisten ist [1]. Um diese Ziele zu erreichen, ist unter anderem ein Batterie Managment System (BMS) zur Regelung erforderlich.

”Modern battery management systems need accurate information concerning the state of charge (SoC)(...), the state of health (SoH)(...), the battery lifetime, the internal temperature and fault diagnosis within cell packs”[2, S. 8 Kanoun 2018]. Mit dieser Aussage ist gemeint, dass ein BMS viele und genaue Messdaten benötigt. Dabei stellt die sogenannte Elektrochemische Impedanzspektroskopie (EIS) eine mögliche Messmethode dar, mit der Gesundheits- und Ladezustand einer Batterie ermittelt werden kann, ohne sie dabei zu beschädigen [2]. Dafür wird die Batterie mit verschieden Frequenzen angeregt und ihre Impedanz ermittelt.

In modernen Fahrzeugen sind jedoch die Innenwiederstände der Batterie sehr klein, so sind Werte von unter 1 mΩ keine Seltenheit [3]. Dies führt zu Problemen, da dementsprechend die Spannungsantwort bei einer Stromanregung ebenfalls niedrig ausfällt. Somit werden hochauflösende ADC benötigt oder der Anregestrom muss gesteigert werden. Diese Steigerung zöge aber auch einen erhöhten Leistungsbedarf des Anregesystems mit sich, was beim Ausblick auf Fahrzeuge mit Batterie Systemen mit 800 V [4] zu nötigen Leistungen im kW-Bereich führen würde. So müssen große Leistungen entweder extern zugeführt werden oder aber Verbraucher die Anregung über eine Entladung durchführen, wobei die Energie verloren geht.

Um dieses Problem zu umgehen, wird in dieser Arbeit ein Ansatz für eine effiziente Anregeschaltung für die EIS-Messung in großen Fahrzeugbatterien verfolgt. Dabei sollen bei niedrigem Leistungsbedarf hohe Ströme möglich sein. Der Grundgedanke der Schaltung ist dabei die Teilung der Batterie in Teilbatterien, wobei eine angeregt wird und die Leistung dafür von der anderen bereitgestellt wird. Der Wechsel zwischen Senke und Quelle soll dabei abwechselnd erfolgen. Dieses Konzept soll nun mit einem Laboraufbau auf seine Wirksamkeit geprüft werden. Dazu werden für die Teilbatterien je drei in Serie geschaltete Einzelzellen verwendet. Für den Laboraufbau muss eine Platine entworfen, die passenden Bauteile ausgewählt und eine Steuerung mit einem Mikrocontroller realisiert werden. Der Aufbau muss dann auf seine Funktion geprüft werden und anschließend Messungen durchgeführt werden. Es wird geprüft, ob mit diesem Ansatz eine EIS-Messung realisiert werden kann. Dabei soll insbesondere die Effizienz der Schaltung überprüft werden.

2 Theorie

Um die Messergebnisse aus Versuchen mit der Anregeschaltung bewerten zu können, soll hier kurz auf die Durchführung und das Nyquist-Diagramm einer EIS mit einem handelsüblichen Impedanz-Spektrum-Analyzer eingegangen werden. Um eine EIS durchzuführen, wird die Batterie über einen bestimmten Frequenzbereich angeregt und die komplexe Impedanz aufgezeichnet [2]. Dabei müssen auch der Ladezustand und die Belastung der Batterie beachtet werden [5].

Wenn die Messung durchgeführt wurde, wird meist ein Nyquist Diagramm verwendet, um die Messwerte darzustellen. Dabei wird der Imaginärteil auf eine umgekehrte Y-Achse und der Realteil auf der X-Achse aufgetragen. In Abbildung 2.1 ist ein solches Diagramm einmal idealisiert aufgeführt.

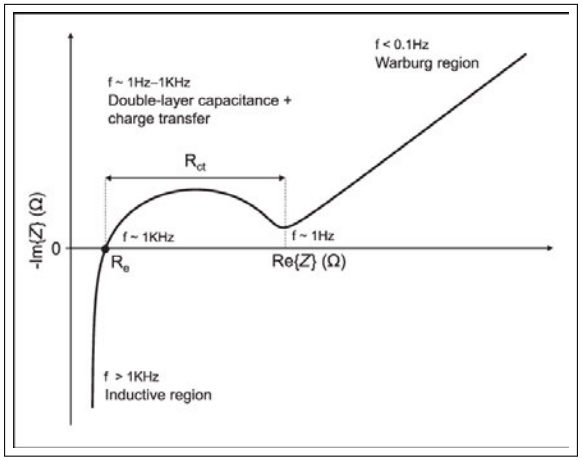


Abbildung 2.1: Idealisiertes Nyquist-Diagramm für die EIS-Messung einer Batterie aus [6, S. 3]

Der untere Frequenzbereich von 100 mHz bis 1 Hz wird dabei von Diffusionseffekten dominiert [2]. Er wird deswegen auch als Diffusionsast bezeichnet. Im Bereich von 1 Hz bis etwa 1 kHz treten dann vor allem Reaktionen der Elektroden auf, wobei dieser Teil des Nyquist-Diagramms halbkreisförmig ist.

Der Frequenzbereich über 1 kHz wird dann von induktiven Effekten beherrscht, was in einem schnellen Anstieg des Imaginärteils bei geringer Änderung des Realteils resultiert. Der Schnittpunkt der Kurve mit der X-Achse stellt dabei den Serienwiderstand R_S der Batterie dar [2].

3 Hardware

3.1 Grundkonzept der Anregeschaltung

Das Grundkonzept der Schaltung ist in Abbildung 3.1 dargestellt. So wird jede Batterie über einen Gleichspannungswandler angeregt. Die Versorgung des Wandlers übernimmt dabei die jeweils andere Batterie. Dabei werden die Batterien nie gleichzeitig angeregt.

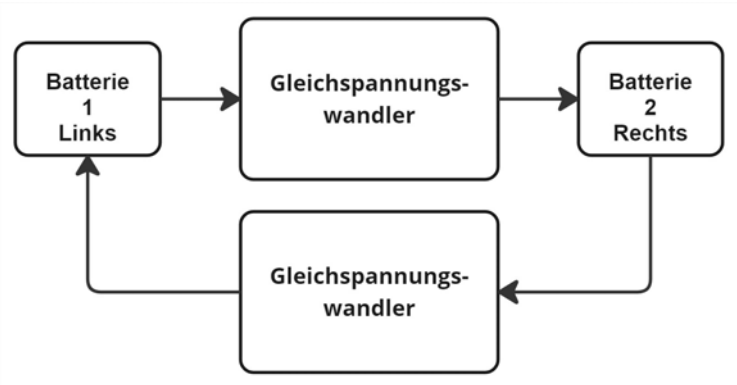


Abbildung 3.1: Grundkonzept der Anregeschaltung

3.2 Anforderungen

Beim Entwurf müssen einige Anforderungen berücksichtigt werden. So müssen die zwei Batterien galvanisch voneinander getrennt sein, um Fehler bei der Messung zu vermeiden. Auch eine Potenzialtrennung zur Steuerung über den XMC 1100 Mikrocontroller ist bei Batteriespannungen von zukünftig bis zu 800 V [4] eine sinnvolle Forderung an die Anregeschaltung.

Es sollte auch eine einfache Möglichkeit gegeben sein, eine externe Messung über die Platine durchzuführen, um so Vergleiche mit separaten EIS-Messgeräten durchzuführen. Weiterhin sollten ausreichend Messabgriffe für Messungen mit dem Oszilloskop verfügbar sein.

Ebenfalls im Platinendesign berücksichtigt werden soll die Möglichkeit, mit einem Magnetfeldsensor den Stromfluss zu messen. So könnte der Leistungsverlust, im Vergleich zur Messung über einen Messwiderstand, reduziert werden.

Als Batterie sollen auf jeder Seite drei in Reihe geschaltete Lithium-Ionen-Zellen, im Formfaktor 21700, verwendet werden. Die Messströme sollten dabei aufgrund der kleinen Innenwiederstände der verwendeten Batteriezellen vom Typ Molicel P45B, von wenigen mΩ [7], möglichst groß sein.

Auch die Pinbelegung des Mikrocontrollers spielt eine Rolle. So sollten, um die Programmierung der Software zu vereinfachen, alle Steuerpins auf dem gleichen Port liegen.

Für den Entwurf spielt auch noch eine Rolle, dass vier Betriebsmodi vorgesehen sind. Dabei fließt der Strom im ersten Modus von links nach rechts (LR), im zweiten von rechts nach links (RL) und im dritten abwechselnd zuerst in die eine und dann in die andere Richtung. Der dritte Modus wird deswegen auch als bipolar (BP) bezeichnet. Die ersten beiden Modi sind hingegen monopolar (MP). Es gibt auch einen Modus, in dem die Schaltung deaktiviert ist, welcher auch als Ruhezustand (IDLE) bezeichnet wird.

Durchgeführt wurden die Komponenten-Auswahl und der Platinenentwurf von Mitarbeitern der Forschungsgruppe.

3.3 Verwendete Komponenten

Bevor mit dem Entwurf der Platine begonnen werden kann, müssen noch die wichtigsten Komponenten ausgewählt werden. Dabei soll neben Widerständen, Kondensatoren und LED insbesondere auf die Gleichspannungswandler, die Schaltmodule und den steuernden Mikrocontroller eingegangen werden.

3.3.1 Gleichspannungswandler Traco Power TMR 6-2412WI

Nach Recherche und Voruntersuchungen wurde der TMR 6-2412WI der Firma Traco Power (TMR) als Gleichspannungswandler gewählt. Bei diesem handelt es sich um einen Sperrwandler mit einer Ausgangsspannung, laut Datenblatt [8], von 12 V und einem maximalen Ausgangsstrom von 0,5 A. Diese Werte sollten für die verwendeten Batterien mit einer kombinierten nominalen Spannung von 10,8 V ausreichend sein [7]. Auch die im Datenblatt angegebene hohe Effizienz von 87 % und der breite Eingangsspannungsbereich von 9 bis 36 V spricht für den TMR [8]. Für die weitere Entwicklung relevant ist auch die Zeit, die vergeht bis eine stabile Ausgangsspannung bereitsteht. Diese wird im Datenblatt mit 30 ms angegeben [8], was bei Messungen bestätigt werden konnte. Wichtig für das Platinendesign ist auch noch, dass der TMR sieben Pins besitzt. Je zwei Pins als Ein- und Ausgänge, zwei weitere Pins ohne Funktion und zuletzt ein Pin zur Steuerung. Dieser schaltet den TMR bei einer Spannung größer als 3 V ab und unterhalb von 0,5 V an [8]. Wenn der Pin also nicht beschaltet wird, ist der TMR standardmäßig eingeschaltet.

Die galvanische Trennung wird im TMR durch den Aufbau als Sperrwandler erreicht. Die Funktionsweise ist dabei in Abbildung 3.2 dargestellt.

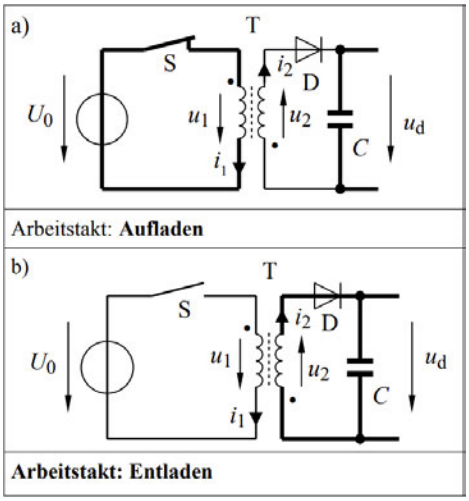


Abbildung 3.2: Funktionsweise eines Sperrwandlers aus [9, S. 359]

Im Sperrwandler werden zwei magnetisch gekoppelte Spulen, die den Speichertransformator T bilden, zur Energieübertragung genutzt. Dabei wird zuerst die Primärseite aufgeladen, indem der Schalttransistor S aktiviert wird. So entspricht die Spannung u_1 an der primären Wicklung von T, dann der Eingangsspannung U_0, wodurch der Strom i_1 sie auflädt.

Im nächsten Schritt wird S dann deaktiviert, worauf der Strom i_2 fließt und die Ausgangsspannung u_d der Spannung u_2 an der sekundären Wicklung von T entspricht. Um jetzt aus der wechselnden Spannung u_2 eine Gleichspannung am Ausgang zu erzeugen, wird der Kondensator C benötigt. Dieser glättet u_d, indem er Strom bereitstellt, während T aufgeladen wird. Dabei führen längere Einschaltzeiten von S zu einer höheren Spannung u_d. Die Diode D verhindert dabei einen Rückfluss des Stroms aus C nach T. Der Sperrwandler ist bei angeschlossener Last nicht leerlauffest, das heißt, er muss aktiv geregelt werden [9].

3.3.2 MOSFET-Schaltmodul HW-532

Zur Erzeugung eines Anregesignals ist es notwendig, dass auf der Platine Schaltvorgänge durchgeführt werden können. Die Schalter sollten dazu in der Lage sein, Frequenzen von bis zu 10 kHz schalten zu können. Auch eine Potenzialtrennung von Mikrocontroller und Anregeschaltung muss von ihnen realisiert werden. Ausgewählt wurde schließlich ein einfach verfügbares MOSFET-Schaltmodul HW-532 (Schaltmodul), das auf dem Optokoppler PC817A [10] und dem NMOS-MOSFET AOD4184A [11] basiert. Ein durch Untersuchungen entwickelter Schaltplan des Moduls ist in Abbildung 3.3 aufgeführt.

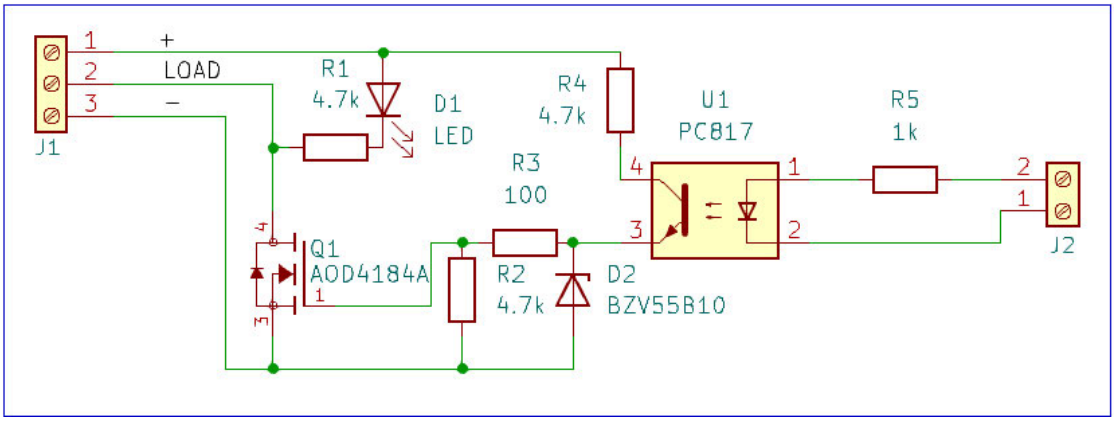


Abbildung 3.3: Schaltplan des MOSFET-Schaltmoduls

Der mit + bezeichnete Pin wird mit der Betriebsspannung, der zu schaltenden Schaltung, verbunden, und der mit − bezeichnete mit ihrer Masse. Der mit *LOAD* bezeichnete Pin wird schließlich in den Massezweig des zu kontrollierenden Bauteils eingebunden. Die Pins eins und zwei auf der rechten Seite werden mit einem Pin bzw. der Masse des Mikrocontrollers verbunden.

So ist die galvanische Trennung sichergestellt, da keine direkte Verbindung zwischen beiden Seiten besteht. Wenn der Mikrocontroller seinen Pin einschaltet, der an Pin 2 von J angeschlossen ist, wird der Optokoppler eingeschaltet und ein Strom fließt von seinem Kollektor (Pin 4) zum Emitter (Pin 3), was ein positives Potenzial am Gate des Mosfets erzeugt. Dadurch kann Strom vom Drain (Pin 4) des Mosfets zu seiner Source (Pin 3) fließen, der Schalter ist also geschlossen. Wenn am Optokoppler hingegen kein Signal anliegt, sind sowohl Optokoppler als auch MOSFET hochohmig, der Schalter also offen.

3.3.3 Mikrocontroller-Board XMC2GO

Beim zur Steuerung verwendeten Mikrocontroller handelt es sich um den Typ XMC 1100 der Firma Infineon. Dieser wird für die Anregeschaltung auf einem XMC2GO genannten Entwicklungsboard eingesetzt, das mit Programmier-IC und Spannungsregler ausgerüstet ist. Die Verwendung eines Entwicklungsboards verringert den Designaufwand, da so kein separater Mikrocontroller mit zusätzlichen Komponenten eingeplant werden muss.

Ein schematischer Aufbau des verwendeten Entwicklungsboards ist in Abbildung 3.4 dargestellt.

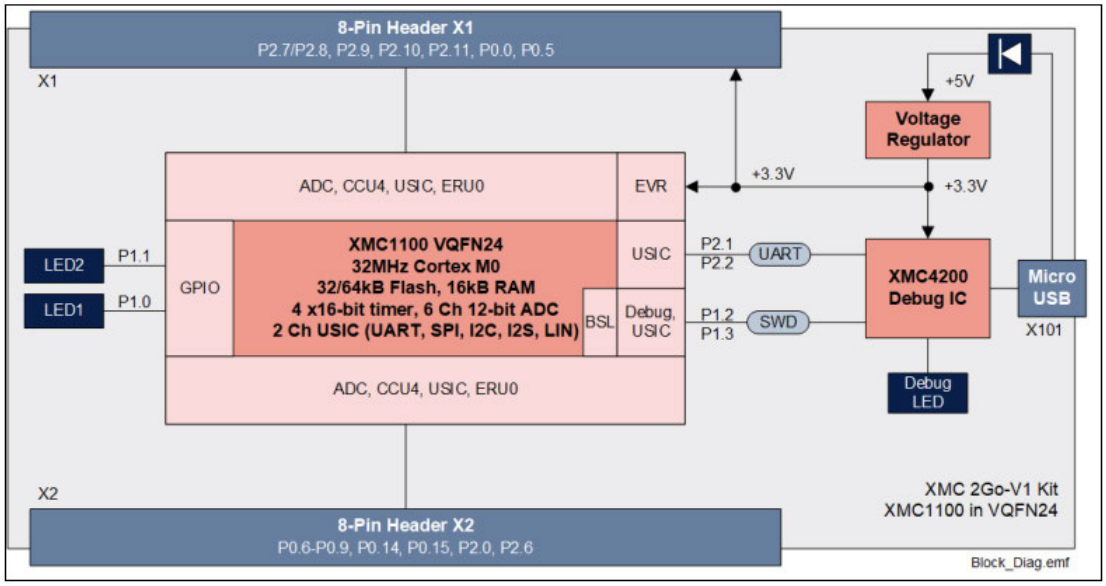


Abbildung 3.4: Schema des Mikrocontroller-Boards aus [12, S. 4]

Aus Abbildung 3.4 kann auch entnommen werden, welche Pins von welchen Ports auf dem Pin-Header des Entwicklungsboards liegen. Da nur ein Port zur Steuerung verwendet werden soll, sollte möglichst Port Null verwendet werden. Dieser weist nämlich mit zehn die größte Anzahl an Pins auf. Dementsprechend wird beim Entwurf der Platine diesen Pins die Steuerung der Schaltmodule zugewiesen.

3.4 Entwurf der Anregeschaltung

Nach der Auswahl der Komponenten kann mit dem Entwurf der Anregeschaltung und dem Zeichnen des Schaltplans begonnen werden.

3.4.1 Schaltungskonzepts der Schaltmodule

Bei der Erstellung eines Schaltungskonzepts für die Schaltmodule müssen zuerst Überlegungen angestellt werden, welche Bauteile überhaupt wo benötigt werden. Da ein Anregesignal erzeugt werden soll, müssen die Ausgänge der TMR auf irgendeine Weise kontrolliert werden. Ein Problem stellen hierbei die 30 ms dar, die der TMR benötigt, bis sein Ausgang einen stabilen Zustand annimmt. Da diese Zeit sowohl nach Anlegen der Spannung am Eingang, als auch beim Einschalten über den Steuerpin abgewartet werden muss, können höhere Anregefrequenzen so nicht erzeugt werden. Aus dieser Tatsache resultiert also, dass ein Anregesignal am besten über das Schalten nach dem TMR-Ausgang generiert wird. Dementsprechend werden zwei Schaltmodule zwischen den Massen der Batterien und den TMR-Ausgängen platziert.

Theoretisch würden die Schaltmodule an den Ausgängen der TMR ausreichen für die Erzeugung eines Anregesignals. Da so aber beide TMR dauerhaft Strom verbrauchen, was wenig effizient ist, werden ihre Eingänge ebenfalls mit Schaltmodulen versehen.

Die vier platzierten Schaltmodule sollten eigentlich für alle Fälle ausreichend sein, da die Steuerpins der TMR nicht beschaltet werden müssen. Wenn die Steuerpins nicht beschaltet werden, sind die TMR bei Anliegen einer Spannung standardmäßig eingeschaltet. Um alle Möglichkeiten offenzuhalten, wurden im ersten Entwurf trotzdem sowohl Schaltmodule zur Masse als auch Pull-up-Widerstände an den Steuerpins platziert.

Wenn die Verschaltung aller Schaltmodule zusammengefasst wird, kann folgendes vereinfachtes Schaltbild gezeichnet werden (Abbildung 3.5). Hier sind auch bereits die Positionen der Messwiderstände R1 und R2 sowie der Pull-up-Widerstände R15 und R25 an den Steuerpins eingezeichnet.

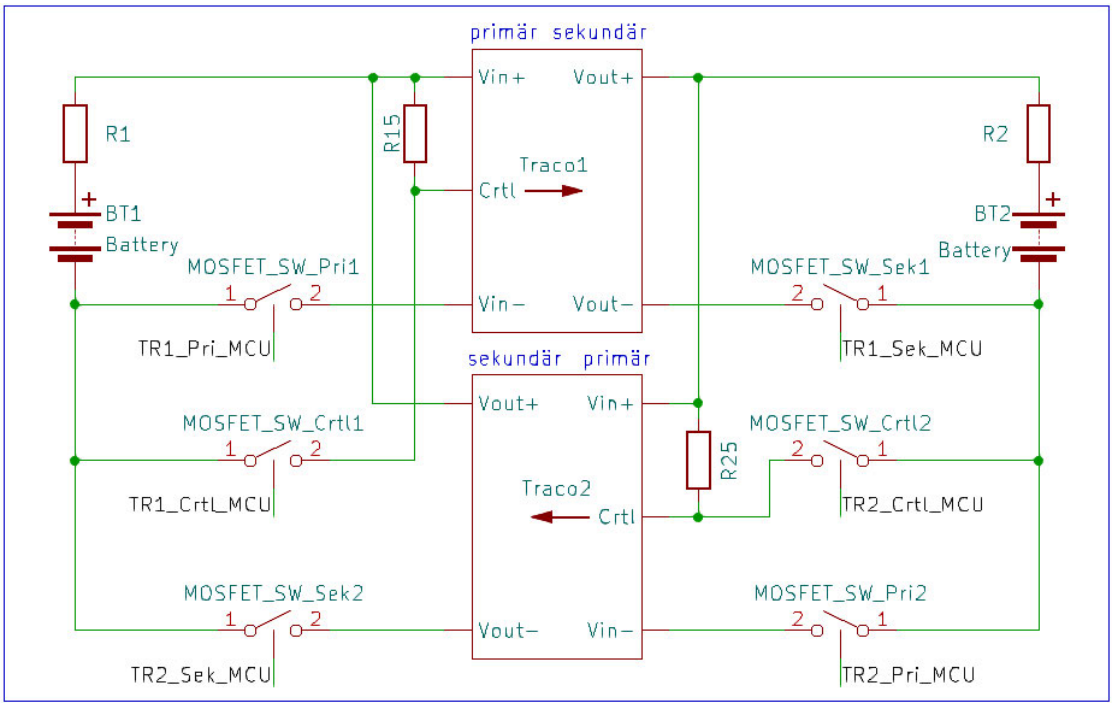


Abbildung 3.5: Schaltbild mit der Anordnung der Schaltmodule

3.4.2 Anschluss der Schalter an den Mikrocontroller

Die Schaltmodule müssen mit dem Mikrocontroller verbunden werden. Dabei muss darauf geachtet werden, freie Pins aus Port Null zu verwenden. Zusätzlich werden an die Pins auch noch LEDs mit Vorwiderständen angeschlossen, um den Schaltzustand der Schaltung zu visualisieren. In Abbildung 3.6 ist das Schaltbild zum Anschluss an den Mikrocontroller angegeben.

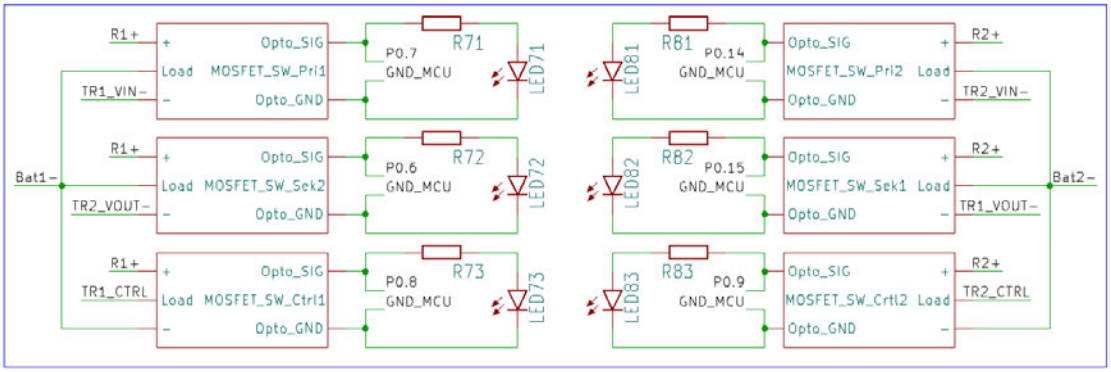


Abbildung 3.6: Schaltbild für den Anschluss der Schaltmodule an den Mikrocontroller

Aus der Verknüpfung der Schalter mit dem Mikrocontroller resultiert, dass das Schalten von Pins am Mikrocontroller eine bestimmte Reaktion der Schaltung hervorruft. In Tabelle 3.1 sind die Mikrocontroller-Pins, die von ihnen kontrollierten Schaltmodule und die Reaktion der Schaltung aufgeführt.

Tabelle 3.1: Durch Schalten der Mikrocontroller Pins betätigte Schalter

Geschalteter Pin	Betätigtes Schaltmodul	Reaktion der Schaltung
P0.6	SW_Sek2	Ausgang TMR 2 einschalten
P0.7	SW_Pri1	Eingang TMR 1 einschalten
P0.8	SW_Crtl1	Steuerpin TMR 1 einschalten
P0.9	SW_Crlt2	Steuerpin TMR 2 einschalten
P0.14	SW_Pri2	Eingang TMR 2 einschalten
P0.15	SW_Sek1	Ausgang TMR 1 einschalten

3.4.3 Zusätzliche Bauteile an den Ein- und Ausgängen der TMR

Um die Ein- und Ausgangsspannungen der TMRs zu glätten und eine Begrenzung der Ströme zu ermöglichen, werden zusätzliche Widerstände und Kondensatoren benötigt. Dabei werden vier 100 nF Kondensatoren (C13, C14, C23 und C24) direkt an den Ein- und Ausgängen der TMR vorgesehen. Sie dienen dazu, elektrisches Rauschen zu reduzieren. Weiterhin werden auch Ein- (R13 und R24) und Ausgangswiderstände (R14 und R23) fest eingeplant, die der Strombegrenzung dienen. Die Widerstandswerte werden aber noch für Versuche offengelassen. Auch für die Ein- (C11 und C22) und Ausgangskondensatoren (C14 und C21) werden Experimente vorbereitet. So werden je drei alternative Positionen für unterschiedliche Baugrößen vorgesehen. Auch für die Messwiderstände an den Batterien (R1 und R2), werden je zwei mögliche Positionen eingeplant.

In Abbildung 3.7 ist das resultierende Schaltbild dargestellt.

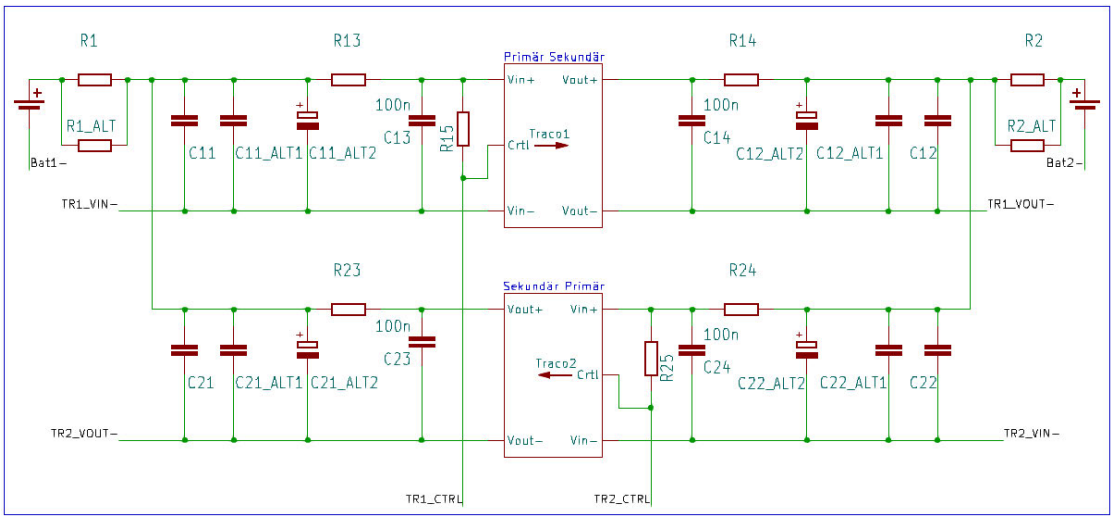


Abbildung 3.7: Schaltbild für die Beschaltung der Ein- und Ausgänge der Traco TMR

3.4.4 Kompletter Schaltplan

Schließlich ergibt sich, nachdem Messabgriffe, Anschlüsse für eine externe Anregung, Jumper zur Auswahl der Anregeart (extern oder über Mikrocontroller) und Pinleiste hinzugefügt wurden, folgender Schaltplan (Abbildung 3.8).

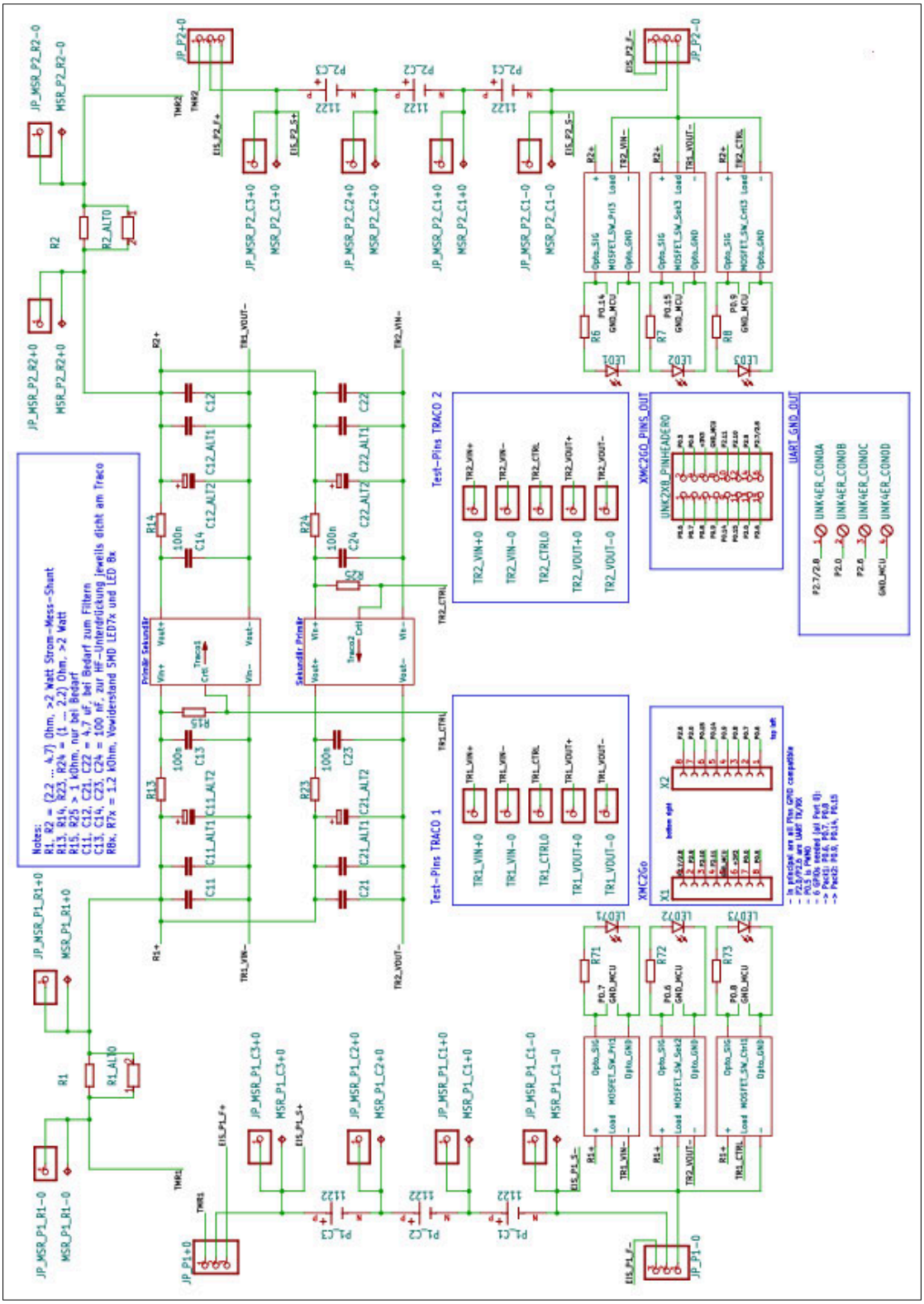
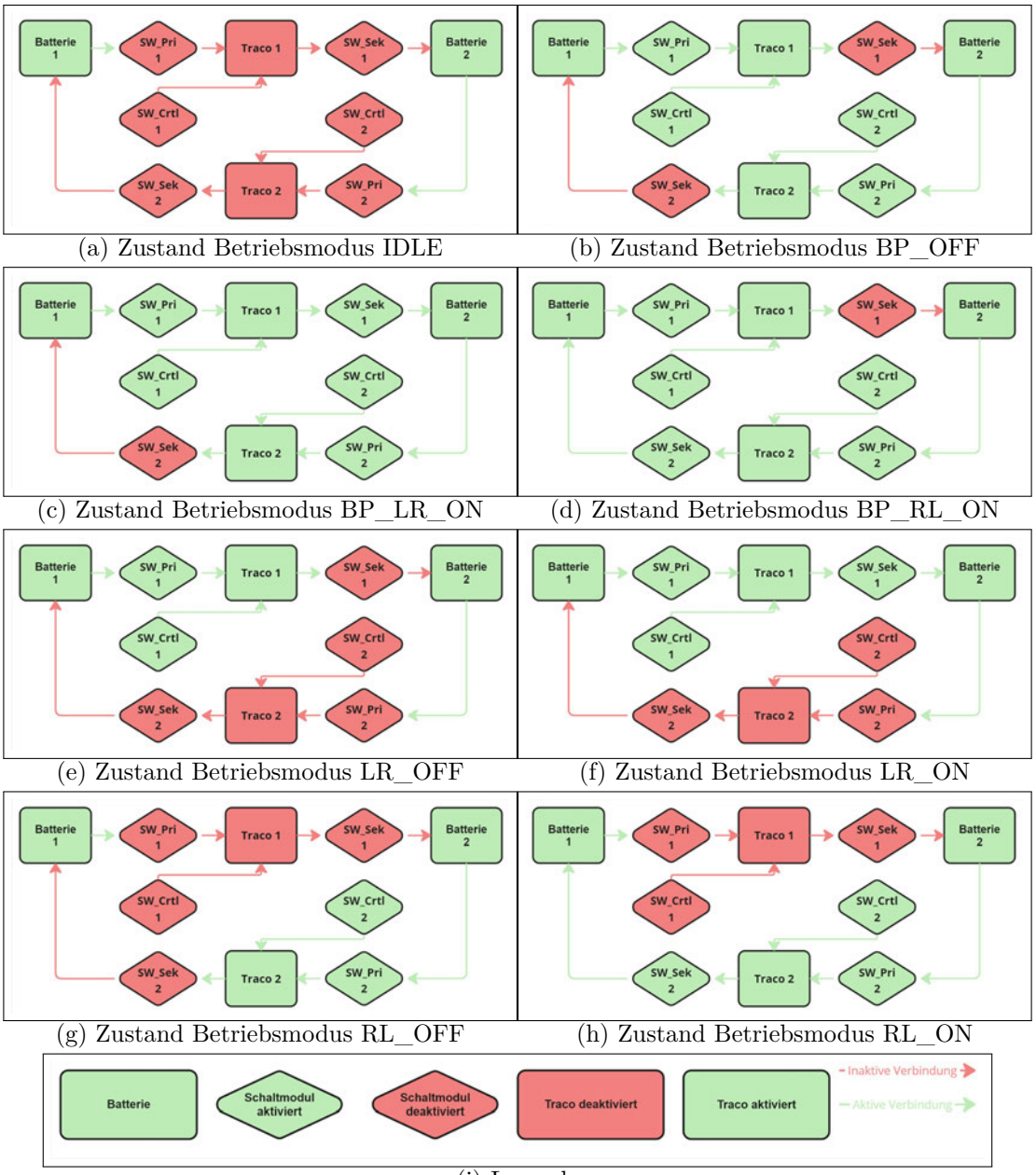


Abbildung 3.8: Vollständiger Schaltplan der Anregeschaltung

3.5 Schaltzustände

Damit die Anregung für jeden Betriebsmodus korrekt durchgeführt werden kann, müssen die Schaltzustände, die die Schaltung annimmt, definiert werden. In Abbildung 3.9 ist dabei eine Übersicht aller Schaltzustände gegeben.



(a) Zustand Betriebsmodus IDLE
(b) Zustand Betriebsmodus BP_OFF
(c) Zustand Betriebsmodus BP_LR_ON
(d) Zustand Betriebsmodus BP_RL_ON
(e) Zustand Betriebsmodus LR_OFF
(f) Zustand Betriebsmodus LR_ON
(g) Zustand Betriebsmodus RL_OFF
(h) Zustand Betriebsmodus RL_ON
(i) Legende

Abbildung 3.9: Schematische Darstellung aller Zustände der Schaltung

Der Betriebsmodus IDLE besitzt dabei einen Zustand, in dem alle Schalter deaktiviert sind (Abbildung 3.9a).

Im Betriebsmodus BP, indem die Anregung zwischen den Seiten wechselt, gibt es drei Zustände. Einen Zustand ohne Anregung BP_OFF(Abbildung 3.9b), wo die TMRs aktiviert werden. Die zwei weiteren Zustände BP_LR_ON (Abbildung 3.9c) und BP_RL_-OFF (Abbildung 3.9d), dienen dann dazu, die rechte bzw. linke Batterie anzuregen. Die Ausgänge der TMRs werden dazu jeweils so geschaltet, dass ein Strom von links nach rechts (LR) bzw. von rechts nach links (RL) fließen kann.

Bei den anderen beiden Betriebsmodi hingegen ist jeweils nur ein TMR in Betrieb. So ergeben sich die Ruhezustände LR_OFF (Abbildung 3.9e) für die Anregung von Batterie Zwei und RL_OFF (Abbildung 3.9g) für die Anregung von Batterie Eins. In diesen Zuständen wird der jeweils benötigte TMR mit Spannung versorgt. Um die Anregung dann zu starten, werden die Ausgänge der TMR dann mit den Batterien verbunden, sodass ein Strom von links nach rechts oder umgekehrt von rechts nach links fließt. Dieser Prozess ist in den Zuständen LR_ON (Abbildung 3.9f) und RL_ON (Abbildung 3.9h) ersichtlich.

4 Software

Im folgenden Kapitel soll näher auf die Überlegungen und Ansätze eingegangen werden, die bei der Erstellung der Software für den Mikrocontroller bedacht wurden. Zur Softwareentwicklung wurde die Sammlung von Bibliotheken und Tools der Infineon AG, ModusToolbox™ genannt, verwendet, die die Konfiguration des Mikrocontroller sowie Anwendungsentwicklung ermöglicht [13]. Die Tools der ModusToolbox™ ließen sich in den Texteditor Visual Studio Code integrieren. Unter Verwendung von Plugins, die z.B. das Debuggen von ARM Cortex-M Prozessoren ermöglichen, konnte Visual Studio Code die Funktionalität einer IDE erreichen und so der Entwicklungsprozess erleichtert werden [14].

4.1 Anforderungen

Aus der Aufgabenstellung, Gesprächen mit Mitarbeitern der Forschungsgruppe und Problemen während einiger Tests ergaben sich bestimmte Anforderungen an die Software auf dem Mikrocontroller. Eine Anregefrequenz im Bereich von 100 mHz bis 10 kHz soll einstellbar sein. Auch der Tastgrad des Rechtecksignals soll von 0 bis 100 % variierbar sein. Tastgrad und Frequenz sollten dabei eine nicht zu große Abweichung vom eingestellten Wert annehmen. Weiterhin muss die Möglichkeit bestehen, sowohl ein Anregesignal unbegrenzter Dauer als auch eines mit einer bestimmten Anzahl an Perioden zu erzeugen. Die Periodenzahl sollte dabei genau eingehalten werden. Ein Wechsel zwischen den vier Betriebsmodi LR, RL, BP und IDLE ist eine weitere erforderliche Funktionalität. Die Steuerung des Mikrocontroller soll über eine UART-Schnittstelle erfolgen und dementsprechend muss ein Format für Steuerbefehle festgelegt werden. Wichtig ist auch eine Absicherung gegen Störungen, sodass bei Problemen ein sicherer Zustand eingenommen wird.

4.2 Portzustände

Aus dem Platinendesign, dem verwendeten Mikrocontroller und den geforderten Betriebsmodi können verschiedene Portzustände abgeleitet werden, die der Ausgangsport annehmen muss. Grundlage dafür bildet dabei die Abbildung 3.9, in der alle notwendigen Schaltzustände abgebildet sind. Weiterhin wird auch Tabelle 3.1 benötigt, in der die Pins des Mikrocontrollers den von ihnen gesteuerten Schaltern zugeordnet sind. Die korrekte Reihenfolge der Portzustände muss ebenfalls eingehalten werden, um einen korrekten Ablauf zu gewährleisten. Dabei ergeben sich acht verschiedene Portzustände, analog zu den Schaltzuständen: je zwei für die Betriebsarten LR und RL, drei für den Modus BP und einer für den Betriebsmodus IDLE. Die Portzustände, das geforderte Verhalten der Schaltung und die dafür aktivierten Pins sind dabei in Tabelle 4.1 dargestellt.

Tabelle 4.1: Zustände des Ausgangsports, Verhalten der Schaltung und aktivierte Mikrocontroller-Pins

Name des Portzustands	Verhalten der Schaltung	Aktivierte Mikrocontroller-Pins
MODE_IDLE_OUT	Alle Funktionen ausgeschaltet	Kein Pin aktiviert
MODE_LR_OFF_OUT	Versorgung für TMR 1 aktiv / Ausgänge abgeschaltet	P0.7, P0.8
MODE_LR_ON_OUT	Versorgung für TMR 1 aktiv / Ausgang TMR 1 aktiv	P0.7, P0.8, P0.15
MODE_RL_OFF_OUT	Versorgung für TMR 2 aktiv / Ausgänge abgeschaltet	P0.9, P0.14
MODE_RL_ON_OUT	Versorgung für TMR 2 aktiv / Ausgang des TMR 2 aktiv	P0.6, P0.9 P0.14
MODE_BP_OFF_OUT	Versorgung beider TMR aktiv / Ausgänge abgeschaltet	P0.7, P0.8, P0.9, P0.14
MODE_BP_LR_ON_OUT	Versorgung beider TMR aktiv / Ausgang TMR 1 aktiv	P0.7, P0.8, P0.9, P0.14 P0.15
MODE_BP_RL_ON_OUT	Versorgung beider TMR aktiv / Ausgang TMR 2 aktiv	P0.6, P0.7, P0.8, P0.9 P0.14

Um das geforderte Verhalten der Schaltung zu erreichen, müssen die Pins des Ausgangsports korrekt geschaltet werden. Dazu werden den Portzuständen die Ausgangsbitmuster für das Output-Modification-Register zugeordnet. Näheres dazu kann aus [15, S. 738] entnommen werden.

Die Portzustände sollen in einer bestimmten Abfolge angesteuert werden. In den Betriebsmodi LR und RL wird zunächst mit dem Portzustand OFF_OUT begonnen und anschließend in den Portzustand ON_OUT gewechselt. Dieser Prozess wird nun so lange wie erforderlich wiederholt. Ein wenig anders verhält es sich in der Betriebsart BP. Hier wird auch mit dem Portzustand OFF_OUT begonnen und dann in den Portzustand RL_ON_OUT gewechselt, um nach einem erneuten Portzustand OFF_OUT im Portzustand LR_ON_OUT zu enden. In der Betriebsart IDLE liegt permanent der Portzustand IDEL_OUT vor, es findet also kein Portzustandswechsel statt.

Um einen einfachen Zugriff auf die Portzustände zu ermöglichen, wird eine Zustandsmatrix angelegt, in der jeder Betriebsmodus von einer Zeile und jeder Zustandszählvariablenwert von einer Spalte abgebildet werden. Eine Vereinfachung des Programmieraufwands wird erreicht, indem jedem Betriebsmodus vier Zustände zugeordnet werden, denen wiederum die erforderlichen Portzustände zugeordnet werden. So muss keine komplizierte Unterscheidung des Betriebsmodus beim Zustandswechsel durchgeführt werden, um den korrekten Portzustand zu erreichen. Dieses Vorgehen erfordert jedoch auch, dass der beschriebene Ablauf der Betriebsmodi LR und RL einmal wiederholt wird, um auf die geforderten vier Zustände zu kommen. Im Betriebsmodus IDLE hingegen wird einfach viermal der Portzustand IDLE_OUT wiederholt. In Tabelle 4.2 ist der Ablauf der Portzustände nach Betriebsart aufgeführt.

Tabelle 4.2: Ablauf der Portzustände in den verschiedenen Betriebsmodi

	Modus LR	Modus RL	Modus BP	Modus IDLE
Zustand 0	LR_-OFF_OUT	RL_-OFF_OUT	BP_-OFF_OUT	IDLE_-OUT
Zustand 1	LR_ON_-OUT	RL_ON_-OUT	BP_RL_-ON_OUT	IDLE_-OUT
Zustand 2	LR_-OFF_OUT	RL_-OFF_OUT	BP_-OFF_OUT	IDLE_-OUT
Zustand 3	LR_ON_-OUT	RL_ON_-OUT	BP_LR_-ON_OUT	IDLE_-OUT

4.3 Konfiguration des Timers für die Signalerzeugung

Grundsätzlich werden beide Timer im Vergleichsmodus betrieben. Dabei zählen sie hoch, bis ein eingestellter Periodenwert erreicht wird.

4.3.1 Formeln zur Bestimmung des Perioden- und Vergleichswerts

Zunächst müssen für die korrekte Konfiguration des Frequenz-Timers Formeln aufgestellt werden, um die Werte zu berechnen.

$$T = \frac{1}{f} \tag{4.1}$$

$$T = \frac{1}{f \cdot 2} \tag{4.2}$$

Formel 4.1 wird verwendet, um die Frequenz in die zu zählende Periodendauer umzuwandeln.

Wenn der Modus BP gewählt ist, muss die Frequenz verdoppelt werden. Das ist nötig, da in diesem Modus vier statt zwei Zustandswechsel für eine vollständige Periode notwendig sind. In der Grafik 4.1 ist dieser Zusammenhang dargestellt. Dementsprechend wird Formel 4.2 verwendet.

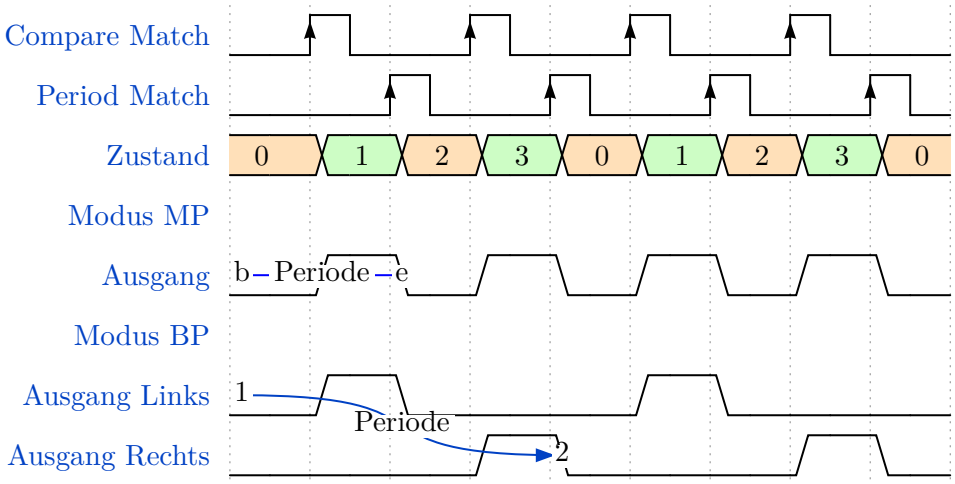


Abbildung 4.1: Vergleich der Ausgangssignale der Betriebsmodi bei gleichen Einstellungen des Timers

Um nun den Wert zu ermitteln, der in das Periodenregister des Mikrocontrollers geschrieben wird, müssen weitere Formeln aufgestellt werden.

$$t_{TimerAufloesung} = \frac{1}{f_{tclk}} \tag{4.3}$$

$$Periodenwert = \frac{T}{t_{Timertick}} - 1 \tag{4.4}$$

Die Zeit, die vergeht, bis der Timer inkrementiert wird, wird auch Timer-Auflösung oder $t_{TimerAufloesung}$ genannt. Sie ergibt sich, indem der Kehrwert vom Timertakt gebildet wird (Formel 4.3). Nun muss Formel 4.4 angewendet werden, um den Periodenwert zu erhalten. Da der Zähler ab Null startet, muss der Periodenwert um eins verringert werden.

Um nun den Vergleichswert, der den Einschaltzeitpunkt und damit den Tastgrad festlegt, korrekt einzustellen, wird Formel 4.5 benötigt.

$$Vergleichswert = Periodenwert \cdot \frac{100 - Tastgrad\ in\ \%}{100} \tag{4.5}$$

Mit Formel 4.5 kann nun der Vergleichswert in Abhängigkeit zum Periodenwert ermittelt werden.

Nachdem der Timer korrekt eingestellt ist, wird er gestartet und löst Interrupts aus, sobald der Vergleichswert oder der Periodenwert erreicht werden.

4.3.2 Problem durch den verwendeten Mikrocontroller

Durch den verwendeten Mikrocontroller ergibt sich allerdings ein Problem. So besitzt er nur 16-Bit-Timer, die die maximal mögliche Periodendauer einschränken. Daraus resultieren die in der Tabelle 4.3 angegebenen Parameter für verschiedene Vorteiler-Werte.

Tabelle 4.3: Mögliche Timer-Einstellungen und daraus folgende Perioden- und Zählerstandswerte

Vorteiler-Wert	Timer-Frequenz	T_{min}	T_{max}	Zählerstand für 0,1 mHz LR oder RL	Zählerstand für 10 kHz BP
1	64 MHz	15,625 ns	1,024 ms	$640 \cdot 10^6$	3200
2	32 MHz	31,25 ns	2,048 ms	$320 \cdot 10^6$	1600
4	16 MHz	62,5 ns	4,096 ms	$160 \cdot 10^6$	800
8	8 MHz	125 ns	8,192 ms	$80 \cdot 10^6$	400
16	4 MHz	250 ns	16,384 ms	$40 \cdot 10^6$	200
32	2 MHz	500 ns	32,768 ms	$20 \cdot 10^6$	100
64	1 MHz	1 ms	65,535 ms	$10 \cdot 10^6$	50
128	500 kHz	2 ms	131,070 ms	$5 \cdot 10^6$	25
256	250 kHz	4 ms	262,140 ms	$2{,}5 \cdot 10^6$	12,5
512	125 kHz	8 ms	524,280 ms	$1{,}25 \cdot 10^6$	6,25
1024	62,5 kHz	16 ms	1,049 s	625 000	3,125
2048	31,25 kHz	32 ms	2,097 s	312 500	1,563
4096	15,625 kHz	64 ms	4,194 s	156 250	0,781
8192	7,8125 kHz	128ms	8,388 s	78 125	0,391
16 384	3,9063 kHz	256ms	16,777 s	39 062,5	0,195
32 768	1,9531 kHz	512ms	33,554 s	19 531,25	0,098

Wie in Tabelle 4.3 zu erkennen ist, muss für eine Anregefrequenz von 0,1 mHz ein großer Vorteiler-Wert von 16 384 gewählt werden, durch den die Timer-Frequenz geteilt wird, um einen 16-Bit-Wert (maximal 65 535) zu erhalten. Daraus resultiert ein eingestellter Wert von 39 063 (gerundet) für den Periodentimer. Für das andere Extrem, eine Anregefrequenz von 10 kHz im Modus BP, ist dies Einstellung allerdings ungeeignet. So ergibt sich ein theoretischer einzustellender Zählerstand von unter eins, was technisch nicht möglich ist.

Wenn also eine hohe Anregefrequenz gewählt wird, sollte also ein möglichst kleiner Vorteiler-Wert eingestellt werden, um eine möglichst große Timer-Auflösung zu erreichen.

4.3.3 Lösungsansatz über konkatenierten Timer

Eine mögliche Lösung für dieses Problem wäre das konkatenieren zweier 16-Bit-Timer, um einen funktionalen 32-Bit-Timer zu erhalten. Dieser würde auch bei einem Vorteiler-Wert von eins einen ausreichend großen Wertebereich besitzen. Allerdings ist auch diese Lösung nicht ohne Fehler. So ist es kein echter 32-Bit-Timer, sondern ein Konstrukt, in dem ein Timer nach Erreichen des Periodenwerts den anderen inkrementiert. Daraus ergibt sich, dass die einzustellenden Timer-Werte aufgeteilt werden müssen. Weiterhin müssen die Vergleichswerte der Timer immer niedriger als die Periodenwerte sein, um die Interrupts korrekt auszulösen. Aus diesen Anforderungen resultiert, dass nicht alle Tastgrade, für die die Vergleichswerte gebraucht werden, eingestellt werden können.

4.3.4 Lösungsansatz über Verstellen des Vorteilers

Die Lösung die für diese Arbeit verwendet wird, basiert darauf, den Vorteiler für verschiedene Frequenz passend einzustellen. Dazu wird für den Vorteiler-Wert beginnend bei eins geprüft, ob T_{max} größer oder gleich der geforderten Periodendauer ist. Wenn diese Forderung erfüllt ist, wird der Periodenwert nach Formel 4.4 eingestellt. Dabei wird auf eine natürliche Zahl gerundet. Sollte T_{max} hingegen zu klein sein, wird der Vorteiler-Wert auf die nächste Stufe erhöht und erneut eine Prüfung vorgenommen. Dieser Vorgang wird wiederholt, bis eine erfolgreiche Einstellung des Periodenwerts stattgefunden hat. Um die Rechenzeit auf dem Mikrocontroller zu minimieren, werden die T_{min} Zeiten vorberechnet und aus einem Array abgerufen. Diese Lösung erweist sich als vorteilhaft, da stets der kleinstmögliche Vorteiler-Wert gewählt wird und so die Auflösung maximiert wird. Eine große Auflösung minimiert Fehler, da Frequenzabweichungen des Mikrocontroller-Takts weniger stark ins Gewicht fallen. Ein Nachteil dieses Verfahrens ist hingegen, dass für eine Änderung des Vorteiler-Werts alle Timer abgeschaltet werden müssen. Ein Frequenzwechsel im laufenden Betrieb ist also nicht möglich, allerdings auch nicht gefordert. Wenn eine solche Funktionalität gefordert ist, sollte ein Mikrocontroller mit einem nativen 32-Bit-Timer verwendet werden.

4.4 Konfiguration des Timers für die Periodenzahlbegrenzung

Eine weitere Anforderung war, dass die Anregung nach einer gewissen Anzahl an Perioden vom Mikrocontroller gestoppt werden kann. Um diese Funktion zu realisieren, wurde ein weiterer Timer verwendet, der die Zustandswechsel zählt. Die Verwendung eines Timers hat gegenüber einer Zählung in der Software einige Vorteile. So ist dieses Verfahren hardwarebasiert, spart also Rechenzeit auf dem Mikrocontroller. Auch das Auslösen eines Interrupts ist einfacher möglich.

In Abbildung 4.2 ist die Vorgehensweise dargestellt.

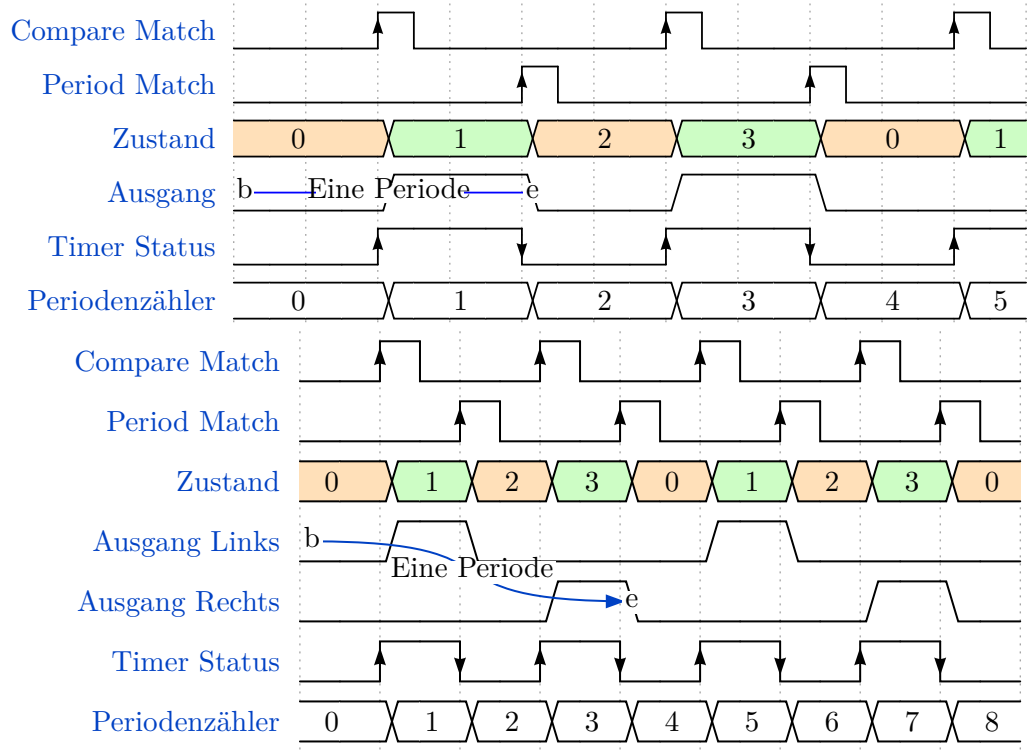


Abbildung 4.2: Funktion des Periodenzählers
oben: Modi LR und RL
unten: Modus BP

Als Eingangssignal für den Timer wird das Statussignal des Frequenz-Timers verwendet. Das Statussignal ist High, wenn der Vergleichswert erreicht oder überschritten ist. Im gegenteiligen Fall, also wenn der Timer-Wert unterhalb des Vergleichswertes liegt, ist das Statussignal Low.

Bei jedem Interrupt, der durch einen erreichten Vergleichswert (Compare-Match) oder Periodenwert (Period-Match) ausgelöst wird, findet also ein Flankenwechsel des Statussignals statt. Nun können die Flankenwechsel gezählt werden, und sobald die geforderte Zahl erreicht ist, wird ein Interrupt getriggert. In diesem Interrupt wird dann die Anregung gestoppt.

Aus Abbildung 4.2 ist ebenfalls ersichtlich, wie der Periodenzähl-Timer eingestellt werden muss. So finden in den Betriebsmodi LR und RL zwei Flankenwechsel pro Periode statt. Im Betriebsmodus BP hingegen liegen vier Flankenwechsel pro Periode vor. Dementsprechend muss die geforderte Periodenzahl mit zwei (LR oder RL) bzw. vier (BP) multipliziert werden. Daraus ergeben sich die Formeln 4.6 und 4.7, mit denen der Periodenzähl-Timer eingestellt werden kann.

$$Periodenzählerwert\ Modus\ LR/RL = Periodenzahl \cdot 2 \tag{4.6}$$

$$Periodenzählerwert\ Modus\ BP = Periodenzahl \cdot 4 \tag{4.7}$$

4.5 Externe Steuerung über UART

Um den Mikrocontroller und damit die Anregeschaltung zu steuern, wird die UART-Schnittstelle verwendet. Dabei wurden eine Baudrate von 115 200, eine Datenwortlänge von acht Bits, eine UART-Framelänge von ebenfalls acht Bits, kein Paritätsbit und ein Stopbit verwendet. Bei den zu übertragenen Informationen handelt es sich um die Frequenz, den Tastgrad in Prozent und die Anzahl der Perioden. Weiterhin muss die Möglichkeit gegeben sein, den Betriebsmodus zu wechseln und die Anregung zu stoppen.

4.5.1 Zusammensetzung der Datenworte

Aus einer Datenwortlänge von acht Bits ergibt sich, dass jedes Datenwort einem Byte entspricht, also vorzeichenfreie Werte von null bis 255 annehmen kann. Wenn Frequenzen von 0 bis 10 000 Hz gesendet werden sollen, müssen also zwei Bytes mit einem vorzeichenfreien Wertebereich von null bis 65 535 verwendet werden. Dabei wird das höherwertige Byte zuerst übertragen (Big-endian). Um den Aufwand zu verringern, werden Frequenzen unter 1 Hz kodiert.

So wird 0,1 Hz etwa mit 65 535 (FFFF im Hexadezimalsystem) übertragen. Diese Frequenzen müssen dementsprechend vorher im Mikrocontroller hinterlegt werden. Für den Tastgrad in Prozent von null bis 100 reicht hingegen ein Byte aus. Ebenso verhält es sich mit der Periodenzahl, da ein Maximalwert von 255 als ausreichend angenommen wird. Sollte irgendwann die Anforderung nach höheren Periodenzahlwerten aufkommen, muss das Datenformat dementsprechend angepasst werden.

4.5.2 Zusätzliche Steuerbefehle

Um die weiteren benötigten Steuerbefehle zu senden, werden keine separaten Datenworte genutzt. Stattdessen werden die bereits vorhandenen verwendet. Eine übertragene Frequenz von null etwa eignet sich, die Anregung zu stoppen. Eine weitere Null als Periodenzahl hingegen wird als Zeichen für eine zeitlich unbegrenzte Anregung gebraucht. Für den Moduswechsel lässt sich nun der ungenutzte Wertebereich des Tastgradbytes von 101 bis 255 verwenden.

4.5.3 Empfang im Mikrocontroller

Ein vollständiges Datenpaket besteht dementsprechend aus vier Datenworten. Deswegen wird der Mikrocontroller so konfiguriert, dass er nach vier empfangenen Datenworten einen Interrupt auslöst. In diesem Interrupt werden nun die Umwandlung und Auswertung der Datenworte vorgenommen. Dabei wird zunächst geprüft, ob Daten im Empfangspuffer vorliegen, vier Bytes eingelesen und eventuell vorhandene überschüssige Daten gelöscht. Anschließend wird die Frequenz gebildet, indem das erste Datenbyte um acht Bits nach links verschoben wird und das zweite Datenbyte dazuaddiert wird. Tastgrad und Periodenzahl lassen sich einfach in Variablen überführen.

4.5.4 Datenpakete

Aus den Anforderungen ergeben sich vier verschiedene Arten von Datenpaketen, die übertragen werden. So stoppt eine übertragene Null die Anregung, wobei die anderen Werte ignoriert werden. Ein Moduswechsel wird dadurch angezeigt, dass ein Wert von 255 bis 252 als Tastgrad übertragen wird. Auch hier werden die anderen Werte verworfen. 255 steht dabei für den Modus LR, 254 für RL, 253 für BP und 252 für IDLE.

Eine zeitlich unbegrenzte Anregung wird gestartet, indem eine Frequenz zwischen 1 und 10 000 Hz oder eine entsprechend kodierte gewählt wird. Zudem muss der Tastgrad zwischen null und 100 liegen und eine Periodenzahl von null gesendet werden. Wenn stattdessen eine Periodenzahl, die nicht null ist, gewählt wird, findet eine zeitlich begrenzte Anregung statt. Dabei entspricht die Anzahl der Perioden des Anregesignals der übertragenen Periodenzahl. Zusätzlich zu den bereits genannten Datenpaketen wird ein weiteres definiert, das nur aus Nullen besteht. Dieses dient dazu, bei Problemen einen Reset des Mikrocontrollers auszulösen. In Abbildung 4.3 findet sich eine Übersicht der verwendeten Datenpakete.

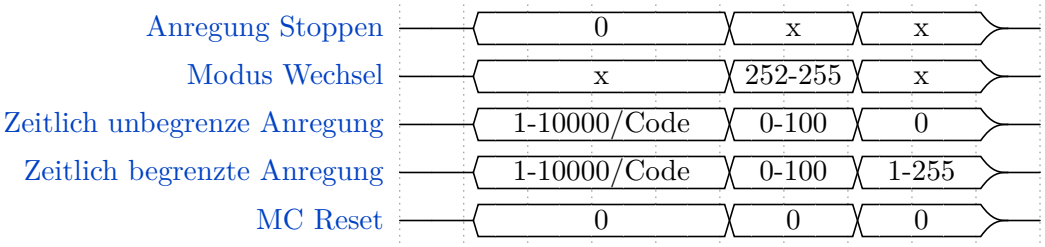


Abbildung 4.3: Aufbau der UART-Datenpakete

4.5.5 Auswertung der Datenpakete

Um die Datenpakete korrekt auszuwerten, wurde eine Auswertelogik erstellt.

Dabei wurde zunächst eine Abstufung der Prioritäten der Datenpakete vorgenommen. So ist das Mikrocontroller-Reset-Datenpaket das Wichtigste und es wird dementsprechend als Erstes auf sein Vorliegen geprüft. Sollte der Test positiv ausfallen, wird der Mikrocontroller sofort resettet und eine Weiterführung der Auswertung entfällt.

Ansonsten wird als Zweites geprüft, ob die Anregung gestoppt werden soll. Die Prüfung fällt dabei positiv aus, wenn eine Frequenz von null übertragen wird.

Als Drittes wird geprüft, ob ein Moduswechsel durchgeführt werden muss. Wenn diese Prüfung positiv ausfällt, der übertragene Tastgrad also zwischen 252 und 255 liegt, wird die Anregung gestoppt und eine Fallunterscheidung vorgenommen. Mit dieser wird der Modus, entsprechend des empfangenen und kodierten Werts, festgelegt. Im Anschluss wird der Ausgangszustand, passend zum Modus, eingestellt. Nach einem Moduswechsel sollten mindestens 30 ms gewartet werden, um den TMR genug Zeit zur Initialisierung zu geben.

Sollte kein Moduswechsel-Datenpaket vorliegen, wird stattdessen eine Gültigkeitsprüfung des Tastgrads vorgenommen. Diese besteht der Tastgrad bei einem Wert zwischen null und 100. Sollte ein ungültiger Tastgrad vorliegen, wird die Auswertung ohne Änderungen an den Einstellungen des Mikrocontrollers beendet.

Die fünfte Prüfung entscheidet, ob der Periodenzähler eingeschaltet wird. Sollte nämlich ein Wert außer null für die Periodenzahl vorliegen, wird der Periodenzähl-Timer auf diese eingestellt.

In jedem Fall wird als Letztes eine Fallunterscheidung für die Frequenz vorgenommen. Diese ist notwendig um eine eventuell vorliegende kodierte Frequenz zu erkennen und sie sowie den Tastgrad, korrekt einzustellen. Wenn keine kodierte Frequenz erkannt wird, werden Frequenz und Tastgrad direkt an die Funktion, die die Einstellungen vornimmt, übergeben. In dieser Funktion findet abschließend noch eine Prüfung statt, ob die geforderte Frequenz innerhalb der Grenzen von 100 mHz und 10 kHz liegt.

In Abbildung 4.4 ist der gesamte Prozess der Umsetzung empfangener UART-Daten in Steuerbefehle in Form eine Nassi-Shneiderman-Diagramms dargestellt.

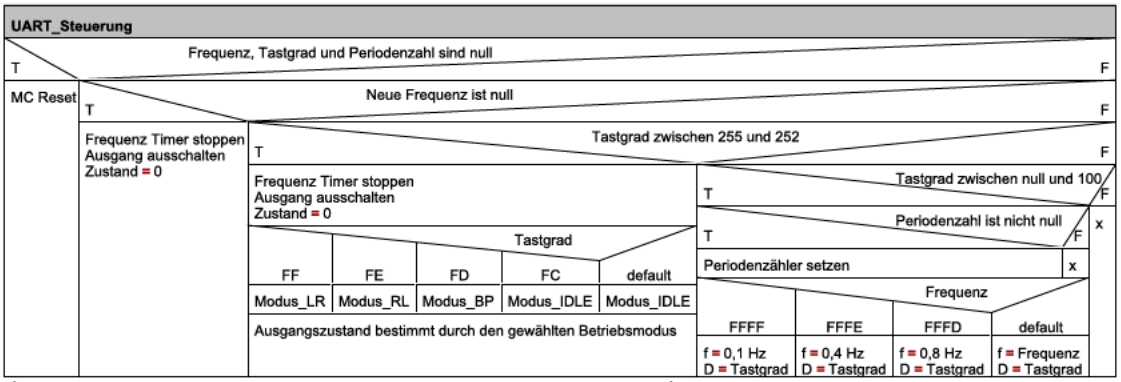


Abbildung 4.4: Nassi-Shneiderman-Diagramm vom Ablauf der Umsetzung empfangener UART-Daten in Steuerbefehle

4.5.6 Fehler in der UART-Kommunikation

Während die UART-Kommunikation getestet wurde, trat ein Fehler beständig auf. So wird, nachdem der Mikrocontroller an einen PC angeschlossen wurde und kein Reset durchgeführt wird, stets zuerst ein Byte mit dem Wert null empfangen. Die darauf folgenden Bytes sind dementsprechend um eine Stelle verschoben. Auch ein Leeren des Empfangspuffers bei zu vielen empfangenen Daten konnte keine Abhilfe schaffen. Ein Löschen der Sendepuffer auf dem PC konnte das Problem ebenfalls nicht lösen.

Im Rahmen dieser Arbeit konnte nur ein Workaround geschaffen werden. So funktioniert die Kommunikation nach einem Reset wieder problemlos, weshalb ein solcher bei fehlerhaft zurückgeschriebenen Daten ausgelöst wird. Dabei muss das notwendige Datenpaket wegen der auftretenden Byte-Verschiebung zweimal gesendet werden, wobei das Starten des Mikrocontrollers abgewartet werden muss. Signalisiert wird die Bereitschaft des Mikrocontrollers der PC-Software dabei durch eine Übertragung von einem Zeilenumbruch.

4.6 Interrupts

Aus der Struktur des Programms ergeben sich vier verschiedene Interrupts, die in Tabelle 4.4 aufgeführt sind.

Tabelle 4.4: Interrupts in der Mikrocontroller Software

Name des Interrupts	Ausgelöst durch	Funktion
Frequenz-Timer Compare Match	Frequenz-Timer erreicht Vergleichswert	Zustandswechsel null auf eins und zwei auf drei (Anregung an)
Frequenz-Timer Period Match	Frequenz-Timer erreicht Periodenwert	Zustandswechsel eins auf zwei und drei auf null (Anregung aus)
Periodenzähl-Timer Period Match	Periodenzähl Timer erreicht Periodenzähl Wert	Stoppen der Anregung
UART Receive Buffer Standard Event	Empfangs Buffer erreicht maximalen Füllstand	Auswertung des empfangenen Datenpakets

Um einen reibungslosen Programmablauf zu gewährleisten, müssen diese Interrupts priorisiert werden. Dabei kann auf dem verwendeten Mikrocontroller auf vier verschiedene Prioritätslevel zurückgegriffen werden.

Die höchste Priorität muss dabei den Frequenz-Timer-Interrupts zugewiesen werden. Ansonsten besteht die Gefahr, dass das Programm bei eingeschalteter Anregung unterbrochen wird. Daraus würde eine Verlängerung der Zustandsphasen und damit eine Verzerrung des Anregesignals resultieren. Eine Unterscheidung zwischen Compare- und Period-Match-Interrupts muss dabei nicht vorgenommen werden, da sie nie gleichzeitig auftreten.

Die Priorität des Periodenzähl-Timer-Interrupts sollte eine Stufe unter der des Frequenz-Timers liegen. So ist gewährleistet, dass ein Schaltvorgang erst abgeschlossen ist, bevor die Anregung gestoppt wird. Damit ist ein eindeutiges Verhalten sicher, da davon ausgegangen werden kann, dass der Frequenz-Timer gestoppt wird, bevor er einen erneuten Interrupt auslöst.

Wenn die Priorität hingegen erhöht wird, kann es dazu kommen, dass ein Frequenz-Timer-Interrupt in den Pending-Status versetzt wird. Dieser würde nach dem Ende des Periodenzähl-Timer-Interrupts dann einen ungewollten Zustandswechsel auslösen.

Dem UART-Interrupt wird schließlich die niedrigste Priorität zugewiesen. Dementsprechend werden Steuerbefehle erst ausgeführt, wenn kein Schaltvorgang erfolgt. So ist gewährleistet, dass das Anregesignal ohne Verzerrungen ausgegeben wird. Weiterhin funktioniert so auch das genaue Abschalten der Anregung, nachdem die geforderte Periodenzahl erreicht ist.

In Abbildung 4.5 ist die Priorisierung der Interrupts noch einmal als Übersicht dargestellt.

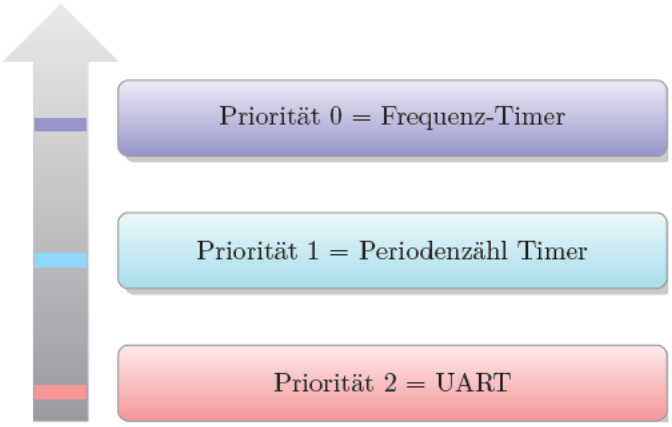


Abbildung 4.5: Prioritäten der Interrupts

4.7 Absicherung über WatchDog-Timer

Um eine Absicherung gegen Fehler zu schaffen, wird ein WatchDog-Timer verwendet. Dieser löst, wenn er in einem bestimmten Zeitraum nicht durch einen Befehl bedient wird, einen Reset aus. Im vorliegenden Programm befindet sich dieser Befehl in einer Endlosschleife in der Main-Funktion. Somit wird sichergestellt, dass Interrupts eine bestimmte Bearbeitungszeit nicht überschreiten. Wichtig ist diese Absicherung besonders für den UART-Interrupt, da durch den WatchDog-Timer verhindert wird, dass Fehler in der Kommunikation und eventuell ungewollte Ausgangszustände ausgelöst werden.

Durch Tests ließ sich ein Grenzwert von 30 ms für den WatchDog-Timer festlegen. Dieser Zeitraum erwies sich als ausreichend, um selbst die am längsten dauernde UART-Anweisung umzusetzen. Sollten in Zukunft weitere Funktionalitäten nachgerüstet werden, muss der Grenzwert eventuell angepasst werden.

Um einen WatchDog-Timer-Reset zu behandeln, wird bei jedem Start des Mikrocontrollers der Grund für den letzten Reset abgefragt. Sollte der WatchDog-Timer diesen verursacht haben, werden zuerst die Ausgänge gesichert, indem sie abgeschaltet werden. Anschließend wird der Reset-Grund zurückgesetzt, damit die Abfrage nach einem Neustart nicht erneut ausgelöst wird. Letztendlich wird dann eine Dauerschleife ausgeführt, in der der WatchDog-Timer bedient wird. Dies dient dazu, dass vom Anwender ein bewusster Reset ausgeführt werden muss, um das Programm normal ausführen zu können. Zudem kann dieser Reset nicht über die UART-Schnittstelle ausgelöst werden. Dieses Verhalten stellt sicher, dass Fehler geprüft und nicht einfach nur bestätigt werden. Zudem erleichtert die Dauerschleife die Fehlersuche, da mit dem Debugger einfach festgestellt werden kann, wenn sie ausgeführt wird. Daraus lässt sich dann schließen, dass die Grenze des WatchDog-Timers überschritten ist und Prüfungen in diese Richtung verlaufen sollten.

4.8 Aufbau des C-Programms auf dem Mikrocontroller

Aus den getätigten Vorüberlegungen resultierte schließlich ein C-Programm, das zur besseren Übersicht in Module aufgeteilt wurde. Grundeinstellungen werden dabei über den Device-Configurator der ModusToolbox™ vorgenommen. Dieser generiert dann automatisch passende Dateien mit Funktionen zur Initialisierung, die in der Main-Funktion ausgeführt werden. So wird die Grundkonfiguration des Mikrocontrollers vorgenommen.

4.8.1 Modi, Zustände und Pin-Modul

Das Modul dient dazu, die Betriebsmodi, Ausgangszustände und Pins festzulegen. Dabei wurde zunächst mittels Typedef ein Alias-Datentyp mode_t für den Betriebsmodus erstellt. Anschließend wurden zwei Alias-Datentypen erstellt, die den Bitmasken zum Ein- und Ausschalten der Ausgangspins verständlichere Namen zuordnen. Als Nächstes wird mit ihrer Hilfe ein Datentyp erzeugt, der die Ausgangszustände mit den passenden Pin-Schaltstellungen belegt. Als letzter Schritt wird dann die Zustandsmatrix als konstante globale Variable erzeugt. Zum Erzeugen der Datentypen wurde der Enumeration-Datentyp verwendet. Dieses Vorgehen hat gegenüber der Verwendung von Präprozessor-Makros den Vorteil, dass der Compiler grobe Fehler besser erkennen kann.

4.8.2 Timer-Modul

Im Timer-Modul werden zunächst zwei Variablen für die T_{min}-Zeiten und die Vorteiler-Werte angelegt. Im nächsten Schritt werden dann drei Funktionen zum Einstellen der Timer angelegt. Dabei gibt es eine Funktion, die den Frequenz-Timer mit einem gegebenen Periodenwert und Tastgrad passend einstellt. Aufgerufen wird diese Funktion dabei von der zweiten Funktion, die eine eingegebene Frequenz in den richtigen Periodenwert umrechnet. Die letzte Funktion dient dann dazu, den Periodenzähl-Timer-Wert festzulegen.

4.8.3 UART-Kontroll-Modul

Im UART-Kontroll-Modul werden empfangene Datenpakete ausgewertet und in Steuerbefehle umgesetzt. Dazu werden vier Funktionen definiert. Die Hauptfunktion dient dabei dazu, die Daten, wie in Abbildung 4.4 dargestellt, auszuwerten. Dabei wurden die Prüfung auf kodierte Frequenzen und die Verarbeitung von Moduswechseln in eigenen Funktionen ausgelagert. Zum Schluss gibt es noch eine Funktion, die die Anregung stoppt. Dazu wird der Frequenz-Timer angehalten und der erste Zustand des Betriebsmodus eingestellt.

4.8.4 Main-Modul

Im Main-Modul werden zuerst zwei globale und private Variablen für den Betriebsmodus und den Zustand definiert.

Das Erstellen der Interrupt-Handler ist dann der nächste Schritt. Dabei werden zuerst die Handler für den Frequenz-Timer angelegt. Ihr Aufbau ist dabei nahezu gleich. Zunächst wird das auslösende Ereignis zurückgesetzt, dann die Zustandsvariable je nach Wert erhöht oder auf null gesetzt und schließlich die Pins gesetzt. Dazu wird die Zustandsmatrix verwendet, indem der passende Wert nach Betriebsmodus und Zustandszählvariablenwert abgerufen wird.

Es folgt der Interrupt-Handler für den Periodenzähl-Timer, der einfach nur die Anregung stoppt.

Im UART-Empfangs-Interrupt-Handler, der als Nächstes folgt, werden dann die empfangenen Daten umgewandelt und an die Auswertefunktion aus dem UART-Kontroll-Modul übergeben. Dazu wird zunächst geprüft, ob der Empfangspuffer leer ist und der Handler, sollte dieser Fall eintreten, abgebrochen. Im Anschluss wird das vier Byte große Datenpaket aus dem Puffer ausgelesen. Sollte der Empfangspuffer danach nicht leer sein, wird er geleert. Das Zurückschreiben der Daten zur Überprüfung ist dann der nächste Schritt. Anschließend wird das Datenpaket in seine Bestandteile zerlegt und passend umgewandelt. Als Letztes wird dann die Auswertefunktion ausgeführt.

In der nun folgenden Main-Funktion wird nach der Initialisierung des Mikrocontrollers zunächst geprüft, ob ein WatchDog-Timer-Reset vorliegt. Der zweite Schritt ist dann das Festlegen der Initialwerte für Betriebsmodus und Ausgangszustände der Pins. Als Drittes werden, nach dem präventiven Leeren des Empfangspuffers, die Prioritätslevel der Interrupts festgelegt und sie ebenfalls aktiviert. Zum Abschluss wird eine Bereitschaftsmeldung über die UART-Schnittstelle gesendet, bevor eine Dauerschleife ausgeführt wird. In dieser wird der WatchDog-Timer bedient, um ein Auslösen zu verhindern.

4.9 Python-Skript zur Steuerung von Mikrocontroller und Oszilloskop

Eine Erleichterung der Ansteuerung des Mikrocontrollers über den PC und ein automatisiertes Aufnehmen von Messreihen wurde über zwei Python-Skripte realisiert. Grundlage für die Skripte bildet dabei zum einen das PySerial Modul (PySerial). PySerial stellt dabei ein Backend bereit, das Python einen Zugriff auf die serielle Schnittstelle ermöglicht [16]. So lässt sich dann der Mikrocontroller steuern. Um auch das Oszilloskop über den PC steuern zu können, wurde das PyVISA Modul (PyVISA) verwendet. Es ermöglicht die Steuerung von Messhardware über die VISA-API. Dabei mussten bestimmte Befehle für das verwendete Oszilloskop Tektronix MSO 44 eingesetzt werden, um es fernzusteuern [17].

Beim ersten verwendeten Skript handelt es sich um eines, welches die Steuerung des Mikrocontrollers über die Kommandozeile erleichtert. Es liest eingegebene Zahlen ein und wandelt diese, unter Berücksichtigung der Bytefolge, in Hexadezimalzahlen um. Anschließend werden die Zahlen in der korrekten Reihenfolge versendet und danach mit den zurückgeschriebenen Werten vom Mikrocontroller verglichen. Bei Fehlern ist das Skript dann in der Lage, einen Reset beim Mikrocontroller auszulösen.

Mit dem zweiten verwendeten Skript können automatisierte EIS-Messungen durchgeführt werden. Dazu muss eine funktionale Verbindung mit dem Mikrocontroller bestehen. Deswegen sollte im Vorfeld eine Prüfung mit dem ersten Skript erfolgen, welches bei Problemen automatisch einen Reset durchführt. Als Erstes wird im Skript eine Funktion angelegt, die die Übertragung zum Mikrocontroller übernimmt. Weiterhin werden Listen mit den Messfrequenzen und Tastgraden angelegt. Anschließend werden Variablen mit den Einstellungen für Oszilloskop und Mikrocontroller angelegt. Darauf folgt die Initialisierung der Kommunikation mit dem Oszilloskop und das Festlegen des Speicherorts für die Messdaten. Als Nächstes wird der Mikrocontroller auf den richtigen Betriebsmodus eingestellt, wobei die Wartezeit eingehalten wird. Wenn anschließend die Anzahl der Datenpunkte im Oszilloskop eingestellt ist, beginnt die Messung. Zunächst wird dazu für jede Frequenz die Abtastrate so eingestellt, dass die geforderte Anzahl an Perioden aufgenommen werden kann. Nun wird das Oszilloskop so konfiguriert, dass es eine einzelne Sequenz misst, sobald es von einem Signal getriggert wird. Nach einer kurzen Pause wird dann die Anregung mit den Werten aus den Listen gestartet. Wenn die Sequenz aufgenommen ist, werden die Daten am konfigurierten Ort gesichert.

Im Anschluss wird zur Sicherheit ein Befehl zum Abschalten der Anregung durchgeführt, obwohl diese im Normalfall durch den Periodenzähl-Timer erfolgt. Nach einer kurzen Pause beginnt dann die Messung für die nächste Frequenz. Dieser Ablauf wird für alle Werte der Frequenzliste wiederholt und, sollten mehrere Durchläufe gewünscht sein, wird die Frequenzliste dementsprechend oft durchlaufen. Wenn alle Messungen abgeschlossen sind, wird der Mikrocontroller in den Betriebsmodus IDLE versetzt und die Verbindung zu ihm und dem Oszilloskop beendet.

5 Messungen und Auswertung

Bevor mit den Messungen begonnen wurde, fand zunächst eine allgemeine Funktionskontrolle statt. Dabei wurde zunächst geprüft, ob alle Bauelemente korrekt verbunden sind. Die Funktion der TMR-Bausteine wurde überprüft. Auch eine Prüfung der Mikrocontroller-Software fand statt, indem die Anregung in allen Betriebsmodi ohne Batterien gestartet wurde. Bei niedriger Frequenz konnte dann eine Prüfung des korrekten Zustandsablaufs über die LED an den Mikrocontroller-Pins durchgeführt werden. Auch ein Funktionstest des Periodenzählers und der UART-Steuerung wurde erfolgreich absolviert. Bei all diesen Tests traten keine Fehler auf, sodass mit Versuchen mit eingesetzten Batterien begonnen werden konnte.

5.1 Auswahl der Widerstände

Bevor mit den Messungen begonnen werden kann, müssen zunächst noch die Ein-, Ausgangs- und Messwiderstände ausgewählt werden. Für die Versuche wurden Messwiderstände (R1 und R2) mit einem Wert von 2,2 Ω und Eingangswiderstände (R13 und R24) mit einem Wert von 1 Ω verwendet.

Die Ausgangswiderstände sollten so gewählt werden, dass der Strom so groß wie möglich wird, dabei jedoch nicht das Limit des TMR von 500 mA [8] überschreitet. Erschwert wird die Auswahl dabei durch die Tatsache, dass die Spannung der angeregten Batterie nicht konstant ist. Da die Spannungsdifferenz zwischen TMR-Ausgang und Batterie variiert, gilt dies ebenfalls für den Ausgangsstrom.

Um den Ausgangsstrom I_a zu berechnen, kann folgende Formel 5.1 angewendet werden.

$$I_a = \frac{Udiff}{R_a + R_{Mess} + R_{Batt} + R_{MosfetSW}} \tag{5.1}$$

Dabei ist U_{diff} die Spannungsdifferenz zwischen Ausgang des TMR und Batterie. R_{Batt} ist der Innenwiderstand der Batterie, in diesem Fall mit 21 mΩ angenommen [7]. Der Widerstand des Schaltmoduls kann aus dem Datenblatt mit etwa 7 mΩ abgelesen werden [11]. Es muss jedoch beachtet werden, dass I_a nur ungefähr bestimmt werden kann, da sich sowohl der Innenwiederstand der Batterie bei verschiedenen Ladezuständen als auch der Widerstand des Schaltmoduls bei unterschiedlichen Spannungen ändert. Gleichzeitig haben sie jedoch, durch ihre geringe Größe, keinen großen Einfluss auf den Stromfluss.

Um eine Entscheidung für die Ausgangswiderstände treffen zu können, sollte auch der maximal mögliche Wert für U_{diff} bestimmt werden. Dazu kann Formel 5.1 nach U_{diff} aufgelöst werden. Daraus resultiert die Formel 5.2, mit der die Spannungsdifferenz bei einem Ausgangsstrom von 500 mA bestimmt werden kann.

$$U_{diff} = I_a \cdot (R_a + R_{Mess} + R_{Batt} + R_{MosfetSW}) \tag{5.2}$$

Mit der Formel 5.1 kann I_a für den kleinsten Wert von U_{diff} bestimmt werden. Dieser liegt wegen der nominalen Spannung der Batterie von 10,8 V [7] bei 1,2 V. Dabei sollte I_a so groß wie möglich sein. Mit Formel 5.2 kann dann der maximale Wert für U_{diff} berechnet werden, was die Ermittlung der niedrigstmöglichen Spannung der zu messenden Batterie ermöglicht. Dabei sollte U_{diff} für einen großen Messbereich bei einem I_a von 500 mA ebenfalls keinen zu geringen Wert annehmen.

In Tabelle 5.1 sind die Werte für I_a und U_{diff} unter Berücksichtigung der Einschränkungen aufgeführt.

Tabelle 5.1: I_a und U_{diff} für verschieden R_a Werte

R_a	I_a für $U_{diff} = 1{,}2\,\mathrm{V}$	U_{diff} für $I_a = 500\,\mathrm{mA}$
1 Ω	372 mA	1,614 V
2,2 Ω	271 mA	2,214 V
4,7 Ω	45 mA	3,464 V

Aus den Berechnungsergebnissen aus Tabelle 5.1 kann 2,2 Ω als bestmöglicher Wert für R_a bestimmt werden. Für diesen Widerstandswert ergibt sich der beste Kompromiss aus der Größe des Messbereichs (U_{diff} = 2,214V) und minimalem Ausgangsstrom (I_a = 271mA). Für ein R_a mit 4,7 Ω ist I_a mit 45 mA sehr klein, was auch auf den U_{diff} Wert von 1,614 V bei 1 Ω zutrifft. Auch der höhere Leistungsverlust bei R_a mit 4,7 Ω ist problematisch. Dementsprechend wurde ein Ausgangswiderstand mit 2,2 Ω gewählt.

Insgesamt lässt sich feststellen, dass das Schaltungsdesign mit dem beschränkten Ausgangsstrom der TMR nicht optimal ist. Eine Messung über den gesamten Spannungsbereich der Batterie von 7,5 bis 10,8 V [7] lässt sich nämlich nur über Wechsel der Ausgangswiderstände realisieren. Ohne Wechsel wird nämlich entweder der Ausgangsstrom zu klein für eine sinnvolle Messung bei hohen Batteriespannungen oder aber der TMR schaltet bei zu großen Strömen und niedrigen Batteriespannungen ab. Als Verbesserung sollte für zukünftige Designs eine Anregung mit konstantem Strom und der Last entsprechend geregelter Spannung gewählt werden. Ein Bauteil mit solchen Eigenschaften ist allerdings nicht zu beschaffen. Dementsprechend müsste eine eigene Schaltung entworfen werden, was mit Aufwand verbunden wäre.

5.2 Auswahl der Kondensatoren

Zur Auswahl der Ein- und Ausgangskondensatoren wurden Messungen des Anregesignals an den Messwiderständen und Batterien vorgenommen. Dabei wird nur für einen MP Modus gemessen, da sich der Modus RL nicht grundlegend vom Modus LR unterscheidet. Weiterhin wird auch für den Modus BP eine Messung durchgeführt. Dabei werden Anregefrequenzen von 1, 100 und 1000 Hz eingestellt. Durchgeführt wurden die Messungen mit dem Tektronix MSO 44-Oszilloskop, dessen Kanäle auf DC-Kopplung eingestellt werden. Aufgenommen wurden zwei Perioden, wobei nach Starten der Anregung 0,5 s gewartet wurde, um eventuelle Störungen zu vermeiden. Der Aufbau der Messung ist in Abbildung 5.1 dargestellt. In Abbildung 5.2 sind die Messabgriffe im vereinfachten Schaltbild zu sehen. Die Spannung der Batterien lag zu Beginn der Messungen bei 10,06 V. Nach der Messung lag die Spannung der linken Batterie bei 9,95 V und die der rechten bei 9,91 V.

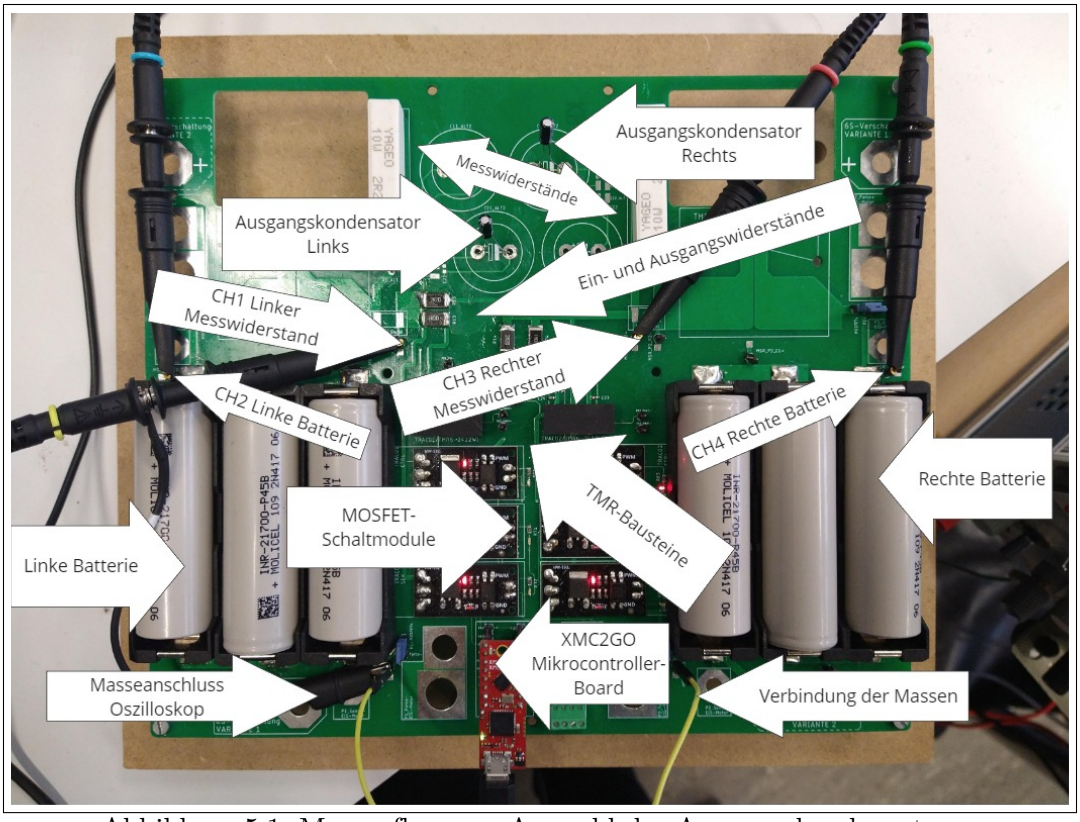


Abbildung 5.1: Messaufbau zur Auswahl der Ausgangskondensatoren

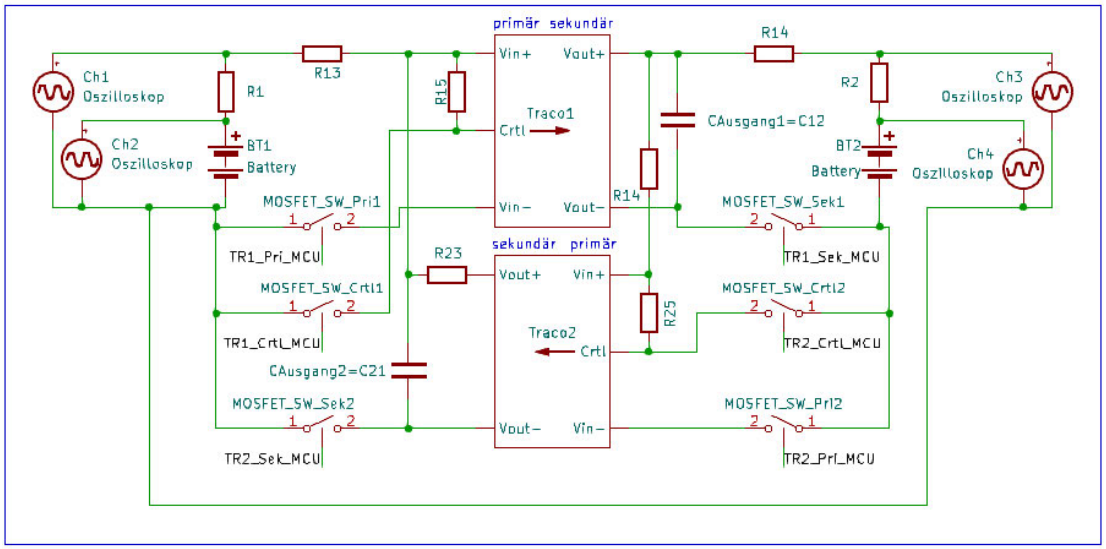


Abbildung 5.2: Schaltbild des Messaufbaus zur Auswahl der Ausgangskondensatoren

5.2.1 Messung ohne Kondensatoren

Als Erstes wurde eine Messung ohne Ein- und Ausgangskondensatoren durchgeführt, um zu prüfen, ob sie für ein minimal verzerrtes Anregesignal erforderlich sind. Dabei wird der Gleichanteil der Signale vor Erstellung der Diagramme mit Matlab durch eine Mittelwertberechnung entfernt.

Zunächst werden die steigenden Flanken der Signale in Abbildung 5.3 betrachtet.

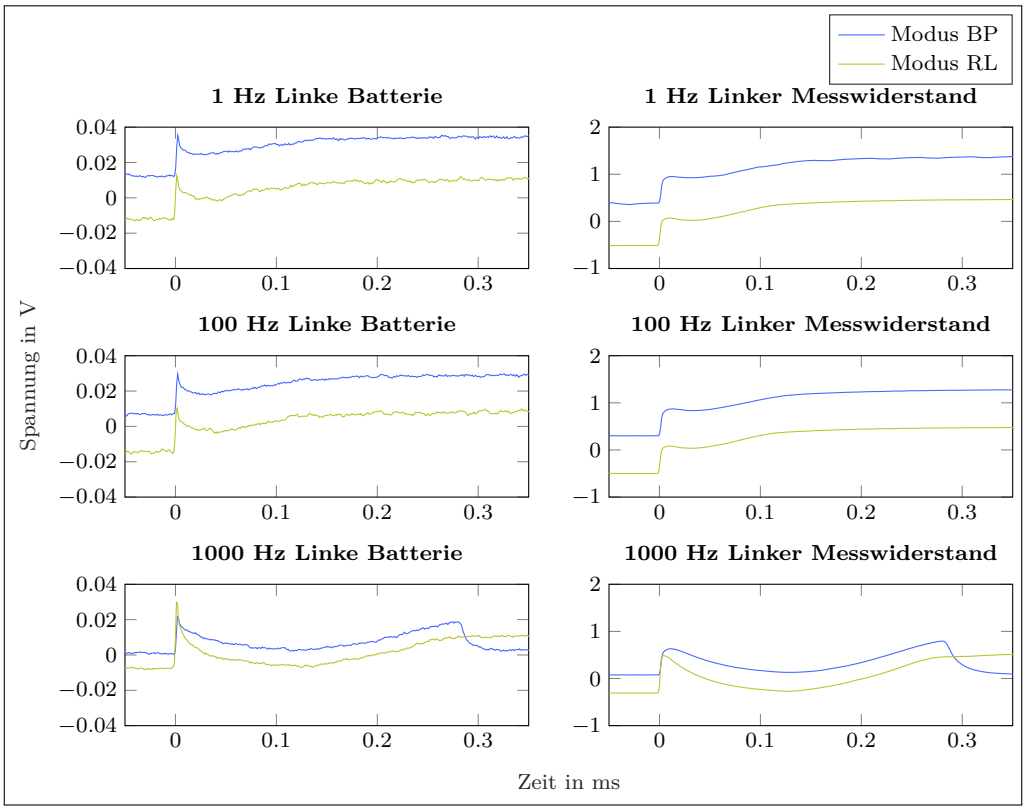


Abbildung 5.3: Steigende Flanke des Anregesignals ohne Kondensator

Die unterschiedlichen Amplituden der Signale resultieren aus Lade- und Entladevorgängen, denen die linke Batterie unterliegt. Dadurch verschiebt sich die Spannungsdifferenz zwischen Ausgang des TMR und der zu messenden Batterie, wodurch sich die Amplitude verändert. Man kann aus den Messungen zudem deutlich ableiten, dass ohne Kondensatoren kein verzerrungsfreies Signal erzeugt werden kann. So tritt für alle Frequenzen an den steigenden Flanken ein deutlicher Spannungseinbruch auf. Dieser ist erst nach etwa 250 µs durch Nachregeln des TMR kompensiert, was sich mit seinem Datenblatt deckt. In diesem wird nämlich eine Reaktionszeit von 250 µs auf Transienten angegeben [8].

Der größere Einbruch bei 1000 Hz resultiert vermutlich aus der reduzierten Zeit zwischen den eingeschalteten Zuständen der TMR-Ausgänge. So hat der TMR weniger Zeit, sich aufzuladen, bevor er wieder belastet wird.

Nach den steigenden Flanken werden in Abbildung 5.4 die fallenden Flanken betrachtet.

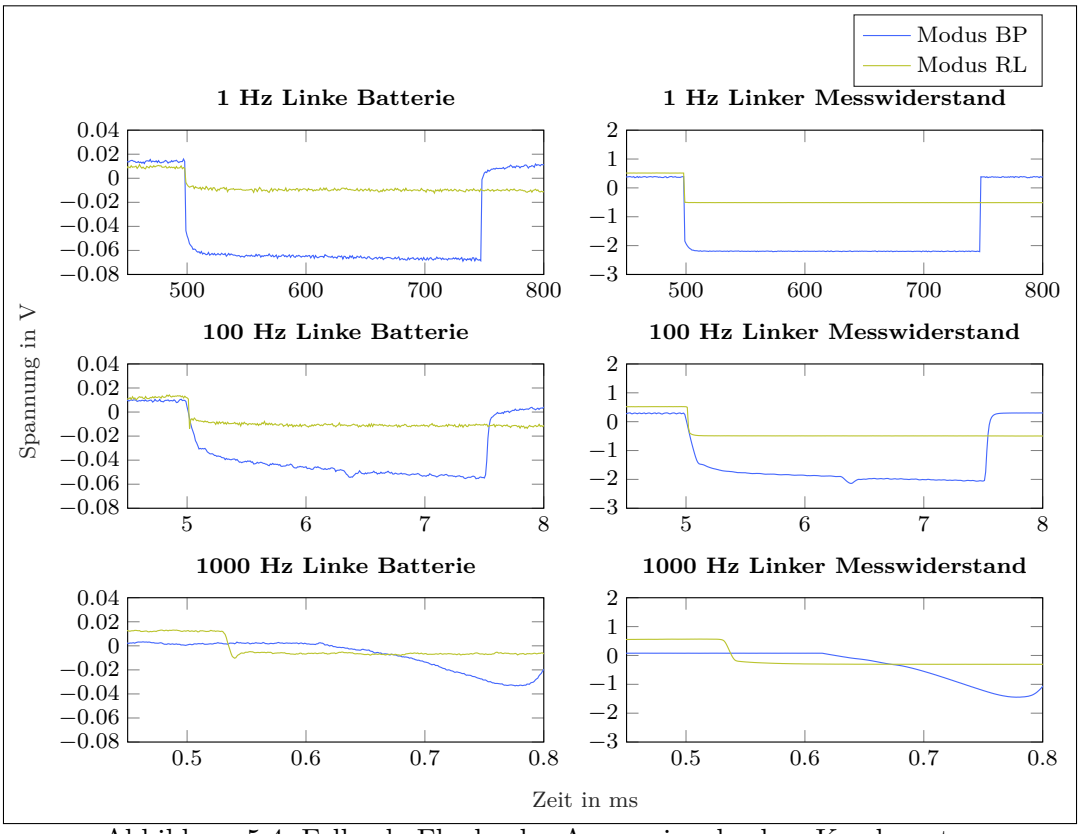


Abbildung 5.4: Fallende Flanke des Anregesignals ohne Kondensator

Für die fallenden Flanken zeigt sich bei 1000 Hz für beide Betriebsmodi eine deutliche Verzögerung. Da der Ausgangskondensator im Modus RL keine Auswirkung auf die fallende Flanke des Anregesignals hat, muss hier ein anderes Problem vorliegen, welches später untersucht wird. Für den Modus BP tritt allerdings ein zusätzliches Problem auf. In Abbildung 5.4 ist bei 1000 Hz zu erkennen, dass der negative Puls noch später auftritt, selbst wenn die Grundverzögerung beider Betriebsmodi berücksichtigt ist. Diese Verzögerung resultiert vermutlich ebenfalls aus der Regelverzögerung des TMR. So wird der negative Puls durch den rechten TMR ausgelöst, sobald dieser einen positiven Puls auf der rechten Seite erzeugt. Da dieser Vorgang einer Regelzeit von etwa 250 µs unterliegt, wird die Spannung auf der linken Seite dementsprechend auch verzögert reduziert.

Es lässt sich auch schließen, dass sich die Abweichungen an den Flanken stärker auf höherfrequente Signale auswirken. Grund ist der größere Anteil, den die Abweichungen durch die kürzeren Periodendauern an höher frequenten Anregesignalen haben. In Abbildung 5.5 ist dieses Verhalten für eine Periode des Signals am Messwiderstand für drei Frequenzen veranschaulicht. Es lässt sich deutlich erkennen, wie das Anregesignal mit steigender Frequenz immer stärker verzerrt wird.

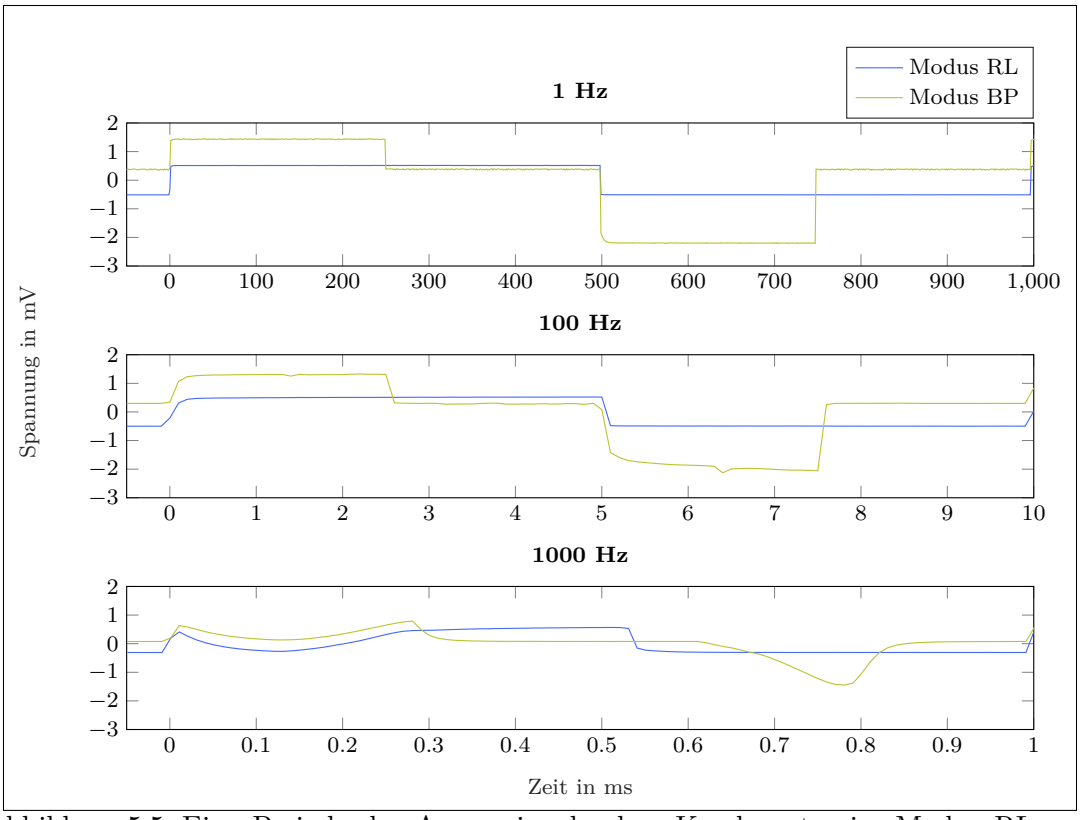


Abbildung 5.5: Eine Periode des Anregesignals ohne Kondensator im Modus RL und BP, am Messwiderstand gemessen

5.2.2 Messung mit Ausgangskondensatoren

Da ein verzerrungsfreies Signal ohne Kondensatoren nicht erreicht werden kann, wurden zunächst drei verschiedene Ausgangskondensatoren eingesetzt. Diese sollen die Ausgangsspannung des TMR zum Einschaltzeitpunkt des Anregesignals stabilisieren. In Abbildung 5.6 für den Modus RL und Abbildung 5.7 für den Modus BP sind die steigenden Flanken des Anregesignals für Ausgangskondensatoren mit den Werten 10 µF, 33 µF und 47 µF aufgeführt.

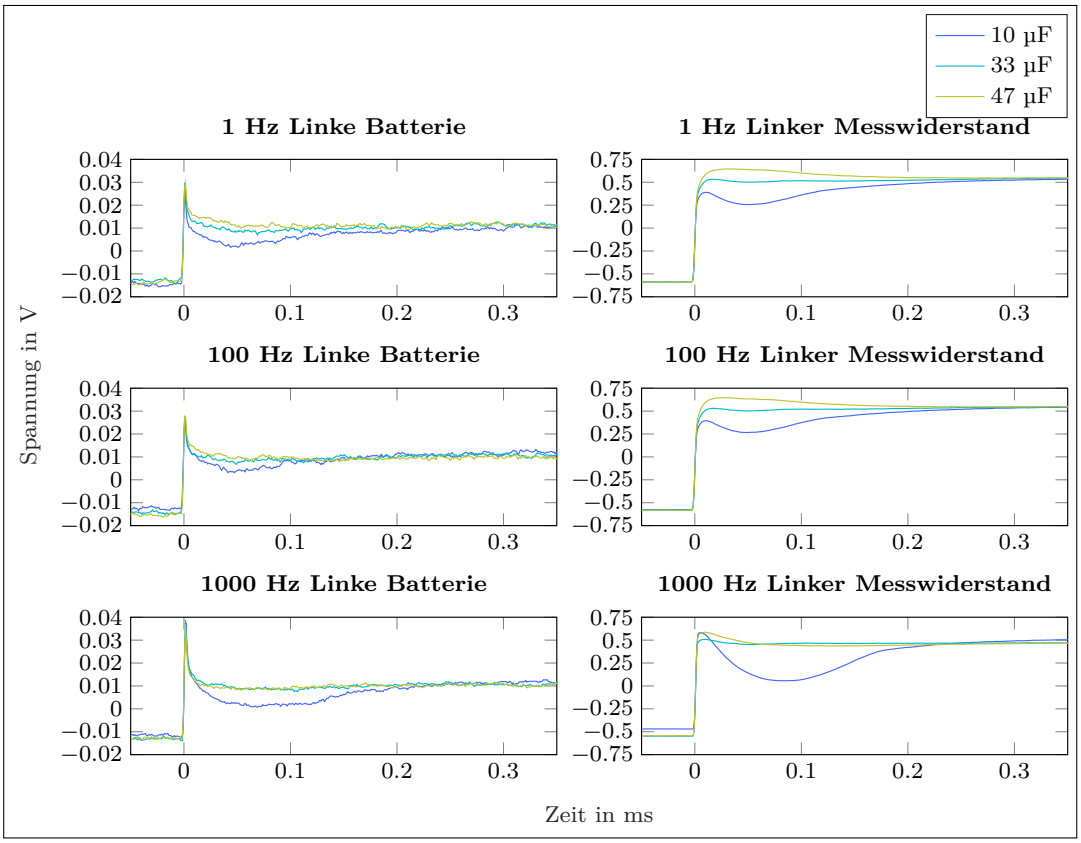


Abbildung 5.6: Steigende Flanke des Anregesignals mit Ausgangskondensatoren im Modus RL

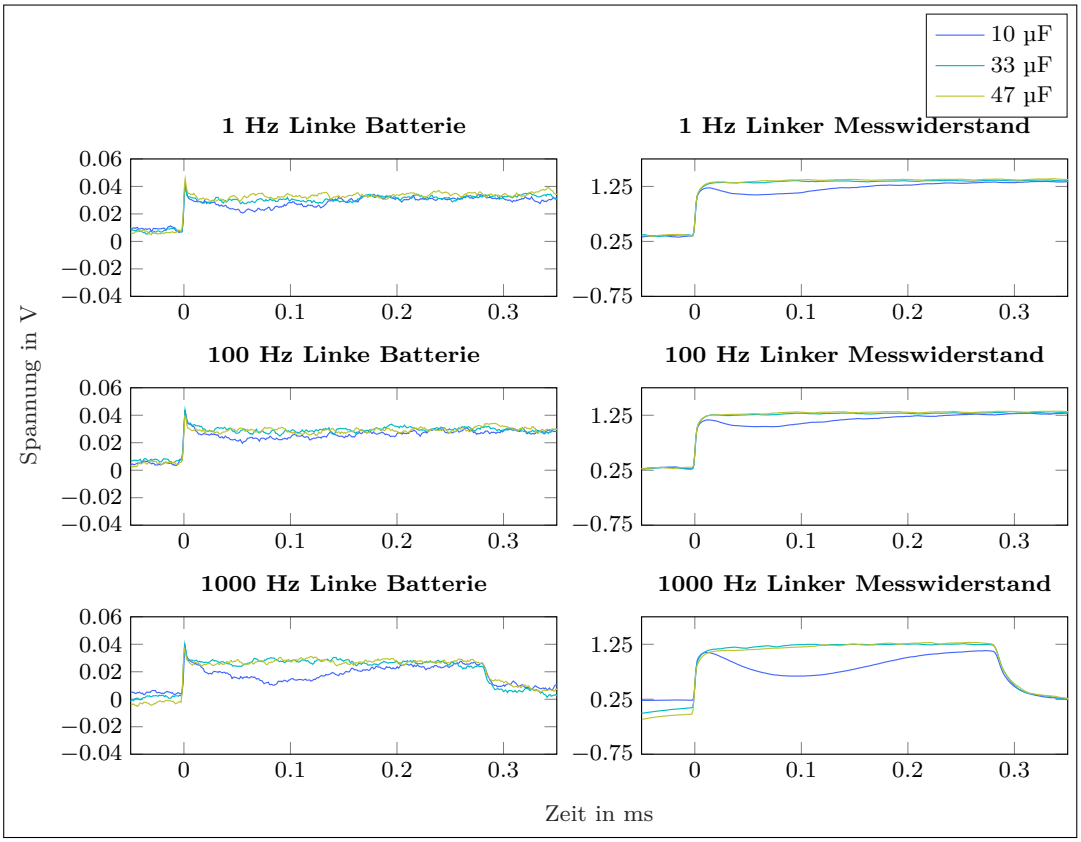


Abbildung 5.7: Steigende Flanke des Anregesignals mit Ausgangskondensatoren im Modus BP

Für die steigende Flanke lässt sich feststellen, dass ein Kondensator mit einem Wert von 33 µF optimal ist. Der 10 µF verringert den Spannungseinbruch nicht ausreichend, während bei 47 µF im Modus RL am Messwiderstand schon ein Überschwingen erkennbar ist.

Für die fallende Flanke des Anregesignals im BP hat der Ausgangskondensator ebenfalls einen positiven Einfluss. Da der TMR ihn bei geöffnetem Schalter am Ausgang auflädt, verteilt sich die Belastung am Eingang über einen größeren Zeitraum, was zu einer verbesserten Pulsform des negativen Pulses in Abbildung 5.8 führt.

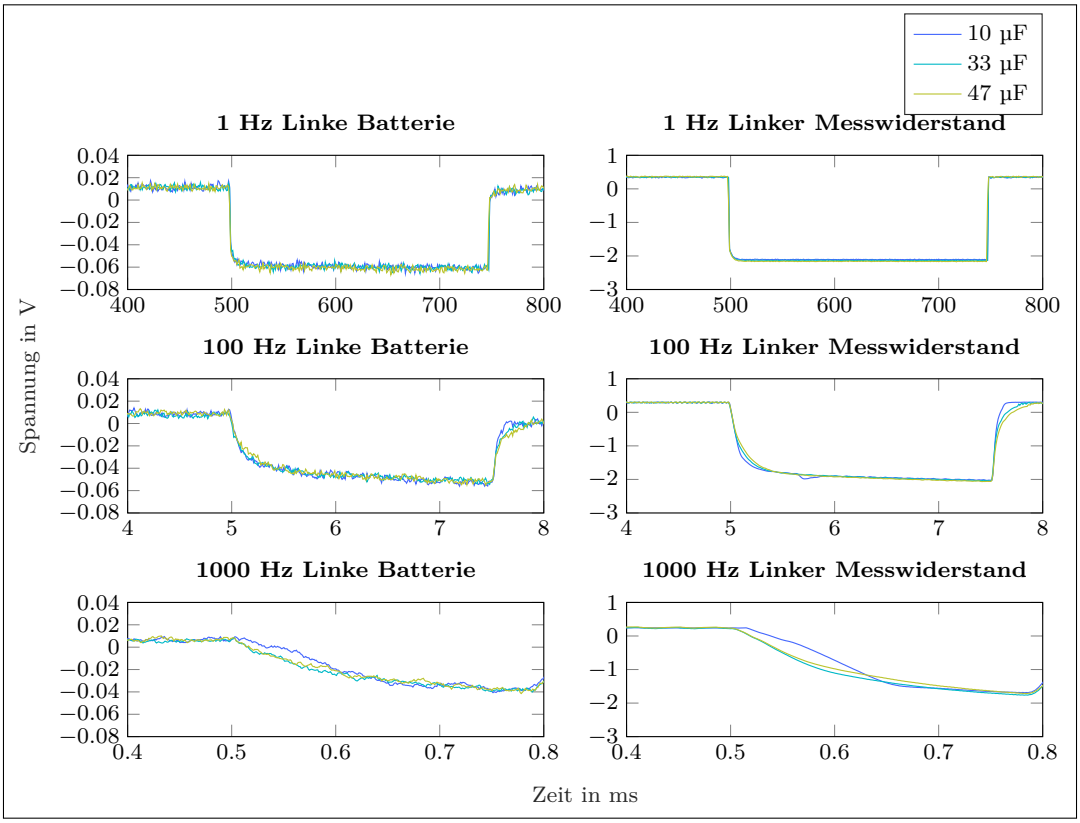


Abbildung 5.8: Fallende Flanke des Anregesignals mit Ausgangskondensatoren im Modus BP

Auch die fallenden Flanken im Modus RL sind durch diesen Effekt steiler geworden, wie in Abbildung 5.9 zu sehen ist.

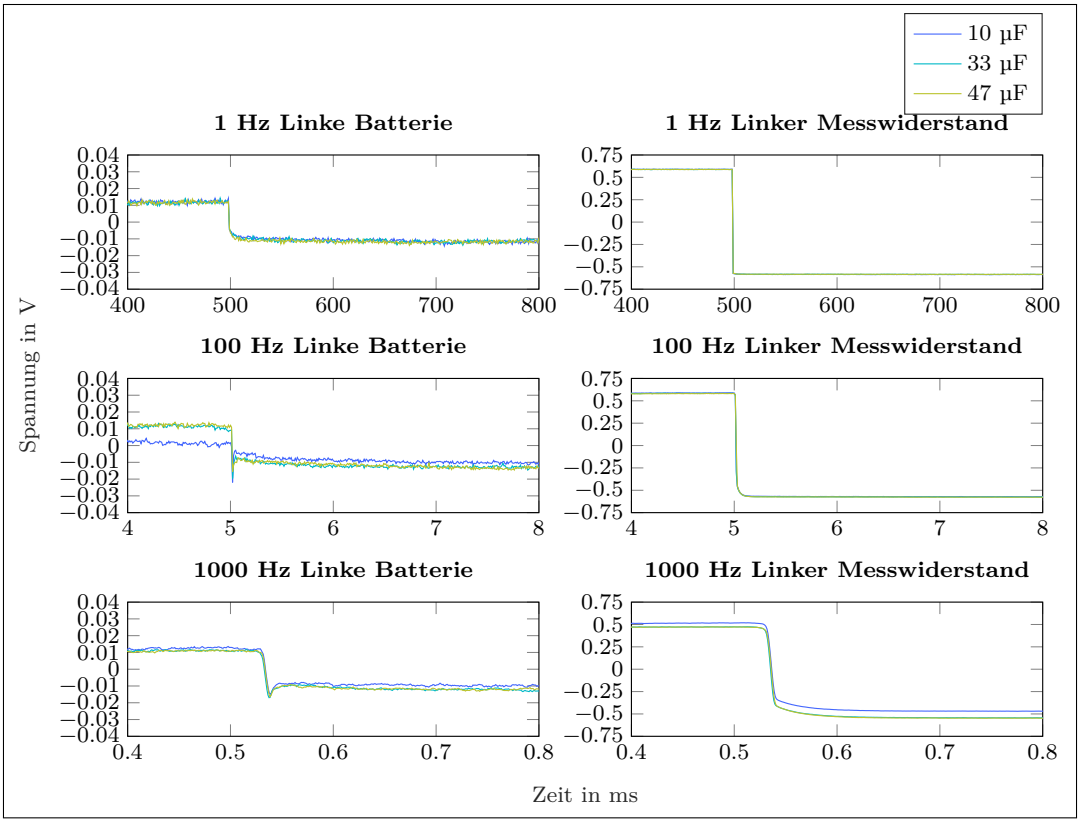


Abbildung 5.9: Fallende Flanke des Anregesignals mit Ausgangskondensatoren im Modus RL

Dabei spielt für das Abfallen des Signals auch der Wert des Kondensators eine Rolle. Ist dieser größer, beginnt auch das Abfallen der Spannung früher, um die höhere Kapazität aufladen zu können. Deswegen wurde sich auch gegen den Einsatz eines Eingangskondensators entschieden. Dieser würde dem beschriebenen Effekt entgegenwirken, indem er die Eingangsspannung des TMR stabilisiert.

Eine Betrachtung des Gesamtanregesignals in Abbildung 5.10 ergibt für beide Betriebsmodi eine deutliche Verbesserung der Signalqualität durch den Einsatz von Ausgangskondensatoren. Für die weiteren Versuche werden daher 33 µF Ausgangskondensatoren verwendet.

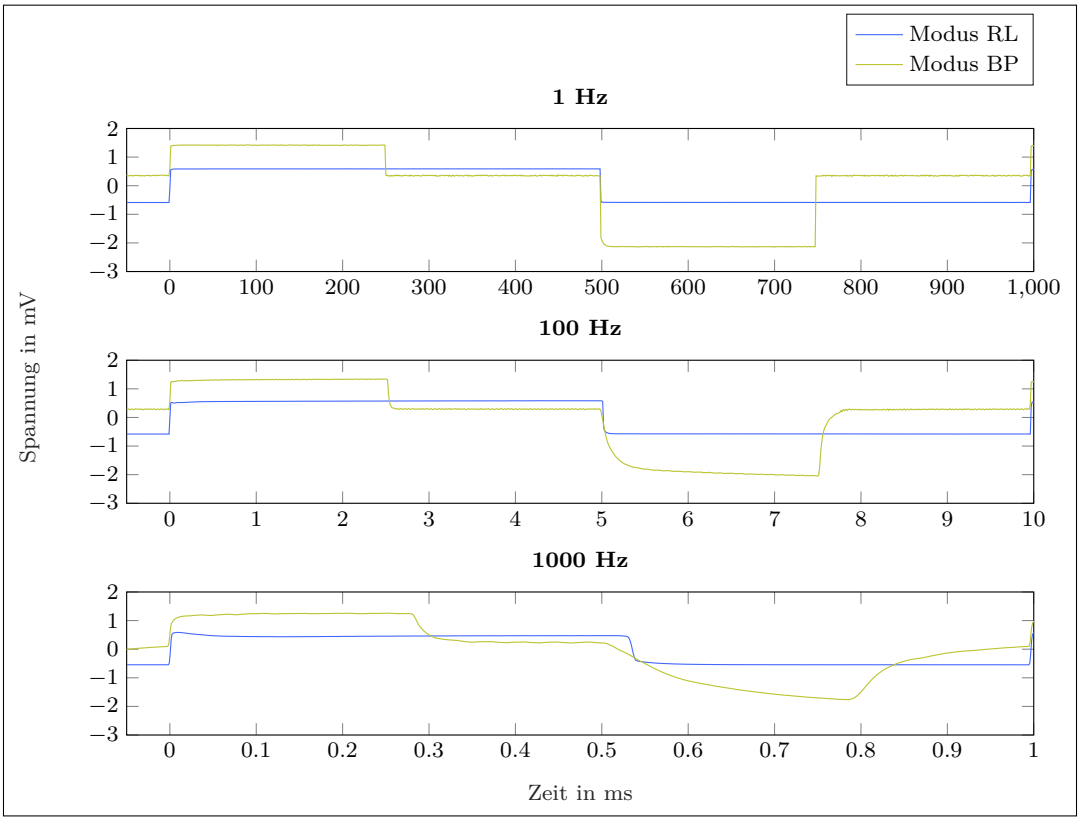


Abbildung 5.10: Eine Periode des Anregesignals mit 33 µF Kondensator im Modus RL und BP am Messwiderstand gemessen

5.3 Kontrollmessung der Frequenz und des Tastgrads

Um die korrekte Funktion der Anregeschaltung weitergehend zu testen, wird nun eine Messung der Anregefrequenzen und ihrer Tastgrade durchgeführt. Dazu wird mit einem Kanal des Oszilloskops über dem Messwiderstand auf der linken Seite gemessen. Ein weiterer Kanal dient dazu, das Signal am Optokoppler-Eingang (Pin 1 von U1 in Abbildung 3.3) zu messen. Dieses entspricht dem Ausgangssignal des Mikrocontrollers und lässt so einen Vergleich zu. Die Spannung der Batterien lag zu Beginn der Messungen bei 10,36 V links und 10,16 V rechts. Nach der Messung lag die Spannung der linken Batterie bei 10,32 V und die der rechten bei 10,10 V.

Für niedrige Frequenz entspricht dabei das Signal am Messwiderstand dem am Optokoppler wie den Abbildungen 5.11 und 5.12 entnommen werden kann.

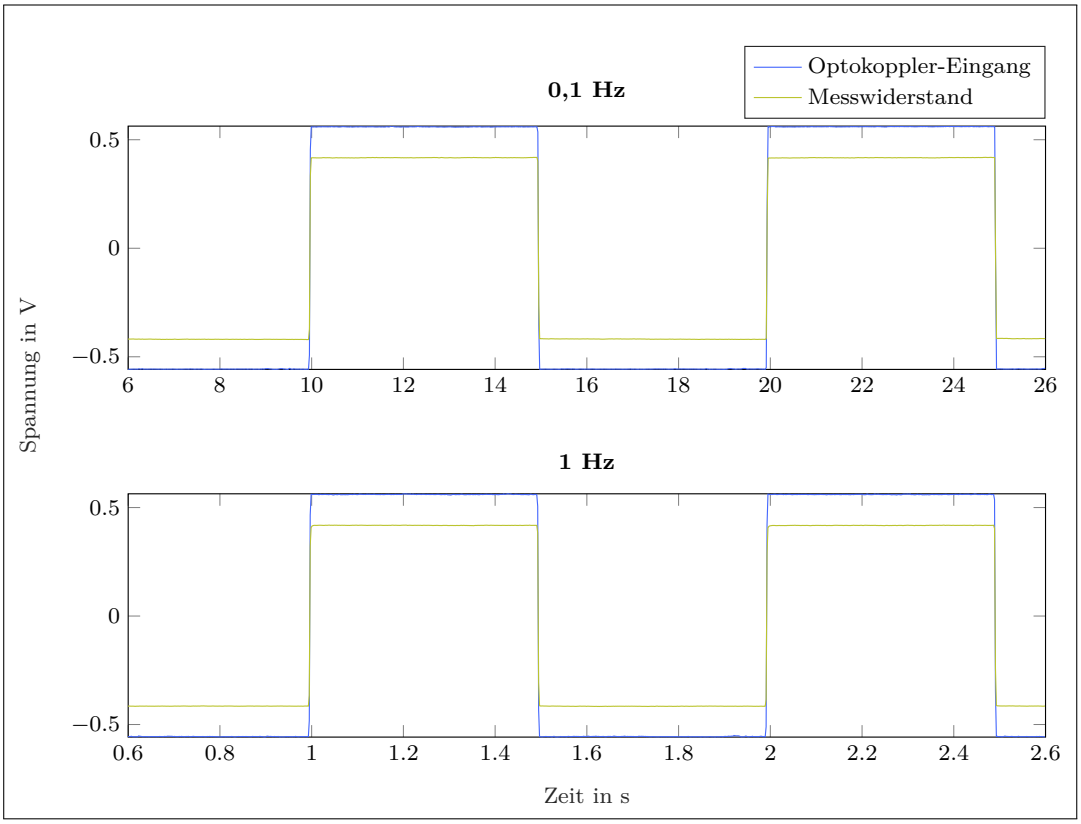


Abbildung 5.11: Anregesignal am Optokoppler und Messwiderstand für niedrige Frequenzen im Modus RL

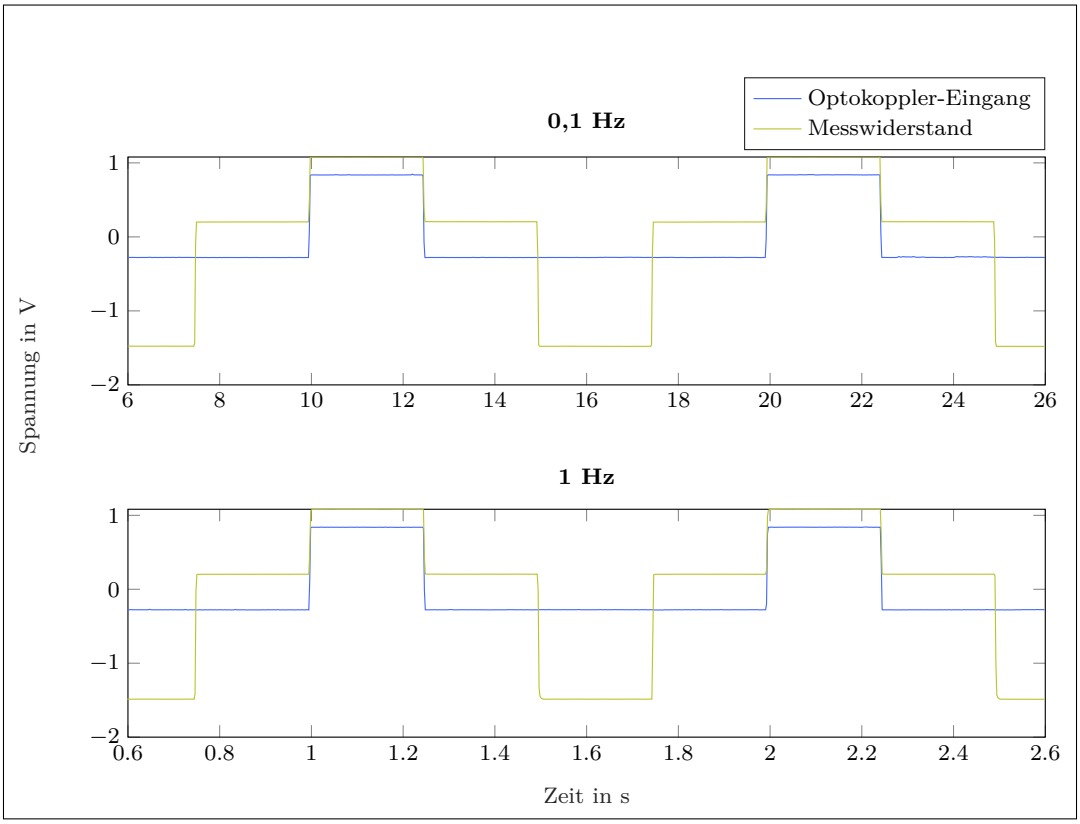


Abbildung 5.12: Anregesignal am Optokoppler und Messwiderstand für niedrige Frequenzen im Modus BP

Für hohe Frequenzen tritt jedoch eine deutliche Verzögerung auf. Wie in Abbildung 5.13 abgelesen werden kann, reagiert die Anregeschaltung bei 10 kHz mit einer Verzögerung von etwa 40 µs auf ein fallendes Signal am Optokoppler-Eingang.

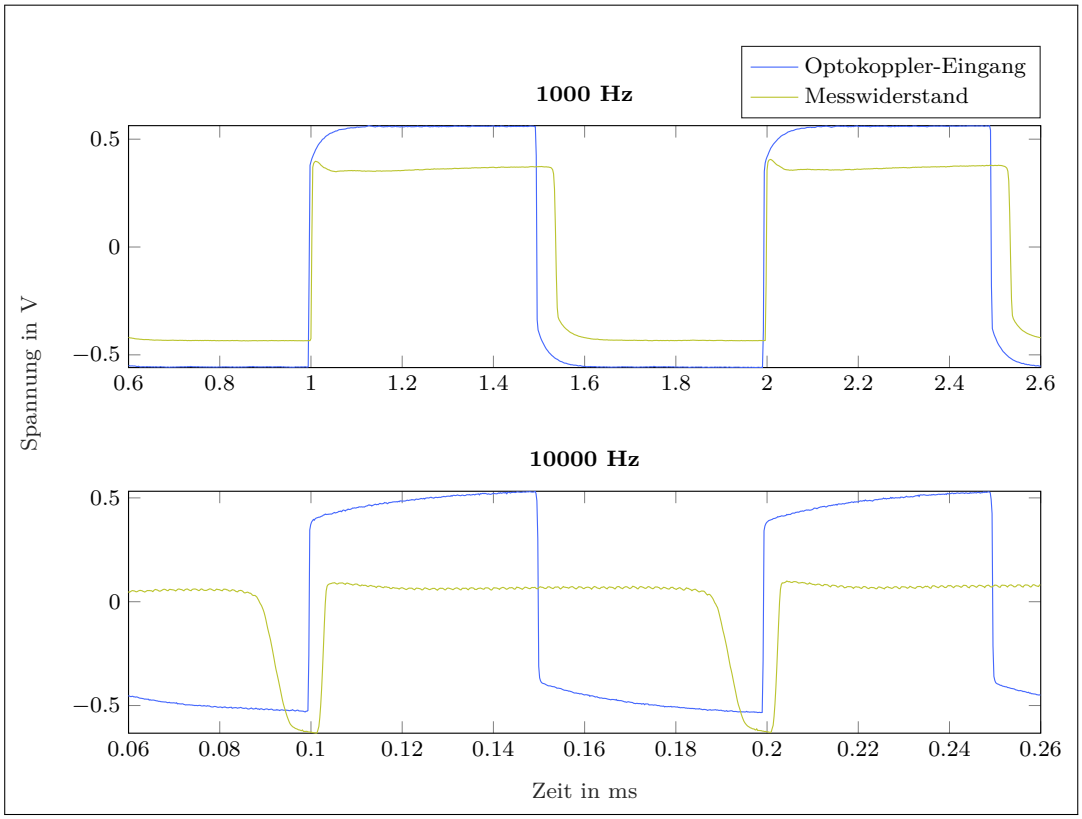


Abbildung 5.13: Anregesignal am Optokoppler und Messwiderstand für hohe Frequenzen im Modus RL

Noch schlechter ist die Übereinstimmung der Signale im Modus BP, wie in Abbildung 5.14 zu sehen ist. Hier ist der Ruhezustand zwischen den positiven und negativen Pulsen nicht mehr vorhanden, sodass ein direkter Übergang zwischen ihnen stattfindet. Für die steigenden Flanken der Signale liegt jedoch keine größere Verzögerung vor, sodass die eingestellten Frequenzen vermutlich gut eingehalten werden. Die Tastgrade hingegen werden hingegen vermutlich durch die späteren Schaltzeitpunkte vergrößert.

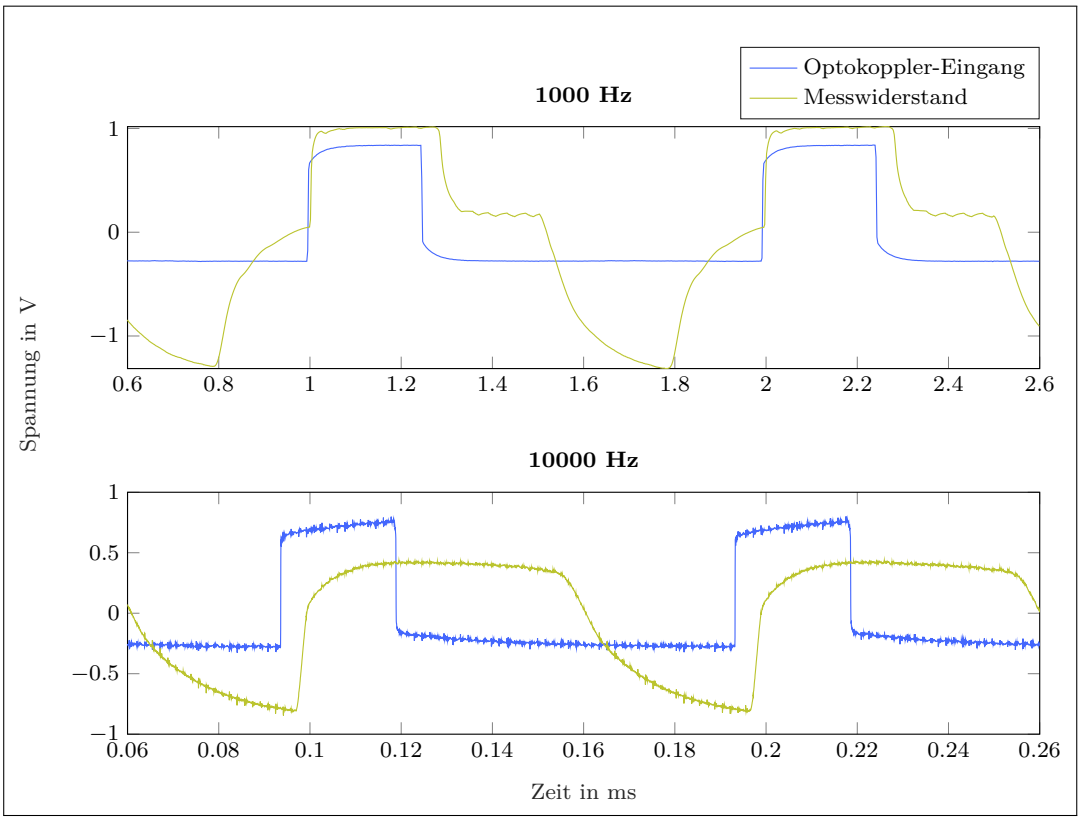


Abbildung 5.14: Anregesignal am Optokoppler und Messwiderstand für hohe Frequenzen im Modus BP

Im nächsten Schritt wird deswegen mit Matlab die Frequenz der Signale ermittelt. Dazu werden mit der Funktion findpeaks die Spitzen der Signale ermittelt und ihr Abstand gemittelt. Als Kehrwert kann dann die Frequenz erhalten werden. Die Ergebnisse sind in den Tabellen 5.2 und 5.3 aufgeführt.

Tabelle 5.2: Abweichung der Frequenzen am Optokoppler-Eingang und Messwiderstand zur eingestellten Frequenz im Modus RL

Eingestellte Anregefrequenz	Frequenz am Optokoppler-Eingang	Abweichung zu eingestellter Frequenz	Frequenz am Messwiderstand	Abweichung zu eingestellter Frequenz
100 mHz	100,4 mHz	0,4000 %	100,4 mHz	0,4000 %
400 mHz	401,3 mHz	0,3250 %	401,3 mHz	0,3250 %
800 mHz	802,3 mHz	0,2875 %	803,3 mHz	0,4125 %
1 Hz	1,0040 Hz	0,4000 %	1,0040 Hz	0,4000 %
4 Hz	4,0150 Hz	0,3750 %	4,0150 Hz	0,3750 %
8 Hz	8,0334 Hz	0,4175 %	8,0334 Hz	0,4175 %
10 Hz	10,0402 Hz	0,4020 %	10,0402 Hz	0,4020 %
40 Hz	40,1499 Hz	0,3748 %	40,1499 Hz	0,3748 %
80 Hz	80,2311 Hz	0,2889 %	80,2311 Hz	0,2889 %
100 Hz	100,3344 Hz	0,3344 %	100,3344 Hz	0,3344 %
400 Hz	401,2841 Hz	0,3210 %	401,4989 Hz	0,3747 %
800 Hz	803,3419 Hz	0,4177 %	803,3419 Hz	0,4177 %
1 kHz	1,0033 kHz	0,3345 %	1,0033 kHz	0,3345 %
4 kHz	4,0107 kHz	0,2674 %	4,0112 kHz	0,2808 %
8 kHz	8,0231 kHz	0,2888 %	8,0334 kHz	0,4177 %
10 kHz	10,033 kHz	0,3344 %	10,035 kHz	0,3512 %

Tabelle 5.3: Abweichung der Frequenzen am Optokoppler-Eingang und Messwiderstand zur eingestellten Frequenz im Modus BP

Eingestellte Anregefrequenz	Frequenz am Optokoppler-Eingang	Abweichung zu eingestellter Frequenz	Frequenz am Messwiderstand	Abweichung zu eingestellter Frequenz
100 mHz	100,3 mHz	0,3000 %	100,3 mHz	0,3000 %
400 mHz	401,3 mHz	0,3250 %	401,3 mHz	0,3250 %
800 mHz	803,3 mHz	0,4125 %	803,3 mHz	0,4125 %
1 Hz	1,0033 Hz	0,3300 %	1,0033 Hz	0,3300 %
4 Hz	4,0107 Hz	0,2675 %	4,0107 Hz	0,2675 %
8 Hz	8,0231 Hz	0,2887 %	8,0231 Hz	0,2887 %
10 Hz	10,0334 Hz	0,3340 %	10,0334 Hz	0,3340 %
40 Hz	40,1284 Hz	0,3210 %	40,1499 Hz	0,3748 %
80 Hz	80,2311 Hz	0,2889 %	80,3342 Hz	0,4177 %
100 Hz	100,3344 Hz	0,3344 %	100,3344 Hz	0,3344 %
400 Hz	401,7140 Hz	0,4285 %	401,4989 Hz	0,3747 %
800 Hz	803,3419 Hz	0,4177 %	802,3107 Hz	0,2888 %
1 kHz	1,0033 kHz	0,3345 %	1,0033 kHz	0,3345 %
4 kHz	4,0033 kHz	0,0833 %	4,0214 kHz	0,5340 %
8 kHz	8,0296 kHz	0,3704 %	8,0714 kHz	0,8920 %
10 kHz	10,053 kHz	0,5348 %	10,014 kHz	0,1389 %

Wie in den Tabellen abgelesen werden kann, stimmen die eingestellten Frequenzen gut mit den gemessenen Werten am Optokoppler-Eingang überein. Auch die Übereinstimmung mit den Werten am Messwiderstand ist sehr hoch, wobei die Abweichungen einen Wert von 1 % nie überschreiten. Dabei sind auch nur geringe Unterschiede zwischen den Frequenzwerten am Optokoppler-Eingang und dem Messwiderstand feststellbar.

Um die Korrektheit der positiven Tastgrade zu bestimmen, wird mit Matlab die Funktion dutycyle angewendet und die resultierenden Werte gemittelt. Die Ergebnisse dieser Berechnung sind in Tabelle 5.4 und 5.5 aufgeführt.

Tabelle 5.4: Abweichung der Tastgrade am Optokoppler-Eingang und Messwiderstand zum eingestellten Wert im Modus RL

Eingestellte Anregefrequenz	Tastgrad am Optokoppler-Eingang	Abweichung zu eingestelltem Tastgrad von 50 %	Tastgrad am Messwiderstand	Abweichung zu eingestelltem Tastgrad von 50 %
100 mHz	49,9969 %	−0,0062 %	49,9986 %	−0,0028 %
400 mHz	50,0025 %	0,0050 %	50,0031 %	0,0062 %
800 mHz	49,9986 %	−0,0028 %	49,9955 %	−0,0090 %
1 Hz	50,0017 %	0,0034 %	50,0018 %	0,0036 %
4 Hz	50,0008 %	0,0016 %	50,0108 %	0,0216 %
8 Hz	50,0024 %	0,0048 %	50,0160 %	0,0320 %
10 Hz	50,0009 %	0,0018 %	50,0291 %	0,0582 %
40 Hz	50,0041 %	0,0082 %	50,1411 %	0,2822 %
80 Hz	50,0061 %	0,0122 %	50,2793 %	0,5586 %
100 Hz	50,0040 %	0,0080 %	50,3552 %	0,7104 %
400 Hz	50,0225 %	0,0450 %	51,4480 %	2,8960 %
800 Hz	50,0324 %	0,0648 %	52,9599 %	5,9198 %
1 kHz	50,0391 %	0,0782 %	53,7041 %	7,4082 %
4 kHz	50,1838 %	0,3676 %	65,1149 %	30,2298 %
8 kHz	50,3529 %	0,7058 %	81,0456 %	62,0912 %
10 kHz	50,4381 %	0,8762 %	89,3482 %	78,6964 %

Tabelle 5.5: Abweichung der Tastgrade am Optokoppler-Eingang und Messwiderstand zum eingestellten Wert im Modus BP

Eingestellte Anregefrequenz	Tastgrad am Optokoppler-Eingang	Abweichung zu eingestelltem Tastgrad von 25 %	Tastgrad am Messwiderstand	Abweichung zu eingestelltem Tastgrad von 25 %
100 mHz	24,9852 %	−0,0592 %	24,9884 %	−0,0464 %
400 mHz	24,9965 %	−0,0140 %	24,9957 %	−0,0172 %
800 mHz	24,9924 %	−0,0304 %	24,9941 %	−0,0236 %
1 Hz	24,9931 %	−0,0276 %	24,9953 %	−0,0188 %
4 Hz	25,0123 %	0,0492 %	25,0173 %	0,0692 %
8 Hz	24,9968 %	−0,0128 %	25,0207 %	0,0828 %
10 Hz	25,0060 %	0,0240 %	25,0431 %	0,1724 %
40 Hz	25,0009 %	0,0036 %	25,1644 %	0,6576 %
80 Hz	25,0023 %	0,0092 %	25,3319 %	1,3276 %
100 Hz	24,9998 %	0,0001 %	25,4949 %	1,9796 %
400 Hz	25,0159 %	0,0636 %	26,7932 %	7,1728 %
800 Hz	25,0377 %	0,1508 %	28,4921 %	13,9684 %
1 kHz	25,0457 %	0,1828 %	29,4488 %	17,7952 %
4 kHz	25,1575 %	0,6300 %	40,6362 %	62,5448 %
8 kHz	25,3244 %	1,2976 %	50,9998 %	103,9992 %
10 kHz	25,4184 %	1,6736 %	55,3637 %	121,4548 %

Aus den Tabellen kann hier deutlich ein Trend abgeleitet werden. So steigen die Abweichungen der Tastgrade am Messwiderstand für steigende Anregefrequenzen stark an. Im Modus RL liegt bei einer Frequenz von 10 kHz etwa eine Abweichung von fast 80 % vor. Im Modus BP, in dem sich der Gesamttastgrad auf den positiven und negativen Puls aufteilt, sind die Abweichungen sogar noch größer. Hier liegt bei 10 kHz eine Abweichung von rund 121 % vor, was einem Tastgrad von 55 % statt den geforderten 25 % entspricht.

Da die Tastgrade am Optokoppler-Eingang für beide Modi mit maximalen Abweichungen von unter 2 % gut mit den eingestellten Werten übereinstimmen, muss die Abweichung auf dem Weg des Signals über das Schaltmodul auftreten.

5.3.1 Signallaufzeit vom Optokoppler-Eingang zum Source-Pin des MOSFET-Schaltmoduls

Um die Signallaufzeit durch das Schaltmodul zu untersuchen, wird eine Anregefrequenz von 10 kHz gewählt und ein vollständiger Signalpuls betrachtet. Dabei werden auf dem Schaltmodul Kanal 2 des Oszilloskops an den Source-Pin des MOSFET, Kanal 3 an den Ausgang und Kanal 4 an den Eingang des Optokopplers angeschlossen. So kann der Punkt ermittelt werden, an dem die Verzögerung auftritt. Der Aufbau der Messung ist dabei in Abbildung 5.15 auf der Platine und in Abbildung 5.16 als Schaltplan dargestellt. Die Spannung der Batterien lag zu Beginn der Messungen bei 10,02 V links und 9,9 V rechts. Nach der Messung lag die Spannung der linken Batterie bei 10,04 V und die der rechten bei 9,66 V.

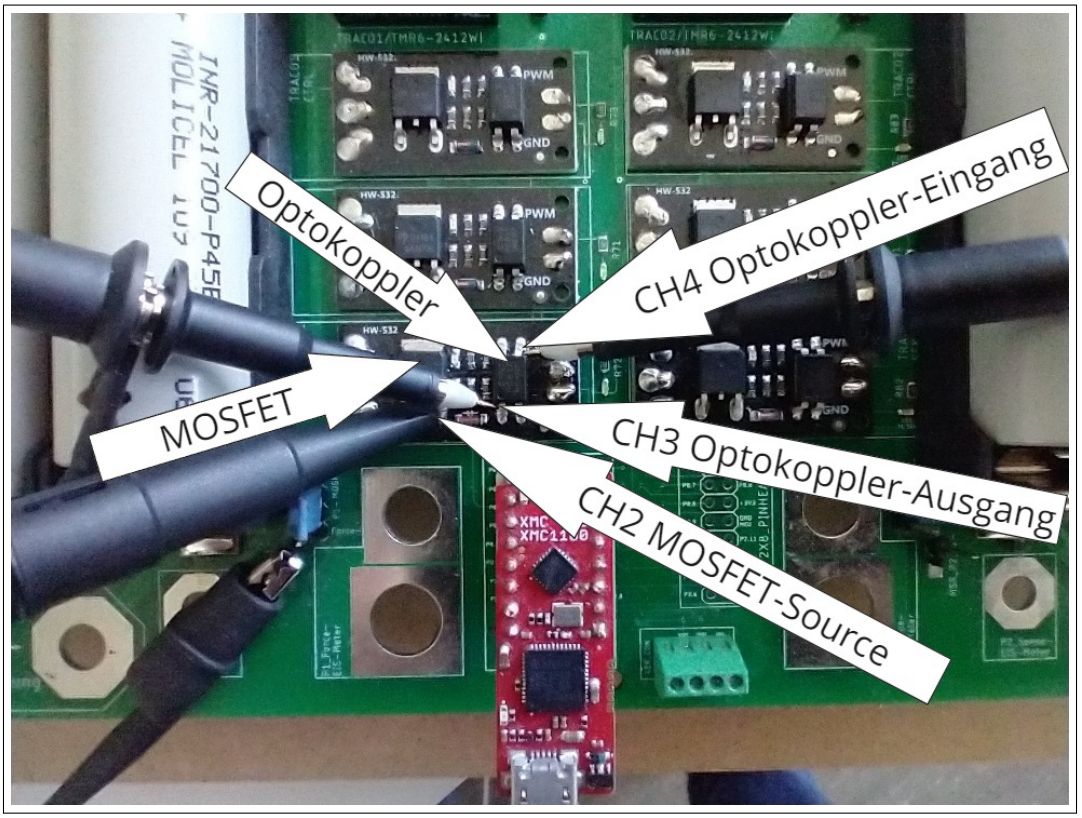


Abbildung 5.15: Messaufbau zur Messung der Signallaufzeit über das Schaltmodul

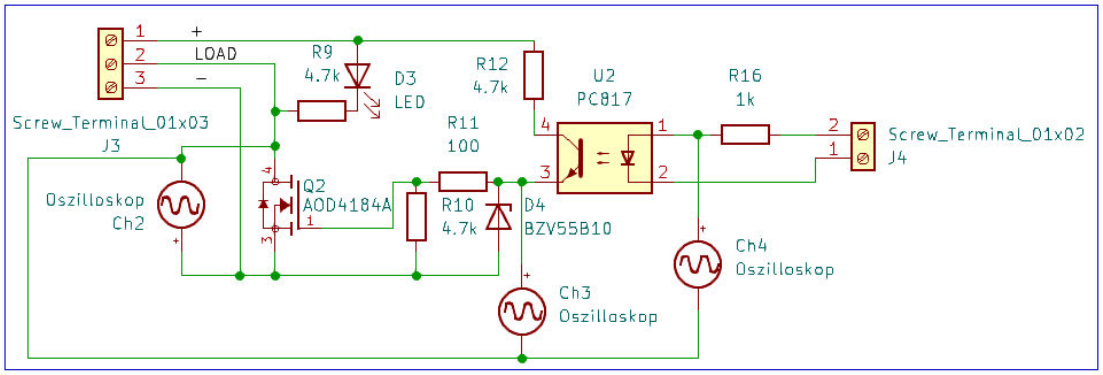


Abbildung 5.16: Schaltbild des Messaufbaus zur Messung der Signallaufzeit über das Schaltmodul

In den Abbildungen 5.17 und 5.18 werden die Messergebnisse dargestellt.

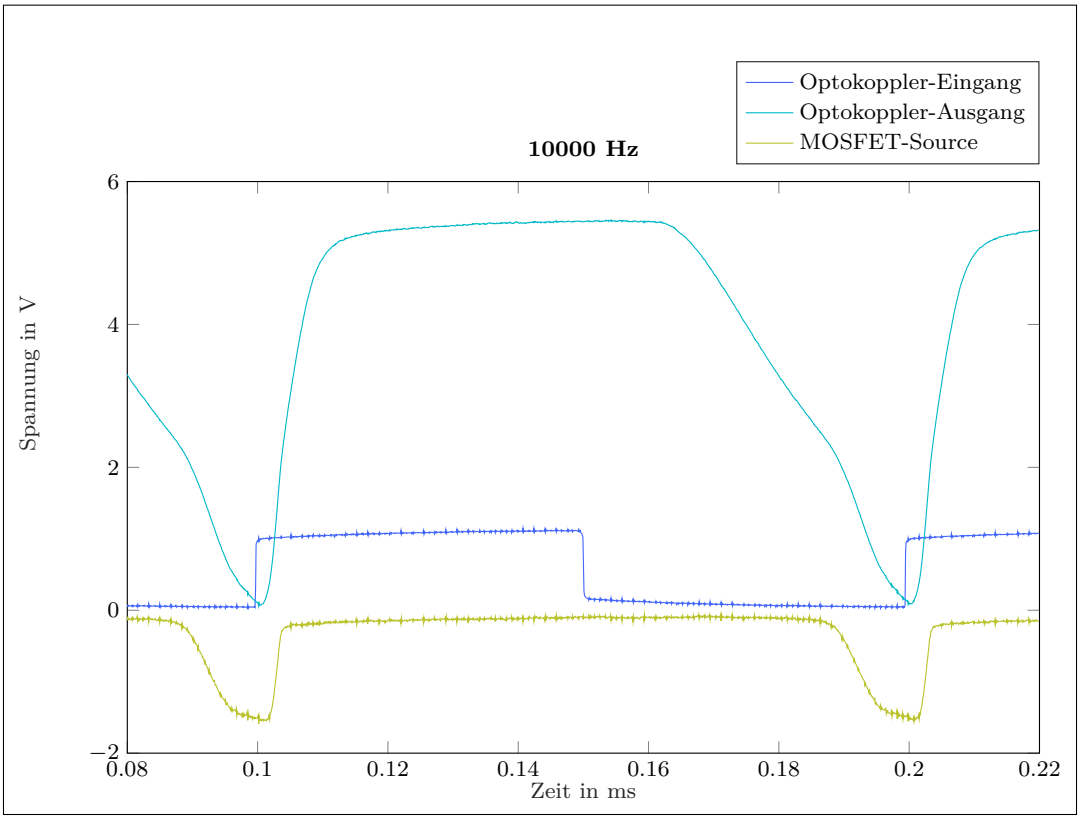


Abbildung 5.17: Signallaufzeit vom Optokoppler-Eingang zum MOSFET-Source bei 10 kHz im Modus RL

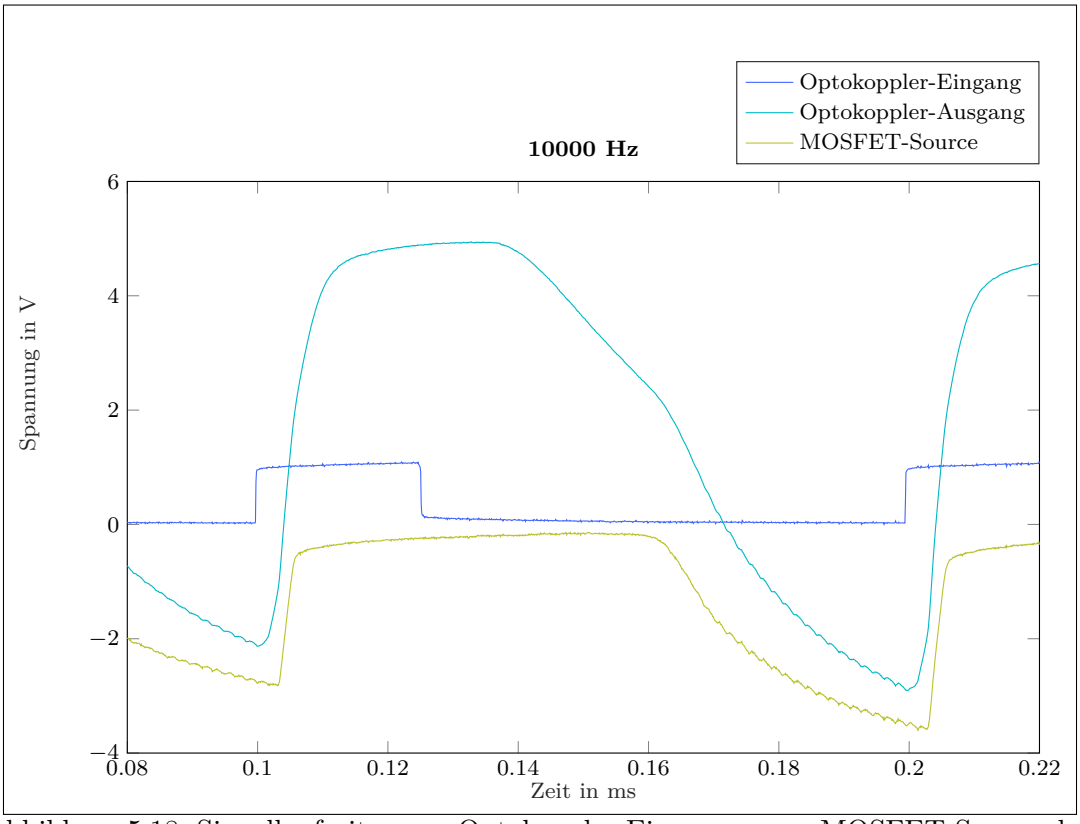


Abbildung 5.18: Signallaufzeit vom Optokoppler-Eingang zum MOSFET-Source bei 10 kHz im Modus BP

Sowohl in Abbildung 5.17 als auch in 5.18 ist deutlich zu erkennen, wie die Verzögerung zustande kommt. So reagieren Optokoppler-Ausgang und MOSFET Source mit nur kurzer Verzögerung auf ein Ansteigen des Signals am Optokoppler-Eingang. So steigt das Signal am Optokoppler-Ausgang ebenfalls, was den MOSFET leitend macht, wodurch das Potenzial an seinem Source auf Masse liegt und dementsprechend Richtung 0 V steigt. Anders sieht es an der fallenden Flanke aus. Hier liegt zunächst eine Abschaltverzögerung von rund 20 µs vor, bis der Ausgang des Optokopplers auf ein fallendes Signal am Eingang reagiert. Zusätzlich kommt noch die Abfallzeit des Optokopplers hinzu. So fällt das Signal, wie in Abbildung 5.17 zu sehen ist, in einem Zeitraum von etwa 30 µs auf 0 V ab. Auch wenn der MOSFET schon bei einer Spannung von etwa 2 V damit beginnt zu sperren, ist dies eine weitere Ursache für eine Signalverzögerung.

Ursächlich für diese Verzögerung dürfte der Phototransistor im Optokoppler sein, der sich nach Abschalten der LED nicht schnell genug entlädt. Auch der mit 4,7 kΩ recht große Widerstand (siehe Abbildung 3.3), über den der Phototransistor entladen wird ist problematisch. So steigt die Abfallzeit mit steigendem Lastwiderstand an [18].

Eine mögliche Maßnahme, um das Problem der Abschaltverzögerung des Optokopplers zu kompensieren, ist das Einstellen kleinerer Tastgrade, was mit der verwendeten Software leicht möglich ist. Aufgrund dieser Tatsache ist dies die für diese Arbeit gewählte Lösung. Als weitere Verbesserung könnte der Lastwiderstand des Optokoppler verkleinert werden. Auch der Einsatz eines anderen, schneller schaltenden, Optokopplers ist eine Möglichkeit. Erste Versuche der Forschungsgruppe hierzu zeigen, das mit anderen Optokopplern eine deutliche Verbesserung erreicht werden kann.

5.3.2 Kompensation der Tastgrad Abweichungen über die Einstellung des Tastgrads im Mikrocontroller

Um die vorhergehend beschriebene Kompensation des Tastgrads zu erreichen, wird in diesem Schritt das Signal am Messwiderstand mit dem Oszilloskop gemessen. Dabei wird dann der Tastgrad so lange reduziert, bis die kleinst mögliche Abweichung erreicht wird. Die aufgezeichneten Signale werden dann erneut mit Matlab geprüft. Modus LR wird ebenfalls geprüft, um eventuelle Bauteilunterschiede zu berücksichtigen. Die Spannung der Batterien lag zu Beginn der Messungen bei 10,35 V links und 10,15 V rechts. Nach der Messung lag die Spannung der linken Batterie bei 10,24 V und die der rechten bei 10,23 V. In den Tabellen 5.6, 5.7 und 5.8 sind die Ergebnisse aufgeführt.

Tabelle 5.6: Tastgrade nach Kompensierung der Abweichungen im Modus LR

Eingestellte Anregefrequenz	Eingestellter Tastgrad	Tastgrad am Messwiderstand	Abweichung zu gefordertem Tastgrad
100 mHz	50 %	50,0033 %	0,0066 %
400 mHz	50 %	50,0022 %	0,0044 %
800 mHz	50 %	50,0015 %	0,0030 %
1 Hz	50 %	49,9985 %	−0,0030 %
4 Hz	50 %	50,0049 %	0,0098 %
8 Hz	50 %	50,0174 %	0,0348 %
10 Hz	50 %	50,0254 %	0,0508 %
40 Hz	50 %	50,1300 %	0,2600 %
80 Hz	50 %	50,2793 %	0,5586 %
100 Hz	50 %	50,3516 %	0,7032 %
400 Hz	49 %	50,4344 %	0,8688 %
800 Hz	47 %	49,8728 %	−0,2544 %
1 kHz	46 %	49,6026 %	−0,7948 %
4 kHz	35 %	49,5670 %	−0,8660 %
8 kHz	21 %	49,7118 %	−0,5764 %
10 kHz	16 %	49,2165 %	−1,5670 %

Tabelle 5.7: Tastgrade nach Kompensierung der Abweichungen im Modus RL

Eingestellte Anregefrequenz	Eingestellter Tastgrad	Tastgrad am Messwiderstand	Abweichung zu gefordertem Tastgrad
100 mHz	50 %	49,9996 %	−0,0008 %
400 mHz	50 %	50,0025 %	0,0050 %
800 mHz	50 %	50,0012 %	0,0024 %
1 Hz	50 %	49,9987 %	−0,0026 %
4 Hz	50 %	50,0149 %	0,0298 %
8 Hz	50 %	50,0157 %	0,0314 %
10 Hz	50 %	50,0308 %	0,0616 %
40 Hz	50 %	50,1433 %	0,2866 %
80 Hz	50 %	50,2826 %	0,5652 %
100 Hz	50 %	50,3505 %	0,7010 %
400 Hz	49 %	50,4749 %	0,9498 %
800 Hz	47 %	49,9636 %	−0,0728 %
1 kHz	46 %	49,7202 %	−0,5596 %
4 kHz	35 %	50,0274 %	0,0548 %
8 kHz	20 %	49,7304 %	−0,5392 %
10 kHz	16 %	50,3483 %	0,6966 %

Tabelle 5.8: Tastgrade nach Kompensierung der Abweichungen im Modus BP

Eingestellte Anregefrequenz	Eingestellter Tastgrad	Tastgrad am Messwiderstand	Abweichung zu gefordertem Tastgrad
100 mHz	50 %	25,0056 %	0,0224 %
400 mHz	50 %	24,9892 %	−0,0432 %
800 mHz	50 %	24,9953 %	−0,0188 %
1 Hz	50 %	25,0042 %	0,0168 %
4 Hz	50 %	25,0174 %	0,0696 %
8 Hz	50 %	25,0319 %	0,1276 %
10 Hz	50 %	25,0419 %	0,1676 %
40 Hz	50 %	25,1655 %	0,6620 %
80 Hz	49 %	24,7982 %	−0,8072 %
100 Hz	49 %	24,8695 %	−0,5220 %
400 Hz	47 %	25,0005 %	0,0020 %
800 Hz	44 %	25,0304 %	0,1216 %
1 kHz	42 %	24,8237 %	−0,7052 %
4 kHz	21 %	25,2053 %	0,8212 %
8 kHz	13 %	24,3264 %	−2,6944 %
10 kHz	14 %	26,2874 %	5,1496 %

Aus den Messungen geht deutlich hervor, dass sich die Tastgrade der Signale durch die Kompensation deutlich verbessert haben. So beträgt die maximale Abweichung in den MP-Modi nur noch -1,567 % bei 10 kHz im Modus RL.

Im Modus BP treten mit -2,6944 und 5,1494 % bei 8 bzw. 10 kHz noch größere Abweichungen auf, was durch die schwerere Einstellung durch die Aufteilung des eingestellten Tastgrads resultiert.

Vermutlich könnte eine noch bessere Kompensation des Tastgrads erreicht werden, indem man ihn noch feiner einstellt. Dafür wäre es jedoch nötig Nachkommastellen über die UART-Steuerung zu übertragen. Diese zusätzliche Komplexität der Software wurde aufgrund der erreichten guten Genauigkeit der Tastgrade als nicht notwendig erachtet.

5.4 Spannungsdrift der Batterien bei großen Periodenzahlen

Da bei der Bestimmung der Ausgangskondensatoren eine Änderung der Batteriespannungen über den Messvorgang feststellbar war, sollen im Folgenden die Auswirkungen einer großen Zahl von Messperioden untersucht werden. Dazu werden für 200 Perioden und die Anregefrequenzen 100 mHz und 10 kHz die Spannungen an den Batterien aufgezeichnet. Als Messaufbau wird der in den Abbildungen 5.1 und 5.2 für Versuch 5.2 dargestellte verwendet.

In Abbildung 5.19 ist für den Modus RL deutlich zu erkennen, dass die Spannung der Batterien bei 100 mHz einer Drift unterliegt. So steigt die Spannung links von 10,35 auf etwa 10,4 V an, während sie im gleichen Zeitraum rechts von 10,2 auf 9,7 V absinkt. Bei 10 kHz ist hingegen kaum eine Änderung der Spannung zu erkennen.

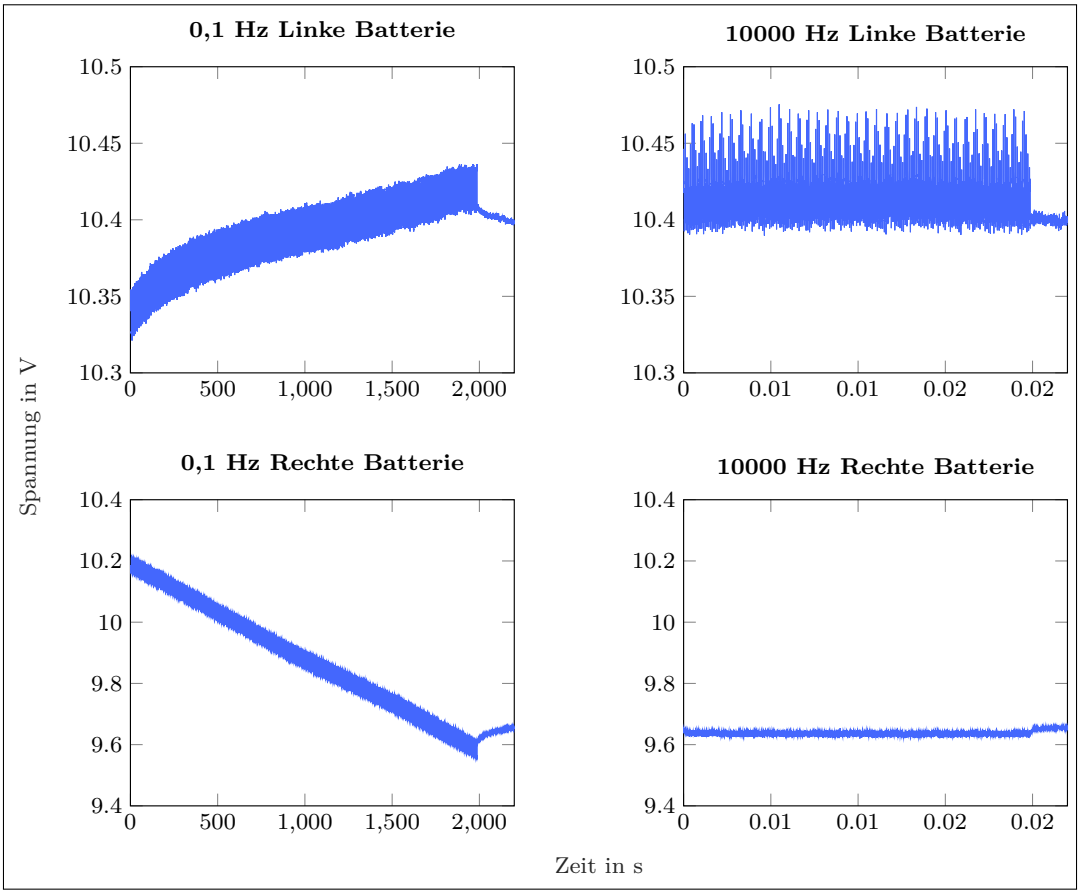


Abbildung 5.19: Spannungsverläufe der Batterien für 200 Perioden im Modus RL

In Abbildung 5.20 ist für den Modus BP zu sehen, dass die Batterien in Summe mehr entladen als geladen werden. So besitzen beide zu Beginn eine Spannung von etwa 10,2 V, die im Laufe der Anregung mit 100 mHz auf 10 V links und 10,1 V rechts absinkt. Der Spannungsverlust fällt also geringer aus, da die Batterien im Wechsel geladen und entladen werden. Durch Verluste in der Schaltung geht jedoch Energie verloren, weshalb die Batterien beide an Spannung verlieren. Bei 10 kHz hingegen ist wieder keine größere Spannungsänderung zu erkennen.

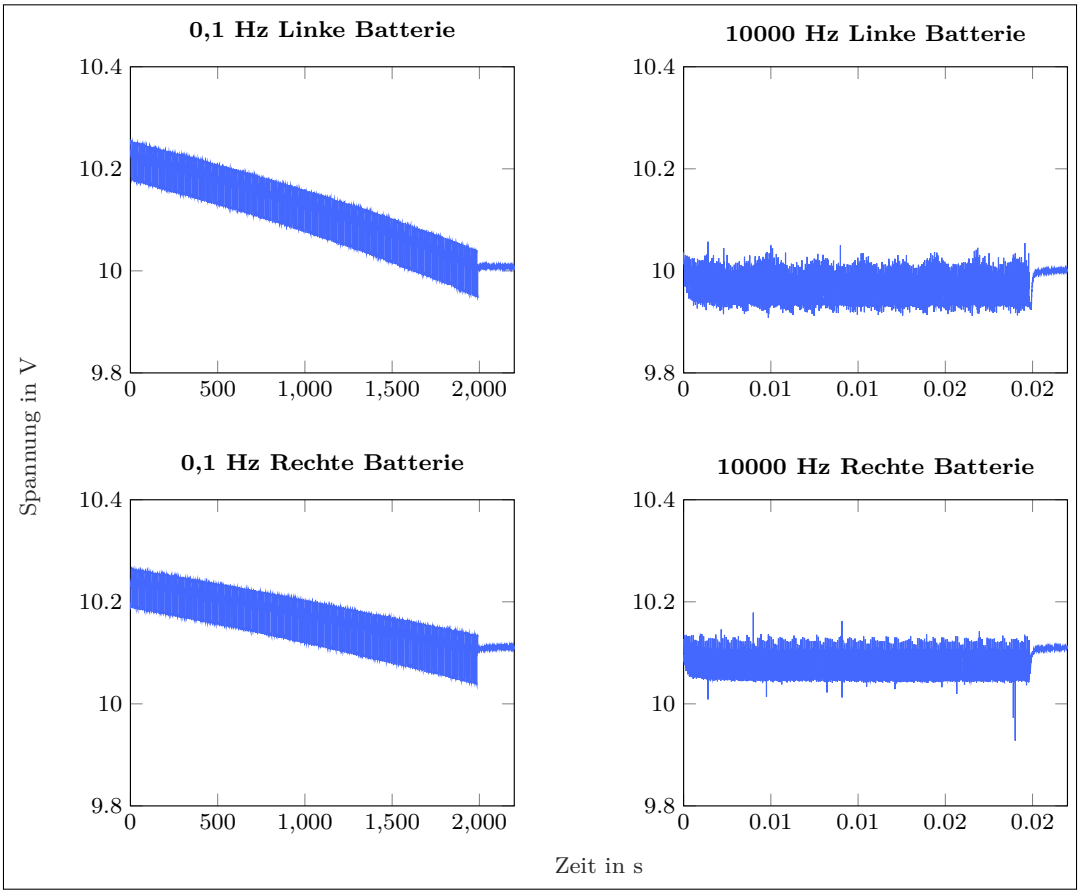


Abbildung 5.20: Spannungsverläufe der Batterien für 200 Perioden im Modus BP

5.4.1 Ladungsfluss

Da der Ladezustand der Batterie Einfluss auf die bestimmte Impedanz hat [5], wird der Ladungsfluss der Batterien untersucht, um so festzustellen, wie groß der Einfluss auf Messergebnisse ist.

Dazu werden die Spannungen, die über den Messwiderstand abfallen, ermittelt. Sie können bestimmt werden, indem die Spannungen, die mit dem Oszilloskop an der Batterie gemessen werden, von den Spannungen am Messwiderstand abgezogen werden. Nun können die ermittelten Spannungen durch den Wert des Messwiderstands von 2,2 Ω dividiert werden und so die fließenden Ströme ermittelt werden. Durch eine Multiplikation mit dem Abtastintervall kann nun die Ladung in Coulomb zu jedem Zeitpunkt bestimmt werden. Eine Summierung aller Werte ergibt dann den gesamten Ladungsfluss. Die Berechnungen wurden dabei mit Matlab durchgeführt und die Ergebnisse in Tabelle 5.9 aufgeführt.

Tabelle 5.9: Ladungsflüsse der Batterien für 100 mHz und 10 kHz in den Betriebsmodi RL und BP

	100 mHz		10 kHz	
Modus	Mittlerer Ladungsfluss linke Batterie	Mittlerer Ladungsfluss rechte Batterie	Mittlerer Ladungsfluss linke Batterie	Mittlerer Ladungsfluss rechte Batterie
RL	389,9181 C	−722,4451 C	0,0039 C	−0,0072 C
BP	−185,3995 C	−230,1997 C	−0,0019 C	−0,0023 C

Die Ladungsflüsse bestätigen die Ergebnisse der vorherigen Beobachtungen für die Anregefrequenz 100 mHz. So beträgt der Anteil der Ladung, im Modus RL, die der linken Batterie zugeführt wird, nur etwa 54 % der Ladung, die die rechte Batterie im gleichen Zeitraum verliert. Im Modus BP hingegen verlieren beide Batterien Ladung, wobei die linke Batterie nur etwa 83 % so viel verliert wie die rechte. Dieser Unterschied könnte aus unterschiedlichen Ladezuständen, Widerstandswerten und sonstigen Abweichungen der Anregeschaltung resultieren.

Der Gesamtverlust an Ladung ist im Modus BP mit 415,5592 C größer als im Modus RL mit 332,4641 C. Die etwa 25 % größeren Verluste resultieren vermutlich aus der gleichzeitigen Versorgung der beiden TMR.

Die Anregefrequenz spielt bei den Ladungsverlusten keine größere Rolle, so sind alle Ladungsflüsse bei 10 kHz etwa 0,01 % so groß wie bei 100 mHz. Dieses Verhältnis ist also ebenso groß wie das der Anregefrequenzen zueinander.

Um eine Aussage über den Einfluss dieser Ladungen auf die Messergebnisse zu geben, sollte zunächst die Kapazität der Batterie in Coulomb ermittelt werden. Im Datenblatt ist für die verwendeten Zellen eine Kapazität von 4,5 Ah angegeben [7]. Die Ladung ergibt sich dann mit $4{,}5\ \text{Ah} \cdot 3600 = 16\,200$ C. Selbst der größte Ladungsfluss mit 722,4451 C macht also nur etwa 4,46 % der Gesamtkapazität der Batterie aus. Deshalb kann die Ladungsänderung der Batterien für die weiteren Messungen mit der Anregeschaltung eher vernachlässigt werden. Um den Einfluss auf die Messungen weiter zu reduzieren, sollte mit den hochfrequenten Anregefrequenzen begonnen werden. So fallen die Veränderungen der Ladezustände der Batterien für hohe Frequenzen kleiner aus, was eine von ihnen verursachte Abweichung der Messwerte minimiert.

Sollten jedoch später größere Batterien mit einem ähnlichen Ansatz vermessen werden, müssen die Ladungsumsätze bei größeren Spannungen und Strömen beachtet werden. Deswegen sollte statt des spannungsgeregelten TMR eine Versorgung gewählt werden, die stromgeregelt ist. So ließe sich der Ladezustand der Batterie halten. [5].

5.5 Vergleich der Messungen mit AC- und DC-Kopplung des Oszilloskops

Um zu entscheiden, ob die Messungen am Oszilloskop mit AC oder DC-Kopplung durchgeführt werden, wird ein Versuch mit beiden durchgeführt. Dazu werden für die Modi RL und BP Messungen mit verschiedenen Anregefrequenzen und jeweils 10 Perioden durchgeführt und die Signale verglichen. Als Messaufbau wird der in den Abbildungen 5.1 und 5.2 für Versuch 5.2 dargestellte verwendet. Die Spannung der Batterien lag zu Beginn der Messungen bei 10,17 V links und 10,16 V rechts. Nach der Messung lag die Spannung der linken und rechten Batterie bei 9,94 V.

5.5.1 Signal Vergleich

Im ersten Schritt sind hier die Signalverläufe am linken Messwiderstand für niedrige Frequenzen in den Abbildungen 5.21 und 5.22 dargestellt. Dabei wird den Signalen der DC-Kopplung mittels Matlab der Gleichanteil abgezogen, um eine bessere Vergleichbarkeit zu erreichen.

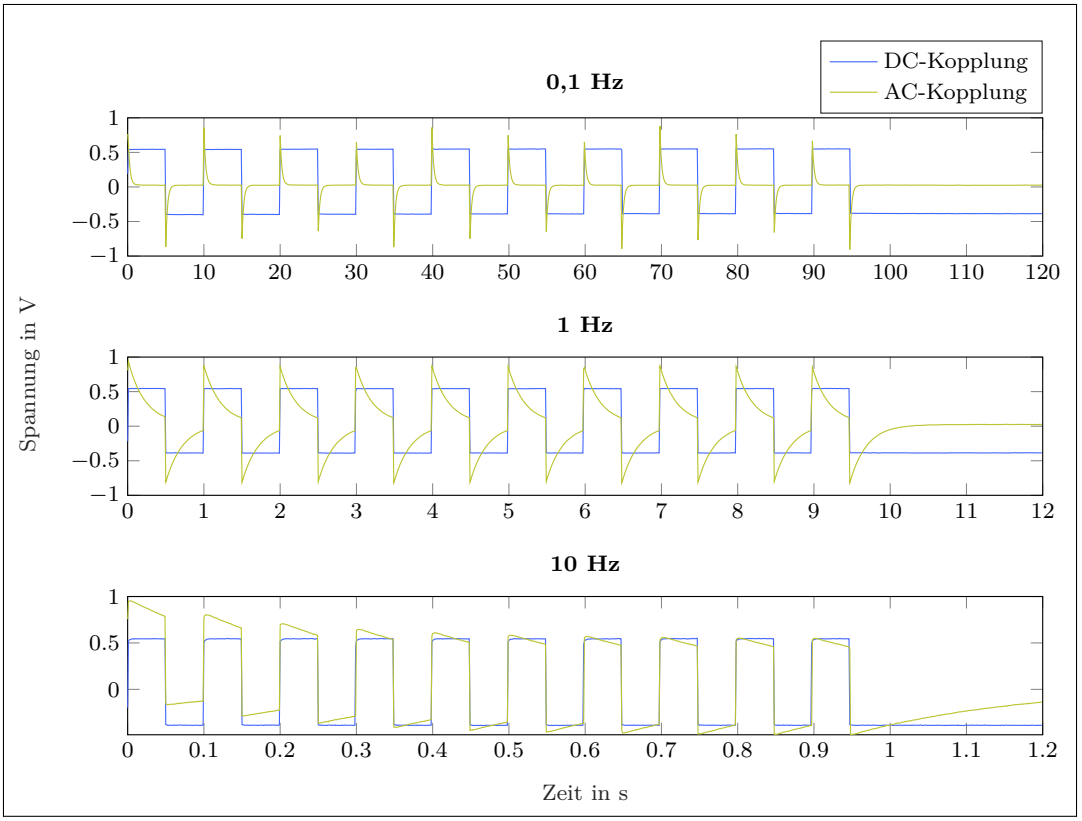


Abbildung 5.21: Signalvergleich AC- und DC-Kopplung für niedrige Frequenzen im Modus RL

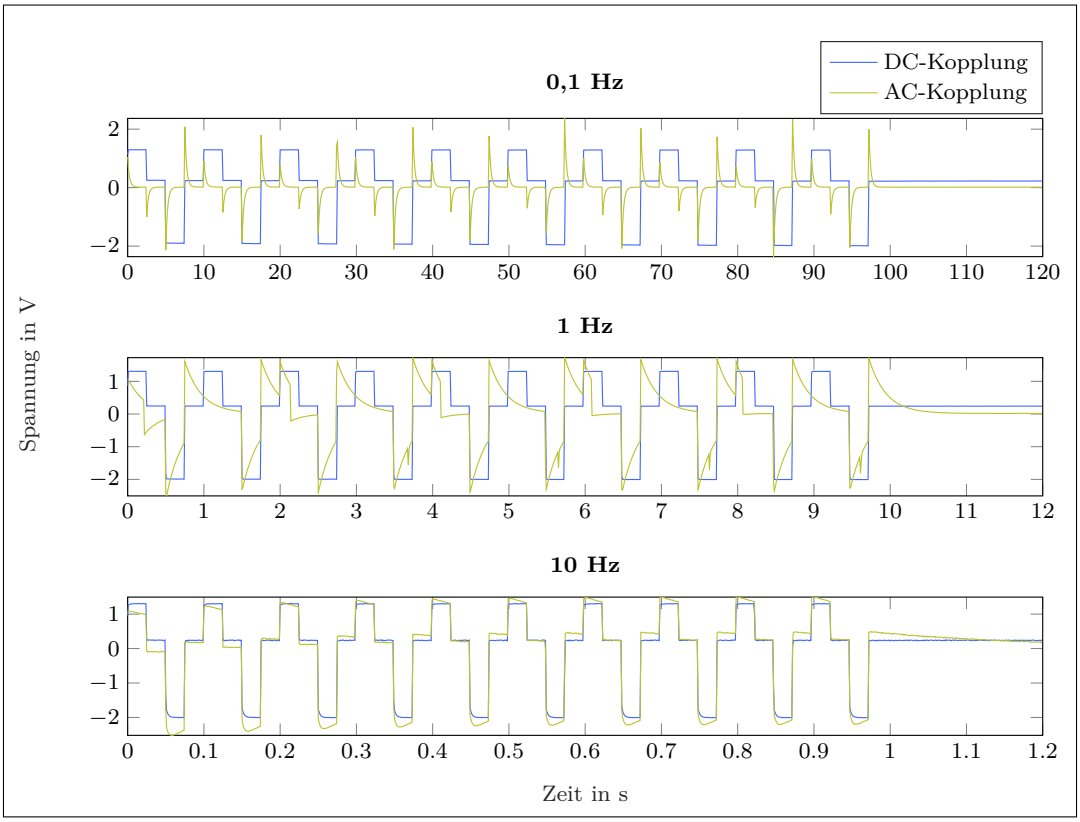


Abbildung 5.22: Signalvergleich AC- und DC-Kopplung für niedrige Frequenzen im Modus BP

Aus den Abbildungen kann abgeleitet werden, dass die Signale bei AC-Kopplung für niedrige Frequenzen deutlich verzerrt werden. Dabei ist erst ab 10 Hz kein größerer Unterschied mehr zu erkennen.

5.5.2 Vergleich der Frequenzspektren

Um weitere Aussagen treffen zu können, sollen hier die einseitigen Frequenzspektren der Signale am linken Messwiderstand verglichen werden. Diese werden mittels Matlab aus den gemessenen Signalen ermittelt und sind in den Abbildungen 5.23 und 5.24 dargestellt.

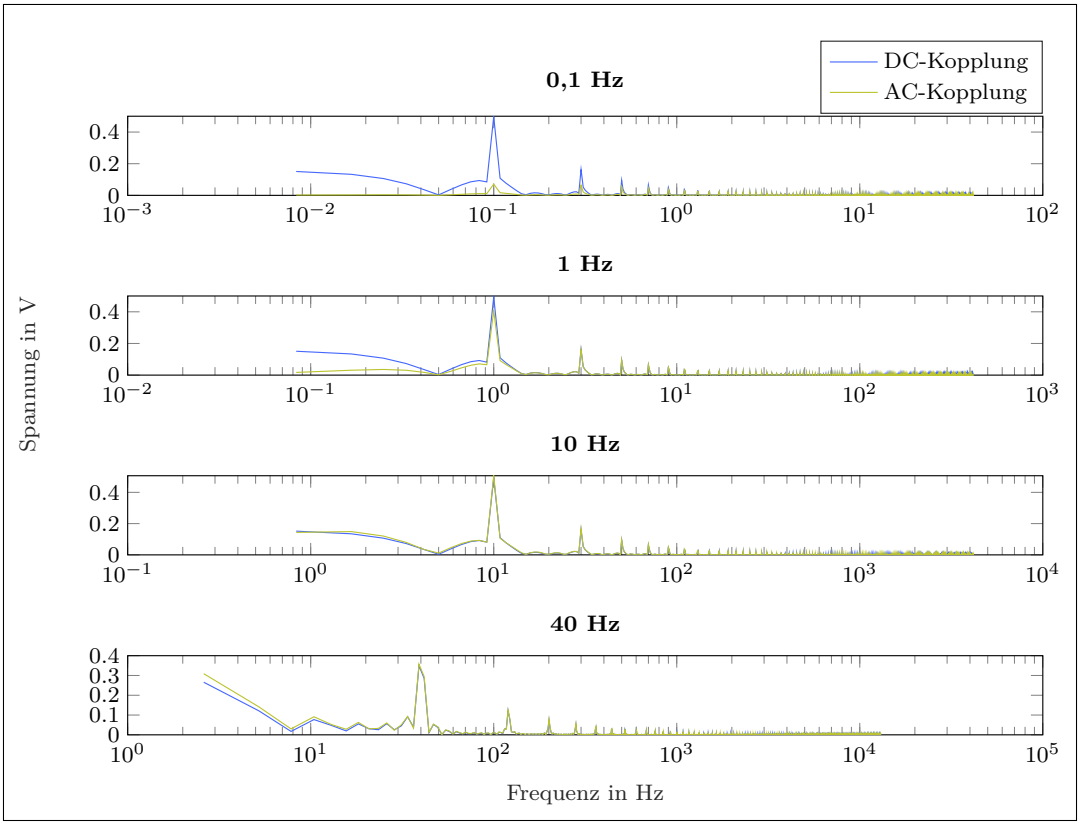


Abbildung 5.23: Frequenzspektren für AC- und DC-Kopplung für niedrige Frequenzen im Modus RL

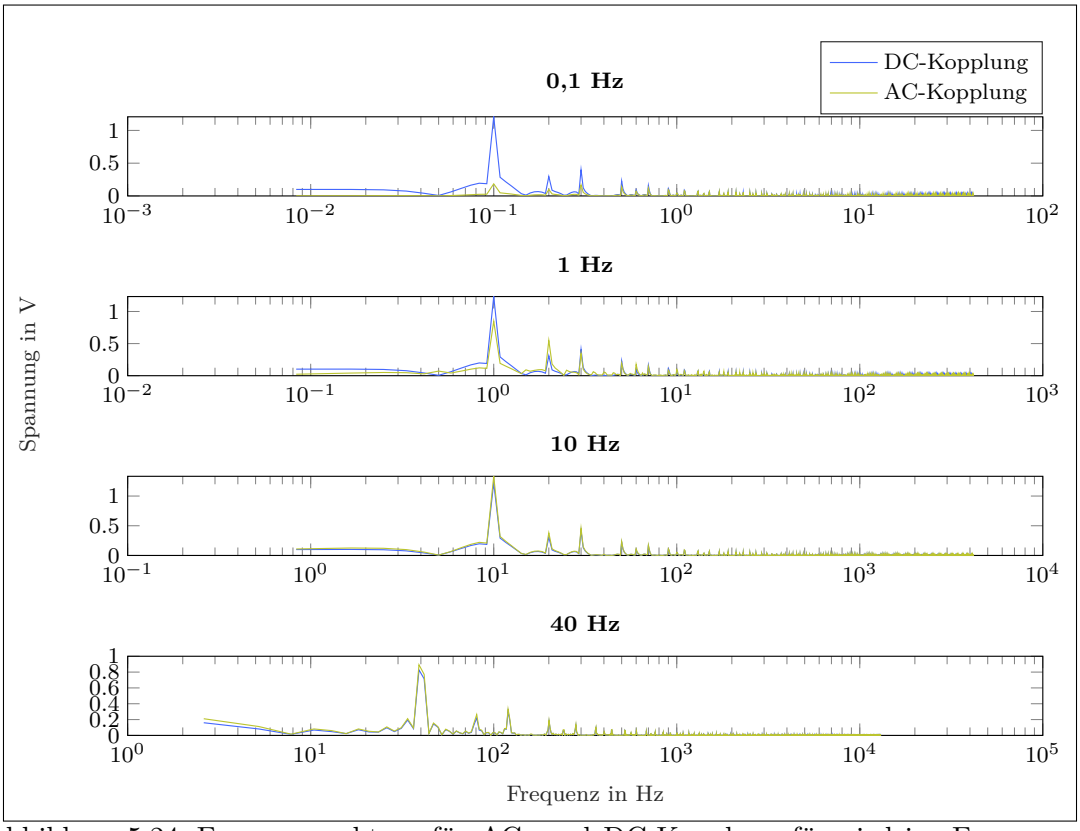


Abbildung 5.24: Frequenzspektren für AC- und DC-Kopplung für niedrige Frequenzen im Modus BP

Wie auch bei den Signalverläufen weisen die Frequenzspektren deutliche Unterschiede auf. So fällt die Amplitude für die AC-Kopplung deutlich gedämpft aus. Diese geringeren Amplituden können bei späteren Berechnungen zu Problemen führen.

Die Dämpfung der Signale resultiert aus dem RC-Tiefpass, der im Oszilloskop zum Unterdrücken des Gleichanteils genutzt wird. Allerdings liegt die Grenzfrequenz wohl um 10 Hz, da Signale unter diesem Wert ebenfalls betroffen sind. Aus diesem Grund wird für die weiteren Messungen die DC-Kopplung verwendet.

5.6 Bestimmung des Wirkungsgrads der Anregeschaltung

Um festzustellen, wie hoch der Wirkungsgrad der Anregeschaltung ist, wird er mithilfe von Matlab bestimmt. Dazu wird der gemittelte Strom, der über den Messwiderstand fließt, mit der gemittelten Spannung, die an der Batterie gemessen wird, multipliziert. Auf diese Weise wird für Links und Rechts die Leistung an der Batterie ermittelt und anschließend der Wirkungsgrad durch die Divison des linken durch den rechten Wert bestimmt. Diese Methode konnte allerdings nicht für den Modus BP verwendet werden, da sich die Berechnung für den Modus BP aufgrund der wechselenden Auf- und Entladevorgänge nicht einfach realisieren ließ. Für diesen Modus wurde stattessen aus dem mittleren Bereich der Messung die Spitzenwerte ermitteln und jeweils das Minimum der einen Seite durch das Maximum der anderen Seite dividiert. Die Ergebnisse sind in Tabelle 5.10 Als Grundlage für die Messung dienen die Daten der Messung der DC-Kopplung aus dem vorherigen Versuch.

Tabelle 5.10: Wirkungsgrad der Anregeschaltung für die Modi RL und BP

Anregefrequenz	Modus RL Wirkungsgrad	Modus BP Wirkungsgrad von Links nach Rechts	Modus BP Wirkungsgrad von Rechts nach Links
100 mHz	55.5068 %	45.3430 %	45.1139 %
400 mHz	55.7309 %	44.3437 %	44.4528 %
800 mHz	55.2887 %	44.1530 %	44.2822 %
1 Hz	54.9744 %	44.0230 %	44.2058 %
4 Hz	55.3304 %	43.7921 %	43.9611 %
8 Hz	55.6462 %	43.9716 %	44.1945 %
10 Hz	55.4075 %	43.9489 %	44.0983 %
40 Hz	56.2483 %	44.2324 %	44.2344 %
80 Hz	56.3786 %	45.2261 %	45.4157 %
100 Hz	56.4791 %	45.5918 %	45.8257 %
400 Hz	57.7943 %	46.6673 %	46.9628 %
800 Hz	59.9245 %	50.7590 %	53.7900 %
1 kHz	60.8801 %	51.4218 %	52.9555 %

Aus der Tabelle kann deutlich abgelesen werden, dass die Schaltung für den Modus RL einen minimalen Wirkungsgrad von etwa 55 % besitzt. So wird stets mehr als die Hälfte der Leistung übertragen. Die Zunahme des Wirkungsgrad mit steigender Anregefrequenz resultiert vermutlich aus der gleichmäßigeren Belastung des TMR, der weniger nachregeln muss. Der Wirkungsgrad für 100 mHz mit 55,5068 % ähnelt zudem der 54 % übertragenden Ladung aus der Messung zu Spannungsdrift.

Für den Modus BP fällt der niedrige Wirkungsgrad, aufgrund der gleichzeitig aktiven TMR, mit etwa 44 % geringer aus. Ebenfalls kann eine Steigerung des Wirkungsgrads für hohe Frequenzen beobachtet werden.

Insgesamt sind die erreichten Wirkungsgrade zufriedenstellend, da nur etwa die Hälfte der für die EIS-Messung benötigten Leistung verloren geht. Dabei ist der Modus RL jedoch um einiges effizienter. Allerdings könnte der Wirkungsgrad für den Modus BP noch gesteigert werden, indem eine Möglichkeit gefunden wird, nicht beide Gleichspannungswandler gleichzeitig versorgen zu müssen.

5.7 Ermittlung der optimalen Periodenzahl für die Messung

Um die optimale Anzahl an Messperioden für die EIS-Messung zu ermitteln, wird ein Versuch durchgeführt. Dazu werden bei einer Frequenz von 1 kHz, um Dauer und Änderung der Ladezustände zur reduzieren, Anregungen mit verschiedenen Periodenzahlen durchgeführt. Es wurden Perioden von 1, 5, 10, 20, 50, 100 und 200 gewählt, um einen größtmöglichen Bereich abzudecken. Dabei wurden die Modi RL und BP getestet. Die Daten wurden mit dem Oszilloskop aufgenommen und mit Matlab ausgewertet und in Nyquist-Diagrammen visualisiert. Die Spannung der linken Batterie betrug zum Zeitpunkt der Messung im Modus BP 10,4 V, die der rechten 9,68 Volt. Im Modus RL war die Spannung der rechten Batterie identisch, die der linken sank auf 10,39 V.

Die aus den Messungen resultierenden Nyquist-Diagramme sind in den Abbildungen 5.25 und 5.26 dargestellt.

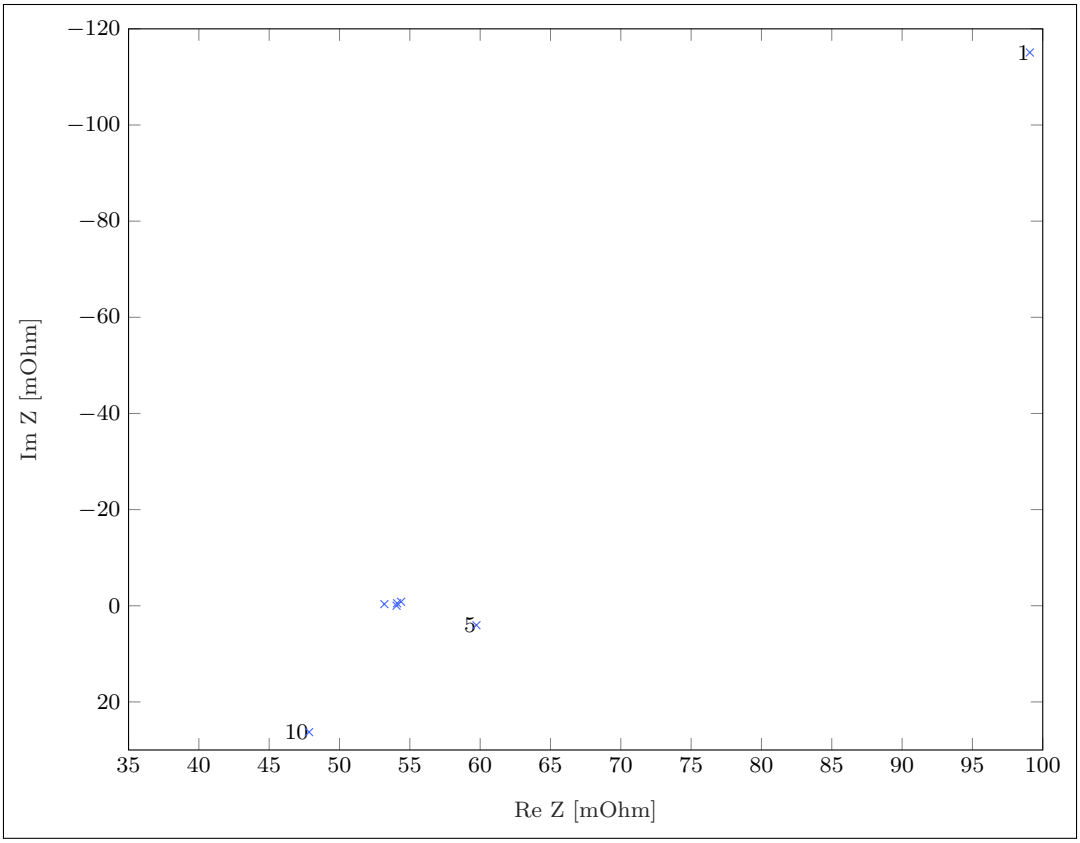


Abbildung 5.25: Nyquist-Diagramm für 1 kHz bei 1, 5, 10, 20, 50, 100 und 200 Messperioden im Modus RL an der linken Batterie
Nicht beschriftet sind 20, 50, 100 und 200 Perioden, die übereinander liegen.

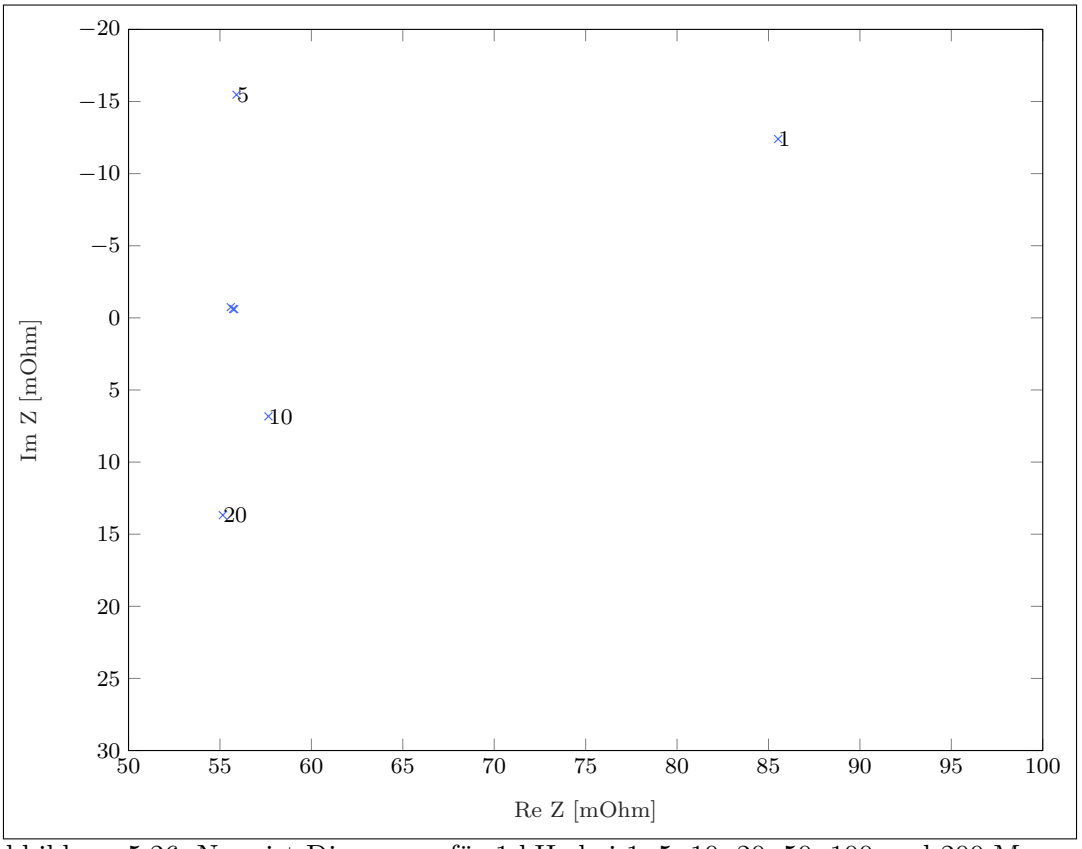


Abbildung 5.26: Nyquist-Diagramm für 1 kHz bei 1, 5, 10, 20, 50, 100 und 200 Messperioden im Modus BP an der linken Batterie
Nicht beschriftet sind 50, 100 und 200 Perioden, die übereinander liegen.

Für beide Betriebsmodi lässt sich festhalten, dass die Messpunkte mit steigender Periodenzahl konvergieren. Dabei gibt es jedoch im unteren Wertebereich teilweise deutliche Abweichungen, wie etwa für 1, 5 und 10 Perioden im Modus RL in Abbildung 5.25 abgelesen werden kann. Für den Modus BP gilt das gleiche, wobei hier zusätzlich noch der Wert mit 20 Perioden deutlich von den höheren Periodenzahlen abweicht.

Um die höheren Periodenzahlen besser unterscheiden zu können, werden sie in Abbildung 5.27 für beide Modi im kleineren Maßstab dargestellt.

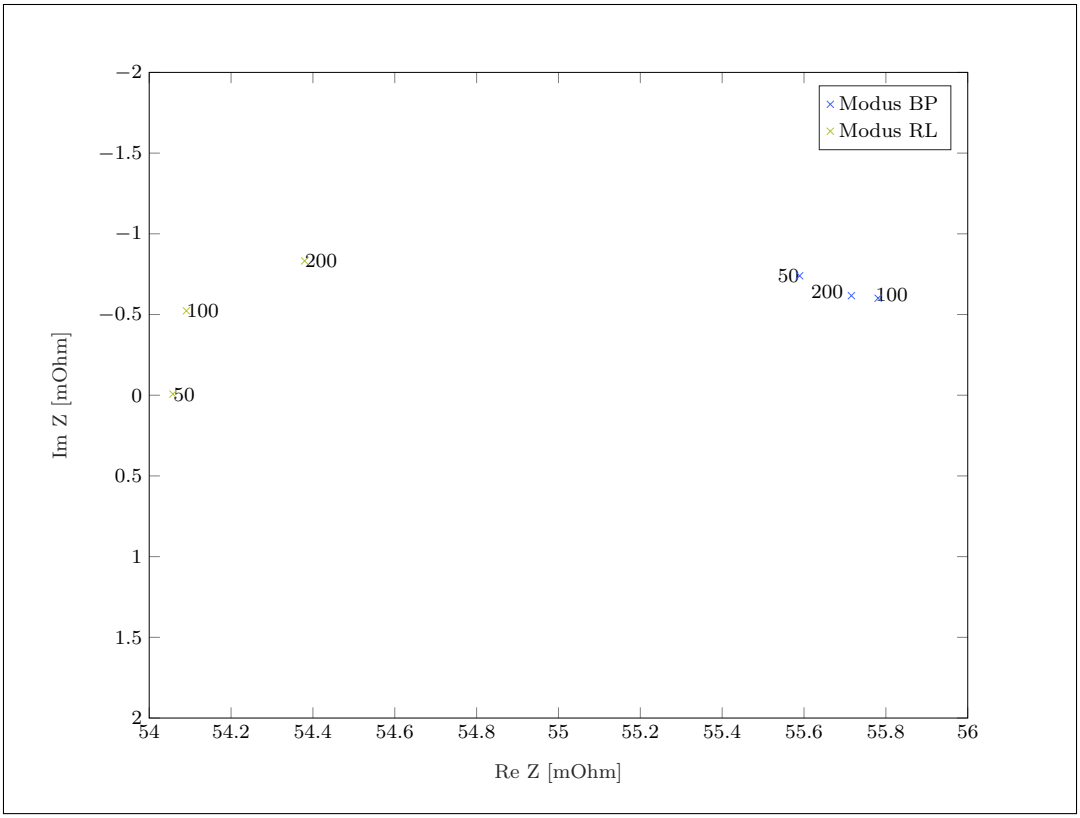


Abbildung 5.27: Nyquist-Diagramm für 1 kHz bei 50, 100 und 200 Messperioden für beide Modi an der linken Batterie

Dabei kann festgestellt werden, dass die Schwankungen der Werte für beide Modi ab 50 Perioden nur noch sehr gering ausfallen. Auch die Unterschiede der Modi untereinander sind gering. Um die Messzeiten kurz und die Ladezustände der Batterien möglichst konstant zu halten, wird daher eine Periodenzahl von 50 für die weiteren Messungen verwendet. Eine weitere Steigerung ist aufgrund der nicht signifikanten Änderung der Ergebnisse nicht notwendig.

5.8 Messung mit 50 Perioden

Um zu überprüfen, ob die Messeinstellungen, Mikrocontroller-Software und PC-Skripte in der Lage sind, eine vollständige EIS-Messung durchzuführen, wurde eine Messung mit 50 Perioden durchgeführt. Dabei betrug die Batteriespannung im Modus BP vor der Messung 10,12 V links und 10,02 V rechts. Während der Messung sanken die Spannungen auf 9,98 V links und 9,9 V rechts. Für eine bessere Vergleichbarkeit der Modi wurde die linke Batterie im Modus RL auf einen ähnlichen Wert von 10,14 V aufgeladen. Die Spannung an der rechten Batterie betrug dann 10,31 V. Nach der Messung war die Spannung an der linken Batterie auf 10,2 V gestiegen, wärend die der rechten auf 9,9 V gesunken war. Die Signale wurden erneut mit dem Oszilloskop aufgezeichnet und mit Matlab ausgewertet. Als Messaufbau wird der in den Abbildungen 5.1 und 5.2 für Versuch 5.2 dargestellte verwendet.

5.8.1 Nyquist-Diagramm

Die Nyquist-Diagramme, die aus den Messdaten erstellt werden, sind in den Abbildungen 5.28 und 5.29 dargestellt.

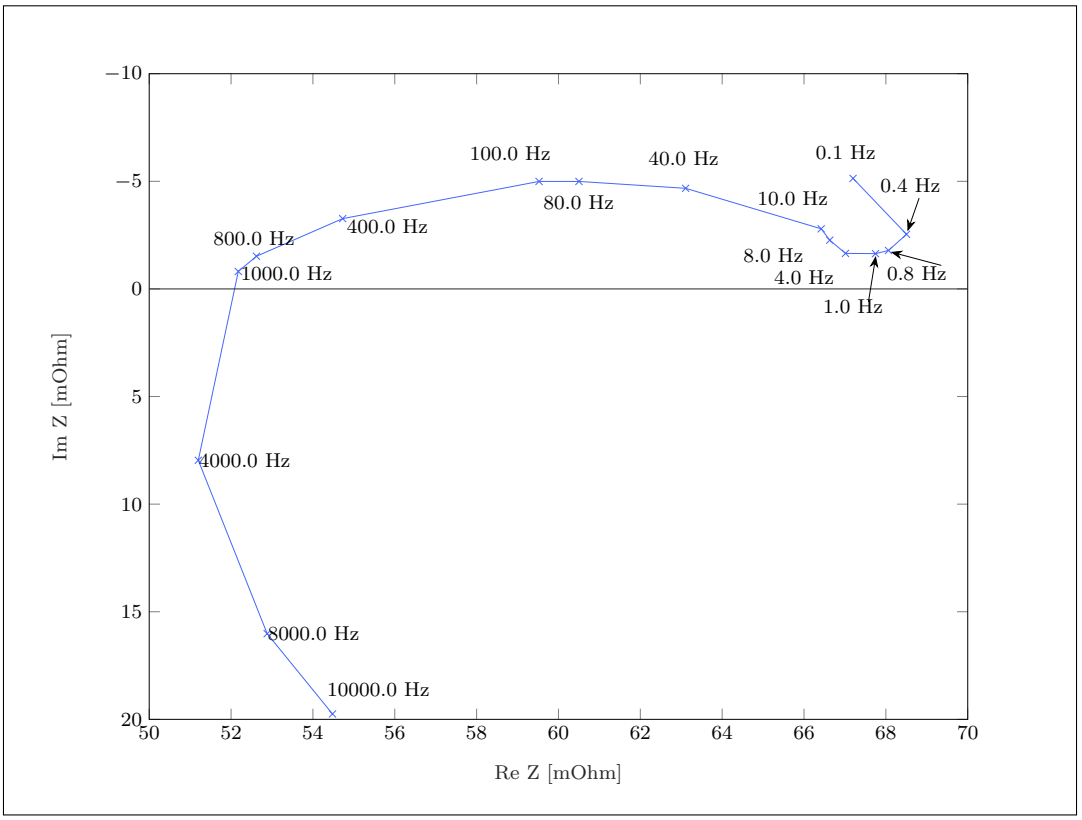


Abbildung 5.28: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus RL an der linken Batterie

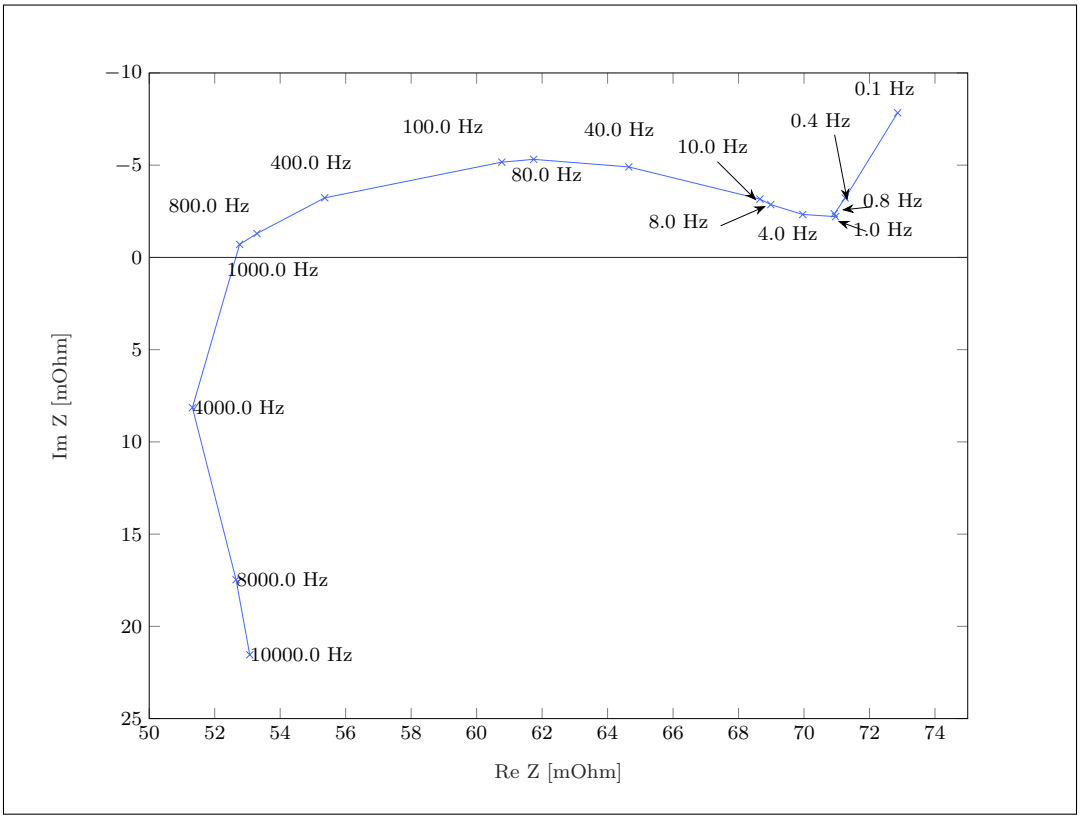


Abbildung 5.29: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus BP an der linken Batterie

Bei der Analyse der Nyquist-Diagramme für den Modus RL in Abbildung 5.28 und für den Modus BP in Abbildung 5.29, kann deutlich die typische Struktur wie in Abbildung 2.1 erkannt werden. So weisen beide Diagramme für niedrige Frequenzen den Diffusionsast auf, wobei er im Modus RL durch einen vermutlichen Fehler für 100 mHz eien Abweichung aufweist. Auch der anschließende Halbkreis von 1 bis 1000 Hz ist deutlich erkennbar. Zum Schluss ist auch der induktive Bereich nach 1000 Hz erkennbar, wo der Imaginärteil für beide Modi stark ansteigt, währen der Realteil recht stabil bleibt.

Um einen Vergleich der Ergebnisse für die Modi untereinander vorzunehmen sind sie in Abbildung 5.30 beide aufgeführt.

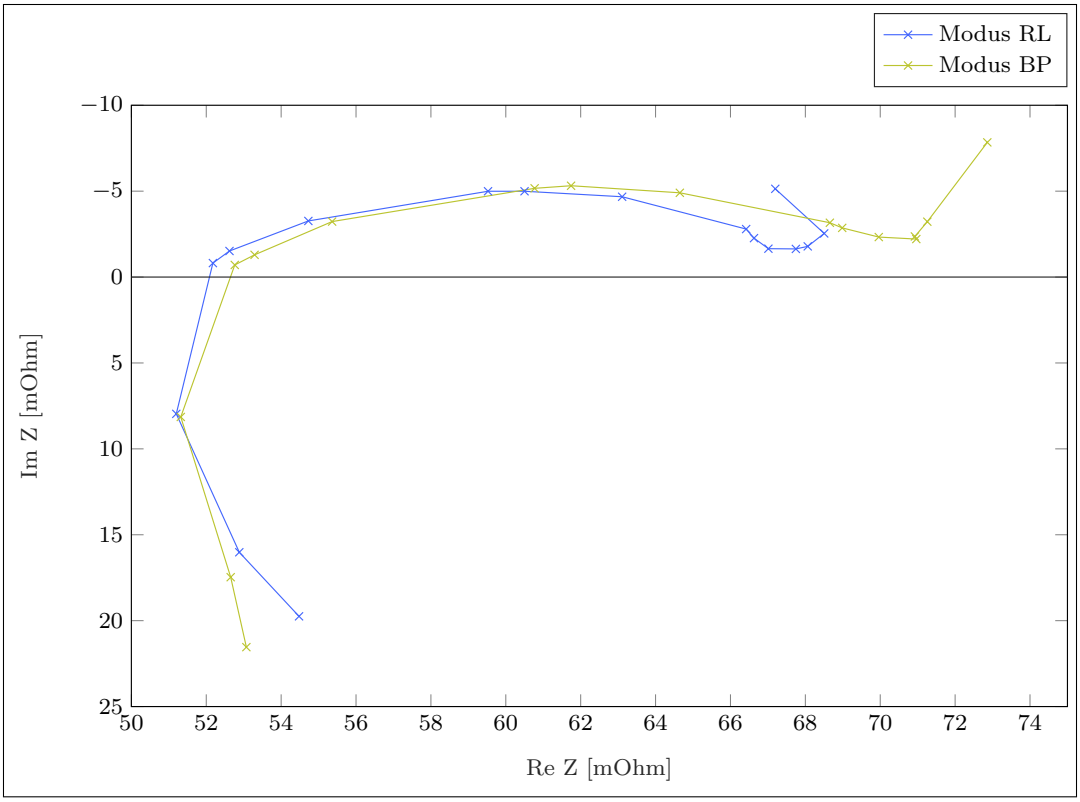


Abbildung 5.30: Nyquist-Diagramm für eine Messung mit 50 Perioden für die Modi RL und BP im Vergleich an der linken Batterie

Hier kann deutlich erkannt werden, dass die Nyquist-Kurven der Modi einen ähnlichen Verlauf aufweisen. Dabei sind die Kurven im höheren Frequenzbereich leicht gegeneinander verschoben, wodurch sich auch der Schnittpunkt mit der X-Achse und damit der Serienwiderstand unterscheidet. Diese Tatsache ist vermutlich auf die unterschiedlichen Ladezustände der linken Batterie vor den Messungen zurückzuführen. Allerdings ist auch zu erkennen, dass der Halbkreis für den Modus RL deutlich gestaucht ist. Der geringere Imaginärteil bei 100 mHz zeigt, dass der Diffusionsast ebenfalls kürzer ist, wenn der vermutlich fehlerhafte Realteil ignoriert wird.

Diese Tatsache resultiert anscheinend aus der höheren Ladungsänderung und damit Belastung der Batterie im Modus RL. Dieser Schluss lässt sich ziehen, da diese Änderungen ein typisches Verhalten bei Erhöhung des Belastungszustands der Batteries ist [5]. In Abbildung 5.31 ist dieser Zusammenhang grafisch dargestellt.

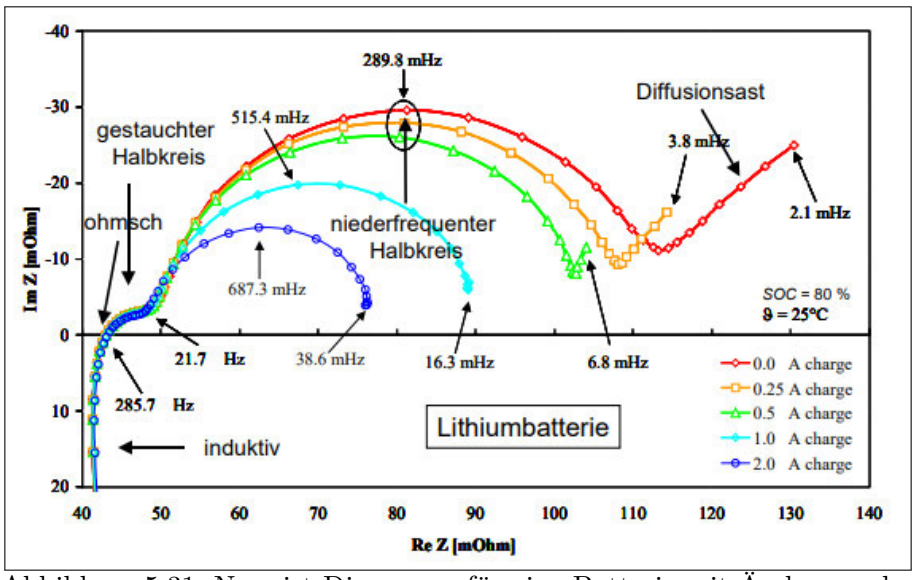


Abbildung 5.31: Nyquist-Diagramm für eine Batterie mit Änderung des Belastungszustands aus [5, S. 5]

5.8.2 Bode-Diagramm

Zum weiteren Vergleich der Betriebsmodi RL und BP werden beide in einem Bode-Diagramm (Abbildung 5.32) dargestellt.

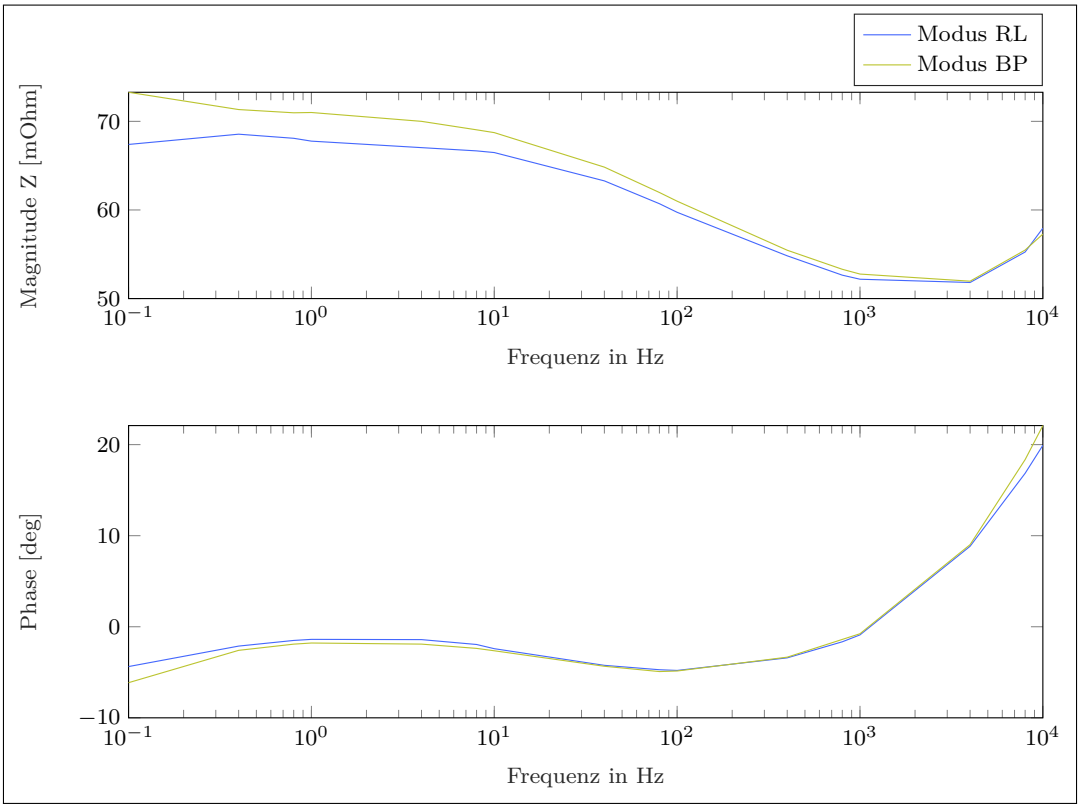


Abbildung 5.32: Bode-Diagramm für eine Messung mit 50 Perioden in den Modi RL und BP an der linken Batterie

Wie auch im Nyquist-Diagramm kann aus dem Bode Diagramm geschlossen werden, dass die zwei Betriebsmodi nur geringfügige Unterschiede aufweisen. Die Unterschiede in Phase und Amplitude, sind vor allem im niedrigeren Frequenzbereich zu beobachten, wo im Modus RL für 100 mHz vermutlich ein Messfehler vorliegt.

5.8.3 Harmonischen nutzbar für Nyquist-Diagramm

In Abbildung 5.33 wird gezeigt, dass bei beiden Modi Oberschwingungen auftreten. Dabei folgt der Modus RL dem Verhalten eines Rechtecksignals mit Oberschwingungen an den für sie typischen Vielfachen der Grundschwingung. Im Modus BP kommen zusätzlich noch Oberschwingungen bei Vielfachen der doppelten Grundfrequenz hinzu.

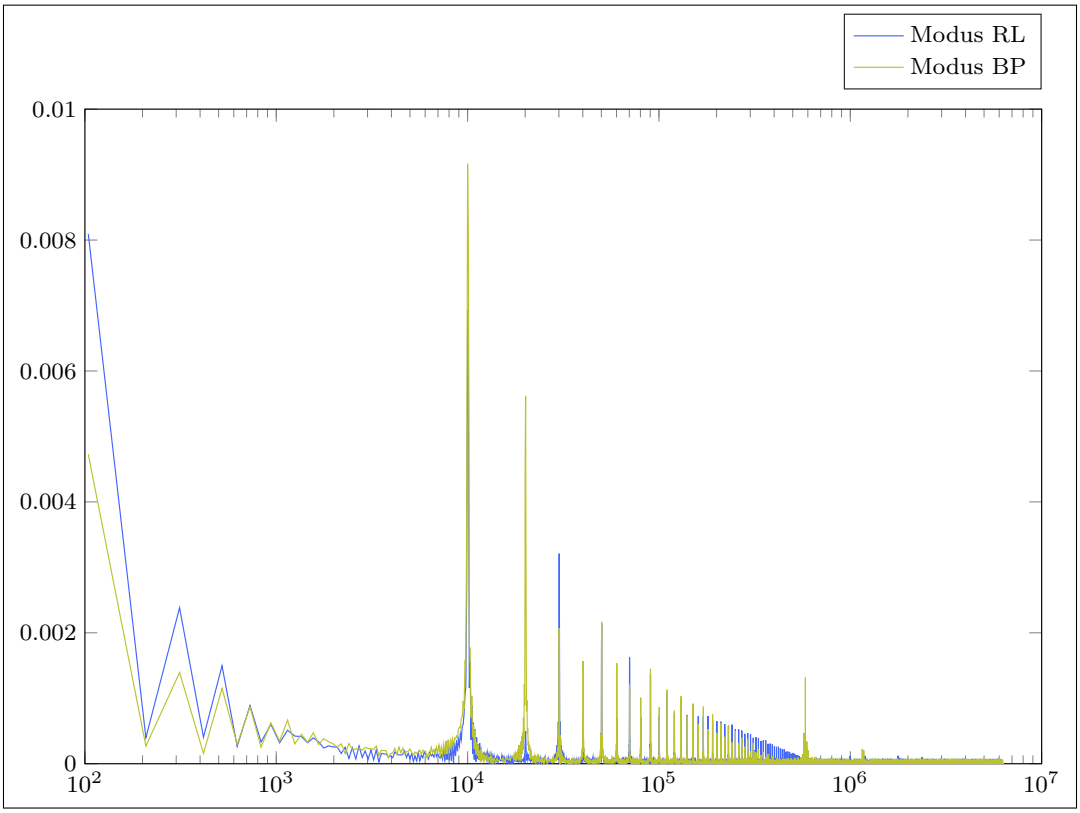


Abbildung 5.33: Frequenzspektrum für beide Modi mit einer Anregefrequenz von 10 kHz

Aufgrund dieses Verhalten kann die Frage gestellt werden, ob sich diese Oberschwingungen als zusätzliche Messpunkte einer EIS-Messung eignen. Deswegen wird mittels Matlab die Spektrallinie der Oberschwingung mit der höchsten Amplitude zusätzlich ins Nyquist-Diagramm übernommen.

Wie in den Nyquist-Diagramme in den Abbildungen 5.34 und 5.35 zu sehen ist, eignen sich die Oberschwingungen durchaus als zusätzliche Messpunkte. Dabei wurden die gleichen Messdaten verwendet wie in den Abbildungen 5.28 und 5.29. Die Abweichungen sind bis auf wenige Ausnahmen, wie bei höheren Frequenzen für den Modus BP oder 0,3 Hz in beiden Modi, eher gering und stimmen gut mit den Anregefrequenzen überein.

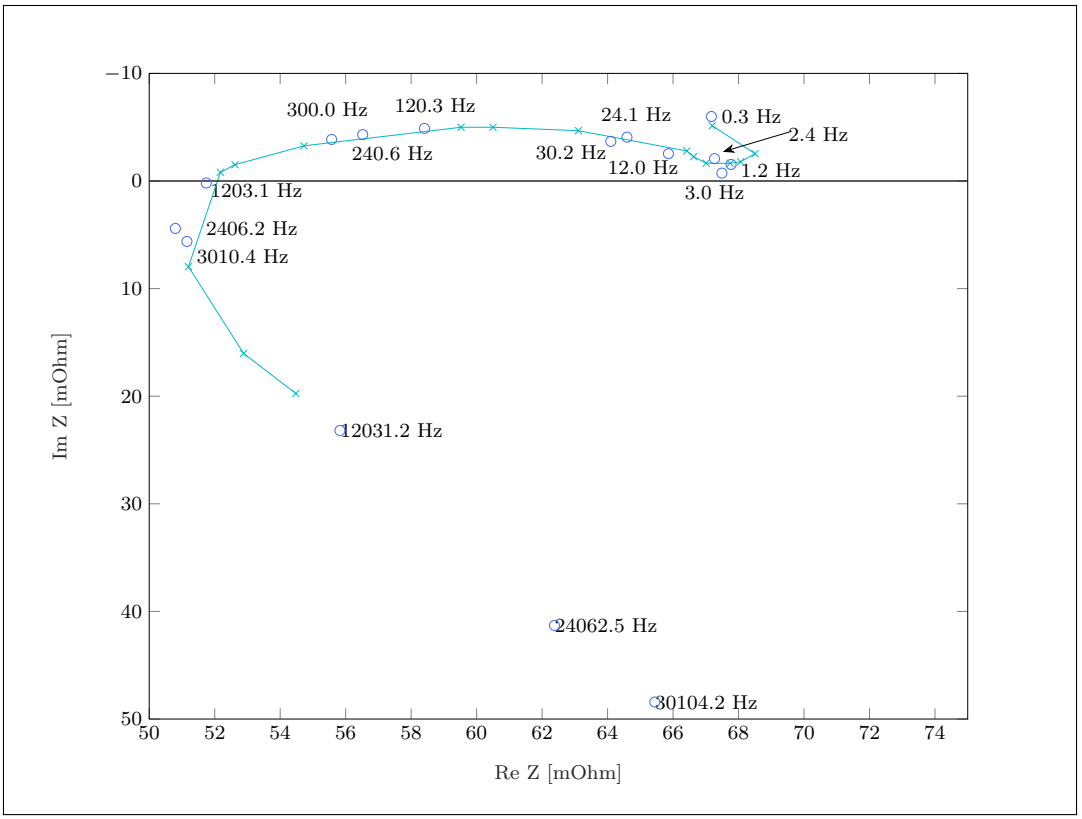


Abbildung 5.34: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus RL mit Nutzung der ersten Oberschwingung
x markiert die gemessenen Frequenzen
o markiert die Oberschwingungen

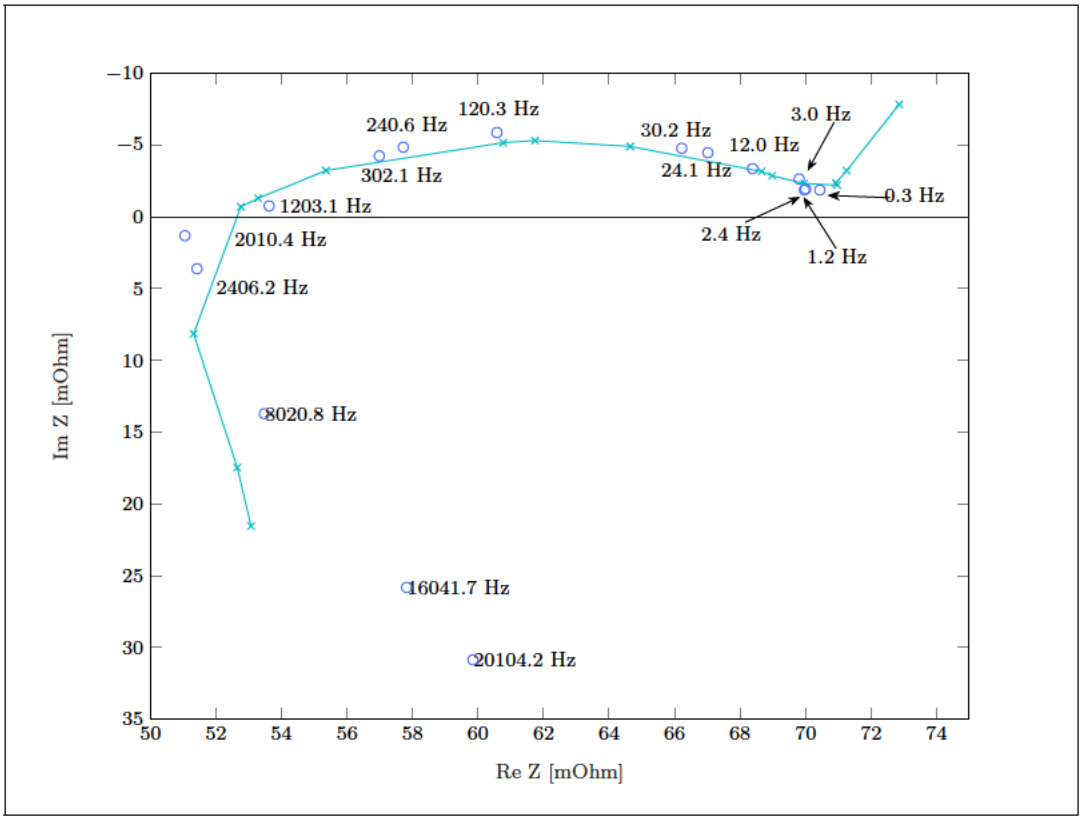


Abbildung 5.35: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus BP mit Nutzung der ersten Oberschwingung
x markiert die gemessenen Frequenzen
o markiert die Oberschwingungen

5.8.4 Harmonischen Bode-Diagramm

Die Bode-Diagramme der Oberschwingungen bestätigen für beide Betriebsmodi den Trend, der bereits in den Nyquist-Diagrammen ersichtlich geworden ist. Wie in Abbildung 5.36 und 5.37 abgelesen werden kann, stimmen sie gut überein und setzen sie für höhere Frequenzen logisch fort. Einzig die Abweichung für höhere Frequenzen im Modus BP und die für 0,3 Hz fallen hier negativ auf.

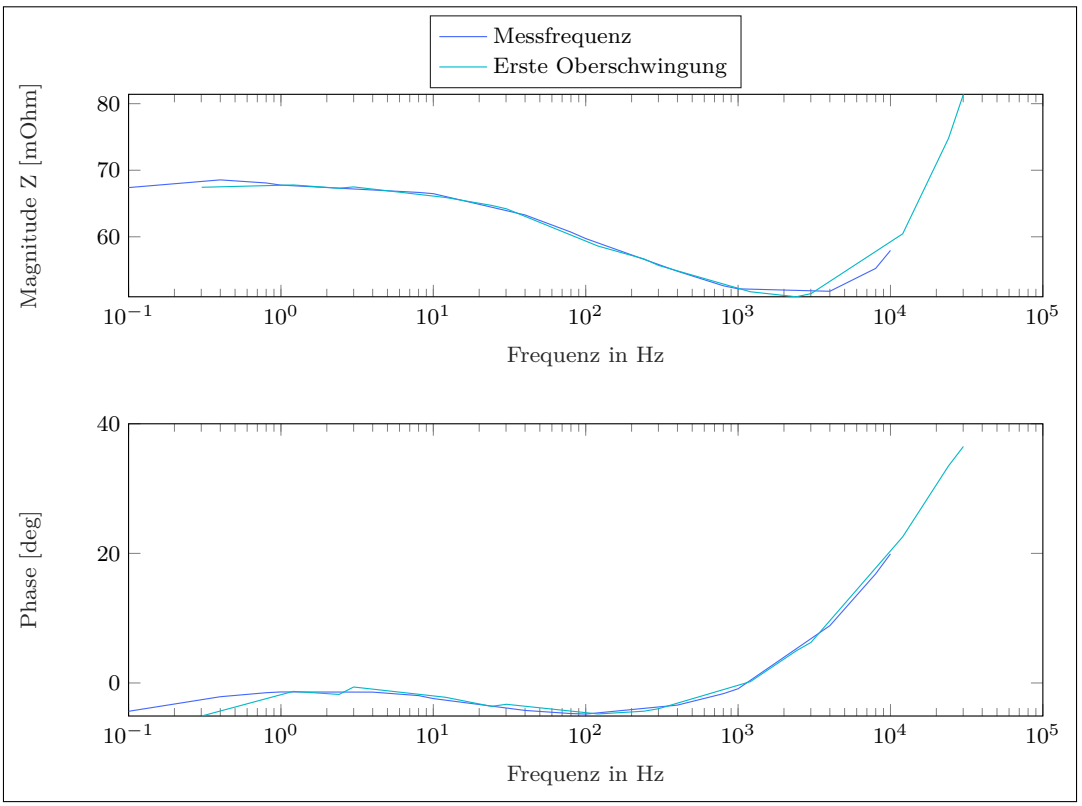


Abbildung 5.36: Bode-Diagramm, Vergleich für eine Messung mit 50 Perioden im Modus RL mit Nutzung der ersten Oberschwingung

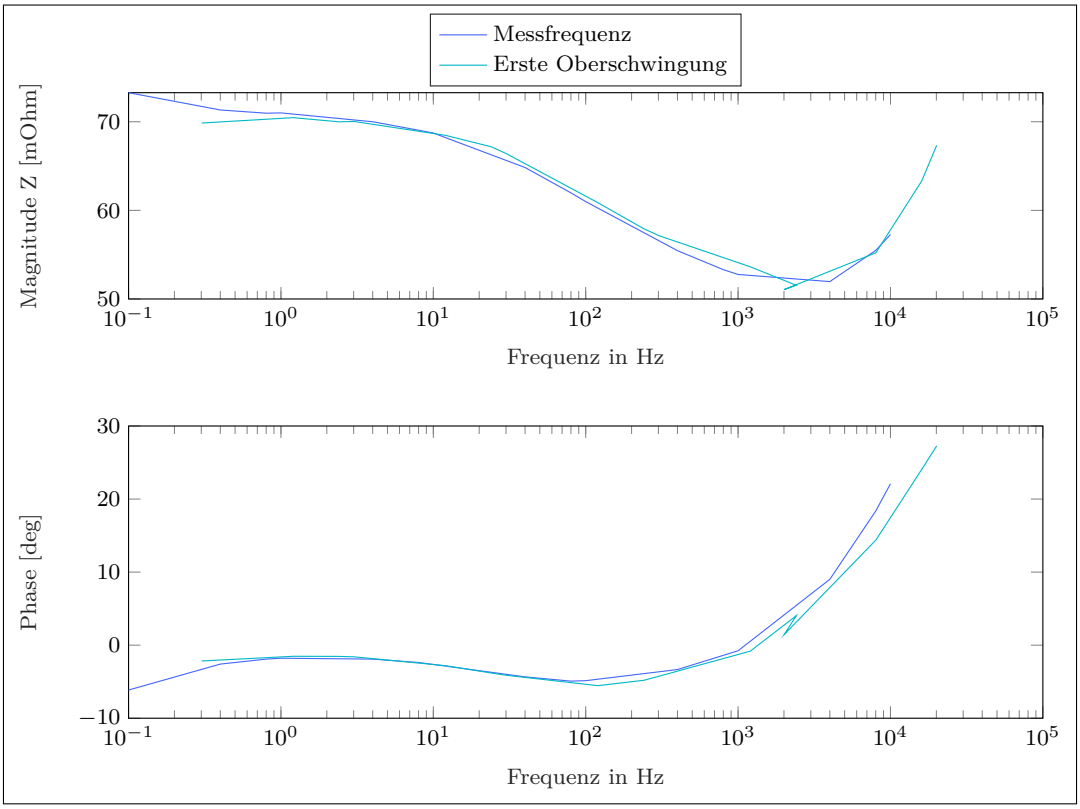


Abbildung 5.37: Bode-Diagramm, Vergleich für eine Messung mit 50 Perioden im Modus BP mit Nutzung der ersten Oberschwingung

Insgesamt lässt sich festhalten, dass sich Oberschwingungen der Anregefrequenzen durchaus als zusätzliche Messpunkte der EIS-Messung eigenen. Diese Tatsache ist insbesondere für die niedrigen Anregefrequenzen interessant, da sie einen großen Teil der Messdauer einnehmen, der so reduziert werden kann. Daher kann dieser Ansatz in weitergehenden Versuchen überprüft und verbessert werden. Dabei muss aber stets berücksichtigt werden, dass ein Messfehler für eine Anregefrequenz auch Auswirkungen auf die Oberschwingung haben kann.

5.8.5 Entladekurven der Modi RL und LR nutzbar für EIS-Messungen

Da in den MP Modi die Batterie auf der nicht aufladenen Seite gleichzigt entladen wird, stellt sich die Frage, ob auf dieser Seite ebenfalls eine EIS-Messung durchgeführt werden kann. Dafür werden die Signale auf der rechten Seite für den Modus RL mit Matlab ausgewertet und in ein Nyquist-Diagramm(Abbildung 5.38) und Bode-Diagramm(Abbildung 5.39) eingetragen.

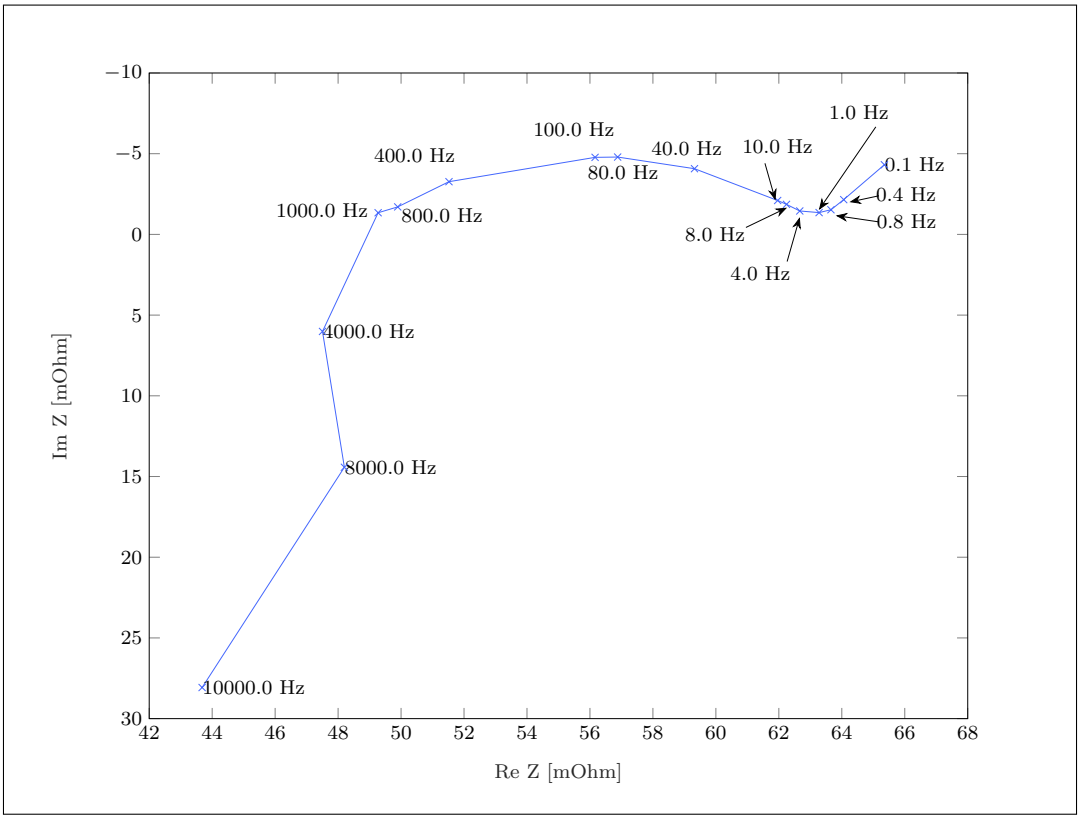


Abbildung 5.38: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus RL auf der Entladeseite

Wie in Abbildung 5.38 abgelesen werden kann, weist auch die EIS-Messung, die mit dem Entladevorgang durchgeführt wird, die typische Struktur auf. Allerdings ist sie nach links verschoben, was vermutlich aus der höheren Batteriespannung zu Messbeginn resultiert. Auch der Messpunkt bei 10 kHz weist im Gegensatz zu den bisherigen Ergebnissen eine Abweichung nach links auf. Dieses Verhalten resultiert vermutlich aus Störungen des Anregesignals.

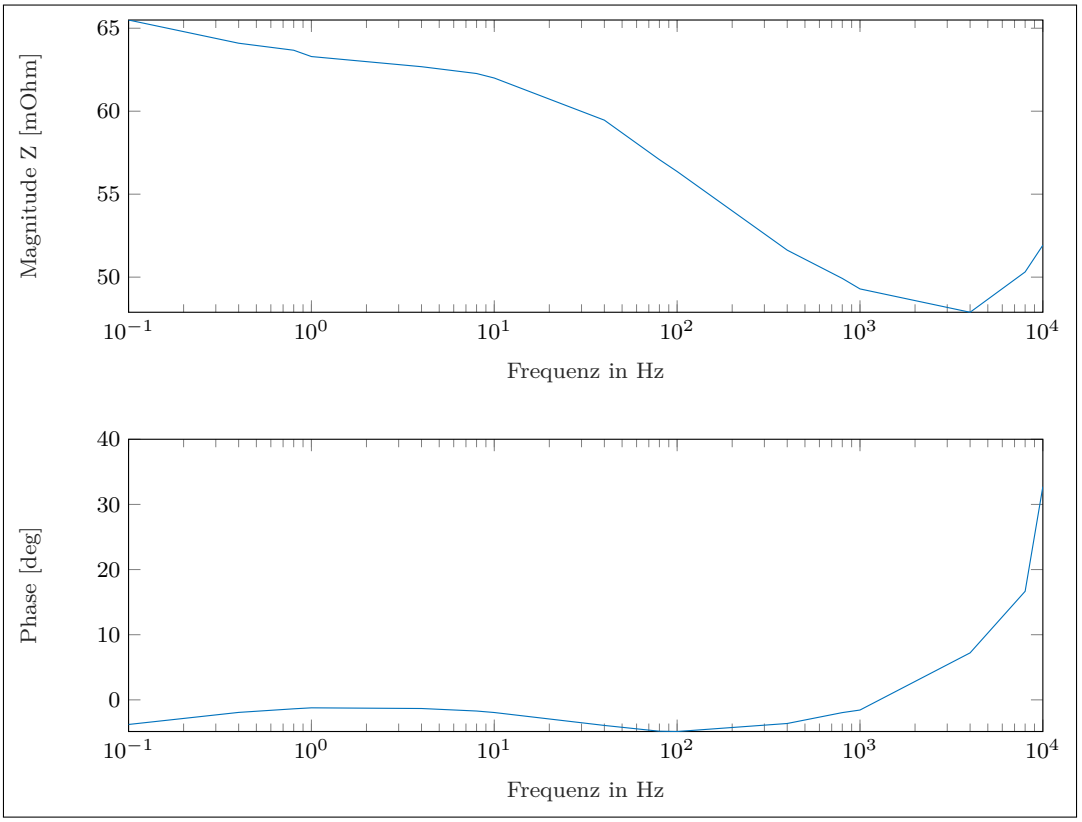


Abbildung 5.39: Bode-Diagramm für eine Messung mit 50 Perioden im Modus RL auf der Entladeseite

Für das Bode-Diagramm gelten die gleichen Beobachtungen wie für das Nyquist-Diagramm. So ist der grundsätzliche Verlauf ähnlich wie bei den vorherigen Messungen. Die Abweichungen in den Amplitude lassen sich auf die Batteriespannungen zurückzuführen. Ebenfalls fällt erneut der Messwert bei 10 kHz auf.

Aus dem Versuch kann gefolgert werden, dass sich eine EIS-Messung während dem Entladen einer Batterie durchführen lässt. Somit eigenen sich die Modi LR und RL wie der Modus BP dazu, beide Batterie gleichzeitig zu vermessen.

5.9 Vergleichsmessung mit konventionellem EIS-Meter

Um eine Bewertung vornehmen zu können, wie gut die Anregeschaltung im Vergleich zu einem stationären EIS-Meter ist, wird eine Messung mit einem konventionellem Gerät durchgeführt. Verwendet wird ein FuelCon TrueData-EIS [19], dass allerdings die Batterie auch immer mit einem DC-Anteil belastet. Um diesen Anteil zu kompensieren wird ein BOP 50-20MGL Netzgerät der Firma KEPCO verwendet. Verwendet wird es, da es sich bei ihm um ein Vier-Quadranten-Netzgerät handelt, dass Leistung sowohl abgeben als auch aufnehmen kann [20]. In Abbildung 5.40 sind beide Geräte in ihrem Gestell abgebildet.

Abbildung 5.40: Gestell mit dem FuelCon TrueData-EIS und dem KEPCO BOP 50-20MGL Netzgerät

Die Messung wird dabei über die Platine der Anregeschaltung durchgeführt, um hier auftretende Impedanzen zu berücksichtigen. In Abbildung 5.41 ist der Anschluss des TrueData-EIS an die Platine dargestellt. Die Spannung betrug dabei vor den Messungen 11,34 V an der linken Batterie und 11,31 V an der rechten, welche auch nach der Messung vorlagen. Die Anregung wurde für 20 Perioden und Anregefrequenzen im Bereich von 100 mHz bis 10 kHz durchgeführt.

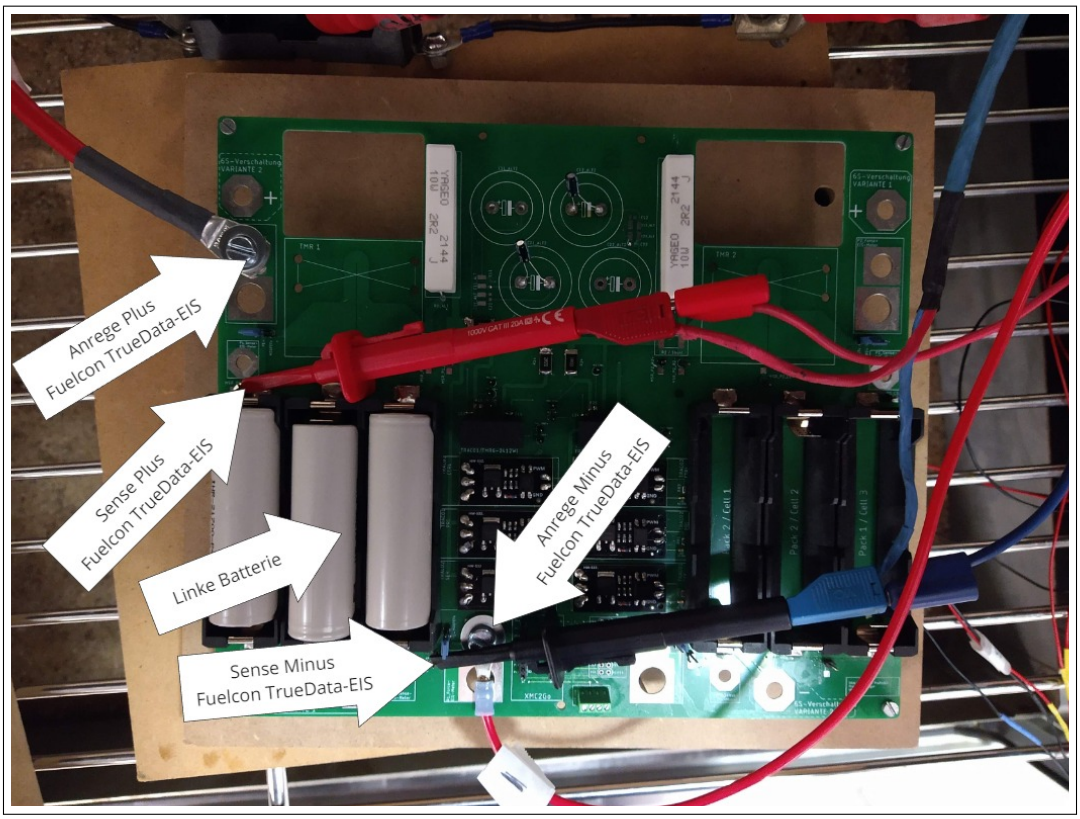


Abbildung 5.41: Anschluss der Platine der Anregeschaltung an das FuelCon TrueData-EIS

Im Anschluss wurde dann eine Messung mit der Anregeschaltung durchgeführt, wobei die Batteriespannungen sich nicht unterschieden, um einen Vergleich zu ermöglichen. Zunächst wurde eine Messung im Modus RL durchgeführt, wobei die Spannung der linken Batterie auf 11,35 V stieg und die der rechten auf 11,29 V abgesunken ist. Nach der anschließenden Messung im Modus BP lagen die Spannungen bei 11,34 V links und 11,28 V rechts. Die Spannungen lagen also für alle Messungen in einem ähnlichen Bereich. Die enstehenden Nyquist- und Bode Diagramme werden anschließend mit Matlab erstellt. Als Messaufbau wird der in den Abbildungen 5.1 und 5.2 für Versuch 5.2 dargestellte verwendet.

5.9.1 Nyquist-Diagramm

In den Abbildungen 5.42 und 5.43 sind die Nyquist-Diagramme für die Anregeschaltung aufgeführt.

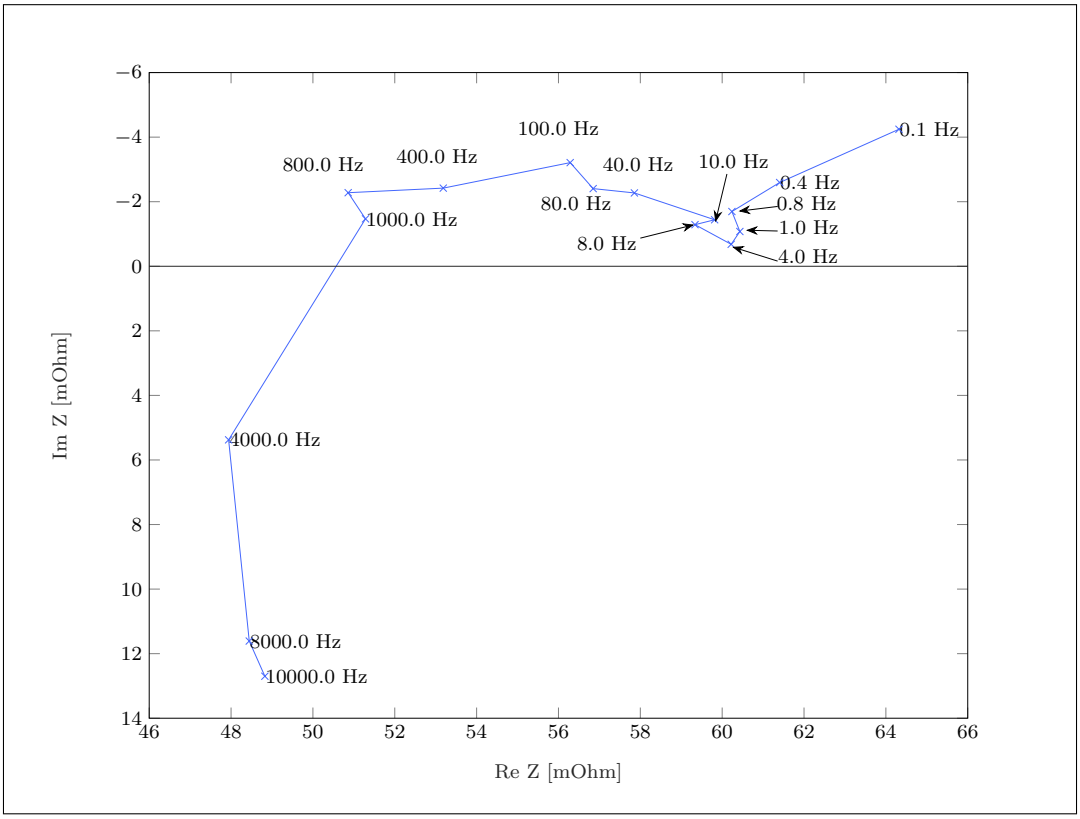


Abbildung 5.42: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus RL

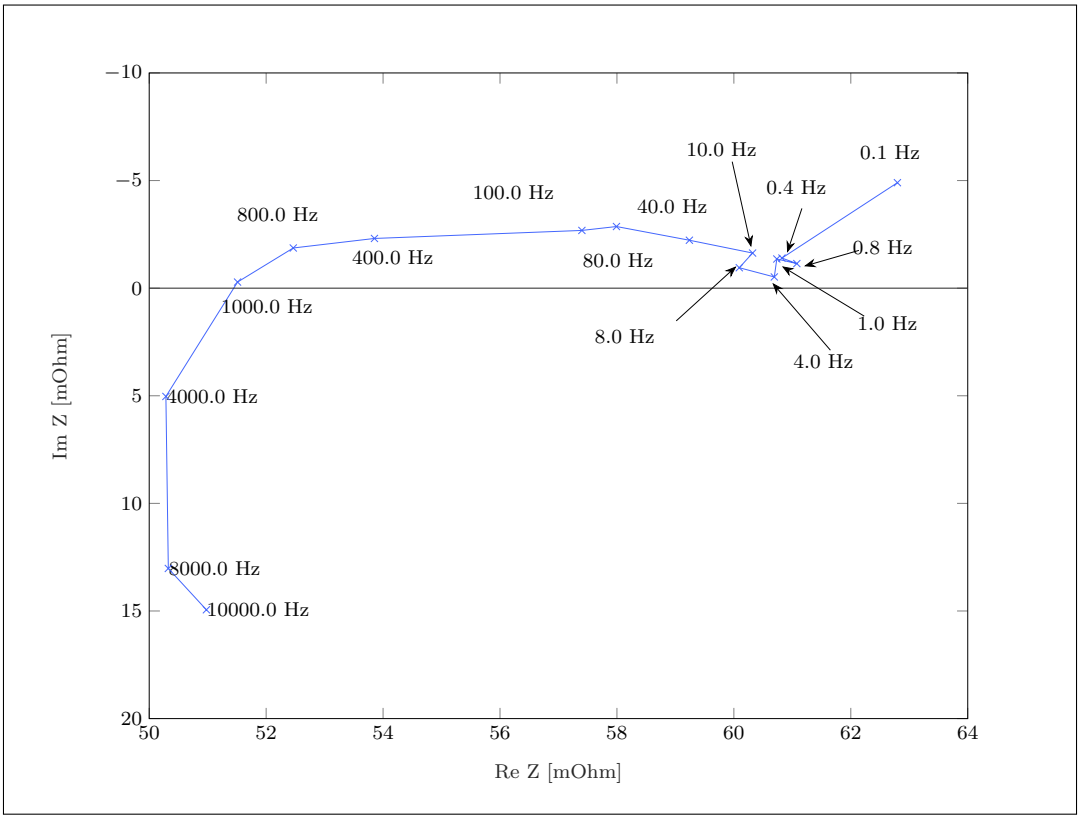


Abbildung 5.43: Nyquist-Diagramm für eine Messung mit 50 Perioden im Modus BP

Wenn man die Nyquist-Diagramme der Anregeschaltung betrachtet, kann festgestellt werden, dass der Modus RL eine sehr verformte Kurve besitzt. So ist der Halbkreis nur schwer zu erkennen und auch der Verlauf bei 0,8 bis 8 Hz entspricht nicht den Erwartungen. Der Diffusionsast hingegen ist klar zu erkennen genau wie der induktive Bereich über 1 kHz. Auch der Schnittpunkt mit der X-Achse liegt mit etwa 50 mΩ im Bereich bisheriger Messungen. Für den Modus BP sind die Messergebnisse besser, so ist hier der Halbkreis klar zu erkennen. Auch der induktive Bereich und Diffusionsast lassen sich feststellen. Allerdings tritt auch hier eine starke Abweichung im Bereich von 0,4 bis 8 Hz auf. Diese Abweichungen resultieren vermutlich aus den hohen Batteriespannungen. Diese liegen mit 11,32 V deutlich über der angenommenen maximalen Spannung mit 10,8 V, was die Ausgangsströme auf etwa 150 mA senkt. Diese reduzierte Anregung verursacht dann vor allem bei niedrigen Frequenzen Probleme.

Das Nyquist-Diagramm des TrueData-EIS Messgeräts ist in Abbildung 5.44 dargestellt.

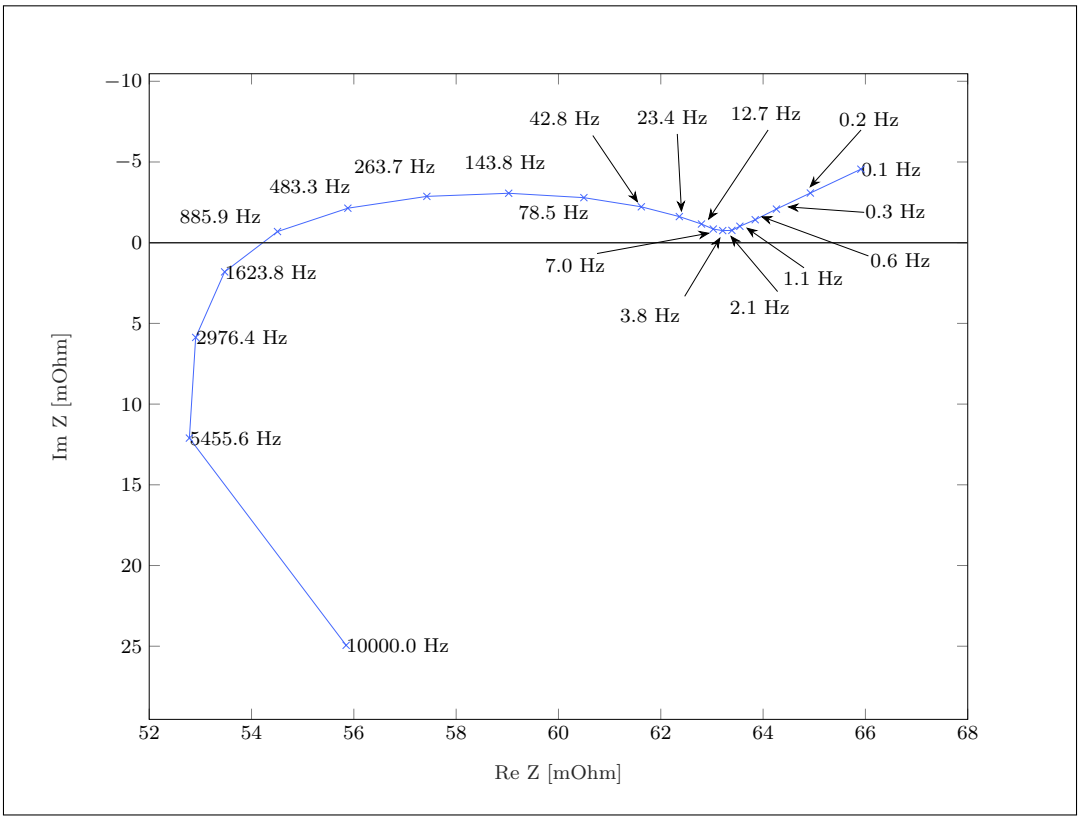


Abbildung 5.44: Nyquist-Diagramm für eine Messung mit 20 Perioden mit dem Fuelcon TrueData-EIS

Wenn mann das Nyquist-Diagramm des TrueData-EIS mit denen der Anregeschaltung vergleicht, fällt zunächst der sehr exakte Kurvenverlauf auf, der sehr gut einem typischen Nyquist Diagramm entspricht. Alle Bereiche wie Diffusionsast, Halbkreis und induktiver Bereich sind klar zu erkennen.

Um nun einen Vergleich zwischen der Anregeschaltung und dem TrueData-EIS vorzunehmen, werden alle drei Kurven in einem Diagramm (Abbildung 5.45) dargestellt.

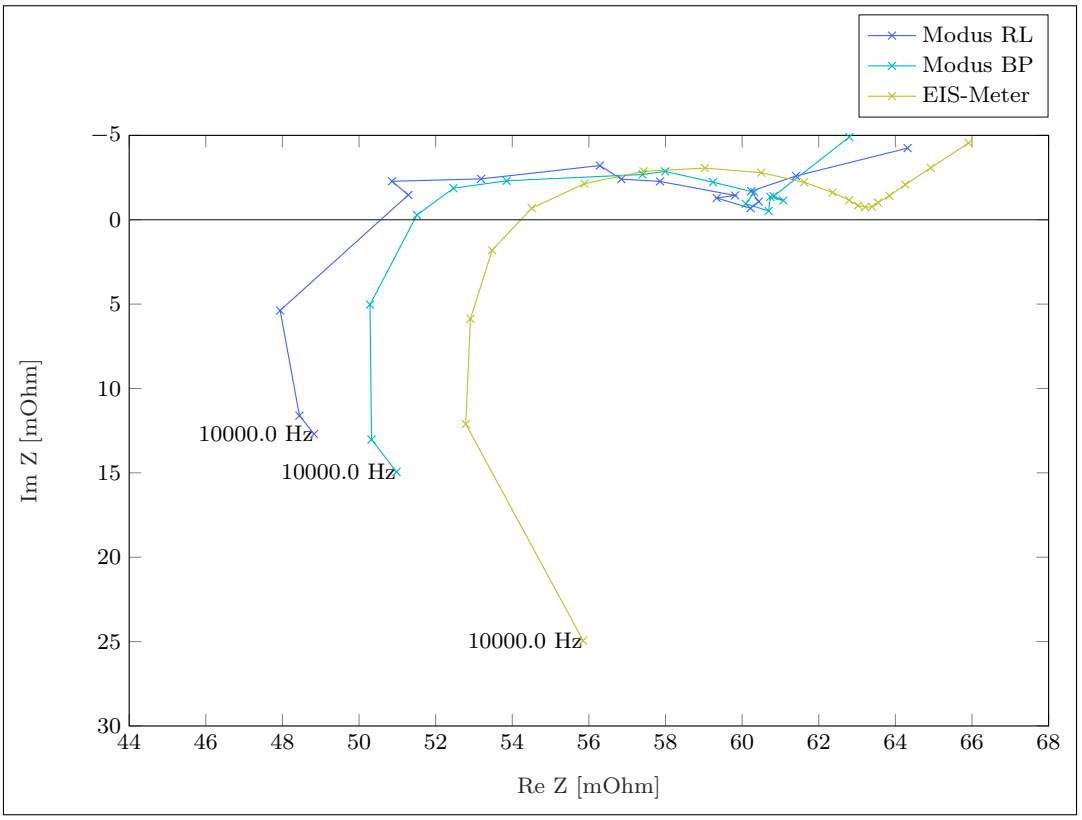


Abbildung 5.45: Nyquist-Diagramm für eine Vergleich der Messungen vom Fuelcon TrueData-EIS und der Anregeschaltung

Aus dem Nyquist-Diagramm geht deutlich hervor, dass die Unterschiede zwischen der Anregung mit dem TrueData-EIS und der Anregeschaltung nicht sonderlich groß sind. So bewegen sich alle Kurven im Bereich zwischen 46 und 66 mΩ auf der X-Achse sowie -5 bis 25 mΩ auf der Y-Achse. Die Verläufe der Kurven sind sich dabei sehr ähnlich jedoch leicht auf der X-Achse verschoben. Auch der Messpunkt bei 10 kHz ist bei der Kurve des TrueData-EIS um einiges stärker auf der Y-Achse ausgeprägt.

5.9.2 Bode-Diagramm

Wenn man einen abschließenden Vergleich der Messdaten im Bode-Diagramm in Abbildung 5.46 vornimmt, bestätigen sich die bisherigen Ergebnisse. So liegen die Amplitude der Messung mit dem TrueData-EIS konstant oberhalb der Messungen mit der Anregeschaltung. Dabei ist die Abweichung nicht sonderlich groß bis auf den Endpunkt bei 10 kHz wie auch bei der Phase. Hier ist die Abweichung sogar quasi nicht vorhanden und erst bei höheren Frequenzen beginnen die Kurven zu divergieren.

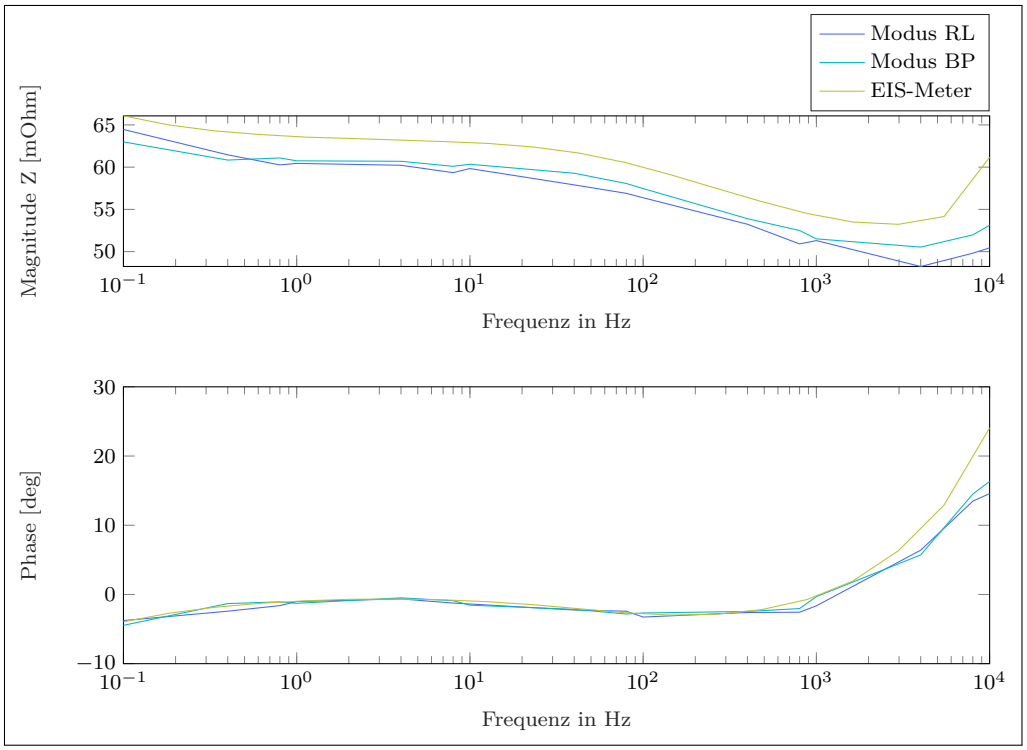


Abbildung 5.46: Bode-Diagramm für einen Vergleich der Messungen vom Fuelcon TrueData-EIS und der Anregeschaltung

Als Ergebnis lässt sich festhalten, dass die Anregeschaltung auch beim Vergleich mit typischer Messhardware gute Ergebnisse erzielt. Die Abweichungen sind nicht zu groß und erlauben somit eine Auswertung mit Standardmethoden. Dabei sollte jedoch auf den Ausgangswiderstand geachtet werden, der den Batteriespannungen angepasst wird. So können die Ströme erhöht werden, was die Messergebnisse bei niedrigen Frequenzen verbessert. Der Modus BP zeigt aufgrund seiner wechselnden Pulse, für niedrige Ströme, ein deutlich besseres Ergebnis als der Modus RL. Diese Tatsache sollte für weitere Versuche berücksichtigt werden.

6 Fazit

Die Entwicklung eines Demonstrators für das Konzept einer effizienten Anregeschaltung für die EIS bei großen Fahrzeugbatterien konnte erfolgreich abgeschlossen werde. So konnte eine Schaltung designt werden, die allen Anforderungen, wie etwa die galvanische Trennung der Teilbatterien, erfüllt. Auch die Entwicklung der Software für den Mikrocontroller konnte den Anforderungen entsprechend umgesetzt werden. So werden alle geforderten Schaltzustände für die Anregeschaltung für die drei Betriebsmodi LR, RL und BP in der korrekten Reihenfolge ausgeführt.

Auch die Einstellung der Frequenz im Bereich von 100 mHz bis 10 kHz, des Tastgrads des Anregesignals und die Festlegung einer bestimmten Periodenzahl über die UART-Schnittstelle konnten realisiert werden. Dabei wurde das Problem der verschobenen Datenbytes allerdings über einen Workaround gelöst.

Um die Qualität der Signale zu verbessern wurden Ausgangskondensatoren eingesetzt, um den Spannungseinbruch beim Einschalten der Anregung zu kompensieren. Dabei wurde versucht, eventuelle Überschwingung zu minimieren. Die Ausgangswiderstände wurden so gewählt, dass mit einem größtmöglichen Strom über einen hohen Spannungsbereich angeregt werden kann.

Die eingestellten Anregefrequenzen wurden dabei von der Anregeschaltung mit nur geringen Fehlern getroffen und die Abweichungen der Tastgrade durch die verwendeten Schaltmodule konnten über die Einstellung niedrigere Werte in der PC-Software kompensiert werden. So konnten für beide Abweichungen, bis auf wenige Ausnahmen, Werte von unter 1 % erreicht werden.

Der für die Anregung beobachtbare Batteriespannungsverlust ist für längere Messzeiten deutlich sichtbar. Dabei fällt er in den MP-Modi für eine Batterie größer aus, wobei die andere dabei leicht aufgeladen wird. Für den Modus BP liegt hingegen bei beiden Batterien ein Spannungsverlust vor.

Wenn die Verluste beim Ladungsaustausch betrachtet werden, kann festgestellt werden, dass der Verlust für den Modus BP um etwa 25 % höher liegt. Diese Verhalten lässt sich auf das gleichzeitige Betreiben der TMR zurückführen. Aufgrund dieser fällt auch der Wirkungsgrad im Modus BP niedriger aus als in den anderen Modi. Dabei ist jedoch ein Wirkungsgrad von mindestens 45 % beziehungsweise 55 %, im Modus RL, immer noch als gut zu bewerten. Denn so geht etwa die Hälfte der aufgewendeten Energie nicht verloren, was eine Verbesserung gegenüber dem einfachen Verbrauch oder externen Zufuhr der nötigen Energie darstellt.

Als die bestmöglichen Parameter für eine EIS-Messung ließen sich die Verwendung einer DC-Kopplung einer am Oszilloskop bei der Aufnahme von 50 Perioden bestimmen. Die Versuchsmessung zeigen dabei Ergebnisse, die denen einer typischen Messung für die EIS einer Lithiumzelle entsprechen. Die Nyquist-Diagramme zeigen den charakteristischen Verlauf und auch die Bode-Diagramme bestätigen das Ergebnis. Weiterhin eignen sich auch die Oberschwingungen und der Entladevorgang der Batterie dazu, Messpunkte für die EIS-Messung zu ermitteln. Ein abschließender Vergleich mit einem EIS-Messgerät fiel ebenfalls positiv aus, obwohl Abweichungen aufgrund der geringen Ströme durch hohe Batteriespannungen auftraten.

Das Projekt konnte somit zu einem positiven Abschluss gebracht werden. Die Software und Hardware arbeiten korrekt und liefern plausible Messergebnisse. Somit ist die Demonstrationsschaltung bereit für weitere Versuche.

6.1 Mögliche Verbesserungen und Ausblick

Als mögliche Verbesserung der Anregeschaltung zeigte sich, dass eine Regelung des Stroms günstiger als die der Spannung wäre. So ist der Anregestrom sehr von der Batteriespannung abhängig was bei der Vergleichsmessung zu Problemen geführt hat. Auch wäre so auch eine einfachere Kontrolle des Ladezustands möglich. Zudem müsste der Strom nicht mehr über Widerstände begrenzt werden, sodass hier keine Leistung mehr verloren geht. Die Möglichkeit, den nicht benötigen Gleichspannungswandler im Modus BP abschalten zu können, wäre ebenfalls eine Verbesserung, da so der Wirkungsgrad der Schaltung gesteigert werden kann. Auch eine Änderung der Optokoppler der Schaltmodule sollte in Betracht gezogen werden, da sie hochfrequente Signale und ihre Tastgrade verfälschen. Erste Tests hierzu von der Forschungsgruppe zeigten, dass Verbesserungen hierdurch möglich sind.

Weitere Versuche könnten in Richtung eines ganzheitlichen Messsystems, wie es in Fahrzeugen eingesetzt werden würde, gehen. So wurden die Messungen bislang über Oszilloskop aufgenommen, während die Steuerung über einen PC erfolgt. Stattdessen kann die Messung mit Mikrocontroller auf den Batteriezellen durchgeführt werden. Auch der Einsatz der von Herrn Dr. Schüthe entwickelten Magnetfeldsensoren zur Strombestimmung könnten anstelle der Messwiderstände verwendet werden. Dabei sollte die Steuerung über einen zentralen Mikrocontroller erfolgen, der auch den Takt vorgibt, was eine Synchronisierung erforderlich macht. Bislang wurden die Tests zudem nur mit wenigen Batteriezellen durchgeführt und so könnte man die Größe der Batterien steigern, um Bedingung zu erhalten, die der Realität näher sind.

Literaturverzeichnis

[1] Pesaran, Ahmad (Hrsg.): *Lithium Ion batteries in electric drive vehicles*. Warrendale, Pennsylvania : SAE International, 2016 (SAE books). – ISBN 9780768082807

[2] Kanoun, Olfa (Hrsg.): *Impedance spectroscopy*. Berlin : Walter de Gruyter, 2018. – ISBN 9783110558920. – Includes bibliographical references

[3] Ansean, David ; Gonzalez, Manuela ; Viera, Juan C. ; Garcia, Victor M. ; Alvarez, Juan C. ; Blanco, Cecilio: Electric Vehicle Li-Ion Battery Evaluation Based on Internal Resistance Analysis. In: *2014 IEEE Vehicle Power and Propulsion Conference (VPPC)*, IEEE, Oktober 2014

[4] Dr. Ing. h.c. F. Porsche AG - Porsche Deutschland: *Porsche Mission E - Porsche Deutschland*. https://www.porsche.com/germany/aboutporsche/christophorusmagazine/archive/374/articleoverview/article01/. – Letzter Zugriff 01.08.2024

[5] Sauer, Dirk U.: Grundlagen der Impedanzspektroskopie fur die Charakterisierung von Batterien. In: *Technische Mitteilungen* 99 (2006), 01, S. 74–80

[6] Rastegarpanah, Alireza ; Hathaway, Jamie ; Ahmeid, Mohamed ; Lambert, Simon ; Walton, Allan ; Stolkin, Rustam: A rapid neural network–based state of health estimation scheme for screening of end of life electric vehicle batteries. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 235 (2020), September, Nr. 3, S. 330–346. http://dx.doi.org/10.1177/0959651820953254. – DOI 10.1177/0959651820953254. – ISSN 2041–3041

[7] E-One Moli Energy Corp: *PRODUCT DATA SHEET MODEL INR-21700-P45B*. https://www.molicel.com/wp-content/uploads/INR21700P45B_1.2_Product-Data-Sheet-of-INR-21700-P45B-80109.pdf. – Letzter Zugriff 01.08.2024

[8] Traco Electronic AG: *TMR 6WI Datasheet*. https://www.tracopower.com/tmr6wi-datasheet. – Letzter Zugriff 24.04.2024

[9] Specovius, Joachim: *Grundkurs Leistungselektronik: Bauelemente, Schaltungen und Systeme*. Springer Fachmedien Wiesbaden, 2015. http://dx.doi.org/10.1007/978-3-658-03309-5. http://dx.doi.org/10.1007/978-3-658-03309-5. – ISBN 9783658033095

[10] Sharp Microelectronics: *PC817 Series Datasheet*. https://www.digikey.de/htmldatasheets/production/34186/0/0/1/pc817-series.pdf. – Letzter Zugriff 16.08.2024

[11] Alpha&Omega Semiconductor: *AOD4184A Datasheet*. https://aosmd.com/res/data_sheets/AOD4184A.pdf. – Letzter Zugriff 06.08.2024

[12] Infineon Technologies AG: *XMC 2Go evaluation kit user guide*. 81726 Munich, Germany: Infineon Technologies AG, April 2022. https://www.infineon.com/dgdl/Infineon-Board_Users_Manual_XMC_2Go_Kit_with_XMC1100_R1.0.pdf-UserManual-v02_00-EN.pdf?fileId=db3a3043444ee5dc014453d6c75078c6. – Letzter Zugriff 26.06.2024

[13] Infineon Technologies AG: *ModusToolbox™ Setup program user guide*. 81726 Munich, Germany: Infineon Technologies AG, September 2023. https://www.infineon.com/dgdl/Infineon-ModusToolbox_1_Setup_Program_1.0_User_Guide-GettingStarted-v01_00-EN.pdf?fileId=8ac78c8c8d2fe47b018e0e4ea3f578b1&redirId=180669. – Letzter Zugriff 26.06.2024

[14] Infineon Technologies AG: *Visual Studio Code for ModusToolbox™user guide*. 81726 Munich, Germany: Infineon Technologies AG, Januar 2024. https://www.infineon.com/dgdl/Infineon-ModusToolbox_3.2_b_VS_Code_User_Guide-UserManual-v01_00-EN.pdf?fileId=8ac78c8c8d2fe47b018e0ea954a278fb. – Letzter Zugriff 26.06.2024

[15] Infineon Technologies AG: *XMC1100 Reference Manual*. 81726 Munich, Germany, August 2016. https://www.infineon.com/dgdl/Infineon-xmc1100-AB_rm-UM-v01_03-EN.pdf?fileId=5546d46249cd1014014a0a8438a65e29. – Letzter Zugriff 26.06.2024

[16] Liechti, Chris: *PySerial Documentaion*, 2020. https://pyserial.readthedocs.io/en/latest/index.html. – Letzter Zugriff 29.07.2024

[17] PyVISA Authors: *PyVISA: Control your instruments with Python*, 2023. https://pyvisa.readthedocs.io/en/stable/. – Letzer Zugriff 29.07.2024

[18] Hassan, Warda H. A. ; Kamh, Sanaa A. ; Soliman, Fouad A. S. ; Abd El-Basit, Wafaa: A closer look to the factors affecting the switching transient times of optocouplers. In: *Optical and Quantum Electronics* 56 (2024), Januar, Nr. 3. http://dx.doi.org/10.1007/s11082-023-06099-4. – DOI 10.1007/s11082–023–06099–4. – ISSN 1572–817X

[19] FUELCON AG: *TrueData-EIS IMPEDANCE SPECTRUM ANALYZER*. https://www.europages.com/filestore/product/9d/10/product_117a7870.pdf. Version: 2018. – Letzter Zugriff 16.08.2024

[20] KEPCO, INC: *QUICK START GUIDE BOP 1KW-MG/ME/MGL/MEL POWER SUPPLY*. https://www.kepcopower.com/support/2281692-r10.pdf. Version: 2021. – Letzter Zugriff 27.08.2024

A Anhang

A.1 Mikrocontroller Software

A.1.1 Mikrocontroller Programm Modi, Zustände und Pin Modul Headerfile

```
#ifndef MODES_STATES_PINS_H
#define MODES_STATES_PINS_H

#include <XMC1100.h>

//  Operating modes
typedef enum { MODE_RL, MODE_LR, MODE_BP, MODE_IDLE } mode_t;

typedef enum {
  PIN_PRI_1_ON  = PORT0_OMR_PS7_Msk,
  PIN_SEK_1_ON  = PORT0_OMR_PS15_Msk,
  PIN_CRTL_1_ON = PORT0_OMR_PS8_Msk,
  PIN_PRI_2_ON  = PORT0_OMR_PS14_Msk,
  PIN_SEK_2_ON  = PORT0_OMR_PS6_Msk,
  PIN_CRTL_2_ON = PORT0_OMR_PS9_Msk
} pin_setter_t;

typedef enum {
  PIN_PRI_1_OFF  = PORT0_OMR_PR7_Msk,
  PIN_SEK_1_OFF  = PORT0_OMR_PR15_Msk,
  PIN_CRTL_1_OFF = PORT0_OMR_PR8_Msk,
  PIN_PRI_2_OFF  = PORT0_OMR_PR14_Msk,
  PIN_SEK_2_OFF  = PORT0_OMR_PR6_Msk,
  PIN_CRTL_2_OFF = PORT0_OMR_PR9_Msk
} pin_resetter_t;

typedef enum {
  // MODE_IDLE (all off)
  MODE_IDLE_OUT     = (PIN_PRI_1_OFF | PIN_SEK_1_OFF | PIN_CRTL_1_OFF | PIN_PRI_2_OFF |
       PIN_SEK_2_OFF | PIN_CRTL_2_OFF),
  // MODE_RL_OFF_OUT   (Traco 1 off & Traco 2 idle)
  MODE_RL_OFF_OUT   = (PIN_PRI_1_OFF | PIN_SEK_1_OFF | PIN_CRTL_1_OFF | PIN_PRI_2_ON |
       PIN_SEK_2_OFF | PIN_CRTL_2_ON),
  // MODE_RL_ON_OUT    (Traco 1 off & Traco 2 output)
  MODE_RL_ON_OUT    = (PIN_PRI_1_OFF | PIN_SEK_1_OFF | PIN_CRTL_1_OFF | PIN_PRI_2_ON |
       PIN_SEK_2_ON | PIN_CRTL_2_ON),
  // MODE_LR_OFF_OUT   (Traco 1 idle & Traco 2 off )
  MODE_LR_OFF_OUT   = (PIN_PRI_1_ON | PIN_SEK_1_OFF | PIN_CRTL_1_ON | PIN_PRI_2_OFF |
       PIN_SEK_2_OFF | PIN_CRTL_2_OFF),
  // MODE_LR_ON_OUT    (Traco 1 idle & Traco 2 off)
  MODE_LR_ON_OUT    = (PIN_PRI_1_ON | PIN_SEK_1_ON | PIN_CRTL_1_ON | PIN_PRI_2_OFF |
       PIN_SEK_2_OFF | PIN_CRTL_2_OFF),
  // MODE_BP_OFF_OUT   (Traco 1 idle & Traco 2 idle )
  MODE_BP_OFF_OUT   = (PIN_PRI_1_ON | PIN_SEK_1_OFF | PIN_CRTL_1_ON | PIN_PRI_2_ON |
       PIN_SEK_2_OFF | PIN_CRTL_2_ON),
  // MODE_BP_RL_ON_OUT (Traco 1 idle & Traco 2 output)
  MODE_BP_RL_ON_OUT = (PIN_PRI_1_ON | PIN_SEK_1_OFF | PIN_CRTL_1_ON | PIN_PRI_2_ON |
       PIN_SEK_2_ON | PIN_CRTL_2_ON),
  // MODE_BP_LR_ON_OUT (Traco 1 output & Traco 2 idle)
```

```
  MODE_BP_LR_ON_OUT = (PIN_PRI_1_ON | PIN_SEK_1_ON | PIN_CRTL_1_ON | PIN_PRI_2_ON |
      PIN_SEK_2_OFF | PIN_CRTL_2_ON)
} output_t;

const extern output_t lookupMatrix[4][4];

#endif
```

A.1.2 Mikrocontroller Programm Modi, Zustände und Pin Modul

```
#include "modes_states_pins.h"

const output_t lookupMatrix[4][4] = {{MODE_RL_OFF_OUT, MODE_RL_ON_OUT, MODE_RL_OFF_OUT,
    MODE_RL_ON_OUT},
                        {MODE_LR_OFF_OUT, MODE_LR_ON_OUT, MODE_LR_OFF_OUT, MODE_LR_ON_OUT},
                          {MODE_BP_OFF_OUT, MODE_BP_RL_ON_OUT, MODE_BP_OFF_OUT,
                              MODE_BP_LR_ON_OUT},
                          {MODE_IDLE_OUT, MODE_IDLE_OUT, MODE_IDLE_OUT, MODE_IDLE_OUT}};
```

A.1.3 Mikrocontroller Programm Timer Modul Headerfile

```
#ifndef TIMER_H
#define TIMER_H

#include "modes_states_pins.h"
#include <XMC1100.h>
#include <math.h>


void setPeriodTime(const double_t period, const uint8_t dutyCycle);

void setFrequency(const double_t frequency, const uint8_t dutyCycle, const mode_t* const mode);

void setPeriodCount(const uint8_t periodCountValue,const mode_t* const mode);

#endif
```

A.1.4 Mikrocontroller Programm Timer Modul

```
#include "timer.h"
#include "modes_states_pins.h"
#include "cycfg_peripherals.h"
#include "xmc_ccu4.h"
#include <math.h>
#include <XMC1100.h>
// Prescaler Value
static const double_t timerTickTimes[] = {1.5625E-8, 3.125e-8, 6.25e-8, 1.25e-7, 2.5e-7, 5e-7,    1e-6,    2e-6,
                        4e-6,      8e-6,     1.6e-5,  3.2e-5,  6.4e-5, 1.28e-4, 2.56e-4, 5.12e-4};

static const XMC_CCU4_SLICE_PRESCALER_t prescalerValues[] = {
   XMC_CCU4_SLICE_PRESCALER_1,    XMC_CCU4_SLICE_PRESCALER_2,    XMC_CCU4_SLICE_PRESCALER_4,
   XMC_CCU4_SLICE_PRESCALER_8,    XMC_CCU4_SLICE_PRESCALER_16,
       XMC_CCU4_SLICE_PRESCALER_32,
   XMC_CCU4_SLICE_PRESCALER_64,   XMC_CCU4_SLICE_PRESCALER_128,
       XMC_CCU4_SLICE_PRESCALER_256,
   XMC_CCU4_SLICE_PRESCALER_512,  XMC_CCU4_SLICE_PRESCALER_1024,
       XMC_CCU4_SLICE_PRESCALER_2048,
   XMC_CCU4_SLICE_PRESCALER_4096, XMC_CCU4_SLICE_PRESCALER_8192,
       XMC_CCU4_SLICE_PRESCALER_16384,
   XMC_CCU4_SLICE_PRESCALER_32768};

void setPeriodTime(double_t const period, const uint8_t dutyCycle) {
```

```
  // Prescaler stoppen um ihn einstellen zu können
  XMC_CCU4_StopPrescaler(ccu4_0_HW);
  uint32_t ticksPeriod  = 0;
  uint32_t ticksCompare = 0;

  // Niedrigst möglichen Prescaler Wert ermitteln der eine vollständige Periode laufen kann
  for (int i = 0; i < 16; i++) {
    if ((timerTickTimes[i] * 0xFFFF) >= period) {
      XMC_CCU4_SLICE_SetPrescaler(timerFreq_HW, prescalerValues[i]);
      // timer ticks period = periodeValue in s / time between timer ticks
      ticksPeriod  = (uint32_t)round(period / timerTickTimes[i]) - 1;
      // timer ticks compare output on for duty cycle in %
      ticksCompare = (uint32_t)round(ticksPeriod * ((100 - dutyCycle) / 100.0));
      break;
    }
  }

  XMC_CCU4_StartPrescaler(ccu4_0_HW);

  XMC_CCU4_SLICE_SetTimerCompareMatch(timerFreq_HW, ticksCompare);
  XMC_CCU4_SLICE_SetTimerPeriodMatch(timerFreq_HW, ticksPeriod);

  XMC_CCU4_EnableShadowTransfer(ccu4_0_HW, (XMC_CCU4_SHADOW_TRANSFER_SLICE_0 |
       XMC_CCU4_SHADOW_TRANSFER_SLICE_1));
  XMC_CCU4_SLICE_StartTimer(timerFreq_HW);
}

void setFrequency(const double_t frequency, const uint8_t dutyCycle, const mode_t* const mode) {
  // Nur Frequenzen zwischen 100 mHz und 10 kHz
  if (frequency >= 0.1 && frequency <= 10000) {
    // Frequenz verdoppeln weil hier vier statt zwei Zustandswechsel stattfinden müssen
    if (*mode == MODE_BP)
      setPeriodTime(1.0 / (frequency * 2.0), dutyCycle);
    else
      setPeriodTime(1.0 / frequency, dutyCycle);
  }
}

void setPeriodCount(const uint8_t periodCountValue, const mode_t* const mode) {
  XMC_CCU4_SLICE_StopClearTimer(timerPeriodCount_HW);
  if (*mode == MODE_RL || *mode == MODE_LR) {
    XMC_CCU4_SLICE_SetTimerPeriodMatch(timerPeriodCount_HW, (2 * periodCountValue));
  } else if (*mode == MODE_BP) {
    XMC_CCU4_SLICE_SetTimerPeriodMatch(timerPeriodCount_HW, (4 * periodCountValue));
  }
  XMC_CCU4_EnableShadowTransfer(ccu4_0_HW, XMC_CCU4_SHADOW_TRANSFER_SLICE_2);
  XMC_CCU4_SLICE_StartTimer(timerPeriodCount_HW);
}
```

A.1.5 Mikrocontroller Programm UART Kontrolle Modul Headerfile

```
#ifndef UART_CONTROL_H
#define UART_CONTROL_H

#include "modes_states_pins.h"
#include <XMC1100.h>
#include <stdint.h>

void stopStimulation(const mode_t *const mode, uint16_t *const state);

void modeSwitch(const uint8_t modeControlCode, mode_t *const mode, const uint16_t* const state);

void frequencySwitch(const uint16_t frequencyControlCode, const uint8_t dutyCycle,
                     const mode_t* const mode);

void uartCommandEvaluation(const uint16_t frequency, const uint8_t dutyCycle,
                           const uint8_t periodCount, mode_t *const mode, uint16_t *const state);
```

```
#endif
```

A.1.6 Mikrocontroller Programm UART Kontrolle Modul

```
#include "uart_control.h"
#include "cycfg_peripherals.h"
#include "modes_states_pins.h"
#include "timer.h"
#include "xmc_ccu4.h"
#include <XMC1100.h>
#include <stdint.h>
#include <xmc_scu.h>

void stopStimulation(const mode_t *const mode, uint16_t *const state) {
  XMC_CCU4_SLICE_StopClearTimer(timerFreq_HW);
  // Sichereren Zustand herstellen
  PORT0->OMR = lookupMatrix[*mode][0];
  *state     = 0;
}

void modeSwitch(const uint8_t modeControlCode, mode_t *const mode, const uint16_t* const state) {
  switch ( modeControlCode) {
  case 0xFF:
    *mode = MODE_LR;
    break;
  case 0xFE:
    *mode = MODE_RL;
    break;
  case 0xFD:
    *mode = MODE_BP;
    break;
  case 0xFC:
  // Wenn keine gültige eingabe automatisch in sicheren IDLE Zustand
  default:
    *mode = MODE_IDLE;
    break;
  }
  // Ausgang in abhängikeit von neuem Zustand setzen
  PORT0->OMR = lookupMatrix[*mode][*state];
}

void frequencySwitch(const uint16_t frequencyControlCode, const uint8_t dutyCycle,
                     const mode_t* const mode) {
  // Kodierte werte für Frequenzen mit Dezimalstelle
  switch (frequencyControlCode) {
  case 0xFFFF:
    setFrequency(0.1, dutyCycle, mode);
    break;
  case 0xFFFE:
    setFrequency(0.4, dutyCycle, mode);
    break;
  case 0xFFFD:
    setFrequency(0.8, dutyCycle, mode);
    break;
  default:
    setFrequency(frequencyControlCode, dutyCycle, mode);
    break;
  }
}

void uartCommandEvaluation(const uint16_t frequency, const uint8_t dutyCycle,
                           const uint8_t periodCount, mode_t *const mode, uint16_t *const state) {
  // Wenn die Frequenz 0 empfangen wird wird die Anregung sofort gestoppt
  // Ansonsten wird geprüft ob über den Tastgrad eine kodierte Anweisung (FF, FE, FD oder FC) zum Moduswechsel
  // übertragen wird Wenn ein Modus wechsel durchgeführt wird zunächst die Anregung gestoppt und dann der Startzustand
  // des Modus auf den Port ausgegeben
```

```
  //! Nach einem Moduswechsel sollte mindestens 30 ms gewartet werden, damit die Tracos sich initialisieren können
  // Alles Null == Master Reset
  if (frequency == 0 && dutyCycle == 0 && periodCount == 0) {
    XMC_SCU_RESET_AssertMasterReset();
  } else if (frequency == 0) {
    stopStimulation(mode, state);
  } else if (dutyCycle <= 0xFF && dutyCycle >= 0xFC) {
    // Modus Wechsel
    stopStimulation(mode, state);
    modeSwitch(dutyCycle, mode, state);
  } else if (dutyCycle >= 0 && dutyCycle <= 100) {
    // Perioden Zähler setzen (0 = Keine begrezung der Periodenzahl)
    if (periodCount != 0) {
      setPeriodCount(periodCount, mode);
    }

    // Frequenz einstellen
    frequencySwitch(frequency, dutyCycle, mode);
  }
}
```

A.1.7 Mikrocontroller Programm Main Funktion

```
/*********************************************
 * File Name:   main.c
 *
 * Description: main.c for the EIS Controller board
 *
 *
 *
 *********************************************
 *
 * Copyright (c) 2015-2021, Infineon Technologies AG All rights reserved.
 *
 * Boost Software License - Version 1.0 - August 17th, 2003
 *
 * Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and
 *accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute,
 *and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the
 *Software is furnished to do so, all subject to the following:
 *
 * The copyright notices in the Software and this entire statement, including the above license grant, this restriction
 *and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative
 *works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code
 *generated by a source language processor.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
      INCLUDING BUT NOT LIMITED TO THE
 *WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND
      NON-INFRINGEMENT. IN NO EVENT SHALL THE
 *COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER
      LIABILITY, WHETHER IN
 *CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
      THE USE OR OTHER DEALINGS IN
 *THE SOFTWARE.
 *
 *********************************************/

#include "cy_utils.h"
#include "cybsp.h"
#include "cycfg_peripherals.h"
#include "modes_states_pins.h"
#include "uart_control.h"
#include <XMC1100.h>
#include <stdint.h>
#include <xmc_scu.h>
#include <xmc_uart.h>
```

```
#include <xmc_usic.h>
#include <xmc_wdt.h>


// Betriebsmodus Variable
static mode_t mode = MODE_IDLE;
// Zustands variable
static uint16_t state     = 0;

void ccu4_0_SR0_INTERRUPT_HANDLER() {
  // Ändern des Zustands in abhängikeit von Betriebsmodus und Zustand
  XMC_CCU4_SLICE_ClearEvent(timerFreq_HW, XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP);

  // Wenn der letzte Zustand ereicht ist, wird von vorne begonnen, ansonsten wird der Zustand inkrementiert
  state      = (state == 3) ? 0 : state + 1;
  // MC Ausgang beschreiben
  PORT0->OMR = lookupMatrix[mode][state];
}

void ccu4_0_SR1_INTERRUPT_HANDLER() {
  // Ändern des Zustands in abhängikeit von Betriebsmodus und Zustand
  XMC_CCU4_SLICE_ClearEvent(timerFreq_HW, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);

  // Wenn der letzte Zustand ereicht ist, wird von vorne begonnen, ansonsten wird der Zustand inkrementiert
  state      = (state == 3) ? 0 : state + 1;
  // MC Ausgang beschreiben
  PORT0->OMR = lookupMatrix[mode][state];
}

void ccu4_0_SR2_INTERRUPT_HANDLER() {
  // Interupt handler nachdem periodCountValue ereicht ist (Wird für jeden compare und period Match erhöht)
  XMC_CCU4_SLICE_ClearEvent(timerPeriodCount_HW, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);
  stopStimulation(&mode, &state);
}

void uart_RECEIVE_BUFFER_STANDARD_EVENT_HANDLER() {
  XMC_USIC_CH_RXFIFO_ClearEvent(uart_HW, XMC_USIC_CH_RXFIFO_EVENT_STANDARD);

  uint8_t receivedData[4];
  uint8_t rxIndex = 0;

  // Wenn keine Steurungs Daten Interrupt Handler verlassen um Fehler zu vermeiden
  if (XMC_USIC_CH_RXFIFO_IsEmpty(uart_HW)) {
    return;
  }

  // Lesen der Daten bis der Input Puffer leer ist und maximal 4 Bytes Empfangen wurden
  while ((!XMC_USIC_CH_RXFIFO_IsEmpty(uart_HW)) && rxIndex < 4) {
    receivedData[rxIndex++] = XMC_UART_CH_GetReceivedData(uart_HW);
  }

  // Wenn RXFIFO nicht leer weil zu viele Daten -> Leeren
  if (!XMC_USIC_CH_RXFIFO_IsEmpty(uart_HW)) {
    XMC_USIC_CH_RXFIFO_Flush(uart_HW);
  }

  // Empfangen Daten zurückschreiben
  // Signal Start
  XMC_UART_CH_Transmit(uart_HW, '\n');

  for (int i = 0; i < rxIndex; i++) {
    XMC_UART_CH_Transmit(uart_HW, receivedData[i]);
  }

  //Signal Stopp
  XMC_UART_CH_Transmit(uart_HW, '\n');

  // Frequenz wert aus den letzten empfangenen bytes zusammen setzen
  uint16_t newFrequency = (receivedData[0] « 8) + receivedData[1];
```

```
  uint8_t dutyCycle = receivedData[2];
  // Anzahl der Perioden
  uint8_t periodCount = receivedData[3];

  // Auswertung der Empgangen Daten
  uartCommandEvaluation(newFrequency, dutyCycle, periodCount, &mode, &state);
}


int main(void) {
  cy_rslt_t result;

  /* Initialize the device and board
   * peripherals */
  result = cybsp_init();
  if (result != CY_RSLT_SUCCESS) {
    CY_ASSERT(0);
  }

  // Wenn der letzte Reset vom Watchdog Timer verursacht wurde sicheren Zustand herstellen
  if (XMC_SCU_RESET_REASON_WATCHDOG & XMC_SCU_RESET_GetDeviceResetReason()) {
    mode       = MODE_IDLE;
    PORT0->OMR = MODE_IDLE_OUT;
    XMC_SCU_RESET_ClearDeviceResetReason();
    // Dauerschleife bis zum neuen Reset
    while (true) {
      XMC_WDT_Service();
      XMC_UART_CH_Transmit(uart_HW, 'x');
    }
  }

  // Start Betriebsmodus
  mode       = MODE_IDLE;
  // Start Ausgabe
  PORT0->OMR = MODE_IDLE_OUT;

  // Sicherstellen das der RXFIFO leer ist
  XMC_USIC_CH_RXFIFO_Flush(uart_HW);

  NVIC_SetPriority(ccu4_0_SR0_IRQN, 0U);
  NVIC_EnableIRQ(ccu4_0_SR0_IRQN);

  NVIC_SetPriority(ccu4_0_SR1_IRQN, 0U);
  NVIC_EnableIRQ(ccu4_0_SR1_IRQN);

  NVIC_SetPriority(ccu4_0_SR2_IRQN, 1U);
  NVIC_EnableIRQ(ccu4_0_SR2_IRQN);

  NVIC_SetPriority(uart_RECEIVE_BUFFER_STANDARD_EVENT_IRQN, 2U);
  NVIC_EnableIRQ(uart_RECEIVE_BUFFER_STANDARD_EVENT_IRQN);

  // Senden das MC bereit
  XMC_UART_CH_Transmit(uart_HW, 0x40);

  for (;;) {
    XMC_WDT_Service();
  }
}

/* [] END OF FILE */
```

A.2 Python Skripte

A.2.1 Skript für die Steuerung des Mikrocontroller über die Kommandozeile

```
import serial
import sys

#Einlesen der zu sendenen Variablen von der Konsole
frequency = int(sys.argv[1])
dutyCycle = int(sys.argv[2])
periodCount = int(sys.argv[3])

#Seriele Schnittstelle starten
ser = serial.Serial("COM7", 115200,timeout=None)
ser.reset_output_buffer()
ser.reset_input_buffer()

#Daten an MC schreiben dabei Umwandlung in passendes Format
ser.write(frequency.to_bytes(2, "big"))
ser.write(dutyCycle.to_bytes(1, "big"))
ser.write(periodCount.to_bytes(1, "big"))

#Warten bis der MC das Beginnen der Rüchsendung signalisiert
ser.read_until(b"\x0a")

#Daten zurücklesen
receivedFreq = ser.read(2)
receivedDuty = ser.read(1)
receivedCount = ser.read(1)

#Warten bis der MC das Ende der Rücksendung signalisiert
ser.read_until(b"\x0a")

#Rückumwandlung der Empfagenen Daten
receivedFreqCon = int.from_bytes(receivedFreq, "big", signed=False)
receivedDutyCon = int.from_bytes(receivedDuty, "big", signed=False)
receivedCountCon = int.from_bytes(receivedCount, "big", signed=False)

#Wenn Daten korrekt übertragen
if(receivedFreqCon == frequency and receivedDutyCon == dutyCycle and receivedCountCon == periodCount):
    print(f"{receivedFreqCon} {receivedDutyCon} {receivedCountCon}")
#Wenn Daten inkorrekt übertragen
else:
   print("Daten nicht korrekt übertragen -> Reset")
   print(f"{receivedFreqCon} {receivedDutyCon} {receivedCountCon}")
   #Reset am MC auslösen
   ser.write(bytes([0,0,0,0]))
   ser.write(bytes([0,0,0,0]))
   #Warten bis MC Neustart signalisiert
   ser.read_until(b"\x40")
```

A.2.2 Skript für das Automatisieren einer Messung

```
from datetime import datetime  # std library
import pyvisa as visa  # https://pyvisa.readthedocs.org/en/stable/
import serial
import time


def setExitation(frequency, dutyCycle, periodCount, ser=serial.Serial()):
    # Wichtig reihen folge in der die Daten übertragen werden
    ser.write(frequency.to_bytes(2, "big"))
```

```
    ser.write(dutyCycle.to_bytes(1, "big"))
    ser.write(periodCount.to_bytes(1, "big"))


# Adresse des Oszilloskop aus Instrumen Manager übernehmen
# ================================================
visa_address = "USB::0x0699::0x0527::C019583::INSTR"
rm = visa.ResourceManager()
# ================================================

# Messfrequenzen

frequencys = [
    10000,
    8000,
    4000,
    1000,
    800,
    400,
    100,
    80,
    40,
    10,
    8,
    4,
    1,
    0.8,
    0.4,
    0.1,
]
# Kompensations Tastgrade
dutyCycleCompLR = [16, 21, 35, 46, 47, 49, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50]
dutyCycleCompRL = [16, 20, 35, 56, 47, 49, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50]
dutyCycleCompBP = [14, 13, 21, 42, 44, 47, 49, 49, 50, 50, 50, 50, 50, 50, 50, 50]


# Betriebsmodi übertragung erfolgt kodiert über duty cycle
LR = 255
RL = 254
BP = 253
IDLE = 252

# Einstellungen
# Comport anpassen
ser = serial.Serial("COM7", 115200)
# Betriebsmodi wählen
MODE = BP
# Anzahl der Datenpunkte
recordLength = 120000
# Mit wie viele Perioden angeregt wird
periodeCount = 50
# Wie viele Durchläufe
runs = 1

# Einstellungen Oszi
scope = rm.open_resource(visa_address)

# Zeit wie lange auf das Oszilloskop gewartet wird
# Muss länger als längste messdauer sein
scope.timeout = 2500e3  # ms

# Standard einstellungen nicht anpassen
scope.encoding = "latin_1"
scope.read_termination = "\n"
scope.write_termination = None
scope.write("*cls")  # clear ESR

# ID Ausgabe zum Testen
print(scope.query("*IDN?"))
```

```
# Setzen des Speicherorts für Daten
scope.write('FILESystem:CWD "E:/"')
print(scope.query("FILESystem:CWD?"))

# Farbschema wenn bilder gespeichert werden
scope.write("SAVE:IMAGE:COMPOSITION INVERTED")

# Nach gewählten Betriebsmodus Kompensations Tastgrade zuordnen
if MODE == LR:
    exitatationSettings = zip(frequencys, dutyCycleCompLR)
elif MODE == RL:
    exitatationSettings = zip(frequencys, dutyCycleCompRL)
elif MODE == BP:
    exitatationSettings = zip(frequencys, dutyCycleCompBP)

# Stop excitation
setExitation(100, IDLE, 0, ser)
time.sleep(1)

# Mode set
setExitation(100, MODE, 0, ser)

# Warten für Traco Intialisierung
time.sleep(0.5)

# Anzahl Datenpunkte festlegen
scope.write(f"HORIZONTAL:RECORDLENGTH {recordLength}")

#Alle Durchläufe
for run in range(1, runs + 1):
    #Alle Messfrequenzen
    for f, dc in exitatationSettings:
        print(f"Durchlauf: {run} Anregefrequenz: {f} Hz")

        # Messdauer
        periodeDuration = 1 / f
        measurementTime = periodeDuration * periodeCount

        # Messdauer verlängern für vollständige Aufnahme durch nicht mögliche Abtastraten des Oszi
        measurementTime = measurementTime + measurementTime / 100 * 30
        if f == 1000 or f == 10000:
            measurementTime = measurementTime + measurementTime / 100 * 10
        samplerate = recordLength / measurementTime

        # Samplerate setzen
        print(f"Samplerate Berechnet: {samplerate}")
        scope.write(f"HORIZONTAL:SAMPLERATE {samplerate}")
        samplerateOszi = scope.query("HORIZONTAL:SAMPLERATE?")
        print(f"Samplerate Oszi: {samplerateOszi}")

        # Single Sequence mode
        scope.write("Acquire:Stopafter Sequence")
        scope.write("Acquire:state RUN")

        # Warten bevor die Anregung gestartet wird
        time.sleep(1)

        # Für unter 1 Hz ansteuerung über codes
        if f == 0.1:
            setExitation(0xFFFF, dc, periodeCount, ser)
        elif f == 0.4:
            setExitation(0xFFFE, dc, periodeCount, ser)
        elif f == 0.8:
            setExitation(0xFFFD, dc, periodeCount, ser)
        else:
            setExitation(int(f), dc, periodeCount, ser)

        # Speichern des Wellenform
```

```
        r = scope.query("*opc?")  # sync
        scope.write(f'SAVE:WAVEFORM ALL, "Run_{run}_{f}_Hz.MAT"')
        r = scope.query("*opc?")  # sync

        #Bild speichern
        scope.write(f'SAVE:IMAGE "Run_{run}_{f}_Hz.png"')
        r = scope.query("*opc?")  # sync

        # Stop excitation
        setExitation(0, 50, 0, ser)
        time.sleep(2)
#MC in IDLE Mode
setExitation(100, IDLE, 0, ser)
scope.close()
rm.close()
ser.close()
```

A.3 Matlab Skripte

A.3.1 Skript für das Messen der Frequenz

```
%Abweichungen der Messfrequez an Messwiderstand und Optokoppler vom Sollwert
addpath("..\..\Mat2Tikz\");

close all;
clear;
format short g;

filesRL = dir("C:\Users\niklas\Documents\Bachelor\Messung\2_Tastgrad\1_RL\");
filesRL = (filesRL(~[filesRL.isdir]));

filesBP = dir("C:\Users\niklas\Documents\Bachelor\Messung\2_Tastgrad\3_BP\");
filesBP = (filesBP(~[filesBP.isdir]));

freq = [0.1 0.4 0.8 1 4 8 10 40 80 100 400 800 1000 4000 8000 10000];

f1 = 1;
f2 = 1;
%Daten RL laden
for i = 1:length(filesRL)
    %Messwiederstand Links
    if contains(filesRL(i).name,"ch1")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        Fs = 1/sampleInterval;

        %Peaks finden
        [pks,locs] = findpeaks(diff(data(2500:end)),Fs,'MinPeakProminence',0.1);
        %Periodenlänge berechnen aus Differenz der Peaks gemittelt
        Period = mean((locs(2:end)+time(2500))-(locs(1:end-1)+time(2500)));
        %Frequenz berechnen und runden
        freqTableRL{f1,4} = round(1/Period,4);
        f1 = f1 + 1;
        %Optokoppler Links
    elseif contains(filesRL(i).name,"ch2")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        Fs = 1/sampleInterval;
        %Peaks finden
        [pks,locs] = findpeaks(diff(data(2500:end)),Fs,'MinPeakProminence',0.1);
        %Periodenlänge berechnen aus Differenz der Peaks gemittelt
        Period = mean((locs(2:end)+time(2500))-(locs(1:end-1)+time(2500)));
        %Frequenz berechnen und runden
        freqTableRL{f2,2} = round(1/Period,4);
        f2 = f2 + 1;
    end
```

```
end

f1 = 1;
f2 = 1;

%Daten BP laden
for i = 1:length(filesBP)
    %Messwiederstand Links
    if contains(filesBP(i).name,"ch1")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        %Für 4, 8 und 10 kHz
        if contains(filesBP(i).name,"4000_Hz") | contains(filesBP(i).name,"8000_Hz") | contains(filesBP(i).name, "10000_Hz")
            Fs = 1/sampleInterval;
            %Peaks finden mit anderer MinPeakProminence
            [pks,locs] = findpeaks(data,Fs,'MinPeakProminence',0.2);
            %Periodenlänge berechnen aus Differenz der Peaks gemittelt
            Period = mean((locs(2:end))-(locs(1:end-1)));
            %Frequenz berechnen und runden
            freqTableBP{f1,4} = round(1/Period,4);
            f1 = f1 + 1;
        else
            Fs = 1/sampleInterval;
            %Gleichanteil abziehen
            data = data - mean(data);
            %Wenn Daten negativ sind, auf 0 setzen
            data(data < 0) = 0;
            %Peaks finden beginnen bei 2500, da vorher keine Peaks
            [pks,locs] = findpeaks(diff(data(2500:end)),Fs,'MinPeakProminence',0.3);
            %Periodenlänge berechnen aus Differenz der Peaks gemittelt
            Period = mean((locs(2:end)+time(2500))-(locs(1:end-1)+time(2500)));
            %Frequenz berechnen und runden
            freqTableBP{f1,4} = round(1/Period,4);
            f1 = f1 + 1;
        end
        %Optokoppler Links
    elseif contains(filesBP(i).name,"ch2")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        %Für 4, 8 und 10 kHz
        if contains(filesBP(i).name,"4000_Hz") | contains(filesBP(i).name,"8000_Hz") | contains(filesBP(i).name, "10000_Hz")
            Fs = 1/sampleInterval;
            %Peaks finden mit anderer MinPeakProminence
            [pks,locs] = findpeaks(data,Fs,'MinPeakProminence',0.6);
            %Periodenlänge berechnen aus Differenz der Peaks gemittelt
            Period = mean((locs(2:end))-(locs(1:end-1)));
            %Frequenz berechnen und runden
            freqTableBP{f2,2} = round(1/Period,4);
            f2 = f2 + 1;
        else
            Fs = 1/sampleInterval;
            %Gleichanteil abziehen
            data = data - mean(data);
            %Peaks finden beginnen bei 2500, da vorher keine Peaks
            [pks,locs] = findpeaks(diff(data(2500:end)),Fs,'MinPeakProminence',0.2);
            %Periodenlänge berechnen aus Differenz der Peaks gemittelt
            Period = mean((locs(2:end)+time(2500))-(locs(1:end-1)+time(2500)));
            %Frequenz berechnen und runden
            freqTableBP{f2,2} = round(1/Period,4);
            f2 = f2 + 1;
        end
    end
end

%Nach Frequenz sortieren
freqTableRL = sortrows(freqTableRL,2);
freqTableBP = sortrows(freqTableBP,2);
%Abweichung berechnen für Messwiderstand und Optokoppler in %
for i = 1:length(freqTableRL)

    freqTableRL{i,3} = round((freqTableRL{i,2} / freq(i) - 1) * 100,4);
```

```
    freqTableRL{i,1} = freq(i);
    freqTableRL{i,5} = round((freqTableRL{i,4} / freq(i) - 1) * 100,4);
end
%Abweichung berechnen für Messwiderstand und Optokoppler in %
for i = 1:length(freqTableBP)
    freqTableBP{i,3} = round((freqTableBP{i,2} / freq(i) - 1) * 100,4);
    freqTableBP{i,1} = freq(i);
    freqTableBP{i,5} = round((freqTableBP{i,4} / freq(i) - 1) * 100,4);
end

%Tabelle erstellen für Latex mit spalten: Frequenz, Frequenz Optokoppler, Abweichung Optokoppler, Frequenz Messwiderstand,
     Abweichung Messwiderstand,
Rl = latex(sym(freqTableRL))
BP = latex(sym(freqTableBP))
```

A.3.2 Skript für das Messen des Tastgrads

```
%Tastgradabweichung berechnen
addpath("..\..\Mat2Tikz\");

close all;
clear;
format short g;

filesRL = dir("C:\Users\niklas\Documents\Bachelor\Messung\2_Tastgrad\1_RL\");
filesRL = (filesRL(~[filesRL.isdir]));

filesBP = dir("C:\Users\niklas\Documents\Bachelor\Messung\2_Tastgrad\3_BP\");
filesBP = (filesBP(~[filesBP.isdir]));

freq = [0.1 0.4 0.8 1 4 8 10 40 80 100 400 800 1000 4000 8000 10000];

f1 = 1;
f2 = 1;

for i = 1:length(filesRL)
    %Messwiderstand
    if contains(filesRL(i).name,"ch1")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        Fs = 1/sampleInterval;
        %Gemittelter Tastgrad in Prozent
        tastTableRL{f1,4} = round(mean(dutycycle(data(1:end-1000)))*100,4);
        tastTableRL{f1,1} = filesRL(i).name;
        f1 = f1 + 1;
    %Optokoppler
    elseif contains(filesRL(i).name,"ch2")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        %Gemittelter Tastgrad in Prozent
        tastTableRL{f2,2} = round(mean(dutycycle((data(1:end-1000))))*100,4);
        f2 = f2 + 1;
    end
end

f1 = 1;
f2 = 1;

for i = 1:length(filesBP)
    %Messwiderstand
    if contains(filesBP(i).name,"ch1")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        %Gleichanteil abziehen
        data = (data-mean(data));
        %Negativwerte auf 0 setzen
        data(data<0) = 0;
        %Geminittelter Tastgrad in Prozent
        tastTableBP{f1,4} = round(mean(dutycycle((data(1:end-1000))))*100,4);
        tastTableBP{f1,1} = filesBP(i).name;
```

```
        f1 = f1 + 1;
    %Optokoppler
    elseif contains(filesBP(i).name,"ch2")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        %Gemittelter Tastgrad in Prozent
        tastTableBP{f2,2} = round(mean(dutycycle((data(1:end-1000))))*100,4);
        f2 = f2 + 1;
    end
end

for i = 1:length(tastTableRL)
    %Frequenz extrahieren
    tastTableRL{i,1} = str2double(extract(tastTableRL{i,1},regexpPattern("(?<=_1_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    %Tastgradabweichung berechnen für Optokoppler
    tastTableRL{i,3} = round((tastTableRL{i,2} / 50 - 1)*100,4);
    %Tastgradabweichung berechnen für Messwiderstand
    tastTableRL{i,5} = round((tastTableRL{i,4} / 50 - 1)*100,4);
end

for i = 1:length(tastTableBP)
    %Frequenz extrahieren
    tastTableBP{i,1} = str2double(extract(tastTableBP{i,1},regexpPattern("(?<=_1_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    %Tastgradabweichung berechnen für Optokoppler
    tastTableBP{i,3} = round((tastTableBP{i,2} / 25 - 1)*100,4);
    %Tastgradabweichung berechnen für Messwiderstand
    tastTableBP{i,5} = round((tastTableBP{i,4} / 25 - 1)*100,4);
end

%Nach Frequenz sortieren
tastTableRL = sortrows(tastTableRL,1);
tastTableBP = sortrows(tastTableBP,1);

tastMatRL = cell2mat(tastTableRL);
tastMatBP = cell2mat(tastTableBP);

%Tabelle für Tastgradabweichung mit spalten Frequenz, Tastgrad Optokoppler, Abweichung Optokoppler, Tastgrad
    Messwiderstand, Abweichung Messwiderstand
Rl = latex(sym(tastMatRL))
Bp = latex(sym(tastTableBP))
```

A.3.3 Skript für das Messen des Ladungsfluss

```
%Bestimmt den Ladungsfluss für die Spannungsdriftmessung
clear all;
format shortG;
%Zu ladende Variablen
myVar = {"time", "data", "recordLength","sampleInterval"};

outFolder = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_4_Drift';

rootdir = "C:\Users\niklas\Documents\Bachelor\Messung\5_Spannugsdrift\2_RL\";


f_01_iL_Rl = load(fullfile(rootdir, "Run_1_0.1_Hz_ch1.mat"), myVar{:});
f_01_iR_Rl = load(fullfile(rootdir, "Run_1_0.1_Hz_ch3.MAT"), myVar{:});
f_01_vL_Rl = load(fullfile(rootdir, "Run_1_0.1_Hz_math1.mat"), myVar{:});
f_01_vR_Rl = load(fullfile(rootdir, "Run_1_0.1_Hz_math2.MAT"), myVar{:});

f_10k_iL_Rl = load(fullfile(rootdir, "Run_1_10000_Hz_ch1.MAT"), myVar{:});
f_10k_iR_Rl = load(fullfile(rootdir, "Run_1_10000_Hz_ch3.MAT"), myVar{:});
f_10k_vL_Rl = load(fullfile(rootdir, "Run_1_10000_Hz_math1.MAT"), myVar{:});
f_10k_vR_Rl = load(fullfile(rootdir, "Run_1_10000_Hz_math2.MAT"), myVar{:});

rootdir = "C:\Users\niklas\Documents\Bachelor\Messung\5_Spannugsdrift\1_BP\";

f_01_iL_Bp = load(fullfile(rootdir, "Run_1_0.1_Hz_ch1.MAT"), myVar{:});
```

```
f_01_iR_Bp = load(fullfile(rootdir, "Run_1_0.1_Hz_ch3.MAT"), myVar{:});
f_01_vL_Bp = load(fullfile(rootdir, "Run_1_0.1_Hz_math1.MAT"), myVar{:});
f_01_vR_Bp = load(fullfile(rootdir, "Run_1_0.1_Hz_math2.MAT"), myVar{:});

f_10k_iL_Bp = load(fullfile(rootdir, "Run_1_10000_Hz_ch1.MAT"), myVar{:});
f_10k_iR_Bp = load(fullfile(rootdir, "Run_1_10000_Hz_ch3.MAT"), myVar{:});
f_10k_vL_Bp = load(fullfile(rootdir, "Run_1_10000_Hz_math1.MAT"), myVar{:});
f_10k_vR_Bp = load(fullfile(rootdir, "Run_1_10000_Hz_math2.MAT"), myVar{:});

%Differenz Spannungen + 10 V weil Offset im Oszilloskop eingestellt
uDiff_01_L_Rl = f_01_iL_Rl.data - (f_01_vL_Rl.data + 10);
uDiff_01_R_Rl = f_01_iR_Rl.data - (f_01_vR_Rl.data + 10);

uDiff_10k_L_Rl = f_10k_iL_Rl.data - (f_10k_vL_Rl.data + 10);
uDiff_10k_R_Rl = f_10k_iR_Rl.data - (f_10k_vR_Rl.data + 10);

uDiff_01_L_Bp = f_01_iL_Bp.data - (f_01_vL_Bp.data + 10);
uDiff_01_R_Bp = f_01_iR_Bp.data - (f_01_vR_Bp.data + 10);

uDiff_10k_L_Bp = f_10k_iL_Bp.data - (f_10k_vL_Bp.data + 10);
uDiff_10k_R_Bp = f_10k_iR_Bp.data - (f_10k_vR_Bp.data + 10);


%Strom mit I = U/R
i_01_L_Rl = uDiff_01_L_Rl / 2.2;
i_01_R_Rl = uDiff_01_R_Rl / 2.2;

i_10k_L_Rl = uDiff_10k_L_Rl / 2.2;
i_10k_R_Rl = uDiff_10k_R_Rl / 2.2;

i_01_L_Bp = uDiff_01_L_Bp / 2.2;
i_01_R_Bp = uDiff_01_R_Bp / 2.2;

i_10k_L_Bp = uDiff_10k_L_Bp / 2.2;
i_10k_R_Bp = uDiff_10k_R_Bp / 2.2;

%Ladungsfluss Q = I * t für jenden Messpunkt aufsummiert
Q_01_L_Rl = sum(i_01_L_Rl * f_01_vL_Rl.sampleInterval);
Q_01_R_Rl = sum(i_01_R_Rl * f_01_vR_Rl.sampleInterval);

Q_10k_L_Rl = sum(i_01_L_Rl * f_10k_vL_Rl.sampleInterval);
Q_10k_R_Rl = sum(i_01_R_Rl * f_10k_vR_Rl.sampleInterval);

Q_01_L_Bp = sum(i_01_L_Bp * f_01_vL_Bp.sampleInterval);
Q_01_R_Bp = sum(i_01_R_Bp * f_01_vR_Bp.sampleInterval);

Q_10k_L_Bp = sum(i_01_L_Bp * f_10k_vL_Bp.sampleInterval);
Q_10k_R_Bp = sum(i_01_R_Bp * f_10k_vR_Bp.sampleInterval);


%"F=0.1 Hz Links Rl";
ausMat{1,1} = Q_01_L_Rl;

%"F=0.1 Hz Rechts Rl";
ausMat{2,1} = Q_01_R_Rl;

%"F=10k Hz Links Rl";
ausMat{3,1} = Q_10k_L_Rl;

%"F=10k Hz Rechts Rl";
ausMat{4,1} = Q_10k_R_Rl;

%"F=0.1 Hz Links Bp";
ausMat{5,1} = Q_01_L_Bp;

%"F=0.1 Hz Rechts Bp";
ausMat{6,1} = Q_01_R_Bp;

%"F=10k Hz Links Bp";
```

```
ausMat{7,1} = Q_10k_L_Bp;

%"F=10k Hz Rechts Bp";
ausMat{8,1} = Q_10k_R_Bp;

latex(sym(ausMat))
```

A.3.4 Skript für das Messen des Tastgrads

```
%Wirungsgrad berechnen
clear all;
format shortG;
myVar = {"time", "data", "recordLength"};

outFolder = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_5_Eff';

filesRL = dir("C:\Users\niklas\Documents\Bachelor\Messung\6_DCAC\3_RL_DC\");
filesRL = filesRL(~[filesRL.isdir]);

filesBP = dir("C:\Users\niklas\Documents\Bachelor\Messung\6_DCAC\1_BP_DC\");
filesBP = filesBP(~[filesBP.isdir]);



f1 = 1;
f2 = 1;
f3 = 1;
f4 = 1;

for i = 1:length(filesRL)
    %Linker Messwiderstand
    if contains(filesRL(i).name,"ch1")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        tastTableRL{f1,1} = filesRL(i).name;
        tastTableRL{f1,2} = data;
        f1 = f1 + 1;
    %Linke Batterie
    elseif contains(filesRL(i).name,"math1")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        tastTableRL{f2,3} = filesRL(i).name;
        %Offset Oszilloskop ausgleichen
        tastTableRL{f2,4} = data + 10;
        f2 = f2 + 1;
    %Rechter Messwiderstand
    elseif contains(filesRL(i).name,"ch3")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        tastTableRL{f3,5} = filesRL(i).name;
        tastTableRL{f3,6} = data;
        f3 = f3 + 1;
    %Rechte Batterie
    elseif contains(filesRL(i).name,"math2")
        load(fullfile(filesRL(i).folder,filesRL(i).name));
        tastTableRL{f4,7} = filesRL(i).name;
        %Offset Oszilloskop ausgleichen
        tastTableRL{f4,8} = data + 10;
        f4 = f4 + 1;
    end
end

f1 = 1;
f2 = 1;
f3 = 1;
f4 = 1;

for i = 1:length(filesBP)
    %Linker Messwiderstand
```

```
    if contains(filesBP(i).name,"ch1")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        tastTableBP{f1,1} = filesBP(i).name;
        tastTableBP{f1,2} = data;
        f1 = f1 + 1;
    %Linke Batterie
    elseif contains(filesBP(i).name,"math1")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        tastTableBP{f2,3} = filesBP(i).name;
        %Offset Oszilloskop ausgleichen
        tastTableBP{f2,4} = data + 10;

        f2 = f2 + 1;
    %Rechter Messwiderstand
    elseif contains(filesBP(i).name,"ch3")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        tastTableBP{f3,5} = filesBP(i).name;
        tastTableBP{f3,6} = data;
        f3 = f3 + 1;
    %Rechte Batterie
    elseif contains(filesBP(i).name,"math2")
        load(fullfile(filesBP(i).folder,filesBP(i).name));
        tastTableBP{f4,7} = filesBP(i).name;
        %Offset Oszilloskop ausgleichen
        tastTableBP{f4,8} = data + 10;
        f4 = f4 + 1;
    end
end

%Frequenz extrahieren
for i = 1:length(tastTableRL(:,1))
    tastTableRL{i,1} = str2double(extract(tastTableRL{i,1},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableRL{i,3} = str2double(extract(tastTableRL{i,3},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableRL{i,5} = str2double(extract(tastTableRL{i,5},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableRL{i,7} = str2double(extract(tastTableRL{i,7},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
end

for i = 1:length(tastTableBP(:,1))
    tastTableBP{i,1} = str2double(extract(tastTableBP{i,1},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableBP{i,3} = str2double(extract(tastTableBP{i,3},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableBP{i,5} = str2double(extract(tastTableBP{i,5},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
    tastTableBP{i,7} = str2double(extract(tastTableBP{i,7},regexpPattern("(?<=_(1|2)_)[0-9]\.?(1|4|8|0)?[0]{0,4}")));
end

%Sortieren nach Frequenz
tastTableBP = sortrows(tastTableBP,1);
tastTableRL = sortrows(tastTableRL,1);

for i = 1:14

    %Spannung über linkem Messwiderstand
    uDiffL = tastTableRL{i,2} - tastTableRL{i,4};
    %Strom aus Batterie links gemittelt
    iBatL = mean(uDiffL / 2.2);
    %Leistung linke Batterie P = U * I gemittelt
    P_L = mean(tastTableRL{i,4}) * iBatL;

    %Spannung über rechtem Messwiderstand
    uDiffR = tastTableRL{i,6} - tastTableRL{i,8};
    %Strom aus Batterie rechts gemittelt
    iBatR = mean(uDiffR / 2.2);
    %Leistung rechte Batterie P = U * I gemittelt
    P_R = mean(tastTableRL{i,8}) * iBatR;

    %Wirkungsgrad
    wirk(i) =  abs(P_L /  P_R);
end
```

```
for i = 1:14
    %Spannung über linkem Messwiderstand
    uDiffL = tastTableBP{i,2} - tastTableBP{i,4};
    %Strom aus Batterie links
    iBatL = uDiffL / 2.2;

    %Spannung über rechtem Messwiderstand
    uDiffR = tastTableBP{i,6} - tastTableBP{i,8};
    %Strom aus Batterie rechts
    iBatR = uDiffR / 2.2;

    %Leistung linke Batterie P = U * I
    P_L = abs(iBatL .* tastTableBP{i,4});
    %Leistung rechte Batterie P = U * I
    P_R = abs(iBatR .* tastTableBP{i,8});

    %Spitzen finden
    pksL = findpeaks(P_L(20000:40000),"MinPeakProminence",0.4);
    pksR = findpeaks(P_R(20000:40000),"MinPeakProminence",0.4);

    %min(pksL) == PnutL max(pksR) == PzuL
    wirkLnachR(i) = min(pksL)/ max(pksR);
    %min(pksR) == PnutR max(pksL) == PzuR
    wirkRnachL(i) = min(pksR)/max(pksL);
end

for i = 1:14
    ausMatRl(i,1) = tastTableRL{i,1};
    %Wirkungsgrad in Prozent
    ausMatRl(i,2) = wirk(i) * 100;
    % Wirkungrad L nach R in Prozent
    ausMatBp(i,1) = wirkLnachR(i) * 100;
    % Wirkungrad R nach L in Prozent
    ausMatBp(i,2) = wirkRnachL(i) * 100;
end

latex(sym(ausMatRl))
latex(sym(ausMatBp))
```

A.3.5 Skript für das Testen der 1. Oberschwingung für die EIS

```
%1. Oberschwingung zur nutzung als daten für EIS prüfen
close all;
clear

addpath("..\..\Mat2Tikz\");

function speichern(outFolder, Fig)
%Leeren des output Ordners
figure(Fig);
matlab2tikz(append(outFolder, '\', Fig.Name, '.tex'));
end


lenFiles = length(dir('C:\Users\niklas\Documents\Bachelor\Messung\8_Test50Per\Data\*.mat'));
rootdir = 'C:\Users\niklas\Documents\Bachelor\Messung\8_Test50Per\Data\';
outFolder1 = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_8_Bsp';
outFolder2 = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_8_Bode';
outFolder3 = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_8_Ober';
outFolder4 = 'C:\Users\niklas\Documents\Bachelor\Latex\template-latex_std\templa-
    te-latex_thesis\template_thesis\chapters\5_bilder\5_8_OberBode';

messFreq = [0.1, 0.4, 0.8, 1, 4, 8, 10, 40, 80, 100, 400, 800, 1000, 4000, 8000, 10000];
```

```
freqLen = length(messFreq);
runs = lenFiles / freqLen / 4;

T(2 * runs * length(messFreq)) = zeros;
Fs(2* runs * length(messFreq)) = zeros;
L(2 * runs * length(messFreq)) = zeros;

rawUData=cell([1 runs * length(messFreq)]);
rawIData=cell([1 runs * length(messFreq)]);

pos = 1;

%Spannungen einlesen
for run = 1:runs
    for f = 1:length(messFreq)

        load(fullfile(rootdir,sprintf('Run__%s__%s__Hz__math1.mat',num2str(run),num2str(messFreq(f)))));
        %Offset Oszilloskop entfernen und Gleichspannung entfernen
        rawUData{pos} = (data + 10) - mean(data);


        T(pos) = sampleInterval;
        Fs(pos) = 1/T(pos);
        L(pos) = recordLength;



        pos = pos + 1;
    end
end

pos = 1;

%Ströme einlesen
for run = 1:runs
    for f = 1:length(messFreq)
        load(fullfile(rootdir,sprintf('Run__%s__%s__Hz__ch1.mat',num2str(run),num2str(messFreq(f)))));

        T(pos) = sampleInterval;
        Fs(pos) = 1/T(pos);
        L(pos) = recordLength;

        %Strom berechnen und Gleichanteil entfernen
        rawIData{pos} = ((data-mean(data)) - rawUData{pos}) / 2.2;

        pos = pos + 1;
    end
end

rawUDataMat = cell2mat(rawUData);
rawIDataMat = cell2mat(rawIData);

%Inf und -Inf Werte entfernen
rawUDataMat(rawUDataMat==-Inf) = min(rawUDataMat(isfinite(rawUDataMat)));
rawIDataMat(rawIDataMat==-Inf) = min(rawIDataMat(isfinite(rawIDataMat)));

rawUDataMat(rawUDataMat==Inf) = max(rawUDataMat(isfinite(rawUDataMat)));
rawIDataMat(rawIDataMat==Inf) = max(rawIDataMat(isfinite(rawIDataMat)));


%Cell Arrays in Matrizen umwandeln
fftUData = fft(rawUDataMat,L(1),1);
fftIData = fft(rawIDataMat,L(1),1);


%Einseitigesspektrum Spannung
P2U = fftUData/L(1);
```

```
P1U = P2U(1:L(1)/2+1,:);
P1U(2:end-1,:) = 2*P1U(2:end-1,:);

%Einseitigesspektrum Strom
P2I = fftIData/L(1);
P1I = P2I(1:L(1)/2+1,:);
P1I(2:end-1,:) = 2*P1I(2:end-1,:);

%Finde spektrale Peaks
for i = 1:32
    %Find peaks in FFT von Spannung
    [pksU lksU] = findpeaks(abs(P1U(1:end,i)));

    peakTableU = [pksU lksU];
    %Sortiere Peaks nach Amplitude
    peakTableU = sortrows(peakTableU,1,"descend");
    %Entferne Peaks die vor der größten Amplitude (Messfrequenz) liegen
    peakTableU = peakTableU(peakTableU(:,2) >= peakTableU(1,2),:);

    %Frequenzvektor erstellen
    freqTest = 0:(Fs(i)/L(i)):(Fs(i)/2-Fs(i)/L(i));

    %Frequenz der Peaks bestimmen
    peakTableU(:,3) = freqTest(peakTableU(:,2));

    %Messfrequenz werte u und i speichern
    uVal(i) = P1U(peakTableU(1,2),i);
    iVal(i) = P1I(peakTableU(1,2),i);

    %Vier höchsten 1. Oberschwingung speichern
    oberU{i} = peakTableU(1:4,:);

    %Für die 1. Oberschwingung die Spannung und Stromwerte speichern
    oberUcmp{i} = P1U(peakTableU(1:4,2),i);
    oberIcmp{i} = P1I(peakTableU(1:4,2),i);
end


Ufreq(runs * length(messFreq)) = zeros;
Ifreq(runs * length(messFreq)) = zeros;

%Impedanz berechnen
for i = 1:32
    Zfreq(i) = uVal(i) ./ iVal(i);
end

realMilli = 1000 * real(Zfreq);
imagMilli = 1000 * imag(Zfreq);


figOberRL = figure("Name","figOberRL");
%Color Table
CT = parula(16);
CT = CT(4:4:end - 4, :);
colororder(CT);
%Rl
for i = 1:16
    %Impedanz der 1. Oberschwingung berechnen
    Zober = oberUcmp{i}(2) ./ oberIcmp{i}(2);
    %Plot der 1. Oberschwingung
    plot(real(Zober)*1000,imag(Zober)*1000,'LineStyle','none','Marker','o','Color',CT(1,:));
    %Nyquist Diagramm
    set(gca, 'YDir','reverse')
    %Frequenz der Oberschwingung als Text neben dem Punkt
    text(real(Zober)*1000,imag(Zober)*1000,sprintfc(' %0.1f Hz',oberU{i}(2,3)),"FontSize",7);
    hold on
end
run = 0;
%Messfrequenz als Kreuz einzeichnen
```

```
plot(realMilli((1+(run*freqLen)):(freqLen+(run*freq-
    Len))),imagMilli((1+(run*freqLen)):(freqLen+(run*freqLen))),'DisplayName',sprintf('Run:
    %d',run),'Marker','x');
yline(0);
xlabel('Re Z [mOhm]');
ylabel('Im Z [mOhm]');
xlim([50 75])

figOberBP = figure("Name","figOberBP");
%Color Table
CT = parula(16);
CT = CT(4:4:end - 4, :);
colororder(CT);
%BP
for i = 17:32
    %Impedanz der 1. 1. Oberschwingung berechnen
    Zober = oberUcmp{i}(2) ./ oberIcmp{i}(2);
    %Plot der 1. Oberschwingung
    plot(real(Zober)*1000,imag(Zober)*1000,'LineStyle','none','Marker','o','Color',CT(1,:));
    %Nyquist Diagramm
    set(gca, 'YDir','reverse')
    %Frequenz der Oberschwingung als Text neben dem Punkt
    text(real(Zober)*1000,imag(Zober)*1000,sprintfc(' %0.1f Hz',oberU{i}(2,3)),"FontSize",7);
    hold on
end
run = 1;
%Messfrequenz als Kreuz einzeichnen
plot(realMilli((1+(run*freqLen)):(freqLen+(run*freq-
    Len))),imagMilli((1+(run*freqLen)):(freqLen+(run*freqLen))),'DisplayName',sprintf('Run:
    %d',run),'Marker','x');
yline(0);
xlabel('Re Z [mOhm]');
ylabel('Im Z [mOhm]');
xlim([50 75]);


%Bode Diagramm der 1. Oberschwingung
for i = 1:32
    %Impedanz der 1. Oberschwingung berechnen
    Zober = oberUcmp{i}(2) ./ oberIcmp{i}(2);
    magOber(i) = 1000 * abs(Zober);
    phaseOber(i) = rad2deg(angle(Zober));
    freqOber(i) = oberU{i}(2,3);
end
mag = abs(Zfreq);
phase = rad2deg(angle(Zfreq));

figBodeOberRL = figure('Name','figBodeOberRL');
%Color Table
CT = parula(16);
CT = CT(4:4:end - 4, :);
colororder(CT);

subplot(2,1,1);
semilogx(messFreq(1:16),mag(1:16)*1000,freqOber(1:16),magOber(1:16));
xlabel("Frequenz in Hz");
ylabel("Magnitude Z [mOhm]")
legend("Messfrequenz","Erste Oberschwingung","Location","northoutside");

subplot(2,1,2);
semilogx(messFreq(1:16),phase(1:16),freqOber(1:16),phaseOber(1:16));
xlabel("Frequenz in Hz");
ylabel("Phase [deg]")

figBodeOberBP = figure('Name','figBodeOberBP');
%Color Table
CT = parula(16);
CT = CT(4:4:end - 4, :);
colororder(CT);
```

```
subplot(2,1,1);
semilogx(messFreq(1:16),mag(17:32)*1000,freqOber(17:32),magOber(17:32));
xlabel("Frequenz in Hz");
ylabel("Magnitude Z [mOhm]")
legend("Messfrequenz","Erste Oberschwingung","Location","northoutside");

subplot(2,1,2);
semilogx(messFreq(1:16),phase(17:32),freqOber(17:32),phaseOber(17:32));
xlabel("Frequenz in Hz");
ylabel("Phase [deg]")

speichern(outFolder4,figBodeOberRL);
speichern(outFolder4,figBodeOberBP);
speichern(outFolder3,figOberRL);
speichern(outFolder3,figOberBP);
speichern(outFolder3,figOberSpek);
```

Glossar

ARM Cortex-M Familie von 32-bit Prozessoren, primär für Mikrocontroller, entwickelt von der Firma ARM Holdings Limited

Output-Modification-Register Register der XMC Mikrocontroller, das die Manipulation der Pins eines Ports über eine Flip-Flop ähnliche Logik ermöglicht

VISA-API Virtual Instrument Software Architecture die die Steuerung von Ethernet/LXI-, GPIB-, seriellen, USB-, PXI- und VXI-Geräten ermöglicht

Visual Studio Code Vielseitig erweitbarer Quelltext-Editor von Microsoft

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original