HAW HAMBURG

Masterarbeit

Tobias Frahm

Sensorsystem für die Impedanzspektroskopie in Fahrzeugbatterien: Analogvorstufe, Signalverarbeitung und Software

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Tobias Frahm

Sensorsystem für die Impedanzspektroskopie in Fahrzeugbatterien: Analogvorstufe, Signalverarbeitung und Software

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informations- und Kommunikationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 19. September 2023

Tobias Frahm

Thema der Arbeit

Sensorsystem für die Impedanzspektroskopie in Fahrzeugbatterien: Analogvorstufe, Signalverarbeitung und Software

Stichworte

Elektrofahrzeuge, E-Mobilität, Automotiv, niederohmige Batteriezellen, Lithium-Ionen-Zelle, Lithium-Ionen-Batterie, Sensorik, Batteriesensor, EIS, Elektrochemische Impedanzspektroskopie, BMS, Batterie Management System, Einzelzellsensor

Kurzzusammenfassung

Die zugrundeliegende Arbeit befasst sich mit der Entwicklung eines Batteriesensors für die elektrochemische Impedanzspektroskopie. Die Herausforderung hierbei ist die Niederohmigkeit der aktuell in Elektrofahrzeugen verbauten Batteriezellen bei verhältnismäßig geringen Frequenzen. Die Arbeit behandelt die Entwicklung von Einzelzellsensoren, dem gegenüber steht der Mehrfachzellsensor.

Tobias Frahm

Title of Thesis

Sensor system for impedance spectroscopy in vehicle batteries: Analog preamplifier, signal processing and software

Keywords

electric car, electric vehicle, ev, low resistance battery, low impedance battery, lithium-ion battery, EIS, electrochemical impedance spectroscopy , BMS, single-cell sensors

Abstract

The underlying thesis deals with the development of a battery sensor for electrochemical impedance spectroscopy. The challenge is the low impedance of the battery cells currently used in electric vehicles at relatively low frequencies. The thesis deals with the development of single-cell sensors, which is contrasted with the multiple-cell sensor.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In dem folgenden Kapitel wird die Motivation zu dieser Arbeit und die daraus resultierende Aufgabenstellung beschrieben.

1.1 Motivation

Elektromobilität ist in der aktuellen Zeit eines der meistdiskutierten Themen in der Automobilindustrie und wesentlich für die Standorterhaltung in Deutschland. Hierbei werden viele Aspekte berücksichtigt, neben dem Ausbau der Ladeinfrastruktur ist die Reichweite von Elektrofahrzeugen und damit eine gewisse Zuverlässigkeit für den Kunden einer der relevanten Aspekte für den Endverbraucher [7]. Möchte man, ohne den Bauraum wesentlich zu verändern, die Reichweite verbessern und eine schnellere Amortisierung des sog. Treibhaus-Rucksacks aus der Herstellung ermöglichen, ist eine hohe Ausnutzung der schon vorhandenen Kapazität notwendig [47]. Ein naheliegender Ansatz, um dies zu erreichen, ist die Verwendung von neuen und effizienteren Zellchemien. Doch auch wenn neue Zellchemie vielversprechende Kapazitätssteigerungen aufweist, ist das verbesserte Ausnutzen der vorhandenen Kapazitäten unabhängig von der Zellchemie eine sinnvolle Methode, um die Effizienz von Elektrofahrzeugen zu steigern.

Für diese Aufgabe ist das Batterie-Management-System (BMS) die zentrale Komponente der Batterie im Elektrofahrzeug. Ein BMS übernimmt die Zustandsüberwachung der Batterie im Elektrofahrzeug. Unter anderem schützt das BMS ein Batteriesystem vor Überspannung und verhindert Tiefenentladung. Es dient außerdem als Datenschnittstelle, durch welche Zustandsinformationen für den Endverbraucher oder Serviceinformationen zur Wartung ausgetauscht werden können.

Um ein möglichst effektives Batterie-Management zu gewährleisten, ist es notwendig, genaue Kenntnis über den Zustand der Batterie zu erhalten. Umso detaillierter die Batterie betrieben werden soll, desto notwendiger wird es, spezifische Sensorik gezielt an den

Batteriesystemen einzusetzen. Hierbei kann der Einsatz von Sensorik auf unterschiedlichen Ebenen des Batteriesystems erfolgen. Je nach Anwendung kann es ausreichen, mit wenigen einzelnen Sensoren den Zustand des Gesamtsystems zu überwachen. Für eine detaillierte Zustandsüberwachung und damit optimierter Ausnutzung der vorhandenen Kapazität eines größeren Batteriesystems ist der Zustand einzelner Zellen jedoch von besonderem Interesse. Die gezielte Identifizierung beschädigter oder vermindert leistungsfähiger Einzelzellen ist ohne eine Einzelzellüberwachung oder die Überwachung von kleineren Zellverbänden schwer umsetzbar.

Weiterhin ist ein nachhaltiger Umgang mit den verfügbaren Ressourcen in vielerlei Hinsicht sinnvoll. Durch einen verlängerten Betrieb aufgrund von besserer Zustandskenntnis der Zellen und ein gezieltes Austauschen der Einzelzellen im Fehlerfall können nicht nur Ressourcen, sondern auch Kosten und Aufwände (z. B. logistische) eingespart werden. Ein Nachteil einer Einzelzellüberwachung ist der erhöhte Aufwand der Sensorik. Diesem kann mit integrierbaren Sensorsystemen entgegengewirkt werden. Integrierbare Sensorik zeichnet sich durch ein hohes Maß an Miniaturisierung aus. Einige Hersteller ermöglichen mittlerweile auch die Programmierung von analogen Schaltungen auf einem integrierten Mikrocontroller [3]. Batteriesysteme werden gerade in der Elektromobilität immer komplexer, sie sind nach wie vor der Flaschenhals beim Betrieb der Elektrofahrzeuge und werden weltweit kontinuierlich weiterentwickelt und erforscht. Möchte man tiefere Kenntnisse über den Zustand des Batteriesystems nutzen, muss entsprechende Sensorik die Überwachung der Parameter ermöglichen. Um die Zustandsüberwachung des Batteriesystems Zell-diskret zu realisieren, wird in dieser Arbeit ein Sensorsystem zur Überwachung von Batteriezellen mithilfe der elektrochemischen Impedanzspektroskopie entwickelt. Ziel ist es, das Konzept in einem Laboraufbau in Betrieb zu nehmen, zu erproben und für weitere Untersuchungen bereitzustellen.

1.2 Aufgabenstellung

Ziel der Arbeit ist die Entwicklung eines Messsystems zur Durchführung der elektrochemischen Impedanzspektroskopie. Es soll ein Laboraufbau in Hard- und Software entwickelt werden. Dieser muss in der Lage sein, die Elektrochemische Impedanzspektroskopie (EIS) durchzuführen. Für das System sollen Platinen jeweils für die Strom- und Spannungsmessung entwickelt werden. Die steuernde Controllerinstanz soll durch das Programm MATLAB auf einem PC realisiert werden. Die Umsetzung des Controllers innerhalb von

dem Programm MATLAB ermöglicht eine bessere Bewertung der Messungen des Systems im Laboraufbau. Das Messsystem soll dabei in der Lage sein, den durch die zu messende Batterie eingebrachten Gleichanteil zu kompensieren und das Signal für eine Messung ausreichend zu verstärken. Die Stromanregung soll dezentral durch einen externen Generator umgesetzt werden. Die Strommessung erfolgt über einen Shunt-Widerstand. Die Soft- und die Hardware müssen die notwendige Synchronisationsanforderung an die Messung umsetzen. Die Messwerte der einzelnen Platinen sollen in der Controllerinstanz mithilfe von MATLAB verarbeitet werden. Dafür muss zudem eine Kommunikation zwischen dem PC und dem Sensorsystem aufgebaut werden. Da die Berechnung perspektivisch auf den Mikrocontrollern erfolgen soll, ist für die Berechnung des Spektrums das Goertzel-Filter einzusetzen.

2 Grundlagen

In dem folgenden Kapitel wird auf die Grundlagen der Arbeit eingegangen. Hierbei sind neben der EIS und den Eigenschaften von Batterien verwendete Geräte, mathematische Mittel und in der Arbeit verwendete Grundlagen beschrieben.

2.1 Fouriertransformation

Die Fouriertransformation ist ein mathematisches Mittel, um den harmonischen Frequenzanteil eines aperiodischen Signals zu einem Gesamtsignal $f(t)$ zu berechnen. Im Allgemeinen wird das erste Fourier-Integral zum Bestimmen des Spektrums des Signals $f(t)$ wie folgt definiert:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} dt \tag{2.1}$$

Um eine Analyse für digitale Signale durchführen zu können, muss die diskrete Variante der Fouriertransformation verwendet werden, diese ergibt sich wie folgt:

$$X(k) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}} \tag{2.2}$$

Wie auch die Fouriertransformation im kontinuierlichen Zeitbereich gibt die Diskrete Fourier Transformation (DFT) die Gewichtung der Frequenz k für das Signal $x[n]$ an und ermöglicht so eine Analyse der einzelnen Frequenzanteile.

Goertzel-Filter

Das Goertzel-Filter [19] ist eine effektive Methode zur Berechnung einzelner Spektralkomponenten. Im folgenden Abschnitt wird die Herleitung nachvollzogen. Das Filter ist eine Implementierung der Berechnung der DFT für genau eine Spektralkomponente. Der Ansatz besteht darin, die DFT für ein k in eine Übertragungsfunktion eines Filters zu überführen. Allgemein wird der Ausgang $y[n]$ eines Übertragungssystems als Faltungssumme wie folgt definiert:

$$y[n] = x[n] * h[n] = \sum_{m=0}^{n} x[m] \cdot h[n-m] \tag{2.3}$$

Für eine vereinfachte Darstellung wird der komplexe Faktor der DFT substituiert:

$$e^{-j\frac{2\pi kn}{N}} = W_N^{k \cdot n} \tag{2.4}$$

Aufgrund der Eigenschaft, dass die Gleichung 2.4 für $n = N$ immer zu eins ergibt, kann die DFT mit dem Term erweitert werden, ohne das Ergebnis zu ändern:

$$X(k) = \sum_{m=0}^{N-1} x[n] \cdot W_N^{k \cdot m} \cdot W_N^{-k \cdot N} = \sum_{m=0}^{N-1} x[n] \cdot W_N^{-k \cdot (N-m)} \tag{2.5}$$

Vergleicht man die Faltungssumme 2.3 mit der erweiterten DFT 2.5, lässt sich für $h[n]$ folgende Gleichung aufstellen:

$$h[n] = (W_N^{-k})^n \cdot u[n] \tag{2.6}$$

für die Kausalität mit

$$u[n] = \begin{cases} 0 & n < 0 \\ 1 & \text{sonst} \end{cases} \tag{2.7}$$

Für $h[n-m]$ ergibt sich aus der Faltungssumme 2.3 und 2.6, wenn $n = N$ folgende Gleichung:

$$y[N] = x[n] * h[n] = \sum_{m=0}^{N} x[m] \cdot W_N^{-k(m-N)} = \sum_{m=0}^{N} x[m] \cdot W_N^{k \cdot m} \cdot W_N^{-k \cdot N} = X[k] \quad (2.8)$$

Damit ergibt sich, dass die Berechnung der DFT an der $k - ten$ Frequenz dem Ausgang $y[n]$ des Filters entspricht, wenn $n = N$ ist [37][58]. Aus der Faltungssumme lässt sich für $n = 0 : N$ eine lineare DGL 1. Ordnung ableiten 2.9.

$$y[n] = x[n] + h[1] \cdot y[n-1] = b_0 \cdot x[n] - a_1 \cdot y[n-1] \quad (2.9)$$

Mit den Koeffizienten

$$a_1 = -W_N^{-k} \text{ und } b_0 = 1$$

ergibt sich die Übertragungsfunktion des Goertzel-Filters 1. Ordnung zu Gleichung 2.10, Abbildung 2.2.

$$H_1(z) = \frac{1}{1 + (-W_N^{-k}) \cdot z^{-1}} \quad (2.10)$$

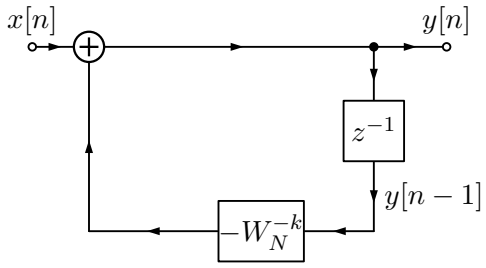


Abbildung 2.1: Die Abbildung zeigt das Signalflussdiagramm des Goertzel-Filters 1.Ordnung.

Das Goertzel-Filter 1.Ordnung erfordert pro Takt eine komplexe Multiplikation, dies entspricht vier reellen Multiplikationen und zwei Additionen. Das Filter kann effizienter gestaltet werden, indem die komplexen Multiplikationen aus dem Rekursionszweig vermieden werden [58]. Dies kann durch konjugiert-komplexes Erweitern der Übertragungsfunktion erreicht werden.

$$H_2(z) = \frac{1 + (-W_N^{-k})^* \cdot z^{-1}}{[1 + (-W_N^{-k}) \cdot z^{-1}] \cdot [1 + (-W_N^{-k})^* \cdot z^{-1}]} \tag{2.11}$$

$$H_2(z) = \frac{1 - e^{-j\frac{2\cdot\pi\cdot k}{N}} \cdot z^{-1}}{1 - 2 \cdot cos(\frac{2\cdot\pi\cdot k}{N}) \cdot z^{-1} + z^{-2}} \tag{2.12}$$

Es ergeben sich folgende Koeffizienten:

$$a_0 = 1,\ a_1 = -2 \cdot cos(\tfrac{2\cdot\pi\cdot k}{N}) \text{ und } a_2 = 1$$

$$b_0 = 1,\ b_1 = -e^{-j\frac{2\cdot\pi\cdot k}{N}} \text{ und } b_2 = 0$$

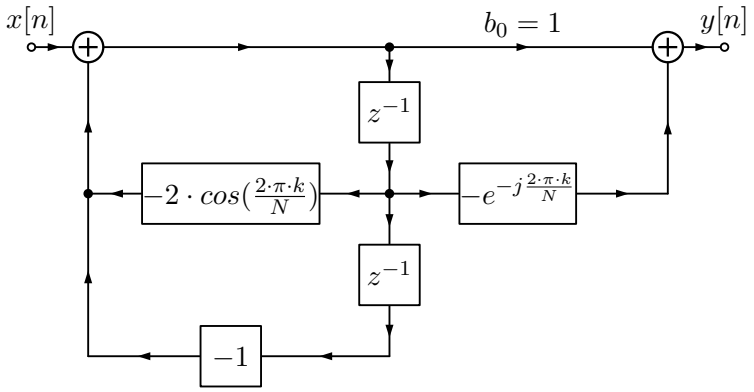


Abbildung 2.2: Die Abbildung zeigt das Signalflussdiagramm des Goertzel-Filters 2.Ordnung. Im rekursiven Zweig befinden sich keine komplexen Operationen mehr.

Durch das Erweitern der Übertragungsfunktion 2.10 zu 2.12 verbessert sich die Effizienz der Berechnung, da im rekursiven Zweig der Übertragungsfunktion keine komplexen Rechnungen mehr durchgeführt werden müssen. Die komplexe Rechnung der Koeffizienten im Nenner muss nur einmal durchgeführt werden. Somit werden insgesamt N reelle Multiplikation, $2 \cdot N$ Additionen und eine komplexe Multiplikation und eine reelle Addition benötigt.

2.2 Batteriesysteme

Ein Batteriesystem ist ein elektrochemisches Energiespeichersystem, welches in der aktuellen Zeit gerade im Bezug auf die Mobilität eine hohe Bedeutung hat. Batteriesysteme werden blockweise oder modular aufgebaut. Der modulare Aufbau ist bei größeren Batteriesystemen zu bevorzugen, da die Wartbarkeit und das Austauschen einzelner Batteriemodule vereinfacht wird. Der Aufbau in größeren Einzelzellen kommt hingegen bei kleineren Gesamtsystemen zum Einsatz [30]. Einzelne Batteriemodule können in Serie oder parallel geschaltet werden. Bei in Serie geschaltete Batterien ergibt sich die Spannungslage des Systems durch die Summe der Zellspannungen. Nachteilig an in Serie geschalteten Lithium-Ionen-Batterien ist der hohe systemische Aufwand, um sicherzustellen, dass die einzelnen Zellen vor Tiefenentladung oder Überspannung geschützt werden. Bei einer Parallelschaltung der Zellen bleibt das Spannungslevel der Zelle erhalten, es erhöht sich aber die Kapazität des Systems. Der Aufwand zur Zustandsüberwachung der Zellen ist geringer als bei der Serienverschaltung. In der Praxis werden oftmals parallele Zellenverbände in Serie geschaltet, um die benötigte Kapazität mit dem entsprechenden Spannungsniveau zu erreichen [30].

2.2.1 Lithium-Ionen-Batterien

Lithium-Ionen-Batterien werden unter anderem aufgrund Ihrer Energiedichte heute in vielen Bereichen verwendet. Sie ermöglichen eine vergleichsweise kompakte Bauform bei hoher Energiedichte. Die Energiedichte wird in die gravimetrische und die volumetrische Energiedichte unterteilt. Die gravimetrische Energiedichte ist das Maß für die Energie pro Masse [$^{Wh}/_{kg}$], die volumetrische Energiedichte gibt hingegen die Energie pro Volumen an [$^{Wh}/_{m^3}$]. Die Lithium-Ionen-Technologie ist beim aktuellen Stand der Technik führend [13].

Funktionsweise

Eine Lithium-Ionen-Zelle besteht, wie in Abbildung 2.3 zu sehen, aus einem Gehäuse, in welchem sich ein Elektrolyt befindet. Der Elektrolyt ist für den Ladungstransport zuständig. Weiterhin befinden sich in dem Elektrolyt zwei Elektroden. In der positiven Elektrode (Kathode), bestehend aus einem Metalloxid, sind Lithium-Atome eingelagert. Zwischen beiden Elektroden befindet sich ein Separator, der die Elektroden voneinander

trennt und einen Kurzschluss an dieser Stelle verhindert. Der Separator lässt ausschließlich die Lithium-Ionen durch. Bei dem Ladevorgang fließen Elektronen von der Kathode zur aus Graphit bestehenden Elektrode (Anode). Den Lithium-Atomen wird dabei ein Elektron entzogen, es bleibt ein Lithium-Ion über. Das Lithium-Ion wandert durch den Separator zur Anode und nimmt wieder ein Elektron auf, um einen ladungsneutralen Zustand herzustellen. Beim Entladen verläuft dieser Prozess umgekehrt, Elektronen fließen von der Anode zur Kathode, aus den ladungsneutralen Lithium-Atomen werden Lithium-Ionen, welche zurück zur Kathode wandern. Sind keine Lithium-Ionen in der Anode mehr vorhanden, ist die Batterie entladen [13][14].

Alterung

Die Alterungseffekte von Batteriezellen sind vielseitig. Im folgenden Abschnitt wird ein Ausschnitt der möglichen Gründe für eine Zellalterung nachvollzogen. Die Alterungsprozesse sind komplex und treten oftmals gleichzeitig im Betrieb auf.

- Initial wird unter bestimmten Bedingungen in der Lithium-Ionen-Batterie der Elektrolyt thermodynamisch instabil. Dadurch kommt es während des Ladevorgangs an der Anodenseite zu einer Zersetzung von Teilen des Elektrolyten. Diese Zersetzungsprodukte werden auf der Anodenoberfläche in Form eines wenige Nanometer dünnen und kompakten Feststofffilms abgeschieden, hierbei handelt es sich um die Solid Electrolyte Interface (SEI)-Schicht [59]. Diese bildet sich beim ersten Ladevorgang zwischen Anode und Elektrolyt. Das Aufbauen der SEI-Schicht verbraucht eine geringe Menge Lithium, welches anschließend nicht mehr für die Zirkulation verwendet werden kann und einen dauerhaften Kapazitätsverlust darstellt [30]. Die SEI-Schicht wächst über die Lebensdauer der Zelle und hat einen wesentlichen Einfluss auf die Performance der Zelle [10]. Als Einflussfaktoren für das Wachstum der SEI-Schicht werden Temperatur, Ladestrom, Zellspannung und die Elektrolytzusammensetzung aufgeführt [49].
- An der Kathode kommt es auch zu einer Elektrolytzersetzung und Bildung einer Reaktionsschicht ähnlich der SEI Schicht. Die CEI-Schicht (engl. cathode electrolyte interface) ist im Vergleich zur SEI-Schicht jedoch überwiegend reversibel. Es handelt sich hierbei auch um eine Reaktion mit dem Elektrolyt und führt zu einer erhöhten Impedanz der Zelle [10].

- Lithium-Plating ist ein weiterer Effekt bei der Zellalterung. Hierbei handelt es sich um Lithium-Ablagerungen an der Anode der Batteriezelle. Dieser Effekt tritt vermehrt auf, wenn die Zelle bei zu geringer Temperatur mit zu hohen Ladeströmen belastet wird. Bei geringen Ladeströmen hingegen lagert sich das Lithium ungleichmäßig ab, was zu einem Durchstechen des Separators und einem Kurzschluss innerhalb der Batterie führen kann [10][13].

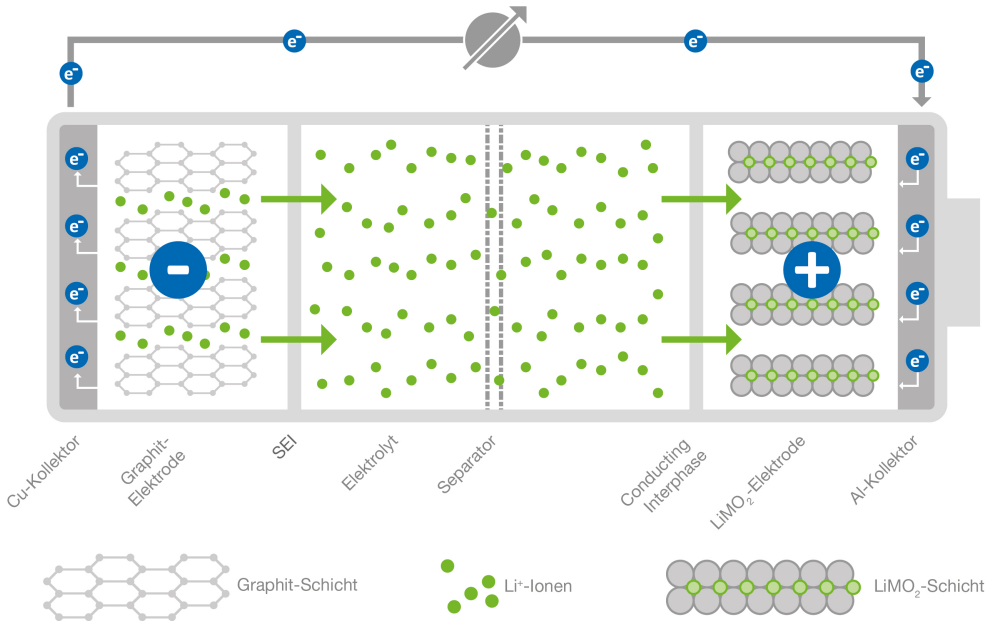


Abbildung 2.3: Die Abbildung zeigt den Aufbau einer Lithium-Ionen-Zelle nach [4].

Bauformen

Bei Lithium-Ionen-Batterien haben sich drei Bauformen etabliert.

- **Zylindrische Zellen** sind gewickelte Zellen in einem festen Metallgehäuse. Diese Bauform ist die geläufigste Bauform im Verbraucherbereich. Die Zellen werden in standardisierten Formaten entwickelt. Die Formatangabe besteht hierbei aus dem Durchmesser und der Höhe der Zelle, eine 18650-Zelle entspricht einer Zelle mit 18 mm Durchmesser und einer Höhe von 65 mm [13].
- **Prismatische Zellen** werden gewickelt oder gestapelt. Vorteile sind hierbei die bessere Wärmeabfuhr bei kompakterer Fertigung. Nachteil ist die aufwendige Fertigung [17][30][13].

- **Pouch Zellen** können in Form und Größe flexibel auf die Anwendung angepasst werden und folgen daher keinem Standard. Das geringe Gewicht und die flexible Außenhülle führt im Gesamtsystem zu einer noch mal höheren Energiedichte [13][10].

Aus den Datenblättern der Batterien lassen sich für die EIS relevante Parameter entnehmen. Aus der Tabelle 2.1 geht z. B. hervor, dass Batterien mit größerer Kapazität einen kleineren Innenwiderstand aufweisen. Werden die einzelnen Lithium-Ionen Zellen so verschaltet, dass eine für das Elektrofahrzeug relevante Kapazität im Gesamtsystem erreicht wird, ergibt immer ein ähnlich geringer Gesamtwiderstand des Batteriesystems. Es zeigt sich, dass trotz unterschiedlichen Materialien, Kapazitäten und Bauformen der Innenwiderstand bei gleicher Kapazität in ähnlichen Bereichen liegt.

Tabelle 2.1: Datenblattangaben von einer Auswahl an Batterien

Hersteller Modell	**Typ**	**Kathodenmaterial**	**Kapazität** (@RT)	**Impedanz** (@1kHz)
Lithium Werks 26650	Round	Eisenphosphat	$2.6Ah$	$\leq 10m\Omega$
Samsung SDI	Prismatic	N.A.	$94Ah$	$\leq 0.75m\Omega$
Tesla Cell	Round	Nickel-Magnesium-Cobalt	$26.5Ah$	$7m\Omega \pm 2$
Panasonic	Prismatic	N.A.	$50Ah$	$\leq 0.8m\Omega$
Molicel INR-21700-P42A	Round	Nickel-Magnesium-Cobalt	$4.2Ah$	$15m\Omega$
LY-LTO-30AH	Prismatic	Lithium-Titanat-Oxid (LTO)	$30Ah$	$\leq 1m\Omega$

Dynamische Eigenschaften der Batterie

Bei einem linearen zeitinvarianten System handelt es sich um ein System, welches lineares Übertragungsverhalten ausweist und unabhängig von zeitlicher Verschiebung ist. Die zeitliche Invarianz gilt für Batterien nur für kurze Zeiträume. Durch die dynamischen Eigenschaften der Batterie wie Ladungsänderung handelt es sich nicht um ein LZI System. Im Allgemeinen ist eine Batterie ein hochdynamisches System, daher kann die LZI Eigenschaft nur angenähert werden. Weiterhin handelt es sich bei einer Batterie weitgehend um ein nicht lineares System [27]. Bei einer Stromanregung durch ein Wechselsignal

kann das Batteriesystem nur für kleine Ausgangsamplituden als linear angenommen werden [44][53].

2.2.2 Batteriemodelle

Der Grad der Komplexität von Batteriemodellen kann relativ groß werden, für die grundlegende Beschreibung reichen aber schon einfache Modellansätze. Ziel der Modelle ist es, das Verhalten der Batterie möglichst genau und effizient widerzuspiegeln. Es lassen sich grob drei Kategorien benennen: Physikalisch-chemisch Modelle, mathematische Modelle und durch elektronische Ersatzschaltbilder beschriebene Modelle [27]. Hierbei verläuft die Genauigkeit gegenläufig zum Rechenaufwand [27].

Bei den **physikalisch-chemischen Modellen** werden vor allem die Materialeigenschaften möglichst genau durch zusammenhängende Differenzialgleichungen beschrieben. Die **mathematischen Modelle** hingegen lassen die physikalischen Eigenschaften weitgehend außen vor. Es handelt sich um eine rein mathematische Betrachtung des Verhaltens einer Batterie. Hierzu zählen analytische Klemmenspannungsmodelle, neuronale Netze, Fuzzy Logiken oder stochastische Batteriemodelle. Die im Ingenieurbereich relevantesten Modelle sind durch **elektronische Ersatzschaltbilder** beschriebene Modelle. Hier wird versucht, über die Kombination verschiedener Standardbauteile das Verhalten einer Batterie nachzubilden. Mithilfe von Widerständen, Kondensatoren und Spannungsquellen kann das Batterieverhalten relativ genau nachgebildet und angepasst werden. Im Folgenden wird beispielhaft ein einfach Batteriemodell mithilfe einer elektronischen Ersatzschaltung beschrieben.

- Spannungsquelle beschreibt den Ladezustand
- Ohmscher Widerstand beschreibt den Widerstand der Anschlüsse, Verbindungen und anderer Batteriematerialen [55]
- ZARC-Element beschreibt den Übergang zwischen Elektrode und Elektrolyt (SEI-Schicht) [22]
- Warburg Impedanz beschreibt die Diffusion von Ladungsträgern im Elektrolyten zwischen den Elektroden (Warburgast) [5]

Ein typisches Ersatzschaltbild für eine Lithium-Ionen-Zelle besteht aus einer idealen Spannungsquelle, um den Ladezustand zu repräsentieren U_{OVC}, einem ohmschen Innenwiderstand R_0 und einem RC-Glied, welches die Durchtrittsreaktion der Lithium-Ionen beschreibt. Die so beschriebenen Modelle basieren auf den Thevenin Modellen, Abbildung 2.4a. Mithilfe des Thevenin Modell kann der Halbkreis im Nyquist-Diagramm dargestellt werden, Abbildung 2.4b. In Abbildung 2.4c ist, aufbauend auf den Thevenin-Modell, das Modell von Randles gezeigt. Der Hauptunterschied zum Thevenin-Modell ist die Einführung der Warburg-Impedanz Z_W, mit welcher sich die Diffusion der Lithium-Ionen modellieren lässt, Abbildung 2.4d. Das Randles-Modell lässt eine detailliertere Beschreibung der Batterie zu und ermöglicht erweiterte Analysen. R_{SEI} und C_{SEI} beschreiben in diesem Modell die SEI-Schicht, dieses RC-Element wird als ZARC Element beschrieben. Die ohmschen Anteile werden durch R_0 beschrieben und die Doppelschichtkapazität mit R_p und C_p [31][56]. Die Warburg-Impedanz lässt sich wie folgt beschreiben:

$$Z_W = \frac{A_W}{\sqrt{j\omega}}$$

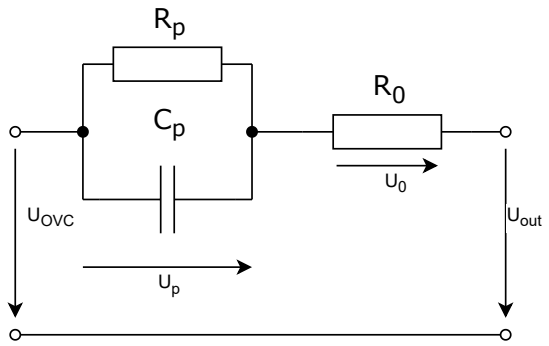
A_W is der Warburg-Koeffizient und setzt sich aus verschiedenen physikalischen Größen zusammen [57].

2.3 Elektrochemische Impedanzspektroskopie

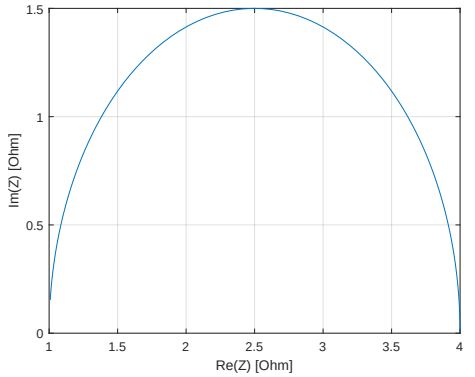
Die EIS ist ein etabliertes Verfahren, bei welchem ein Wechselsignal mit einer gegebenen Frequenz auf ein elektrochemisches System aufgebracht wird. Anhand der Impedanz kann eine Aussage über den aktuellen Zustand eines elektrochemischen Systems getroffen werden [21][41]. Anwendungsbereiche sind elektrochemische System wie beispielsweise:

- Brennstoffzellen [15][60]
- Batterien [26][44][43]
- Materialforschung [32]
- Medizintechnik [29]

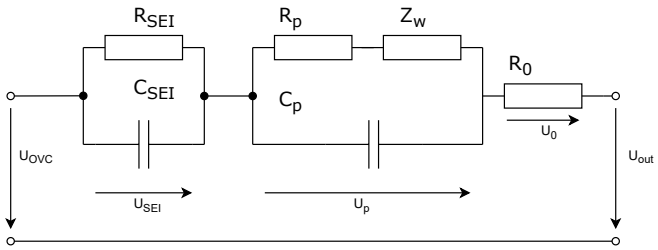
Die verwendeten Frequenzen sind stark anwendungsbezogen und können theoretisch von wenigen Mikrohertz bis in den Megahertz-Bereich reichen, ein für Batterien typischer Messbereich liegt zwischen 1 mHz und 10 kHz [44]. Voraussetzung für die Anwendung der



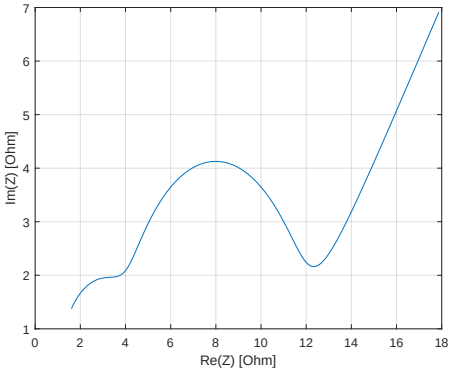
(a) Modell nach Thevenin



(b) Nyquistdarstellung des Thevenin-Modells



(c) Modell nach Randles



(d) Nyquistdarstellung des Randles-Modells

Abbildung 2.4: Einfache Batteriemodelle mit dazugehörigem Nyquistdarstellung. Das Thevenin Modell (a) ist die Basis zur Erweiterung und wird durch ein einfaches RC-Glied dargestellt. Erweitert man das Modell um ein weiteres RC-Glied nach Randles und fügt die Warburg-Impedanz hinzu (c), erhält man die Effekte der SEI-Schicht sowie die Diffusionseffekte. Die Nyquistdarstellung zeigt die schrittweise Annäherung an das Verhalten einer realen Lithium-Ionen-Zelle.

Impedanzspektroskopie ist, dass es sich bei dem untersuchten System um ein lineares, zeitinvariantes (LTI) System handelt. Batterien weisen generell ein über den Ladezustand hinweg höchst nicht lineares Verhalten auf. Bei der Anwendung der Impedanzspektroskopie ist daher zu beachten, dass sich während der Messungen der Zustand nicht zu stark ändert, um eine ausreichende Linearität in dem aktuellen Arbeitspunkt zu gewährleisten vgl. [27]

2.3.1 Komplexer Wechselstromwiderstand

Zur Bestimmung der EIS, muss der komplexe Wechselstromwiderstand $\underline{Z}$ der Batterie bei verschiedenen Frequenzen bestimmt werden.

Zur Berechnung des komplexen Wechselstromwiderstandes wird analog das ohmsche Gesetz angewandt.

$$\underline{Z}(\omega_0) = \frac{\underline{U}(\omega_0)}{\underline{I}(\omega_0)} \tag{2.13}$$

mit der Kreisfrequenz ω_0

$$\omega_0 = 2\pi f \tag{2.14}$$

In der komplexen Wechselstromrechnung sind folgende Formeln für Strom $\underline{I}$ und Spannung $\underline{U}$ definiert.

$$\underline{U}(\omega_0) = U_0 \cdot (cos(\omega t + \varphi_U) + jsin(\omega t + \varphi_U)) \tag{2.15}$$

$$\underline{I}(\omega_0) = I_0 \cdot (cos(\omega t + \varphi_I) + jsin(\omega t + \varphi_I)) \tag{2.16}$$

$$\Delta\varphi = \varphi_U - \varphi_I \tag{2.17}$$

mithilfe der eulerschen Darstellung $e^x = (cos(x) + jsin(x)))$ und der Phasenlage $\Delta\varphi$ ergibt sich die Formel 2.18 für die Impedanz, den komplexen Wechselstromwiderstand.

$$Z_0 \cdot (cos(\Delta\varphi) - jsin(\Delta\varphi)) = Z \cdot e^{j\Delta\varphi} \tag{2.18}$$

2.3.2 Ableitbare Zustandsgrößen

Mithilfe der durch die EIS bestimmten Impedanz können verschiedene Zustandsgrößen der Zelle abgeleitet werden. Hierzu zählen unter anderem Druck, Temperatur, State of

Charge (SoC) und State of Health (SoH) [46]. Was von den Messungen abgeleitet werden kann, ist vor allem von den verwendeten Modellen abhängig. Im Folgenden werden beispielhaft einige durch die EIS ableitbare Zustandsgrößen beschrieben.

Temperatur

Die Zellimpedanz gerade von Lithium-Ionen-Zellen ist besonders sensitiv für Temperaturänderungen, dies führt dazu, dass durch die EIS eine Bestimmung der mittleren Zelltemperatur über die Impedanz möglich ist [46]. Es ergibt sich in Anbetracht der LZI-Eigenschaften die Voraussetzung, dass die Messung bei konstanter Temperatur erfolgt. Dies ist aufgrund der zeitlichen Ausdehnung der Messung bei niedrigen Messfrequenzen besonders zu beachten.

Druck

Durch z. B. mechanische Verformung entsteht Druck, welcher eine Impedanzänderung hervorruft. Dies geschieht vor allem durch die Verformung der porösen Materialien in der Batterie [46]. Dies kann durch mechanische Vorspannung einer Batteriehalterung oder durch externe Einflüsse wie Beschädigung von Zelle hervorgerufen werden und ist mithilfe der EIS messbar.

Ladezustand

Auch der Ladezustand (engl. State of Charge (SoC)) steht im Zusammenhang mit der Impedanz. Bei Bleibatterien hat dieser einen direkten Einfluss auf den Elektrolyten, wobei der SoC bei Lithium-Ionen auf Prozesse, wie den Ladungsdurchtritt und die Festkörperdiffusion größeren Einfluss hat [46]. Der SoC lässt sich rechnerisch wie folgt bestimmen:

$$SoC = \frac{C_{ist}}{C_N} \tag{2.19}$$

Der SoC wird in Prozent angegeben und durch das Verhältnis der Nennkapazität zur aktuellen Kapazität berechnet. Eine vollständig entladene Batterie hat demnach einen Ladezustand von 0% eine vollgeladene Batterie entsprechend 100%.

Alterungszustand

Über die Lebensdauer ändert sich die Kapazität und damit auch die Impedanz einer Batteriezelle. Der Alterungszustand wird als SoH angegeben. Die Impedanzänderung der Batterie durch die Alterung wird durch eine Vielzahl von Faktoren wie z. B. durch die Zunahme der SEI-Schicht beeinflusst [46]. Der Alterungszustand führt über die Zeit zu einem Drift der von ihm abhängigen Parametern. Daraus folgt, dass z.B die Temperaturmessung über die Zeit eine adaptive Kennlinienanpassung benötigt, um die Messung präzise interpretieren zu können. Der Alterungszustand der Batterie lässt sich durch das Verhältnis der noch maximal vorhandenen Kapazität der Zelle C_m und der Nennkapazität berechnen.

$$SoH = \frac{C_m}{C_N} \tag{2.20}$$

Leistungsfähigkeit

Der State of Function (SoF) beschreibt die Leistungsfähigkeit der Batterie und trifft eine Aussage über die maximale Leistungsabgabe einer Batterie [48]. Die Berechnung kann hierbei vergleichsweise komplex werden vgl. [20]. Der SoF gibt den Zusammenhang von SoH und SoC an, anders als z.B der SoH berücksichtigt der SoF die Leistungsfähigkeit bei variablen Temperaturen, SoC und SoH. Der SoF soll verwendet werden, um vorauszusagen, ob die Batterie im aktuellen Zustand in der Lage ist, die bevorstehende Aufgabe zu erfüllen [39].

3 Stand der Technik

Der Stand der Technik ist aktuell zum großen Teil auf Laborgeräte beschränkt. Auch wenn die EIS ein grundsätzlich etabliertes Verfahren ist, wird sie bisher wenig in Elektrofahrzeugen eingesetzt. In diesem Kapitel wird zunächst auf die möglichen Entwicklungsansätze und anschließend auf bestehende Messsysteme eingegangen.

3.1 Entwicklungsansätze

In Abbildung 3.1 sind verschiedene Möglichkeiten zur Entwicklung eines EIS-Sensor-Systems aufgezeigt. Hierbei kann die Anregung über eine Quelle gezielt an jeder Zelle oder zentral erfolgen. Eine passive Anregung erfolgt durch das Bordnetz des Fahrzeugs. Es wird nicht gezielt mit einer Frequenz angeregt, sondern es werden die im Lastverlauf befindlichen Frequenzen ausgenutzt, um eine EIS-Messung durchzuführen. Von einer ladungsneutralen Anregung spricht man, wenn sich der Ladezustand der Batterie nach der Messung nicht verändert hat. Eine ladungsnegative Anregung hingegen verändert den Ladezustand der Batterie. Die Berechnung der Impedanz erfolgt im Anschluss entweder zentral oder dezentral, wobei aufgrund des Datenaufkommens die dezentrale Auswertung nach [46][43] zu bevorzugen ist. Grundsätzlich kann die EIS auch beim Ladepro-

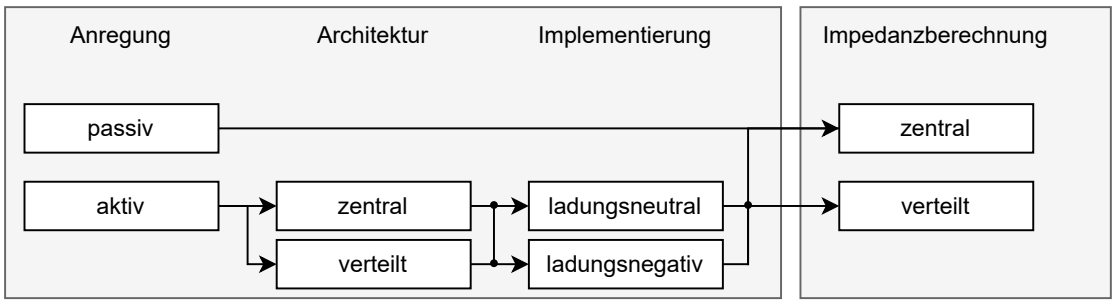


Abbildung 3.1: Die Abbildung zeigt die grundlegenden Ansätze, um ein EIS-Sensor-System zu implementieren. Bild nach [46].

zess durchgeführt werden, womit der Gesamtprozess auch als Ladungspositiv aufgefasst werden kann. Für die Einordnung des eigentlichen Messprozesses sind die Ansätze der ladungsneutralen oder der ladungsnegativen Messung jedoch üblich.

3.2 Multifrequenzanregung

Üblicherweise werden zur Durchführung der EIS einzelne Frequenzen verwendet. Dies führt bei kleinen Frequenzen zu einer langen Messzeit. Die Autoren aus [61] stellen kein konkretes Sensorsystem vor, aber eine beschleunigte Messmethode zur Bestimmung der EIS. Der Ansatz besteht darin, mehrere Anregungsfrequenzen gleichzeitig zu nutzen und auszuwerten. Der Vorteil an diesem Ansatz ist eine EIS, die über das Spektrum von 1 Hz − 1 kHz ca. 60 mal schneller ermittelt werden kann als mit bisherigen Verfahren. Laut der Aussage der Autoren ist es möglich, auch dynamische Zustandsänderung durch die EIS zu erfassen. Zu diesem und ähnlichen Ansätzen gibt [62] eine Übersicht. Die Stromanregung wird mithilfe unterschiedlicher Verfahren realisiert. Verwendet werden Rechtecksignale oder sich überlagernde Sinuswellen mit verschiedenen Frequenzen. Eine weitere Methode, eine EIS durchzuführen, geht über Stromimpulsen und Spannungsantworten [33]. Die mit dieser Methode erhaltenen EIS stimmen mit denen überein, die mit der traditionellen Methode erhalten wurden [61].

3.3 Labormessgeräte zur Impedanzmessung

Laborgeräte unterscheiden sich zu integrierbarer Sensorik. Es gibt für Laborgeräte keine wesentlichen Anforderungen in Größe und Gewicht. Sie haben die vollständige Messkette in einem Gerät implementiert, hierzu zählen:

- Anregung (Strom oder Spannung)
- Messung von Strom und Spannung
- Filterung
- Impedanzberechnung

Tabelle 3.1: Datenblattangaben einer Auswahl an EIS-Metern für den Laborgebrauch. Die Phasengenauigkeit beträgt bei den aufgeführten Geräten ±1°.

Gerät	Impedanz Bereich	Impedanz Genauigkeit	Frequenz Bereich
BioLogic [8]	N/A	N/A	10 µHz bis 1 MHz
FuelCon True EIS [18]	5 µΩ bis 15 Ω	0,1 mΩ	200 µHz bis 100 kHz
Digatron EIS Meter [12]	0,3 mΩ bis 3000 mΩ	±1%	1 mHz bis 6,5 kHz
EIS-Box [16]	N/A	±1%	10 µHz bis 100 kHz

Wie in Tabelle 3.1 zu sehen, decken die Labormessegeräte in der Regel einen großen Frequenzbereich bei hoher Genauigkeit ab. Im Vergleich zu einer dezentralen Sensorik können die Laborgeräte jedoch keine verteilte Messung durchführen, sondern beschränken sich auf einzelne Zellen oder Zellverbände. Der Einsatzbereich ist wie Kapitel 2.3 beschrieben, breit und beschränkt sich nicht auf Batteriezellen.

3.4 Sensorik zur Impedanzmessung

Es gibt stand heute wenig frei auf dem Markt verfügbare Sensorik zur Impedanzmessung. Neben einigen nicht kommerziellen Arbeiten von Fach- und Forschungsgruppen gibt es nur wenige kommerzielle Anbieter von EIS-Sensorik. Sensorik in diesem Bereich zeichnet sich durch ihre kleine Bauform und den flexiblen Einsatzbereich aus. Sie ist weniger genau als kommerzielle Laborgeräte und meist auch nicht in der Lage, die gesamte Messung eigenständig durchzuführen. Es wird eine externe Stromanregung benötigt und je nach Bauart liefert das Sensorsystem Strom und Spannungswerte oder direkt eine Impedanz.

3.4.1 Beispiel kommerzieller Lösung

Die Firma Datang NXP bietet einen *Einzelzell* EIS-Sensor an. Für den Integrated Circuit (IC) werden folgende Daten angegeben:

- Automotive qualified per AEC-Q100
- Genauigkeit der Spannungsmessung: ±2 mV
- Zellspannung: 1,9 V - 5,5 V
- Temperaturgenauigkeit: ±2,5 K

Weiterhin ist explizit aufgeführt, dass der Sensor für niederohmige Zellen verwendet werden kann (ohne Angaben) [36]. Bei dem IC handelt es sich um ein Einzelzellsensor. Der Sensor benötigt eine externe Anregung. Ob es sich um eine passive oder eine aktive Anregung handelt, ist nicht weiter spezifiziert. Die Berechnung der Impedanz erfolgt auf dem IC. Der IC kann mithilfe einer Daisy-Chain kommunizieren bei bis zu 225 Teilnehmern. Ein kommerzielles integrierbares Multi-Cell-Messsystem ist zum Zeitpunkt der Erstellung der Arbeit nicht bekannt.

3.4.2 Vorarbeiten in der Forschungsgruppen

In einer früheren Arbeit in der Forschungsgruppe der Hochschule für Angewandte Wissenschaften Hamburg wurde ein EIS-Sensorsystem basierend auf einer Funkkommunikation entwickelt [42]. Das System erreichte einen Proof-of-Concept Status. Die Messung der EIS erfolgt über ein Messverfahren, welches in zwei Phasen aufgeteilt ist. Zunächst wird hier ein Subtrahierer mit nachgeschalteter Verstärkerstufe verwendet, um den Gleichanteil der Batterie zu kompensieren und das Nutzsignal anschließend in der zweiten Phase ausreichend zu verstärken. Der Autor geht hier von einer Signalamplitude von ca. 3 mV aus und löst das Signal mit mindestens 100 Quantisierungsstufen auf. Die Stromanregung erfolgt extern und zentral. Die Auflösung über die 100 Quantisierungsstufen wird über das Verstärken des Nutzsignals erreicht. Verglichen mit einem konventionellen EIS-Meter konnten gute Ergebnisse erzielt werden. Eine konkrete Genauigkeit wird nicht angegeben.

- Frequenzbereich: 100 mHz - 2 kHz
- 12-Bit Analog Digital Converter (ADC)
- Verteilte Signalverarbeitung
- Einzelzellmessung

3.4.3 Arbeiten anderer Forschungsgruppen

In der Forschungsgruppe der Technische Universität Chemnitz ist ein Sensorsystem zur mobilen Durchführung der EIS-Messung entwickelt worden. Das beschriebene System ermöglicht neben der Messung auch die Anregung des zu messenden Systems. Ein besonderer Fokus liegt auf der Anregung mit Binärsequenzen über eine zuvor synthetisiertes

Multifrequenzsignal. Die Messung erfolgt jeweils an einem Batteriestack von vier Zellen, die Berechnung der Impedanz erfolgt mithilfe des Goertzel-Filters. Eine Angabe zum Messbereich und der erreichten Genauigkeit wird nicht gemacht [53].

4 Konzeption

In diesem Kapitel wird die Erarbeitung der Konzeption des Sensorsystems beschrieben. Es wird auf unterschiedliche Möglichkeiten zur Problemlösung eingegangen und die Entscheidungsgrundlage dargestellt.

Von den in Kapitel 3.1 aufgeführten Lösungsmöglichkeiten für eine EIS-Sensorik wird hier der Ansatz der ladungsneutralen Einzelzellmessung mit zentraler Anregung und dezentraler Verarbeitung verfolgt. Das System soll in der Lage sein, die Impedanz einer Zelle bzw. eines kleinen Zellverbundes eines größeren Gesamtsystems zu bestimmen.

Ein besonderer Fokus liegt hierbei auf den unterschiedlichen Methoden der Kompensierung des Gleichanteils und einer maximalen Signalaussteuerung bei gegebener Dynamik der Batterie.

4.1 Auswahl des Mikrocontrollers

Für den Einzelzellsensor muss ein Mikrocontroller pro Sensor verbaut werden. Der Mikrocontroller soll über einen ADC verfügen, um das analoge Signal aufnehmen zu können. Weiterhin müssen einfache Bussysteme wie SPI oder I^2C unterstützt werden. Eine ausreichende Anzahl an GPIOs muss verfügbar sein. Diese Kriterien werden von vielen Mikrocontrollern auf dem Markt erfüllt. Ein bewährter Chip ist der ARM-CORTEX-M0, hierbei handelt es sich um den kleinsten Chip der ARM-Cortex Reihe, welcher von einer Vielzahl von Herstellern verbaut wird. Beispielsweise von STMicroelectronics bei dem STM32L073Z-EVAL, Texas Instruments, bei dem MSPM0L1304, oder Infineon mit dem Mikrocontroller XMC1100.

Die Tabelle 4.1 zeigt eine Auswahl der Spezifikation der verschiedenen Mikrocontroller. In der Arbeit wird der XMC1100 verwendet, dieser hat ausreichend ADC Channel und verfügt im Vergleich zu dem MSPM0L1304 ausreichend SRAM-Speicher. Der *XMC1100*

Mikrocontroller	GPIOs	SRAM	SPI	ADC (Channel)
XMC1100	40	16 kB	2	12-Bit (12)
STM32L0	84	20 kB	2	12-Bit (1)
MSPM0L1304	28	4 kB	1	12-Bit (10 extern)

Tabelle 4.1: Die Tabelle zeigt eine Auswahl an Spezifikationsdaten verschiedener Mikrocontroller.

wird auf einem Evaluationsboard verwendet. Der XMC1100 hat eine Kernfrequenz von 32 MHz und bis zu 64 MHz Peripherietaktung. Weiterhin verfügt der XMC1100 über 64 kB Flash Speicher und 16 kB SRAM. Die Betriebsspannung liegt im Bereich von $1{,}8\,\mathrm{V} - 5{,}5\,\mathrm{V}$. Der *XMC1100* wird mit der Programmiersprache *C* und der Infineon eigenen Entwicklungsumgebung DAvE programmiert. Da es sich um einen ARM-CORTEX M0 Chip handelt, ist eine benutzerdefinierte Entwicklungsumgebung möglich. Der Controller selbst hat eine kompakte Bauform und ist auch mit dem XMC2Go Breakoutboard noch klein, aber ohne größeren Aufwand direkt verwendbar.

Entwicklungsumgebung

Die DAvE Entwicklungsumgebung besitzt einen Quellcodegenerator, welcher die Konfiguration der Peripherien vereinfachen soll. Der so erzeugte modulare Quellcode wird von DAvE als *App* bezeichnet. Die so generierte *App* beinhaltet neben der Konfiguration und Initialisierung aller notwendigen Peripheriekomponenten auch eine Schnittstelle für das Nutzen des generierten Moduls. Der Quellcodegenerator generiert eine Abstraktionsschicht zwischen der eigentlichen Anwendung und dem Board Support Package des Controllers. Die Abbildung 4.1 zeigt die durch DAvE erzeugten Abstraktionsschichten. Hierbei wird bei vielen Funktionen die Peripherie des Controllers kombiniert verwendet. Der von DAvE erzeugte Quellcode enthält keine Programm- oder Anwendungslogik, sondern liefert lediglich die Schnittstelle zur Verwendung der Funktionen des Mikrocontrollers.

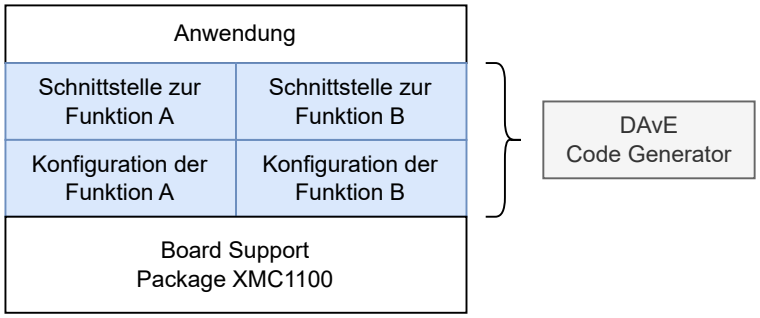


Abbildung 4.1: Die Abbildung zeigt die von *DAvE* generierten Abstraktionsschichten (blau). Der Quellcodegenerator vereinfacht die Konfiguration und Verwendung der Controllerperipherie.

4.2 Bestimmung der Systemanforderungen

Im Folgenden werden die äußeren Rahmenbedingungen des Systems aufgeführt. Die Anforderungen ergeben sich aus physikalischen Gegebenheiten, typischen Daten von im Elektroauto verwendeten Batteriezellen und den Ziel- und Eingangsgrößen. Hierbei werden auch Annahmen getroffen, um einen Rahmen zu schaffen, in welchem das finale Sensorsystem betrieben werden kann.

4.2.1 Elementare Einflussgrößen

Die Anforderungen an das Sensorsystem leiten sich vornehmlich von dem zu messenden Objekt ab. Bei dem Messobjekt handelt es sich um eine Lithium-Ionen-Zelle, diese hat einen signifikanten Einfluss auf das Gesamtsystem. Bei der Messgröße handelt es sich bei der Impedanzmessung implizit um die Signalamplituden von Strom und Spannung sowie deren relative Phase zueinander.

Lithium-Ionen-Batterie

Die Batterie selbst hat einen Ladezustand (SoC) und bringt damit einen Gleichanteil mit in die Messkette ein. Ein üblicher Spannungsbereich liegt zwischen 1,8 V und 4,2 V, wobei die Nennspannung ca. 3,6 V beträgt [35], siehe Kapitel 2.2. Daraus folgt für die Messung, dass der zu kompensierende Gleichanteil von 1,8 V bis 4,2 V zum Nutzsignal addiert wird.

Weiterhin ist der Innenwiderstand der Batteriezelle ein wesentlicher Parameter in der Messkette, da dieser direkt Einfluss auf die zu erwartende Amplitude des Messsignals hat. Wie in Kapitel 2.2 zu sehen, sind übliche Werte für den Innenwiderstand, R_i, bei der für das Elektrofahrzeug relevanten Kapazität in der Regel im Mikroohm-Bereich. Wenn zum Messen kein hochauflösender ADC verwendet werden soll, ergibt sich die Notwendigkeit, den durch die Batterie eingebrachten Gleichanteil zu kompensieren, um den ADC durch die nötige Verstärkung nicht in die Sättigung zu treiben. Der Sensor muss in der Lage sein, einen Gleichanteil von 1,8 V bis 4,2 V zu kompensieren.

Messsignal

Das zu messende Signal, um eine EIS durchzuführen, wird extern durch eine Stromanregung der Batterie erzeugt. Die gewählte Amplitude der Stromanregung ist abhängig von dem Innenwiderstand R_i der Batterie und muss groß genug sein, um eine Messung durchführen zu können. Der zu erwartende Innenwiderstand der Batterie liegt wie in Abschnitt 2.2 aufgeführt im Milliohm-Bereich. Die Abbildung 4.2 zeigt hier die theoretisch mögliche Amplitude der Spannungsantwort bei unterschiedlichen Stromanregungen und Innenwiderständen der Batteriezellen. Dies zeigt, dass sich die Messamplitude im Millivolt-Bereich befindet.

Der Sensor muss demnach in der Lage sein, eine Amplitude von $V_{pp} < 1\,\mathrm{mV}$ zu messen. Um eine möglichst hohe Genauigkeit der EIS erzielen zu können, muss das Signal dabei so hoch wie möglich aufgelöst werden. Die ist entweder durch einen hochauflösenden ADC oder durch eine signifikante Verstärkung des Signals zu erreichen. In dieser Arbeit wird der interne ADC des Mikrocontrollers verwendet, dieser löst mit 12-Bit auf, was eine vorherige Verstärkung voraussetzt.

Messfrequenzen

Die Messfrequenzen der EIS sind von der späteren Verwendung der Messung abhängig. Für eine Bestimmung des SoC werden niedrige Messfrequenzen im Millihertz-Bereich verwendet [28][34]. Eine Messung des SoH findet im zweistelligen Hertzbereich statt [34][40] und die Temperaturbestimmung mithilfe der EIS wird eher im Bereich von einigen Kilohertz durchgeführt [6][52]. Laborgeräte haben hier einen breiten Messbereich von wenigen

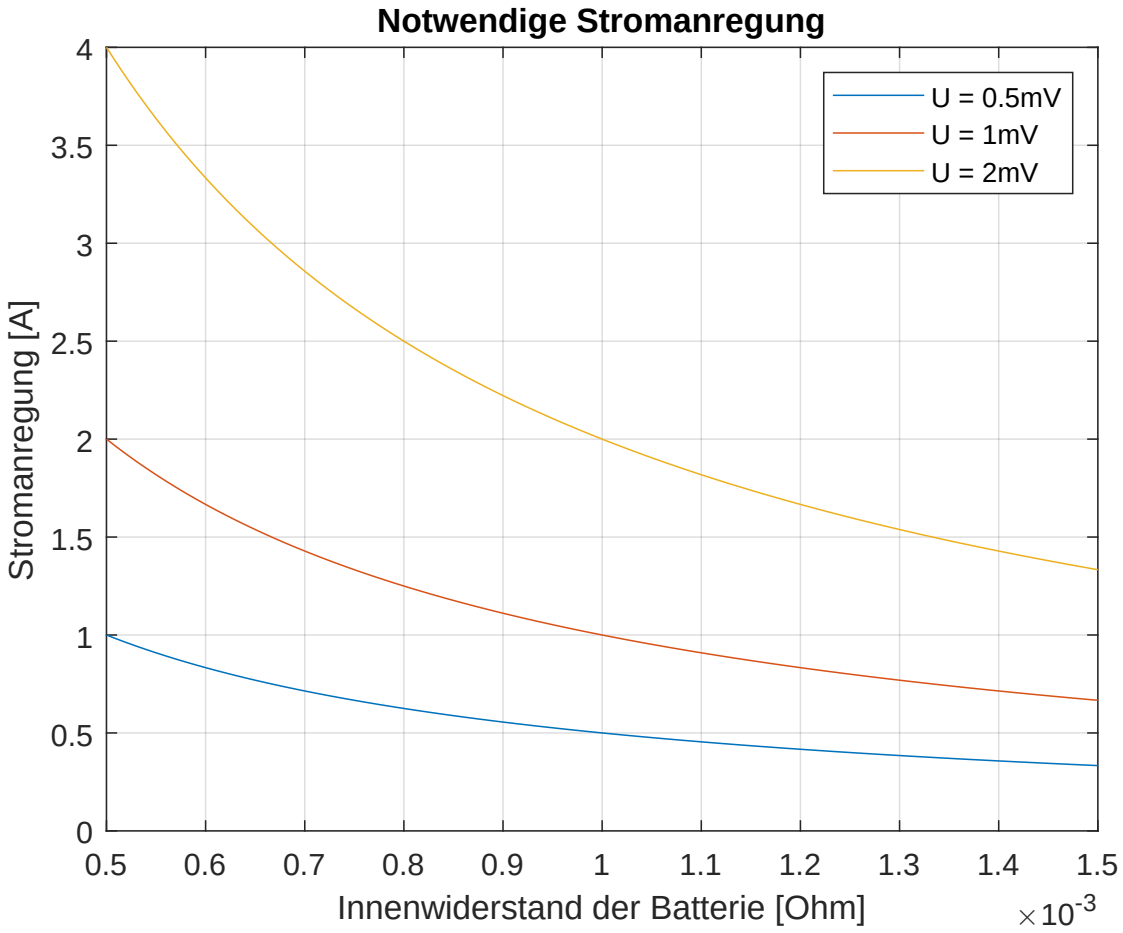


Abbildung 4.2: Die Abbildung zeigt die notwendige Stormanregung bei unterschiedlichen Innenwiderständen der Batterie.

Mikrohertz bis in den Kilohertz-Bereich, siehe Tabelle 3.1. Beim Einsatz im Elektrofahrzeug wird dieser beschränkt. Gerade Frequenzen im Millihertz-Bereich und darunter benötigen verhältnismäßig viel Zeit. Beispielsweise braucht eine Periode T bei 1 mHz schon 16,6 min. Üblicherweise müssen hier mehrere Perioden aufgezeichnet werden. Den genauen Messbereich, welcher von dem Sensor später abgedeckt werden kann, wird im Versuch ermittelt. Ziel ist es, einen üblichen Bereich abzudecken, dieser ist mit 1 mHz bis 10 kHz angegeben [56].

4.2.2 Anforderungen

Aus dem Abschnitt 4.2.1 ergeben sich die Anforderungen an das Sensorsystem. Das Sensorsystem muss in der Lage sein, das Nutzsignal zu erfassen und möglichst hoch aufzulösen. Aus den Datenblättern der Labormessgeräte wird weiterhin abgeleitet, dass der Sensor eine Phasengenauigkeit von 1° aufweisen soll, um eine aussagekräftige Impedanz der Batteriezelle bestimmen zu können. Die Anforderungen sind Orientierungsgrößen für das Gesamtsystem.

1. Spannungsbereich: 1,8 V bis 4,2 V
2. Nutzsignal: $V_{pp} < 1\,\mathrm{mV}$
3. Amplitudenauflösung: 1%
4. Phasengenauigkeit: 1°
5. Messfrequenzen: $1\,\mathrm{mHz} - 10\,\mathrm{kHz}$

4.3 Kompensierung des Gleichanteils und Nutzsignalverstärkung

Das Messsignal ist ein *AC* Signal, welches mit einem hohen *DC* Gleichanteil behaftet ist. Typischerweise ist $U_{DC} >> U_{AC}$ und liegt bei einem Verhältnis von weniger als $1/100$. Bei dem Gleichanteil handelt es sich um die Batteriespannung. Das Gesamtsignal setzt sich demnach wie folgt zusammen:

$$U_{ges} = U_{DC} + U_{AC} + U_{noise}$$

$$U_{ges} = U_{bat} + U_{signal} + U_{noise} \tag{4.1}$$

Zur Kompensierung des Gleichanteils können unterschiedliche Ansätze verfolgt werden:

- AC-Kopplung
- Subtrahierer mithilfe eines programmierbaren Verstärkers (PGA)

- Kompensierung durch PWM und Tiefpassfilter
- Kompensierung durch einstellbaren Spannungsteiler

Unabhängig von der verwendeten Lösung muss nach der Kompensierung des Gleichanteils das Signal verstärkt werden. Daher besteht die analoge Vorverarbeitung aus zwei Stufen:

1. Gleichanteil kompensieren
2. Signal verstärken

4.3.1 Vorüberlegung zu digitalen Bauelementen

Im Folgenden werden in jeder Lösung digitale/programmierbare Bauteile verwendet. Eine ausschließlich analoge Beschaltung der Verstärker würde nicht die notwendige Flexibilität bieten. Da für die Batteriespannung nur ein Bereich angegeben werden kann, muss es möglich sein, die Beschaltung auf die aktuelle Batteriespannung anzupassen. In [42] wurde hierfür ein digitales Rheostat verwendet. Damit können z. B. Verhältnisse von Spannungsteilern oder Verstärkungsfaktoren variabel eingestellt werden. Ein Nachteil der digitalen Beschaltung ist die Diskretisierung. Wohingegen der in [42] verwendete Rheostat mit 1024 einstellbaren Stufen quasi kontinuierlich ist, gibt es bei Programable Gain Aplifier (PGA) auch diskrete Verstärkerstufen, diese sind oftmals jedoch nicht äquidistant. Dies ist nicht für jede Anwendung flexibel genug. Grundsätzlich bringt die Möglichkeit, in analogen Schaltungen eine programmatisch einstellbare Komponente einzubringen mehr Flexibilität. Diese ist auch bei der zugrunde liegenden Problemstellung notwendig.

4.3.2 Signalverstärkung

Nachdem das Nutzsignal isoliert wurde, muss es für eine ausreichende Auflösung mithilfe von einer Verstärkerschaltung verstärkt werden. Da die genaue Amplitude des Messsignals nicht bekannt ist, muss eine einstellbare Verstärkung möglich sein. Es ist bekannt, dass ein hoher Verstärkungsfaktor benötigt wird. Hier hat [42] einen nicht invertierten Messverstärker mit Rheostat im Rückkopplungszweig eingesetzt und ist so in der Lage, einen Verstärkungsfaktor k von bis zu $k = 100$ in 1024 Stufen einzustellen.

4.3.3 AC-Kopplung

Der einfachste Ansatz ist die AC-Kopplung über eine RC-Hochpass-Filterung. Von einer AC-Kooplung spricht man, wenn das Signal mithilfe eines Kondensators von dem Gleichanteil entkoppelt wird. Der Gleichanteil wird durch den Kondensator vollständig blockiert. Zur Veranschaulichung wird die Gleichung der frequenzabhängigen Kondensatorimpedanz-Gleichung 4.2 betrachtet.

$$Z_c = \frac{u(\omega)}{i(\omega)} = \frac{1}{j\omega C} \tag{4.2}$$

Für $\omega \to 0$ geht $Z_c \to \infty$. Die Impedanz des Kondensators blockiert demnach den Gleichanteil.

Abbildung 4.4 zeigt einen RC-Hochpassfilter. Es muss eine ausreichend große Kapazität genutzt werden, um kleine Frequenzen nicht zu blockieren. Dies ist auch ein wesentlicher Nachteil des Ansatzes. Die Kapazität des Kondensators muss deutlich größer sein als die mithilfe von Gleichung 4.3 berechneten Kapazität, da ab der Grenzfrequenz schon eine Dämpfung um 3 dB stattfindet. Die Abbildung 4.3 zeigt hier, wie sich die Frequenz zur Kondensatorkapazität rechnerisch verhält.

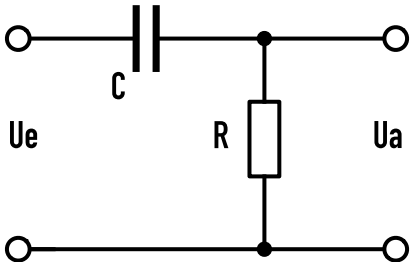


Abbildung 4.4: Die Abbildung zeigt ein RC-Hochpassfilter. Das Filter blockiert niedrige Frequenzen und lässt nur den Wechselanteil passieren [1].

Umso kleiner die Frequenz, desto größer muss die Kapazität des Kondensators sein, um den Gleichanteil zu blockieren, aber das niederfrequente Signal passieren zu lassen. Durch die notwendige Kapazität ist die bauliche Ausdehnung relativ groß. Die zugrunde liegende Gleichung zur Berechnung der Grenzfrequenz f_g ergibt sich wie folgt:

$$f_g = \frac{1}{2 \cdot \pi \cdot R \cdot C} \tag{4.3}$$

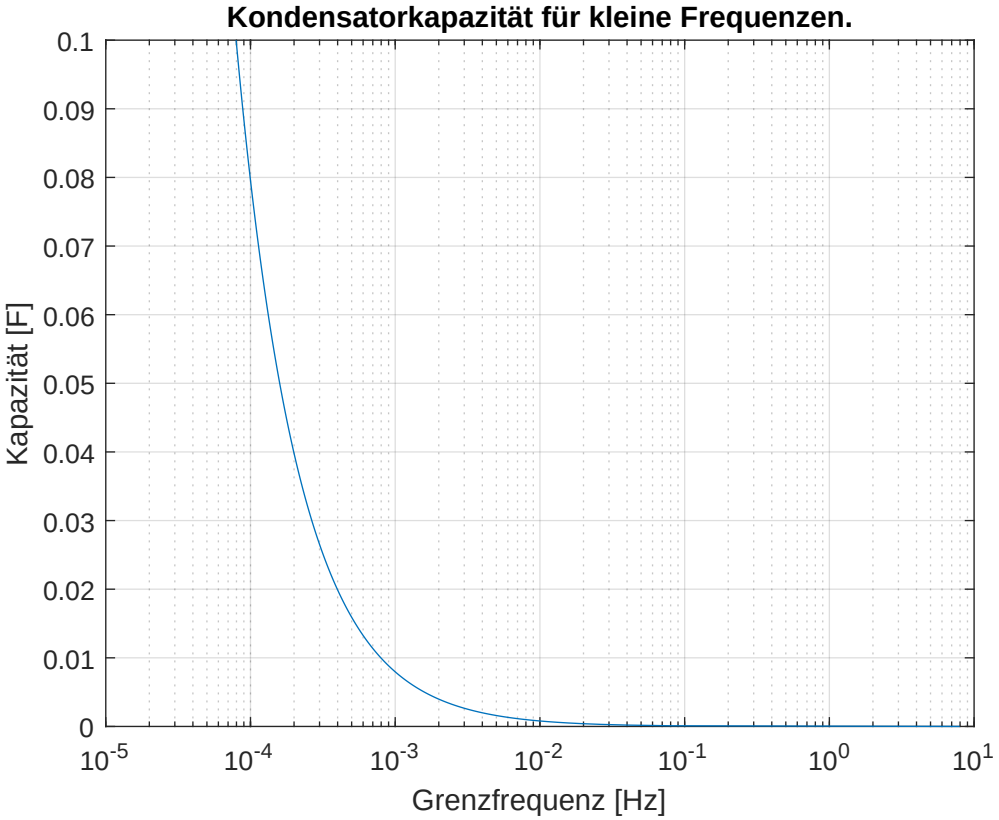


Abbildung 4.3: Die Abbildung zeigt die notwendige Kapazität für eine AC-Kopplung über Filterung mit einem 20 kΩ Filterwiderstand bei kleinen Frequenzen.

Aus Gleichung 4.3 zeigt sich: Umso größer RC, desto kleiner die Grenzfrequenz fg. Die Abbildung 4.5 zeigt die Simulation eines RC-Hochpassfilters mit der Grenzfrequenz $f_g = 0{,}001\,\text{Hz}$. Die Abbildung zeigt, dass die berechnete Grenzfrequenz deutlich unter der eigentlichen Signalfrequenz liegen muss, um keinen Einfluss auf Amplitude und Phase des Messsignals zu haben.

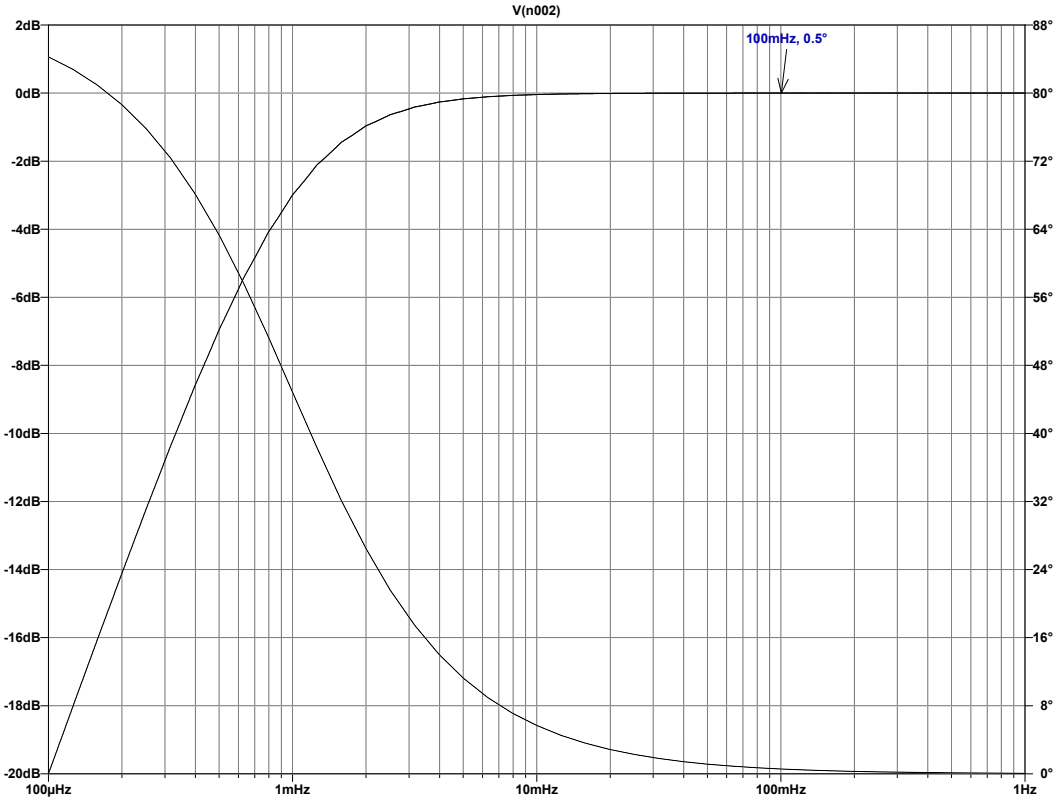


Abbildung 4.5: Die Abbildung zeigt den Amplituden- und Phasengang des RC-Hochpassfilters bei einer Grenzfrequenz von 1 mHz.

4.3.4 Subtrahiererschaltung

Der zweite Ansatz über eine Subtrahiererschaltung mithilfe eines Operationsverstärkers ermöglicht das Subtrahieren zweier Signale voneinander. Dies ergibt sich bei gleichen Widerständen am Operationsverstärker. Die Beschaltung des Operationsverstärkers ist in Abbildung 4.6 zu sehen. Ziel der Schaltung ist das bei $U2 = U1$ die Ausgangsspannung U_a dem Arbeitspunkt des Operationsverstärkers entspricht. Die Ausgangsspannung U_a ergibt sich dabei allgemein wie folgt:

$$U_a = U_{e2} \cdot \frac{R1 + R2}{R1} \cdot \frac{R4}{R3 + R4} - \frac{R2}{R1} \cdot U_{e1} \tag{4.4}$$

Wenn $R1 = R3$ und $R2 = R4$ vereinfacht sich die Gleichung zu:

$$U_a = \frac{R2}{R1} \cdot (U_{e2} - U_{e1}) \tag{4.5}$$

Der Verstärkungsfaktor wird durch das Verhältnis von $R2$ zu $R1$ eingestellt. Außerdem zeigt Gleichung 4.5, dass sich bei gleichen Widerständen $R1 = R2$ eine Verstärkung von eins einstellt und die Schaltung nur noch die Differenz zwischen (U_{e2} und U_{e1} passieren lässt.

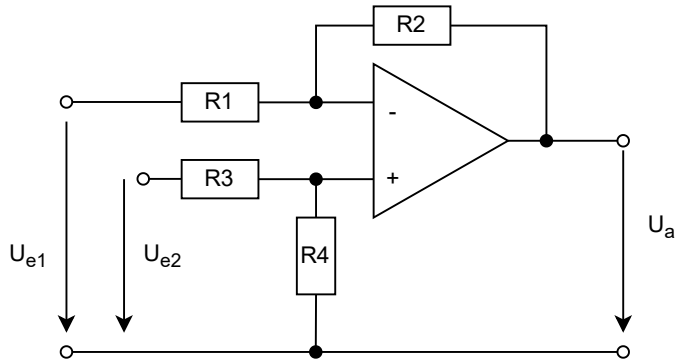


Abbildung 4.6: Die Abbildung zeigt eine Subtrahiererschaltung bzw. einen Differenzenverstärker. Sind alle Widerstände gleich, subtrahiert die Schaltung die beiden Eingangssignale voneinander.

Im Vergleich zu der AC Kopplung ist hier der Vorteil, dass auch sehr geringe Frequenzen passieren können. Um den Batteriespannungsbereich zuverlässig vom Nutzsignal zu trennen, muss U_{e1} oder U_{e2} so anpassbar sein, dass $U_{e1} = U_{e2}$ im statischen Zustand entspricht.

4.3.5 Programmable Gain Amplifier

In dieser Arbeit soll ein PGA zum Einsatz kommen. Der PGA kann mithilfe einer Kommunikationsschnittstelle programmiert werden und je nach Modell unterschiedliche diskrete Verstärkerstufen einstellen. In der Abbildung 4.7 ist zu sehen, dass für die Schaltung wenig Bauteile benötigt werden. Zur Kompensierung des Gleichanteils soll als kostengünstige Alternative des Rheostats der Gleichanteil über das Generieren einer Referenzspannung durch eine Pulsweiten Modulation (PWM) und einem RC Tiefpass Glättungsfilter erreicht werden. Die PWM wird am Mikrocontroller erzeugt, über den Tastgrad (engl. duty cycle) wird bestimmt, wie groß die erzeugte Referenzspannung sein soll.

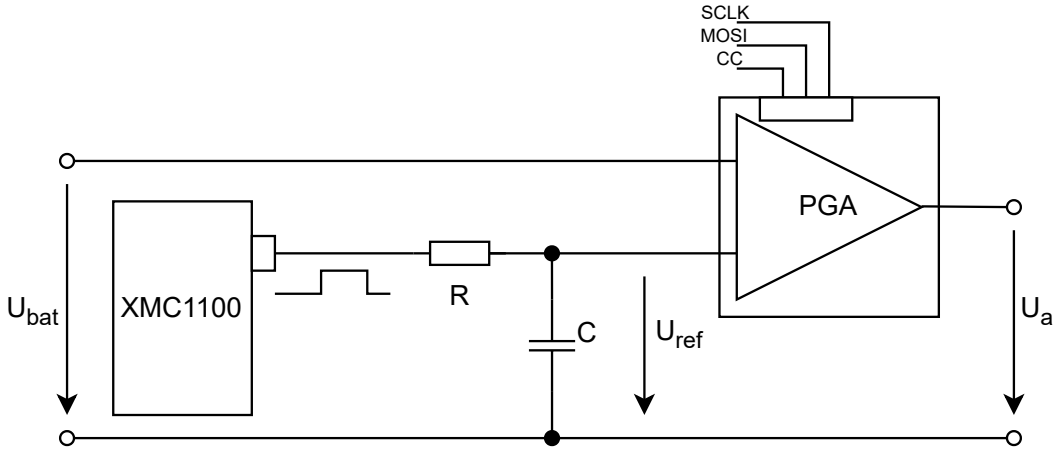


Abbildung 4.7: Die Abbildung zeigt einen PGA mit Referenzspannungsgenerator via PWM und einem Glättungsfilter. Der PGA arbeitet als Subtrahierer und kann gleichzeitig die Differenz beider Signale in diskreten Stufen verstärken.

Ein Nachteil bei der Verwendung eines PGA sind die nicht äquidistanten Verstärkerstufen. Hierdurch kann es vorkommen, dass der ideale Verstärkungsfaktor zwischen zwei Verstärkerstufen des PGA liegt und der ADC somit nicht optimal ausgesteuert ist.

Restwelligkeit

Die Referenzspannungserzeugung via PWM ist eine einfache Lösung, eine potenzielle Herausforderung ist jedoch die Restwelligkeit der so erzeugten Spannung. Auch nach dem Glätten weist das Signal eine Restwelligkeit auf. Vor allem bei hohen Verstärkungsfaktoren kann dies schon bei wenigen Millivolt den ADC in Sättigung treiben. Da es sich jedoch um einen technisch einfachen Ansatz handelt, soll überprüft werden, ob die PWM für diesen Einsatz geeignet ist. Die Restwelligkeit definiert sich über das Verhältnis vom Effektivwert des Wechselanteils zum Betrag des Gleichanteils wie folgt:

$$r_u = \frac{U_{\sim}}{|U_{-}|} \tag{4.6}$$

Die Restwelligkeit gibt das Verhältnis von Gleich- zu Wechselanteil im Signal an. Da in der Schaltung mit hohen Verstärkungsfaktoren gearbeitet wird, muss $r_u < 1\,\mathrm{mV}$ sein.

Praktisch wird sich zeigen, ob die Filterung mit dem Tiefpassverhalten des PGAs ausreichend ist, um ein hinreichend stabiles Signal mithilfe der PWM zu erzeugen. Alternativ kann die Referenzspannung auch über einen GPIO-Ausgang und einem variablen Spannungsteiler mithilfe eines Rheostats erzeugt werden. Hierfür hat der Autor in [42] einen einstellbaren Spannungsteiler verwendet. Der Spannungsteiler wurde mit einem Rheostat so ausgelegt, dass der Spannungsbereich der Batterie abgedeckt ist.

4.4 Auslegung: Sensorhardware

In dieser Arbeit soll ein System zur ladungsneutralen Einzelzellmessung mit zentraler Anregung und Impedanzberechnung entwickelt werden, wobei die Berechnung perspektivisch aufgrund des hohen Datenaufkommens dezentral erfolgt, dabei aber zentral gesteuert wird vgl. Abbildung 3.1.

4.4.1 Initialer Spannungsteiler

Wie aus Kapitel 4.2.1 hervorgeht, wird die Batteriezelle einen Gleichanteil zwischen 1,8 V und 4,2 V in die Messung bringen. Dieser Spannungsbereich übersteigt den Messbereich des Mikrocontrollers von $0\,\mathrm{V} - 3{,}3\,\mathrm{V}$. Um den Spannungsbereich wieder in den messbaren Bereich zu schieben, wird der Messschaltung ein konstanter Spannungsteiler vorgeschaltet. Nachteil dieser Lösung ist, dass auch die zu messende Amplitude reduziert wird. Ziel ist es, den Spannungsbereich der Batterie zu halbieren, daher werden die Widerstände $R1 = R2 = 10\,\mathrm{k\Omega}$ zunächst gleich gewählt. Im späteren Verlauf kann eine Anpassung aufgrund des Innenwiderstands der nachgeschalteten Schaltung notwendig sein.

4.4.2 Kompensierung des Gleichanteils

Die Kompensierung des Gleichanteils soll mithilfe der in Kapitel 4.3.5 beschriebenen Schaltung aus Abbildung 4.7 umgesetzt werden. Hierfür muss eine Referenzspannung erzeugt werden. Der Ansatz über die PWM soll geprüft werden und dem variablen Spannungsteiler aus [42] gegenübergestellt werden.

Rheostat

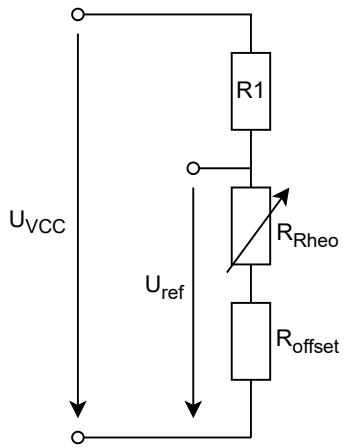


Abbildung 4.8: Die Abbildung zeigt den Aufbau des Spannungsteilers. Durch den Verschiebungswiderstand R_{offset} wird der einstellbare Spannungsbereich verkleinert, was die Auflösung des Rheostats erhöht.

Für die Lösung mit dem Rheostat wird die Ausgangsspannung eines GPIO-Pins des Mikrocontrollers verwendet. Durch den einstellbaren Widerstand des Rheostats kann die Referenzspannung am PGA variiert werden. Um den für die Messung relevanten Spannungsbereich möglichst hoch aufzulösen, wird der Spannungsteiler mit dem Rheostat neben dem Rheostat aus zwei weiteren Widerständen bestehen, zum einen aus dem Teilerwiderstand $R1$ und zum anderen aus einem Verschiebungswiderstand R_{offset}. Die Abbildung 4.8 zeigt den Aufbau des Spannungsteilers. Durch den Verschiebungswiderstand R_{offset} wird der einstellbare Spannungsbereich auf 1,2 V...2,8 V begrenzt. Dieser Bereich kann in 1024 Stufen eingestellt werden. Die so einstellbaren Spannungen sind in Abbildung 4.9 zu sehen.

$$Uref = U_{vcc} \cdot \frac{R2}{R1 + R2} \tag{4.7}$$

mit

$$R2 = R_{Rheo} + R_{offset} \tag{4.8}$$

ergibt sich:

$$U_{ref} = U_{vcc} \cdot \frac{R_{Rheo} + R_{offset}}{R1 + R_{Rheo} + R_{offset}} \tag{4.9}$$

Der Widerstand $R1$ wird auf $R1 = 20k\Omega$ festgelegt. Um den Widerstand R_{offset} zu ermitteln, wird $R_{Rheo} = 0\,\Omega$ gesetzt und $U_{Ref} = 1{,}2\,\mathrm{V}$ gewählt. Damit liegt die minimale Spannung unterhalb der zu erwartenden Batteriespannung.

$$R_{offset} = \frac{-U_{Ref} \cdot R1}{U_{Ref} - U_{vcc}} \tag{4.10}$$

Eingesetzt:

$$R_{offset} = \frac{-1{,}2\,\mathrm{V} \cdot 20\,\mathrm{k\Omega}}{1{,}2\,\mathrm{V} - 3{,}3\,\mathrm{V}} \approx 11{,}5\,\mathrm{k\Omega} \tag{4.11}$$

Der Widerstand R_{offset} aus Gleichung 4.11 wird nach der E12 Reihe auf $R_{offset} = 12\,\mathrm{k\Omega}$ gesetzt. Damit ergibt sich nach Gleichung 4.9 mit $R_{Rheo} = 0\,\Omega$ eine minimale Referenzspannung von $U_{Ref} = 1{,}24\,\mathrm{V}$. Der Rheostat kann auf bis zu $100\,\mathrm{k\Omega}$ geschaltet werden, was nach Gleichung 4.9 mit $R_{Rheo} = 100\,\mathrm{k\Omega}$ eine maximale Referenzspannung $U_{Ref} = 2{,}8\,\mathrm{V}$ ergibt.

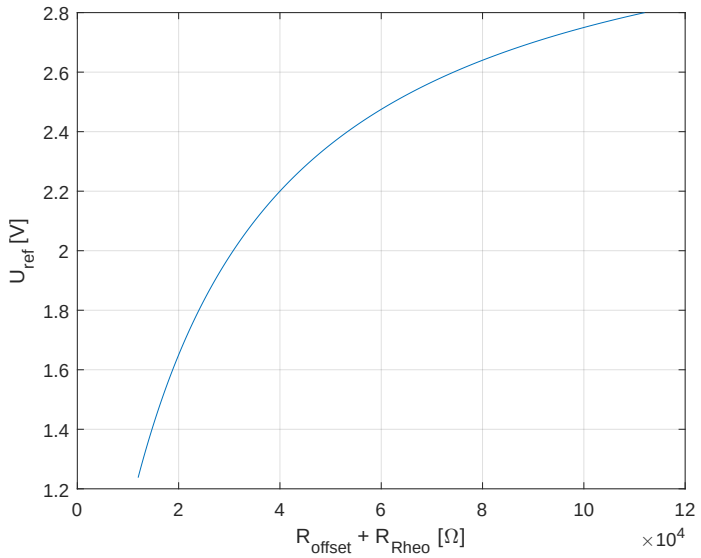


Abbildung 4.9: Die Abbildung zeigt die durch den Rheostat theoretisch einstellbare Referenzspannung.

Die Abbildung 4.9 zeigt, dass mit dem Einstellungsbereich 1,6 V abgedeckt werden. Mit dem initialen Spannungsteiler kann damit jedes für Batterien notwendige Spannungslevel erzeugt werden.

Pulsweiten-Modulation und Filterung

Grundsätzlich wurde schon in 4.3.5 das Problem der Restwelligkeit beschrieben. Um die Restwelligkeit möglichst gering zu halten, gibt es bei der Filterung unterschiedliche Stellmöglichkeiten. Um die Restwelligkeit gegenüber dem Mittelwert klein zu halten, muss die Zeitkonstante τ hinreichend groß gewählt werden [54]. Weiterhin muss $f_g \ll f$ gelten. Die Gleichungen 4.12 und 4.13 zeigen, wie τ und f_g zusammenhängen.

$$\tau = R \cdot C \tag{4.12}$$

$$f_g = \frac{1}{2 \cdot pi \cdot \tau} \tag{4.13}$$

Um dem Kriterium $f_g \ll f$ gerecht zu werden, wird f_g auf $f_g = 330\,\mathrm{Hz}$ festgelegt. Der Filterwiderstand wird auf $R = 10\,\mathrm{k\Omega}$ festgelegt. Damit ergibt sich ein Kapazitätswert von $C = 48{,}82\,\mathrm{nF}$.

Die so erzeugte Gleichspannung ist über den Tastgrad einstellbar und kann als Referenzspannung an dem PGA verwendet werden. Die Restwelligkeit des Signals beträgt in der Simulation $r_u < 1\,\mathrm{mV}$.

4.4.3 Signalverstärkung

Bei einer Zielgenauigkeit von ca. 1% und einer Signalamplitude von $V_{pp} \leq 1mV$ ist eine Verstärkung des Messsignals notwendig, um eine ausreichende Auflösung des Signals zu erreichen. Für die Auflösung der Signalamplitude sind die Anzahl der ausgesteuerten Quantisierungsstufen n ausschlaggebend. Bei einer Signalamplitude V_{pp} ergibt sich die minimale Anzahl notwendiger Quantisierungsstufen n_{min}, wie in Gleichung 4.14 gezeigt.

$$n_{min} \geq \frac{V_{pp}}{100} \tag{4.14}$$

Durch den geringen Innenwiderstand R_i der Batterie in Verbindung mit dem initialen Spannungsteiler aus Kapitel 4.4.1 ist eine Amplitude von unter 1 mV zu erwarten. Demnach muss der PGA Verstärkungen von mehr als 200 zulassen. Dies führt aufgrund der

beschränkten Auswahl PGAs im Markt dazu, dass eine zweite Verstärkerstufe vorgesehen werden muss. Zunächst für die zweite Stufe als Referenzspannung des Verstärkers $V_{cc}/2$ vorgesehen werden, da der Gleichanteil in der ersten Stufe vollständig kompensiert sein soll. Wird dies nicht erreicht, durch z. B. das Übertragungsverhalten des PGAs, kann eine weitere Filterschaltung mit PWM hinzu geschaltet werden.

4.5 Validierung des Messdatensatzes und Sättigungskriteriums

Die Größe der Messamplitude ist zum Zeitpunkt der Messung unbekannt. Daher ist ein festgelegter Verstärkungsfaktor nicht möglich. Das Signal muss aber, wie Gleichung 4.14 zeigt, eine Mindestauflösung erreichen, um dem Genauigkeitsanspruch der Messung zu genügen. Durch Rauschanteile des Signals oder einem nicht optimal kompensierten Gleichanteil ist vor der Bestimmung der Amplitude nicht eindeutig zu sagen, ob das Signal ausreichend ausgesteuert ist. Um eine möglichst große Messauflösung unabhängig von der anliegenden Amplitude zu ermöglichen, soll das Messsignal so weit wie möglich verstärkt werden, ohne dabei den ADC in Sättigung zu treiben. Hierfür werden im folgenden Abschnitt mögliche Methoden thematisiert.

4.5.1 Sättigungskriterium

Das Sättigungskriterium ist für die Regelung der Signalaussteuerung von zentraler Bedeutung. Um festzustellen, ob sich das Signal in Sättigung befindet, können unterschiedliche Methoden verwendet werden, z. B:

- Bestimmung von Minimal- und Maximalwerten
- Berechnung der Datenänderung - Ableitung
- Berechnung des Klirrfaktors
- Auswerten des Histogramms

Minimal- und Maximalwert

Ein Sättigungskriterium ist das Vorhandensein von Minimal- und Maximalwerten in einem Datensatz. Hierfür wird jeder Messwert überprüft, sobald sich ein Messwert im Minimum oder Maximum befindet, muss der Datensatz verworfen werden. Hierbei kann jedoch nicht zwischen Rauschanteil und Signal unterschieden werden. Bei einer sehr kleinen Messamplitude ist das Singal-zu-Rauschverhältnis unter Umständen initial sogar kleiner eins. Bei einer hohen Verstärkung kann dies dazu führen, dass sich das Signal noch nicht in Sättigung befindet, wohingegen die Rauschanteile das Sättigungskriterium schon erfüllen und der Datensatz verworfen wird.

Datenänderung - Ableitung

Über die Ableitung zum Zeitpunkt t kann bestimmt werden, ob sich ein Signal in Sättigung befindet. Diese Methode ist bei einem rauschfreien Signal ein zuverlässiger Ansatz, wenn man sinusförmige Schwingungen betrachtet. Gleichung 4.15 zeigt das Verhalten der Ableitung eines rauschfreien Sinussignals. Ist das Signal in Sättigung, ergibt sich die Ableitung zu null.

$$sin(t)\frac{d}{dt} = \begin{cases} 0 & \text{wenn Signal in Sättigung} \\ cos(t) & \text{sonst} \end{cases} \tag{4.15}$$

Je nach Rauschverhalten des Signals wird die Interpretation jedoch schwieriger. Wie die Abbildung 4.10 zeigt, wird die Ableitung mit abnehmendem SNR weniger aussagekräftig. Um bei geringem SNR ein Sättigungskriterium zu bilden, muss zusätzlicher Speicher verwendet werden, das heißt, eine Betrachtung über einen längeren Zeitraum. Da in diesem Fall die Ableitungswerte aufgrund des Rauschens keine konstante Steigung von null annehmen. In besonderen Fällen ist es denkbar, dass ein sich in Sättigung befindliches Signal niemals zwei gleiche aufeinander folgende Werte aufweist.

Klirrfaktor

Der Klirrfaktor gibt an, wie groß der Anteil der Oberschwingungen eines Signals zum Gesamtsignal ist. Diese Methode erfordert das Wissen über die Amplitude der harmonischen Frequenzen der Grundschwingung. Daher ist eine Fourieranalyse des Signals notwendig.

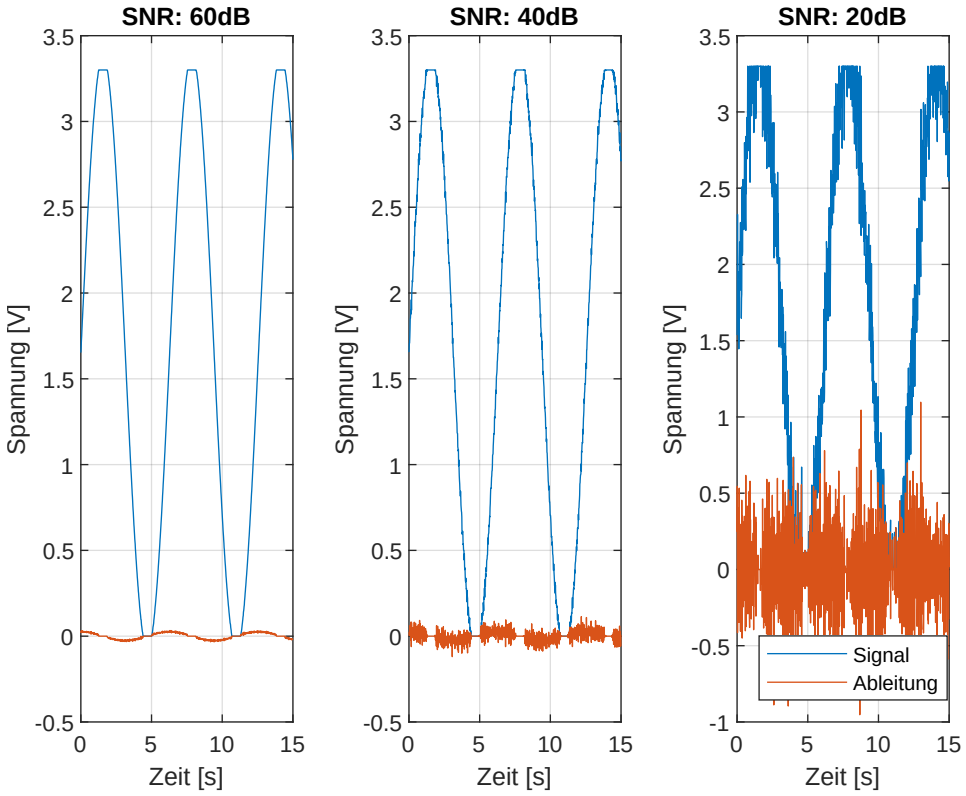


Abbildung 4.10: Die Abbildung zeigt gesättigte Signale und deren Ableitung. Befindet sich das Signal in Sättigung, zeigt die Ableitung dies an. Bei abnehmendem SNR und einer Teilsättigung ist die Ableitung zunächst kein geeignetes Mittel als Sättigungskriterium.

Mithilfe der Fourieranalyse können die Amplituden der Grund- und Oberschwingungen bestimmt werden. Anschließend kann über die Gleichung 4.16 der Klirrfaktor k bestimmt werden. Wobei f der Grundfrequenz entspricht.

$$k = \sqrt{\frac{U_{2f}^2 + U_{3f}^2 + ... + U_{nf}^2}{U_{1f}^2 + U_{2f}^2 + U_{3f}^2 + ... + U_{nf}^2}} \tag{4.16}$$

In Abbildung 4.11 ist das Spektrum eines gesättigten Signals zu sehen. Deutlich zu erkennen ist, dass die Amplitude des Signals sich auf die durch die Sättigung entstandenen Oberwellen aufteilt und die Amplitude der Grundschwingung nicht mehr die des ursprünglichen Signals darstellt. Wenn $k > 0$ ist, dann befindet sich das Signal in Sättigung. Hierbei gilt zu beachten, dass dies nur für rauschfreie Signale gilt.

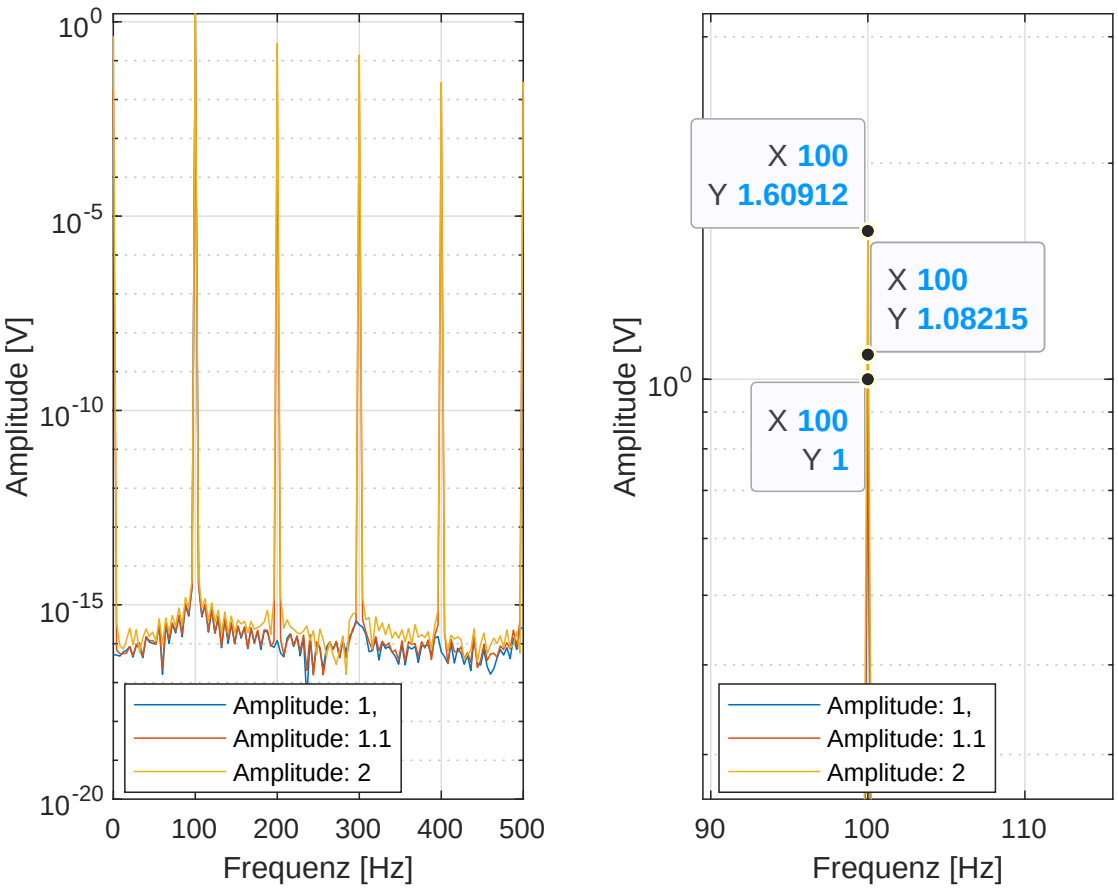


Abbildung 4.11: Die Abbildung zeigt links die entstehenden Oberwellen durch die Sättigung des Signals. Rechts ist der Fehler der Amplituden der Grundschwingung zu erkennen. Umso größer der Anteil des Signals in Sättigung ist, desto größer die Abweichung der Amplitude.

Die Abbildung 4.12 zeigt den Klirrfaktor. Sobald die Signalamplitude größer eins ist, befindet sich das Signal im Sättigungsbereich. Die Abbildung zeigt bis dahin einen konstanten Klirrfaktor von null. Sobald die Signalamplitude größer eins wird, steigt der Klirrfaktor an. Das Signal befindet sich in Sättigung.

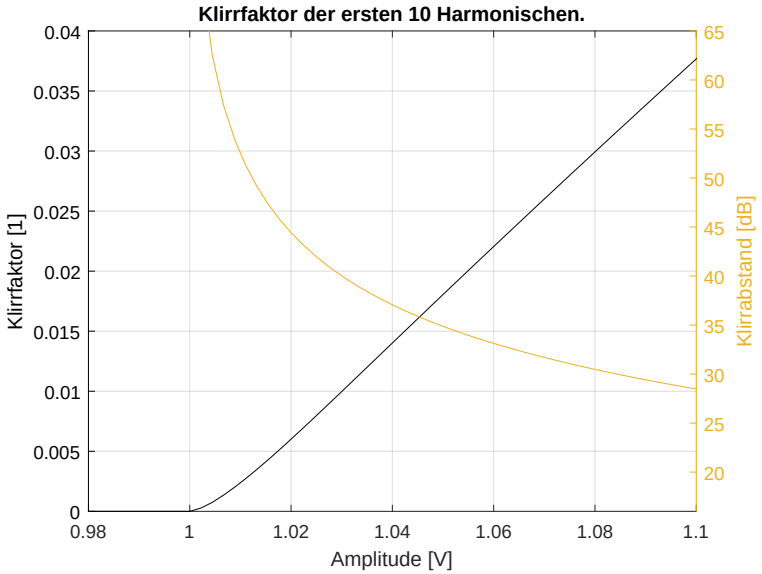


Abbildung 4.12: Die Abbildung zeigt den Klirrfaktor. Es ist zu erkennen, dass der Klirrfaktor bei einer Amplitude unter eins bei null liegt. Sobald sich das Signal in Sättigung befindet, steigt der Wert an.

Für ein ideales Signal, das sich nicht in Sättigung befindet, liegt der Erwartungswert bei $k = 0$.

Histogramm

Ein Histogramm gibt die Wertverteilung in einem Bereich an. Der Ansatz dieser Methode ist, die Quantisierungsstufen des ADCs in Bereiche aufzuteilen und diese ins Verhältnis zum Gesamtsignal zu setzten. Damit soll erreicht werden, dass sich in Sättigung befindliche Rauschanteile des Gesamtsignals nicht zum Verwerfen des Datensatzes führt. Weiterhin ist diese Methode weniger rechenintensiv als z. B. die Berechnung des Klirrfaktors. Allgemein wird bei einem Histogramm ein Wertebereich in k Klassen aufgeteilt. Üblicherweise sind die Klassen des Histogramms gleich groß, dies ist jedoch keine Notwendigkeit. In Abbildung 4.13 ist die Werteverteilung eines periodengenau abgetasteten Sinus zu sehen. Die Werteverteilung eines Sinus ist charakteristisch. Teilt man das Histogramm in der Mitte und bildet das Verhältnis zueinander, ergibt sich dies immer zu eins, auch wenn die Phase des Eingangssignals verschoben wird, wie in Abbildung 4.13 (rechts) zu sehen ist.

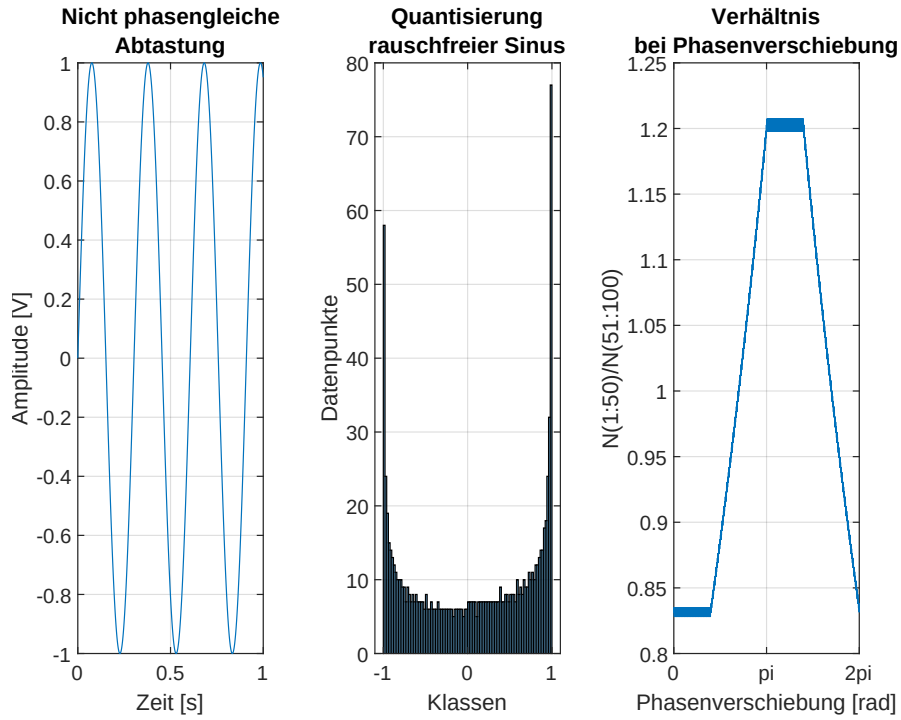


Abbildung 4.13: Die Abbildung zeigt einen periodengenauen Sinus (links), die Werteverteilung des Eingangssignals als Histogramm (mittig) und den Einfluss der Phase auf das mittlere Verhältnis (rechts). Die Abweichungen von eins sind auf numerische Einflüsse zurückzuführen.

Die Abbildung 4.14 zeigt: Wird das Signal nicht periodengenau abgetastet, entsteht ein Ungleichgewicht im Histogramm und auch das Verhältnis um null verschiebt sich in Abhängigkeit der Phase.

Für eine periodengenaue Messung wird in die verfügbaren Speicherstellen n die Anzahl p Perioden geschrieben. Daraus ergibt sich der in Gleichung 4.17 bzw. 4.18 gezeigter Zusammenhang mit der Periode T und der Abtastperiode T_s

$$T_s = \frac{p \cdot T}{n} \tag{4.17}$$

für die Abtastfrequenz ergibt sich:

$$f_s = \frac{n \cdot f}{p} \tag{4.18}$$

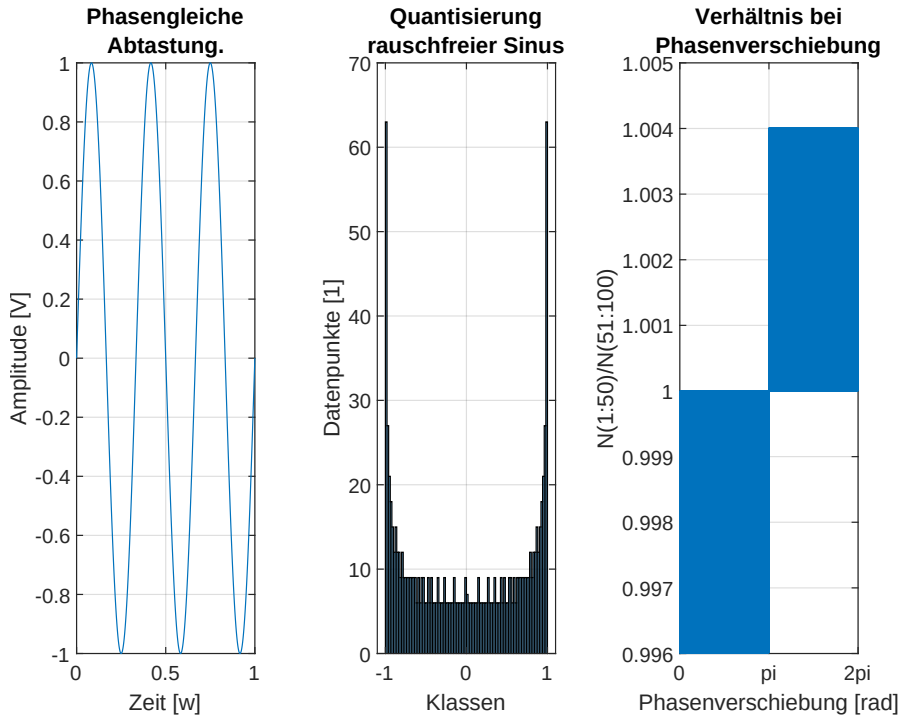


Abbildung 4.14: Die Abbildung zeigt (links) den nicht periodengenau abgetasteten Sinus, die Werteverteilung des Eingangssignals als Histogramm (mittig) und den Einfluss der Phase auf das mittlere Verhältnis (rechts).

Der Ansatz, um abzuschätzen, ob sich das Signal oder nur der Rauschanteil in Sättigung befindet, geht über die Anzahl der Messpunkte in den Randbereichen. Die Summe aller Datenpunkte in einem Histogramm ist in Gleichung 4.19 definiert. Wobei k die Anzahl der Klassen ist und m_i die Anzahl der Datenpunkte innerhalb einer Klasse.

$$n = \sum_{i=0}^{k} m_i \tag{4.19}$$

Mithilfe der Gleichung 4.19 lässt sich ein Verhältnis für den Schwellwert N_h als Sättigungskriterium beschreiben. Die Gleichung 4.20 beschreibt die Definition des Schwellwerts N_h für den unteren Sättigungsbereich.

$$N_{h,0} = \frac{m_0}{\sum_{i=0}^{k} m_i} = \frac{m_0}{n} \tag{4.20}$$

Analog zu der Gleichung 4.20 wird für den oberen Schwellwert die Gleichung 4.21 definiert.

$$N_{h,k} = \frac{m_k}{\sum_{i=0}^{k} m_i} = \frac{m_k}{n} \tag{4.21}$$

Rauschtoleranz

Die Gleichungen 4.21 und 4.20 definieren die Berechnung des Schwellwerts. Die eigentliche Grenze, wann sich das Signal in Sättigung befindet oder nicht, wird zunächst mithilfe von der MATLAB-Software ermittelt. Die Abbildung A.2 im Anhang zeigt, dass sich bei einem Randbereich von jeweils 10% und starkem Rauschen ca. 30% der Messpunkte außerhalb der festgelegten Grenzen befinden. Um bestimmen zu können, ob sich das Signal im oberen oder unteren Sättigungsbereich befindet, wird der Schwellwert auf die jeweiligen Bereiche aufgeteilt. Mit Erkenntnis aus der Simulation wird N_h initial auf $N_{h,0} = N_{h,k} = 0,15$ festgelegt.

Periodengenaue Messung

Um eine periodengenaue Messung zu gewährleisten, muss die Abtastfrequenz f_s mit der Frequenz des Signals f in Abhängigkeit des verfügbaren Speichers gekoppelt werden.

- Perioden: p
- Speicherstellen: l
- Frequenz: f
- Abtastfreqenz: f_s

Mithilfe der Anzahl der Perioden und der verfügbaren Speicherstellen ergibt sich die Abtastfrequenz f_s.

$$f_s = \frac{l \cdot f}{p} \tag{4.22}$$

Mit dem Nyquist-Shannon-Abtasttheorem gilt außerdem [2]:

$$f_s > 2 \cdot f$$

4.5.2 Messdatensatzvalidierung

Die Messung besteht aus mehreren Messungen für jede Frequenz. Für die Validierung der Messungen wird über die Verteilung der Messwerte über den Messbereich der Schwerpunkt des Datensatzes bestimmt. Der Schwerpunkt wird mithilfe des Histogramms der Phasenmessung ermittelt. Hierfür wird das Histogramm in $k = 3600$ Klassen unterteilt. Es ergeben sich zehn Klassen pro Grad. Unter der Annahme, dass sich eine Häufung der Messwerte in dem richtigen Bereich befinden, wird der Schwerpunkt des Histogramms bestimmt. Hierbei handelt es sich um die Klasse, in welche die meisten Messpunkte fallen. In der Messung werden die Ergebnisse um den Schwerpunkt herum berücksichtigt. Bei einer angestrebten Genauigkeit von 1° werden demnach zehn Klassen berücksichtigt.

4.6 Synchronisierung der Messung

Für eine gezielte Bestimmung der Impedanz müssen alle Sensoren gleichzeitig abtasten. Für den Laboraufbau wird zunächst nur ein Sensor für die Spannungsmessung und ein Sensor für die Strommessung verwendet. In einem größeren Kontext wird für jede Lithium-Ionen-Zelle oder Zellverbund ein Sensor für die Spannungsmessung benötigt, die Strommessung erfolgt zentral. Um eine Synchronisierung zwischen allen Sensoren zu erreichen, wird hier zunächst auf eine Kabelverbindung gesetzt. In dieser Arbeit wird zunächst über ein zentrales Triggersignal Signal die Abtastfrequenz der einzelnen Sensoren vorgegeben. Dies soll perspektivisch zentral von einem übergeordneten Modulcontroller oder dem BMS übernommen werden. Durch den zentralen Ansatz ergibt sich der Vorteil, dass die Abtastfrequenz aus Sicht des einzelnen Zellsensors extern vorgegeben wird. Damit wird eine variable Einstellung der Abtastrate auf dem Mikrocontroller und eine aufwendige Synchronisierung zwischen den einzelnen Mikrocontrollern vermieden.

4.7 Auslegung: Software

Die Software besteht aus zwei Teilen. Teil eins umfasst Steuerung und die Regelung, welche in der MATLAB-Software umgesetzt werden soll. Teil zwei der Software befindet sich auf dem Mikrocontroller. Dieser ist zunächst nur für die Weiterleitung der Stellgrößen sowie das Messen des Signals zuständig, perspektivisch soll die Berechnung der Frequenz und die Regelung der digitalen Bausteine auf dem Mikrocontroller umgesetzt werden.

4.7.1 Kommunikationsschnittstelle

Die Kommunikation zwischen der MATLAB-Software und dem Mikrocontroller findet über UART statt, dafür wird ein einfaches Datentelegramm entwickelt. Das Telegramm muss ermöglichen, folgende Funktionen abzubilden:

- Messdatensatz anfordern
- Verstärkung setzen
- Tastgrad setzen
- Ausführung bestätigen

Der Datenteil kann je nach Steuerbefehl auch leer sein. Mit der Bitbreite von vier Bit für die Steuerung ist es möglich $2^4 = 16$ Steuerbefehle zu definieren. Das Datenfeld muss mindestens sieben Bit breit sein, da schon für die Einstellung des Tastgrads der PWM eine Zahlendarstellung von $1 - 100$ möglich sein muss. Weiterhin wird ein Vorzeichenbit vorgesehen. Da UART nur Vielfache von acht Bit überträgt, wird die Breite des Telegramms auf 16 Bit festgelegt. Damit wird genug Flexibilität ermöglicht, um später ggf. auch größere Datenwerte zu übertragen. Es ergibt sich der in Abbildung 4.15 zu sehende Aufbau, je nach Anwendung mit oder ohne Vorzeichenbit.

Die Übertragung erfolgen mit einem USB Kabel zwischen Mikrocontroller und PC.

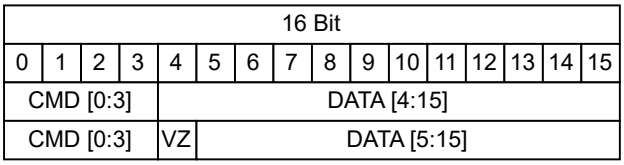


Abbildung 4.15: Das Telegramm hat 16-Bit-Breite, die unteren vier Bits werden als Kommandobits verwendet. Die folgenden Bits sind den Daten vorbehalten. Optional kann das Datenfeld um ein Bit reduziert werden, um das fünfte Bit als Vorzeichenbit zu nutzen.

4.7.2 Regelung und Signalverarbeitung

Die Implementierung in MATLAB bildet den Großteil der Funktionalität des Systems ab. Neben der Auswertung der Messergebnisse ist die Kompensierung des Gleichanteils sowie die Regelung der Signalaussteuerung Aufgabe der Implementierung in MATLAB. Es wird ein objektorientierter Ansatz verfolgt.

Abstraktion

Für jeden Funktionskontext wird eine Klasse in der MATLAB-Software erstellt. Enumerationen sollen helfen, die Lesbarkeit des Quellcodes zu verbessern und nachvollziehbar zu gestalten. Hierfür ergibt sich der in Abbildung 4.16 zu sehende Zusammenhang der Klassen. Unter Funktionskontext werden die folgenden Bereiche verstanden:

- Sensor (Spannung oder Strom)
- Regler
- Messautomation
- (Auswertung)

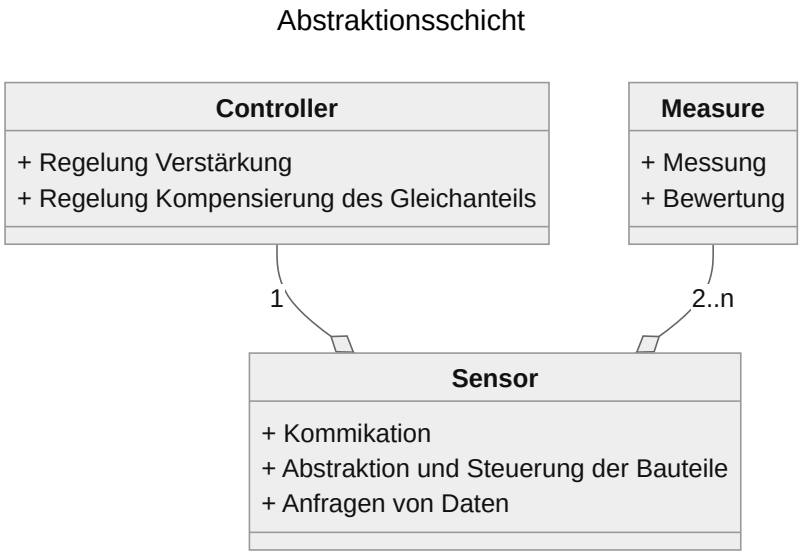


Abbildung 4.16: Die Abbildung zeigt den Zusammenhang zwischen den einzelnen Klassen zur Abstraktion der Funktionen des Gesamtsystems. Die Controllerinstanz regelt jeweils einen Sensor. Die Messung muss von mindestens zwei Sensoren durchgeführt werden.

Kommunikation

Die Kommunikation zwischen der MATLAB-Software und dem Mikrocontroller geht von der MATLAB-Instanz aus. Die MATLAB-Software sendet Befehle an den Mikrocontroller XMC1100, welcher diese ausführt und nach Ausführung eine Bestätigung zurückschickt. Die MATLAB-Software muss die Befehle in einer für die Kommunikation zuständigen Klasse entgegennehmen und diesen in das Übertragungstelegramm verpacken. Das so generierte Datenpaket wird via UART an den Mikrocontroller übertragen. Die Abbildung 4.17 zeigt den allgemeinen Ablauf der Kommunikation zwischen dem Mikrocontroller und der MATLAB-Software. Nach der Ausführung des Befehls bestätigt der Mikrocontroller die Ausführung an die MATLAB-Software, damit ist sichergestellt, dass die Software den weiteren Programmablauf blockiert, bis das Kommando ausgeführt wurde.

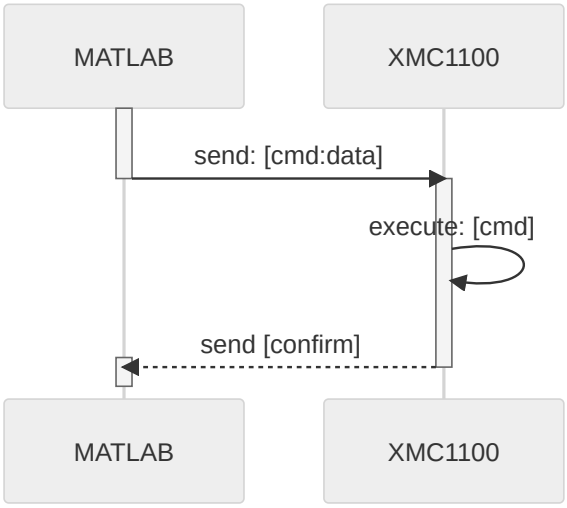


Abbildung 4.17: Die Abbildung zeigt die Kommunikation zwischen der MATLAB-Software und dem Mikrocontroller XMC1100. Durch die Bestätigung der Ausführung des Mikrocontrollers XMC1100 wird sichergestellt, dass die Software blockiert.

Regelung

Die Abbildung 4.18 zeigt den Ablauf der Regelung. Das Signal wird solange verstärkt, bis es sich oben, unten oder oben und unten in Sättigung befindet. Für die ersten beiden Fälle muss die Referenzspannung nach gesteuert werden. Befindet sich das Signal oben und unten in Sättigung, ist der maximale Verstärkungsfaktor $k_{max} = k - 1$ gefunden. Die Verstärkung wird demnach um einen Schritt reduziert und der Vorgang beendet.

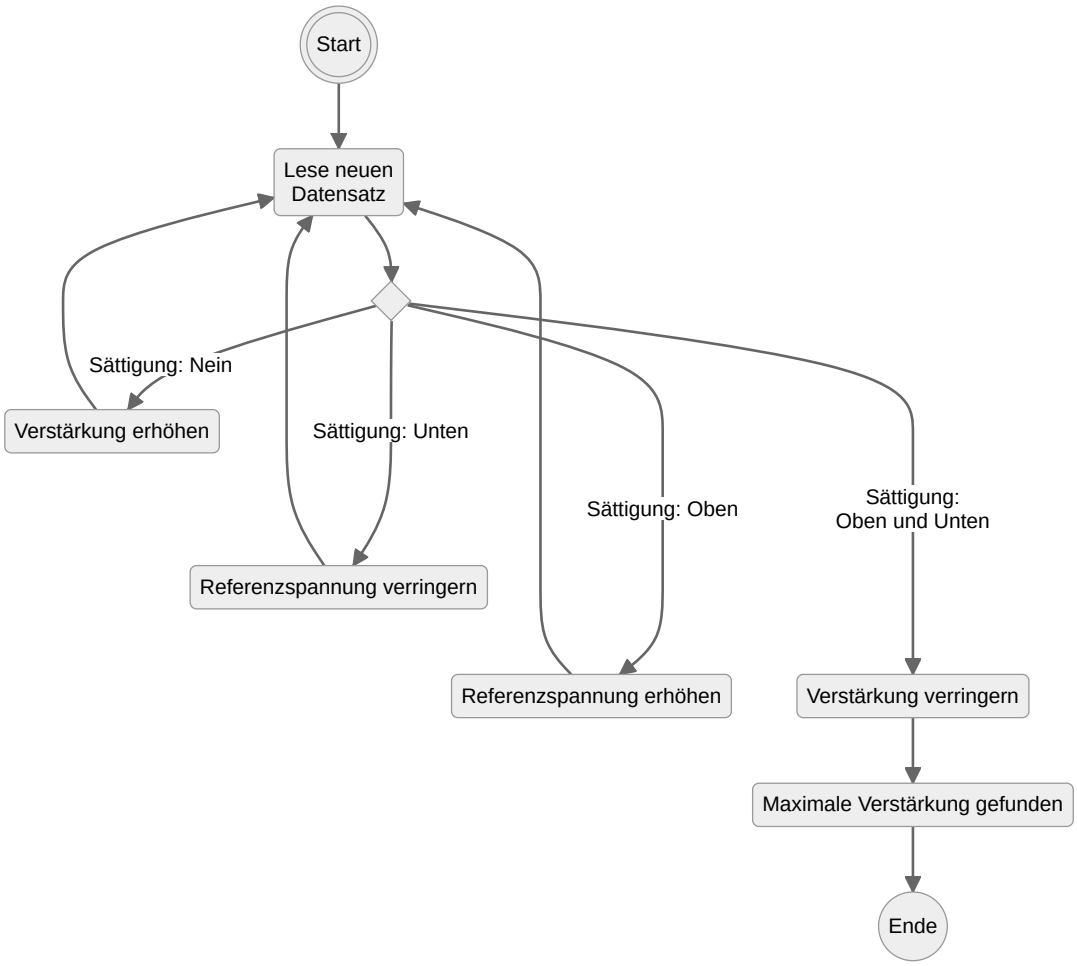


Abbildung 4.18: Die Abbildung zeigt das Vorgehen beim Ausregeln der Referenzspannung und dem Finden der maximalen Verstärkung bei unbekannter Signalamplitude.

Messung

Die Messungen sollen automatisiert ablaufen, hierfür muss die MATLAB-Software die Messungen steuern und die Daten verarbeiten bzw. für eine weitere Verarbeitung speichern. Für die Signalgenerierung ist eine Stromanregung vorgesehen, welche durch einen Signalgenerator gesteuert wird. Der Signalgenerator AFG1061 von Tektronix [51] kann über eine USB-Schnittstelle angesteuert werden. Über denselben Signalgenerator wird auch die Abtastfrequenz auf einem zweiten Kanal des Geräts automatisiert und pas-

send zur Anregungsfrequenz eingestellt. Eine Implementierung der Schnittstelle zu dem AFG1061 in MATLAB ist in der Arbeitsgruppe schon vorhanden. Die Kommunikation erfolgt in der Backus-Naur-Form vlg. [50].

4.7.3 XMC1100-Software

Der Mikrocontroller ist für den Laboraufbau zunächst als Schnittstelle zu den digitalen Bauteilen sowie für Aufnahme der Messwerte verantwortlich. Hierfür müssen die von der Implementierung in MATLAB kommenden Befehle ausgewertet und der entsprechenden Funktion zugeführt werden.

Die Software des Mikrocontrollers hat folgende Aufgaben:

- Kommunikation
- Datenerfassung
- Einstellung des Tastgrads der Referenzspannung
- Ansteuerung digitaler Bausteine
  - PGA(s)
  - Rheostat
- Steuerung der Messung

Kommunikation

Die Software des Mikrocontrollers soll die von der Implenemntierung in MATLAB eingehenden Steuerbefehle umsetzen. Der Mikrocontroller muss die über UART übertragenen Daten wieder aufteilen und den Kontrollbefehl von den Nutzdaten separieren. Weiterhin muss die MATLAB-Software solange blockieren, bis der Befehl auf dem Mikrocontroller ausgeführt wurde, um einen zeitlich konsistenten Ablauf zu garantieren. Der Empfang von Daten via UART wird mithilfe der *DAvE* App API umgesetzt. Es müssen, wie in Abbildung 4.15 zu sehen, 16-Bit empfangen werden. Die Kommunikationsschnittstelle auf dem Mikrocontroller arbeitet wie in Abbildung 4.19 zu sehen als eine Art Multiplexer. Auf Basis des separierten Kommandos werden die Daten an die entsprechende Funktion weitergeleitet und verarbeitet.

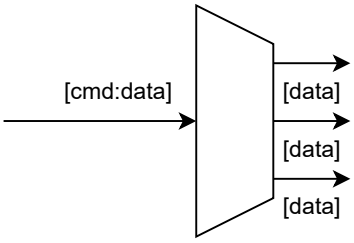


Abbildung 4.19: Die Kommunikationsschnittstelle decodiert das Datentelegramm und leitet die Daten an die entsprechende Funktion weiter.

Datenerfassung

Die Datenerfassung soll über ein externes Signal ausgelöst werden, dass externe Signal gibt damit auch die Abtastfrequenz vor. Intern löst das Signal über die Event Request Unit (ERU) einen Interrupt Service Request (ISR) aus. Die ERU ist eine Infineon eigene Lösung für das Verarbeiten von externen und internen Events. Ein Event kann z.B das Wechseln eines Signalpegels an einem GPIO-Pin sein. Der ISR liest nach jeden von der ERU erfasstem Event den ADC aus und speichert den Wert in einen Puffer. Dieser kann auf Anfrage von der MATLAB-Software abgerufen werden.

Steuerung digitaler Bausteine

Für die Verwendung digitaler Bausteine soll eine Abstraktionsschicht implementiert werden, welche die Funktionalitäten der Bausteine durch Software abbildet. Die Abstraktionsschicht soll die Kommunikation zu dem digitalen Baustein ermöglichen, wenn mehr als ein Baustein derselben Art vorhanden ist, muss eine Unterscheidung möglich sein. Ggf. müssen Lookup-Tabellen für die einstellbaren Werte erzeugt werden. Die Kommunikation mit den digitalen Bausteinen soll über einen geräteinternen Bus wie I^2C oder SPI umgesetzt werden. Aufgrund der beschränkten Anzahl an verfügbaren GPIO-Pins ist SPI zu bevorzugen, da der Rheostat aus [42], welcher auch in dieser Arbeit verwendet werden soll, über eine SPI-Schnittstelle verfügt.

- Nutzerschnittstelle
- Zuordnung der Bauteile
- Übergabe an die Kommunikationsschicht

5 Entwicklung der Hardware des Zellsensors

Im folgenden Kapitel wird die Entwicklung der Hardware für den eigentlichen Zellsensor beschrieben. Grundlage hierfür ist die Konzeptionsphase. Das Design soll die Strom- und Spannungsmessung im Laboraufbau ermöglichen. Weiterhin sollen die im Konzept vorgestellten Lösungen zur Gleichanteilkompensierung im Design berücksichtigt werden. Hierfür werden AC-Kopplung, PGA und Rheostat sowie PGA und PWM auf der Platine vorgesehen. Die verschiedenen Lösungen sollen mithilfe von steckbaren Kurzschlussbrücken zu- und abgeschaltet werden. Im Anhang ist der vollständige Schaltplan (A.2.1) zu finden.

5.1 Bauteilauswahl

Für das Entwickeln der Platine müssen die digitalen Bauteile vorweg ausgewählt werden. Hierfür wurde anhand von festgelegten Kriterien eine Auswahl getroffen. Eine gemeinsame Anforderung ist die Möglichkeit, mit dem Bauteil via SPI zu kommunizieren.

Programmable Gain Amplifier

Hersteller	**Modell**	**Gain**	**Bus**
Texas Instruments	PGA280 [25]	0.125/128	SPI
Analog Devices	MAX9939 [38]	0.125/157	SPI
Analog Devices	LTC6915 [11]	1/4096	SPI

Tabelle 5.1: Eine Auswahl an PGAs

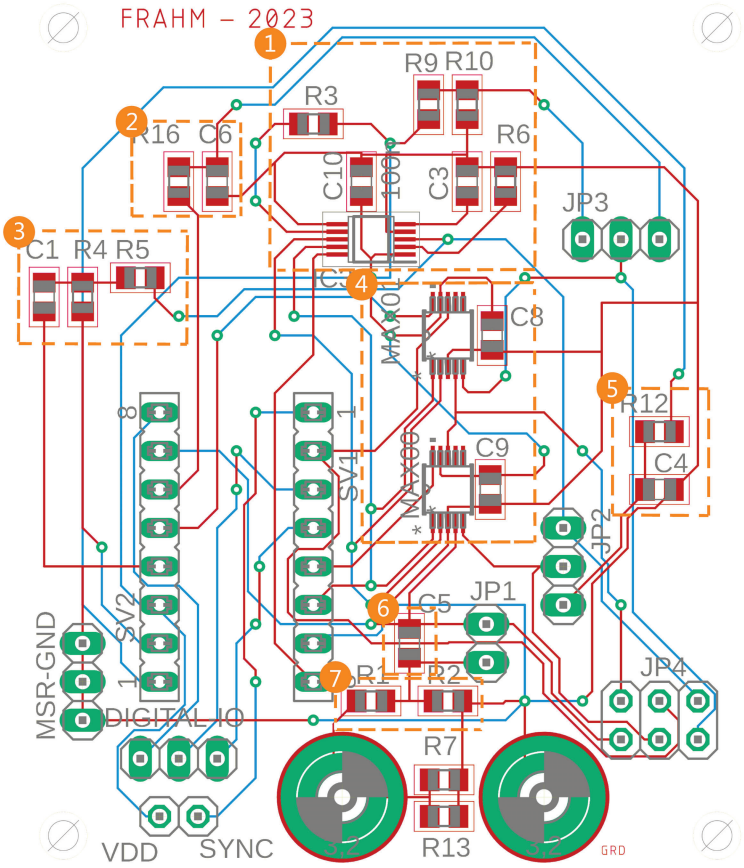


Abbildung 5.1: Die Abbildung zeigt die Platine eines Strom- oder Spannungssensors. 1) Rheostat und Beschaltung, 2) Spannungsteiler, 3) TP-Glättungsfilter, 4) MAX9939, 5) TP-Glättungsfilter, 6) AC-Kopplung (optional), 7) initialer Spannungsteiler.

In der Tabelle 5.1 sind verschiedene PGAs gelistet. Hier ist vor allem der maximale Verstärkungsfaktor wesentlich. Dies ist aufgrund der geringen Signalamplitude eine Notwendigkeit. Bei $V_{pp} < 1\,\text{mV}$ ist eine Verstärkung von über 3000 möglich. Vergleicht man die Verstärkungsfaktoren der Verstärker LTC6915 und dem MAX9939, dann ist eindeutig, dass der MTC6915 deutlich höher verstärken kann. Der Verstärkungsbereich ist in 13 Stufen eingeteilt. Beim MAX9939 ist es möglich, die Verstärkung von 1 bis 157 in neun Stufen einzustellen. Wird zusätzlich ein zweiter Verstärker des Typs MAX9939 verwendet, erhält man einen deutlich höheren Verstärkungsfaktor von bis zu 24649 bei

mehr Flexibilität. Die Tabelle: 5.2 zeigt alle möglichen Verstärkerstufen beim Einsatz von zwei PGA des Typs MAX9939. Zu beachten ist hier die steigende Schrittweite bei zunehmender Verstärkung.

Gain/Gain	**1**	**10**	**20**	**30**	**40**	**50**	**60**	**80**	**120**	**157**
1	1	10	20	30	40	50	60	80	120	157
10	10	100	200	300	400	500	600	800	1200	1570
20	20	200	400	600	800	1000	1200	1600	2400	3140
30	30	300	600	900	1200	1500	1800	2400	3600	4710
40	40	400	800	1200	1600	2000	2400	3200	4800	6280
50	50	500	1000	1500	2000	2500	3000	4000	6000	7850
60	60	600	1200	1800	2400	3000	3600	4800	7200	9420
80	80	800	1600	2400	3200	4000	4800	6400	9600	12560
120	120	1200	2400	3600	4800	6000	7200	9600	14400	18840
157	157	1570	3140	4710	6280	7850	9420	12560	18840	24649

Tabelle 5.2: Mögliche Verstärkungsfaktoren bei der Verwendung von zwei Verstärkerstufen.

Der PGA280 von Texas Instruments hat einen ähnlichen Verstärkungsfaktor wie der MAX9939, anders als der MAX9939 ist der Ausgang hier aber differenziell. Weiterhin bietet der Verstärker MAX9939 die Möglichkeit, das Offset zwischen den beiden Eingängen von bis zu $\pm 17{,}6\,\mathrm{mV}$ intern zu kompensieren. Dies kann zur späteren Feineinstellung der Subtrahiererschaltung verwendet werden. Aufgrund der Flexibilität in der Verstärkung und der Zusatzfunktion des Verstärkers MAX9939 wird für den Aufbau der Schaltung der MAX9939 in zwei Stufen verwendet.

Rheostat

Bei dem Rheostaten ist wie bei dem PGA die Flexibilität ein wesentlicher Faktor. Daher ist eine möglichst hohe Anzahl an einstellbaren Stufen wünschenswert. Aus Tabelle 5.3 wird ersichtlich, dass der Rheostat AD5270 mit 1024 Stufen die besten Einstellmöglichkeiten bietet. Die aufgeführten Rheostaten können via Serial Peripheral Interface (SPI) kommunizieren. Dies passt zu dem Verstärker MAX9939 und ermöglicht eine Implementierung der Kommunikation mit relativ wenig GPIOs des Mikrocontrollers XMC1100, da so lediglich ein weiterer Chip-Select-Pin benötigt wird.

Hersteller	Modell	Widerstand[Ω]	Stufen	BUS
Texas Instruments	TPL0501	100k	256	SPI
Texas Instruments	MCP4141	5k/10k 50k/100k	129	SPI
Analog Devices	AD5270/1	50k/100k	1024	SPI

Tabelle 5.3: Die Tabelle zeigt drei Rheostaten von unterschiedlichen Herstellern.

5.2 Anpassung: Initialer Spannungsteiler

Damit nach dem initialen Spannungsteiler noch mindestens 50% Signalamplitude vorhanden ist, muss eine nachträgliche Überprüfung des Widerstandsverhältnisses durchgeführt werden, da der untere Widerstand, wie in Abbildung 5.2 zu sehen, parallel zur Gesamtschaltung liegt. In Abhängigkeit des Innenwiderstands R_i der Schaltung verändert sich das Widerstandsverhältnis. Der Spannungsteiler ist zunächst mit einem Verhältnis von $U_2/U_{ges} = 1/2$ ausgelegt. Daraus folgt, dass $R_1 = R_2$ ist. Eine Messung zeigt, dass $U_2 \neq U_{ges}/2$. Für die Anpassung wurden in einer Messung Werte für U_2 und $U1 = Uges - U_2$ ermittelt. Für die Messung gelten folgende Werte:

- $U_{ges} = 1\,\mathrm{V}$
- $R_1 = R_2 = 10\,\mathrm{k\Omega}$
- $U_2 = 0{,}482\,\mathrm{V}$

Allgemein gilt für den Spannungsteiler

$$\frac{U_1}{U_2} = \frac{R_1}{R_2} \tag{5.1}$$

Für parallel geschaltete Widerstände ergibt sich der Gesamtwiderstand R_{ges} mit der Gleichung 5.2

$$R_1 \parallel R_2 = R_{12} = \frac{R_1 \cdot R2}{R_1 + R_2} \tag{5.2}$$

Folgt man den Bezeichnungen aus 5.2, entspricht R_2 aus Gleichung 5.1 dem Gesamtwiderstand $R_2 \parallel R_i = R_{2i}$ der Parallelschaltung. Mit der Gleichung 5.1 nach R_2 umgestellt und die gegebenen Werte eingesetzt, ergibt sich ein Gesamtwiderstand von $R_{2i} = 9{,}31\,\mathrm{k\Omega}$ (Gleichung 5.3)

Eingesetzt in Gleichung 5.1 und nach R_i aufgelöst, ergibt sich Gleichung 5.3

$$R_{2i} = \frac{R_1 \cdot U_2}{U_1} = \frac{10\,\mathrm{k\Omega} \cdot 0{,}482\,\mathrm{V}}{1\,\mathrm{V} - 0{,}482\,\mathrm{V}} = 9{,}31\,\mathrm{k\Omega} \tag{5.3}$$

Mit R_{ges} kann der Innenwiderstand R_i berechnet werden. Dazu wird Gleichung 5.2 $R2 = R_{2i}$ umgestellt und die Werte eingesetzt. Ergibt sich wie in Gleichung 5.4 gezeigt, ein Innenwiderstand R_i von $R_i = 133{,}44\,\mathrm{k\Omega}$.

$$R_i = \frac{R_{2i} \cdot R_2}{R_2 - R_{2i}} = \frac{9{,}31\,\mathrm{k\Omega} \cdot 10\,\mathrm{k\Omega}}{10\,\mathrm{k\Omega} - 9{,}31\,\mathrm{k\Omega}} = 133{,}89\,\mathrm{k\Omega} \tag{5.4}$$

Damit der Spannungsteiler das angestrebte Verhältnis von $U_2/U_{ges} = 1/2$ aufweist, muss mit dem berechneten Innenwiderstand R_i der Widerstandswert für R_2, mit dem Zielwert von $R_{2i} = 10\,\mathrm{k\Omega}$ neu berechnet werden. Hierfür wird Gleichung 5.2 nach R_2 umgestellt und der zuvor berechnete R_i eingesetzt. Der mit Gleichung 5.5 neu berechnete Widerstandswert für R_2 ergibt sich zu $R_2 = 10\,807{,}2\,\Omega$

$$R_2 = \frac{R_{2i} \cdot R_i}{R_i - R_{2i}} = \frac{10\,\mathrm{k\Omega} \cdot 133{,}89\,\mathrm{k\Omega}}{133{,}89\,\mathrm{k\Omega} - 10\,\mathrm{k\Omega}} = 10\,807{,}2\,\Omega \tag{5.5}$$

Der nächste Widerstand in der E12 Reihe ist $R_2 = 10\,\mathrm{k\Omega}$, welcher in Reihe mit einem $800\,\Omega$ Widerstand verwendet wird.

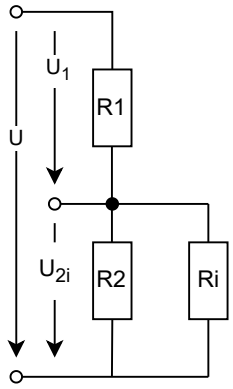


Abbildung 5.2: Der Innenwiderstand des Sensors beeinflusst das Verhältnis des Spannungsteilers und damit den Spannungsabfall U_2.

5.3 Kompensierung des Gleichanteils

Die Kompensierung des Gleichanteils wird, wie im Konzept vorgestellt, mit drei verschiedenen optionalen Ansätzen realisiert. Um jeden Ansatz unabhängig voneinander testen zu können, werden die jeweilig dazugehörigen Schaltungen über Kurzschlussbrücken (engl. Jumper) zu- oder abgeschaltet.

AC-Kopplung

Die AC-Kopplung mithilfe eines Kondensators wird direkt hinter dem initialen Spannungsteiler in Reihe geschaltet.

Mithilfe der parallelen Kurzschlussbrücke kann der Kondensator überbrückt werden und optional zu- oder abgeschaltet werden, siehe Abbildung 5.1 JP1.

Die Größe des Kondensators ist abhängig von der kleinsten zu messenden Frequenz und der maximal verfügbaren Baugröße. Geläufig sind bei Keramikkondensatoren Kapazitäten C von bis zu $C = 100\,\mu\text{F}$. Aus Gleichung 4.3 und dem Filterwiderstand $R = R_i = 16{,}2\,\text{k}\Omega$ ergibt sich die Grenzfrequenz hierbei wie folgt:

$$f_g = \frac{1}{2 \cdot \pi \cdot R_i \cdot C} = 98{,}24\,\text{mHz} \tag{5.6}$$

Die so errechnete Grenzfrequenz entspricht der Frequenz, bei der das Eingangssignal um 3 dB gedämpft wird. Bei der errechneten Grenzfrequenz wird das Signal demnach bereits durch das Messsystem beeinflusst.

Pulsweitenmodulation

Das Glättungsfilter zum Erzeugen der Referenzgleichspannung wird jeweils einmal für jede Verstärkerstufe vorgesehen. Das Glättungsfilter wird wie mit Gleichung 4.13 aus Kapitel 4.4.2 ausgelegt und auch verbaut. Über die Kurzschlussbrücken (Abbildung 5.1 JP2, JP3) kann die Referenzspannung optional zu- und abgeschaltet werden.

Rheostat

Der Rheostat erfordert die umfangreichste Beschaltung der ausgewählten Bauteile. Durch die limitierte Anzahl an GPIOs des Mikrocontrollers XMC1100 kann der Rheostat AD7250 nicht in vollem Umfang verwendet werden. Aufgrund dessen wird SPI-Bus des Rheostats nur unidirektional betrieben. Die Beschaltung des Rheostats ist aus dem Datenblatt entnommen [9]. Durch den unidirektionalen Betrieb muss der SDO-Pin des SPI-Bus an einen Pull-Up Widerstand geschaltet werden. Weiterhin wird angegeben, dass VSS sowie VDD mit 0,1 μF entkoppelt werden sollten. Dies ist wie in Abbildung 5.1 zu sehen, umgesetzt worden. Für den persistenten Speicher des Rheostats ist ein weiterer 1 μF Kondensator zwischen EXT_CAP und VSS vorgesehen. Über den Anschlüssen W und A wird der Widerstand des Rheostats eingestellt. Für die Begrenzung des einstellbaren Bereichs wird, wie in dem Konzept 4.4.2 vorgestellt, ein Offsetwiderstand $R_{Offset} = 12\,\text{k}\Omega$ eingesetzt. Die vollständige Beschaltung ist dem Anhang A.2.1 zu entnehmen. Der Rheostat ist wie die PWM an die erste Verstärkerstufe über eine Kurzschlussbrücke (Abbildung 5.1 JP2) optional verwendbar.

5.4 Verstärkerstufe

Für die Verstärkerstufe wird der Verstärker des Typs MAX9939 eingesetzt. Der PGA kann via SPI programmiert werden und bietet neben der Verstärkerfunktion auch die Möglichkeit, eine interne Kompensierung der differenziellen Eingänge durchzuführen. Da

bei der erwarteten Signalamplitude die maximale Verstärkung von $k = 157$ nicht ausreicht, um den ADC voll auszusteuern, wird ein zweiter Verstärker nachgeschaltet. Der Verstärker MAX9939 benötigt keine äußere Beschaltung für den Betrieb. Es wird ein Blockkondensator zwischen V_{cc} und GRD geschaltet. Die Abbildung 5.1 zeigt mittig, wie die Verstärkerstufen auf der Platine gesetzt sind.

Die Ausgangsspannung des Verstärkers berechnet sich mit der aus dem Datenblatt [38] entnommenen Gleichung 5.7.

$$V_{out} = \frac{Vcc}{2} - Gain \cdot (V_{inA}^{+} - V_{inA}^{-} + V_{os}) \tag{5.7}$$

Bei ersten Tests mit einem einzelnen Verstärker zeigt sich das in Abbildung 5.3 zu sehende Übertragungsverhalten. Die Abbildung 5.3 zeigt auch, dass das Übertragungsverhalten durch die vorgeschalteten Widerstände in der Steigung beeinflusst wird. Die berechnete Ausgangsspannung trifft nur für den Arbeitspunkt von 1,65 V zu. Dies ist unabhängig vom Widerstand. Aufgrund des Übertragungsverhaltens kann zwischen den Verstärkerstufen ein Offset entstehen. Dieses Offset kann, wenn nötig, mit der internen Offsetkompensierung oder mithilfe der externen PWM ausgeglichen werden. Dies ist mithilfe einer Kurzschlussbrücke optional zuschaltbar, siehe Abbildung 5.1 JP3.

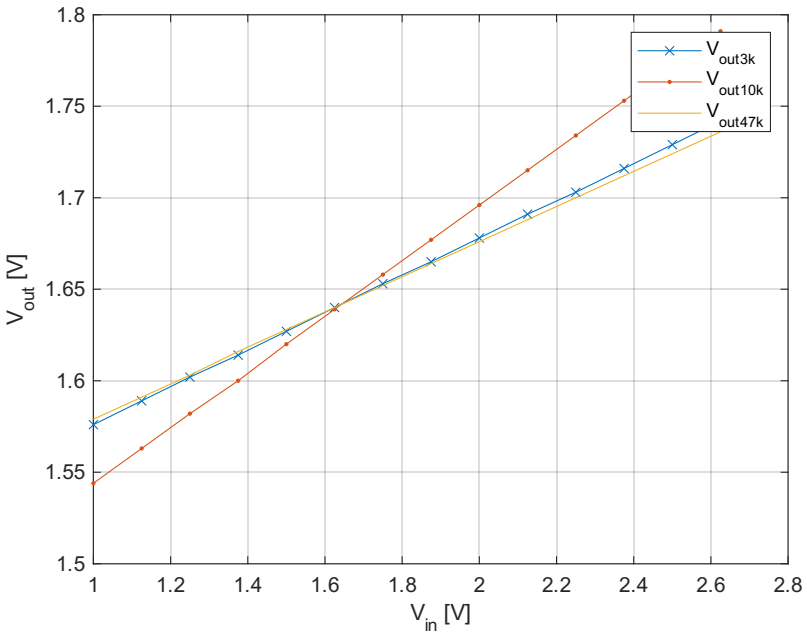


Abbildung 5.3: Der Verstärker MAX9939 weist ein lineares, aber kein konstantes Übertragungsverhalten auf. Daher kann ein Offset zwischen den Verstärkerstufen entstehen.

6 Entwicklung der Software

Im folgenden Kapitel wird die in Kapitel 4.7 erarbeitete Softwarearchitektur implementiert. Durch die Festlegung der Hardware in Kapitel 5 kann das Konzept ggf. verfeinert und implementiert werden. Die Software für den XMC1100 wird in der Programmiersprache *C* geschrieben. Steuerung, Regelung und die Auswertung wird zunächst in MATLAB programmiert. Auf die Konfiguration der Hardware wird nur in besonderen Fällen eingegangen, die Details können aus dem Quellcode entnommen werden (Anhang A.2.7).

6.1 XMC1100-Software

Der Mikrocontroller XMC1100 wird im Laboraufbau mit wenig Programmlogik versehen. Hauptaufgabe der Software auf dem Mikrocontroller XMC1100 ist die Kommunikations- und Steuerungsschnittstelle zwischen PC und den digitalen Bauteilen sowie das eigentliche Lesen des Signals via ADC.

6.1.1 Peripherie

Für die Ansteuerung der Peripherie wie UART, PWM und ADC werden die DAvE Apps verwendet. Bei DAvE handelt es sich um die Infineon eigene Entwicklungsumgebung mit Quellcodegenerator. Der Quellcodegenerator übernimmt die Konfiguration der Peripherie und generiert zudem eine Abstraktionsschicht der einzelnen Peripherien. Mithilfe des so generierten Quellcodes kann in der eigentlichen Anwendung einfach auf die Peripherie des Mikrocontrollers XMC1100 zugegriffen werden.

Kommunikation

Auf dem Mikrocontroller XMC1100 sind von der Software zwei Kommunikationswege vorgesehen. Zwischen der MATLAB-Software und dem Mikrocontroller XMC1100 wird eine UART-Kommunikation verwendet. Hierfür sind die GPIOs P2.1 (TX) und P2.2 (RX) vorgesehen (vgl. [23]). Dabei ist es möglich, zeitgleich den Debugger via SWD/SPD und dem Micro-USB-Anschluss des XMC2GO Breakoutboards zu verwenden.

Der zweite Kommunikationskanal ist ein unidirektionaler Kanal und befindet sich zwischen dem Spannungs- und Stromsensor. Hierbei handelt es sich um eine einfache GPIO-Pin Verbindung. Auf beiden Mikrocontrollern wird die gleiche Software verwendet, über die Anschlüsse wird festgelegt, welcher Mikrocontroller Sender und welcher Mikrocontroller Empfänger ist. Der sendende Mikrocontroller schaltet einen GPIO-Pin high, der Empfänger detektiert den Flankenwechsel mittels einer Interrupt Service Routine. Die Verbindung wird verwendet, um die Messung möglichst synchron durchführen zu können.

Pulsweitenmodulation

Die PWM wird verwendet, um Gleichspannung für die Kompensierung des Gleichanteils zu generieren. Über den Tastgrad der PWM wird die Ausgangsspannung hinter dem Tiefpass eingestellt. Hierfür muss der Tastgrad möglichst fein aufgelöst werden können. Dies steht jedoch im Gegensatz zu der Frequenz der PWM. Um ein möglichst glattes Ausgangssignal zu erzeugen, ist eine möglichst hohe Frequenz wünschenswert. Die Abbildung 6.1 zeigt die Auflösung der Referenzspannung in Abhängigkeit von der Frequenz. Für eine hohe Einstellauflösung muss die PWM eine relativ kleine Frequenz aufweisen. Der Mikrocontroller XCM1100 kann laut Datenblatt eine maximale PWM-Frequenz von 16 MHz erzeugen [24]. Die Gleichung 6.1 gibt an, in wie vielen Schritten N der Tastgrad von $1 - 100\%$ eingestellt werden kann.

$$N = \frac{f_{PCLK}}{f_{PWM}} \tag{6.1}$$

Die Gleichung 5.7 des PGAs zeigt, dass der Tastgrad der PWM nach dem Tiefpass mit der internen Kompensierung des PGAs $V_{os} = \pm 17{,}6\,\text{mV}$ mindestens auf 34,4 mV genau einstellbar sein muss, damit der verbleibende Gleichanteil mithilfe der internen Regelung

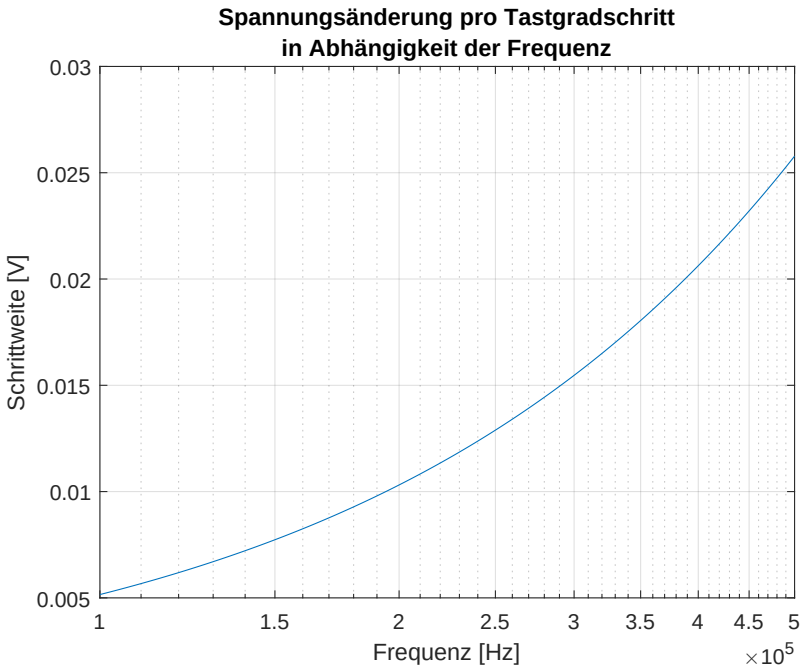


Abbildung 6.1: Die Abbildung zeigt die Änderung der durch die PWM erzeugten Referenzspannung. Mit steigender Frequenz nimmt auch die Schrittweite zu.

kompensiert werden kann. Mit der Gleichung 6.2 wird die Änderung der Referenzspannung pro Schritt U_{Step} berechnet. Umgestellt nach N ergibt sich mit Gleichung 6.1 dann die Anzahl an Stufen bei gegebener Schrittweite U_{Step}.

$$U_{Step} = \frac{U_{max}}{N} \Longleftrightarrow N = \frac{U_{max}}{U_{Step}} \tag{6.2}$$

$$N = \frac{3300\,\mathrm{mV}}{34{,}4\,\mathrm{mV}} = 95,93 \tag{6.3}$$

Bei einer Auflösung von $r = 34{,}4\,\mathrm{mV}$ pro Stufe muss der Tastgrad nach 6.2 mit mindestens $U_{Step} = 95,93$ Stufen aufgelöst werden. Damit ergibt sich nach Gleichung 6.2 eine maximale Frequenz von $f_{PWM} = 667{,}153\,\mathrm{kHz}$. Um einen sicheren Übergangsbereich zwischen PWM und interner Kompensierung zu ermöglichen, wird für die Implementierung die Frequenz f_{PWM} so gewählt, dass der Tastgrad mit $N = 200$ Stufen aufgelöst werden kann. Damit ergibt sich ein $U_{Step} = 16{,}5\,\mathrm{mV}$.

Die Implementierung und Ansteuerung erfolgt über eine DAvE-App. Der so erzeugte Quellcode liefert die Schnittstellen für die Einstellung von Frequenz und Tastgrad. Die Frequenz wird während des Betriebs nicht verändert, es kommt nur die Schnittstelle zur Änderung des Tastgrads zum Einsatz. Die so generierte Schnittstelle erwartet wie in Listing 6.1 neben dem Zeiger zum anzusteuernden PWM-Modul einen Integerwert von 32-Bit $0 - 10000$.

```
1 PWM_CCU4_STATUS_t PWM_CCU4_SetDutyCycle(
2     PWM_CCU4_t* const handle_ptr,
3     uint32_t duty_cycle);
```

Listing 6.1: Beispielhafte Schnittstelle zum Einstellen des Tastgrads

Der Integerwert wird aus einer Lookup-Tabelle mithilfe des von MATLAB erhaltenen Index entnommen. Die Berechnung des Index ist in Kapitel 6.2.1 beschrieben. An dieser Stelle werden der Wirkungsgrad und die EMV-Eigenschaften der PWM nicht beachtet.

Analog-Digital-Wandler

Der ADC soll in der Implementierung zwei Messwerte aufnehmen. Zum einen den Gleichanteil der Batteriespannung und zum anderen das verstärkte Signal. Für das Auslesen wird wie in Kapitel 4.7.3 beschrieben, die ERU verwendet. Die ERU löst bei einem Flankenwechsel am GPIO eine ISR aus, welche den ADC ausliest. Durch diese Umsetzung kann die Abtastfrequenz extern vorgegeben werden, was eine aufwendige Synchronisierung der Abtastung vermeidet.

Die Datenaufnahme wird mithilfe von zwei Pufferspeichern umgesetzt. Hierbei ist ein Puffer für das kontinuierliche Speichern der Messwerte vorgesehen, ein zweiter Buffer (Kommunikationsbuffer) wird verwendet, um immer die letzten 3000 Messwerte vorzuhalten. Bei Bedarf wird der Kommunikationsbuffer von der MATLAB-Instanz abgefragt.

6.1.2 Abstraktion digitaler Bauteile

Für die Ansteuerung der digitalen Bauteile wird eine eigene Abstraktionsschicht implementiert. Die Abstraktionsschicht hat die Aufgabe, den Quellcode zu modularisieren und

Tabelle 6.1: MAX9939 Offset- und Verstärkungsregister

MSB D7	D6	D5	D4	D3	D2	D1	LSB D0	Funktion
SHND	MEAS	V4	V3	V2	V1	V0	SEL = 0	Offset
SHND	MEAS	x	G3	G2	G1	G0	SEL = 1	Verstärkung

lesbar zu gestalten. Weiterhin wird so auch die Anwendung vereinfacht. Die Kommunikation zwischen den digitalen Bauteilen und dem Mikrocontroller ist mit SPI umgesetzt. Die Abbildung 6.2 zeigt den Aufbau der Schnittstellen zum PGA und dem Rheostaten.

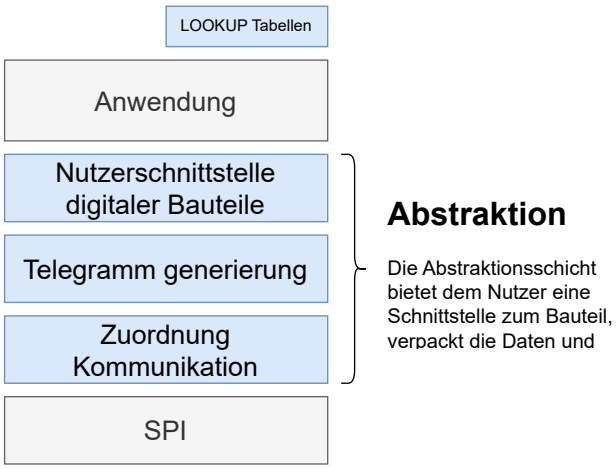


Abbildung 6.2: Der Nutzer der Schnittstelle muss die Nutzdaten sowie das Bauteil, welches programmiert werden soll, an die Schnittstelle übergeben. Die Kommunikation und das Verpacken der Daten in das Übertragungsformat wird von der Schnittstelle übernommen. Existiert nur ein Bauteil der Art, ist die Auswahl des Bauteils durch den Funktionsaufruf implizit. Die Kommunikation erfolgt in jedem Fall über SPI. Die Adressierung ist über den *Chip Select* des SPI realisiert.

Programmierbarer Verstärker MAX9939

Wie aus dem Datenblatt hervorgeht, verfügt der Verstärker MAX9939 über zwei wesentliche Funktionen, die Verstärkung und die interne Offseteinstellung vgl. [38]. Der Verstärker MAX9939 unterscheidet zwischen Offset und Verstärkung durch das *SEL*-Bit. Die Tabelle 6.1 zeigt das Telegramm für die beiden Funktionen mit *SEL*-Bit.

In der Implementierung gibt es jeweils eine Funktion, die die Aufgabe hat, das entsprechende Telegramm zusammenzusetzen und an den Verstärker MAX9939 zu übertragen.

Das Listing 6.2 zeigt die vom Nutzer zu verwendenden Funktionen. Aus dem Quellcodeausschnitt geht auch hervor, dass vom Nutzer zwischen erster und zweiter Verstärkerstufe unterschieden werden muss, hierfür wurde ein Datentyp definiert.

```
1 void max_set_gain(uint8_t* gain, max_t max);
2 void max_set_offset(uint8_t* offset, enum SIGN sign, max_t max);
```

Listing 6.2: Nutzerschnittstelle zur Ansteuerung des Verstärkers MAX9939

Die Funktionen erstellen aus dem Verstärkungs- bzw. dem Offsetwert jeweils die Nachricht an den Verstärker MAX9939 und übergibt diese an die Kommunikationsschnittstelle innerhalb des PGA Treibers. Hier wird die Unterscheidung zwischen den jeweiligen Verstärkerstufen vorgenommen, indem der entsprechende *Chip Select* der SPI Schnittstelle ausgewählt wird. Anschließend wird beides dem SPI Treiber übergeben. Für den Nutzer nicht direkt nutzbar ist die Kommunikation zwischen dem Mikrocontroller XMC1100 und der jeweiligen Verstärkerstufe, da diese Funktion mithilfe des *static* Schlüsselworts implementiert ist und daher nur innerhalb der eigenen Datei verwendet werden kann.

Für beide Funktionen ist eine Lookup-Tabelle erstellt worden. Die MATLAB-Instanz übermittelt nur den Index der entsprechenden Tabelle. Mithilfe des Index kann direkt auf den Wert zugegriffen werden und dieser in die entsprechende Funktion übergeben werden.

Beide Funktionen greifen wiederum auf eine gemeinsame Schnittstelle für die SPI Kommunikation zu, hier wird zwischen der ersten und zweiten Verstärkerstufe unterschieden und die Nachricht an den SPI Treiber weitergeleitet.

Rheostat AD5270

Die Software des Rheostats AD5270 ist ähnlich aufgebaut wie bei die Schnittstelle zum PGA. Die Schnittstelle verfügt über die Möglichkeit, den Widerstand des Rheostats einzustellen. Intern muss der Treiber hier nicht zwischen zwei baugleichen Bauteilen unterscheiden, die Ansteuerung ist jedoch komplexer. Aus dem Datenblatt geht hervor, dass der AD5270 zwei Register und eine serielle Schnittstelle besitzt (vgl. [9]).

Wie in Abbildung 6.3 zu sehen, ist der Widerstandswert zwischen Ausgang A und W über das *RDAC Register* einzustellen. Aufgrund des Schreibschutzes des *RDAC Registers* muss zum Schreiben wie folgt vorgegangen werden:

1. Deaktivieren des Schreibschutzes
2. Schreiben des *RDAC Registers*
3. Aktivieren des Schreibschutzes

Der Nutzer muss durch die Gestaltung der Schnittstelle nur den Widerstandswert angeben. Dieser kann wieder über eine Lookup-Tabelle abgerufen werden. Es können Widerstandswerte von $0 - 100\,\mathrm{k\Omega}$ in 1024 Stufen eingestellt werden.

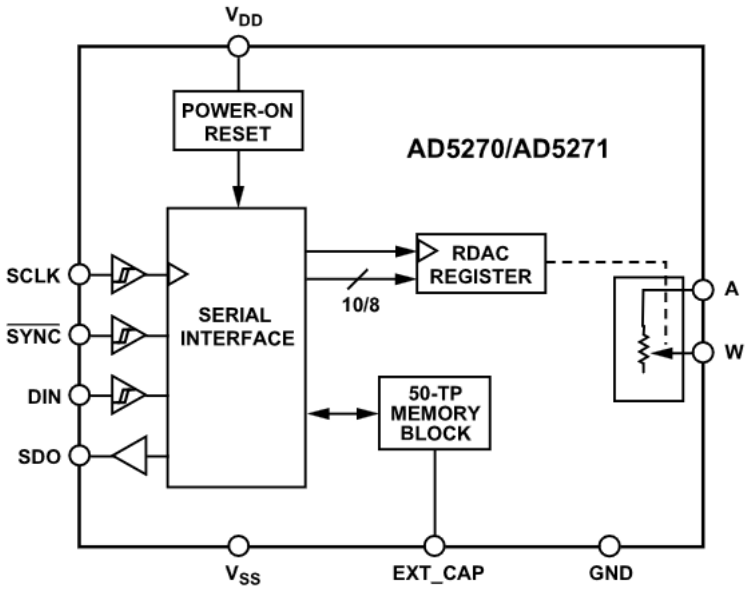


Abbildung 6.3: Die Daten werden via SPI in die serielle Schnittstelle geschrieben. Anschließend kann im *RDAC Register* der Widerstandswert eingestellt werden. Alternativ kann über SPI auch ein zuvor im *50-TP Block* gespeicherter Widerstandswert in das *RDAC Register* geladen werden [9].

6.2 MATLAB-Software

Mithilfe von MATLAB-Skripten werden die Funktionen des Sensors gesteuert und die Messungen ausgewertet. Für die Software wird daher im Folgenden eine Klasse zur Ab-

straktion des Sensors implementiert, diese enthält die Regelungen der Gleichanteilkompensierung sowie der Verstärkung.

6.2.1 Strom- und Spannungssensor

Die Sensorhardware wird mit der Software abstrahiert, da es sich um die gleiche Hardware handelt. Es wird via Software unterschieden, ob eine Strom- oder Spannungsmessung erfolgen soll. Die Klasse implementiert die aufgeführten Funktionen:

1. Kommunikation
    - Zuordnung der Steuerbefehle
    - Verkettung von Steuerbefehl und Daten
    - Empfang von Datensätzen
2. Indexbestimmung der Lookup Tabellen für ...
    - ... den Tastgrad der PWM
    - ... die Offsetspannung des MAX9939 (I. Stufe)
    - ... die Verstärkung (I. u. II. Stufe)
    - ... den Widerstand des AD7250
3. Funktion zur synchronen Messung durch Steuerung des Signalgenerators

Daraus ergibt sich das in Abbildung 6.4 zu sehende Klassendiagramm. Die serielle Kommunikation via UART wird von der Klasse übernommen. Die zentrale Klasse ist der Controller. Der SensorHandler abstrahiert den Zustand der einzelnen Sensorplatine und leitet die Befehle an den eigentlichen Sensor weiter.

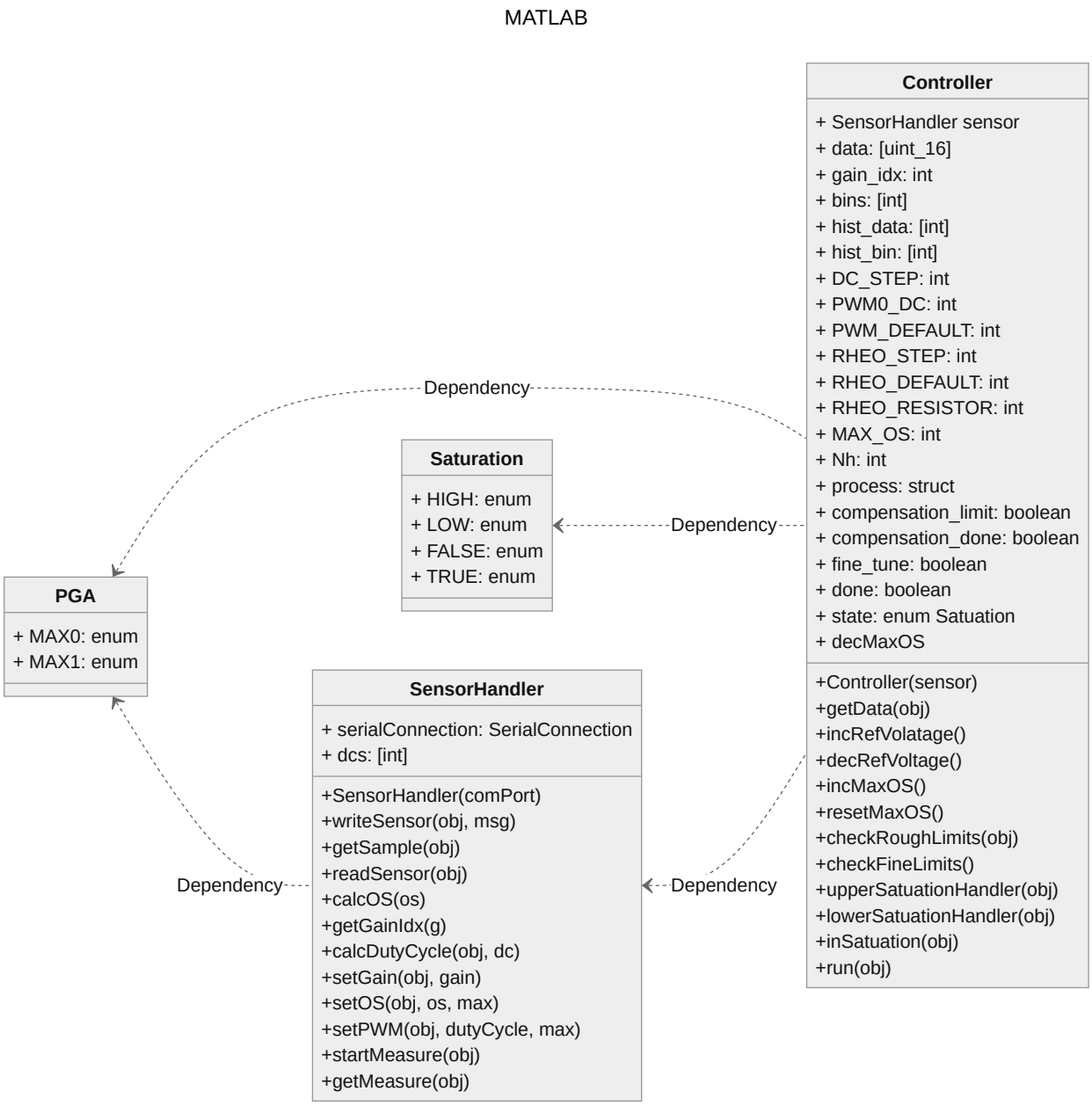


Abbildung 6.4: Die Abbildung zeigt das Klassendigramm der MATLAB-Implementierung. Die Enumerationen PGA und Saturation dienen der Lesbarkeit des Quellcodes und werden jeweils von der SensorHandler-Klasse und der Controller-Klasse verwendet. Die SensorHandler-Klasse ist die Abstraktion der Hardware und interagiert direkt mit den Sensoren via UART. Die Controller-Klasse ist die übergeordnete Instanz. Sie benötigt eine SensorHandler-Klasse und steuert die Funktionen des Sensors.

Bestimmung des Verstärkungsfaktors

Bei der Verstärkung gibt es die Besonderheit, dass über zwei Stufen hinweg verstärkt werden kann. Um hier dem äußeren Regelalgorithmus zu ermöglichen, eine Gesamtverstärkung zu übergeben, wird innerhalb der Klasse entschieden, welcher Verstärker welchen Verstärkungsfaktor einstellen soll. Jede der Verstärkerstufen kann nur bestimmte diskrete Verstärkungen einstellen. Der vom Nutzer übergebene Verstärkungsfaktor muss demnach ein Produkt aus einer Kombination der möglichen Verstärkungsfaktoren einer einzelnen Stufe sein. Die Funktion besteht aus dem in Listing 6.3 zu sehenden Quellcode. Die Funktion gewährleistet, dass die kleinere der beide Verstärkungsfaktoren immer in der zweiten Stufe eingestellt wird.

```
function setGain(obj, gain)
    % Die Funktion errechnet
    % mithilfe der Verstärkerstufen des MAX9934
    % und der vom Nutzer übergebene Verstärkung
    % die Verstärkungsfaktoren für die einzelnen
    % Verstärkerstufen.
    gain_ref = [1, 10, 20, 30, 40, 60, 80, 120, 157];
    gain0 = 1;
    gain1 = 1;
    for i=1:length(gain_ref)
        temp_gain = gain/gain_ref(i);
        if ismember(temp_gain, gain_ref)
            gain0 = gain_ref(i);
            gain1 = temp_gain;
        end
    end
    obj.setMAX(gain0, PGA.MAX0)
    obj.setMAX(gain1, PGA.MAX1)
end
```

Listing 6.3: Die Funktion teilt die vom Nutzer übergebene Gesamtverstärkung auf beide Verstärkerstufen auf. Die Gesamtverstärkung muss ein Produkt aus zwei Einzelverstärkungen sein. Die größere Verstärkung wird immer in der ersten Stufe gesetzt.

6.2.2 Kompensierung des Gleichanteils und Verstärkung

Für die Implementierung des in Abbildung 4.18 gezeigten Regelansatzes müssen neben dem Ausgangszustand zunächst weitere Bedingungen ermittelt werden und das grundsätzliche Vorgehen definiert werden. Die Verstärkung wird hierbei nicht weiter betrachtet, diese geht aus Abbildung 4.18 hervor. Das Signal wird konstant verstärkt, solange es sich nicht in Sättigung befindet. Sobald der Regler erkennt, dass sich das Signal in Sättigung befindet, übernimmt hierfür eine Funktion das weitere Vorgehen. Abbildung 6.5 zeigt den Ablauf anhand des Beispiels der oberen Sättigung. Die Funktion wird aufgerufen, sobald der ADC sich in obere Sättigung befindet. Es wird der Tastgrad solange verkleinert, bis sich das Signal entweder nicht mehr in Sättigung befindet oder zuvor bereits so weit verstärkt wurde, dass sich das Signal oben sowie unten in Sättigung befindet. Sollte sich das Signal mit einer minimalen Veränderung des Tastgrads in unterer Sättigung befinden, oszilliert die Regelung.

Mit den Gleichungen 5.7 und 6.2 wird ersichtlich, dass bei hoher Verstärkung schon eine Minimaländerung des Tastgrads ausreichend ist, um mit dem Ansatz aus Abbildung 4.18 den Regler in einen oszillierenden Zustand zu versetzen. Hier versucht der Regler dann mit der Feineinstellung des Verstärkers MAX9939 in Millivoltschritten, das Signal zu korrigieren. Gelingt dies nicht, wird der Vorgang abgebrochen und der letzte stabile Zustand wiederhergestellt. Die Abbildung 6.6 zeigt den etwas komplexeren Subalgorithmus der Feineinstellung. Grundsätzlich ist das Vorgehen jedoch das Gleiche wie in dem übergeordneten Regelalgorithmus.

Für die konkrete Implementierung wird eine Controller-Klasse implementiert. Für die Kommunikation mit dem Sensor benötigt der Konstruktor eine Instanz der Sensor-Klasse. Diese wird verwendet, um die Funktionen des Sensors anzusteuern und um Datensätze anzufordern. Der Einstiegspunkt der Regelung wird mit einer einzigen Methode für den Nutzer bereitgestellt. Die *run()* - Methode stellt damit die äußere Schleife der Regelung dar und greift von hier aus auf unterstützende Methoden zu. Für jeden der beschriebenen Zustände ist eine spezialisierte Funktion implementiert. Ob sich der ADC in Sättigung befindet, wird mithilfe des Histogramms als Sättigungskriterium aus Kapitel 4.5.1 bewertet. In Listing 6.4 wird die Gleichung 4.21 implementiert. Die Funktion gibt für die bessere Lesbar- und Wartbarkeit des Quellcodes eine Enumeration zurück.

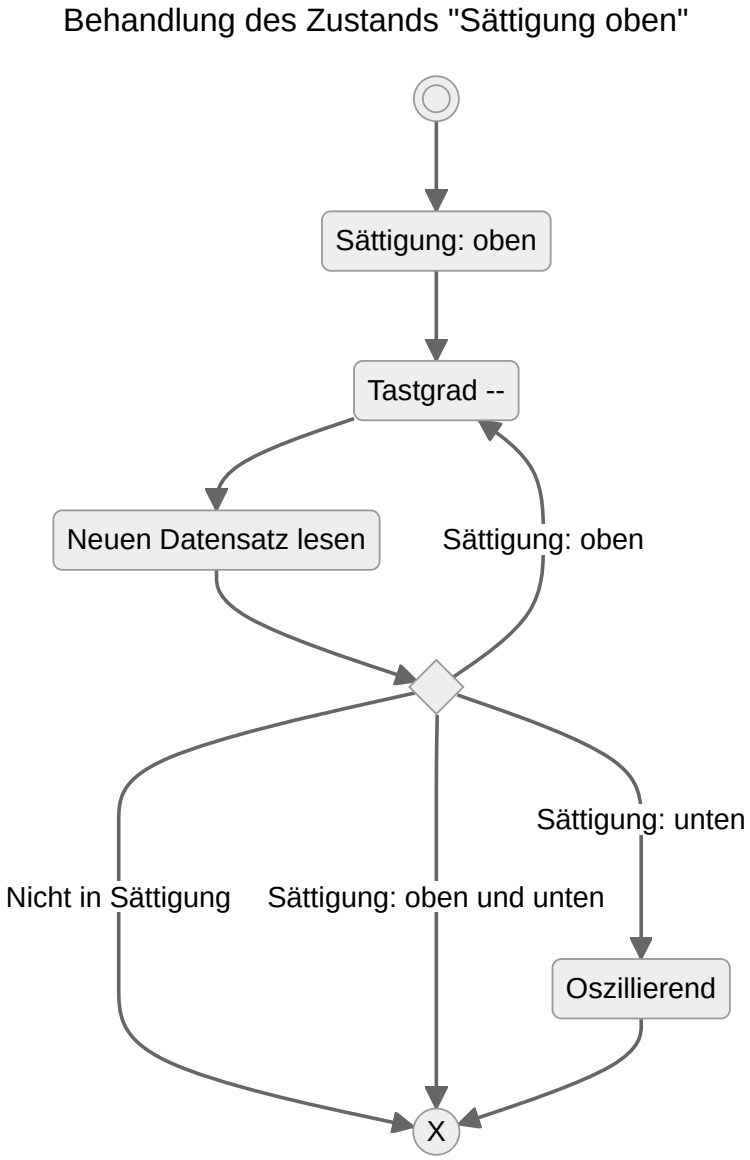


Abbildung 6.5: Die Abbildung zeigt eine leicht vereinfachte Darstellung der Gleichanteilkompensierung für das Beispiel des Zustands der Sättigung im oberen Bereich. Oszilliert der Regler zwischen oberer und unterer Sättigung, greift eine weiter Funktion und versucht über die Feineinstellung das Signal zu stabilisieren.

```
function saturation = inSaturation(obj)
    % Die Funktion überprüft
    % ob sich der ADC in Sättigung befindet
    saturation = Saturation.FALSE;
    obj = obj.getData()
    if obj.hist_data(100)/sum(obj.hist_data) > obj.No
        saturation = Saturation.HIGH;
    end
    if obj.hist_data(1)/sum(obj.hist_data) > obj.Nu
        saturation = Saturation.LOW;
    end

    if (obj.hist_data(1) > obj.Nu) && (obj.hist_data(100) > obj.No)
        saturation = Saturation.TRUE;
    end
end
```

Listing 6.4: Die Funktion liest einen neuen Datensatz ein und überprüft anhand des Histogramms mit den Gleichungen 4.21 und 4.20, ob sich das Signal in Sättigung befindet.

6.2.3 Synchronisierung der Messung

Da es sich bei dem EIS-Sensorsystem um einen verteiltes System handelt, muss das Signal für die genaue Bestimmung der Phase hoch synchron abgetastet werden. Die Abtastung wird mithilfe des programmierbaren Signalgenerators *Tektronix - AFG1062* durch ein Rechtecksignal ausgelöst. Der Ablauf der Synchronisierung ist in der Abbildung 6.7 gezeigt. Die in Abbildung 6.7 markierten Bereiche zeigen sequenzielle Bereiche, welche die Steuerung der Strom- und Spannungssensoren betreffen, an diesen Stellen ist eine vollständige Synchronisierung des Ablaufs mit dem Laboraufbau nicht möglich. Aufgrund des so entstehenden Zeitdeltas Δt wird eine Phasenverschiebung zwischen den beiden Messungen erwartet. Die Gleichung 6.4 zeigt den Zusammenhang zwischen Phase, Periode und dem entstanden Δt.

$$\frac{\Delta t * 360°}{T} = \Delta\varphi \tag{6.4}$$

Die so entstehende Phasenverschiebung ist frequenzabhängig und muss für jede Frequenz individuell berechnet werden. Die Frequenzabhängigkeit ergibt sich durch das Konstante

Δt. Betrachtet man die Übertragung isoliert von der Ausführzeit der Software, welche zwischen der MATLAB-Software und dem physikalischen UART-Signal liegt, dann ist die Übertragungszeit konstant. Die Ausführungszeit der Software auf der Ebene des Betriebssystems zeigt jedoch starke Schwankungen. Somit ist eine korrekte Messung der Phasenverschiebung nicht möglich. Um die Zeit zwischen den Messungen konstant und möglichst kleine zu halten, wird die Kommunikation zwischen den beiden Mikrocontrollern verwendet. Mit diesem Ansatz der Kommunikation ist die nicht konstante Zeit der UART-Übertragung nicht mehr Teil der Messung.

Die Kommunikation findet wie in Kapitel 6.1.1 beschrieben statt. Mithilfe von der MATLAB-Software wird wie zuvor die Messung über die UART-Schnittstelle gestartet. Der angesprochene Mikrocontroller schaltet dann einen GPIO-Pin von 0 V auf 3,3 V. Dieser Wechsel der Flanke wird vom zweiten Mikrocontroller erkannt, welcher daraufhin die Messung startet. Die Verzögerung zwischen den Mikrocontroller ist damit minimiert und konstant. Dadurch wird auch der potenziell entstehende Fehler konstant und kann rechnerisch kompensiert werden.

6.2.4 Histogrammbasierte Bewertung der Messung

Um Messausreißer zu erkennen, ist es nötig, die einzelnen Messungen zu bewerten. Hierfür wird der Ansatz über die Messwertverteilung aus Kapitel 4.7.2 verwendet.

Die Funktion verwendet das Histogramm als zentrale Methode, um zu beurteilen, ob ein Datensatz plausibel ist. Hierfür werden die Messwerte der Phase in 3600 Klassen eines Histogramms aufgeteilt. Bei 3600 Klassen entspricht jede Klasse $1/10°$. Es wird der Index der Klasse i_{max} bestimmt, in welcher sich die meisten Messwerte befinden. Damit ergibt sich für eine geforderte Genauigkeit von 1°, dass ausgehend von dem Index i_{max} nur Messwerte innerhalb von $i_{max} \pm 5$ berücksichtig werden. In der Implementierung aus listing 6.5 wird gezeigt, wie das Maximum im Histogramm bestimmt wird. Die Indizes der Phase entsprechen denen der Impedanz, daher muss nur einer der beiden Datenfelder betrachtet werden.

```
hh = histogram(phase, bins);
[~, idx] = max(hh.Values); % bestimmung des Maximums
low_idx = idx - round((bins/(bins / 10)) * 0.5);
hi_idx = idx + round((bins/(bins / 10)) * 0.5);
if (low_idx > 0) && (hi_idx < bins)
  % Festlegen der Grenzwertklassen
  lower_edge = hh.BinEdges(low_idx);
  upper_edge = hh.BinEdges(hi_idx);
else
  % Grenzen außerhalb des Arrays
  filtered_impedance = [];
  filtered_phase = [];
  return
end

idxToDelete = [];
for i = 1:length(phase)
  if (phase(i) < lower_edge) || (phase(i) > upper_edge)
    % Bestimmung der zu löschenden Indizes
    idxToDelete = [idxToDelete, i];
  end
end
impedance(idxToDelete) = [];
phase(idxToDelete) = [];
```

Listing 6.5: Der Quellcode zeigt einen Ausschnitt der Funktion zur Validierung der Messungen. Zunächst wird der Index des Maximums bestimmt, anschließend werden die Grenzwerte berechnet. Alle Messwerte außerhalb der Grenzen werden verworfen.

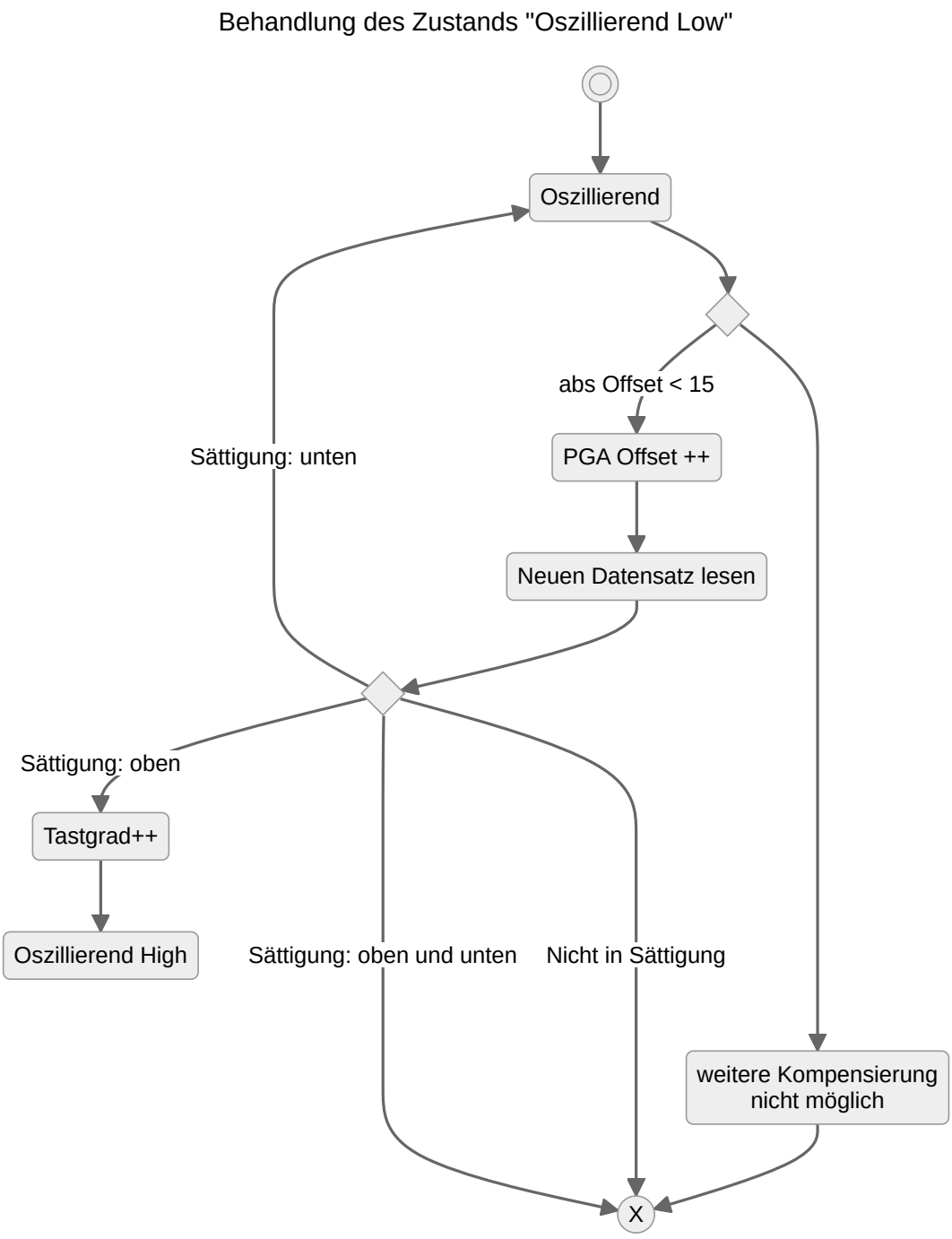


Abbildung 6.6: Die Abbildung zeigt den vereinfachten Regelalgorithmus für die Feineinstellung. Anstelle des Tastgrads der PWM wird hier die vom MAX9939 intern bereitgestellte Kompensierung verwendet. Sollte die Kompensierung zu einem Schwingen führen, versucht der Algorithmus das Signal von ″oben″zu stabilisieren.

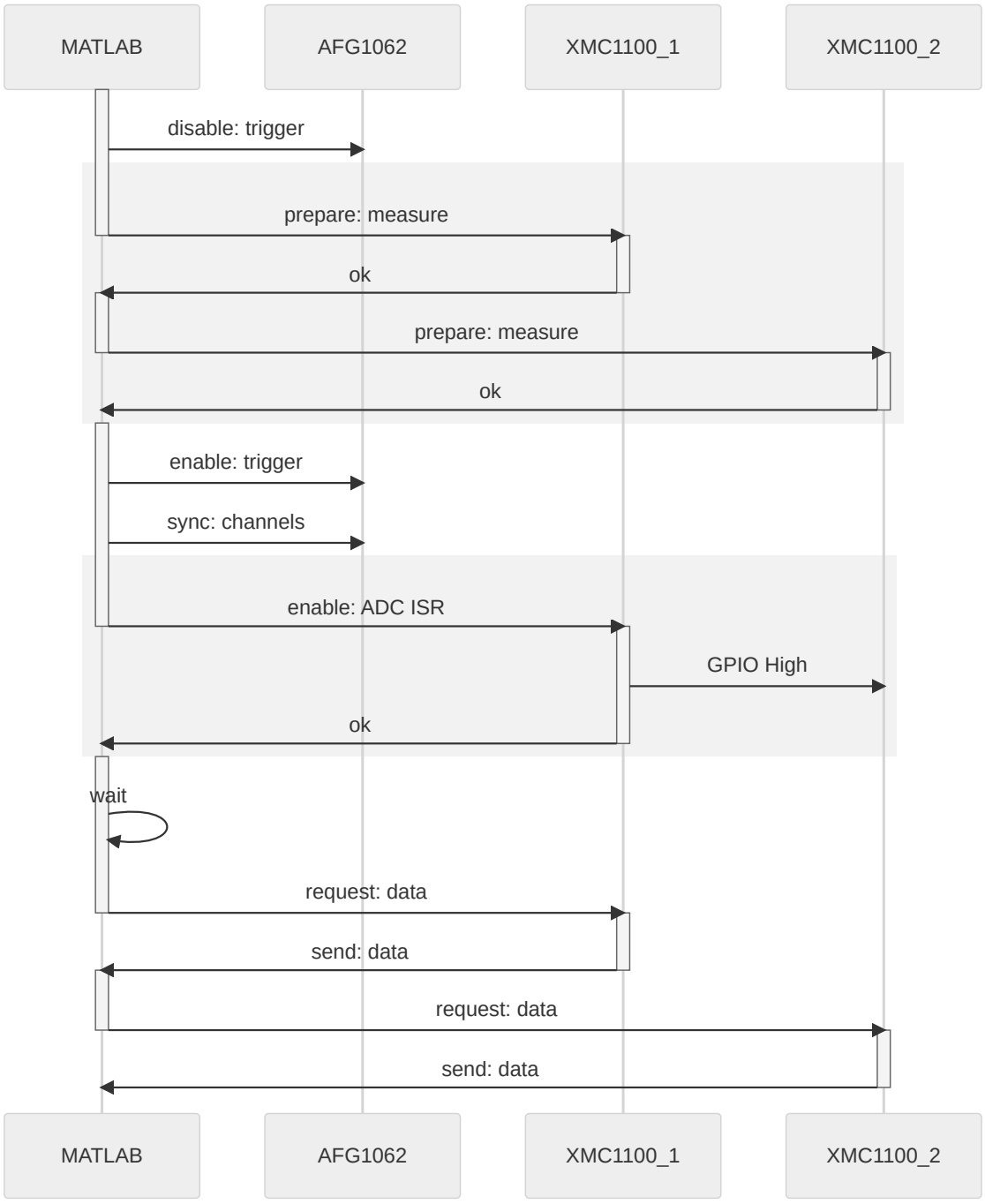


Abbildung 6.7: Die Abbildung zeigt die notwendige Kommunikation zur Synchronisierung der Messung

7 Messung und Erprobung

Im zugrunde liegenden Kapitel wird das Gesamtsystem aufgebaut und erprobt. Es werden die Möglichkeiten und Grenzen des Sensorsystems durch Messungen ermittelt und dargestellt.

7.1 Laboraufbau

Der Laboraufbau ist so gestaltet, dass alle Messungen ohne wesentliche Umbauten vorgenommen werden können. Eine Herausforderung sind die unterschiedlichen Potenziale der Messung. Beide Sensoren werden via USB mit Strom versorgt, müssen sich aber aufgrund der Messung an der Zelle dem Potenzial der Batterie anpassen. Hierfür wird die Stromversorgung des Sensors galvanisch getrennt. Das Gleiche gilt für die Kommunikation zwischen den einzelnen Sensoren sowie des Triggersignals aus dem Signalgenerator. Hierfür wird für jedes Potenzial eine galvanische Trennung mithilfe eines Optokopplers vorgesehen. Die Abbildung 7.1 zeigt, wie der Sensor später im Laborkontext verwendet werden soll. Die Implementierung in MATLAB, als zentrale Instanz steuert auch den Signalgenerator für die Stromanregung.

7.2 Inbetriebnahme

Bei der Inbetriebnahme der Sensorplatine werden die Implementierungen auf ihre Funktion hin überprüft. Es werden die in Tabelle A.1 aufgeführten Funktionen überprüft. Aus der Inbetriebnahme ergibt sich, dass die implementierten Konzepte in ihrer Funktion den Erwartungen entsprechen. Bei den Tests ist aufgefallen, dass die Regelung relativ langsam ist. Dies hat vor allem den Grund, dass es sich um eine externe Regelung handelt. Der Regelalgorithmus in der MATLAB-Software muss nach jedem Korrekturschritt auf einen neuen Datensatz des Sensors warten, um auf die Änderung reagieren zu können.

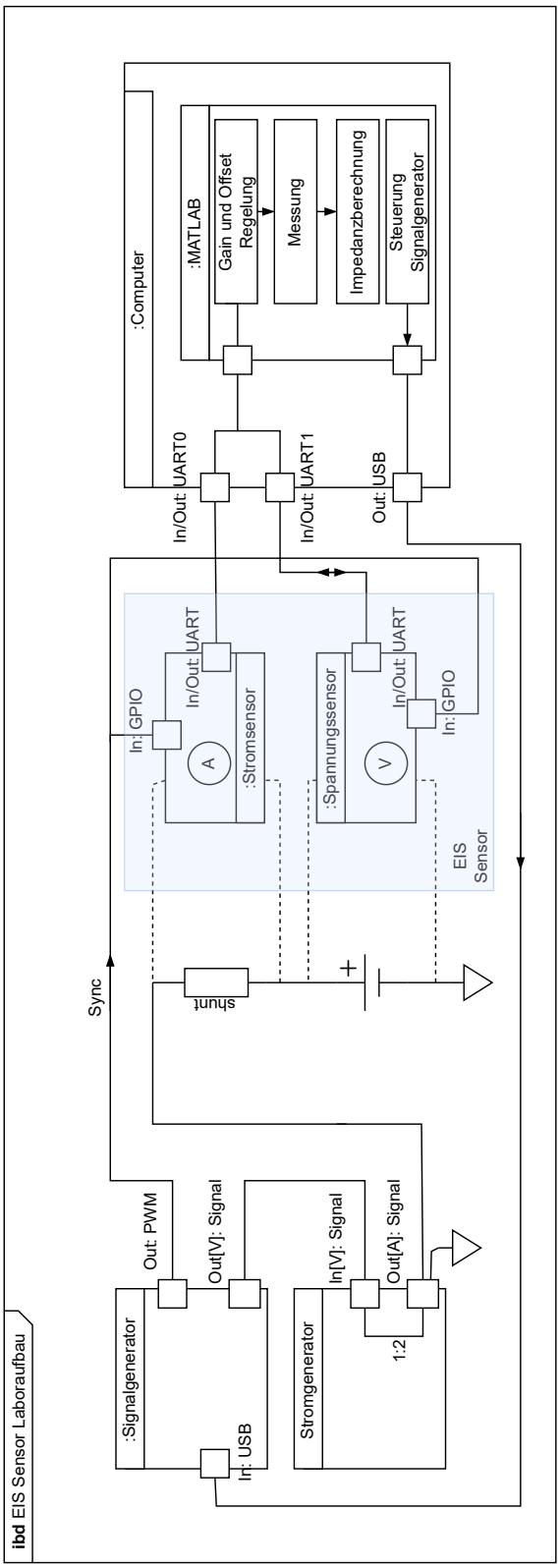


Abbildung 7.1: Die Abbildung zeigt den Laboraufbau des Systems. Ein externer Signalgenerator erzeugt neben der Stromanregung auch ein Synchronisierungssignal, um die Signalabtastung zu synchronisieren. Die Sensoren sind via USB- und UART-Kommunikation an einen Computer angeschlossen, Regelung, Auswertung und Steuerung werden in MATLAB implementiert.

Für die Übertragung selbst lässt die mithilfe der Gleichung 7.1 die minimale Übertragungszeit bestimmen.

$$t = \frac{n_{bit}}{baude} \tag{7.1}$$

Für die Kommunikation zwischen der Implementierung in MATLAB und dem Mikrocontroller XMC1100 wird das in Abbildung 4.15 zu sehende 16-Bit lange Telegramm verwendet. Dieses wird zum Anfragen eines Datensatzes einmalig an den Mikrocontroller gesendet. Nach dem Erhalt der Daten werden diese ausgewertet und es wird, wenn nötig, ein entsprechender Steuerbefehl an den Mikrocontroller XMC1100 zurückgesendet. Die Übertragung via UART erfolgt im Modus 8N1, daraus ergeben sich mit Start- und Stoppbit 10 Bit pro Byte. Bei den Datensätzen handelt es sich jeweils um 3000 16-Bit-Integer. Für eine Übertragung von zwei Byte werden 20 Bit benötigt. In Summe werden für einen Regelschritt demnach 3002 Werte 16-Bit-Integer zwischen der MATLAB Instanz und dem Mikrocontroller XMC1100 ausgetauscht. Es ergibt sich die mit Gleichung 7.2 errechnete Übertragungszeit pro Regelungsschritt von $t = 0{,}52\,\mathrm{s}$ bei einer Bauderate von 115200 $\frac{bit}{s}$

$$t = \frac{3002 \cdot 20Bit}{115200} = 520\,\mathrm{ms} \tag{7.2}$$

Um den Vorgang zu beschleunigen, wird die Baudrate auf 921600 $\frac{bit}{s}$ erhöht. Mit der beschleunigten Übertragung verringert sich die Übertragungszeit auf $t = 65\,\mathrm{ms}$. Im Normalbetrieb kann durch sinnvolle Startwerte der Regelung die benötigte Zeit minimiert werden. Ein maximaler Wert ergibt sich bei Extremwerten der zu überwachenden Batteriezelle. Bei der Verwendung des Rheostaten kann die Referenzspannung in 1024 Stufen eingestellt werden, in Randfällen kann dies zu mehr als 512 Regelschritten führen.

7.3 Bestimmung des Schwellwerts durch Messung

Der in Kapitel 4.5.1 durch die Simulationsergebnisse festgelegte Schwellwert wird im folgenden Abschnitt durch Messungen untersucht. Der Ansatz zur Bestimmung des Schwellwerts N_h besteht darin, durch eine Klirrfaktoranalyse den Schwellwert für N_h zu überprüfen. N_h muss hierbei so gewählt werden, dass sich noch keine wesentliche Amplitudenabweichung der Grundfrequenz ausgeprägt hat. In der Simulation aus Kapitel 4.5.1

ist zuvor ein Schwellwert für $N_{h0} = N_{hm} = 0,15$ ermittelt worden. Das Verhältnis N_h ist hier mit 10% Grenzen gebildet worden, die rot markierten Bereiche sind in Abbildung 7.3 (mittig) zu sehen. Für die Bestimmung von N_h wird ein verrauschtes Signal von einer einzelnen Sensorplatine aufgenommen. Nach jeder Messung wird die Amplitude des Signals schrittweise erhöht, bis sich das Signal vollständig in Sättigung befindet, anschließend wird der Klirrfaktor für jede Messung berechnet. Die Amplitude wird von $0{,}6\,\mathrm{V_{pp}}$ in 20 Schritten auf $3\,\mathrm{V_{pp}}$ erhöht. In dem Spektrum der Messergebnisse in Abbildung 7.2 (oben) sind neben der Grundfrequenz von 61 Hz die durch Sättigung hervorgerufenen harmonischen Spektralanteile deutlich zu sehen. Auch der berechnete Klirrfaktor in Abbildung 7.2 (unten) zeigt wie erwartet mit zunehmender Sättigung einen steigenden Klirrfaktor. Entgegen der Simulation zeigt die Messung schon bei einer Amplitude von ca. $0{,}85\,\mathrm{V_{pp}}$ einen deutlichen Anstieg des Klirrfaktors. Außerdem zeigt die Messung, dass sich rechnerisch auch für Signale außerhalb der Sättigung des ADCs ein Klirrfaktor größer null bei $k \approx 0,01$. Der in der Simulation festgelegte Schwellwert $N_h = 0,15$ kann durch die Messung als plausibel angenommen werden. In Abbildung 7.3 ist im oberen Graph zu sehen, dass die Abweichung in der Amplitude der Grundfrequenz bei weniger als $0{,}05\,\mathrm{V}$ liegt, was einem Klirrfaktor von $k \approx 0,05$ entspricht. In derselben Abbildung (unten) zeigt sich, dass $N_h = 0,15$ einem Klirrfaktor von $k \approx 0,05$ entspricht. Wie stark die Amplitude der Grundfrequenz von der eigentlichen Amplitude abweichen darf, ist abhängig von den Anforderungen der Anwendung. Gleiches gilt für die Festlegung der Grenzen, hierbei ist die Anzahl der Quantisierungsstufen des ADC maßgeblich, um eine ausreichende Auflösung des Signals gewährleisten zu können.

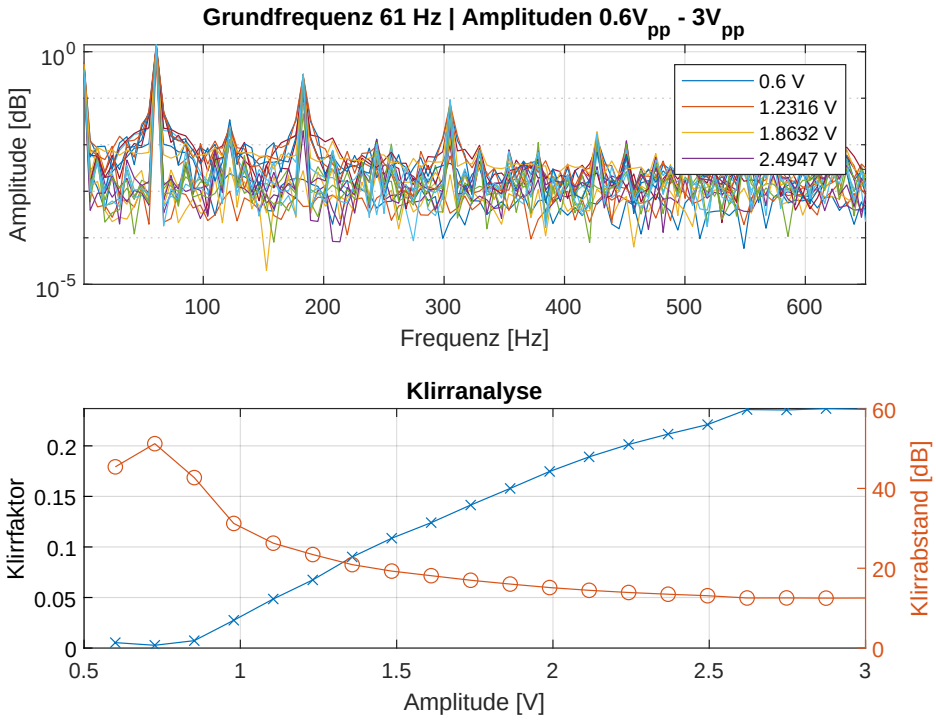


Abbildung 7.2: Die Abbildung zeigt das Spektrum der Messungen bei 61 Hz mit verschiedenen Amplituden (oben). Deutlich zu erkennen sind die durch Sättigung hervorgerufenen harmonischen Spektralkomponenten. In der Analyse (unten) zeigt sich ein deutlicher Anstieg des Klirrfaktors mit ansteigender Amplitude.

7.4 Relative Phasengenauigkeit

Die Phasengenauigkeit beschreibt die relative Phase zwischen den Sensorplatinen. Um eine potenzielle konstante Abweichung zu ermitteln. Dafür wird der Labormessplatz zunächst mit ohmschen Widerständen aufgebaut. Außerdem wird mit der Messung ein potenzieller Phasen- und Amplitudengang bestimmt. Eine konstante Abweichung kann durch die Laufzeit der Kommunikation zwischen den Spannungs- und Stromsensor zustande kommen.

Bei konstanter Phasenverschiebung kann ein Fehler durch eine relative Verschiebung der Messwerte von Spannungs- und Stromsensoren behoben werden. Die Messung wurde mit den in Tabelle 7.1 zu sehenden Parametern durchgeführt. Die Messungen zeigten einen

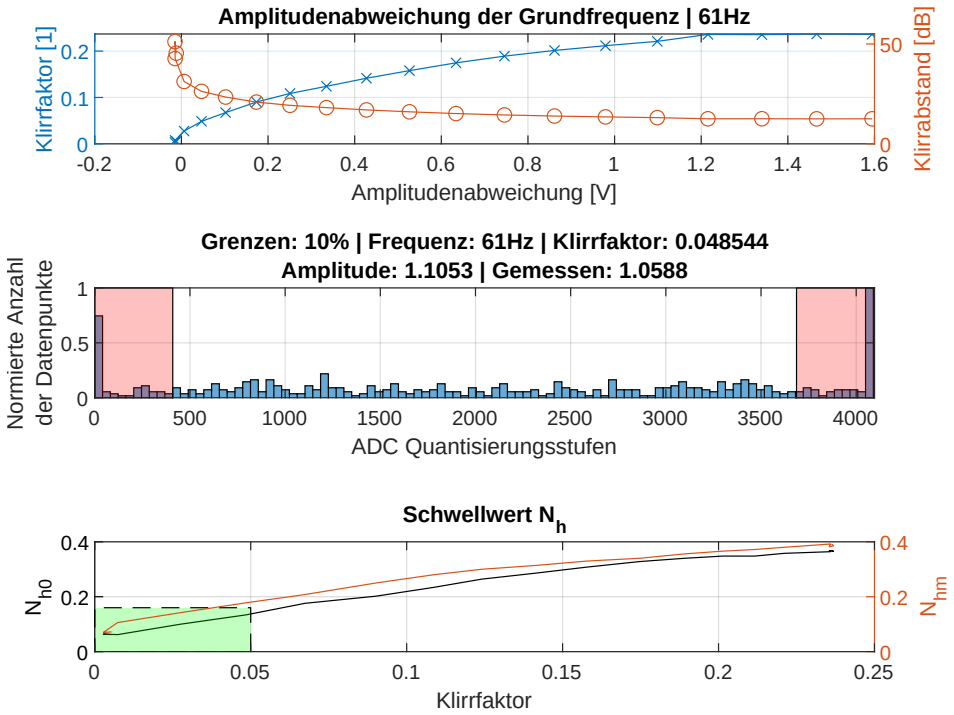


Abbildung 7.3: Die Abbildung zeigt den Klirrfaktor bzw. die Klirrdämpfung in Abhängigkeit der Amplitudenabweichung (oben). Das Histogramm zu der Messung, die gerade noch unterhalb der Klirrfaktorgrenze liegt (mittig) mit farblich markierten Grenzbereichen von jeweils 10% und die Berechnung der Schwellwerte N_h (unten). Der grüne Bereich markiert die Grenze.

konstanten Versatz von $\Delta\varphi = 21,6°$. Durch einen relativen Versatz der Messwerte von Strom- und Spannungssensor um zwei Messpunkte ist die Abweichung vor der Berechnung der Impedanz kompensiert. Die Abbildung 7.4 zeigt die kompensierte Messung. In Abbildung 7.4 ist das Übertragungsverhalten abzulesen. Der Amplituden- und Phasengang weist ab einer Frequenz von $f = 200\,\mathrm{Hz}$ geringe Abweichungen vom Soll auf.

7.5 Messung am RC-Glied

Im folgenden Kapitel wird die Messung der Impedanz mithilfe des entwickelten Systems gezeigt. Die Messung wird an einem RC-Glied durchgeführt. Es soll ermittelt werden, ob das System in der Lage ist, eine EIS durchzuführen. Es wird weiterhin untersucht, ob ein messbarer Unterschied zwischen der Lösung mithilfe des Rheostaten und der PWM besteht. Die Ergebnisse werden abschließend ausgewertet.

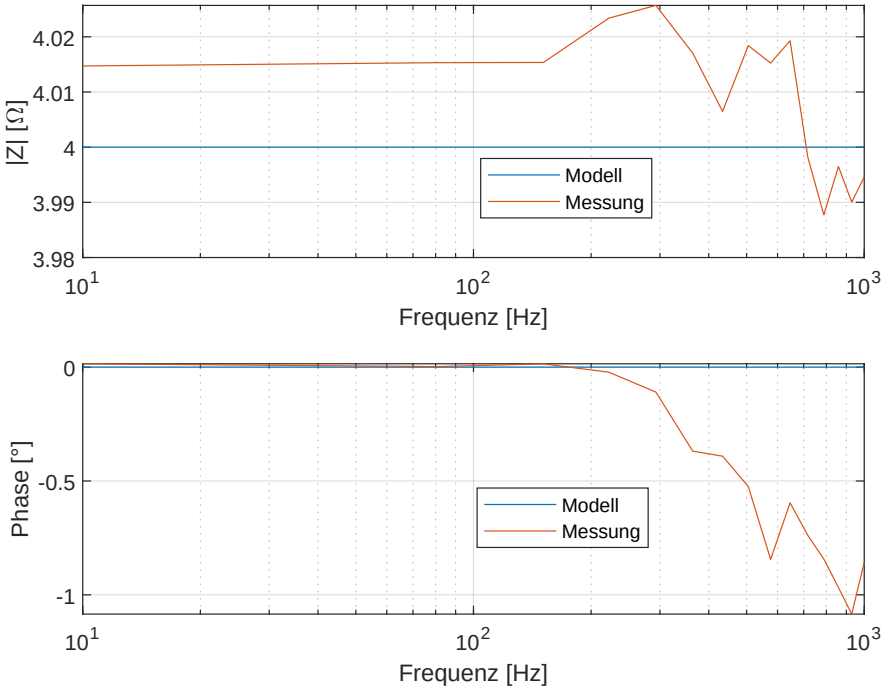


Abbildung 7.4: Die Abbildung zeigt den Betrag und die Phase der Messung an einem ohmschen Widerstand.

7.5.1 Durchführung

Tabelle 7.1: Die Tabelle zeigt Messparameter für die Durchführung der Messungen.

Parameter	**Ohmsch**	**PWM** (10 mV)	**Rheostat**	**PWM** (1 mV)
f_{min}	10 Hz	10 Hz	10 Hz	0,10 Hz
f_{max}	1000 Hz	1000 Hz	1000 Hz	1000 Hz
Anzahl der Messfrequenzen	30	30	30	30
Messungen pro Frequenz	25	25	25	25
Amplitude (AFG 1062)	0,01 V	0,01 V	0,01 V	0,001 V
DC Offset (AFG 1062)	0,5 V	0,5 V	0,5 V	0,5 V
Verstärkung (Strom)	60	60	60	200
Verstärkung (Spannung)	60	60	60	200
Widerstand (Shunt)	4 Ω	4 Ω	4 Ω	4 Ω
Widerstand (RC-Glied)	4 Ω	4 Ω	4 Ω	4 Ω
Kapazität (RC-Glied)	-	100 µF	100 µF	100 µF

Es wurden mehrere Messungen mit den in Tabelle 7.1 angegeben Werten durchgeführt. Die Kompensierung des Gleichanteils ist für die Messung jeweils einmal vor der Messung

bei einer Frequenz von 10 Hz durchgeführt worden. Die Aussteuerung wird durch den Algorithmus aus Kapitel 4.7.2 durchgeführt. Die Messung wurde mit den Parametern aus Tabelle 7.1 durchgeführt. Die Messungen zeigen qualitativ vergleichbare Ergebnisse. In den Abbildungen 7.5 sind die Messergebnisse im Bode-Diagramm dargestellt.

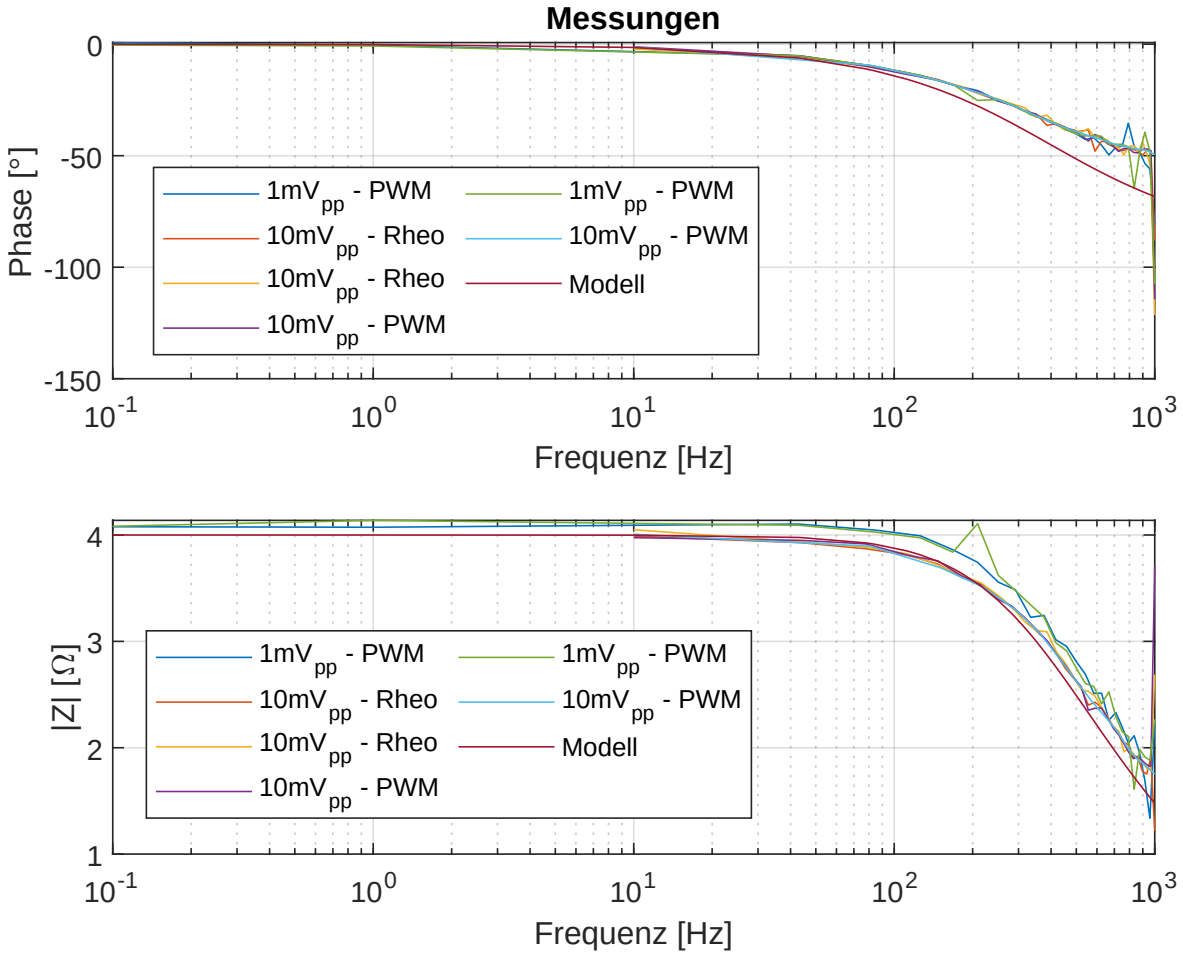


Abbildung 7.5: Die Abbildung zeigt die Messergebnisse der EIS für verschiedene Messungen.

7.5.2 Auswertung

Die Darstellung der Abweichung vom Erwartungswert wird in Abbildung 7.6 gezeigt. Ein signifikanter Unterschied zwischen den Messungen geht nicht hervor. Ein Unterschied zwischen Rheostat und PWM kann nicht festgestellt werden. Tendenziell ist zu erkennen,

dass kleinere Amplituden und damit einhergehend größere Verstärkungen schon bei kleineren Frequenzen Ausreißer aufweisen. Es besteht weiterhin ein systematischer Fehler, welcher abgesehen von Messausreißern ein relativ lineares Verhalten aufweist. Dies geht besonders aus der Abweichung der Phase hervor. Die Messungen liefern innerhalb der gegebenen Abweichungen ein reproduzierbares Ergebnis und können aufgrund des linearen Verhaltens mithilfe einer Ausgleichsfunktion korrigiert werden.

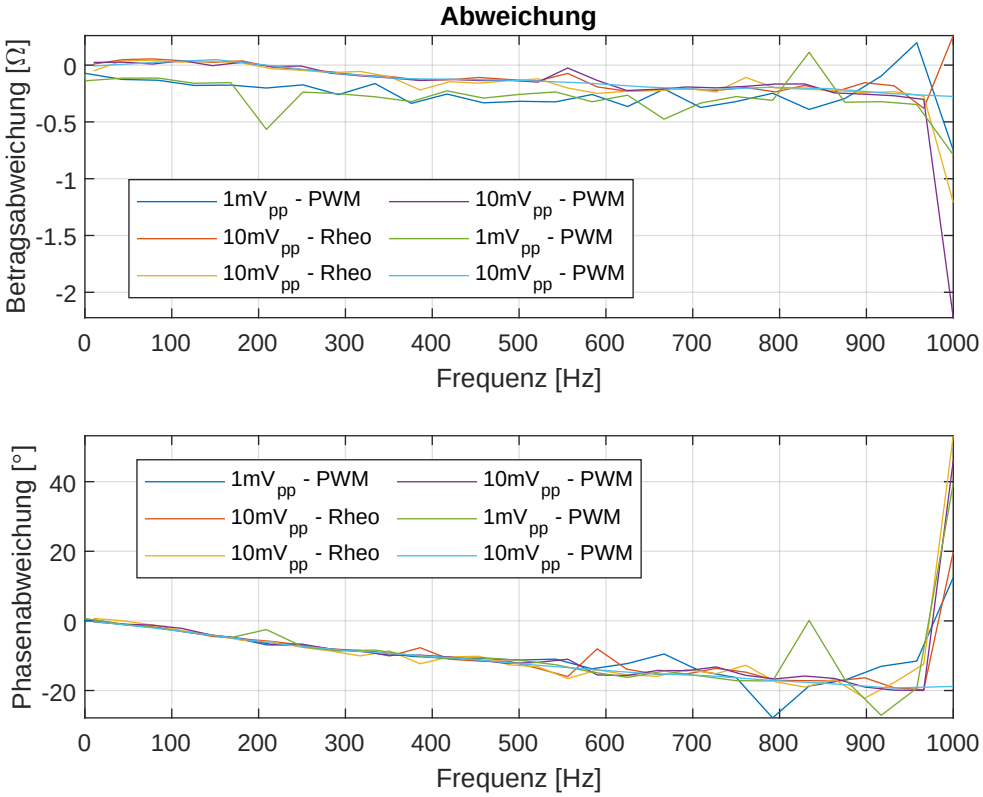


Abbildung 7.6: Die Abbildung zeigt die Abweichung der Messungen.

Exemplarisch sind alle gemessenen Phasenwerte eines Messvorgangs in Abbildung 7.7 dargestellt. Hierbei wird ersichtlich, dass die Messwerte Cluster in bestimmten Abständen bilden. Die Clusterbildung entsteht durch eine fehlerhafte Synchronisation zwischen den Sensorplatinen. Der Zusammenhang ergibt sich aus dem Verhältnis l/p aus Gleichung mat:periodengenau-freq in Kapitel 4.5.1, welches die Anzahl der Abtastpunkte pro Periode angibt. Die Cluster finden sich immer in einem Abstand von $\Delta\varphi_n = 10,8°$ zum nächsten Cluster wieder. Dies ergibt sich durch die folgende Gleichung:

$$\Delta\varphi_n = \frac{360° \cdot p}{l} = \frac{360° \cdot 15}{500} = 10,8° \tag{7.3}$$

Die fehlerhafte Synchronisation führt dazu, dass die Messungen nicht zeitgleich auf beiden Platinen starten. Durch den relativen zeitlichen Versatz um genau einen Datenpunkt entsteht in der Auswertung ein Fehler von genau $\Delta\varphi_n = 10,8°$. Hinzu kommt die Streuung innerhalb des Clusters. Weiterhin zeigt die Abbildung 7.7, dass die Streuung innerhalb der Cluster mit steigender Frequenz zunimmt. Hierdurch verzerrt die Bildung des Mittelwerts die Messung zusätzlich, wie besonders aus den Frequenzen 500 Hz bis 600 Hz, 900 Hz und 1000 Hz hervorgeht.

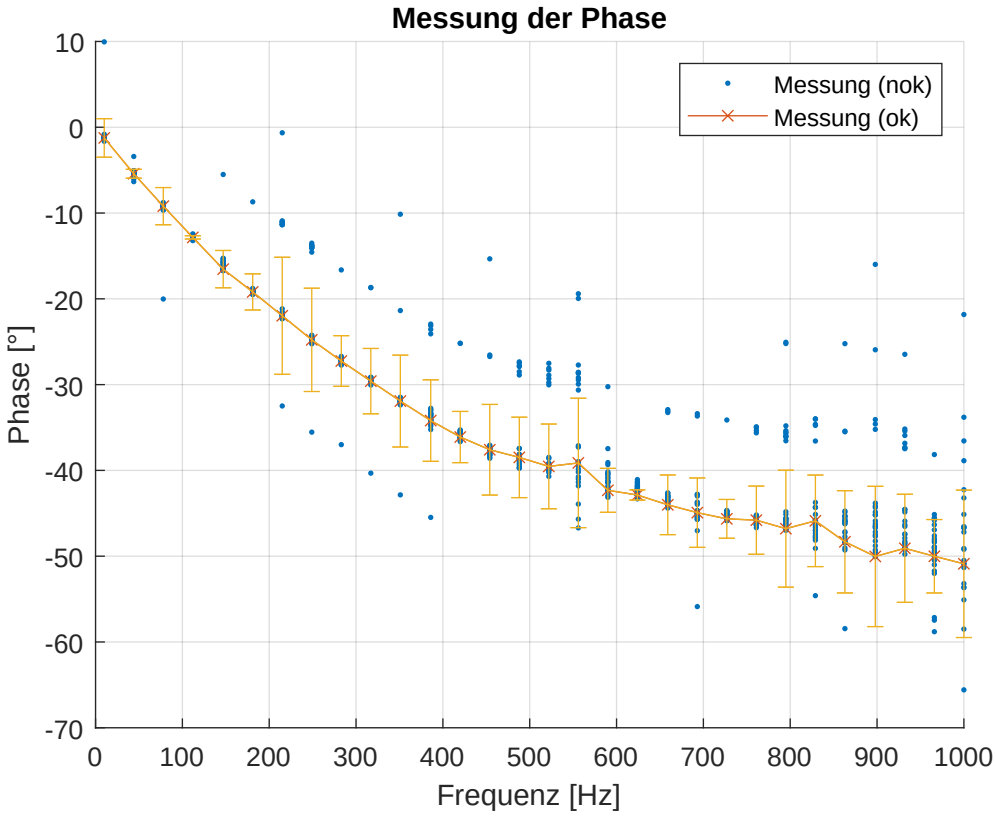


Abbildung 7.7: Die Abbildung zeigt die Messergebnisse der Phasenbestimmung. Deutlich zu sehen ist, dass die Messausreißer Cluster bilden und sich um ein Vielfaches von $\Delta\varphi_n$ vom Sollwert entfernt befinden. Die Streuung innerhalb der Cluster nimmt mit steigender Frequenz zu.

Die histogrammbasierte Messdatenvalidierung aus Kapitel 4.5.2 ist in der Lage, den Schwerpunkt der Messung zu bestimmen und Messwerte, welche weiter mehr als $0,5°$

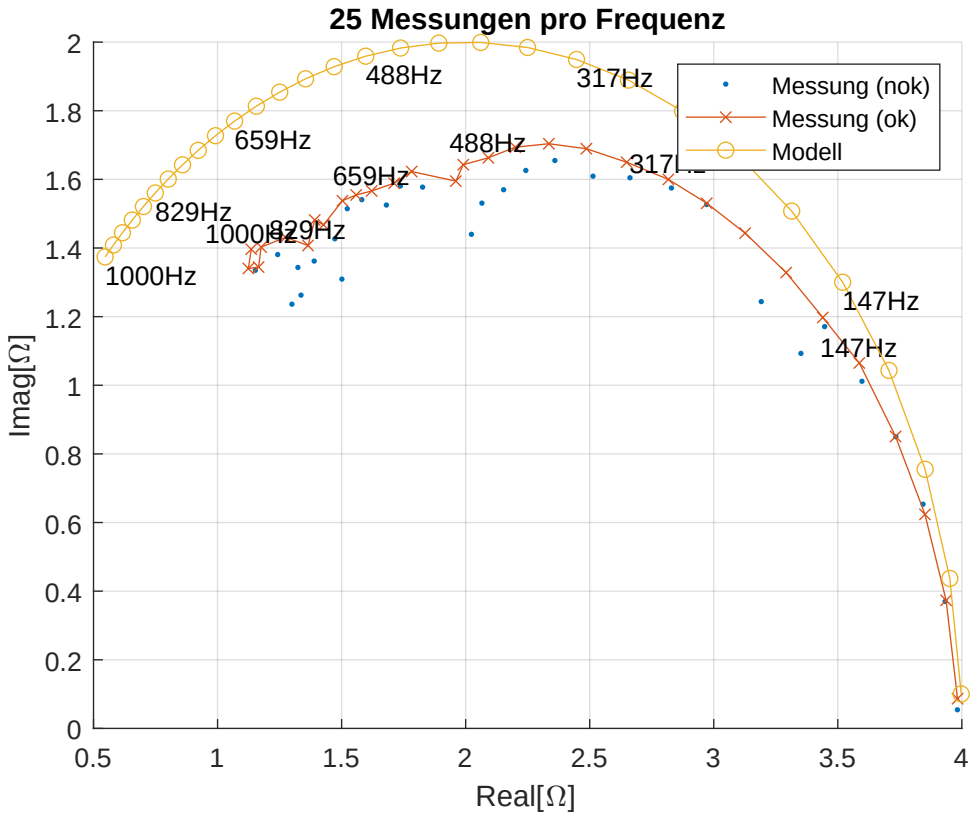


Abbildung 7.8: Die Abbildung zeigt die Messergebnisse in der Nyquistdarstellung und der modellbasierten Erwartung.

vom Schwerpunkt entfernt liegen, zu filtern. Anfällig ist die Methode hingegen bei zu großer Streuung der Messwerte. Bei besonders starker Streuung kann dies dazu führen, dass sich keine Messwerte innerhalb der gegebenen Grenzen befinden und die anschließende Berechnung des Mittelwerts nur einen einzelnen Wert berücksichtigt. Außerdem ist es denkbar, dass rechnerisch kein Schwerpunkt ermittelt werden kann, da jeder Klasse des Histogramms nur maximal einen Datenpunkt zugeordnet werden kann. Die Auswirkung auf die gemessene Impedanz wird aus Abbildung 7.8 deutlich. Vergleicht man die Darstellungen Abbildung 7.8 und Abbildung 7.7, es sich um dieselbe Messung. Es wird deutlich, dass die Impedanz in Real- und Imaginärteil dem Modell qualitativ zunächst folgt. Da die Messungen reproduzierbar sind, lässt sich der quantitative Fehler in diesem Bereich gut durch eine Ausgleichsfunktion korrigieren. Durch die erhöhte Streuung innerhalb der Cluster in den höheren Frequenzen ist eine allgemeine Korrektur nur schwer möglich.

In Abbildung 7.9 sind die Standardabweichungen der Messung mit und ohne histogrammbasierten Messdatensatzvalidierung (Kapitel 4.5.2) zu sehen. Alle durchgeführten Messungen wiesen hier ein ähnliches Verhalten auf, die Clusterbildung sowie die Streuung innerhalb der Cluster nehmen mit steigender Frequenz zu.

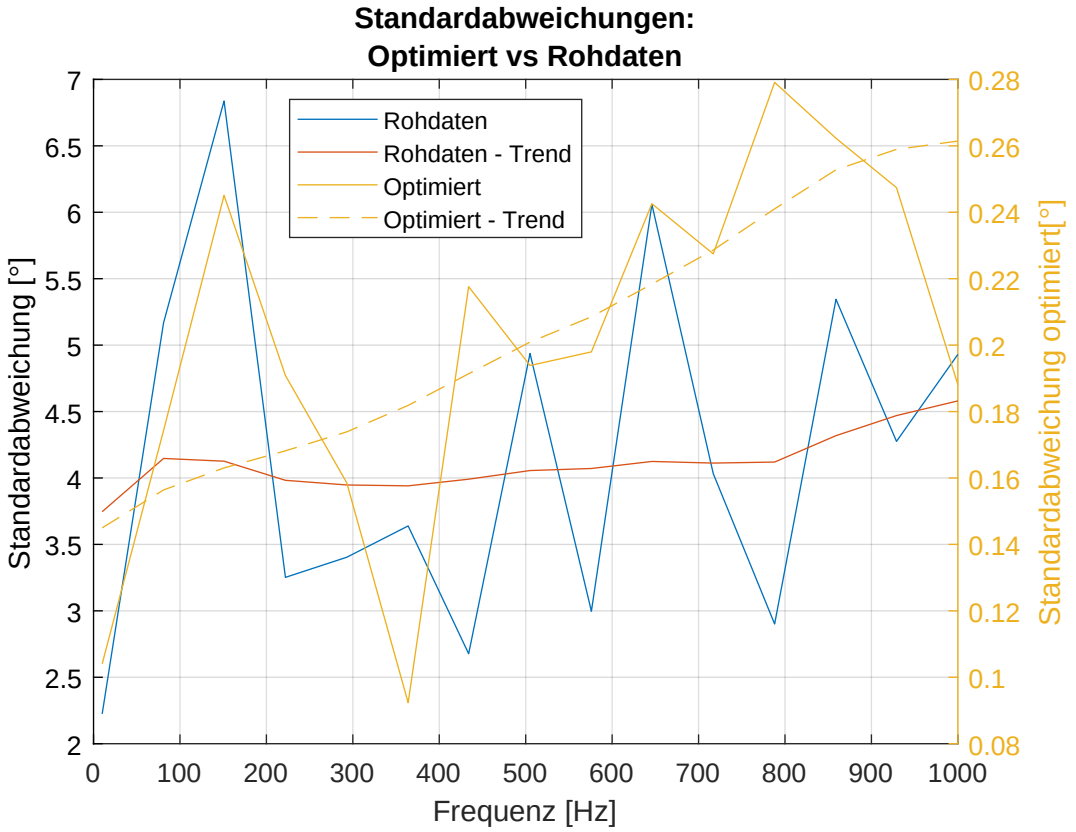


Abbildung 7.9: Die Abbildung zeigt die Standardabweichung der Messdaten vor und nach der histogrammbasierten Messdatensatzvalidierung. Die Zunahme der Streuung ist in der Standardabweichung deutlich zu sehen.

8 Zusammenfassung und Ausblick

In dem folgenden Kapitel wird die Arbeit zusammengefasst. Die in dieser Arbeit verwendeten Konzepte werden vorgestellt, anschließend erfolgt eine kurze Bewertung. Weiterhin werden die erreichten Ziele und umgesetzten Anforderung thematisiert. Abschließend erfolgt ein Ausblick auf potenzielle Weiterentwicklung des System und Punkte, die im Rahmen dieser Arbeit nicht behandelt werden konnten.

8.1 Bewertung der Ergebnisse

In dieser Arbeit ist ein Sensorsystem für die Durchführung einer elektrochemischen Impedanzspektroskopie an niederohmigen Batterien entwickelt worden. Neben den aufgestellten Anforderungen an das System, welche nur teilweise erfüllt werden konnten, war ein Ziel der Arbeit verschiedene Methoden zur Kompensierung des Gleichanteils zu untersuchen. Hierfür wurden zwei Methoden gegenübergestellt, wobei beide Methoden auf einer Subtrahiererschaltung basieren. Der Unterschied beider Ansätze liegt in der Erzeugung der zu subtrahierenden Referenzspannung. Der wesentliche Vorteil der Subtrahiererschaltung ist die Möglichkeit, auch besonders kleine Frequenzen messtechnisch zu verarbeiten. Schon bekannt ist der Ansatz über einen variablen Spannungsteiler mithilfe eines programmierbaren Rheostaten, neu ist die Erzeugung einer Referenzspannung mithilfe einer vom Mikrocontroller generierten PWM mit nachgeschaltetem Glättungsfilter. Hier stand infrage, ob das Signal aufgrund der Restwelligkeit nach dem Glätten bei sehr hohen Verstärkungsfaktoren eine ausreichende Stabilität zum Messen der Amplitude aufweist. Während der Arbeit hat sich gezeigt, dass die PWM dem Rheostat nicht nachsteht. Die PWM ist auch bei hohen Verstärkungen in der Lage, eine ausreichend stabile Referenzspannung zu erzeugen, um eine Messung durchzuführen. Die Messergebnisse zeigen, dass beide Methoden zur Kompensierung des Gleichanteils qualitativ ähnliche Ergebnisse liefern. Einen Rückschluss durch die Messergebnisse auf die eingesetzte Methode war in dem verwendeten Kontext nicht möglich. Es konnten keine signifikanten Unterschiede im

Ergebnis festgestellt werden. Die PWM ist aufgrund der einfachen und kostengünstigen Umsetzung zu bevorzugen. Auch der Programmieraufwand ist deutlich geringer, als dies beim Rheostaten der Fall ist.

Weiterhin ist der Einsatz von PGAs in diesem Kontext untersucht worden. Durch einen direkt programmierbaren Verstärker soll sichergestellt werden, dass das Signal immer ideal ausgesteuert ist. Der eingesetzte PGA MAX9939 bietet einen guten Kompromiss zwischen Flexibilität und einem hohen Verstärkungsfaktor. Trotz des hohen maximalen Verstärkungsfaktors eines einzelnen MAX9939, muss aufgrund der besonders kleinen Signalamplitude eine zweite Verstärkerstufe nachgeschaltet werden, um eine ausreichende Verstärkung gewährleisten zu können. Nachteilig am Einsatz der PGAs sind die nicht äquidistant verteilten Verstärkungsstufen. Umso größer der Verstärkungsfaktor, desto größer wird auch die Schrittweite, in welcher die Verstärkung eingestellt werden kann. Dies führt bei besonders großen Verstärkungsfaktoren zu einem nicht ideal ausgesteuerten ADC . Ist das Signal nicht optimal ausgesteuert, verringert sich die Auflösung und damit die erreichbare Genauigkeit des Gesamtsystems. Grundsätzlich erwiesen sich die PGAs als praktikabel, aufgrund der nicht äquidistant verteilten Verstärkungsfaktoren ist die Verwendung nicht in jedem Fall optimal.

Zur Beurteilung, ob das Signal durch die große Verstärkung den ADC bereits in Sättigung getrieben hat, wurde in dieser Arbeit das histogrammbasierte Sättigungskriterium vorgestellt. Die Grundannahme ist, dass das durch die notwendige Verstärkung auch die Rauschanteile des Messsignals mit großen Faktoren verstärkt werden. Das Sättigungskriterium soll eine gewisse Rauschtoleranz gewährleisten. Das histogrammbasierte Sättigungskriterium erlaubt durch eine Unterscheidung von Signal- und Rauschanteilen die Signalverstärkung bis in die Sättigung des ADC ohne signifikante Einbußen in der Genauigkeit. Der Ansatz macht sich hierbei die charakteristische Verteilung eines sinusförmigen Signals im Histogramm zunutze. Im Vergleich zu anderen Methoden ist diese Methode weniger rechenintensiv. Sie erwies sich im Rahmen dieser Arbeit als zuverlässig und praktikabel einsetzbar.

Das Gesamtsystem zeigte in den Messungen gute Ergebnisse. Die Messergebnisse zeigen, dass das System einen Proof-of-Concept Status erreicht hat und zeigt die Anwendbarkeit der eingesetzten Konzepte. Das System ist in der Lage, eine EIS durchzuführen. Die verteilte Systemarchitektur macht einen Einsatz im Elektrofahrzeug möglich. Messtechnisch weist das System noch Schwächen auf, besonders im Bezug auf die Synchronisation des Messvorgangs. Dies äußert sich in Abweichungen der Linearität der Messung. Das

System ist in der Lage, Messungen über den gesamten Spannungsbereich der Batteriezelle durchzuführen, den Gleichanteil vom Nutzsignal zu trennen und entsprechend nach der Verstärkung auswerten zu können. Auch die besonders kleine Amplitude von 1 mV konnte von dem System erfasst, ausreichend verstärkt und ausgewertet werden. Der Einsatz von zwei Verstärkern hat sich bei Amplituden von weniger als 10 mV als notwendig erwiesen.

Die gestellten Ziele aus Kapitel 4.2.2 konnten teilweise erfüllt werden:

- Nicht erreicht wurde die absolute Phasengenauigkeit von 1°. Es ist eine lineare Abweichung in Betrag und Phase über den Frequenzbereich festgestellt worden.
- Teilweise erreicht ist die Messung im Frequenzbereich von 1 mHz − 10 kHz. Es konnten Messungen im Bereich von 1 mHz−1 kHz durchgeführt werden, das System weist im hohen Frequenzbereich noch Schwächen auf. Die Streuung der Messwerte nimmt mit der Frequenz zu.
- Erreicht ist die Durchführung der Messung in dem gegebenen Spannungsbereich der Batteriezelle.
- Erreicht ist die Signalauflösung von 1% der Amplitude. Es war in jedem Fall möglich, eine Aussteuerung von mehr als 100 Stufen im ADC zu ermöglichen.
- Erreicht ist die Messung besonders kleiner Signale von 1 mV. Aufgrund der zwei Verstärkungsstufen und der genauen Kompensierung des Gleichanteils ist eine besonders hohe Verstärkung möglich.

8.2 Ausblick

Die folgenden offenen Punkte sind während der Erstellung der Arbeit im Besonderen aufgefallen und bieten noch Potenzial für weitere Untersuchungen an dem Entwickelten System.

8.2.1 Adaptive Anpassung der Verstärkung

Die Messungen zeigen neben der fehlerhaften Synchronisation einen linearen Fehler. Ein Ansatz, um diesem Fehler entgegenzuwirken, ist die adaptive Anpassung der Verstärkung. Die Messungen an rein ohmschen Widerständen zeigten, dass das System an sich keinen signifikanten Amplituden- oder Phasengang aufweist. Bei der Messung am RC-Glied ist jedoch ein deutliches Verhalten zu erkennen. Eine Vermutung ist, dass durch den Amplitudengang des RC-Glieds die Auflösung der Amplitude mit steigender Frequenz abnimmt und damit zu einer ungenaueren Messung führt. Dies betrifft ganz allgemein die Auflösung der Messamplitude, welche bei Messungen von Impedanzen mit steigender Frequenz in der aktuellen Umsetzung variiert. Hierfür muss der Messalgorithmus angepasst werden. In der aktuellen Implementierung wird vor einer Messung zunächst die Verstärkung eingestellt, anschließend wird mit der ermittelten Einstellung die Messung durchgeführt. Für eine adaptive Anpassung muss das Signal bei jeder Frequenz neu ausgesteuert bzw. die Verstärkung angepasst werden. Durch die Anpassung der Verstärkung bei jeder Frequenz wird sichergestellt, dass das Signal immer optimal ausgesteuert ist und entsprechend gut aufgelöst wird.

8.2.2 Kommunikation und Synchronisation

Während der Arbeit war eine besondere Herausforderung, eine stabile Kommunikation zwischen den Sensoren untereinander zu realisieren. Aufgrund der verschiedenen Potenziale, auf denen die Sensorik liegt, ist eine direkte Kommunikation via Kabelverbindung nicht möglich. Die Sensoren müssen galvanisch getrennt sein. In der aktuellen Implementierung wurden hierfür Optokoppler eingesetzt. Die in Kapitel 7.5.2 gezeigten Messungen weisen sprunghafte Abweichungen vom Sollwert auf, dies weist auf eine fehlerhafte Synchronisation hin. Der Grund für den zeitlichen Versatz konnte bis zum Abschluss der Arbeit nicht ermittelt werden. Während der Entwicklung wurden unterschiedliche Fehlerquellen untersucht und ggf. angepasst. Beachtung fanden dabei vor allem die folgenden beiden Punkte:

- Da das System Schwankungen im Zeitverhalten aufweist, wurde zunächst eine nicht echtzeitfähige Implementierung vermutet. Die Programmierung wurde auf ein Minimum reduziert, es ist auf eine echtzeitfähige Implementierung geachtet worden, um die mögliche zeitliche Varianz zu minimieren.

- Weiterhin wurde das Zeitverhalten der Optokoppler als mögliche Fehlerquelle identifiziert. Hier ist nach Messungen aufgefallen, dass die Optokoppler zwar synchron, jedoch nicht zeitgleich durchschalten. Die anfangs verwendeten Optokoppler das Typs LTV-817-A für größere Frequenzen ungeeignet. Es wurden folgende Typen verwendet:

  1. LTV-817-A
  2. ADUM1201ARZ
  3. TLP2348

Abschließend konnte nicht identifiziert werden, an welcher Stelle das fehlerhafte Zeitverhalten auftritt. Ein grundlegend anderer Ansatz der Kommunikation wird in [45] beschrieben. Der Ansatz beschreibt die Kommunikation zwischen den Sensoren mittels IrDA - Infrarot in einem Lichtleitkörper. Perspektivisch kann der Austausch der vorhandenen kommunikation zu einer Verbesserung des Zeitverhaltens führen.

Literaturverzeichnis

[1] : *Einfacher RC-Hochpass.* – URL https://de.wikipedia.org/wiki/Hochpass#/media/Datei:Hochpass.svg. – Zugriffsdatum: 04.06.2023

[2] : *Nyquist-Shannon-Abtasttheorem.* – URL https://de.wikipedia.org/wiki/Nyquist-Shannon-Abtasttheorem. – Zugriffsdatum: 23.07.2023

[3] : *Smart Analog Combo.* – URL https://www.ti.com/lit/wp/slaa835b/slaa835b.pdf?ts=1691942117975&ref_url=https%253A%252F%252Fduckduckgo.com%252F

[4] : *VDE - FAQ Batterie.* – URL https://www.vde.com/topics-de/energy/faq-batterien. – Zugriffsdatum: 11.08.2023

[5] Bard, AJ ; Faulkner, LR: 3. 5 MULTISTEP MECHANISMS. In: *ELECTROCHEMICAL METHODS Fundamentals and Applications 2nd ed.; Harris, D., Ed. JOHN WILEY and SONS, INC.: New York* (2001), S. 107–115

[6] Beelen, HPGJ ; Raijmakers, LHJ ; Donkers, MCF ; Notten, PHL ; Bergveld, HJ: A comparison and accuracy analysis of impedance-based temperature estimation methods for Li-ion batteries. In: *Applied Energy* 175 (2016), S. 128–140

[7] Berlin, Carsharing-Flotten am B. von: Carsharing und Elektromobilität in urbanen Räumen.

[8] BioLogic: Essential-potentiostats-web-081222. . – URL https://www.biologic.net/products/vsp/. – Zugriffsdatum: 29.07.2023

[9] Decives, Analog: *Data Sheet AD5270/AD5271.* – URL https://www.analog.com/media/en/technical-documentation/data-sheets/ad5270_5271.pdf. – Zugriffsdatum: 24.06.2023

[10] Deich, Tobias: *Optimierung der äußeren Druckbedingungen auf Li-Ionen Zellen zur Erhöhung der Lebensdauer*, Dissertation, 2022

[11] Devices, Analog: *LTC6915 Zero Drift, Precision Instrumentation Amplifier with Digitally Programmable Gain*. – URL https://www.analog.com/media/en/technical-documentation/data-sheets/6915fb.pdf. – Zugriffsdatum: 18.09.2023

[12] Digatron: *Electrochemical Impedance Spectroscopy EIS-Meter*. – URL https://www.digatron.com/Portals/38/Images/documents/PRODUKTBLATT_EIS_EN.PDF?ver=2019-06-12-091051-297. – Zugriffsdatum: 06.07.2023

[13] Dippon, Michael: *Bestimmung der Betriebsgrenzen für das Schnellladen von Lithium-Ionen Batterien*. Bd. 97. KIT Scientific Publishing, 2022

[14] Doppelbauer, Martin: Grundlagen der Elektromobilität. In: *Grundlagen der Elektromobilität* (2020)

[15] Fricke, Birger ; Sanders, Tilman ; Baumhofer, T ; Sauer, Dirk U. ; Wippermann, Klaus: Ortsaufgeloste Impedanzspektroskopie an Brennstoffzellen. In: *TECHNISCHE MITTEILUNGEN-ESSEN-* 99 (2006), Nr. 1/2, S. 231

[16] Gamry: *EIS-Box*. – URL https://www.gamry.com/potentiostats/eis-box/. – Zugriffsdatum: 06.07.2023

[17] Gellenbeck, Dennis: *Grundlagen und Technologien zur Primärgewinnung von Lithium und der Rückgewinnung aus Altbatterien*, Technische Universität Clausthal, Dissertation, 2021

[18] GmbH, FuelCon: *Data Sheet TrueData-EIS*. – URL https://www.horiba-fuelcon.com/en/product-data-sheets/. – Zugriffsdatum: 06.07.2023

[19] Goertzel, Gerald: An algorithm for the evaluation of finite trigonometric series. In: *American Math. Monthly* 65 (1958), S. 34–35

[20] Gould, Chris ; Wang, Jiabin ; Stone, Dave ; Foster, Martin: EV/HEV Li-ion battery modelling and State-of-Function determination. In: *International Symposium on Power Electronics Power Electronics, Electrical Drives, Automation and Motion*, 2012, S. 353–358

[21] Haussmann, Peter ; Melbert, Joachim: Spannungsgeregelte Impedanzspektroskopie mit breitbandigen Anregungssignalen für Lithium-Ionen-Zellen in Kfz-Anwendungen. In: *tm - Technisches Messen* 84 (2017), Nr. 6, S. 411–425. – URL https://doi.org/10.1515/teme-2017-0018

[22] Heil, Thomas ; Jossen, Andreas: Continuous approximation of the ZARC element with passive components. In: *Measurement Science and Technology* 32 (2021), jul, Nr. 10, S. 104011. – URL https://dx.doi.org/10.1088/1361-6501/ac0466

[23] Infineon: *XMC 2Go evaluation kit user guide*. – URL https://www.infineon.com/dgdl/Board_Users_Manual_XMC_2Go_Kit_with_XMC1100_R1.0.pdf?fileId=db3a3043444ee5dc014453d6c75078c6. – Zugriffsdatum: 16.07.2023

[24] Infineon: *XMC1100 AB-Step*. – URL https://www.infineon.com/dgdl/Infineon-xmc1100-AB_rm-UM-v01_03-EN.pdf?fileId=5546d46249cd1014014a0a8438a65e29. – Zugriffsdatum: 16.07.2023

[25] Instruments, Texas: *PGA280 Zerø-Drift, High-Voltage, Programmable Gain Instrumentation Amplifier*. – URL https://www.ti.com/lit/ds/symlink/pga280.pdf?ts=1695033379808&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fde-de%252FPGA280. – Zugriffsdatum: 18.09.2023

[26] Käbitz, Stefan ; Tübke, Jens ; Sauer, Dirk U.: Untersuchung der Alterung von Lithium-Ionen-Batterien mittels Elektroanalytik und elektrochemischer Impedanzspektroskopie / Institut für Stromrichtertechnik und Elektrische Antriebe. 2016. – Forschungsbericht

[27] Keil, Peter ; Jossen, Andreas: Aufbau und parametrierung von batteriemodellen. In: *19. DESIGN&ELEKTRONIK-Entwicklerforum Batterien & Ladekonzepte*, 2012

[28] Koch, Reinhold: *On-line Electrochemical Impedance Spectroscopy for Lithium-Ion Battery Systems*, Technische Universität München, Dissertation, 2017

[29] Köhler, Oskar L.: *Magnetische Nanopartikel für medizinische und weitere Anwendungen*, Universitätsbibliothek Mainz, Dissertation, 2017

[30] Köhler, Uwe ; Korthauer, Reiner (Hrsg.): *Aufbau von Lithium-Ionen-Batteriesystemen*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – URL https://doi.org/10.1007/978-3-642-30653-2_8. – ISBN 978-3-642-30653-2

[31] Kohs, Alexander: Batteriemodelle. In: *Batteriemodell zur Prädiktion des Gesundheitszustands von Lithium-Ionen-Batterien*. Springer, 2022, S. 47–64

[32] Leitl, Michael: *Strukturchemische und impedanzspektroskopische Untersuchungen an silberionenleitenden Substanzen, Münzmetallthiophosphaten und Kupferargyroditen*, Dissertation, 2008

[33] Li, Weiheng ; Huang, Qiu-An ; Yang, Changping ; Chen, Jian ; Tang, Zhepeng ; Zhang, Fangzhou ; Li, Aijun ; Zhang, Lei ; Zhang, Jiujun: A fast measurement of Warburg-like impedance spectra with Morlet wavelet transform for electrochemical energy devices. In: *Electrochimica Acta* 322 (2019), S. 134760. – URL https://www.sciencedirect.com/science/article/pii/S0013468619316317. – ISSN 0013-4686

[34] Locorotondo, Edoardo ; Cultrera, Vincenzo ; Pugi, Luca ; Berzi, Lorenzo ; Pierini, Marco ; Lutzemberger, Giovanni: Development of a battery real-time state of health diagnosis based on fast impedance measurements. In: *Journal of Energy Storage* 38 (2021), S. 102566. – URL https://www.sciencedirect.com/science/article/pii/S2352152X2100311X. – ISSN 2352-152X

[35] Möller, Kai-Christian: *Übersicht über die Speichersysteme/Batteriesysteme*. Springer, 2013

[36] NXP, Datang: *DNB1168*. – URL https://www.datangnxp.com/en/details/products/43. – Zugriffsdatum: 06.07.2023

[37] Oppenheim, Alan V. ; Schafer, Ronald W. ; Buck, John R.: *Zeitdiskrete Signalverarbeitung*. Bd. 2. Oldenbourg München Wien, 1992

[38] Products, Maxim I.: *SPI Programmable-Gain Amplifier with Input VOS Trim and Output Op Amp*. – URL {https://www.analog.com/en/products/max9939.html}. – Zugriffsdatum: 24.06.2023

[39] Reif, Konrad ; Kannenberg, Christian: *Batterien, Bordnetze und Vernetzung*. Springer, 2010

[40] Richardson, Robert R. ; Ireland, Peter T. ; Howey, David A.: Battery internal temperature estimation by combined impedance and surface temperature measurement. In: *Journal of Power Sources* 265 (2014), S. 254–261. – URL https://www.sciencedirect.com/science/article/pii/S0378775314006302. – ISSN 0378-7753

[41] Roth, Joachim G.: *Impedanzspektroskopie als Verfahren zur Alterungsanalyse von Hochleistungs-Lithium-Ionen-Zellen*. Verlag Dr. Hut, 2013

[42] Sassano, Nico: *Entwicklung eines Messsystems zur funksynchronisierten elektrochemischen Impedanzspektroskopie an Batterie-Zellen*, Fachhochschule Westküste Fachbereich u. Hochschule für Angewandte Wissenschaften Hamburg Fakultät Technik und Informatik Department Informations- und Elektrotechnik, Diplomarbeit, 2015

[43] Sassano, Nico ; Roscher, Valentin ; Riemschneider, Karl-Ragmar: Verteilte Sensor-Signalverarbeitung für die Impedanzspektroskopie von vielzelligen Batterien. In: *14. GI/ITG KuVS Fachgespräch Sensornetze* (2015), S. 5

[44] Sauer, Dirk U.: Grundlagen der Impedanzspektroskopie fur die Charakterisierung von Batterien. In: *Technische Mitteilungen* 99 (2006), Nr. 1, S. 74–80

[45] Schiepel, Philipp ; Ernsting, Jonas ; Rittweger, Florian ; Müller, Günter ; Riemschneider, Karl-Ragmar ; Modrzynski, Christian: Measurement data communication within vehicle battery modules by plastic light guide bodies. In: *Batterietagung 2021* Haus der Technik Essen HDT (Veranst.), 2021, S. 1–8

[46] Schmidt, Jan P. ; Hammerschmidt, Thomas: *Impedanzsensorik für Batteriezellen in Elektro-Fahrzeugen.* S. 99–126. In: Tille, Thomas (Hrsg.): *Automobil-Sensorik 2: Systeme, Technologien und Applikationen.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2018. – URL https://doi.org/10.1007/978-3-662-56310-6_5. – ISBN 978-3-662-56310-6

[47] Schulze, Olaf: Elektrofahrzeug und Elektromobilität. In: *Elektromobilität–ein Ratgeber für Entscheider, Errichter, Betreiber und Nutzer: Facetten zu Ladeinfrastruktur, Subventionsregeln, Kosten und Handling.* Springer, 2022, S. 9–68

[48] Shen, Ping ; Ouyang, Minggao ; Lu, Languang ; Li, Jianqiu: State of Charge, State of Health and State of Function Co-Estimation of Lithium-Ion Batteries for Electric Vehicles. In: *2016 IEEE Vehicle Power and Propulsion Conference (VPPC)*, 2016, S. 1–5

[49] Steinhauer, Miriam ; Stich, Michael ; Kurniawan, Mario ; Seidlhofer, Beatrix-Kamelia ; Trapp, Marcus ; Bund, Andreas ; Wagner, Norbert ; Friedrich, K A.: In situ studies of solid electrolyte interphase (SEI) formation on crystalline carbon surfaces by neutron reflectometry and atomic force microscopy. In: *ACS applied materials & interfaces* 9 (2017), Nr. 41, S. 35794–35801

[50] Tektronix: *AFG1000 Programming Manual*. – URL https://download.tek.com/manual/AFG1000-Programmer-Manual-EN-077112900.pdf. – Zugriffsdatum: 18.09.2023

[51] Tektronix: *AFG1000 Series Datasheet*. – URL https://download.tek.com/datasheet/AFG1000_Series_Function_Generator_Datasheet-EN_US_75W-60160-05.pdf. – Zugriffsdatum: 14.09.2023

[52] Telmasre, Tushar ; Goswami, Neha ; Concepción, Anthony ; Kolluri, Suryanarayana ; Pathak, Manan ; Morrison, Gerald ; Subramanian, Venkat R.: Impedance response simulation strategies for lithium-ion battery models. In: *Current Opinion in Electrochemistry* (2022), S. 101140

[53] Thomas Günther, Paul Büschel und Olfa K.: *Eingebettetes Impedanzmesssystem für das Batteriemanagement in Elektrofahrzeugen*. tm – Technisches Messen 2014, 2014. – URL https://books.google.de/books?id=bMWkPQAACAAJ. – ISBN DOI 10.1515/teme-2014-1052

[54] Tietze, U. ; Schenk, C.: *Halbleiter-Schaltungstechnik*. Springer, 2002. – URL https://books.google.de/books?id=i2_iwAEACAAJ. – ISBN 9783540428497

[55] Tille, Thomas: *Automobil-Sensorik*. Springer, 2016

[56] Tröltzsch, Uwe ; Kanoun, Olfa ; Tränkler, Hans-Rolf: Characterizing aging effects of lithium ion batteries by impedance spectroscopy. In: *Electrochimica Acta* 51 (2006), Nr. 8, S. 1664–1672. – URL https://www.sciencedirect.com/science/article/pii/S0013468605007899. – Electrochemical Impedance Spectroscopy. – ISSN 0013-4686

[57] Vedalakshmi, R ; Saraswathy, V ; Song, Ha-Won ; Palaniswamy, N: Determination of diffusion coefficient of chloride in concrete using Warburg diffusion coefficient. In: *Corrosion Science* 51 (2009), Nr. 6, S. 1299–1307

[58] Werner, Martin: *Digitale Signalverarbeitung mit MATLAB*. Springer, 2012

[59] Winkler, Volker: *Oberflächenanalytische Charakterisierung der SEI auf Graphit Anodenschichten in Lithium Ionen Batterien*, Dissertation, Albert-Ludwigs-Universität Freiburg, Dissertation, 2016

[60] Wippermann, Klaus ; Kulikovsky, Andrej A. ; Schmitz, Heinz ; Mergel, Jürgen ; Fricke, Birger ; Sanders, Tilman ; Sauer, Dirk U. ; Elemente, Brennstoffzellen sind galvanische: Grundlagen zur Impedanzspektroskopie an Brennstoffzellen und ortsaufgeloste Messungen. In: *TECHNISCHE MITTEILUNGEN-ESSEN-* 99 (2006), Nr. 1/2, S. 96

[61] Zappen, Hendrik ; Ringbeck, Florian ; Sauer, Dirk U.: Application of Time-Resolved Multi-Sine Impedance Spectroscopy for Lithium-Ion Battery Characterization. In: *Batteries* 4 (2018), Nr. 4. – URL https://www.mdpi.com/2313-0105/4/4/64. – ISSN 2313-0105

[62] Zhang, Ming ; Liu, Yanshuo ; Li, Dezhi ; Cui, Xiaoli ; Wang, Licheng ; Li, Liwei ; Wang, Kai: Electrochemical Impedance Spectroscopy: A New Chapter in the Fast and Accurate Estimation of the State of Health for Lithium-Ion Batteries. In: *Energies* 16 (2023), Nr. 4. – URL https://www.mdpi.com/1996-1073/16/4/1599. – ISSN 1996-1073

A Anhang

A.1 Inbetriebnahme

Name	Funktion	Vorgehen	Testdaten	Validierung	Erfüllt
IB0	Gleichanteil-kompensie-rung	Batteriespannungsbereich durchfahren und den Gleich-anteil von Dem Sensor kompensieren lassen. Jeweils via PWM und Rheostat.	Batteriespannung: 1.8V – 4.2V Schrittweite: 0.1V	Jede eingestellte Spannung muss vom Sensor auf ca. 1mV Kompensiert werden können.	Ja
IB1	Frequenzbereich	Frequenzbereich durchfahren und in MATLAB die mithil-fe des Goertzelalgorithmus die Frequenz berechnen	Frequenzbereich: 100mHz – 1kHz Schrittweite: Linear	Eingestellter mit berechneter Frequenz vergleichen	Ja
IB2	Verstärkung	Kleinste mögliche Amplitude des Laboraufbaus einstellen und von der Sensorplatine ver-stärken lassen.	Amplitude 1mVpp	Die Amplitude muss mindes-tens 100 Quantisierungsstufen ausgesteuert sein. (Entspricht Vpp ca. 100mV) Die einge-stellte Frequenz muss korrekt erfasst sein.	Ja
IB3	Ansteuerung digi-taler Bauteile	Die digitalen Bauteile müssen mit dem XMC100 kommuni-zieren können und die gegebe-nen Werte einstellen können. Dies wird implizit durch IB0, IB1 und IB2 getestet.		Implizit durch IB0-2	Ja

Tabelle A.1: Inbetriebnahme der Sensorplatine.

A.2 Hardware

A.2.1 Sensor - Schaltplan

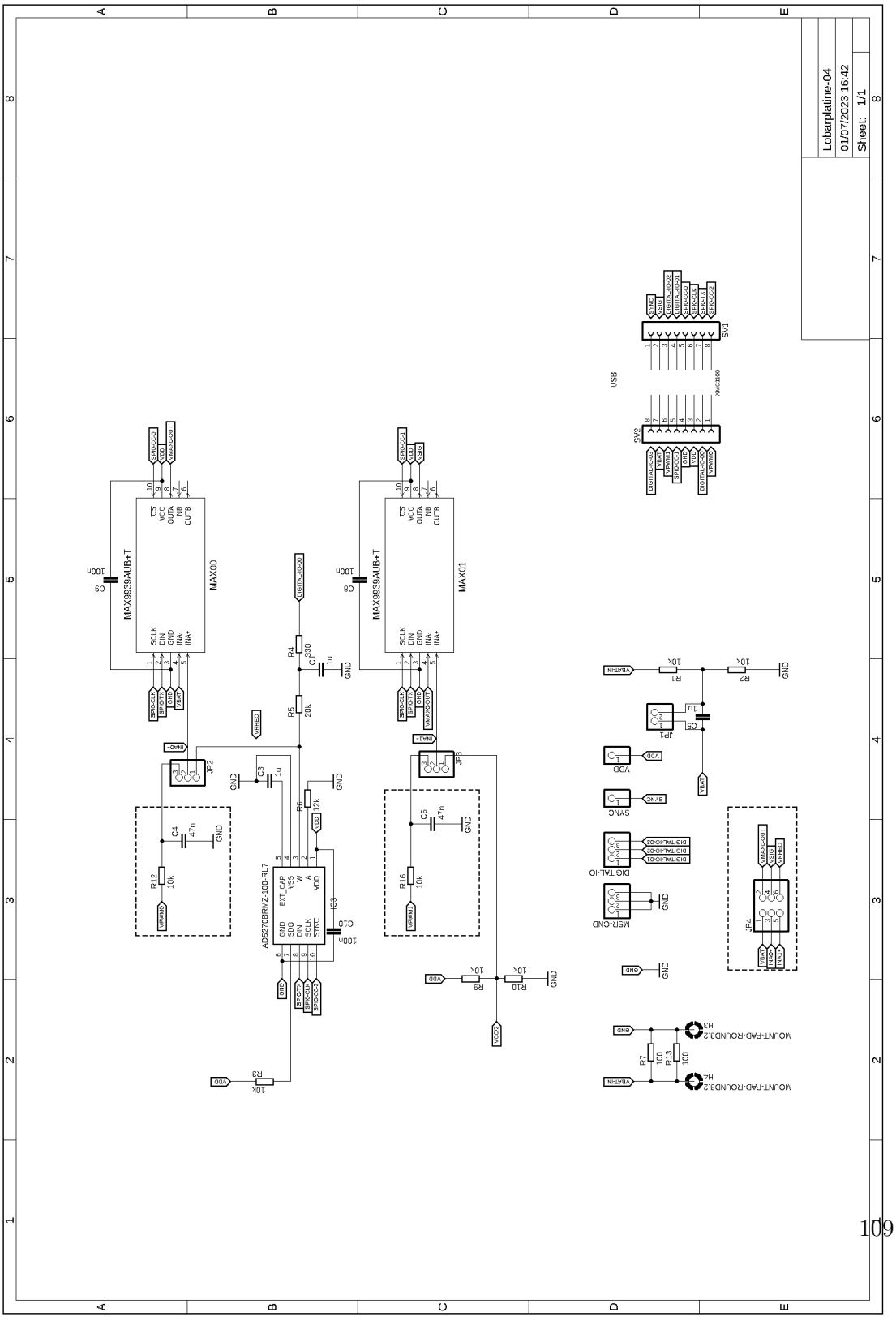


Abbildung A.1: Die Abbildung zeigt den Schaltplan der Sensorplatine.

A.2.2 Sensor - BOM

Part	Value	Device	Package	Description
C1	1u	C-EUC0805	C0805	CAPACITOR, European symbol
C2	1n	C-EUC0805	C0805	CAPACITOR, European symbol
C3	1u	C-EUC0805	C0805	CAPACITOR, European symbol
C4	47n	C-EUC0805	C0805	CAPACITOR, European symbol
C5	1u	C-EUC0805	C0805	CAPACITOR, European symbol
C6	47n	C-EUC0805	C0805	CAPACITOR, European symbol
C8	100n	C-EUC0805	C0805	CAPACITOR, European symbol
C9	100n	C-EUC0805	C0805	CAPACITOR, European symbol
C10	100n	C-EUC0805	C0805	CAPACITOR, European symbol
DIGITAL-IO		PINHD-1X3	1X03	PIN HEADER
H3	MOUNT-PAD-ROUND3.2	MOUNT-PAD-ROUND3.2	3,2-PAD	MOUNTING PAD, round
H4	MOUNT-PAD-ROUND3.2	MOUNT-PAD-ROUND3.2	3,2-PAD	MOUNTING PAD, round
IC3	AD5270BRMZ-100-RL7	AD5270BRMZ-100-RL7	SOP50P490X110-10N	1024-/256-Position 1% Resistor Tolerance Error SPI Interface and 50-TP Memory Digital Rheostat
JP1		PINHD-1X2	1X02	PIN HEADER
JP2		PINHD-1X3	1X03	PIN HEADER
JP3		PINHD-1X3	1X03	PIN HEADER
JP4		PINHD-2X3	2X03	PIN HEADER
MAX00	MAX9939AUB+T	MAX9939AUB+T	21-0061L_MXM	
MAX01	MAX9939AUB+T	MAX9939AUB+T	21-0061L_MXM	
MSR-GND		PINHD-1X3	1X03	PIN HEADER
R1	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R2	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R3	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R4	330	R-EU_R0805	R0805	RESISTOR, European symbol
R5	20k	R-EU_R0805	R0805	RESISTOR, European symbol
R6	12k	R-EU_R0805	R0805	RESISTOR, European symbol
R7	100	R-EU_R0805	R0805	RESISTOR, European symbol
R9	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R10	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R12	10k	R-EU_R0805	R0805	RESISTOR, European symbol
R13	100	R-EU_R0805	R0805	RESISTOR, European symbol
R16	10k	R-EU_R0805	R0805	RESISTOR, European symbol
SV1		FE08-1	FE08	FEMALE HEADER
SV2		FE08-1	FE08	FEMALE HEADER
SYNC		PINHD-1X1	1X01	PIN HEADER
VDD		PINHD-1X1	1X01	PIN HEADER

APP Instance Name	Resource	Port-Pin	Pin Number
ADC_MEASUREMENT_0	pin_VSIG	(P2.0)	#1
DIGITAL_IO_0	pin	(P2.10)	#7
DIGITAL_IO_CC0	pin	(P0.9)	#20
DIGITAL_IO_CC1	pin	(P2.11)	#8
DIGITAL_IO_CC2	pin	(P0.6)	#17
DIGITAL_IO_CLK	pin	(P0.8)	#19
DIGITAL_IO_MISO	pin	(P0.7)	#18
DIGITAL_IO_REF	pin	(P0.0)	#15
DIGITAL_IO_SYNC	pin	(P1.0)	#14
PIN_INTERRUPT_0	Pin	(P2.7 ,P2.8)	#5
PIN_INTERRUPT_1	Pin	(P2.6)	#4
PWM_CCU4_0	PWM_CCU4 Channel Out	(P0.5)	#16
UART_0	Receive Pin	(P2.2)	#3
UART_0	Transmit Pin	(P2.1)	#2

A.2.3 XMC1100 - Pinout

A.2.4 XMC1100 - Resource Mapping

App Instance Name	Resource	Mapped Resource
ADC_MEASUREMENT_0	Background	vadc/0/backgnd
ADC_MEASUREMENT_0	Background class group 0	vadc/0/class/0
ADC_MEASUREMENT_0	Background class group 1	vadc/0/class/1
ADC_MEASUREMENT_0	Global Result Register	vadc/0/global_result
ADC_MEASUREMENT_0	VSIG	vadc/0/group/0/ch/5
ADC_MEASUREMENT_0	VSIG_pin	p/2/pad/0
CCU4_SLICE_CONFIG_0	CCU40 Config	ccu4/0/cc4/1
CLOCK_XMC1_0	CLOCK CONTROL UNIT	scu/0/ccu/config
CLOCK_XMC1_0	DCO CLOCK	scu/0/dco/2
CPU_CTRL_XMC1_0	hardfault_exception	cpu/0/exception/hardfault
CPU_CTRL_XMC1_0	swd0_pin0	p/0/pad/14
CPU_CTRL_XMC1_0	swd0_pin1	p/0/pad/15
DIGITAL_IO_0	pin	p/2/pad/10
DIGITAL_IO_CC0	pin	p/0/pad/9
DIGITAL_IO_CC1	pin	p/2/pad/11
DIGITAL_IO_CC2	pin	p/0/pad/6
DIGITAL_IO_CLK	pin	p/0/pad/8
DIGITAL_IO_MISO	pin	p/0/pad/7
DIGITAL_IO_REF	pin	p/0/pad/0
DIGITAL_IO_SYNC	pin	p/1/pad/0
GLOBAL_ADC_0	Global	vadc/0/global
GLOBAL_ADC_0	Group0	vadc/0/group/0/config
GLOBAL_CCU4_0	CCU4 sync start	scu/0/gcu/ccu4_global_enable/0
GLOBAL_CCU4_0	Global	ccu4/0/global
PIN_INTERRUPT_0	External Event	cpu/0/nvic/interrupt/3
PIN_INTERRUPT_0	Pin	p/2/pad/8
PIN_INTERRUPT_0	ers_etl	eru/0/ers_etl/3
PIN_INTERRUPT_0	ogu	eru/0/ogu/0
PIN_INTERRUPT_1	External Event	cpu/0/nvic/interrupt/4
PIN_INTERRUPT_1	Pin	p/2/pad/6
PIN_INTERRUPT_1	ers_etl	eru/0/ers_etl/2
PIN_INTERRUPT_1	ogu	eru/0/ogu/1
PWM_CCU4_0	CC4 Config	ccu4/0/cc4/0
PWM_CCU4_0	PWM_CCU4 Channel Out	p/0/pad/5
TimerSWISR	NVIC Node	cpu/0/nvic/interrupt/21
UART_0	Channel	usic/0/channel/0

A.2.5 XMC1100 - Signal Assignment

APP Instance Name	Signal	APP Instance Name	
ADC_MEASUREMENT_0	pad_signal_VSIG	ADC_MEASUREMENT_0	
ADC_MEASUREMENT_0	pad_signal_VSIG	ADC_MEASUREMENT_0	
ADC_MEASUREMENT_0	VSIG_pin_signal	ADC_MEASUREMENT_0	
CCU4_SLICE_CONFIG_0	event_cmp_match	TimerSWISR	
CLOCK_XMC1_0	clk_dco2_output	CLOCK_XMC1_0	
DIGITAL_IO_0	pin	DIGITAL_IO_0	
DIGITAL_IO_0	pin_signal	DIGITAL_IO_0	
DIGITAL_IO_CC0	pin	DIGITAL_IO_CC0	
DIGITAL_IO_CC0	pin_signal	DIGITAL_IO_CC0	
DIGITAL_IO_CC1	pin	DIGITAL_IO_CC1	
DIGITAL_IO_CC1	pin_signal	DIGITAL_IO_CC1	
DIGITAL_IO_CC2	pin	DIGITAL_IO_CC2	
DIGITAL_IO_CC2	pin_signal	DIGITAL_IO_CC2	
DIGITAL_IO_CLK	pin	DIGITAL_IO_CLK	
DIGITAL_IO_CLK	pin_signal	DIGITAL_IO_CLK	
DIGITAL_IO_MISO	pin	DIGITAL_IO_MISO	
DIGITAL_IO_MISO	pin_signal	DIGITAL_IO_MISO	
DIGITAL_IO_REF	pin	DIGITAL_IO_REF	
DIGITAL_IO_REF	pin_signal	DIGITAL_IO_REF	
DIGITAL_IO_SYNC	pin	DIGITAL_IO_SYNC	
DIGITAL_IO_SYNC	pin_signal	DIGITAL_IO_SYNC	
GLOBAL_ADC_0	global_signal	ADC_MEASUREMENT_0	backg
GLOBAL_CCU4_0	ccu4_global	PWM_CCU4_0	
GLOBAL_CCU4_0	ccu4_global	CCU4_SLICE_CONFIG_0	
PIN_INTERRUPT_0	external_event_pin	PIN_INTERRUPT_0	
PIN_INTERRUPT_0	external_event_pin	PIN_INTERRUPT_0	
PIN_INTERRUPT_0	iout	PIN_INTERRUPT_0	
PIN_INTERRUPT_0	Pin_signal	PIN_INTERRUPT_0	
PIN_INTERRUPT_0	trigger_out	PIN_INTERRUPT_0	
PIN_INTERRUPT_1	external_event_pin	PIN_INTERRUPT_1	
PIN_INTERRUPT_1	external_event_pin	PIN_INTERRUPT_1	
PIN_INTERRUPT_1	iout	PIN_INTERRUPT_1	
PIN_INTERRUPT_1	Pin_signal	PIN_INTERRUPT_1	
PIN_INTERRUPT_1	trigger_out	PIN_INTERRUPT_1	
PWM_CCU4_0	PWM_CCU4 Channel Out	PWM_CCU4_0	P
PWM_CCU4_0	PWM_CCU4 Channel Out_signal	PWM_CCU4_0	
PWM_CCU4_0	pwm_output	PWM_CCU4_0	
UART_0	dout0_output	UART_0	
UART_0	event_protocol	UART_0	
UART_0	Receive Pin_signal	UART_0	
UART_0	rxd_pin	UART_0	
UART_0	rxd_pin	UART_0	

A.2.6 MATLAB - Quellcode

Main

```
% ************************************************
% EIS-sensor-system-automated-measurement
% function generator: Tektronix AFG11062
% author: Tobias Frahm <tobias.frahm@haw-hamburg.de>
% ************************************************
import SensorHandler.*
import PGA.*
import Offset.*
import Controller.*

%% Initialisierung
clear all; close all;
PGAs = [PGA.MAX0, PGA.MAX1];

voltageMeasure = SensorHandler('COM7', 'Voltage');
currentMeasure = SensorHandler('COM4', 'Current');
voltageMeasure.enableADC();

for i = length(PGAs)
    voltageMeasure.setGain(1);
    currentMeasure.setGain(1);

    voltageMeasure.setOS(0, PGAs(i));
    currentMeasure.setOS(0, PGAs(i));

    voltageMeasure.setPWM(50, PGAs(i));
    currentMeasure.setPWM(50, PGAs(i));
end

%% Ausregelung und Verstärkung
% AWG connect
try
    fclose(instrfind);
    delete(instrfind);
catch clc
end

[iObj, AWG] = connect_funcgen();
connect(AWG);
```

```
offset_chan_1 = 0.5;
l = 500;
p = 15;
f = 10;
fs = l*f/p;
set(AWG.Frequency(1),'Frequency', f);
set(AWG.Voltage(1), 'Amplitude', 0.001)
set(AWG.Waveform(1),'Shape','sin');
fwrite(iObj,['SOURce1:VOLTage:LEVel:IMMediate:OFFSet
↪  ',mat2str(offset_chan_1),'V'],'int8')
set(AWG.Frequency(2), 'Frequency', fs);

set(AWG.Output(1),'State','on');
set(AWG.Output(2),'State','on');


% Regelung Offset/GAIN
clt_curr = Controller(currentMeasure);
clt_curr = clt_curr.run();
currentMeasure.gain = clt_curr.GAIN(clt_curr.gain_idx);

clt_volt = Controller(voltageMeasure);
clt_volt = clt_volt.run();
voltageMeasure.gain = clt_volt.GAIN(clt_volt.gain_idx);

gains = [clt_volt.GAIN(clt_volt.gain_idx), clt_curr.GAIN(clt_curr.gain_idx)];
set(AWG.Output(1),'State','off');

%% Messung
freqs = [0.1, round(linspace(1, 1000, 25))]; % [1, 12, 23, 31, 47, 51, 65,
↪  78, 88, 101, 212, 223, 231, 247, 251, 265, 278, 288, 312, 323, 331, 347,
↪  351, 365, 378, 388, 412, 423, 431, 447, 451, 465, 478, 488, 512, 523,
↪  531, 547, 551, 565, 578, 588];
freqs = (unique(freqs));

clear impedan phase
close all;
impedan = [];
cnt = 240;
n = 3;
nn = n;
phase = [];
T = 0;
set(AWG.Output(1),'State','on');
```

```
80
81  if exist('results', 'var')
82      L = length(results)-1;
83  else
84      results = {};
85      L = 1;
86  end
87  pold = 0;
88  for j=n:nn
89      for ff = L:length(freqs)
90          results{ff,4} = [];
91          disp("[" + string(datetime("now")) + "] " + freqs(ff) + "Hz")
92          while isempty(results{ff,4})
93              [impedan, phase, vol, curr] = measure(freqs(ff), 25,
                ↪  voltageMeasure, currentMeasure, iObj, AWG, pold);
94              results{ff,1} = freqs(ff);
95              results{ff,2} = impedan;
96              results{ff,3} = phase;
97              results(ff,6) = {vol};
98              results(ff,7) = {curr};
99              [results{ff,4}, results{ff,5}] = sample_validation(results{ff,2},
                ↪  results{ff,3});
100             pold = mean(results{ff,5});
101             disp("[" + string(datetime("now")) + "] " + mean(results{ff,4}) +
                ↪  "Ohm");
102             disp("[" + string(datetime("now")) + "] " + mean(results{ff,5}) +
                ↪  "°");
103         end
104         %plot_histogramm(results{ff,4}, results{ff,5}, freqs(ff));
105     end
106     no = cnt + j;
107     filename = "G:\Meine Ablage\DATA\eis-100yF-25mean-" + no + ".mat";
108     save(filename, "results")
109     clear results
110 end
111 clear cnt j
112 set(AWG.Output(1),'State','off');
113 set(AWG.Output(2),'State','off');
114
115 %% Nyquist Graph
116 close all;
117 figureNo = numel(findall(0, 'Type', 'figure'));
118 if ~figureNo
119     figureNo = 1;
```

```
120 end
121
122
123 for ff = 1:length(freqs)
124     figure(figureNo + 1);
125     re = real(mean(results{ff,4}));
126     im = (imag(mean(results{ff,4})));
127     plot(re, -im, 'x')
128     hold on;
129     plot(real(mean(results{ff,2})), -(imag(mean(results{ff,2}))), '.')
130     hold on;
131     xlabel('Real[Ohm]')
132     ylabel('Imag[Ohm]')
133     grid("on");
134     annotationText = results{ff,1} + "Hz";
135     text(re, -im, annotationText, 'HorizontalAlignment', 'left',
        ↪  'VerticalAlignment', 'bottom');
136     figure(figureNo + 2);
137     plot(results{ff,1}, results{ff,5}, 'x')
138     hold on;
139     plot(results{ff,1}, results{ff,3}, '.')
140     hold on;
141     xlabel('Freq [Hz]')
142     ylabel('Phase')
143     grid("on");
144     title("Phase [°]")
145 end
146 hold off;
147 hold off;
148
149
150 %% Plot
151
152 x = linspace(0,1,length(buffer{1}));
153 f1 = figure(ff);
154 tiledlayout(f1, 2, 3);
155
156 nexttile;
157 plot(x, buffer{1}); hold on;
158 plot(x, buffer{2}); hold off;
159 legend('Spannung', 'Strom')
160 title("Signal " + ff)
161 grid("on");
162 xlabel("Zeit [s]")
```

```
163 ylabel("Amplitude [V]")
164
165 nexttile;
166 h = histogram(measures{1}, 100); hold on;
167 h = histogram(measures{2}, 100); hold off;
168 legend('Spannung', 'Strom')
169 title("Histogramm")
170
171 grid on;
172 nexttile;
173 g = semilogy(f, ffts{1}); hold on;
174 g = semilogy(f, ffts{2}); hold off;
175 legend('Spannung', 'Strom')
176 title("DFT: " + freqs(ff) + "Hz")
177 % dt = datatip(g, 46, ffts{i}(46));
178 grid("on");
179 xlabel("Frequenz [Hz]")
180 ylabel("Amplitude [Vrms]")
181
182 nexttile;
183 stem(freqs, abs(dft_data{1})); hold on;
184 stem(freqs, abs(dft_data{2})); hold off;
185 grid("on");
186 xlabel('Frequency (Hz)')
187 ylabel('DFT Magnitude')
188
189 nexttile;
190 plot(real(impedance(ff)), imag(impedance(ff)), 'x')
191 xlabel('Real')
192 ylabel('Imag')
193 grid("on");
194
195 nexttile;
196 plot(freqs, phase(ff), 'x')
197 xlabel('Freq [Hz]')
198 ylabel('Phase[°]')
199 grid("on");
200
201
202
203
204 figure();
205 plot(real(impedance), imag(impedance), 'x')
206 xlabel('Real')
```

```
ylabel('Imag')
grid("on");
figure();
plot(freqs, phase, 'x')
xlabel('Freq [Hz]')
ylabel('Phase')
grid("on");
%% plot STD/Histo of impedanz and phase

plot_histogramm(impedance, phase);
[i , p] = sample_validation(impedance, phase);
plot_histogramm(i , p);
```

Controller

```
classdef Controller
    % Die Controller-Klasse verwendet einen Sensorhandler
    % Sie ist Zuständig für die Regelung der Verstärkung und
    % dem Kompensieren des Gleichanteils
    properties
        sensor;
        data;
        hist_data;
        hist_edges;
        bins = [0:40.96:4096];
        GAIN = [1, 10, 20, 30, 40, 60, 80, 120, 157, 200, 300, 400, 600, 800,
        ↪  900, 1200, 1570, 1600, 1800, 2400, 3140, 3200, 3600, 4710, 4800,
        ↪  6280, 6400];
        gain_idx = 1;
        n = 2
        %
        COMPENSATION_TYPE = Compensation.PWM
        PWM0_DC = 58; % Xn
        PWM_BASE = 50; % Xb
        DC_STEP = 0.5;
        RHEO_STEP = 10;
        RHEO_DEFAULT = 512;
        RHEO_RESITOR = 212;
        OS = 0;
        Nh = 0.15;
        fine_tune = false;
```

```
25          process = struct('os', 0, 'dc', 50, 'gain_idx', 0, 'state',
            ↪  Saturation.FALSE, 'rheo', 512);
26          compensation_limit = false;
27          compensation_done = false;
28          done = false;
29          state = Saturation.FALSE
30          name
31          figureNo
32      end
33
34      methods
35          function obj = Controller(sensor)
36              obj.sensor = sensor;
37              obj.name = sensor.name;
38              figureNo = numel(findall(0, 'Type', 'figure'));
39              if ~figureNo
40                  obj.figureNo = 1;
41              else
42                  obj.figureNo = figureNo + 1;
43              end
44          end
45
46          function obj = incRefVoltage(obj)
47              if (obj.COMPENSATION_TYPE == Compensation.PWM)
48                  obj.PWM0_DC = obj.PWM0_DC + obj.DC_STEP;
49                  obj.sensor.setPWM(obj.PWM0_DC, PGA.MAX0);
50              elseif (obj.COMPENSATION_TYPE == Compensation.RHEOSTAT)
51                  obj.RHEO_RESITOR = obj.RHEO_RESITOR + obj.RHEO_STEP;
52                  obj.sensor.setAD(obj.RHEO_RESITOR);
53              end
54          end
55
56          function obj = decRefVoltage(obj)
57              if (obj.COMPENSATION_TYPE == Compensation.PWM)
58                  obj.PWM0_DC = obj.PWM0_DC - obj.DC_STEP;
59                  obj.sensor.setPWM(obj.PWM0_DC, PGA.MAX0);
60              elseif (obj.COMPENSATION_TYPE == Compensation.RHEOSTAT)
61                  obj.RHEO_RESITOR = obj.RHEO_RESITOR - obj.RHEO_STEP;
62                  obj.sensor.setAD(obj.RHEO_RESITOR);
63              end
64          end
65
66          function obj = setHistogramThreshold(obj, th)
67              obj.Nh = th;
```

```
68          end
69
70          function obj = incGain(obj)
71              obj.gain_idx = obj.gain_idx + 1;
72              obj.sensor.setGain(obj.GAIN(obj.gain_idx))
73          end
74
75          function obj = decGain(obj)
76              obj.gain_idx = obj.gain_idx - 1;
77              obj.sensor.setGain(obj.GAIN(obj.gain_idx))
78          end
79
80          function obj = resetGain(obj)
81              obj.sensor.setGain(obj.GAIN(1))
82          end
83
84          function obj = getData(obj)
85              pause(2)
86              obj.data = obj.sensor.getSample();
87              figure(obj.figureNo);
88              plot(obj.data)
89              grid on;
90              title(obj.name + " | " + "Gain: " + obj.GAIN(obj.gain_idx))
91              [obj.hist_data, obj.hist_edges] = histcounts(obj.data, obj.bins);
92          end
93
94          function obj = checkRoughLimits(obj)
95              if (obj.PWM0_DC > 0) && (obj.PWM0_DC < 100)
96                  obj.fine_tune = false;
97              elseif (obj.RHEO_RESITOR > 0 ) && (obj.RHEO_RESITOR < 1024)
98                  obj.fine_tune = false;
99              else
100                 obj.fine_tune = true;
101             end
102         end
103
104         function obj = checkFineLimits(obj)
105             if (abs(obj.OS) >= 15)
106                 obj.compensation_limit = true;
107             end
108         end
109
110         function obj = checkLimits(obj)
```

```
111             if ((obj.PWM0_DC <= 0) || (obj.PWM0_DC >= 100)) && (abs(obj.OS)
                ↪  >= 15)
112                 obj.compensation_limit = true;
113             end
114         end
115
116         function obj = decMaxOS(obj)
117             obj.OS = obj.OS - 1;
118             obj.sensor.setOS(obj.OS, PGA.MAX0);
119             obj.state = obj.inSaturation();
120         end
121
122         function obj = incMaxOS(obj)
123             obj.OS = obj.OS + 1;
124             obj.sensor.setOS(obj.OS, PGA.MAX0);
125             obj.state = obj.inSaturation();
126         end
127
128         function obj = resetMaxOS(obj)
129             obj.OS = 0;
130             obj.sensor.setOS(obj.OS, PGA.MAX0);
131             obj.state = obj.inSaturation();
132         end
133
134         function obj = upperSaturationHandler(obj)
135             % solange ich noch in sättigung bin
136             % go low
137             % wenn ich dann in der anderen sättigung bin
138             % go high und versuche mit maxOS
139
140             obj.compensation_done = false;
141             while obj.inSaturation() == Saturation.HIGH
142                 obj = obj.checkLimits();
143                 if obj.compensation_limit
144                     break;
145                 end
146
147                 if obj.compensation_done
148                     break;
149                 end
150
151                 if (obj.fine_tune == false)
152                     obj = obj.checkRoughLimits();
153                     % normaler regelprozess (grob)
```

```
154                     obj = obj.incRefVoltage();
155                     obj.state = obj.inSaturation();
156 
157                     if obj.state == Saturation.FALSE
158                         obj.compensation_done = true;
159                         break;
160                     elseif obj.state == Saturation.TRUE
161                         obj.compensation_done = true;
162                         break;
163                     elseif obj.state == Saturation.LOW
164                         obj.fine_tune = true;
165                     end
166                 else
167                     % fine tune prozess
168                     while obj.inSaturation() == Saturation.LOW
169                         obj = obj.checkFineLimits();
170                         if (obj.compensation_limit == false)
171                             obj = obj.incMaxOS();
172                         else
173                             % PWM UND fine tune maximal.
174                             obj = obj.resetMaxOS();
175                             break;
176                         end
177 
178                         if obj.state == Saturation.FALSE
179                             obj.compensation_done = true;
180                             break;
181                         elseif obj.state == Saturation.TRUE
182                             obj.compensation_done = true;
183                             break;
184                         elseif obj.state == Saturation.HIGH
185                             % reset OS
186                             % PWM "andere Richtung"
187                             % OS in die andere richtung durchfahren.
188                             obj = obj.decRefVoltage();
189                             obj = obj.resetMaxOS();
190                             break;
191                         end
192                     end
193 
194                     while obj.inSaturation() == Saturation.HIGH
195                         obj = obj.checkFineLimits();
196                         if (obj.compensation_limit == false)
197                             obj = obj.decMaxOS();
```

```
198                         else
199                             % PWM UND fine tune ausgereitzt.
200                             obj = obj.resetMaxOS();
201                             break;
202                         end
203 
204                         if obj.state == Saturation.FALSE
205                             obj.compensation_done = true;
206                             break;
207                         elseif obj.state == Saturation.TRUE
208                             obj.compensation_done = true;
209                             break;
210                         elseif obj.state == Saturation.LOW
211                            % OS zweiter MAX
212                             % Kompletter Abbruch dieser aktion
213                             obj.compensation_limit = true;
214                             break;
215                         end
216                     end
217                 end
218             end
219         end
220 
221         function obj = lowerSaturationHandler(obj)
222 
223             obj.compensation_done = false;
224             while obj.inSaturation() == Saturation.LOW
225                 obj = obj.checkLimits();
226                 if obj.compensation_limit
227                     break;
228                 end
229 
230                 if obj.compensation_done
231                     break;
232                 end
233 
234                 if (obj.fine_tune == false)
235                     obj = obj.checkRoughLimits();
236                     % normaler Regelprozess (Grob)
237                     obj = obj.decRefVoltage();
238                     obj.state = obj.inSaturation();
239 
240                      if obj.state == Saturation.FALSE
241                         obj.compensation_done = true;
```

```
242                         break;
243                     elseif obj.state == Saturation.TRUE
244                         obj.compensation_done = true;
245                         break;
246                     elseif obj.state == Saturation.HIGH
247                         obj.fine_tune = true;
248                      end
249                 else
250                     %fine tune
251                     obj = obj.checkFineLimits();
252                     while obj.inSaturation() == Saturation.HIGH
253                         if (obj.compensation_limit == false)
254                             obj = obj.decMaxOS();
255                         else
256                             % PWM UND fine tune maximal.
257                             obj = obj.resetMaxOS();
258                             break;
259                         end
260
261                         if obj.state == Saturation.FALSE
262                             obj.compensation_done = true;
263                             break;
264                         elseif obj.state == Saturation.TRUE
265                             obj.compensation_done = true;
266                             break;
267                         elseif obj.state == Saturation.LOW
268                             % reset OS
269                             % PWM "andere Richtung"
270                             % OS in die andere richtung durchfahren.
271                             obj = obj.incRefVoltage();
272                             obj = obj.resetMaxOS();
273                             break;
274                         end
275                     end
276
277                     while obj.inSaturation() == Saturation.LOW
278                         obj = obj.checkFineLimits();
279                         if (~obj.compensation_limit)
280                             obj = obj.incMaxOS();
281                         else
282                             % PWM UND fine tune ausgereitzt.
283                             obj = obj.resetMaxOS();
284                             break;
285                         end
```

```
286
287                         if obj.state == Saturation.FALSE
288                             obj.compensation_done = true;
289                             break;
290                         elseif obj.state == Saturation.TRUE
291                             obj.compensation_done = true;
292                             break;
293                         elseif obj.state == Saturation.HIGH
294                             % OS zweiter MAX
295                             % Kompletter Abbruch dieser aktion
296                             obj.compensation_limit = true;
297                             break;
298                         end
299                     end
300                 end
301             end
302         end
303
304         function saturation = inSaturation(obj)
305             % checks if the signal is in saturation
306             % High: upper saturation
307             % LOW: lower saturation
308             % TRUE: upper AND lower saturation
309             % FALSE: not in saturation
310             % TODO: saturation variable could be class property?
311             saturation = Saturation.FALSE;
312             obj = obj.getData()
313             if obj.hist_data(100)/sum(obj.hist_data) > obj.Nh
314                 saturation = Saturation.HIGH;
315             end
316             if obj.hist_data(1)/sum(obj.hist_data) > obj.Nh
317                 saturation = Saturation.LOW;
318             end
319
320             if (obj.hist_data(1) ~= 0) && (obj.hist_data(100) ~= 0)
321                 saturation = Saturation.TRUE;
322             end
323         end
324
325         function obj = run(obj)
326             obj.resetGain()
327             obj.done = 0;
328             obj.compensation_limit = 0;
329             while ~obj.done
```

```
330
331                 if obj.compensation_limit
332                     % reset into last stable state not in statuation
333                     obj.process(1).gain_idx
334                     obj.sensor.setGain(obj.GAIN(obj.process(1).gain_idx))
335                     obj.sensor.setOS(obj.process(1).os, PGA.MAX0)
336                     obj.sensor.setPWM(obj.process(1).dc, PGA.MAX0)
337                     break;
338                 end
339                 obj.process(1).state = obj.state;
340                 obj.state = obj.inSaturation();
341                 if (obj.state == Saturation.FALSE)
342                     % save stable state
343                     obj.process(1).os = obj.OS;
344                     obj.process(1).gain_idx = obj.gain_idx;
345                     obj.process(1).dc = obj.PWM0_DC;
346
347                     if obj.gain_idx < length(obj.GAIN)
348                         obj = obj.incGain();
349                     else
350                         obj.compensation_limit = true;
351                     end
352                 elseif obj.state == Saturation.HIGH
353                     % Offset
354                     if obj.process(1).state ~= Saturation.LOW
355                         obj = obj.upperSaturationHandler();
356                     else
357                         % Oszillating
358                         obj.compensation_limit = true;
359                     end
360                 elseif obj.state == Saturation.LOW
361                     % Offset
362                     if obj.process(1).state ~= Saturation.HIGH
363                         obj = obj.lowerSaturationHandler();
364                     else
365                         % Oszillating
366                         obj.compensation_limit = true;
367                     end
368                 end
369                 if obj.state == Saturation.TRUE
370                     obj = obj.decGain();
371                     obj.done = true;
372                 end
373             end
```

```
374             obj = obj.getData()
375         end
376     end
377
378 end
379
```

Sensor Handler

```
1  classdef SensorHandler
2      % Der SensorHandler abstrahiert die Funktionen der Sensorplatine
3      % die Klasse ist zuständig für die Kommunikation zu der Sensorplatine.
4
5      properties
6          serialConnection;
7          dcs;
8          name;
9          gain;
10     end
11
12     methods
13         function obj = SensorHandler(comPort, name)
14             obj.serialConnection = serialport(comPort, 115200, 'Timeout',
               ↪  20);
15             obj.dcs = round(linspace(0, 200, 200), 1);
16             obj.name = name;
17             obj.gain = 1;
18         end
19
20         function gain = getGain(obj)
21             gain = obj.gain;
22         end
23
24         function writeSensor(obj, msg)
25             while true
26                 write(obj.serialConnection, msg, "uint16");
27                 pause(0.5)
28                 read(obj.serialConnection, 1, "uint16");
29                 if isempty(lastwarn())
30                     break;
31                 else
32                     lastwarn('', '');
33                 end
```

```
34              end
35          end
36
37          function result = getSample(obj)
38              write(obj.serialConnection, 0x00, "uint16")
39              result = obj.readSensor();
40          end
41
42          function result = readSensor(obj)
43              result = read(obj.serialConnection, 500, "uint16");
44          end
45
46          function offset = calcOS(~, os)
47              % +-
48              u_os = abs(os);
49              if os < 0
50                  sign = 0x0000;
51              else
52                  sign = 0x0010;
53              end
54
55              switch u_os
56                  case 0 % 0mV
57                      offset = bitor(0x0000, sign);
58                  case 1
59                      offset = bitor(0x0001, sign);
60                  case 2
61                      offset = bitor(0x0002, sign);
62                  case 3
63                      offset = bitor(0x0003, sign);
64                  case 4 % 4.9mV
65                      offset = bitor(0x0004, sign);
66                  case 5 % 6.10mV
67                      offset = bitor(0x0005, sign);
68                  case 6
69                      offset = bitor(0x0006, sign);
70                  case 7 % 8.4mV
71                      offset = bitor(0x0007, sign);
72                  case 8 % 10.6mV
73                      offset = bitor(0x0008, sign);
74                  case 9
75                      offset = bitor(0x0009, sign);
76                  case 10
77                      offset = bitor(0x000A, sign);
```

```
78                  case 11
79                      offset = bitor(0x000B, sign);
80                  case 12
81                      offset = bitor(0x000C, sign);
82                  case 13
83                      offset = bitor(0x000D, sign);
84                  case 14
85                      offset = bitor(0x000E, sign);
86                  case 15 % 17.6mV
87                      offset = bitor(0x000F, sign);
88              end
89          end
90
91          function gain = getGainIdx(~, g)
92              switch g
93                 case 1
94                    gain = 0x0000;
95                 case 10
96                    gain = 0x0001;
97                 case 20
98                    gain = 0x0002;
99                 case 30
100                   gain = 0x0003;
101                case 40
102                   gain = 0x0004;
103                case 60
104                   gain = 0x0005;
105                case 80
106                   gain = 0x0006;
107                case 120
108                   gain = 0x0007;
109                case 157
110                   gain = 0x0008;
111                case 0.2
112                   gain = 0x0009;
113                otherwise
114                   gain = 0x0000;
115             end
116         end
117
118         function dutyCycle = calcDutyCycle(obj, dc)
119             % dc in [%] [0...100]
120             dc = dc * 2;
121             [~,closestIndex] = min(abs(dc-obj.dcs'));
```

```
122             dutyCycle = uint16(closestIndex);
123         end
124
125         function setMAX(obj, gain, max)
126             ga = obj.getGainIdx(gain);
127             if max == PGA.MAX0
128                 msg = bitor(0x1000, ga);
129             else
130                 msg = bitor(0x2000, ga);
131             end
132
133             obj.writeSensor(msg)
134         end
135
136         function obj = setGain(obj, gain)
137             obj.gain = gain;
138             gain_ref = [1, 10, 20, 30, 40, 60, 80, 120, 157];
139             gain0 = 1;
140             gain1 = 1;
141             for i=1:length(gain_ref)
142                 temp_gain = gain/gain_ref(i);
143                 if ismember(temp_gain, gain_ref)
144                     gain0 = gain_ref(i);
145                     gain1 = temp_gain;
146                 end
147             end
148             obj.setMAX(gain0, PGA.MAX0)
149             pause(1)
150             obj.setMAX(gain1, PGA.MAX1)
151         end
152
153         function setOS(obj, os, max)
154             offset = obj.calcOS(os);
155
156             if max == PGA.MAX0
157                 msg = bitor(0x5000, offset);
158             else
159                 msg = bitor(0x6000, offset);
160             end
161
162             obj.writeSensor(msg)
163         end
164
165         function obj = setAD(obj, r)
```

```
166             % r == index 0 - 1023
167             msg = bitor(0xA000, r - 1);
168             obj.writeSensor(msg)
169         end
170
171         function setPWM(obj, dutyCycle, max)
172             dc = obj.calcDutyCycle(dutyCycle);
173
174             if dutyCycle > 100
175                 error('Duty Cycle must between 0...100')
176             end
177             if max == PGA.MAX0
178                 msg = bitor(0x3000, dc);
179             else
180                 msg = bitor(0x4000, dc);
181             end
182             obj.writeSensor(msg)
183         end
184
185         function startMeasure(obj)
186             msg = 0x7000; % start measure
187             obj.writeSensor(msg);
188
189         end
190
191         function enableADC(obj)
192             msg = 0x8000; % enable ADC
193             obj.writeSensor(msg);
194         end
195
196         function disableADC(obj)
197             msg = 0x9000; % disable ADC
198             obj.writeSensor(msg);
199         end
200
201         function result = getMeasure(obj)
202             result = obj.readSensor();
203         end
204     end
205 end
206
```

Messung

```
function [impedance, phase, vol, curr] = measure(freq, n, voltageMeasure,
↪  currentMeasure, iObj, AWG, p_old)
% Die Funktion für die Messung für eine Frequenz durch
    freqs = repmat(freq, n, 1);
    p = 15;
    l = 500;
    phase = [];
    fs = calcFs(freq, p, l);
    setAWG(freq, AWG, iObj, fs);
    for ff = 1:length(freqs)
        clear voltage current buffer measures dft_data
        pause(0.1)
        % Sync

        voltage = 0;
        current = 0;
        offset = 1;

        % Measure
        set(AWG.Output(2),'State','off'); % disable trigger;
        voltageMeasure.startMeasure();
        currentMeasure.startMeasure();
        set(AWG.Output(2),'State','on'); % enable trigger;
        fwrite(iObj,'SOURce1:PHASe:INITiate')
        voltageMeasure.enableADC();
        pause(p/freqs(ff)) % delay in order to avoid serial timeout

        voltage = voltageMeasure.getMeasure();
        current = currentMeasure.getMeasure();
        % plt(voltage, current, freqs(ff));
        voltage = voltage/voltageMeasure.gain;
        current = current/currentMeasure.gain;

        threshold = 360/(l/p);
        nok = true;
        os = 2;
        [v, c] = shiftRight(voltage, current, os);
        measures = {v, c};
        % measures = {voltage, current};
        dft_data = calcAmplitude(measures, freqs(ff), fs);
        impedance(ff) = dft_data{1}/dft_data{2};
        phase(ff) = toDegrees("radians", angle(impedance(ff)));
```

```
42
43          vol{ff}  = voltage;
44          curr{ff} = current;
45
46      end
47  end
48
49  function [voltage, current] = compOffset(voltage, current, freq)
50      persistent old_phi ;
51      if isempty(old_phi)
52          old_phi = 0;
53      end
54      % Wenn der Betrag der alte Phase größer 24 ist
55      % dann muss geschoben werden. (p = alte Phase - neue Phase)
56      % bei negativem Vorzeichen muss nach rechts
57      % bei positivem Vorzeichen nach links
58      measures = {voltage, current};
59      dft_data = calcAmplitude(measures, freq);
60      impedance = dft_data{1}/dft_data{2};
61      phase = toDegrees("radians", angle(impedance));
62      offset = 1;
63      delta_phi  = old_phi  - phase;
64      m = {voltage, current};
65      while abs(delta_phi) > 24
66          old_phi = phase;
67          offset = offset + 1;
68          if delta_phi > 0
69              [vol, curr] = shiftRight(measures{1}, measures{2}, offset);
70          else
71              [vol, curr] = shiftLeft(measures{1}, measures{2}, offset);
72          end
73          m = {vol, curr};
74          dft_data = calcAmplitude(m, freq);
75          impedance = dft_data{1}/dft_data{2};
76          phase = toDegrees("radians", angle(impedance));
77          delta_phi = old_phi  - phase;
78      end
79      voltage = m{1};
80      current = m{2};
81  end
82
83  function [voltage, current] = shiftRight(voltage, current, os)
84      voltage = [voltage(os:end)];
85      current = [current(1:end-(os-1))];
```

```
86  end
87
88  function [voltage, current] = shiftLeft(voltage, current, os)
89      voltage = [voltage(1:end-(os-1))];
90      current = [current(os:end)];
91  end
92
93  function dft_data = calcAmplitude(measures, f, fs)
94      shunt = 4; % Ohm
95      freq_indices = [];
96      L = length(measures{1});
97      freq_indices = [freq_indices, round(f/fs*L) + 1];
98      for i = 1:length(measures)
99          buffer{i} = measures{i} - mean(measures{i});
100         buffer{i}  = buffer{i}/4096 * 3.3;
101         if i == 2 || i == 4
102             buffer{i} = buffer{i}/shunt;
103         end
104         %ffts{i} = ((abs(fft(buffer{i}))/L)*2)/sqrt(2); % * 2 for 0-max, * 4
            ↪  for peak-peak
105         dft_data{i} = (goertzel(buffer{i}, freq_indices)/L)*2;
106     end
107 end
108
109 function setAWG(f, AWG, iObj, fs)
110     offset_chan_1 = 0.5;
111     set(AWG.Frequency(2), 'Frequency', fs);
112     set(AWG.Frequency(1),'Frequency', f);
113     set(AWG.Voltage(1), 'Amplitude', 0.01)
114     set(AWG.Waveform(1),'Shape','sin');
115     fwrite(iObj,['SOURce1:VOLTage:LEVel:IMMediate:OFFSet
        ↪  ',mat2str(offset_chan_1),'V'],'int8')
116     set(AWG.Output(1),'State','on');
117 end
118
119 function plt(voltage, current, f)
120     figureNo = numel(findall(0, 'Type', 'figure')) + 1;
121     if ~figureNo
122         figureNo = 1;
123     end
124     figure(figureNo);
125     plot(voltage)
126     hold on;
127     title("f: " + f + " Hz")
```

```
128     plot(current)
129     grid on;
130     legend('Spannung','Strom')
131     hold off;
132 end
133 
134 function fs = calcFs(f, p, l)
135     % p number of periods to measure
136     % l sample buffer length
137     fs = (l/p)*f;
138     % while fs > 15000
139     %     p = p + 1;
140     %     fs = l*f/p;
141     % end
142 end
```

Klirrfaktoranalyse

```
1  clear all;
2  close all;
3  % Frequenzbereich (in Hz)
4  fs = 44100; % Beispiel: 44.1 kHz Abtastrate, anpassen, falls erforderlich
5  duration = 2;
6  t = 0:1/fs:duration-1/fs;
7  fundamental_freq = 2000;
8  fundamental_wave = 1.1 * sin(2*pi*fundamental_freq*t);
9  
10 for i=1:length(fundamental_wave)
11     if fundamental_wave(i) > 1
12         fundamental_wave(i) = 1;
13     end
14 end
15 audio_signal = fundamental_wave;
16 % Anzahl der Abtastpunkte
17 N = length(audio_signal);
18 % Fourier-Transformation des Signals
19 freq_vector = fs*(0:(N/2)-1)/N;
20 fft_audio = abs(fft(audio_signal)/N)*2;
21 fft_audio = fft_audio(1:N/2);
22 [val, idx] = max(fft_audio);
23 semilogy(freq_vector, fft_audio(1:N/2))
24 grid on;
25 
```

```
26 %% Berechne die Amplitude der Grundfrequenz
27
28 grundfrequenz_amplitude = abs(fft_audio(idx));
29 harmonischen_amplituden_summe = 0;
30
31 for k = 2:6
32     % ersten 5 harmoischen
33     h_idx = k * (idx - 1) + 1;
34     harmonischen_amplituden_summe = harmonischen_amplituden_summe +
       ↪  abs(fft_audio(h_idx));
35 end
36
37 % Berechne den Klirrfaktor
38 klirrfaktor = harmonischen_amplituden_summe / (grundfrequenz_amplitude +
   ↪  harmonischen_amplituden_summe);
39
40 % Ausgabe des Klirrfaktors
41 fprintf('Klirrfaktor: %.2f\n', klirrfaktor);
42
43 %%
44 clear all
45 close all
46 signal_freq = 100;              % 100 Hz
47 signal_period = 1/signal_freq;  % s
48 number_periods = 25;
49
50 N_Samples = 3000;               % Number of Samples
51
52 Fs = N_Samples / signal_period / number_periods;
53 T = 1/Fs;
54 t = (0:1/Fs:(signal_period*number_periods)-1/Fs);            % Time vector
55 s = 1 * sin(2*pi*signal_freq*t);
56 ss = 1.1 * sin(2*pi*signal_freq*t);
57 sss = 2 * sin(2*pi*signal_freq*t);
58 % sättigung
59 for i = 1:length(ss)
60     if ss(i) > 1
61         ss(i) = 1;
62     end
63     if sss(i) > 1
64         sss(i) = 1;
65     end
66 end
67
```

```
68  fft_s = fft(s);
69  fft_ss = fft(ss);
70  fft_sss = fft(sss);
71
72  Ps = abs(fft_s/N_Samples)*2;
73  Pss = abs(fft_ss/N_Samples)*2;
74  Psss = abs(fft_sss/N_Samples)*2;
75  P1 = Ps(1:N_Samples/2+1);
76  P1(2:end-1) = P1(2:end-1);
77
78  P2 = Pss(1:N_Samples/2+1);
79  P2(2:end-1) = P2(2:end-1);
80
81  P3 = Psss(1:N_Samples/2+1);
82  P3(2:end-1) = P3(2:end-1);
83  tiledlayout(1,2)
84  nexttile;
85  f = Fs*(0:(N_Samples/2))/N_Samples;
86  semilogy(f,P1, f,P2, f,P3)
87  xlim([0 500])
88  legend({'Amplitude: 1,' 'Amplitude: 1.1', ' Amplitude: 2'})
89  ylabel("Amplitude [V]")
90  xlabel("Frequenz [Hz]")
91  grid on
92  nexttile;
93  semilogy(f,P1, f,P2, f,P3)
94  xlim([0 500])
95  legend({'Amplitude: 1,' 'Amplitude: 1.1', ' Amplitude: 2'})
96  grid on
97  ylabel("Amplitude [V]")
98  xlabel("Frequenz [Hz]")
99  xlim([89.5 115.6])
100 ylim([0.26 3.23])
101 ax = gca;
102 chart = ax.Children(3);
103 datatip(chart,100,1,"Location","southwest");
104 chart = ax.Children(1);
105 datatip(chart,100,1.609);
106 chart = ax.Children(2);
107 datatip(chart,100,1.082);
108 datatip(chart,100,1.082);
109 datatip(chart,100,1.082);
110
111 %% plot klirrfaktor vs. overshoot
```

```
112 clear all
113 close all
114 signal_freq = 100;              % 100 Hz
115 signal_period = 1/signal_freq;  % s
116 number_periods = 850;
117
118 N_Samples = 6000;               % Number of Samples
119
120 fs = 40000 ; %N_Samples / signal_period / number_periods;
121 f = fs*(0:(N_Samples/2))/N_Samples;
122
123 T = 1/fs;
124 t = (0:1/fs:(signal_period*number_periods)-1/fs);            % Time vector
125 amplitude = linspace(0.98, 1.2, 100);
126 for i=1:length(amplitude)
127     signal{i} = amplitude(i) * sin(2*pi*signal_freq*t);
128     signal2{i} = amplitude(i) * sin(2*pi*signal_freq*t);
129     awgn(signal{i}, 20, 'measured');
130     for j=1:length(signal{i})
131         % saturation
132         if signal{i}(j) > 1
133             signal{i}(j) = 1;
134         end
135         if signal{i}(j) < -1
136             signal{i}(j) = -1;
137         end
138     end
139     fft_sig{i} = abs(fft(signal{i})/N_Samples)*2;
140     fft_sig2{i} = abs(fft(signal2{i})/N_Samples)*2;
141     [~, idx] = max(fft_sig{i}(1:N_Samples/2+1));
142     first_harmonic(i) = (abs(fft_sig{i}(idx)));
143     first_harmonic2(i) = (abs(fft_sig2{i}(idx)));
144     harmonics(i) = 0;
145
146     for k = 2:11
147         h_idx = k * (idx - 1) + 1;
148         harmonics(i) = harmonics(i) + (abs(fft_sig{i}(h_idx)))^2;
149     end
150     klirr(i) = (sqrt(harmonics(i) / (first_harmonic(i)^2 + harmonics(i))));
151     klirr_db(i) = 20*log10(1/klirr(i));
152     plot(f, fft_sig{i}(1:N_Samples/2+1))
153     hold on;
154 end
155 hold off;
```

```
156
157 yyaxis left;
158 plot(amplitude, klirr)
159 ylabel("Klirrfaktor")
160 yyaxis right
161 plot(amplitude, klirr_db)
162 ylabel("Klirrabstand [dB]")
163 ylim([16, 65])
164 xlim([0.98, 1.1])
165 xlabel("Amplitude [V]")
166 title("Klirrfaktor der ersten 10 harmonischen.")
167 grid on;
168 %
    ↪ exportgraphics(gcf,'C:\Users\frahmt\projects\masterthesis\chapters\img\klirrfaktor.pdf')
169 %% Impedanz
170 figure
171 plot(klirr, first_harmonic./first_harmonic2);
172 hold on;
173 plot(klirr, first_harmonic2./first_harmonic);
174 hold off;
175 grid on;
```

Klirrfaktormessung

```
1  %% Ausregelung und Verstärkung
2  %  AWG: free all devices before connection
3  clear all;
4  close all;
5  try
6      fclose(instrfind);
7      delete(instrfind);
8  catch clc
9  end
10
11 [iObj, AWG] = connect_funcgen();
12 fclose(iObj);
13 iObj.OutputBufferSize = 10e6;
14 fopen(iObj);
15 [funcgen_settings]  = funcgen_settings_method_1;
16 connect(funcgen_settings)
17 %% Aussteuerung
18 voltageMeasure = SensorHandler('COM7', 'Voltage');
19 voltageMeasure.setGain(1);
```

```
20
21  % Regelung OS/GAIN
22  clt_volt = Controller(voltageMeasure);
23  clt_volt = clt_volt.run();
24
25  %% Messung
26  f = 61;
27  fs = 500 * f / 10;
28  set(AWG.Frequency(1),'Frequency', f);
29  set(AWG.Frequency(2), 'Frequency', fs);
30  x = linspace(0.6, 3, 20);
31  set(AWG.Voltage(1), 'Amplitude', x(1))
32  for i=1:length(x)
33      set(AWG.Voltage(1), 'Amplitude', x(i))
34      data{i, 1} = x(i);
35      data{i, 2} = clt_volt.getData();
36  end
37
38  %% Analyse
39  close all;
40  data = load("G:\Meine Ablage\DATA\Klirrfaktor\data02.mat");
41  data = data.data;
42  L = length(data{1,2}.data);
43  leg = [];
44  tiledlayout(2,1);
45  nexttile;
46  gain = 10;
47  for i=1:length(x)
48      Y = fft(data{i, 2}.data/4096 * 3.3);
49      P2 = abs(Y/L)/gain;
50      P2 = abs(fft(data{i, 2}.data./4096 * 3.3)/L)/sqrt(2);
51      P1 = P2(1:L/2+1);
52      P1(2:end-1) = 2*P1(2:end-1);
53      data{i, 3} = P1;
54
55      freqV = fs*(0:(L/2))/L;
56      semilogy(freqV,P1)
57      leg = [leg, data{i, 1}];
58
59      idx = 11;
60      grundfreq(i) = P1(idx)^2;
61      h(i) = 0;
62      for ii=2:11
63          p = (idx-1) * ii + 1;
```

```
64          h(i) = h(i) + (P1(p))^2;
65      end
66      k(i) = sqrt(h(i)/(grundfreq(i) + h(i)));
67      klirr_db(i) = 20*log10(1/k(i));
68      % clear h grundfreq
69      hold on;
70      grid on;
71  end
72
73  xlim([1 650])
74  hold off;
75  ylabel("Amplitude [dB]")
76  xlabel("Amplitude [V]")
77  title("Grundfrequenz 61 Hz | Amplituden 0.6V_{pp} - 3V_{pp}")
78  legend(string(leg(1:5:end)))
79  nexttile;
80  s = 1;
81  plot(leg(s:end), k(s:end), 'x-')
82  hold on
83  yyaxis left
84  ylabel("Klirrfaktor [1]")
85  yyaxis right
86  plot(leg(s:end), klirr_db(s:end), 'o-')
87  ylabel("Klirrabstand [dB]")
88  xlabel("Amplitude [V]")
89  title("Klirranalyse")
90  grid on
91
92  n = 5;
93  tiledlayout(3,1);
94  nexttile;
95  for ii=1:length(data)
96      ampl_error(ii) =  data{ii, 1} - data{ii, 3}(11);
97  end
98  yyaxis left
99  plot(ampl_error, k, 'x-')
100 grid on;
101 ylabel("Klirrfaktor [1]")
102 yyaxis right
103 plot(ampl_error, klirr_db, 'o-')
104 ylabel("Klirrabstand [dB]")
105 xlabel("Amplitudenabweichung [V]")
106 title("Amplitudenabweichung der Grundfrequenz | 61Hz")
107
```

```
nexttile;
hh = histogram(data{n, 2}.data, 100);
hold on;
hh.BinCounts = hh.Values /(max(hh.Values)); % Normierun auf 1
lower_end = 4096 * 0.10;
upper_end = 4096 - lower_end;
yy1 = [upper_end upper_end 4096 4096];
yy2 = [lower_end lower_end 0 0];
xx = [0 1 1 0];
ylim([0, 1]);
xlim([0, 4096])
fill(yy1, xx, 'r', 'FaceAlpha', 0.25); hold on
fill(yy2, xx, 'r', 'FaceAlpha', 0.25); hold on
grid on;
ylabel(["Normierte Anzahl", "der Datenpunkte [1]"])
xlabel("ADC Quantisierungsstufen [1]")
title(["Grenzen: 10% | "+ "Frequenz: 61Hz | " + "Klirrfaktor: " + k(n) ,
↪  "Amplitude: " + data{n, 1} + " | Gemessen: " + data{n, 3}(11)])
hold off;

% Hier jetzt Klirrfaktor zu Verhältnis Plotten
% 10% Grenzen
nexttile;
for ii=1:length(data)
    his = histogram(data{ii, 2}.data, 100);
    Nh0(ii) = sum(his.Values(1:10))/sum(his.Values(1:end));
    Nhm(ii) = sum(his.Values(end-10:end))/sum(his.Values(1:end));
end

yyaxis left
plot(k, Nh0)
hold on;
yy = [0.05 0.05 0 0];
xx = [0 0.16 0.16 0];
schwelle = repmat(0.15, length(k));
fill(yy, xx, 'g', 'FaceAlpha', 0.25)
hold off;
ylabel("N_{h0} [1]")

yyaxis right
plot(k, Nhm)
ylabel("N_{hm} [1]")

title("Schwellwert N_h")
```

```
151 xlabel("Klirrfaktor [1]")
152 grid on;
153 %
    ↪ exportgraphics(gcf,'C:\Users\frahmt\projects\masterthesis\chapters\img\schwellwert_messung
154
155 %%
156 % Spannung
157 % Strom
158 signal_freq = 60;               % 100 Hz
159 signal_period = 1/signal_freq;  % s
160 number_periods = 50;
161
162 L = 6000;                % Number of Samples
163 fs = 40000 ; %N_Samples / signal_period / number_periods;
164 f = fs*(0:(L/2))/L;
165 T = 1/fs;
166 t = (0:L-1)*T;             % Time vector
167 phi = pi/6;
168
169 spannung = 1 * sin(2*pi*signal_freq*t);
170 strom = 1 * sin(2*pi*signal_freq*t + phi);
171 figure
172 plot(t, strom)
173 hold on;
174 plot(t, spannung)
175 freq_indices = round(f/fs*L) + 1;
176
177 fft_strom = fft(strom/L)/sqrt(2);
178 fft_spannung = fft(spannung/L)/sqrt(2);
179
180 impedanz = fft_spannung/fft_strom;
181
182 amplituden = linspace(0.6, 3);
183
184 for i=1:length(amplituden)
185     spannung = amplituden(i) * sin(2*pi*signal_freq*t);
186     strom = amplituden(i) * sin(2*pi*signal_freq*t + phi);
187 end
188
189
190
191
192
```

RC-Modell

```
function [out] = rcModell(p, omega)
    Hej = p(2)./(1 + 1i.*p(2).*p(3).*omega);
    out = complex(real(Hej), imag(Hej));
end
```

Messdatensatzvalidierung

```
function [filtered_impedance, filtered_phase] = sample_validation(impedance,
↪  phase)
    bins = 3600; % seconds
    figure();
    hh = histogram(phase, bins, "BinEdges", linspace(-180, 180, bins));
    [~, idx] = max(hh.Values);
    low_idx = idx - round((bins/(bins / 10)) * 0.5);
    hi_idx = idx + round((bins/(bins / 10)) * 0.5);
    if (low_idx > 0) && (hi_idx < bins)
        lower_edge = hh.BinEdges(low_idx);
        upper_edge = hh.BinEdges(hi_idx);
    else
        filtered_impedance = [];
        filtered_phase = [];
        return
    end
    idxToDelete = [];
    for i = 1:length(phase)
        if (phase(i) < lower_edge) || (phase(i) > upper_edge)
            idxToDelete = [idxToDelete, i];
        end
    end
    impedance(idxToDelete) = [];
    phase(idxToDelete) = [];

    if ~isempty(impedance)
        filtered_impedance = impedance;
    else
        filtered_impedance = [];
    end
    if ~isempty(phase)
        filtered_phase = phase;
    else
        filtered_phase = [];
```

```
34      end
35      close gcf
36  end
37
```

Helfer und kleinere Berechnungen

```
1   %% Anpassung: Initialer Widerstand
2
3   close all;
4   clear all;
5   U = 1.000
6   U2 = 0.482
7   U1 = U - U2
8
9   R1 = 10000
10  R2 = 10000
11  R2i = (R1 * U2) / U1 % R2 || Ri
12
13  %Ri ?
14  Ri = R2i * R2 / (R2 - R2i)
15
16  % jetzt möchte ich R2 haben für RR = 10k
17  R2i = 10000
18  RR2 = R2i * Ri / (Ri - R2i)
19  %% Periodengenaues Abtasten
20
21
22  disp("Anzahl der zu messenden Perioden")
23  p = 500
24
25  disp("Bufferlänge auf dem XMC1100")
26  l = 3000
27
28  disp("Anzahl der datenpunkte pro Periode")
29  n = l/p
30
31  disp("Phasenabstand zwischend den Datenpunkten")
32  phase_dist = 360/n
33
34  disp("Es gibt zwischen den beiden XMC1100 eine verzögerung")
35  disp("Wenn diese Verzögerung größer als Ts ist")
36  disp("dann habe ich bei nur 10 Datenpunkten pro ")
```

```
37 disp("Periode schon einen Fehler von 36°")
38 f = 1000; % zunächst festgelegte maximal freq
39 p = linspace(1, 3000, 30000);
40 fs = l.*f./p;
41 Ts = 1./fs;
42 n = l./p;
43
44 yyaxis left
45 plot(n, fs)
46 xlabel("Datenpunkte pro Periode")
47 ylabel("fs [Hz]")
48 title("Für f = 1kHz")
49 yyaxis right
50 phasen_schritt = 360./n; % in °
51 plot(n, phasen_schritt) % Umsomehr schritte ich habe desto kleiner
   ↪ Phasenfehler
52 xlabel("Datenpunkte pro Periode")
53 ylabel("Phasenabstand pro Datenpunkt")
54 grid on;
55 legend("Fs/Abtastpunkte", "Schrittweite/Abtastpunkte")
56 %% PWM Frequenz gegen Tastgradauflösung
57
58 clear all;
59 freq = linspace(1e5, 5e5, 100000);
60 f = 64e6;
61 N = f./freq;
62 volt_res = 3.3./N
63
64 semilogx(freq, volt_res)
65 title({"Spannungsänderung pro Tastgradschritt","in Abhängigkeit der
   ↪ Frequenz"})
66 xlabel("Frequenz [Hz]")
67 ylabel("Schrittweite [mV]")
68 grid on;
69 exportgraphics(gcf,'C:\Users\tobi\projects\masterthesis\chapters\img\tastgrad.pdf')
70 %% Kondensatorkapazität
71
72
73 clear all
74 fg = 0:0.00001:0.01; % 1mHz
75 R = 20000;
76 for i=1:length(fg)
77     C(i) = 1./(2*pi*R*fg(i));
78 end
```

```
79
80  plot(fg, C)
81  grid on;
82  title("Notwendige Kapaziät die AC-Kopplung bei kleinen Frequenzen (20kOhm)")
83  xlabel("Frequenz [Hz]")
84  ylabel("Kapazität [F]")
85  %% Übertragungsverhalten MAX9939
86
87  % ---------------3k---------------
88  vin = [1. 1.125 1.250 1.375 1.5 1.625 1.750 1.875 2 2.125 2.25 2.375 2.5
    ↪  2.625];
89  pvin3k = [1.17 1.126 1.35 1.45 1.54 1.63 1.73 1.82 1.92 2.01 2.10 2.20 2.29
    ↪  2.39];
90  nvin3k = [1.11 1.21 1.32 1.43 1.53 1.64 1.74 1.85 1.96 2.06 2.17 2.28 2.38
    ↪  2.49];
91  vout3k = [1.576 1.589 1.602 1.614 1.627 1.640 1.653 1.665 1.678 1.691 1.703
    ↪  1.716 1.729 1.742];
92  % ---------------47k---------------
93  pvin47k = [1.56 1.58 1.59 1.61 1.63 1.65 1.66 1.68 1.69 1.71 1.72 1.74 1.76
    ↪  1.78];
94  nvin47k = [1.51 1.54 1.56 1.59 1.62 1.65 1.68 1.75 1.78 1.81 1.84 1.87 1.9
    ↪  1.93];
95  vout47k = [1.579 1.591 1.603 1.616 1.628 1.640 1.652 1.664 1.676 1.688 1.700
    ↪  1.712 1.724 1.736];
96  % ---------------10k---------------
97  pvin10k = [1.33 1.39 1.45 1.51 1.58 1.63 1.69 1.75 1.81 1.87 1.94 2.00 2.06
    ↪  2.13];
98  nvin10k = [1.29 1.36 1.44 1.52 1.60 1.68 1.76 1.84 1.92 1.98 2.03 2.10 2.19
    ↪  2.26];
99  vout10k = [1.544 1.563 1.582 1.600 1.620 1.639 1.658 1.677 1.696 1.715 1.734
    ↪  1.753 1.772 1.791];
100 plot(vin, vout3k,'x-', vin, vout10k, '.-', vin, vout47k)
101 ylabel("V_{out} [V]");
102 xlabel("V_{in} [V]");
103 legend('V_{out3k}','V_{out10k}', 'V_{out47k}');
104 grid on;
105 exportgraphics(gcf,'C:\Users\tobi\projects\masterthesis\chapters\img\übertragung-max-crop.pdf'
106 %% Histogrammbasiertes Sättigungkriterium
107
108 clear all;
109 close all;
110
111 x = linspace(0, 12.5, 1000);
112 ssc = ((4096/2) - 4096 * 0.06) * sin(x) + 4096/2;
```

```
113
114 ss = awgn(ssc, 15,'measured');
115 % Sättigung
116 for c = 1:length(ss);
117     if ss(c) > 4096;
118         ss(c) = 4096;
119     end
120     if ss(c) < 0;
121         ss(c) = 0;
122     end
123 end
124
125 lower_end = 4096 * 0.05;
126 upper_end = 4096 - lower_end;
127 mid_low = (4096/2) - (3300/2);
128 mid_high = (4096/2) + (3300/2);
129 median_area_low = 2048 - 2048*0.025;
130 median_area_high = 2048 + 2048*0.025;
131
132 lower_end_arr = repmat(lower_end, 1, 1000);
133 upper_end_arr = repmat(upper_end, 1, 1000);
134 x = linspace(0, 1, 1000);
135 xx = [0 1 1 0];
136 yy1 = [upper_end upper_end 4096 4096];
137 yy2 = [lower_end lower_end 0 0];
138 yy3 = [mid_low mid_low mid_high mid_high];
139 yy4 = [median_area_low median_area_low median_area_high median_area_high];
140 figure;
141 yyaxis right
142 %h = histogram(s, 100); hold on
143 hh = histogram(ss, 100); hold on
144 hh.BinCounts = hh.Values /(max(hh.Values));
145 %h.BinCounts = h.Values /(max(h.Values));
146 ylabel('[n]');
147 ylim([0, 1]);
148 xlim([0, 4096])
149 htmlGray = [128 128 128]/255;
150 fill(yy1, xx, 'r', 'FaceAlpha', 0.4); hold on
151 fill(yy2, xx, 'r', 'FaceAlpha', 0.4); hold on
152 %fill(yy4, xx, 'g', 'FaceAlpha', 0.2); hold on
153 fill(yy3, xx, htmlGray, 'FaceAlpha', 0.4); hold on
154
155 yyaxis left
156 %plot(s, x);
```

```
157 plot(ssc, x, 'LineWidth',2);
158 plot(ss, x);
159 ylabel('[t]');
160 xlabel('Quantisierungsstufen')
161 grid on;
162
163 %% Periodengenaue Messung
164
165
166 clear all
167 close all
168 x = linspace(0, 1, 1000);
169 phis = linspace(0, 2*pi, 10000);
170
171 s = sin(x*2*pi);
172 for i = 1:10000
173     s = sin(x*2*pi + phis(i));
174     n = histcounts(s, 10000);
175     N(i) = sum(n(1:length(n)/2))/sum(n(length(n)/2+1:length(n)));
176 end
177
178 tiledlayout(1,3)
179 nexttile;
180 s = sin(x*2*pi*3);
181 plot(x, s)
182 title("Phasengleiche Abtastung.")
183 xlabel("Zeit [w]")
184 ylabel("Amplitude [V]")
185 grid on
186 nexttile;
187 histogram(s, 100)
188 title("Quantisierung rauschfreier Sinus (100 nibs)")
189 ylabel("Datenpunkte [1]")
190 xlabel("Klassen")
191 grid on
192 nexttile;
193
194
195 plot(phis, N)
196 title("Histogramverältnis(10k nibs) bei Phasenverschiebung, ohne Rauschen")
197 ylabel("N(1:50)/N(51:100)")
198 xlabel('Phasenverschiebung [rad]')
199 xticks([0,pi,2*pi])
200 xticklabels({'0', 'pi', '2pi'})
```

```
grid on

%% Klirrfaktor


clear all
close all
x = linspace(0, 1, 1000);
phis = linspace(0, 2*pi, 10000);

s = sin(x*2*pi);
for i = 1:10000
    s = sin(x*2*pi*3.3 + phis(i));
    n = histcounts(s, 10000);
    N(i) = sum(n(1:length(n)/2))/sum(n(length(n)/2+1:length(n)));
end

tiledlayout(1,3)
nexttile;
s = sin(x*2*pi*3.3);
plot(x, s)
title("Nicht phasengleiche Abtastung.")
xlabel("Zeit [s]")
ylabel("Amplitude [V]")
grid on
nexttile;
histogram(s, 100)
title("Quantisierung rauschfreier Sinus (100 nibs)")
ylabel("Datenpunkte [1]")
xlabel("Klassen")
grid on
nexttile;


plot(phis, N)
title("Histogramverältnis(10k nibs) bei Phasenverschiebung, ohne Rauschen")
ylabel("N(1:50)/N(51:100)")
xlabel('Phasenverschiebung [rad]')
xticks([0,pi,2*pi])
xticklabels({'0', 'pi', '2pi'})
grid on
%% Ableitung


```

```
245 clear all
246 close all
247 signal_freq = 100;              % 100 Hz
248 signal_period = 1/signal_freq;  % s
249 number_periods = 25;
250
251 N_Samples = 3000;               % Number of Samples
252
253 Fs = N_Samples / signal_period / number_periods;
254 T = 1/Fs;
255 t = (0:1/Fs:(signal_period*number_periods)-1/Fs);            % Time vector
256 s = 1 * sin(2*pi*signal_freq*t);
257 ss = 1.1 * sin(2*pi*signal_freq*t);
258 sss = 2 * sin(2*pi*signal_freq*t);
259 % sättigung
260 for i = 1:length(ss)
261     if ss(i) > 1
262         ss(i) = 1;
263     end
264     if sss(i) > 1
265         sss(i) = 1;
266     end
267 end
268
269 fft_s = fft(s);
270 fft_ss = fft(ss);
271 fft_sss = fft(sss);
272
273 Ps = abs(fft_s/N_Samples)*2;
274 Pss = abs(fft_ss/N_Samples)*2;
275 Psss = abs(fft_sss/N_Samples)*2;
276 P1 = Ps(1:N_Samples/2+1);
277 P1(2:end-1) = P1(2:end-1);
278
279 P2 = Pss(1:N_Samples/2+1);
280 P2(2:end-1) = P2(2:end-1);
281
282 P3 = Psss(1:N_Samples/2+1);
283 P3(2:end-1) = P3(2:end-1);
284 tiledlayout(1,2)
285 nexttile;
286 f = Fs*(0:(N_Samples/2))/N_Samples;
287 semilogy(f,P1, f,P2, f,P3)
288 xlim([0 500])
```

```
289 legend({'Amplitude: 1,' 'Amplitude: 1.1', ' Amplitude: 2'})
290 ylabel("Amplitude [V]")
291 xlabel("Frequenz [Hz]")
292 grid on
293 nexttile;
294 semilogy(f,P1, f,P2, f,P3)
295 xlim([0 500])
296 legend({'Amplitude: 1,' 'Amplitude: 1.1', ' Amplitude: 2'})
297 grid on
298 ylabel("Amplitude [V]")
299 xlabel("Frequenz [Hz]")
300 xlim([89.5 115.6])
301 ylim([0.26 3.23])
302 ax = gca;
303 chart = ax.Children(3);
304 datatip(chart,100,1,"Location","southwest");
305 chart = ax.Children(1);
306 datatip(chart,100,1.609);
307 chart = ax.Children(2);
308 datatip(chart,100,1.082);
309 datatip(chart,100,1.082);
310 datatip(chart,100,1.082);
```

A.2.7 XMC1100 - Quellcode

Main

```
1  /*
2   * main.c
3   *
4   *  Created on: 2023 Feb 13 16:48:25
5   *  Author: tobi
6   */
7
8  #include "DAVE.h"                 //Declarations from DAVE Code Generation
   ↪  (includes SFR declaration)
9  #include "src/max9939.h"
10 #include "src/ad5270.h"
11 #include "src/max9939-LOOKUP.h"
12 #include "src/AD5270BRMZ-LOOKUP.h"
13 #include "src/PWM-DC-LOOKUP.h"
14
```

```
15 #define SIZE 500
16 bool RDY = false;
17
18 const DIGITAL_IO_t DIGITAL_IO_ADC =
19 {
20   .gpio_port = XMC_GPIO_PORT0,
21   .gpio_pin = 15U,
22   .gpio_config = {
23     .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
24     .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
25
26   },
27   .hwctrl = XMC_GPIO_HWCTRL_DISABLED
28 };
29
30 enum CMD {REQ_DATA = 0, SET_GAIN0 = 1, SET_GAIN1 = 2, SET_DUTY0 = 3,
   ↪  SET_DUTY1 = 4, SET_OS0 = 5 , SET_OS1 = 6, MSR = 7, ENABLE_ADC = 8,
   ↪  DISABLE_ADC = 9, SET_RHEO = 10};
31
32 typedef struct detailed_result_struct {
33     uint8_t channel_num;
34     uint8_t group_num;
35     uint16_t conversion_result;
36 } detailed_result_struct_t;
37
38 uint32_t result;
39 bool valid_result;
40 detailed_result_struct_t detailed_result[10];
41 XMC_VADC_RESULT_SIZE_t res;
42
43 uint16_t BATTERIE = 0;
44 uint16_t VSIG = 0;
45 uint16_t ok[1] = {0};
46 uint16_t signal[SIZE] = {0};
47 uint16_t signal_out[SIZE] = {0};
48 uint16_t CNT = 0;
49 uint8_t MSR_MODE = 0;
50
51 uint8_t uart_cmd[2];
52
53 void enable_adc(){
54         NVIC_EnableIRQ((IRQn_Type)4U);
55 }
56
```

```
57 void syn_error(){
58         uint8_t valid_str[] = "ESYN";
59         UART_Transmit(&UART_0, valid_str, sizeof(valid_str)-1);
60         //while(UART_StartReceiveIRQ(&UART_0, buff, 1) !=
           ↪  UART_STATUS_SUCCESS);
61 }
62
63 void nois_error(){
64         uint8_t valid_str[] = "ENOIS";
65         UART_Transmit(&UART_0,valid_str, sizeof(valid_str)-1);
66         //while(UART_StartReceiveIRQ(&UART_0, buff, 1) !=
           ↪  UART_STATUS_SUCCESS);
67 }
68
69 void bit_error(){
70         uint8_t valid_str[] = "EBIT";
71         UART_Transmit(&UART_0,valid_str, sizeof(valid_str)-1);
72         //while(UART_StartReceiveIRQ(&UART_0, buff, 1) !=
           ↪  UART_STATUS_SUCCESS);
73 }
74
75 void adc_measurement_handler()
76 {
77     static uint8_t index;
78     uint32_t result;
79
80
81     valid_result = (bool) false;
82     result = ADC_MEASUREMENT_GetGlobalDetailedResult();
83
84     //if ((bool) (result >> VADC_GLOBRES_VF_Pos))
85     {
86
87         valid_result = (bool) true;
88         detailed_result[index].channel_num = (result & VADC_GLOBRES_CHNR_Msk)
           ↪  >> VADC_GLOBRES_CHNR_Pos;
89         detailed_result[index].group_num = ADC_MEASUREMENT_VSIG.group_index;
90         detailed_result[index].conversion_result = (result &
           ↪  VADC_GLOBRES_RESULT_Msk)
91                 >> ((uint32_t)
                   ↪  ADC_MEASUREMENT_0.iclass_config_handle->conversion_mode_standard
                   ↪  * (uint32_t) 2);
92
93         // Channels: Battery and the Signal
```

```
94          if (detailed_result[index].channel_num == 1)
95          {
96              BATTERIE = detailed_result[index].conversion_result;
97          }
98
99          if (detailed_result[index].channel_num == 5)
100         {
101             VSIG = detailed_result[index].conversion_result;
102
103              if (CNT >= SIZE) {
104                     CNT = 0;
105                     NVIC_DisableIRQ((IRQn_Type)4U);
106                                 memcpy(signal_out, signal,
                                    ↪  sizeof(signal_out));
107                                 RDY = true;
108                                 if (MSR_MODE){
109                                         while(UART_Transmit(&UART_0,
                                            ↪  signal_out, sizeof(signal_out))
                                            ↪  != 0);
110                                         MSR_MODE = 0;
111                                 }
112                                 NVIC_EnableIRQ((IRQn_Type)4U);
113             }
114             else {
115                                 signal[CNT] = VSIG;
116                                 CNT++;
117             }
118         }
119     }
120     index++;
121     if (index > 9)
122         index = 0;
123 }
124
125 enum CMD decode_command(uint16_t * cmd){
126         enum CMD __cmd = REQ_DATA;
127         uint16_t cmd_bits = *(uint16_t*)cmd & 0xF000;
128
129         if (cmd_bits == 0x1000)
130                 __cmd = SET_GAIN0;
131         else if (cmd_bits == 0x2000)
132                 __cmd = SET_GAIN1;
133         else if (cmd_bits == 0x3000)
134                 __cmd = SET_DUTY0;
```

```
135         else if (cmd_bits == 0x4000)
136                 __cmd = SET_DUTY0;
137         else if (cmd_bits == 0x5000)
138                 __cmd = SET_OS0;
139         else if (cmd_bits == 0x6000)
140                 __cmd = SET_OS1;
141         else if (cmd_bits == 0x7000)
142                 __cmd = MSR;
143         else if (cmd_bits == 0x8000)
144                 __cmd = ENABLE_ADC;
145         else if (cmd_bits == 0x9000)
146                 __cmd = DISABLE_ADC;
147         else if (cmd_bits == 0xA000)
148                 __cmd = SET_RHEO;
149         return __cmd;
150 }
151
152
153 int main(void)
154 {
155     DAVE_STATUS_t status;
156     max_t max0 = 0;
157     max_t max1 = 1;
158
159     status = DAVE_Init(); /* Initialization of DAVE APPs  */
160     if (status != DAVE_STATUS_SUCCESS)
161     {
162                 /* Placeholder for error handler code. The while loop below
                    ↪  can be replaced with an user error handler. */
163                 XMC_DEBUG("DAVE APPs initialization failed\n");
164                 while (1U)
165                 {
166
167                 }
168     }
169
170     enum CMD cmd;
171     enum SIGN sign = NEGATIVE;
172
173     while (1U)
174     {
175
176                 if(UART_Receive(&UART_0, uart_cmd, 2) ==
                    ↪  UART_STATUS_SUCCESS){
```

```
177                     cmd = decode_command(&uart_cmd);
178                     if (cmd == REQ_DATA){
179                             while(!RDY){};
180                             while(UART_Transmit(&UART_0, signal_out,
                                ↪  sizeof(signal_out)) != 0);
181                             RDY = false;
182                             MSR_MODE = 0;
183                     }
184                     else if (cmd == SET_RHEO){
185                             uint16_t idx = *(uint16_t*)uart_cmd & 0x0FFF;
186                             uint16_t value = resistor_value[idx];
187                             AD5270_WriteRDAC(value);
188                             while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                ↪  != 0);
189                             CNT = 0; // global adc counter
190                     }
191                     else if (cmd == SET_GAIN0){
192                             uint16_t idx = *(uint16_t*)uart_cmd & 0x00FF;
193                             max_set_gain(&gain[idx], max0);
194                             while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                ↪  != 0);
195                             CNT = 0; // global adc counter
196                     }
197                     else if (cmd == SET_GAIN1){
198                             uint16_t idx = *(uint16_t*)uart_cmd & 0x00FF;
199                             max_set_gain(&gain[idx], max1);
200                             while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                ↪  != 0);
201                             CNT = 0; // global adc counter
202                     }
203                     else if (cmd == SET_DUTY0){
204                             uint16_t idx = *(uint16_t*)uart_cmd & 0x00FF;
205                             PWM_CCU4_SetDutyCycle(&PWM_CCU4_0,
                                ↪  dutycycle[idx-1]);
206                             while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                ↪  != 0);
207                             CNT = 0; // global adc counter
208                     }
209                     else if (cmd == SET_DUTY1){
210                             uint16_t idx = *(uint16_t*)uart_cmd & 0x00FF;
211                             // PWM_CCU4_SetDutyCycle(&PWM_CCU4_1,
                                ↪  dutycycle[idx-1]);
212                             while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                ↪  != 0);
```

```
213                     CNT = 0; // global adc counter
214             }
215             else if (cmd == SET_OS0){
216                     uint16_t idx = *(uint16_t*)uart_cmd & 0x000F;
217                     uint16_t s = *(uint16_t*)uart_cmd & 0x0010;
218                     if (s)
219                             sign = POSITIVE;
220                     else
221                             sign = NEGATIVE;
222
223                 max_set_offset(&offset[idx], sign, max0);
224                 while(UART_Transmit(&UART_0, ok, sizeof(ok)) !=
                    ↪  0);
225                 CNT = 0; // global adc counter
226             }
227             else if (cmd == SET_OS1){
228                     uint16_t idx = *(uint16_t*)uart_cmd & 0x000F;
229                     uint16_t s = *(uint16_t*)uart_cmd & 0x0010;
230                     if (s)
231                             sign = POSITIVE;
232                     else
233                             sign = NEGATIVE;
234                 max_set_offset(&offset[idx], sign, max1);
235                 while(UART_Transmit(&UART_0, ok, sizeof(ok)) !=
                    ↪  0);
236                 CNT = 0; // global adc counter
237             }
238             else if (cmd == MSR){
239                     NVIC_DisableIRQ((IRQn_Type)4U);
240                     memset(signal, 0x00, sizeof(signal));
241                     memset(signal_out, 0x00, sizeof(signal_out));
242                     CNT = 0; // global adc counter
243                     MSR_MODE = 1;
244                     while(UART_Transmit(&UART_0, ok, sizeof(ok))
                        ↪  != 0);
245             }
246             else if (cmd == ENABLE_ADC){
247                     DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_0);
248                     NVIC_EnableIRQ((IRQn_Type)4U);
249                     while(UART_Transmit(&UART_0, ok, sizeof(ok))
                        ↪  != 0);
250                     DIGITAL_IO_SetOutputLow(&DIGITAL_IO_0);
251             }
252             else if (cmd == DISABLE_ADC){
```

```
253                                     NVIC_DisableIRQ((IRQn_Type)4U);
254                                     while(UART_Transmit(&UART_0, ok, sizeof(ok))
                                        ↪  != 0);
255                                 }
256                             }
257         }
258     }
```

AD5270

```
1   /**
2   *   @file     AD5270.c
3   *   @brief    Source file for AD5270 rheostat
4   *   @author   Analog Devices Inc.
5   *
6   ********************************************************************************
7   * Copyright 2016(c) Analog Devices, Inc.
8   *
9   * All rights reserved.
10  *
11  * Redistribution and use in source and binary forms, with or without
12  * modification, are permitted provided that the following conditions are met:
13  *  - Redistributions of source code must retain the above copyright
14  *    notice, this list of conditions and the following disclaimer.
15  *  - Redistributions in binary form must reproduce the above copyright
16  *    notice, this list of conditions and the following disclaimer in
17  *    the documentation and/or other materials provided with the
18  *    distribution.
19  *  - Neither the name of Analog Devices, Inc. nor the names of its
20  *    contributors may be used to endorse or promote products derived
21  *    from this software without specific prior written permission.
22  *  - The use of this software may or may not infringe the patent rights
23  *    of one or more patent holders.  This license does not release you
24  *    from the requirement that you obtain separate licenses from these
25  *    patent holders to use this software.
26  *  - Use of the software either in source or binary form, must be run
27  *    on or directly connected to an Analog Devices Inc. component.
28  *
29  * THIS SOFTWARE IS PROVIDED BY ANALOG DEVICES "AS IS" AND ANY EXPRESS OR
30  * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, NON-INFRINGEMENT,
31  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
32  * IN NO EVENT SHALL ANALOG DEVICES BE LIABLE FOR ANY DIRECT, INDIRECT,
```

```
33  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    ↪  NOT
34  * LIMITED TO, INTELLECTUAL PROPERTY RIGHTS, PROCUREMENT OF SUBSTITUTE GOODS
    ↪  OR
35  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
36  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    ↪  LIABILITY,
37  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
    ↪  USE
38  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
39  *
40  ******************************************************************************/
41  #include "AD5270.h"
42  #include "eis-spi.h"
43
44
45
46  /**
47   * @brief sets a new value for the RDAC
48   * @param resistance new value for the resistance
49   * @return actual value of the resistance in the RDAC
50   */
51  void AD5270_WriteRDAC(uint16_t RDAC_val)
52  {
53
54      AD5270_WriteReg(WRITE_CTRL_REG, RDAC_WRITE_PROTECT); // RDAC register
        ↪  write protect -  allow update of wiper position through digital
        ↪  interface
55      AD5270_WriteReg(WRITE_RDAC, RDAC_val); // write data to the RDAC register
56      AD5270_WriteReg(WRITE_CTRL_REG, RDAC_WRITE_PROTECT); // RDAC register
        ↪  write protect -  allow update of wiper position through digital
        ↪  interface
57  }
58
59  /**
60   * @brief Writes 16bit data to the AD5270 SPI interface
61   * @param data to be written
62   * @return data returned by the AD5270
63   */
64   void AD5270_WriteReg(uint16_t command, uint16_t value)
65   {
66        uint16_t msg = (command | value);
67        SPI_AD5270_TX(&msg);
68   }
```

```
69
70  /**
71   * Changes the device mode, enabled or shutdown
72   * @param mode - new mode of the device
73   */
74  void AD5270_ChangeMode(AD5270Modes_t mode)
75  {
76
77     AD5270_WriteReg(SW_SHUTDOWN, (uint16_t)(mode));
78  }
```

```
1   /**
2   *   @file     AD5270.h
3   *   @brief    Header file for AD5270 rheostat
4   *   @author   Analog Devices Inc.
5   *
6   ********************************************************************************
7   * Copyright 2016(c) Analog Devices, Inc.
8   *
9   * All rights reserved.
10  *
11  * Redistribution and use in source and binary forms, with or without
12  * modification, are permitted provided that the following conditions are met:
13  *  - Redistributions of source code must retain the above copyright
14  *    notice, this list of conditions and the following disclaimer.
15  *  - Redistributions in binary form must reproduce the above copyright
16  *    notice, this list of conditions and the following disclaimer in
17  *    the documentation and/or other materials provided with the
18  *    distribution.
19  *  - Neither the name of Analog Devices, Inc. nor the names of its
20  *    contributors may be used to endorse or promote products derived
21  *    from this software without specific prior written permission.
22  *  - The use of this software may or may not infringe the patent rights
23  *    of one or more patent holders.  This license does not release you
24  *    from the requirement that you obtain separate licenses from these
25  *    patent holders to use this software.
26  *  - Use of the software either in source or binary form, must be run
27  *    on or directly connected to an Analog Devices Inc. component.
28  *
29  * THIS SOFTWARE IS PROVIDED BY ANALOG DEVICES "AS IS" AND ANY EXPRESS OR
30  * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, NON-INFRINGEMENT,
31  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
32  * IN NO EVENT SHALL ANALOG DEVICES BE LIABLE FOR ANY DIRECT, INDIRECT,
```

```
33  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    ↪  NOT
34  * LIMITED TO, INTELLECTUAL PROPERTY RIGHTS, PROCUREMENT OF SUBSTITUTE GOODS
    ↪  OR
35  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
36  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    ↪  LIABILITY,
37  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
    ↪  USE
38  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
39  *
40  ******************************************************************************/
41
42  #ifndef AD5270_H
43  #define AD5270_H
44
45  #include <stdint.h>
46
47  #ifdef __cplusplus
48  extern "C" {
49  #endif
50
51
52      /// AD5270 commands
53      typedef enum {
54          NO_OP               =  0x0000,    ///< No data
55          NO_OP_cmd           =  0x0000,  ///< 16 bit no data
56          WRITE_RDAC          =  0x0400,    ///< Write to the RDAC Register
57          READ_RDAC           =  0x0800,    ///< Read from the RDAC Register
58          STORE_50TP          =  0x0C00,    ///< Store RDAC setting to 50-TP
59          SW_RST              =  0x1000,    ///< Software reset to last memory
            ↪  location
60          READ_50TP_CONTENTS  =  0x1400,    ///< Read the last memory contents
61          READ_50TP_ADDRESS   =  0x1800,    ///< Read the last memory address
62          WRITE_CTRL_REG      =  0x1C00,    ///< Write to the control Register
63          READ_CTRL_REG       =  0x2000,    ///< Read from the control Register
64          SW_SHUTDOWN         =  0x2400,    ///< Software shutdown (0) -
            ↪  Normal, (1) - Shutdown
65          HI_Zupper           =  0x8000,    ///< Get the SDO line ready for
            ↪  High Z
66          HI_Zlower           =  0x0100,    ///< Puts AD5270 into High Z mode
67          HI_Z_Cmd            =  0x8001   ///< Puts AD5270 into High Z mode*/
68      } AD5270Commands_t;
69
```

```
70      typedef enum {
71              PROGRAM_50TP_ENABLE = 1,
72              RDAC_WRITE_PROTECT = 2,
73              R_PERFORMANCE_ENABLE = 4,
74              MEMORY_PROGRAM_SUCCESFUL = 8
75          } AD5270ControlRegisterBits_t;
76
77      typedef enum {
78          NORMAL_MODE = 0,
79          SHUTDOWN_MODE = 1
80      } AD5270Modes_t;
81
82      void delay();
83      void AD5270_ChangeMode(AD5270Modes_t mode);
84      void AD5270_WriteReg(uint16_t command, uint16_t value);
85      void AD5270_WriteRDAC(uint16_t RDAC_val);
86      void TX_SPI_AD5270(uint16_t * cfg_val);
87
88  #ifdef __cplusplus
89  }
90  #endif // __cplusplus
91
92  #endif
```

MAX9939

```
1   /*
2    * max9939.c
3    *
4    *  Created on: 16 Feb 2023
5    *      Author: tobi
6    */
7
8   #include "max9939.h"
9   #include "ad5270.h"
10  #include "eis-spi.h"
11
12
13  static void max_set(uint8_t* msg, max_t max) {
14      if(max) {
15          SPI_MAX1_TX(msg, 1);
16      }   else {
17          SPI_MAX0_TX(msg, 1);
```

```
18      }
19  }
20
21  void max_set_gain(uint8_t* gain, max_t max){
22      uint8_t gain_mask = 0x01;
23      uint8_t msg = ((*gain << 1 ) | gain_mask) & 0xff;
24
25      max_set(&msg, max);
26  }
27
28  void max_set_offset(uint8_t* offset, enum SIGN sign, max_t max){
29      uint8_t offset_mask = 0x00;
30      uint8_t _sign = (uint8_t) sign;
31      uint8_t msg = ((_sign << 5) | (*offset << 1 ) | offset_mask) & 0xff;
32      max_set(&msg, max);
33  }
```

```
1   /*
2    * max9939.h
3    *
4    *  Created on: 16 Feb 2023
5    *      Author: tobi
6    */
7
8   #ifndef SRC_MAX9939_H_
9   #define SRC_MAX9939_H_
10
11  #include <stdbool.h>
12  #include <DAVE.h>
13
14  typedef bool max_t;
15  enum SIGN {POSITIVE = 0, NEGATIVE = 1};
16
17  void max_set_gain(uint8_t* gain, max_t max);
18  void max_set_offset(uint8_t* offset, enum SIGN sign, max_t max);
19
20
21  #endif /* SRC_MAX9939_H_ */
```

SPI

```
1   /*
2    * eis-spi.c
```

```
3   *
4   *  Created on: 15 Feb 2023
5   *      Author: tobi
6   */
7
8   #include "eis-spi.h"
9   #include <DAVE.h>
10
11  static bool FLAG = 0;
12
13  void TimerIRQHandler() {
14          FLAG = 1;
15          XMC_CCU4_SLICE_StartTimer(CCU40_CC40);
16  }
17
18  void delay() {
19          while (!FLAG);
20          FLAG = 0;
21  }
22
23  SPI_EIS_STATUS_t SPI_AD5270_TX(uint16_t * dataptr) {
24          SPI_EIS_STATUS_t state = SPI_EIS_STATUS_SUCCESS;
25          if (DIGITAL_IO_GetInput(&DIGITAL_IO_CC2) != 0)
26                  DIGITAL_IO_SetOutputLow(&DIGITAL_IO_CC2); // inactive high
27          delay();
28          delay();
29          DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_CLK);
30
31          for (uint8_t clk = 0; clk < 15; clk++) {
32                  uint8_t SPI_DATAn = ((*dataptr & (0x8000 >> clk)) >> (15 -
                    ↪  clk)); // MSB first -> schieben -> nur eins oder null
                    ↪  wird ausgegeben
33                  if (SPI_DATAn)
34                          DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_MISO);
35                  else
36                          DIGITAL_IO_SetOutputLow(&DIGITAL_IO_MISO);
37
38                  delay();
39                  DIGITAL_IO_SetOutputLow(&DIGITAL_IO_CLK);
40                  delay();
41                  DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_CLK);
42          }
43          delay();
44          DIGITAL_IO_SetOutputLow(&DIGITAL_IO_CLK);
```

```
45          delay();
46          delay();
47          if (!DIGITAL_IO_GetInput(&DIGITAL_IO_CC2))
48                  DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_CC2); // gpio CS default
                    ↪ high
49          return state;
50
51  }
52
53  static SPI_EIS_STATUS_t SPI_TX(uint8_t *dataptr, uint8_t count, DIGITAL_IO_t
    ↪ CC) {
54          SPI_EIS_STATUS_t state = SPI_EIS_STATUS_SUCCESS;
55          DIGITAL_IO_SetOutputLow(&CC);
56          delay();
57
58          for (uint8_t clk = 0; clk < 8; clk++) {
59                  uint8_t SPI_DATAn = ((*dataptr & (0x01 << clk)));
60                  if (SPI_DATAn) {
61                          DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_MISO);
62                  } else {
63                          DIGITAL_IO_SetOutputLow(&DIGITAL_IO_MISO);
64
65                  }
66
67                  delay();
68                  DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_CLK);
69                  delay();
70                  DIGITAL_IO_SetOutputLow(&DIGITAL_IO_CLK);
71          }
72          delay();
73          delay();
74          DIGITAL_IO_SetOutputHigh(&CC);
75          return state;
76  }
77
78  SPI_EIS_STATUS_t SPI_MAX0_TX(uint8_t* dataptr, uint8_t count) {
79          return SPI_TX(dataptr, count, DIGITAL_IO_CC0);
80  }
81
82  SPI_EIS_STATUS_t SPI_MAX1_TX(uint8_t* dataptr, uint8_t count) {
83          return SPI_TX(dataptr, count, DIGITAL_IO_CC1);
84  }
85
```

```c
#include <DAVE.h>


typedef enum SPI_EIS_STATUS {
    SPI_EIS_STATUS_SUCCESS = 0U, /**< Status success */
    SPI_EIS_STATUS_FAILURE, /**< Status failure */
    SPI_EIS_STATUS_BUSY, /**< Busy state */
    SPI_EIS_STATUS_BUFFER_INVALID, /**< If input buffer and length is invalid */
    SPI_EIS_STATUS_MODE_MISMATCH /**< API invoked by a handle configured with different mode.
     e.g, If SPI_MASTER_StartTransmitDMA is invoked for an instance
     which has transmit mode configured as "Interrupt", will
     return this status.*/
} SPI_EIS_STATUS_t;

SPI_EIS_STATUS_t SPI_AD5270_TX(uint16_t * dataptr);
SPI_EIS_STATUS_t SPI_MAX0_TX(uint8_t* dataptr, uint8_t count);
SPI_EIS_STATUS_t SPI_MAX1_TX(uint8_t* dataptr, uint8_t count);

```

Lookuptabellen

```c
/*
 * max9939-LOOKUP.h
 *
 *  Created on: 16 Feb 2023
 *      Author: tobi
 */

#ifndef SRC_MAX9939_LOOKUP_H_
#define SRC_MAX9939_LOOKUP_H_


uint8_t gain[11] = {0x00, // 1
                    0x01, // 10
                    0x02, // 20
                    0x03, // 30
                    0x04, // 40
                    0x05, // 60
                    0x06, // 80
                    0x07, // 120
                    0x08, // 157
```

```
                    0x09, // 0.2 (Vcc5V) / 0.25 (Vcc3V3)
                    0x0A, // 1
};


uint8_t offset[16] = {0x00, // 0
                      0x01, // 1.30
                      0x02, // 2.50
                      0x03, // 3.80
                      0x04, // 4.90
                      0x05, // 6.10
                      0x06, // 7.3
                      0x07, // 8.4
                      0x08, // 10.6
                      0x09, // 11.7
                      0x0A, // 12.7
                      0x0B, // 13.7
                      0x0C, // 14.7
                      0x0D, // 15.7
                      0x0E, // 16.7
                      0x0F  // 17.6
};


#endif /* SRC_MAX9939_LOOKUP_H_ */
```

```
/*
 * PWD-DC-LOOKUP.h
 *
 *  Created on: 3 May 2023
 *      Author: frahmt
 */

#ifndef SRC_PWM_DC_LOOKUP_H_
#define SRC_PWM_DC_LOOKUP_H_

/*
 *
 Columns 1 through 13

         0    1.5600    3.1300    4.6900    6.2500    7.8100    9.3800
         ↪  10.9400   12.5000   14.0600   15.6300   17.1900   18.7500

  Columns 14 through 26
```

```
18
19      20.3100   21.8800   23.4400   25.0000   26.5600   28.1300   29.6900
        ↪  31.2500   32.8100   34.3800   35.9400   37.5000   39.0600
20
21    Columns 27 through 39
22
23      40.6300   42.1900   43.7500   45.3100   46.8800   48.4400   50.0000
        ↪  51.5600   53.1300   54.6900   56.2500   57.8100   59.3800
24
25    Columns 40 through 52
26
27      60.9400   62.5000   64.0600   65.6300   67.1900   68.7500   70.3100
        ↪  71.8800   73.4400   75.0000   76.5600   78.1300   79.6900
28
29    Columns 53 through 65
30
31      81.2500   82.8100   84.3800   85.9400   87.5000   89.0600   90.6300
        ↪  92.1900   93.7500   95.3100   96.8800   98.4400  100.0000
32
33   */
34  uint16_t dutycycle[200] = {0, 50, 101, 151, 201, 251, 302, 352, 402, 452,
    ↪  503, 553, 603, 653, 704, 754, 804, 854, 905,
35                             955, 1005, 1055, 1106, 1156, 1206, 1256, 1307,
                               ↪  1357, 1407, 1457, 1508, 1558, 1608, 1658,
36                             1709, 1759, 1809, 1859, 1910, 1960, 2010, 2060,
                               ↪  2111, 2161, 2211, 2261, 2312, 2362, 2412,
37                             2462, 2513, 2563, 2613, 2663, 2714, 2764, 2814,
                               ↪  2864, 2915, 2965, 3015, 3065, 3116, 3166,
38                             3216, 3266, 3317, 3367, 3417, 3467, 3518, 3568,
                               ↪  3618, 3668, 3719, 3769, 3819, 3869, 3920,
39                             3970, 4020, 4070, 4121, 4171, 4221, 4271, 4322,
                               ↪  4372, 4422, 4472, 4523, 4573, 4623, 4673,
40                             4724, 4774, 4824, 4874, 4925, 4975, 5025, 5075,
                               ↪  5126, 5176, 5226, 5276, 5327, 5377, 5427,
41                             5477, 5528, 5578, 5628, 5678, 5729, 5779, 5829,
                               ↪  5879, 5930, 5980, 6030, 6080, 6131, 6181,
42                             6231, 6281, 6332, 6382, 6432, 6482, 6533, 6583,
                               ↪  6633, 6683, 6734, 6784, 6834, 6884, 6935,
43                             6985, 7035, 7085, 7136, 7186, 7236, 7286, 7337,
                               ↪  7387, 7437, 7487, 7538, 7588, 7638, 7688,
44                             7739, 7789, 7839, 7889, 7940, 7990, 8040, 8090,
                               ↪  8141, 8191, 8241, 8291, 8342, 8392, 8442,
45                             8492, 8543, 8593, 8643, 8693, 8744, 8794, 8844,
                               ↪  8894, 8945, 8995, 9045, 9095, 9146, 9196,
```

```
46                              9246, 9296, 9347, 9397, 9447, 9497, 9548, 9598,
                                ↪  9648, 9698, 9749, 9799, 9849, 9899, 9950,
47                              10000,};
48
49
50  uint16_t __dutycycle[64] = {156, 312, 468, 625, 781, 937, 1094, 1250, 1406,
    ↪  1563, 1719, 1875, 2031, 2188, 2144, 2500,
51                              2656, 2813, 2969, 3125, 3281, 3438, 3594, 3750,
                                ↪  3906, 4063, 4219, 4375, 4531, 4688, 4844,
52                              5000, 5156, 5313, 5469, 5625, 5781, 5938, 6094,
                                ↪  6250, 6406, 6563, 6719, 6875, 7031, 7188,
53                              7344, 7500, 7656, 7813, 7969, 8125, 8281, 8438,
                                ↪  8594, 8750, 8906, 9063, 9219, 9375, 9531,
54                              9688, 9844, 10000};
55
56
57  #endif /* SRC_PWM_DC_LOOKUP_H_ */
```

```
1   /*
2    * AD5270BRMZ-LOOKUP.h
3    *
4    *  Created on: 15 Feb 2023
5    *      Author: tobi
6    */
7
8   #ifndef SRC_AD5270BRMZ_LOOKUP_H_
9   #define SRC_AD5270BRMZ_LOOKUP_H_
10
11  #include <DAVE.h>
12
13  uint16_t cmd[10] = {0x00, // NOP: do nothing.
14                      0x1000, // Write contents of serial register data to
                        ↪  RDAC.
15                      0x2000, // Read contents of RDAC wiper register.
16                      0x3000, // Store wiper setting: store RDAC setting to
                        ↪  50-TP.
17                      0x4000, // Software reset: refresh RDAC with last 50-TP
                        ↪  memory stored value.
18                      0x5000, // Read contents of 50-TP from SDO output in the
                        ↪  next frame.
19                      0x6000, // Read address of last 50-TP programmed memory
                        ↪  location
20                      0x7000, // Write contents of serial register data to
                        ↪  control register.
```

```
21                      0x8000, // Read contents of control register.
22                      0x9000 // Software shutdown.  D0 = 0; normal mode. D0 =
                        ↪  1; device placed in shutdown mode
23  };
24
25  uint16_t resistor_value[1024] = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8,
    ↪  0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf, 0x10,
26                                   0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                                     ↪  0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d,
                                     ↪  0x1e,
27                                   0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25,
                                     ↪  0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b,
                                     ↪  0x2c,
28                                   0x2d, 0x2e, 0x2f, 0x30, 0x31, 0x32, 0x33,
                                     ↪  0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
                                     ↪  0x3a,
29                                   0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40, 0x41,
                                     ↪  0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
                                     ↪  0x48,
30                                   0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
                                     ↪  0x50, 0x51, 0x52, 0x53, 0x54, 0x55,
                                     ↪  0x56,
31                                   0x57, 0x58, 0x59, 0x5a, 0x5b, 0x5c, 0x5d,
                                     ↪  0x5e, 0x5f, 0x60, 0x61, 0x62, 0x63,
                                     ↪  0x64,
32                                   0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b,
                                     ↪  0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x71,
                                     ↪  0x72,
33                                   0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
                                     ↪  0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
                                     ↪  0x80,
34                                   0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
                                     ↪  0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d,
                                     ↪  0x8e,
35                                   0x8f, 0x90, 0x91, 0x92, 0x93, 0x94, 0x95,
                                     ↪  0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b,
                                     ↪  0x9c,
36                                   0x9d, 0x9e, 0x9f, 0xa0, 0xa1, 0xa2, 0xa3,
                                     ↪  0xa4, 0xa5, 0xa6, 0xa7, 0xa8, 0xa9,
                                     ↪  0xaa,
37                                   0xab, 0xac, 0xad, 0xae, 0xaf, 0xb0, 0xb1,
                                     ↪  0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
                                     ↪  0xb8,
```

```
38                                  0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
                                    ↪ 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5,
                                    ↪ 0xc6,
39                                  0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd,
                                    ↪ 0xce, 0xcf, 0xd0, 0xd1, 0xd2, 0xd3,
                                    ↪ 0xd4,
40                                  0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xdb,
                                    ↪ 0xdc, 0xdd, 0xde, 0xdf, 0xe0, 0xe1,
                                    ↪ 0xe2,
41                                  0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
                                    ↪ 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
                                    ↪ 0xf0,
42                                  0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
                                    ↪ 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd,
                                    ↪ 0xfe,
43                                  0xff, 0x100, 0x101, 0x102, 0x103, 0x104,
                                    ↪ 0x105, 0x106, 0x107, 0x108, 0x109,
                                    ↪ 0x10a,
44                                  0x10b, 0x10c, 0x10d, 0x10e, 0x10f, 0x110,
                                    ↪ 0x111, 0x112, 0x113, 0x114, 0x115,
                                    ↪ 0x116,
45                                  0x117, 0x118, 0x119, 0x11a, 0x11b, 0x11c,
                                    ↪ 0x11d, 0x11e, 0x11f, 0x120, 0x121,
                                    ↪ 0x122,
46                                  0x123, 0x124, 0x125, 0x126, 0x127, 0x128,
                                    ↪ 0x129, 0x12a, 0x12b, 0x12c, 0x12d,
                                    ↪ 0x12e,
47                                  0x12f, 0x130, 0x131, 0x132, 0x133, 0x134,
                                    ↪ 0x135, 0x136, 0x137, 0x138, 0x139,
                                    ↪ 0x13a,
48                                  0x13b, 0x13c, 0x13d, 0x13e, 0x13f, 0x140,
                                    ↪ 0x141, 0x142, 0x143, 0x144, 0x145,
                                    ↪ 0x146,
49                                  0x147, 0x148, 0x149, 0x14a, 0x14b, 0x14c,
                                    ↪ 0x14d, 0x14e, 0x14f, 0x150, 0x151,
                                    ↪ 0x152,
50                                  0x153, 0x154, 0x155, 0x156, 0x157, 0x158,
                                    ↪ 0x159, 0x15a, 0x15b, 0x15c, 0x15d,
                                    ↪ 0x15e,
51                                  0x15f, 0x160, 0x161, 0x162, 0x163, 0x164,
                                    ↪ 0x165, 0x166, 0x167, 0x168, 0x169,
                                    ↪ 0x16a,
```

```
52                                  0x16b, 0x16c, 0x16d, 0x16e, 0x16f, 0x170,
                                    ↪  0x171, 0x172, 0x173, 0x174, 0x175,
                                    ↪  0x176,
53                                  0x177, 0x178, 0x179, 0x17a, 0x17b, 0x17c,
                                    ↪  0x17d, 0x17e, 0x17f, 0x180, 0x181,
                                    ↪  0x182,
54                                  0x183, 0x184, 0x185, 0x186, 0x187, 0x188,
                                    ↪  0x189, 0x18a, 0x18b, 0x18c, 0x18d,
                                    ↪  0x18e,
55                                  0x18f, 0x190, 0x191, 0x192, 0x193, 0x194,
                                    ↪  0x195, 0x196, 0x197, 0x198, 0x199,
                                    ↪  0x19a,
56                                  0x19b, 0x19c, 0x19d, 0x19e, 0x19f, 0x1a0,
                                    ↪  0x1a1, 0x1a2, 0x1a3, 0x1a4, 0x1a5,
                                    ↪  0x1a6,
57                                  0x1a7, 0x1a8, 0x1a9, 0x1aa, 0x1ab, 0x1ac,
                                    ↪  0x1ad, 0x1ae, 0x1af, 0x1b0, 0x1b1,
                                    ↪  0x1b2,
58                                  0x1b3, 0x1b4, 0x1b5, 0x1b6, 0x1b7, 0x1b8,
                                    ↪  0x1b9, 0x1ba, 0x1bb, 0x1bc, 0x1bd,
                                    ↪  0x1be,
59                                  0x1bf, 0x1c0, 0x1c1, 0x1c2, 0x1c3, 0x1c4,
                                    ↪  0x1c5, 0x1c6, 0x1c7, 0x1c8, 0x1c9,
                                    ↪  0x1ca,
60                                  0x1cb, 0x1cc, 0x1cd, 0x1ce, 0x1cf, 0x1d0,
                                    ↪  0x1d1, 0x1d2, 0x1d3, 0x1d4, 0x1d5,
                                    ↪  0x1d6,
61                                  0x1d7, 0x1d8, 0x1d9, 0x1da, 0x1db, 0x1dc,
                                    ↪  0x1dd, 0x1de, 0x1df, 0x1e0, 0x1e1,
                                    ↪  0x1e2,
62                                  0x1e3, 0x1e4, 0x1e5, 0x1e6, 0x1e7, 0x1e8,
                                    ↪  0x1e9, 0x1ea, 0x1eb, 0x1ec, 0x1ed,
                                    ↪  0x1ee,
63                                  0x1ef, 0x1f0, 0x1f1, 0x1f2, 0x1f3, 0x1f4,
                                    ↪  0x1f5, 0x1f6, 0x1f7, 0x1f8, 0x1f9,
                                    ↪  0x1fa,
64                                  0x1fb, 0x1fc, 0x1fd, 0x1fe, 0x1ff, 0x200,
                                    ↪  0x201, 0x202, 0x203, 0x204, 0x205,
                                    ↪  0x206,
65                                  0x207, 0x208, 0x209, 0x20a, 0x20b, 0x20c,
                                    ↪  0x20d, 0x20e, 0x20f, 0x210, 0x211,
                                    ↪  0x212,
```

```
66                                  0x213, 0x214, 0x215, 0x216, 0x217, 0x218,
                                    ↪  0x219, 0x21a, 0x21b, 0x21c, 0x21d,
                                    ↪  0x21e,
67                                  0x21f, 0x220, 0x221, 0x222, 0x223, 0x224,
                                    ↪  0x225, 0x226, 0x227, 0x228, 0x229,
                                    ↪  0x22a,
68                                  0x22b, 0x22c, 0x22d, 0x22e, 0x22f, 0x230,
                                    ↪  0x231, 0x232, 0x233, 0x234, 0x235,
                                    ↪  0x236,
69                                  0x237, 0x238, 0x239, 0x23a, 0x23b, 0x23c,
                                    ↪  0x23d, 0x23e, 0x23f, 0x240, 0x241,
                                    ↪  0x242,
70                                  0x243, 0x244, 0x245, 0x246, 0x247, 0x248,
                                    ↪  0x249, 0x24a, 0x24b, 0x24c, 0x24d,
                                    ↪  0x24e,
71                                  0x24f, 0x250, 0x251, 0x252, 0x253, 0x254,
                                    ↪  0x255, 0x256, 0x257, 0x258, 0x259,
                                    ↪  0x25a,
72                                  0x25b, 0x25c, 0x25d, 0x25e, 0x25f, 0x260,
                                    ↪  0x261, 0x262, 0x263, 0x264, 0x265,
                                    ↪  0x266,
73                                  0x267, 0x268, 0x269, 0x26a, 0x26b, 0x26c,
                                    ↪  0x26d, 0x26e, 0x26f, 0x270, 0x271,
                                    ↪  0x272,
74                                  0x273, 0x274, 0x275, 0x276, 0x277, 0x278,
                                    ↪  0x279, 0x27a, 0x27b, 0x27c, 0x27d,
                                    ↪  0x27e,
75                                  0x27f, 0x280, 0x281, 0x282, 0x283, 0x284,
                                    ↪  0x285, 0x286, 0x287, 0x288, 0x289,
                                    ↪  0x28a,
76                                  0x28b, 0x28c, 0x28d, 0x28e, 0x28f, 0x290,
                                    ↪  0x291, 0x292, 0x293, 0x294, 0x295,
                                    ↪  0x296,
77                                  0x297, 0x298, 0x299, 0x29a, 0x29b, 0x29c,
                                    ↪  0x29d, 0x29e, 0x29f, 0x2a0, 0x2a1,
                                    ↪  0x2a2,
78                                  0x2a3, 0x2a4, 0x2a5, 0x2a6, 0x2a7, 0x2a8,
                                    ↪  0x2a9, 0x2aa, 0x2ab, 0x2ac, 0x2ad,
                                    ↪  0x2ae,
79                                  0x2af, 0x2b0, 0x2b1, 0x2b2, 0x2b3, 0x2b4,
                                    ↪  0x2b5, 0x2b6, 0x2b7, 0x2b8, 0x2b9,
                                    ↪  0x2ba,
```

```
80                                  0x2bb, 0x2bc, 0x2bd, 0x2be, 0x2bf, 0x2c0,
                                    ↪  0x2c1, 0x2c2, 0x2c3, 0x2c4, 0x2c5,
                                    ↪  0x2c6,
81                                  0x2c7, 0x2c8, 0x2c9, 0x2ca, 0x2cb, 0x2cc,
                                    ↪  0x2cd, 0x2ce, 0x2cf, 0x2d0, 0x2d1,
                                    ↪  0x2d2,
82                                  0x2d3, 0x2d4, 0x2d5, 0x2d6, 0x2d7, 0x2d8,
                                    ↪  0x2d9, 0x2da, 0x2db, 0x2dc, 0x2dd,
                                    ↪  0x2de,
83                                  0x2df, 0x2e0, 0x2e1, 0x2e2, 0x2e3, 0x2e4,
                                    ↪  0x2e5, 0x2e6, 0x2e7, 0x2e8, 0x2e9,
                                    ↪  0x2ea,
84                                  0x2eb, 0x2ec, 0x2ed, 0x2ee, 0x2ef, 0x2f0,
                                    ↪  0x2f1, 0x2f2, 0x2f3, 0x2f4, 0x2f5,
                                    ↪  0x2f6,
85                                  0x2f7, 0x2f8, 0x2f9, 0x2fa, 0x2fb, 0x2fc,
                                    ↪  0x2fd, 0x2fe, 0x2ff, 0x300, 0x301,
                                    ↪  0x302,
86                                  0x303, 0x304, 0x305, 0x306, 0x307, 0x308,
                                    ↪  0x309, 0x30a, 0x30b, 0x30c, 0x30d,
                                    ↪  0x30e,
87                                  0x30f, 0x310, 0x311, 0x312, 0x313, 0x314,
                                    ↪  0x315, 0x316, 0x317, 0x318, 0x319,
                                    ↪  0x31a,
88                                  0x31b, 0x31c, 0x31d, 0x31e, 0x31f, 0x320,
                                    ↪  0x321, 0x322, 0x323, 0x324, 0x325,
                                    ↪  0x326,
89                                  0x327, 0x328, 0x329, 0x32a, 0x32b, 0x32c,
                                    ↪  0x32d, 0x32e, 0x32f, 0x330, 0x331,
                                    ↪  0x332,
90                                  0x333, 0x334, 0x335, 0x336, 0x337, 0x338,
                                    ↪  0x339, 0x33a, 0x33b, 0x33c, 0x33d,
                                    ↪  0x33e,
91                                  0x33f, 0x340, 0x341, 0x342, 0x343, 0x344,
                                    ↪  0x345, 0x346, 0x347, 0x348, 0x349,
                                    ↪  0x34a,
92                                  0x34b, 0x34c, 0x34d, 0x34e, 0x34f, 0x350,
                                    ↪  0x351, 0x352, 0x353, 0x354, 0x355,
                                    ↪  0x356,
93                                  0x357, 0x358, 0x359, 0x35a, 0x35b, 0x35c,
                                    ↪  0x35d, 0x35e, 0x35f, 0x360, 0x361,
                                    ↪  0x362,
```

```
94                                  0x363, 0x364, 0x365, 0x366, 0x367, 0x368,
                                    ↪  0x369, 0x36a, 0x36b, 0x36c, 0x36d,
                                    ↪  0x36e,
95                                  0x36f, 0x370, 0x371, 0x372, 0x373, 0x374,
                                    ↪  0x375, 0x376, 0x377, 0x378, 0x379,
                                    ↪  0x37a,
96                                  0x37b, 0x37c, 0x37d, 0x37e, 0x37f, 0x380,
                                    ↪  0x381, 0x382, 0x383, 0x384, 0x385,
                                    ↪  0x386,
97                                  0x387, 0x388, 0x389, 0x38a, 0x38b, 0x38c,
                                    ↪  0x38d, 0x38e, 0x38f, 0x390, 0x391,
                                    ↪  0x392,
98                                  0x393, 0x394, 0x395, 0x396, 0x397, 0x398,
                                    ↪  0x399, 0x39a, 0x39b, 0x39c, 0x39d,
                                    ↪  0x39e,
99                                  0x39f, 0x3a0, 0x3a1, 0x3a2, 0x3a3, 0x3a4,
                                    ↪  0x3a5, 0x3a6, 0x3a7, 0x3a8, 0x3a9,
                                    ↪  0x3aa,
100                                 0x3ab, 0x3ac, 0x3ad, 0x3ae, 0x3af, 0x3b0,
                                    ↪  0x3b1, 0x3b2, 0x3b3, 0x3b4, 0x3b5,
                                    ↪  0x3b6,
101                                 0x3b7, 0x3b8, 0x3b9, 0x3ba, 0x3bb, 0x3bc,
                                    ↪  0x3bd, 0x3be, 0x3bf, 0x3c0, 0x3c1,
                                    ↪  0x3c2,
102                                 0x3c3, 0x3c4, 0x3c5, 0x3c6, 0x3c7, 0x3c8,
                                    ↪  0x3c9, 0x3ca, 0x3cb, 0x3cc, 0x3cd,
                                    ↪  0x3ce,
103                                 0x3cf, 0x3d0, 0x3d1, 0x3d2, 0x3d3, 0x3d4,
                                    ↪  0x3d5, 0x3d6, 0x3d7, 0x3d8, 0x3d9,
                                    ↪  0x3da,
104                                 0x3db, 0x3dc, 0x3dd, 0x3de, 0x3df, 0x3e0,
                                    ↪  0x3e1, 0x3e2, 0x3e3, 0x3e4, 0x3e5,
                                    ↪  0x3e6,
105                                 0x3e7, 0x3e8, 0x3e9, 0x3ea, 0x3eb, 0x3ec,
                                    ↪  0x3ed, 0x3ee, 0x3ef, 0x3f0, 0x3f1,
                                    ↪  0x3f2,
106                                 0x3f3, 0x3f4, 0x3f5, 0x3f6, 0x3f7, 0x3f8,
                                    ↪  0x3f9, 0x3fa, 0x3fb, 0x3fc, 0x3fd,
                                    ↪  0x3fe,
107                                 0x3ff,};
108
109 #endif /* SRC_AD5270BRMZ_LOOKUP_H_ */
```

A.3 Rauschtoleranz

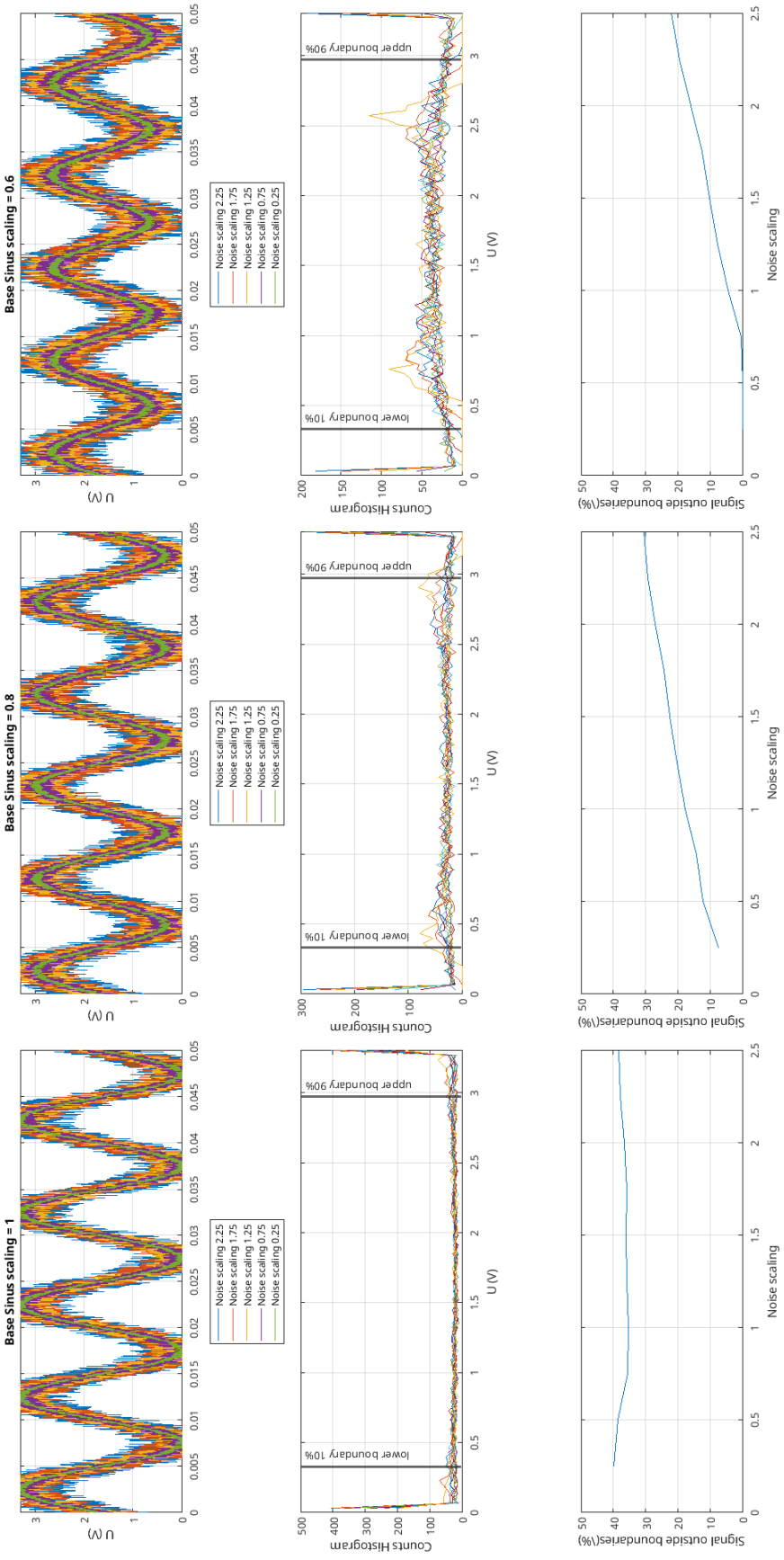


Abbildung A.2: Die Abbildung zeigt, wie sich der Schwellwert bei unterschiedlichen Rausch- und Signalskalierungen, bei einem festgelegten Grenzbereich von 10% verhält. Bildquelle: Dr. Florian Rittweger

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