Bachelorthesis

Julian Akwaa

Anregeschaltung und Controllersoftware für die elektrochemische Impedanzspektroskopie bei Lithium-Fahrzeug-Batterien

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Julian Akwaa

Anregeschaltung und Controllersoftware für die elektrochemische Impedanzspektroskopie bei Lithium-Fahrzeug-Batterien

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
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.-Ing. Jürgen Vollmer

Abgegeben am 8. Juni 2016

Julian Akwaa

Thema der Bachelorthesis

Anregeschaltung und Controllersoftware für die elektrochemische Impedanzspektroskopie bei Lithium-Fahrzeug- Batterien

Stichworte

Elektrochemische Impedanzspektroskopie, EIS, Batterie, Leistungstransistor

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung einer Anregeschaltung für die elektrochemische Impedanzspektroskopie. Hierfür wird die elektronische Schaltung und die zugehörige Steuersoftware für die Erzeugung eines zeitlich veränderlichen Entladestroms an einer Lithium- Ionen-Fahrzeug-Batterie konzipiert, realisiert und ihre grundsätzliche Funktionalität erprobt. Dabei sollen Erkenntnisse gewonnen und eine Grundlage geschaffen werden, um das bestehende Messsystem um ein Anregesystem zu erweitern und als autonomes Gesamtmesssystem zu realisieren.

Julian Akwaa

Title of the paper

Stimulus Circuit and Controller Software for the electrochemical Impedance Spectroscopy at Lithium Ion Automotive Batteries

Keywords

Electrochemical impedance spectroscopy, EIS, battery, power transistor

Abstract

In this thesis the development of an electrical circuit for the electrochemical impedance spectroscopy is discussed. The circuit is designed to modulate a discharge current on lithium ion batteries for vehicular use. The goal of this work is to gain experience and create a foundation for further development in order to implement a stand-alone measurement system.

Danksagung

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

Mein Dank gebührt Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider, der mir die Möglichkeit gegeben hat, diese Arbeit, im Rahmen des Forschungsprojekts „Drahtlose Zellensensoren für Autobatterien" zu schreiben.
Weiterhin möchte ich mich bei dem Team des Projekts bedanken. Das Klima im Team war stets positiv und Studenten und Mitarbeiter waren stets bereit, bei Problemen zu Helfen und mit Ideen und Anregungen zum Fortschritt beizutragen.
In diesem Zusammenhang möchte ich mich auch bei Günther Müller, Sergej Pereguda und Valentin Roscher bedanken, die mir bei Bedarf, mit Rat und Tat zur Seite standen.
Ganz besonders möchte ich mich bei meinem Betreuer Nico Sassano bedanken, der das Ergebnis dieser Arbeit stets mit Optimismus, Anregungen und helfender Hand gefördert hat.

Nur durch Unterstützung, die ich hatte, konnte das Ergebnis der Arbeit das werden, was es geworden ist.

Inhaltsverzeichnis

1. Einführung

In heutigen Fahrzeugen sind Batterien als Starterbatterie und für Elektronik und Multimedia unverzichtbar. Zukunftsweisende Elektrofahrzeuge benötigen leistungsfähige Batterien, die kurze Ladezeiten, große Kapazitäten und eine lange Lebensdauer aufweisen sollen. Um diese Anforderungen zu gewährleisten und um Batterien während des Betriebs zu überwachen, werden Batterie-Management-Systeme benötigt. Diese müssen, nicht nur in der Lage sein, den aktuellen Ladezustand (SoC) einer Batterie zu ermitteln, sondern auch Aussagen über den Alterungszustand (SoH) treffen. Im Zuge der zunehmenden Popularität von Elektromobilität, steigt der Bedarf der Fahrzeugindustrie an Batterie-Management-Systemen zur Batterieüberwachung kontinuierlich.
Eine Arbeitsgruppe der Hochschule für angewandte Wissenschaften Hamburg, unter der Leitung von Herrn Prof. Dr.-Ing. Riemschneider, befasst sich innerhalb mehrerer Projekte, im Zusammenhang mit Elektromobilität, mit dem Thema Batterien. Im Zuge dessen wird seit 2013 in dem Projekt „Drahtlose Zellensensoren für Autobatterien" ein System entwickelt, dass in der Lage sein wird, die Messmethode der elektrochemischen Impedanzspektroskopie (EIS) an Batterien durchzuführen. Geräte zur Durchführung von EIS-Messungen sind üblicherweise platzraubend und teuer. Das entwickelte System soll hingegen in der Lage sein, die Messung in situ, beispielsweise in einem Kraftfahrzeug, durchzuführen. Das bestehende System ist bedeutend kleiner als handelsübliche Messgeräte. Es ist bereits in der Lage, die Messungen an jeder, der in Reihe geschalteten Batteriezellen einer Batterie durchzuführen. Das System besteht aus einem Batteriesteuergerät, von dem aus der Benutzer Messungen und weitere Aktionen initialisieren kann und aus einem Zellensensor für jede Zelle der zu messenden Batterie. Eine bedeutende technische Errungenschaft dabei ist, dass die Kommunikation zwischen dem Batteriesteuergerät und den Zellensensoren kabellos durchgeführt wird. Die Zellensensoren sind in der Lage, die Temperatur und Spannung einer Zelle zu überwachen. Mit Hilfe des Batteriesteuergeräts kann der Strom durch die Zelle ermittelt werden. Als Resultat der Durchführung der EIS-Messung an einer Batterie, können Aussagen über den SoC und den SoH der einzelnen Zellen getätigt werden. Zudem ist das System in der Lage, Ladungsbalancierungen an einzelnen Zellen vorzunehmen [1].

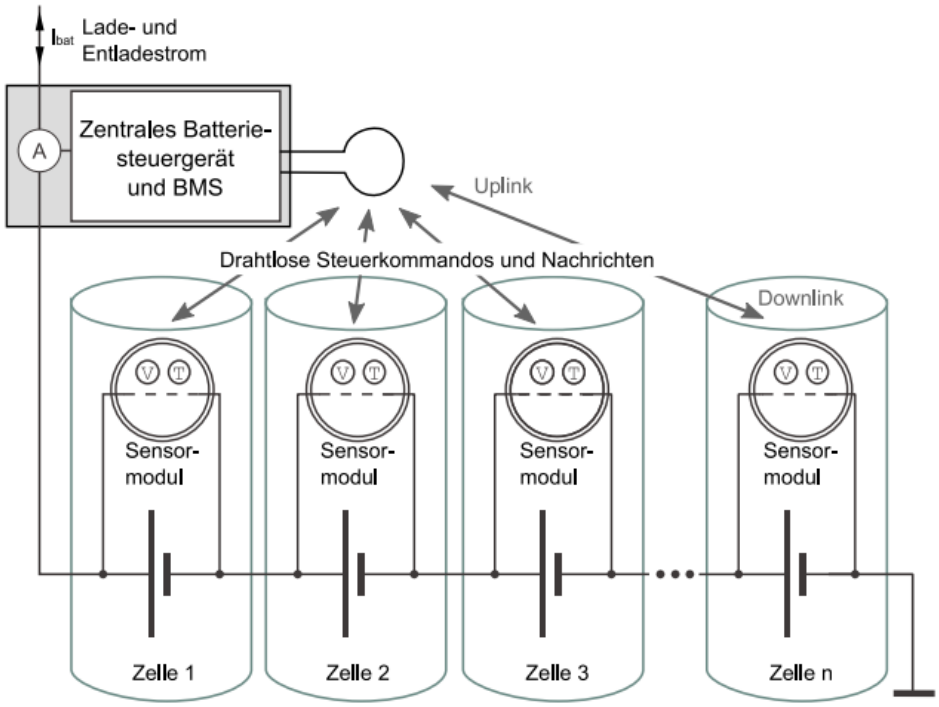

Abbildung 1.1.: Aufbau des Batterie-Management-Systems (BMS). Es zeigt das Prinzip des Messsystems zur Zellenüberwachung nach [2].

1.1. Der aktuelle Stand des Projekts Drahtlose Zellensensoren für Autobatterien

Um den Forstschritt des Projekts „Drahtlose Zellensensoren für Autobatterien" zu bewerten, wird im Folgenden die Einstufung anhand des Technology Readiness Level (TRL) herangezogen, um den Stand des Projekts besser einordnen und die Zielsetzung dieser Arbeit klar definieren zu können.
Das Technology Readiness Level ist eine erstmals in den späten achtziger Jahren von der NASA aufgestellte Skala, anhand derer die Marktreife einer neu entwickelten Technologie bewertet wird. Im Zuge des EU-Förderprogramms Horizont 2020 zur Förderung von Forschung und Innovation, wird das TRL nach wie vor als Maßstab zur Einstufung von Produktentwicklung verwendet. Es gliedert sich in neun Stufen der Entwicklung, Technology Readiness Level 1 bis 9. Diese sind in Abbildung 1.2 abgebildet [3].

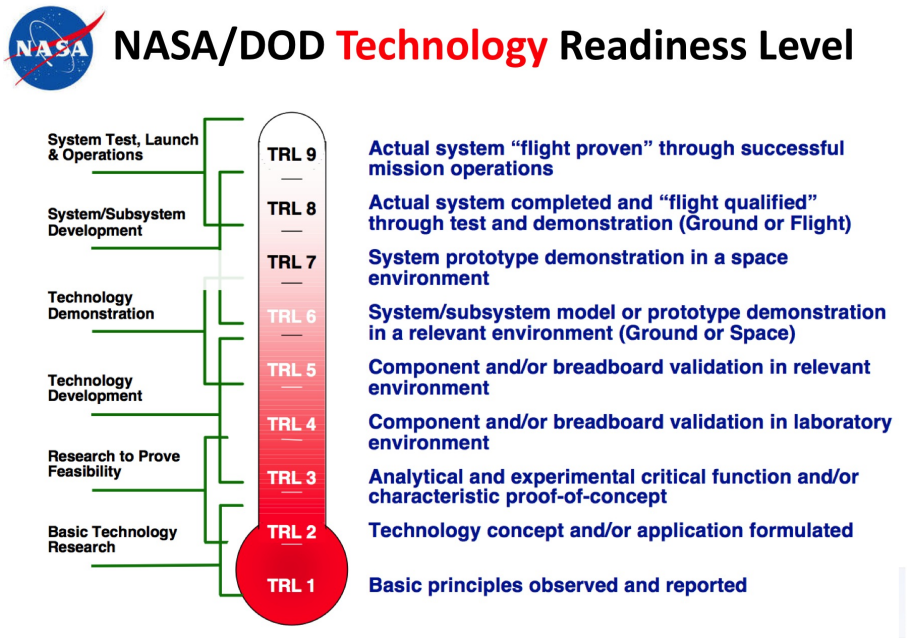

Abbildung 1.2.: Technology-Readiness-Level-Skala der NASA [3]. Sie ordnet den Stand eines Projekt in der Forschung und Entwicklung in einen Fortschrittsstand ein.

Das Projekt lässt sich zum gegenwärtigen Zeitpunkt in das TRL 4 einordnen. Mit der Arbeit [1] wurde der experimentelle Beleg des Konzepts erbracht. Konzept und Durchführung haben sich im Rahmen der Laborversuche als erfolgreich erwiesen. Das Ziel ist, die Messung und die Auswertung einer elektrochemischen Impedanzspektroskopie autonom, etwa in einem Fahrzeug, durchführen zu können. Damit würde das Projekt in TRL 6 fallen und den Grundstein für den industriellen Einsatz derartiger Batteriemanagementsysteme auf Basis des entwickelten Systems legen.

1.2. Motivation

Durch die in dieser Arbeit entwickelten Anregung und einer zukünftigen Implementierung dieser in das bestehende System, können die Kriterien für TRL 5 erfüllt werden. Damit wäre eine autonome EIS-Messung einer Fahrzeugbatterie möglich. In der Thesis „Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISMBand bei 434 MHz" von Phillip Durdaut [4] wurde 2013 erstmals das Grundkonzept für das Gesamtsystem ausgearbeitet und dokumentiert. Eine erste Version eines Batteriesteuergeräts und den dazugehöri-

gen Zellensensoren wurde entwickelt und getestet. Mittlerweile existiert eine weiterentwickelte Version der Zellensensoren. Diese sind in der Lage, kabellos bidirektional zu kommunizieren. Dabei können sie Informationen über Zellenspannung und Temperatur ermitteln und verarbeiten. Mit Hilfe des Batteriesteuergeräts können so aussagekräftige EIS-Messungen durchgeführt werden. Abbildung 1.3 zeigt schematisch wie eine EIS-Messung durchgeführt wird. Die die notwendige Anregung der Batterie ist jedoch noch nicht in das System implementiert und muss mit Hilfe externer Laborhardware durchgeführt werden. Deshalb soll in dieser Arbeit ein System zur Anregung einer Batterie für die EIS entwickelt werden. Sie soll einen wichtigen Beitrag zur Möglichkeit leistet, die EIS-Messung autonom durchzuführen.

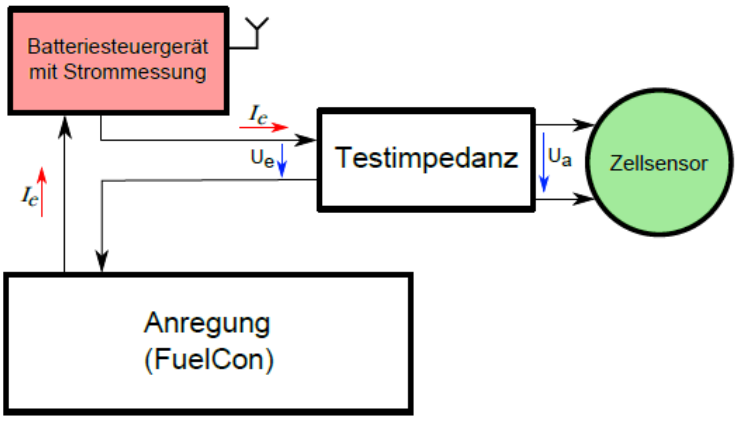

Abbildung 1.3.: Aufbau zur EIS-Messung mit dem bestehenden System nach [1]. Die benötigte Anregung des Prüflings erfolgt extern durch ein EIS-Meter.

1.2.1. Die elektrochemische Impedanzspektroskopie

Die elektrochemische Impedantzspektroskopie wird heute vorwiegend in der Materialforschung sowie in der Medizintechnik verwendet [5]. Sie spielt aber auch im Bereich elektrochemischer Energiespeicher eine große Rolle. Bei der EIS wird ein zu messender Prüfling, aktiv oder passiv, mit einer elektrischen Größe (Strom oder Spannung) angeregt, das heißt ge- oder entladen. Die elektrische Größe besteht dabei aus einem Wechselanteil. Nach bedarf wird zusätzlich ein Gleichanteil dazugegeben. Ziel ist es, das Impedanzverhalten des Prüflings innerhalb einer gewissen Frequenzbandbreite zu ermitteln. Jede Messung von Strom und Spannung über das zu messende System, ergibt einen komplexen Widerstand (Impedanz), bestehend aus einem reellem und einem imaginären Widerstandsanteil. Werden die Messergebnisse verschiedener Frequenzen im Bode-Diagramm aufgetragen, so resultiert das zugehörige Impedanzspektrum. Zur Auswertung der Messungen können

die ermittelten Werte alternativ auch als Nyquist-Plot beziehungsweise als Ortskurve in der Gauß'schen Ebene aufgetragen werden.
Wird die EIS bei Batteriezellen angewendet, so ergibt sich im Nyquist-Plot aufgetragen eine charakteristische Ast-Form (Abbildung 1.4). Aus ihr lassen sich unter anderem Aussagen über den SoC und den SoH der gemessenen Batteriezelle treffen. Zusätzlich lässt sich Stärke von Einflüssen verschiedener, in bestimmten Batteriemodellen festgehaltener Größen bestimmen, welche Einfluss auf das Batterieverhalten haben [5].

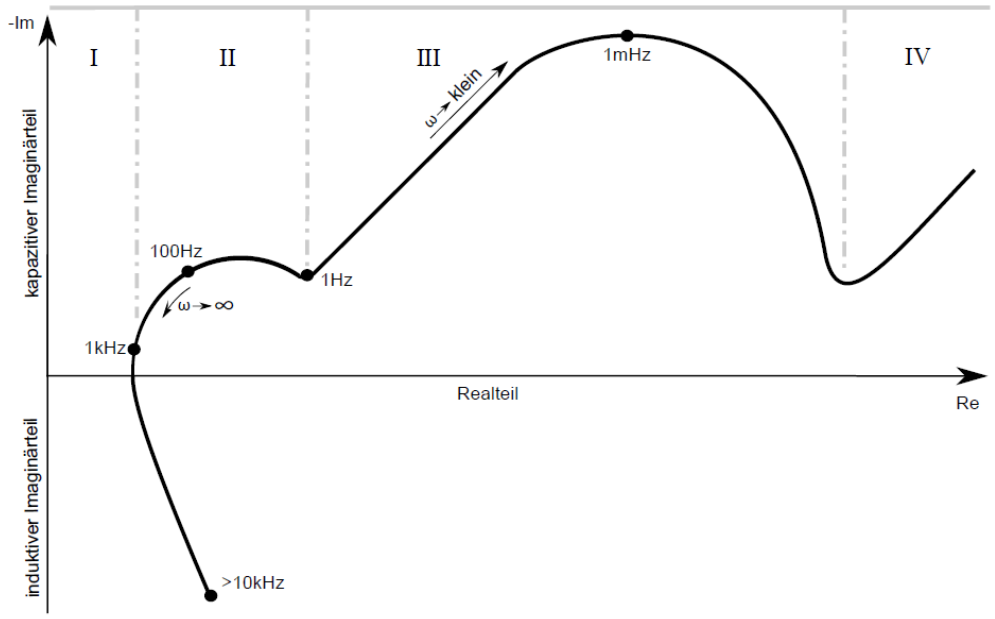

Abbildung 1.4.: Ergebnis einer EIS-Messung nach [6]. Sie hat die typische Ast-Form im Nyquist-Plot. Dabei stellen sich die Ergebnisse der EIS in den Bereichen I. bis IV., durch verschiedene chemische Einflüsse innerhalb der gemessenen Batterie ein.

1.2.2. Anregung für die EIS

Die Anregung ist ein elementarer Teil der EIS. Sie ermöglicht erst, die Messung von Impedanzen bei verschiedenen Frequenzen. Batteriezellen können für die Durchführung einer EIS-Messung mittels eingeprägtem moduliertem Strom geladen oder entladen werden. Dabei wird der Strom bei einer gewünschten Frequenz moduliert. Für diese Arbeit wird ein Sinus als Signalform der Anregung gewählt, da die Ergebnisse der EIS mit sinusförmiger Anregung bekannt und geeignet sind [1]. Die Anregung kann mit aktiver oder passiver Anregung erfolgen. Die Aktive Anregung lädt die Batterie mit einem modulierten Strom. Die

Passive Anregung entlädt die Batterie mit einem modulierten Strom. Soll das System in Fahrzeugen mit Verbrennungsmotoren Anwendung finden, würde sich die aktive Anregung aufwändig gestalten. Zwar werden Fahrezugbatterien üblicherweise bei laufendem Motor durch die Lichtmaschine geladen, jedoch ist die Modulation des Ladestrom aufwändig. In Elektrofahrzeugen wäre dies ausschließlich während des Ladevorgangs der Batterie an einer Ladestation möglich. Dagegen steht die passive Anregung, d.h. die Entladung der Zelle zur Durchführung der EIS. Die Realisierung der passiven Anregung ist weit weniger kompliziert, da die Batterie künstlich entladen werden kann. Wird die Modulation einer Entladung für eine Messung durchgeführt, so muss lediglich die von der Bordnetz-Seite ausgehende Gesamtbelastung der Batterie berücksichtigt werden, um die EIS-Messung nicht zu verfälschen. Es muss hierzu zu ein geeigneter Zeitpunkt bezüglich der Belastung der Fahrzeugbatterie ermittelt werden, in dem die EIS Messung durchführt werden kann. Die größte Belastung für die Batterie liegt bei Fahrzeugen mit Verbrennungsmotor zum Zeitpunkt des Motorstarts vor. Zum Betreiben des Anlassers muss die Batterie hohe Ströme zur Verfügung stellen. Ein geeigneter Zustand ist der Parkzustand, in dem ein Großteil der Fahrzeugsysteme abgeschaltet sind und die Batterie nur minimal belastet wird.

1.3. Ziel dieser Arbeit

Um das bestehende Messsystem autonom verwendbar zu machen, muss dieses um die Funktionalität erweitert werden, die zu messende Batterie selbstständig bei der gewünschten Frequenz und mit den gewünschten Strömen anregen zu können. Dabei sollen basierend auf dem bestehenden Messsystem Frequenzen von 0,1 Hz bis 2 kHz erzeugt werden können [1]. Der Lade- bzw. Entladestrom muss einen ausreichend großen Spannungsabfall an den Innenwiderständen der Zellen hervorrufen. Diese liegen im Milliohm-Bereich. Die Ströme müssen entsprechend groß sein. Die Ströme bestehen aus einem Gleichanteil und einem zusätzlichem Wechselanteil in Sinusform. Die Erzeugung der Ströme soll über Leistungstransistoren erfolgen, welche mittels Mikrocontroller angesteuert werden. Eine entsprechende Schaltung soll entworfen und realisiert werden. Die Auswahl und Dimensionierung von Bauteilen soll begründet werden. Die Software zur Ansteuerung der Transistoren soll so geschrieben werden, dass die Parameter Gleichstrom, Wechselstrom und Frequenz einstellbar sind. Das Anregesystem soll so an das bestehende Batteriesteuergerät angebunden werden können, dass es eine Automatisierung der Anregung bei verschiedenen Frequenzen ermöglicht, siehe Abbildung 1.5. Die Funktionalität der Schaltung soll bewertet werden. Abschließend sollen Messungen des erweiterten Gesamtsystem mit den Ergebnissen eines industriellen EIS-Messgeräts verglichen werden.

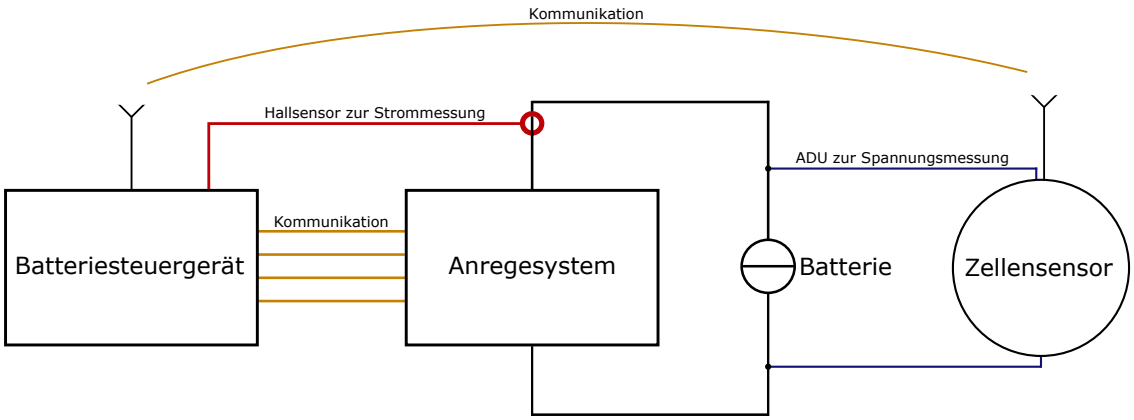

Abbildung 1.5.: Der Gesamtmessaufbau des Messsystems mit dem zu entwickelnden Anregesystem.

2. Aufbau der Leistungsstufe der Anregung

Zur praktischen Erprobung der Anregung wird zunächst eine Schaltung im Labor aufgebaut, um die grundsätzliche Funktionalität zu überprüfen und die Design-Entscheidungen zu evaluieren. Um Messungen mit einer sauberen Anregung durchzuführen, wird nach Abschluss des grundsätzlichen Designs, ein kompakter Hardwareaufbau realisiert. Bei diesem können Messungenauigkeiten aufgrund des fliegenden Aufbaus, langer Laborkabel usw. vermieden werden, die der Laboraufbau mit sich bringt. Es stellt mit dem bestehenden Batteriesteuergerät den ersten Prototypen für ein autonomes EIS-Messsystem dar.

2.1. Anforderung an die Schaltung

Die zu entwickelnde Schaltung soll als Stromsenke dienen, die ermöglicht, eine Batterie mit einem Strom vorgegebener Stärke mit einem Gleich- und einem Wechselanteil zu entladen. Um diesen Strom dimensionieren zu können, muss sich an der Batterie ein Lastwiderstand einstellen, der den geforderten Strom in Abhängigkeit der aktuellen Zellenspannung hervorruft.

Mit der Änderung des SoC, ändert sich auch die Spannung einer Batterie. Im voll geladenem Zustand hat sie die maximale Spannung. Im entladenen Zustand ist ihre Spannung am geringsten. Man spricht von Ladeschlussspannung und Entladeschlussspannung. Im Labor werden die Zellen $ANR26650\ 3,3\,V; 2,5\,Ah$ [7] und $ECC-LFPP\ 3,3\,V; 45\,Ah$ [8] verwendet. Diese haben jeweils Entladeschlussspannungen und Ladeschlussspannungen, die im Datenblatt nicht genau definiert sind. Sie liegen etwa im Bereich von 2,5 V bis 2,8 V beziehungsweise 3,4 V bis 3,6 V. Nach Erfahrung verschiedener Mitglieder des Projekts liegen die Schlussspannungen bei 2,5 V und 3,6 V. Deshalb wird in der weiteren Arbeit von diesen Spannungen als Lade- und Entladeschlussspannungen ausgegangen.
Die Anregung soll mindestens eine Signalfrequenz im Bereich von 0,1 Hz bis 2 kHz bei einem Sinussignal einschließen. Auch die Aussteuerung mit anderen Signalformen soll mit entsprechenden Frequenzen potentiell ermöglicht werden. Zunächst wird sie jedoch für ein

Sinussignal ausgelegt, da die zu erwartenden Ergebnisse der EIS-Messung bei diesem Signal bekannt sind. Hierbei soll für eine EIS-Messung der oben genannten ECC-Zelle nach [1] ein so großer Strom aus der Zelle fließen, dass eine ausreichend große Spannung über ihrem Innenwiderstand beziehungsweise Innenimpedanz abfällt [1]. Im Folgenden sind die Anforderungen noch einmal zusammengefasst.

- Soll geeignet sein für Batteriezellen mit einem Innenwiderstand von ca. 1 ... 10 mΩ (bei 1 kHz)
- Anregestrom: Ausreichend hoch, um den für die Messung notwendige Spannungsamplitude am Innenwiderstand zu erzeugen
- Signalfrequenz: 0,1 ... 2 kHz

Die Schaltung soll dabei mittels Software, über das bestehende Batteriesteuergerät, angesteuert werden können. Das System soll zusätzlich in der Lage sein, den Entladestrom über eine Rückführung ermitteln zu können. Es kann diesen dann mit dem geforderten Strom vergleichen. Das ermöglicht es, den geforderten Strom korrekt einzustellen.

12 V-Fahrzeugbatterien sind elektronisch betrachtet vier in Reihe geschaltete 3,3 V-Zellen. Die maximale Batteriespannung einer solchen Batterie ergibt sich aus der Summe Ladeschlussspannungen der einzelnen Zellen. Die Gesamt-Ladeschlussspannung einer solchen Batterie liegt bei $4 \cdot 3,6\,V = 14,4\,V$. Bei angestrebten Entladeströmen von bis zu insgesamt 12 A und Spannungen bis zu 14,4 V ergibt sich eine maximale Gesamtverlustleistung von 144 W über dem Entladepfad.

$$P_{V,max} = 10\,A \cdot 14,4\,V = 144\,W \tag{2.1}$$

Die zu entwickelnde Schaltung muss in der Lage sein, die Wärme dieser Verlustleistung abzuführen. Das ermöglicht die Anregung mit beliebigen Signalformen bei beliebigen Frequenzen. Gegebenenfalls muss für diesen Fall eine zusätzliche Kühlung eingesetzt werden.

2.1.1. Notwendiger Gleichstromanteil der Anregung

Der Gleichanteil der Anregung für die EIS wird zur Messung der Impedanz per Definition nicht berücksichtigt, so wird auch im bestehenden Messsystem der Gleichanteil subtrahiert, sodass der gemessene DC-Wert zu Null wird. Der Gleichanteil ist aber dennoch notwendig, um den Arbeitspunkt der Batteriezelle einzustellen. Dies verringert, ähnlich wie beim Einstellen des Arbeitspunkts eines Transistors im A-Betrieb, Nichtlinearitäten, welche das Ergebnis der Impedanzmessung verfälschen würden [9]. Der Gleichanteil dient also zum Einstellen des Arbeitspunkts in den linearsten Bereich. Abbildung 2.1 beschreibt, wie sich ein Strom

bei Änderung von Potential an einem elektrochemischen System, wie einer Lithium-Ionen-Batterie, ändert. In dem, in Abbildung 2.1 dargestellten Fall, liegt der linearste Bereich bei 0 A. Der Gleichanteil sollte deshalb entsprechend 0 A betragen, damit die Aussteuerung so linear wie möglich ist. Oftmals liegt der Arbeitspunkt höher, sodass ein Gleichanteil benötigt wird. Es sei angemerkt, dass die Wechselamplitude möglichst klein sein sollte (Kleinsignalaussteuerung), damit sich die Anregung möglichst wenig aus dem linearen Bereich entfernt.

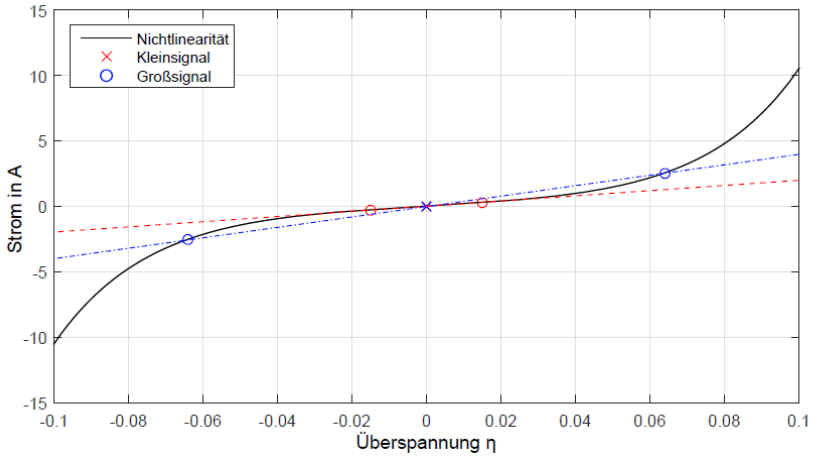

Abbildung 2.1.: Verlauf der Butler-Volmer-Gleichung bei einer Beispielzelle nach [1]. Die Zelle weist ein schlechtes lineares Verhalten für Großsignal- und für Kleinsignalaussteuerung auf. Der linearste Bereich liegt am Wendepunkt, Überspannung = 0 V. An diesem Punkt wirkt eine Anregung mit Wechselgrößen am wenigsten verzerrend.

2.1.2. Notwendiger Wechselanteil der Anregung

Der Entladestrom einer Zelle ruft einen Spannungsabfall über ihrem Innenwiderstand hervor. Dieser Spannungsabfall ist entscheidend, denn er wird, neben dem Strom, bei der EIS-Messung vom Zellensensor und dem Batteriesteuergerät gemessen, um die Impedanz des Prüflings zu ermitteln. Sensorseitig wird für die Messung lediglich der Wechselanteil berücksichtigt. Ferner sollte die Spannungsamplitude möglichst groß, bis hin zum maximalen Aussteuerbereich des Analog-Digital-Umsetzers sein, damit dieser genaue Spannungswerte aufnehmen kann. Zellenseitig muss bei der Wahl der Amplitude über dem Innenwiderstand beachtet werden, dass eine zu große Aussteuerung zu der Nichtlinearität der Zelle führt. In [1] wird darauf näher eingegangen. Relevant für eine Messung der Batterieimpedanz ist, dass der Prüfling auf einen Arbeitspunkt im linearen Bereich eingestellt wird, damit

das Strom-Spannungs-Verhältnis sich nicht in Abhängigkeit der Aussteuerung ändert. Am Verlauf der Butler-Volmer-Funktion Abbildung 2.1 ist zu erkennen, dass mit der Vergrößerung des Abstandes vom Arbeitspunkt (schwarzes Kreuz) einer Zelle, die Nichtlinearität des Strom-Spannungsverhältnisses steigt. Um hier Nichtlinearitäten in der Impedanzmessung zu vermeiden, muss die Anregung folglich mit einer möglichst geringen Amplitude durchgeführt werden.
Es muss also ein Kompromiss für die Größe der Entladeamplitude gefunden werden. Es gibt hierzu verschiedene wissenschaftliche Ergebnisse: Nach Macdonald [10] soll die Wechselspannungsamplitude den Wert von 10 mV nicht überschreiten. Laut Kiel [5] soll die Wechselspannungsamplitude über dem Innenwiderstand 3 mV nicht überschreiten. In Angold [11] wird eine Wechselspannungsamplitude von 5 mV angegeben. Um die Anregung flexibel zu gestalten, wäre es ideal, den zu erzeugenden Anregestrom in einem Strombereich einstellen zu können, der alle Amplitudenwerte abdeckt.
Die notwendigen Ströme, welche die Anregeschaltung zur Verfügung stellen muss, um die gewünschte Spannungsamplitude über dem Innenwiderstand der zu messenden Batteriezelle zu erzeugen, können anhand der Impedanz der Zelle und der Spannung, die an ihr abfallen soll, ermittelt werden. Dabei muss der Zellentyp berücksichtigt werden, denn der Innenwiderstand und das Impedanzverhalten variieren von Typ zu Typ (üblicherweise im Milliohm-Bereich). Es ist unter anderem abhängig vom SoC und vom SoH. Folglich muss der Anregestrom im Idealfall an den Innenwiderstand angepasst werden, um den exakten gewünschten Spannungsabfall hervorzurufen.

Zur Entwicklung des Zellensensorsystems wurden zunächst Versuche mit zwei Batterietypen gefahren. Die Batterie mit der kleineren Kapazität ist eine Lithium-Ionen-Zelle vom Typ *ANR26650* von *A123 Systems* [7]. Sie hat eine nominale Klemmspannung von 3,3 V und eine nominale Kapazität von 2,5 Ah. Ihre Innenimpedanz beträgt laut Datenblatt 6 mΩ bei 1 kHz. Die zweite, größere Batterie, ist eine Lithium-Ionen-Zelle vom Typ *ECC-LFPP 45 Ah* der *ECC Repenning GmbH* [8]. Sie hat ebenfalls eine nominale Klemmspannung von 3,3 V, eine nominale Kapazität von 45 Ah und eine Innenimpedanz von etwa 1 mΩ bei 1 kHz. Abbildung 2.2 zeigt die Stromwerte der beiden Zellen, die für die drei zuvor angegebenen Amplitudenwerte 3 mV, 5 mV und 10 mV über den Innenwiderständen benötigt werden. Diese Stromwerte beschreiben lediglich den Wechselanteil, der in Form einer Sinusaussteuerung auf die Batterie gegeben wird. Der für die Anregung nötige Gleichanteil wird bei dieser Überlegung zunächst vernachlässigt.

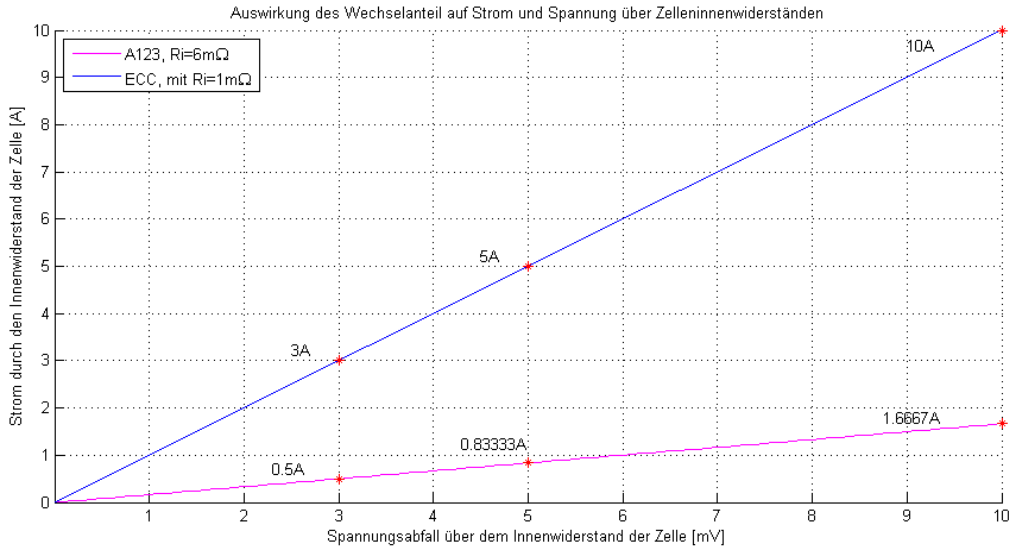

Abbildung 2.2.: Notwendiger Strom, um an den Innenwiderständen der Zelle einen bestimmten Spannungsabfall zu erzeugen. Markiert sind die Stromwerte bei einem Spannungsabfall von 3 mV, 5 mV und 10 mV. Dabei wird von den reellen Widerständen der Zellen bei 1 kHz ausgegangen.

Nach Abbildung 2.2 wird also eine Wechselstromamplitude von mindestens 500 mA bis etwa 1,7 A für die 2,5 Ah Batteriezelle und eine Mindestwechselstromamplitude von 3 A und eine maximale Amplitude von 10 A für die 45 Ah-Batteriezelle benötigt. Eine Schaltung zur Anregung für die EIS an beiden Zellentypen müsste somit in der Lage sein, zusätzlich zu einem Gleichstromanteil, einen Wechselstromanteil (Amplitude) von 500 mA bis 10 A zu erzeugen. In [1] wird die ECC 45 Ah-Zelle zur Messung mit einer Welchselamplitude von 2 A angeregt. Dieser Wert liegt zwar noch unter dem in Abblildung 2.2 ermitteltem Wert für einen Spannungsabfall von 3 mV, die Messungen haben aber gezeigt, dass trotzdem genaue und aussagekräftige Ergebnisse erzielt werden. Der Gleichanteil liegt bei diesen Messungen bei 10 A. Aufbauend auf [1] wird der für die Anregung benötigte Maximalstrom deshalb auf 10 A Gleichanteil mit 2 A Wechselamplitude für eine Anregung mit Sinussignal festgelegt.

2.2. Überlegungen zum Prinzip der Schaltung

Ziel ist es, eine Lithium-Ionen-Batterie mit einem Sinussignal oder in Zukunft auch mit anderen Signalformen zu entladen. Dazu soll eine Last angelegt werden, die den Entladestrom verursacht. Die Last muss im Widerstandswert bzw. im Strom-Spannungsverhältnis folglich

variabel sein. Als solche variable Last eignet sich ein Transistor, durch den, mittels entsprechender Ansteuerung, ein gewünschter Strom fließt. Die Ansteuerung soll durch eine Software vorgegeben werden, die auf einem Mikrocontroller läuft. Dabei soll das bestehende Batteriesteuergerät in der Lage sein, die Ströme für die EIS-Messung einzustellen. Hierzu werden Werte über einen Digital-Analog-Umsetzer in Spannungswerte konvertiert, die dann zur Ansteuerung des Transistors genutzt werden.

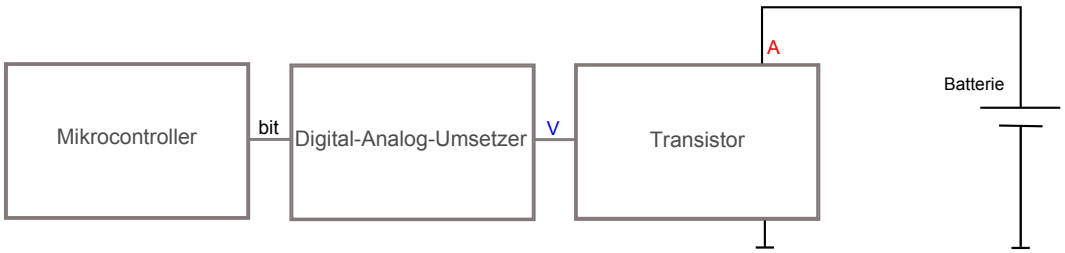

Abbildung 2.3.: Konzeptioneller Aufbau der Anregeschaltung.

Das Batteriesteuergerät hat im bestehenden System die Aufgabe, den Strom durch die Batterie zu messen und die Funkkommunikation mit den Zellensensoren durchzuführen. Hinzu kommt die Kommunikation mit dem PC, von dem aus die Messung gesteuert wird. Auf dem Steuergerät werden zur Zeit der Entwicklung dieser Arbeit sechs Interrupthandler ausgeführt (Abbildung 2.4). Für die EIS-Messung ist der Handler für die Burstmessung und Stromaufnahme der wichtigste, da die Timings für die Messung von Strom und Spannung zur selben Zeit, kritisch sind [1]. Über ihm liegen prioritär nur noch der Handler für Timer und der Systemtakt-Handler. Die Ausgabe eines zeitlich veränderlichen Signals mittels Digital-Analog-Umsetzers nach dem Prinzip aus Abbildung 2.3, erfordert eine kontinuierliche Ausgabe von veränderlichen Bit-Werten. Eine Realisierung durch Software erfordert deshalb eine kontinuierliche und vor allem zeitlich äquidistante Wertänderung. Ein hierfür vorgesehener Handler sollte also möglichst die höchste Priorität haben. Ein Eingriff in die Prioritätsverteilung des bestehenden, zeitkritischen Systems, die Veränderungen der Mess-Timings zur Folge haben könnte, sollen vermieden werden.

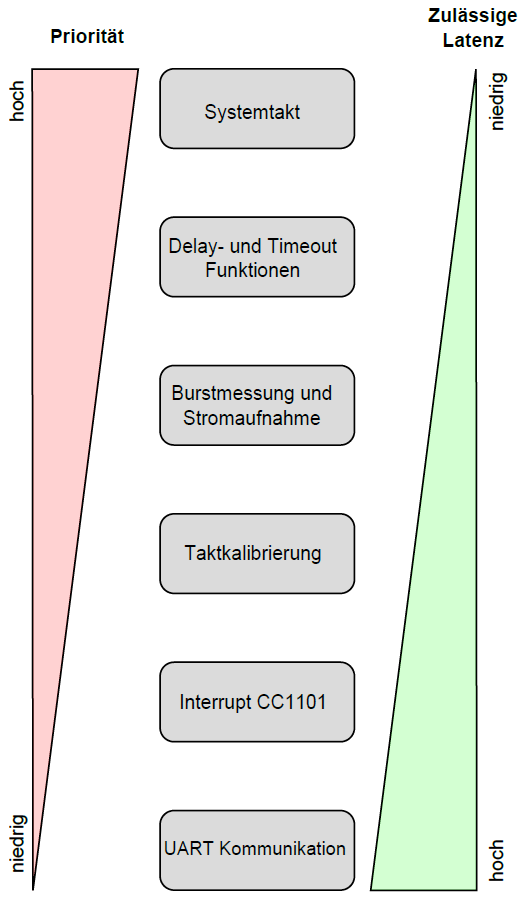

Abbildung 2.4.: Prioritäten der Interrupthandler der Software des Batteriesteuergeräts [1].

Aufgrund dessen wird entschieden, den Softwareteil des Anregesystems auf einem zweiten Mikrocontroller zu realisieren, anstatt diesen in die Software des bereits bestehenden Controllers zu implementieren. Die Entwicklung auf einem zweiten Controller ist für die erste Version eines Anregesystems zur Funktionsverifikation praktischer, da die neue Software, unabhängig von bestehender Software, geschrieben werden kann. Durch diese Art der Realisierung der Ansteuerkontrolle, muss eine allerdings eine Kommunikation zwischen beiden Controllern realisiert werden, damit das Steuergerät die Funktion des Masters beibehält.

2.3. Auswahl des Transistors

Ein Transistor, der wie in Abbildung 2.3 verwendet wird, fungiert als spannungs- bzw. stromgesteuerte Stromquelle. Dabei hat er eine verstärkende Wirkung. Ideal wäre eine Systembeschreibung des Transistors mit einem Verstärkungsfaktor G wie in Formel 2.2.

$$I_a = G \cdot I_e \quad \text{bzw.} \quad I_a = \frac{G}{[\Omega]} \cdot U_e \,. \tag{2.2}$$

So bestünde ein linearer Zusammenhang des Faktors G zwischen Eingangsgröße und Ausgangsgröße. Real ist dieser Zusammenhang jedoch bestenfalls annähernd und nur in bestimmten Arbeitsbereichen linear. Die Übertragungskennlinie eines Transistors zeigt diesen Zusammenhang zwischen Eingangs- und Ausgangsgröße (Abbildung 2.5).

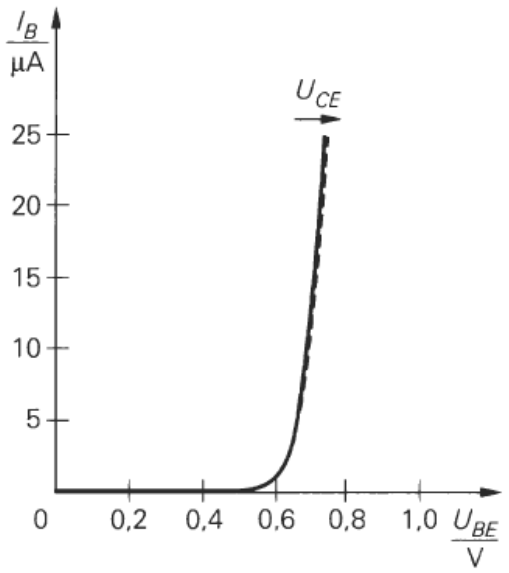

Abbildung 2.5.: Eingangskennlinienfeld eines Bipolartransistors nach [9].

Aufgrund der Nichtlinearität der Übertragungskennlinie entstehen Verzerrungen in der Ausgangsgröße. Das Ausgangssignal entspricht also nicht exakt dem Eingangssignal. Um diese Verzerrungen zu minimieren, wird für den Wechselanteil der Stromanregung angestrebt, den verwendeten Transistor im linearen Bereich zu verwenden, um ein möglichst verzerrungsfreies Stromsignal zu erzeugen, welches dem erzeugten Spannungssteuersignal möglichst entspricht. Nichtlinearitäten können aufseiten der Software vorkompensiert werden. Dies

ist nicht nur umständlich, zeitaufwändig und unter Umständen recht ungenau, es würde auch einen Wechsel des Transistors, z.B. aufgrund von zukünftiger Weiterentwicklung des Systems, und eine angepasste Software erfordern. Die Linearisierung müsste also für jeden Transistor erneut durchgeführt werden.

2.3.1. Auswahl von Feldeffekt- oder Bipolartransistor

Es muss entschieden werden, ob ein Feldeffekt-Transistor oder ein Bipolar-Transistor als Senke in der Schaltung verwendet werden soll. Um dies zu klären, werden zunächst die wesentlichen Unterschiede der beiden Transistortypen erläutert und verglichen. Aufbauend auf den Ergebnissen des Vergleichs kann dann entschieden werden, welcher Transistortyp sich für die erste Realisierung der Schaltung besser eignet. Dazu sind zunächst die unterschiedlichen Eigenschaften zu betrachten. Im Folgenden werden lediglich das grundsätzliche Verhalten sowie die für die Begründung relevanten mathematischen Annäherungen erläutert, da diese ausreichen, um die Wahl des Transistors zu begründen.
Für die Anwendung in einer Schaltung nach Kapitel 2.2 werden lediglich npn-Bipolartransistoren und äquivalente selbstsperrende n-Kanal Metall-Oxid-Schicht Feldeffekttransistoren (MOSFET) betrachtet. Weitere Transistortypen werden nicht berücksichtigt.

Sowohl der Bipolar-, als auch der Feldeffekttransistor sind Halbleiterbauteile, die integriert, aber auch einzeln in elektronischen Schaltungen Anwendung finden. Einzeln verfügen beide Standardvarianten über drei Anschlüsse, Basis, Kollektor und Emitter bzw. Gate, Drain und Source. Der Basis- bzw. der Gate-Anschluss wird mit einer Steuerspannung angesteuert. Über der Kollektor-Emitter- bzw. der Drain-Source-Strecke stellt sich, in Abhängigkeit der Steuerspannung, dann eine Spannung respektive ein entsprechender Strom ein.

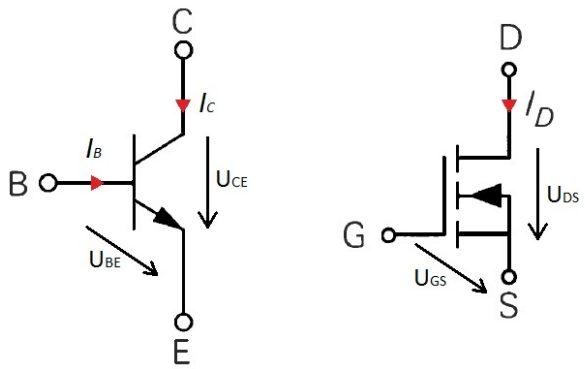

Abbildung 2.6.: Schaltzeichen für einen npn-Bipolartransistor (links) und einen n-Kanal MOSFET (rechts), verändert nach [9].

Linearität

Wesentlich für die Wahl des Typs ist das Übertragungskennlinienfeld (Abbildung 2.7). Es beschreibt den Kollektorstrom I_C in Abhängigkeit der Steuerspannung U_{BE}. Der Verlauf der Kennlinie resultiert aus der Shockley-Gleichung, die aus dem pn-Übergang der Basis-Emitter-Strecke resultiert. Formel 2.3 beschreibt den Verlauf der Übertragungskennlinie.

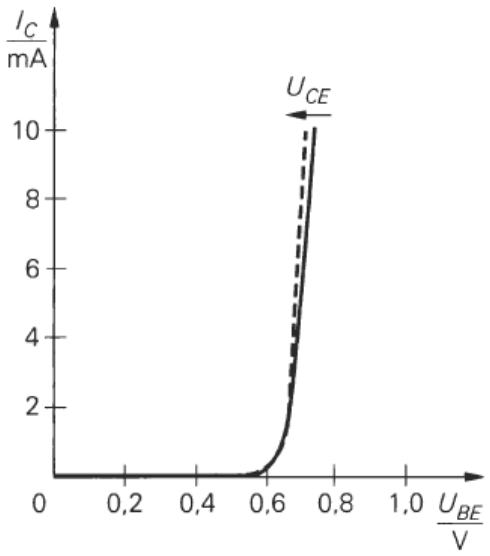

Abbildung 2.7.: Übertragungskennlinienfeld eines Bipolartransistors nach [9]. Im Bereich über 2 mA der Kennlinie ist der Kurvenverlauf annähernd linear. Dies ist der Bereich, in dem ausgesteuert wird, wenn eine lineare Übertragung erreicht werden soll.

Der Kollektorstrom I_C berechnet sich nach Formel 2.3. Dabei ist I_S der Sperrstrom, der bei einer nichtleitenden Diode fließt, U_T die Temperaturspannung, die sich aus Temperatur, Boltzmann-Konstante und Menge der Elementarladungen ergibt, ($U_T = \frac{k_B \cdot T}{e}$) und U_A die Early-Spannung, die sich aus dem Schnittpunkt der verlängerten Ausgangskennlinie im negativen Bereich mit der U_{CE}-Achse ergibt (Siehe Abbildung 2.8) [9].

$$\text{Kollektorstrom } I_C(U_{BE}) = I_S \cdot e^{\frac{U_{BE}}{U_T}} \cdot \left(1 + \frac{U_{CE}}{U_A}\right) \tag{2.3}$$

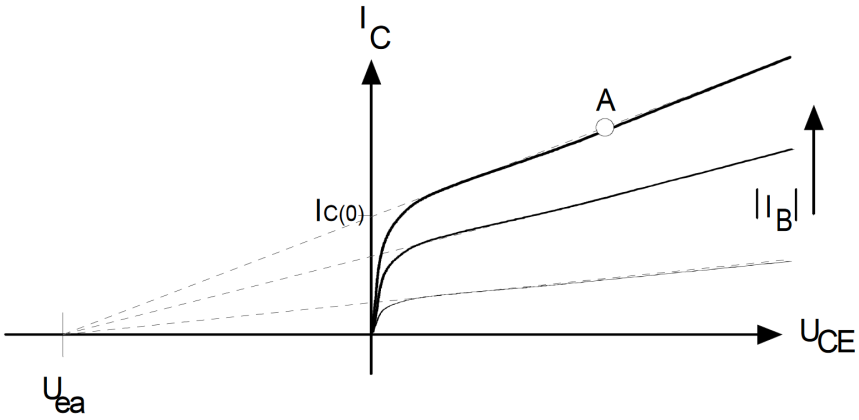

Abbildung 2.8.: Grafisches Ermitteln der Early-Spannung U_{ea} anhand der Ausgangskennlinie nach [12]. Sie führt zu der leichten Steigung des Kurvenverlaufs am Arbeitspunkt A.

Aus der idealisierten Formel für die Stromverstärkung wird ersichtlich, dass ein linearer Zusammenhang zwischen Ein- und Ausgangsstrom besteht (Formel 2.4).

$$B \approx \frac{I_C}{I_B} \tag{2.4}$$

Das Übertragungskennlinienfeld des Feldeffekttransistors hat einen ähnlichen Verlauf wie das des Bipolartransistors. Sein Verlauf wird durch Formel 2.5 beschrieben.

$$I_D(U_{GS}) = \frac{K}{2}(U_{GS} - U_{th})^2 \tag{2.5}$$

K ist dabei der Steilheitskoeffizient und beschreibt die Steigung der Übertragungskennlinie. U_{th} ist die Spannung, bei der der Transistor leitend wird. In Abbildung 2.9 wird durch die gestrichelte Linie verdeutlicht, dass die Kennlinie, wie bereits aus Formel 2.5 ersichtlich wird, quadratisch ist.

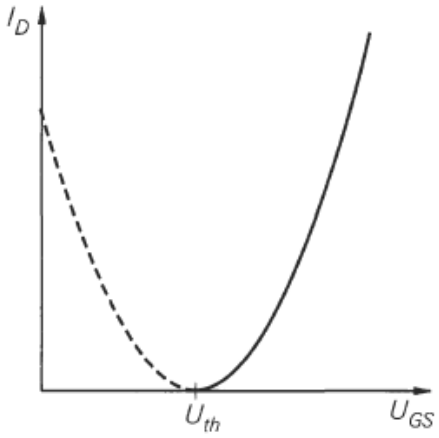

Abbildung 2.9.: Übertragungskennlinienfeld eines Feldeffekttransistors nach [9].

Um ein Eingangssignal (Wechselgröße) unverzerrt am Ausgang eines Systems zu erhalten, muss das Übertragungsverhalten des Systems linear sein. Die Verläufe der betrachteten Übertragungskennlinien bei Transistoren geben Auskunft darüber, ob und wie stark das Eingangssignal, welches in den Transistor gegeben wird, durch ihn verzerrt wird. Um Verzerrungen zu vermeiden, sollte der Arbeitspunkt möglichst an einer linearen Stelle der Kennlinie und die Aussteuerung möglichst klein sein. Der Arbeitspunkt wird dabei durch äußere Beschaltung des Transistors bzw. durch einen anliegenden Gleichanteil festgelegt. Er legt Ruheströme und Ruhespannungen fest. Bei Aussteuerung „in der Mitte" der Kennlinie wird vom A-Betrieb gesprochen. Abbildung 2.10 zeigt die Aussteuerung im A-Betrieb am Beispiel einer Eingitterröhre (Triode). Das Übertragungsverhalten beider Bauelemente ist annähernd identisch [13].

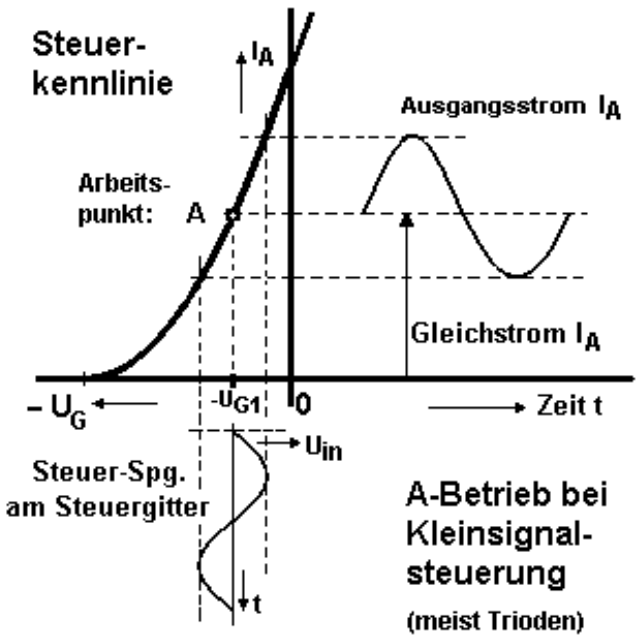

Abbildung 2.10.: Sinusaussteuerung und Ausgangssignal im A-Betrieb am, zum Transistor äquivalentem Beispiel einer Elektronenröhre [13].

Je linearer der Bereich um den Arbeitspunkt herum ist, desto geringer fallen Verzerrungen am Ausgang aus. Im Idealfall würde der Kurvenverlauf im Aussteuerbereich dem einer Tangente am Arbeitspunkt entsprechen (Abbildung 2.11). Es würden keine Verzerrungen auftreten. Praktisch ist es möglich, den Aussteuerbereich sehr klein zu wählen, sodass sich ein quasi-lineares Verhalten ergibt. Es wird von Kleinsignalverhalten gesprochen[9].
Im Vergleich der Übertragungskennlinienfelder der beiden Transistortypen spielt besonders der Unterschied in der mathematischen Beschreibung, in die Qualität eines Ausgangssignals hinein. Die Kennlinie eines Bipolartransistors wird durch Formel 2.3 beschrieben. Sie verläuft exponentiell. Die eines Feldeffekttransistors, Formel 2.5, verläuft quadratisch. Der exponentielle Verlauf knickt früh scharf ab und ist im Gegensatz zum quadratischen Verlauf weniger „rund". Die Abweichung zu einer linearen Geraden ist folglich geringer als ein quadratischer Verlauf. Zur einfachen Ansteuerung mit einem Wechselsignal eignet sich deshalb ein Bipolartransistor besser als ein Feldeffekttransistor.

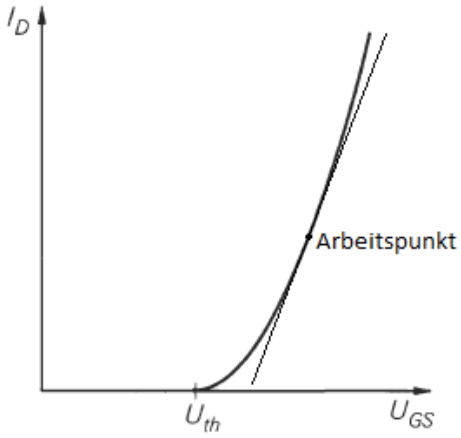

Abbildung 2.11.: Übertragungskennlinienfeld eines Feldeffekttransistors mit eingezeichneter Arbeitspunkt-Tangente, verändert nach [9].

Ansteuerung

Die Ansteuerung der beiden Transistortypen unterscheidet sich. Der Bipolartransistor wird durch den Basisstrom I_B angesteuert. Über der Basis-Emitter-Halbleiterschicht stellt sich durch ihn die Basis-Emitter-Spannung U_{BE} ein. Es zeigt sich ein Strom-Spannungsverhältnis, das dem einer Diode gleicht. Das Batteriesteuergerät ist üblicherweise recht gering. So hat der Transistor *BC547*, ein handelsüblicher npn-Kleinsignaltransistor, bei einem maximalen Kollektorstrom I_C von 100 mA, eine Stromverstärkung von etwa 100, also einen Basisstrom von 1 mA [14]. Ein Leistungstransistor des Typs *2N3055* mit einer geringen Stromverstärkung von 20 bis 70, benötigt für einen maximalen Kollektorstrom I_C allerdings bereits bis zu 750 mA Basisstrom. Für die Ansteuerverlustleistung gilt

$$P_B = U_{BE} \cdot I_B \, . \tag{2.6}$$

Der MOSFET wird durch die Steuerspannung U_{GS} angesteuert. Dabei fließt kein Strom in das Gate. Wird jedoch das dynamische Verhalten betrachtet (z.B Anregung mit Sinus oder Rechteck), so müssen parasitäre Kapazitäten zwischen den verschiedenen Bereichen des MOSFET berücksichtigt werden. Das Übertragungsverhalten des Transistors wird durch die Übertragungskennlinie beschrieben. Die Kanalkapazitäten, Überlappungskapazitäten und Sperrschichtkapazitäten wirken sich jedoch zusätzlich auf das Übertragungsverhalten aus. Abbildung 2.12 zeigt den Einfluss dieser Kapazitäten auf die Eingangs-Aussteuerspannung

U_{GS}. Sie führen, je nach Transistor und Aussteuerung, zu einer zusätzlichen Verzerrung des Eingangssignals.

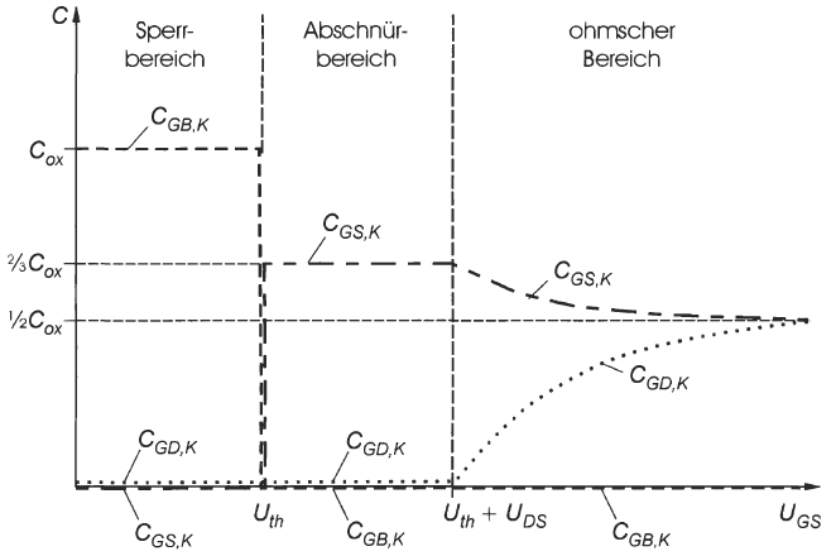

Abbildung 2.12.: Einfluss der Steuerspannung auf die parasitären Kapazitäten in einem MOSFET nach [9]. Diese verschlechtern das dynamische Verhalten und die Fähigkeit, Verzerrungen der Übertragung zu minimieren.

Idealisiert ergibt sich die Ansteuerleistung eines MOSFET zu

$$P_G = U_{GS} \cdot I_G \text{ mit } I_G = 0A \Rightarrow P_G = 0W \; . \tag{2.7}$$

Real muss aber berücksichtigt werden, dass die parasitären Kapazitätsgrößen bei Leistungstransistoren sogar bis in den Nanofarad-Bereich gehen. Bei Änderungen der Gate-Source-Spannung müssen diese Kapazitäten zunächst ge- bzw. entladen werden. Die Ansteuerung ist deshalb nicht leistungslos, da Lade- und Entladeströme aufzubringen sind. Um diese Effekte zu minimieren, ist es notwendig, solch einen Transistor sehr niederohmig anzusteuern.

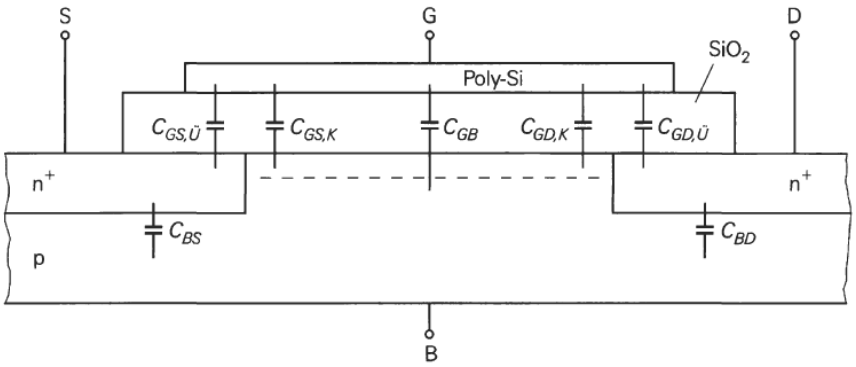

Abbildung 2.13.: Ursprung parasitärer Kapazitäten in MOSFETs nach [9].

Stromfluss

Nach der aufgestellten Anforderung in Kapitel 2.1.1 und Kapitel 2.1.2 soll die Schaltung in der Lage sein, die zu messende Batterie mit einem Gesamtstrom von 10 A Gleichanteil und weiteren 2 A Wechselstromamplitude zu entladen. Es soll also ein Transistor gefunden werden, der in der Lage ist, ein Eingangssignal am Ausgang mindestens 12 A linear zu treiben. Im Bereich der Leistungselektronik werden bevorzugt Feldeffekttransistoren verwendet [9]. Transistoren zu finden, die einen Drainstrom von 12 A treiben können, ist daher kein Problem. Ein Beispiel für einen solchen Transistor ist der *IRFR 024N Leistungs-MOSFET* von *International Rectifier* [15]. Im Elektronikfachhandel sind aber auch problemlos MOSFETs zu bekommen, die weit über 100A Drainstrom liefern. Zwar sind Hochleistung-Bipolartransistoren nicht so verbreitet wie entsprechende MOSFETs, aber auch hier lassen sich diverse Typen finden, die in der Lage sind 12A zu treiben. Ein Beispiel ist der Transistor 2N3055, der zum Beispiel von On-Semiconductor erhältlich ist [16]. Es ist zu erwähnen, dass die Stromverstärkung bei Bipolartransistoren, die ein Maß für das Verhältnis von Kollektor- zu Basisstrom ist, mit steigendem Kollektorstrom sinkt. Die Anforderung an den zu treibenden Strom kann also zunächst einmal von beiden Transistortypen erfüllt werden.

Gegenüberstellung und Auswahl

In Tabelle 2.1 werden die Vor- und Nachteile von Feldeffekt- und Bipolartransistor noch einmal unter dem Gesichtspunkt der Entwicklung der ersten Prototyp-Schaltung aufgelistet. Die Schaltung soll eine Anregung an einer zu messenden Batterie durchführen und die Eignung eines solchen Aufbaus bestätigen. Ein Feldeffekttransistor eignet sich nach Tabelle 2.1 aufgrund von Ansteuerung und Hauptstrom besser als Senke, als ein Bipolartransistor.

Ausschlaggebend für die Auswahl des Transistors ist allerdings seine Linearität. Es soll ein möglichst verzerrungsfreier Strom erzeugt werden. Unter MOSFETs lassen sich zwar problemlos Transistoren finden, die hohe Drainströme vertragen, allerdings ist der Aspekt der Linearität kritisch. Nach der Bewertung der Tabelle 2.1 hat der Bipolartransistor in keinem der angeführten Punkte ideale Eigenschaften, eignet sich aber in jedem Punkt, wenngleich mit Abstrichen. Aus diesem Grund wird für die Funktion als Stromsenke ein Bipolartransistor verwendet.

Tabelle 2.1.: Vergleich - Feldeffekttransistor zu Bipolartransistor

	Feldeffekttransistor	**Bipolartransistor**
Linearität	- Quadratische Kennlinie	+ Exponentielle Kennlinie
Ansteuerung	- Gate-Kapazität muss ge- und entladen werden.	+ Relativ geringer Basisstrom
Hauptstrom	++ Problemlos über 100A	+ Hoher Kollektorstrom auf Kosten von hohem Basisstrom

2.3.2. Grundschaltungen des Bipolartransistors

Nachdem die Auswahl des Transistortyps getroffen wurde, muss festgelegt werden, wie der Transistor verschaltet werden soll. Darauf aufbauend kann ein Transistor ausgewählt werden, der die daraus resultierenden spezifischen Anforderungen erfüllt.

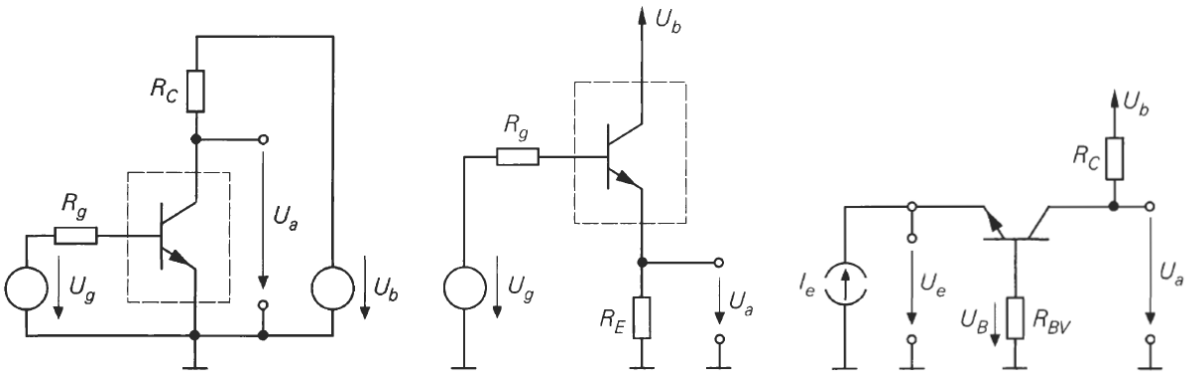

Abbildung 2.14.: Grundschaltungen des Transistors von links nach rechts: Emitter-, Kollektor- und Basisschaltung [9].

Basisschaltung

Bei der Basisschaltung wird die Ausgangsspannung am Kollektor des Transistor gegen das Bezugspotential abgegriffen, wobei der Eingang am Emitter liegt. Gemeinsames Bezugspotential ist das Basispotential, daher die Bezeichnung (siehe Bbbildung 2.14). Die Basisschaltung weist eine Stromverstärkung von 0,95 bis 0,999 und einen Ausgangswiderstand im Kiloohm-Bereich auf [17]. Für das Schaltungsprinzip nach Kapitel 2.2 würde dies bedeuten, dass der Ansteuerstrom größer als der Hauptstrom aus der Batterie sein müsste. Nach Anforderung wäre also ein Strom größer 12 A Ansteuerung nötig. Dies kann nur mit hohem Aufwand erreicht werden. Ausgehend von der Möglichkeit, eine Transistorverschaltung zu finden, die eine Stromverstärkung größer 1 hat, ist die Basisschaltung daher für die gewünschte Funktion ungeeignet. Sie wird typischerweise als Längsregler oder innerhalb von HF-Verstärkerschaltungen verwendet.

Emitterschaltung

Bei der Emitterschaltung wird die Ausgangsspannung am Kollektor gegen das gemeinsame Bezugspotential abgegriffen (siehe Abbildung 2.15). Daraus resultiert ein relativ hoher Ausgangswiderstand. Dabei fungiert die Basis als Eingang. Sie ist recht hochohmig. Gemeinsames Bezugspotential von Ein- und Ausgang ist der Emitter. Der Ausgangsstrom läuft durch den Kollektor und wird deshalb auch als Kollektorstrom I_C bezeichnet. In Abhängigkeit des Eingangsstroms ergibt sich mit der Stromverstärkung B ein Ausgangsstrom

$$I_C \approx B \cdot I_B \, . \tag{2.8}$$

Dabei liegt die Stromverstärkung, je nach Transistorparametern, bei etwa 10 bis 50. Sie ist auch von der Beschaltung abhängig. Die Ausgangsspannung der grundlegenden Beschaltung eines Transistors in Emitterschaltung ergibt sich zu

$$U_a = U_{Bat} + (I_a - I_C)R_C \overset{I_a=0}{=} U_{Bat} - R_C I_C \mid_{U_a=U_{CE}} \, . \tag{2.9}$$

Dabei ist zu beachten, dass sich die Ausgangsspannung mit steigendem Eingangsstrom verringert. Die Emitterschaltung invertiert also die Eingangsgröße bzw. weist eine Phasenverschiebung von 180 ° an der Ausgangsspannung, gegenüber der Eingangsgröße auf. Ein eingesetzter Basiswiderstand R_g begrenzt die maximale Aussteuerung. Er verringert den Basisstrom und somit den Ausgangsstrom, welcher den Spannungsabfall über einer Last

hervorruft. Ein Kollektorwiderstand R_C begrenzt den maximalen Kollektorstrom. Er verringert dadurch die Höhe der Ausgangsspannung. Die Emitterschaltung hat eine hohe Spannungsverstärkung und die oben genannten Widerstandseigenschaften. Sie wird deshalb als Spannungsverstärker oder im Schaltbetrieb verwendet [17] [9].

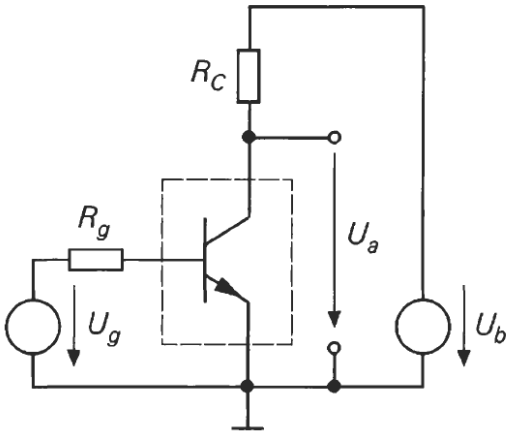

Abbildung 2.15.: Grundlegende Beschaltung eines npn-Bipolartransistors in Emitterschaltung nach [9].

Kollektorschaltung

Die Kollektorschaltung oder auch Emitterfolger ähnelt von der Beschaltung der Emitterschaltung. Im Unterschied zu ihr, wird das Ausgangssignal hier jedoch am Emitter abgegriffen. Die Folge ist ein kleiner Ausgangswiderstand. Auch hier dient die Basis als Steuereingang. Deshalb hat auch diese Schaltung einen hohen Eingangswiderstand. Sie weist eine Stromverstärkung von $(B + 1)$ auf. Ihr Ausgangsstrom ergibt sich somit zu

$$I_E = I_B \cdot (B + 1) \, . \tag{2.10}$$

Die Ausgangsspannung der grundlegenden Beschaltung eines Transistors in Kollektorschaltung ergibt sich zu

$$U_a = (I_C + I_B + I_a)R_E \overset{I_a=0}{=} (I_B + I_C)R_E \, . \tag{2.11}$$

Im Gegensatz zur Emitterschaltung, wird in der Kollektorschaltung das Ausgangssignal nicht invertiert. Die Ausgangsspannung folgt also der Eingangsspannung. Auch hier begrenzt der

Basiswiderstand R_g den Basisstrom und somit die Aussteuerung des Transistors. Der Emitterwiderstand ist der Arbeitswiderstand. Er hat neben der Aufteilung der Spannung, den Effekt der Reduzierung der Aussteuerung, da die Spannung an der Basis hier nicht nur über U_{BE}, sondern auch über R_E abfällt. Er hat zudem den zusätzlichen Effekt der Begrenzung des Emitter- und somit auch des Kollektorstroms.
Die Kollektorschaltung weist eine hohe Stromverstärkung bei einer Spannungsverstärkung kleiner 1 auf. Deshalb eignet sie sich als Stromverstärker. Aufgrund der genannten Widerstandseigenschaften wird sie auch als Impedanzwandler verwendet [17] [9].

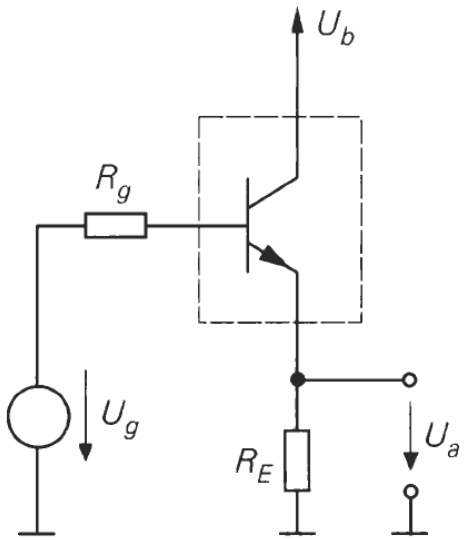

Abbildung 2.16.: Grundlegende Beschaltung eines npn-Bipolartransistors in Kollektorschaltung nach [9].

Gewählte Transistorgrundschaltung

Für die Anwendung eines Transistors nach Kapitel 2.2 kommt lediglich die Kollektorschaltung in Frage. Sie bietet alle notwendigen Eigenschaften. Eine hohe Stromverstärkung, einen hohen Eingangswiderstand bzw. einen geringen Steuerstrom und einen kleinen Ausgangswiderstand. Der Transistor erhitzt sich aufgrund der umgesetzten Leistung während des Betriebs. Dies führt zu einem Anstieg des Basisstroms welcher wiederum einen Anstieg des Kollektorstroms in Abhängigkeit der Temperatur zur Folge hat. Dies ist für eine lineare verzerrungsfreie Übertragung eines veränderlichen Steuerstroms fatal, denn die Temperatur des Transistors ist bei veränderlichen Eingangsgrößen nie konstant. Der Emitterwiderstand

in der Schaltung dient neben der notwendigen Aufteilung der Spannung auch zur Temperaturkompensation. Steigt der Emitterstrom, so steigt auch die Spannung über dem Emitterwiderstand. Dies führt zu einer Senkung der Basis-Emitter-Spannung, die wiederum den Basisstrom reduziert.

Die Schaltung soll in der Lage sein, mittels Software, den im Hauptstrang (Leitungen, durch die der Entladestrom fließt) fließenden Strom korrigieren zu können. Es wird also eine Rückführung in das steuernde System, in diesem Fall ein Mikrocontroller, benötigt. In diesem Zusammenhang wird für die Messung ein Messwiderstand (Shunt) benötigt. Der Spannungsabfall über diesen soll wiederum vom Mikrocontroller aufgenommen werden, um den Strom regeln zu können. Dieses wird mittels Mikrocontroller-eigenem AD-Umsetzer realisiert. Im Idealfall benötigt dieser keine Vorverarbeitung des Spannungsabfalls über dem Shunt. Das erfordert jedoch eine Messung gegen das Massepotential. Die Kollektorschaltung bietet in diesem Zusammenhang die Möglichkeit, den spannungsaufteilenden Emitterwiderstand als Shunt zu verwenden. Der Vorteil der Messung des Stroms über den Emitterwiderstand ist, dass dieser ohnehin vorhanden ist. Es muss also kein zusätzlicher Messwiderstand in den Hauptstrang geschaltet werden. Ein Nachteil kann allerdings sein, dass der Emitterwiderstand klein gewählt wird. Es soll ein maximaler Strom von 12 A durch ihn fließen, jedoch nicht zu viel Spannung über ihm abfallen, sodass der Transistor in Sättigung gehen würde (siehe Kapitel 2.3.3). Zudem erfordert ein kleinerer Shunt immer auch eine genauere Spannungsmessung für die Ermittlung eines Stroms.
Der Emitterwiderstand hat zusätzlich die Funktion der Stromgegenkopplung. Diese Regelung soll Veränderungen im Übertragungsverhalten aufgrund von Temperaturänderungen des Transistors (und des Emitterwiderstands) vermindern. Die Erhöhung der Temperatur des Transistors, aufgrund der an ihm umgesetzten Leistung, führt zu einem höheren Kollektorstrom bei gleicher Eingangsspannung Basis gegen Masse. Soll eine EIS-Messung durchgeführt werden, so ist eine Änderung des Ausgangsstroms bei gleicher Basis-Emitter-Spannung ungünstig, denn sie führt zu einer Verzerrung des Eingangssignals bei Änderung der Transistortemperatur. Die Gegenkopplung durch den Emitterwiderstand wirkt diesem Effekt entgegen. Der Emitterwiderstand hält die Aussteuerung stabil. Steigt der durch ihn fließende Strom, so steigt auch die über ihm abfallende Spannung. Das führt zu einer Verringerung der Basis-Emitter-Spannung sobald sich der Kollektorstrom erhöht. Die kleinere Basis-Emitter-Spannung reduziert wiederum die Aussteuerung und führt zu einem reduzierte. Kollektorstrom. Um eine gute Kompensation der Temperatur zu erhalten, sollte der Emitterwiderstand dazu etwa die Größe des Basis-Emitter-Widerstands r_{bE} haben [9].

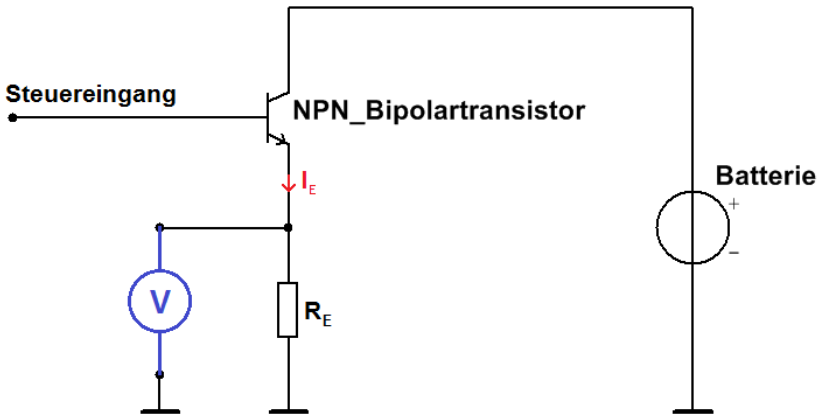

Abbildung 2.17.: Grundsätzliche Verschaltung eines npn-Bipolartransistors als modulierbare Last an einer Batterie. Dabei dient der Emitterwiderstand R_E als Shunt zur Messung des Stroms, welcher aus der Batterie fließt.

2.3.3. Anforderungen an den Transistor

Abbildung 2.17 zeigt die Grundschaltung, die zu Anregung verwendet werden soll. Zunächst wird versucht, die Schaltung so auszulegen, dass sie in der Lage ist, sowohl eine 12 V (Fahrzeug-) Batterie, als auch eine 3,3 V Zelle mit 500 mA bis 12 A zu entladen. Im Folgenden werden die Anforderungen an die einzelnen Attribute des Transistors näher betrachtet.

Auf die Spannungsfestigkeit bis zu einer maximalen Zellenspannung von 14,4 V Ladeschlussspannung wird nicht näher eingegangen. Leistungstransistoren weisen eine maximale Kollektor-Emitter-Spannung auf, die üblicherweise weit über 20 V liegt.

Kollektorstrom

Nachdem die Grundbeschaltung des Transistors festgelegt ist, werden Transistoren gesucht, welche die notwendigen Anforderungen erfüllen. Zu den Anforderungen gehören ein maximaler Kollektorstrom größer 12 A , sodass ein Strom von 12 A noch möglichst linear übertragen wird. Wäre der Transistor bei 12 A bereits in Sättigung, so wäre die Verzerrung bei der Übertragung eines Sinus mit Amplitude bei einem Gleichanteil von 12 A extrem groß. Abbildung 2.18 zeigt qualitativ wie sich eine Aussteuerung mit Sinus bis in die Sättigung eines Transistors im Kollektorstrom auswirkt.

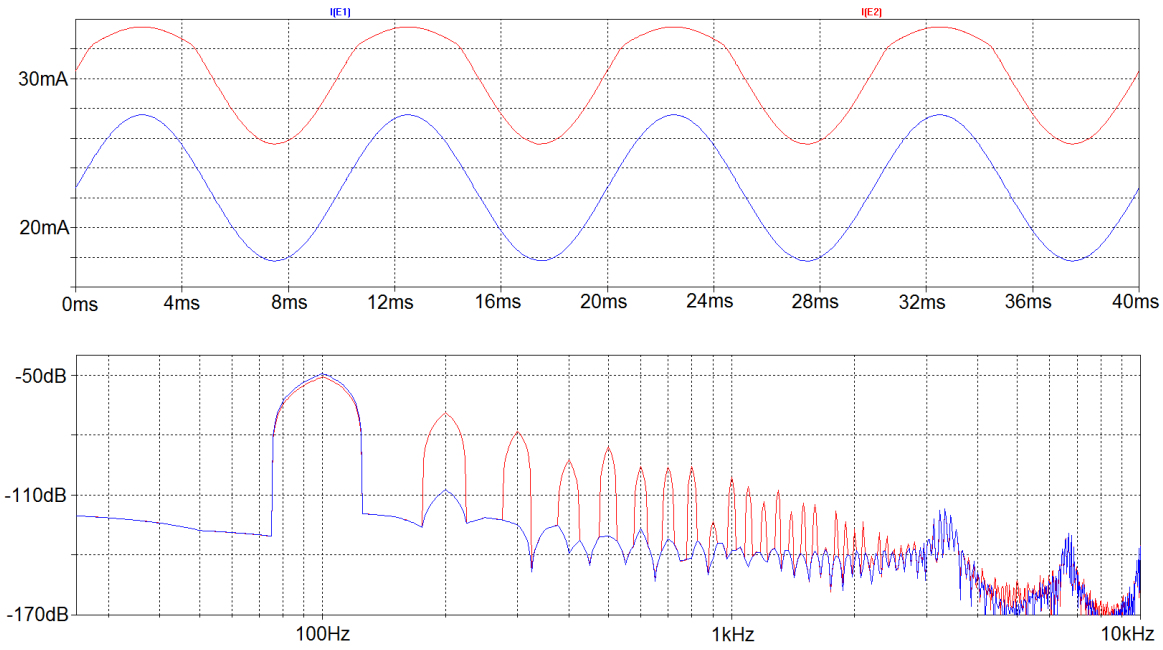

Abbildung 2.18.: Auswirkung der Erhöhung des Gleichanteils der Steuerspannung auf den Ausgangsstrom am Simulationsbeispiel BC546B in Kollektorschaltung: $R_E = 100\Omega$ mit 100Hz Sinusansteuerung. Dabei geht der Transistor bei Aussteuerung (rot) 2 bereits in Sättigung. Spannungsaussteuerung 1: 0,5 V mit 3 V Gleichanteil (blau); Spannungsaussteuerung 2: 0,5 V mit 3,8 V Gleichanteil (rot). Die Verzerrungen vom Aussteuerung 2 gegenüber Aussteuerung 1 sind mit dem bloßem Auge zu erkennen. Das Spektrum zeigt noch einmal den hohen Gehalt an Oberwellen in Aussteuerung 2 im Verhältnis zu Aussteuerung 1.

Basisstrom

Auf der Suche nach geeigneten Transistoren lässt sich feststellen, dass mit größerem Kollektorstrom I_C bei Transistoren die Stromverstärkung abnimmt. Der benötigte Basisstrom nimmt zu. Abbildung 2.3 zeigt die geplante Ansteuerung der Transistorbasis. Im Idealfall (Basisstrom ausreichend gering) lässt sich dieser direkt durch einen Analog-Digital-Umsetzer ansteuern.

Am Beispiel des Transistors *MJ15003* soll gezeigt werden, welches Problem bezüglich des Basisstroms typischerweise bei der Auswahl eines geeigneten Transistors auftritt. Es handelt

sich hierbei um einen Standard-Bipolar-Leistungstransistor mit einem maximalen Kollektorstrom von 20 A. Die Anforderung an den Kollektorstrom wird erfüllt. Bei einem Kollektrostrom von 5 A (Halbleitertemperatur 25 °C), liegt die Stromverstärkung (h_{FE}) zwischen 25 und 150 2.19. Sie weist hohe Varianz bei einem sehr geringen Minimum auf [18].

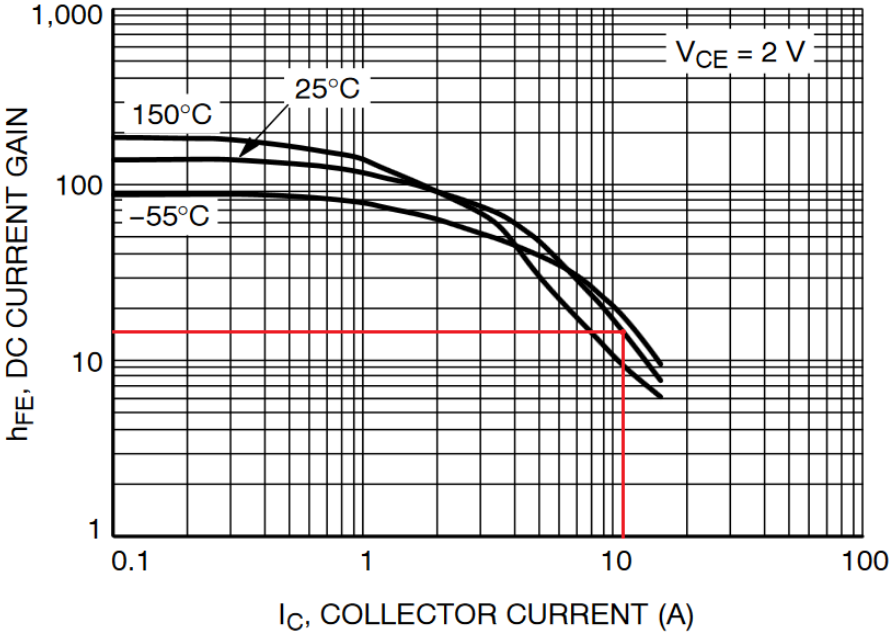

Abbildung 2.19.: Datenblattauszug: MJ15003 - Stromverstärkung in Abhängigkeit vom Kollektorstrom I_C. Bei etwa 12 A ergibt sich eine Verstärkung von 16.

Bei einer Verstärkung von 25 ergibt sich bei 12 A Kollektorstrom für den Basisstrom

$$I_B = \frac{I_C}{B} = \frac{12\,A}{25} = 480\,mA\,. \tag{2.12}$$

Das bedeutet, ein DA-Umsetzer, der zur Ansteuerung verwendet wird, muss diesen Strom am Ausgang aufbringen können. Üblicherweise sind DA-Umsetzer lediglich in der Lage, wenige Milliampere zu treiben. Dieser Wert für die Strom und auch eine geringe Stromverstärkung unter 100 sind typisch für Leistungstransistoren mit dieser Kollektorstromstärke. Dies führt zu dem Schluss, dass ein normaler Leistungstransistor ohne Vorschaltung einer Treiberstufe ungeeignet ist. Eine Möglichkeit, dieses Problem zu umgehen kann das Zwischenschalten einer Treiberstufe in Form eines Operationsverstärkers oder eines weiteren Transistors zwischen DA-Umsetzer und Leistungstransistor sein. Ein Operationsverstärker

für diesen Zweck müsste in der Lage sein, Ströme der berechneten Größenordnung zu treiben. Entsprechende Exemplare sind erhältlich. Problem des Einsatzes eines Operationsverstärkers ist jedoch die Versorgungsspannung. Es würde für diesen Fall eine Versorgungsspannung benötigt, die den benötigten Basisstrom und die zusätzliche Verlustleistung des Operationsverstärkers aufbringen kann. Das Messsystem ist zum aktuellen Zeitpunkt nicht in der Lage, einen solchen Strom zur Verfügung zu stellen. Weiterhin könnte der Operationsverstärker aus der zu messenden Batterie versorgt werden. Dies hätte zur Folge, dass neben dem Anregestrom auch der Versorgungsstrom aus der Batterie entnommen würde. Der entnommene Strom wäre bei einer Wechselanregung beliebiger Signalform inkonstant. Dies widerspricht der Grundidee der Anregeschaltung, deren Ziel es ist, den Entladestrom der Batterie möglichst genau zu bestimmen, ohne sie zusätzlich zur Messbelastung noch weiter zu belasten.

Der Einsatz eines weiteren Transistors würde zu einer Kaskadierung zweier Transistoren führen. Zwei kaskadierte Transistoren sind als Darlington-Transistor bekannt. Bei ihnen Multiplizieren sich die Stromverstärkungen der beiden kaskadierten Transistoren [9].

$$\text{Stromverstärkung } B \approx B_1 \cdot B_2 \tag{2.13}$$

Darlington-Transistoren weisen genau die bisher erläuterten benötigten Eigenschaften auf. Sie haben eine weit höhere Stromverstärkung, was zu einem weit geringerem Steuerstrom an der Basis führt. Abbildung 2.20 zeigt einen Darlington Transistor mit maxmimalem Kollektorstrom 30 A. Seine Stromverstärkung liegt trotz höherem maximalen Kollektorstroms bei etwa 9000 [19].

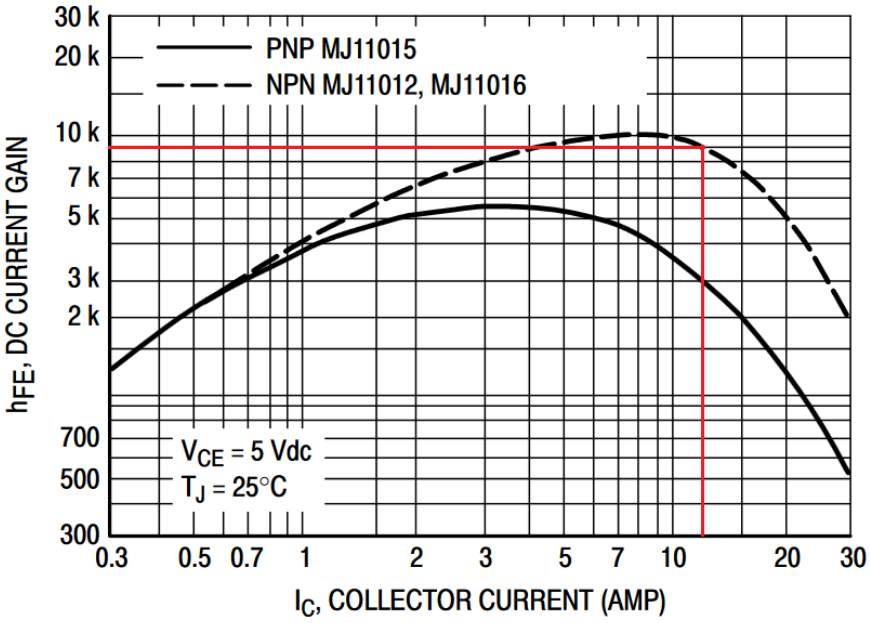

Abbildung 2.20.: Datenblattauszug: MJ11012[19] - Stromverstärkung in Abhängigkeit vom Kollektorstrom I_C. Bei etwa 12 A ergibt sich eine Verstärkung von 9000.

Aus diesem Grund wird ein npn-Darlington-Transistor für die Anregung verwendet. Ein solcher Transistor kann durch Kaskadierung von zwei Transistoren diskret aufgebaut werden, ist aber auch in Integrierter Ausführung erhältlich. Nachteil der Verwendung eines Darlington-Transistors ist, dass der Anstieg der Übertragungskennlinie, also ein Stromfluss durch den Hauptstrang, anstatt bei etwa 0,6 V bis 0,7 V Basis-Emitter-Spannung, erst bei etwa der doppelten Eingangsspannung eintritt. Auch der lineare Bereich beginnt folglich etwas später. Das bedeutet auch, dass der geringste einstellbare Strom im linearen Bereich entsprechend der Übertragungskennlinie steigt. Ob dies trotzdem noch den geringsten geforderten Entladestrom von 500 mA suaberer Wechselstromamplitude bei entsprechendem Gleichstromanteil erlaubt, ist noch zu klären.

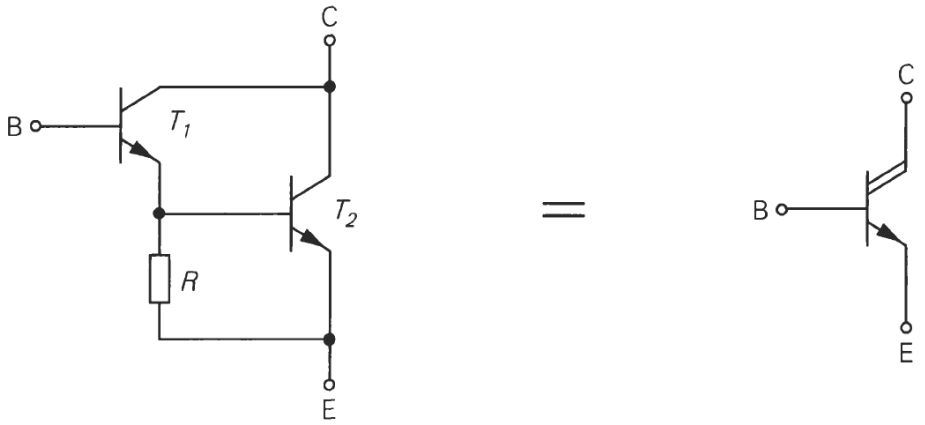

Abbildung 2.21.: Prinzip des Darlington-Transistors. Die Stromverstärkung ist das Produkt der Stromverstärkungen der beiden einzelnen Transistoren [9].

Kollektor-Emitter-Sättigungsspannung

Mit der Entscheidung einen Darlington-Transistor zu verwenden, ergibt sich ein weiteres Problem. Bei der Anregung einer 3,3 V Zelle wie der Zelle *ANR26650* von *A123 Systems* muss noch eine ausreichend große Spannung über der Kollektor-Emitter-Strecke abfallen, damit der Transistor voll durchgesteuert wird. Sinkt die Kollektor-Emitter-Spannung auf die Sättigungsspannung, d.h. befindet sich der Transistor in Sättigung, so kann er nicht mehr weiter durchgesteuert werden. Die Kollektor-Emitter-Spannung muss also immer über $U_{CE,Sat}$ liegen. Bei Darlington-Transistoren ist die Sättigungsspannung $U_{CE,Sat}$ generell höher als bei Einzeltransistoren, da die Spannung über zwei Kollektor-Emitter-Strecken abfällt.

Am Beispiel des Darlington-Transistors *MJ11012* von *On Semiconductor* soll das Problem verdeutlicht werden. Es handelt sich bei diesem npn-Darlington-Transistor um einen Leistungstransistor mit einem maximalen Kollektorstrom I_C von 30 A. Bei einem Kollektorstrom von 12 A liegt die Stromverstärkung bei etwa 9000 (Abbildung 2.20). Bei 20 A Kollektorstrom wird eine Sättigungsspannung von 3V angegeben. Diese Angabe gilt bei einer Stromverstärkung von 100. Steigt die Verstärkung, so steigt auch die Sättigungsspannung. Der Wert der Sättigungsspannung für eine Verstärkung von 9000 ist nicht aus dem Datenblatt zu entnehmen. Für dieses Rechenbeispiel sei dieser aber einmal als 3 V angenommen. Es sei die Batteriespannung 3,3 V. Die Tatsache, dass die Entladeschlussspannung geringer als die Nennspannung ist sei hier erwähnt, aber für das Rechenbeispiel vernachlässigt, da sie für die Veranschaulichung nicht relevant ist. Für die Dimensionierung des Emitterwiderstands ergibt sich mittels Maschenregel mit 12 A Kollektorstrom und unter Vernachlässigung des Basisstroms

$$U_{Bat} - U_{CE} + I_E \cdot R_E = 0 \tag{2.14}$$

mit $U_{CE} = U_{CE,Sat}$

$$R_E = \frac{U_{Bat} - U_{CE,Sat}}{I_E} = \frac{3,3\,V - 3\,V}{12\,A} = 25\,m\Omega\,. \tag{2.15}$$

Damit dieser Transistor bei Strömen kleiner 12 A nicht in Sättigung gerät, darf der Emitterwiderstand also höchstens 25 mΩ haben. Messungen haben im Vergleich ergeben: Eine 4 mm-Labormessleitung der Länge 1,6 m hat einen Widerstand von etwa 40 mΩ. Selbst dickere Zuleitungen würden einen starken Einfluss auf einen Widerstandswert dieser Größenordnung haben. Es ist somit für einen Laboraufbau praktisch ungünstig, einen so kleinen Widerstand zu verschalten.

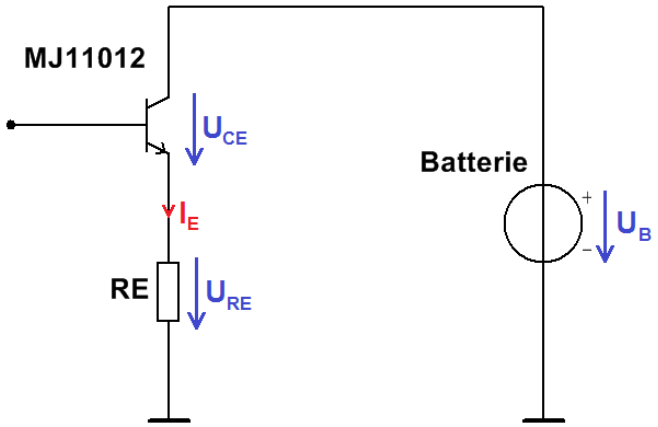

Abbildung 2.22.: Kollektorschaltung mit relevanten Größen zur Berechnung des Emitterwiderstands für die Auslegung von I_E 12 A, sodass der Transistor nicht in Sättigung geht.

Für die Lösung dieses Problems bietet es sich an, einen Transistor zu nutzen, dessen Sättigungsspannung bei großer Verstärkung geringer ist als bei dem Transistor aus dem Beispiel. Um die Sättigungsspannung genau bestimmen zu können, muss der Emitterwiderstand dimensioniert werden. Dabei wird angestrebt, den Emitterwiderstand so gering wie möglich zu

wählen. Dieser muss jedoch noch in der Lage sein, die an ihm erzeugte Wärme durch Verlustleistung abzuführen. Zudem sollte er trotzdem groß genug sein, um Leitungswiderstände im Aufbau vernachlässigbar zu machen. Es wird ein Wert von 470 mΩ gewählt. Dieser Wert ist nach wie vor deutlich unter einem Ohm, ist aber weit über dem Wert der genannten Leitungswiderstände. 470 mΩ liegen weit unter dem Wert des Basis-Emitter-Widerstands innerhalb des Transistors. Die Bedingung $R_E \approx r_{BE}$ zur idealen Kompensation der Transistortemperatur wird folglich nicht erfüllt. Unter diesem Umständen ist davon auszugehen, dass die Erwärmung des Transistors im Betrieb deshalb zu einer Veränderung des Kollektorstroms führt. Es muss davon ausgegangen werden, dass im Betrieb eine erhebliche Verlustleistung an diesem Widerstand umgesetzt wird. Dieser geringe Wert erfüllt die Bedingung des Emitterwidertandswerts für eine gute Temperaturkompensation nicht (siehe 2.1), da der Wert weit geringer ist als der Basis-Emitter-Widerstand der typischer Weise im Kiloohm-Bereich liegt [20].

$$P_v = \frac{U^2}{R} \rightarrow \quad R \downarrow \rightarrow \quad P_V \uparrow \tag{2.16}$$

Der Widerstand darf deshalb nicht zu klein gewählt werden, denn je kleiner der Widerstandswert bei der gleichen maximalen Verlustleistung, desto größer das Bauelement. Mit dem gewählten Wert 470 mΩ ergibt sich für die erlaubte Sättigungsspannung in Formel 2.14. den Transistor bei Weitem nicht erfüllt wird.

$$U_{CE}\,|_{U_{CE}=U_{CE,Sat}} = U_{Bat} - I_E \cdot R_E = 3,3\,V - 12\,A \cdot 470\,m\Omega = -2,34\,V \;\lightning \tag{2.17}$$

Der Wert ist geringer Null. Dies zeigt, dass es nicht möglich ist, alle Parameter, wie große Verstärkung, großer Kollektorstrom und geringe Sättigungsspannung in einem Darlington-Transistor zu vereinen. Insbesondere die Anforderung an die Sättigungsspannung ist niemals mit einer Zellenspannung von 3,3 V erfüllbar.

Änderung des Schaltungskonzepts

Die Anforderungen an den Transistor können mit der aktuellen Konzeption der Schaltung durch keinen Transistor erfüllt werden. Um die hohe Sättigungsspannung zu umgehen, wäre es theoretisch möglich, die Betriebsspannung U_{Bat} zu erhöhen, siehe Formel 2.15. Dies würde praktisch bedeuten, dass die Schaltung eine 3,3 V Batteriezelle nicht über den kompletten

Aussteuerbereich entladen könnte. Dies ist zwar für die Erfüllung der Zielsetzung nicht notwendig, wäre aber für die praktische Nutzung der Schaltung im Labor von Vorteil. Eine Reduzierung des, nach Zielsetzung geforderten Entladestroms wäre eine weitere Möglichkeit, die Anforderungen an einen Transistor zu reduzieren. Folge wäre, dass die EIS-Messung an der „größeren" Zelle im Labor, *ECC-LFPP* mit 45 Ah, nicht mit dem Entladestrom durchgeführt werden kann, mit dem sie vorgesehen ist (10 A Gleichanteil mit 2 A Sinusamplitude). Ein Parameter, der verändert werden kann, ist die Verstärkung. Eine Verringerung der Verstärkung wirkt sich auf den benötigten Basisstrom aus. Realisiert werden kann dies durch die Verwendung eines einfachen Transistors im Gegensatz zu einem Darlington-Transistor. Die Folge wäre ein hoher Basisstrom zur Ansteuerung, welcher eine Treiberstufe zwischen DA-Umsetzer und Transistor-Basis erfordern würde. Diese Option ist aufgrund der Versorgungsspannung für den Treiber, wie bereits in Kapitel 2.3.3 erläutert, ungeeignet.

Wenn die Zielsetzung nicht verändert werden soll, muss das Konzept der Schaltung noch einmal überdacht werden. Die zu messende Batterie soll mit einem hohen Gleichanteil und einem zusätzlichen Sinussignal durch den Transistor entladen werden. Dabei sind wichtige Aspekte Arbeitspunkteinstellung, Stromverstärkung und Spannungsabfall über der Kollektor-Emitter-Strecke des Transistors. Zur Lösung des Problems wird zunächst das Problem des Kollektorstroms betrachtet. Der maximale Kollektorstrom kann verdoppelt werden, indem zwei Transistoren parallel geschaltet werden. Es ist so möglich, über jeden der beiden Transistoren mit den maximalen Kollektorstrom zu entladen. Der maximale Gesamtentladestrom verdoppelt sich. Gleichzeitig reduziert sich dadurch zusätzlich die maximale abzuführende Leistung am Transistor sowie am Emitterwiderstand, da dieser zur Hälfte auf einen zweiten Strang auslagert wird. Dabei werden zwei möglichst gleich Übertragungskennlinien angestrebt. Das erlaubt simultane Ansteuerung beider Transistoren. Des Weiteren erlaubt es den Funktionsaustausch, sollte Gleichanteil und Wechselanteil jeweils auf einen der beiden Transistoren durchgeführt werden. Aus diesem Grund wird von der Nutzung zwei gleicher Transistoren ausgegangen, bei denen sich die Kennlinie nur noch durch Exemplarstreuung unterscheidet. Da sich die Kollektorströme addieren, können zwei Transistoren verwendet werden, bei denen der Fokus auf der Sättigungsspannung U_{CE} liegt. Um den Ansteuerstrom gering zu halten, werden dennoch Darlington-Transistoren gesucht. Es bleibt zu klären, mit welchem Signal die einzelnen Transistoren angesteuert werden, um den gewünschten Sinusstrom und den Gleichanteil zu erzeugen. Eine Möglichkeit ist die Ansteuerung beider Transistoren mit gleicher Spannung. Die Folge wäre eine Aufteilung der Eingangsspannung auf zwei Kennlinien. Durch diese Variante der Ansteuerung, lassen sich die Transistoren im linearen Bereich aussteuern. Der lineare Aussteuerbereich wird durch die zwei Transistoren vergrößert. Problem bei dieser Variante ist, dass die Transistoren bei der geforderten Messung möglicherweise im oberen Bereich der Kennlinie ausgesteuert werden. Der Abstand des Arbeitspunktes zum linearen Aussteuerbereich nimmt dabei zu und der Transistor geht letztendlich zunehmend in die Sättigung. Dies führt, genau wie bei Ansteuerung im unteren Bereich der Kennlinie, zu Verzerrungen des Ausgangssignals (Abbildung 2.18).

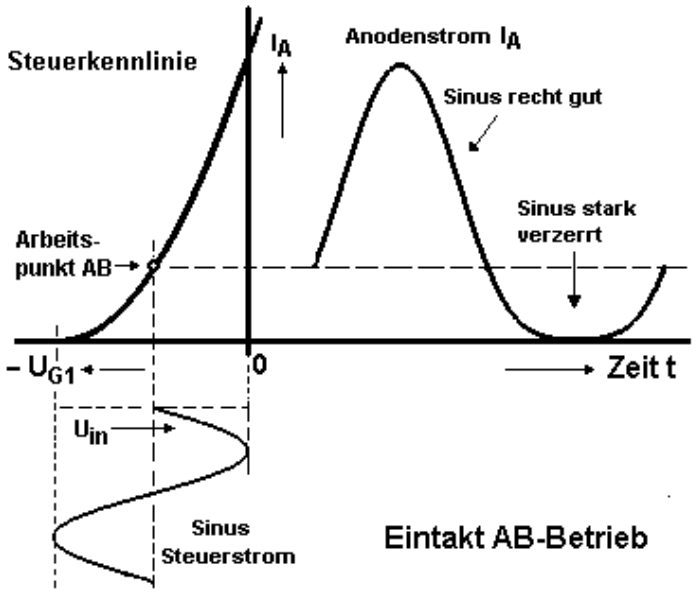

Abbildung 2.23.: Zu hohe Aussteuerung am Beispiel einer Triode bei einem Arbeitspunkt im AB-Betrieb verändert nach [13]. Deutlich sind auftretende Verzerrungen zu erkennen. Diese treten bei zu hoher Aussteuerung, aber auch bei zu weit nach oben oder nach unten verschobenem Arbeitspunkt (Gleichanteil) auf. Dies gilt es zu vermeiden.

Eine weitere Möglichkeit ist die Aussteuerung eines Transistors mit dem Großteil des Gleichanteils und die Aussteuerung des anderen mit dem Wechselanteil zuzüglich notwendigem Offset. Diese Variante der Ansteuerung eignet sich, wenn ein hoher Gleichanteil gewünscht, die Amplitude des Wechselsignals aber geringer ist. In diesem Fall wird nur der lineare Bereich des Wechselanteil-Transistors verwendet, während ein Großteil des Gleichanteils auf den zweiten Transistor verlagert wird. Bei dem Ansatz mit zwei Transistoren kann der Großteil des Gleichanteils mit einem Transistor erzeugt werden, während der zweite Transistor lediglich den Wechselanteil erzeugt, der für eine gute Arbeitspunkteinstellung der die Sinusaussteuerung benötigt wird. Es ist also möglich, einen hohen Gleichstrom erzeugen, der sich nicht auf die Qualität des Wechselsignals auswirkt und trotzdem einen guten Arbeitspunkt für den A-Betrieb zu wählen, der es ermöglicht, Verzerrungen gering zu halten. Werden die gleichen Transistoren simultan ausgesteuert, so besteht die Gefahr, dass die Amplitudenaussteuerung durch den hohen Gleichanteil bereits in den Sättigungsbereich der Transistoren läuft. Verzerrungen wären die Folge.

Für die gegebene Anforderung eines Gleichanteils von 10 A mit einer Wechselamplitude von

2 A eignet sich die zweite Ansteuerungsvariante besser. Wird zweite Variante gewählt, so ist auch ein zweiter Ansteuerstrang nötig. Dieser besteht aus einem weiteren DA-Umsetzer der ebenfalls vom Mikrocontroller gesteuert wird (siehe Abbildung 2.24). Dies ist durchaus realisierbar und wird deshalb umgesetzt.

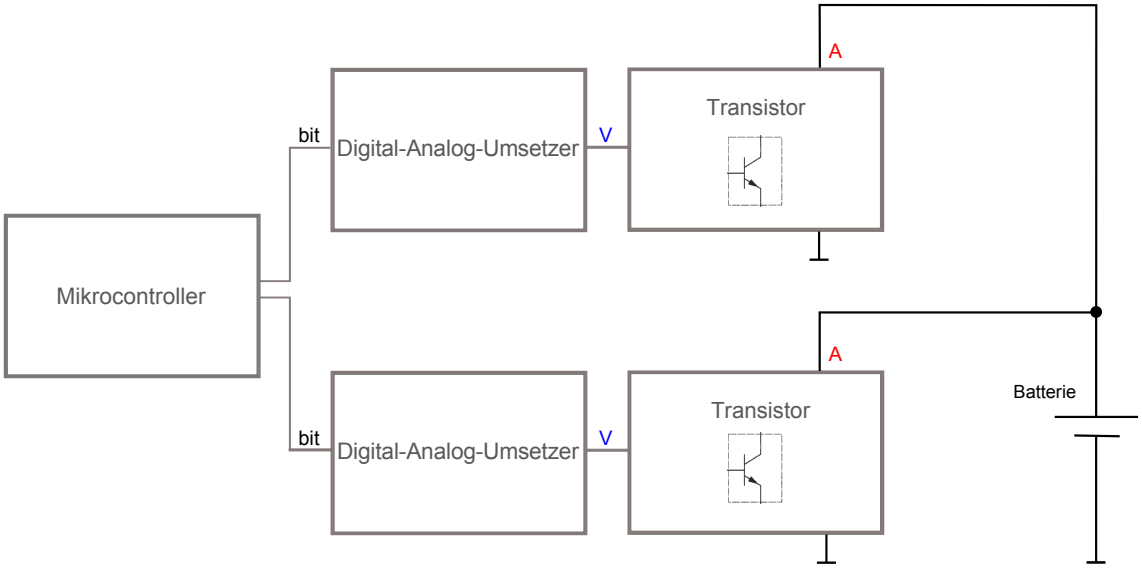

Abbildung 2.24.: Schaltungsprinzip mit zwei Transistoren und entsprechender Ansteuerung. Mit diesem Prinzip lässt sich eine Batterie mit zweichfachem Kollektorstrom entladen.

Trotz dieser Neukonzeption bleibt das Problem der Sättigungsspannung bei einer Batteriespannung von 3,3 V, unter Berücksichtigung der Entladeschlussspannung von etwa 2,5 V, bestehen [7] [8]. Die schlussendliche Zielsetzung bezieht sich auf Lithium-Fahrzeug-Batterien. Diese haben eine Nennspannung von 12 V. Davon ausgehend, wird das Schaltungskonzept deshalb von hier an auf eine 12 V Batterienennspannung ausgerichtet. So reduziert sich die Anwendungsflexibilität auf 12 V Batterien, sodass Einzelzellen mit 3,3 V nicht unabhängig voneinander angeregt werden können. Dennoch kann die Anregung mehrerer Zellen in Reihe genutzt werden, um im Zuge der EIS die Impedanz einzelner Zellen zu messen.

Frequenzverhalten

Transistoren neigen dazu, mit ansteigenden Ansteuerfrequenzen Verstärkung einzubüßen. Diese Eigenschaft von Verstärkern wird durch das Verstärkungs-Bandbreite-Produkt (Gain

Bandwidth Product, GBWP) beschrieben. Es beschreibt das Produkt von Verstärkung β und Frequenzbandbreite f (siehe Abbildung 2.25).

$$\text{GBWP} = f \cdot \beta(f) = f_{go} \cdot \beta_o = f_T \cdot 1 = \text{konstant} \tag{2.18}$$

Dabei liegt eine Grenzfrequenz f_{go} vor, ab der die Verstärkung mit 20 dB / Dekade sinkt. Abbildung 2.25 beschreibt dieses Verhalten. β_o ist die Grundverstärkung bei $f << f_{go}$ und f_t die Transitfrequenz, bei der die Verstärkung auf 1 sinkt [17].

Für die EIS Messung sollen Strom und Spannung bei verschiedenen Anregefrequenzen gemessen werden. Die Anforderung, die sich durch die bestehende Messung mittels Batteriesteuergerät ergeben, umfasst einen Frequenzbereich von 0,1 Hz bis 2 kHz [1]. Die maximale Frequenz von 2 kHz ist gering in Vergleich zu den maximalen Frequenzen, bei der die verwendeten Bauteile verwendet werden könnten. Die Frequenzanforderungen an die Transistoren für die Schaltung ist deshalb problemlos bei jedem in Betracht kommenden Transistor einhaltbar.

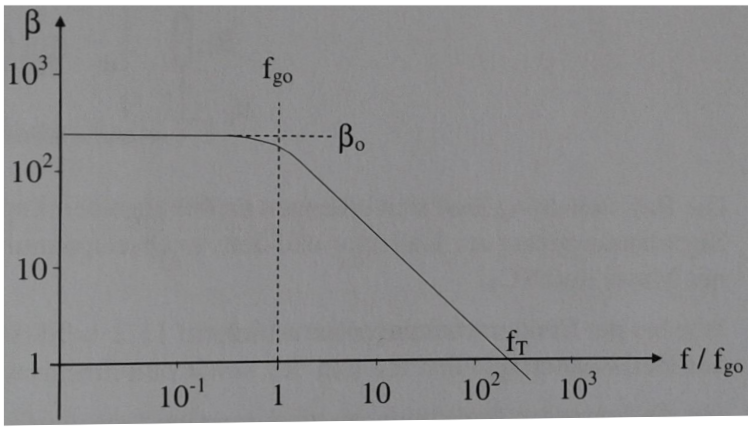

Abbildung 2.25.: Verstärkungs-Bandbreite-Produkt eines Transistors [17]. Ab einer Eingangsfrequenz, die etwa der Grenzfrequenz (bei f_{go} bereits -3 dB) entspricht, sinkt die Verstärkung von β_o und geht mit steigender Frequenz gegen 0.

Verlustleistung

Der Transistor muss in der Lage sein, einen Spitzenstrom von 12 A bei einer Spannung bis zur Ladeschlussspannung der Batterie abzuführen. Diese beträgt bei der verwendeten

Lithium-Batterie *ECC LFPP 45 Ah*, unter Berücksichtigung der Ladeschlussspannung von etwa 3,6 V pro Zelle etwa 4 · 3,6 V = 14,4 V [8]. Die abfallende Spannung über dem Emitterwiderstand wird bei dieser Abwägung vernachlässigt, da seine Dimensionierung noch zu klären ist. Es wird also zunächst einmal mit der Gesamtspannung gerechnet. Die abzuführende Gesamtverlustleistung der in der Schaltung umgesetzten Leistung beträgt

$$P_V = 14,4\,V \cdot 12\,A = 172,8\,W\ . \tag{2.19}$$

Dieser extrem hohe Wert ist zunächst mal ungenau und wird praktisch noch sinken. So fällt über dem Emitterwiderstand ein Teil der Leistung ab. der Wert veranlasst dennoch bereits dazu, davon auszugehen, dass mit großer Verlustwärme zu rechnen ist. Auf das Abführen dieser Wärme wird in Kapitel 2.4.6 noch einmal näher eingegangen.

2.3.4. Gewählter Transistor

Nach den Festlegungen der endgültigen Verschaltung werden zwei Transistoren eines Typs gesucht, der folgende Anforderungen erfüllt:

- Typ Bipolar npn-Darlington Transistor mit einer möglichst hohen Stromverstärkung
- Kollektorstrom größer 6 A
- Kollektor-Emitter-Sättigungsspannung so gering wie möglich
- Verstärkungs-Bandbreite-Produkt, welches keine Einbuße der Verstärkung bei 2 kHz hervorruft
- Möglichst hohe Leistungsstabilität bzw. maximale Sperrschichttemperatur

Ein Transistor der in Frage kommt ist der *2sd2390* der Firma *Sanken* [20]. Es handelt sich bei diesem Typen um einen Bipolar-npn-Darlington Transistor mit einer maximalen Stromverstärkung von etwa 15000 und einem maximalen Kollektorstrom von 10 A. Seine Sättigungsspannung beträgt maximal 2,5 V. Sein Verstärkungs-Bandbreite-Produkt beträgt 55 MHz und als maximale Verlustleistung werden 100 W und eine Sperrschichttemperatur von 150 °C angegeben. Der Transistor erfüllt alle Bedingungen und wird deshalb verwendet.

2.4. Auswahl der weiteren Bauteile

In Kapitel 2.3 wurde das Schaltungskonzept soweit angepasst, dass nun feststeht, welche weiteren Elemente für die Schaltung benötigt werden.

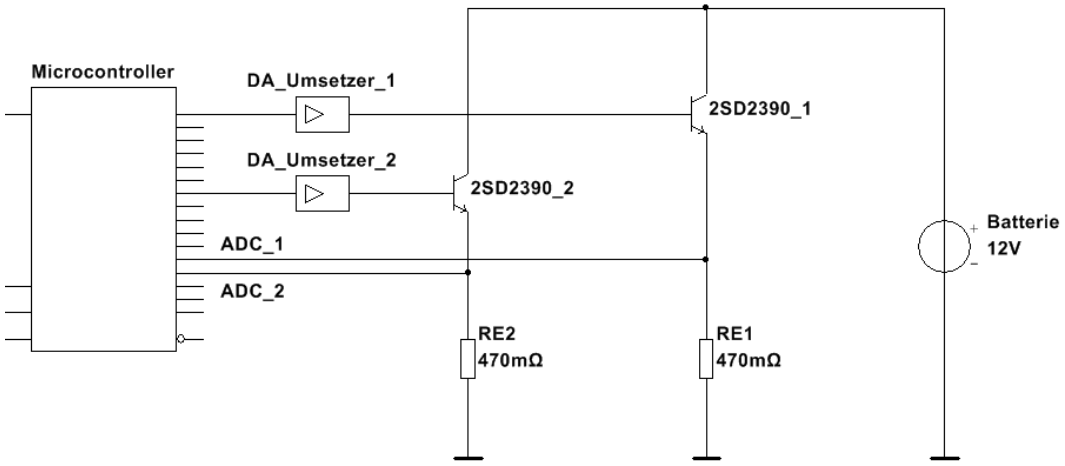

Abbildung 2.26.: Schaltungskonzept zur Bestimmung weiterer Bauteile für die Schaltung. Fehlende externe Peripherie sind die beiden DA-Umsetzer.

2.4.1. Mikrocontroller

In Kapitel 2.2 wurde bereits erwähnt und begründet, warum ein zweiter Mikrocontroller verwendet wird. Ausgehend vom Batteriesteuergerät, wird zur Softwareansteuerung der elektronischen Schaltung ein identischer Mikrocontroller verwendet. Es handelt sich dabei um einen, mit einem ARM Cortex M4 ausgestattetes LaunchPad von Texas Instruments, das EK-TM4C1294XL [21]. Es hat sich im Labor als zuverlässig erwiesen und das Team der Forschungsgruppe ist mit diesem Controller vertraut. Er verfügt über 3,3 V und 5 V Versorgungsspannung.

Die mit einem 120 MHz getaktete 32 Bit-CPU ist leistungsstark und wie sich im weiteren Verlauf noch zeigen wird, absolut ausreichend für die geschriebene Software.

Relevante Hardwarekomponenten des EK-TM4C1294XL LaunchPads sind Hardware Timer, zwei 12 Bit-AD-Umsetzer, Synchrone Serielle Schnittstellen, UART-Schnittstellen sowie USB-Schnittstellen [21].

2.4.2. Interne AD-Umsetzer

Bei den internen AD-Umsetzern des Mikrocontrollers handelt es sich um 12 Bit-Umsetzer mit einer Full-Scale-Spannung von 3,3 V. Die maximale Spannung darf über dem Emitterwiderstand, der als Shunt dient, 3,3 V nicht überschreiten. Bei voller Durchsteuerung des verwendeten Transistors ergibt sich der Maximalstrom von 10 A. Mit dem in Kapitel 2.3.3 bestimmten Emitterwiderstandswert von 470 mΩ ergibt sich über diesem eine maximale Spannung von

$$U_{RE} = I_E \cdot R_E = 10\,A \cdot 470\,m\Omega = 4,7\,V \tag{2.20}$$

Dies überschreitet das Spannungsmaximum der AD-Umsetzer. Es ist deshalb notwendig, die Spannung auf den Maximalwert von 3,3 V herunter zu teilen bzw. zu dämpfen. Der maximale Dämpfungsfaktor bzw. das maximale Teilerverhältnis ergibt sich mit R_2 als Widerstand gegen Masse zu

$$G = \frac{R_2}{R_1 + R_2} = \frac{3,3\,V}{4,7\,V} = 0.7021\,. \tag{2.21}$$

Für die einfache Realisierung einer Spannungsverstärkerschaltung eignen sich die Grundschaltungen des invertierenden- sowie die des nicht-invertierenden Verstärkers (siehe Abbildung 2.27). Es soll eine Verstärkung von etwa 0,7 eingestellt werden. Aus der Übertragungsfunktion der nicht-invertierenden Verstärkers

$$U_a = U_e \cdot \left(1 + \frac{R_N}{R_2}\right) \tag{2.22}$$

geht hervor, dass diese Schaltung nicht für Verstärkungen unter 1 verwendbar ist. Die Übertragungsfunktion der invertierenden Schaltung offenbart ein Problem, welches bereits aus dem Namen der Schaltung ersichtlich wird:

$$U_A = U_e \left(-\frac{R_N}{R_2}\right) \tag{2.23}$$

Die Schaltung invertiert das Eingangssignal, bedeutet also eine negative Ausgangsspannung gegen Masse [9]. Zu diesem Zeitpunkt der Planung liegen Versorgungsspannungen von 5 V, 3,3 V vom Mikrocontroller sowie nicht konstante 12 V der zu messenden Batterie für

die Versorgung zur Verfügung. Keine dieser Versorgungspotentiale ist negativ gegen Masse gerichtet.

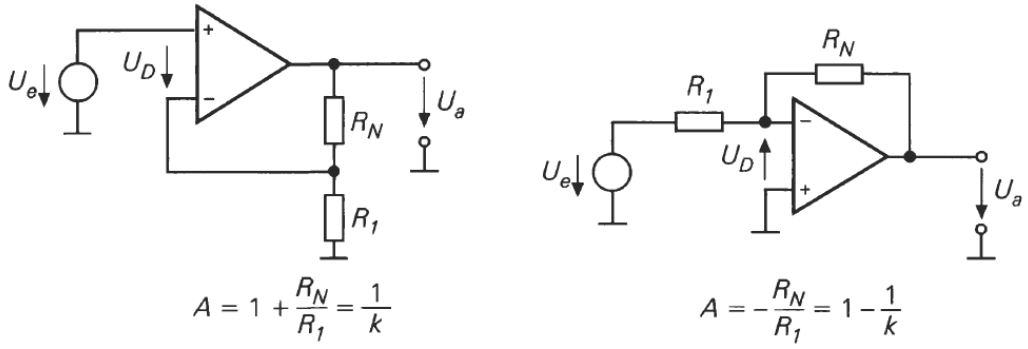

Abbildung 2.27.: Nichtinvertierende (links) und Invertierende Verstärkerschaltung (rechts) und deren Verstäkungsgleichungen [9].

Zudem müsste das negative Ausgangssignal noch einmal invertiert oder die Differenz ins Positive gerichtet werden. Alle Lösungen dieser Variante sind aufwändig. Eine Möglichkeit zur Reduzierung der AD-Umsetzer-Eingangsspannung ist ein hochohmiger Spannungsteiler parallel zu R_E. Unter Einbeziehung von vorhandenen Widerstandswerten und Widerstandstoleranzen ist der errechnete Faktor G = 0,7021 nach unten zu korrigieren. Dabei sollen die Werte für den parallelen Spannungsteiler einerseits sehr hoch im Verhältnis zum bestehenden Widerstand 470 mΩ sein, dürfen andererseits nicht zu groß gewählt werden. Hochohmige Widerstandsteiler begünstigen Störeinflüsse, die unerwünscht sind und die Qualität der Messung negativ beeinflussen. Diese Störeinflüsse würden dynamische Messungen verfälschen und könnten im schlimmsten Fall zu einer Überschreitung der Spannung über dem Messwiderstand und somit zur Beschädigung des AD-Umsetzer-Eingangs führen.
Als mittlerer Kompromisswert wird für den Messwiderstand $R_2 = 100\,k\Omega$ gewählt. Für den Widerstand R_1 und dem Teilerverhältnis aus 2.21 ergibt sich G = 0,7021.

$$G = 0,7021 = \frac{R_2}{R_1 + R_2} \tag{2.24}$$

$$G \cdot R_1 + G \cdot R_2 = R_2 \tag{2.25}$$

$$R_1 = R_2 \cdot \frac{(1-G)}{G} = 100k\Omega \cdot \frac{(1-0.7021)}{0.7021} = 42,43k\Omega \tag{2.26}$$

Es gilt mit diesem Ergebnis einen realisierbaren Widerstandswert zu verschalten, der tendenziell kleiner ist als das berechnete Ergebnis für R_1, sodass es bei einem geplanten Strom von 10 A nicht zu einer zu hohen Eingangsspannung am AD-Umsetzer führt. Des Weiteren muss die Widerstandstoleranz berücksichtigt werden. Bei dem ersten Testaufbau im Labor handelt es sich bei den, für den Spannungsteiler verschalteten Widerständen um Kohleschichtwiderstände der E12-Reihe mit einer Widerstandstoleranz von 5%. Im Prototypen-Aufbau sind für den Spannungsteiler SMD-Widerstände der E24-Reihe mit 1% Toleranz verschaltet.

Der nächste höhere Widerstand von $42,43k\Omega$ der E12-Reihe ist $47k\Omega$. Inklusive einer Toleranz von -5% auf R_1 und +5% auf R_2 (Worst Case) ergibt sich die zu messende Maximalspannung bei 10 A zu 3,3 V.

$$U_{2,max} = U_{RE} \cdot \frac{R_2}{R_1 + R_2} = 4,7\,V \cdot \frac{1,05 \cdot 100k\Omega}{1,05 \cdot 100k\Omega + 0,95 \cdot 47k\Omega} = 3,3\,V \qquad (2.27)$$

Dieser Wert entspricht dem Maximum.

Der nächste höhere Widerstand von $42,43k\Omega$ der E24-Reihe ist $43k\Omega$. Inklusive einer Toleranz von -1% auf R_1 und +1% auf R_2 (Worst Case) ergibt sich die zu messende Maximalspannung bei 10 A zu 3,06 V.

$$U_{2,max} = U_{RE} \cdot \frac{R_2}{R_1 + R_2} = 4,7\,V \cdot \frac{1,01 \cdot 100k\Omega}{0,99 \cdot 43k\Omega + 1,01 \cdot 100k\Omega} = 3,06\,V \qquad (2.28)$$

Dieser Wert überschreitet den Maximalwert für die Eingangsspannung des Umsetzers zwar nur minimal, zur Sicherheit wird aber dennoch ebenfalls der nächst höhere Widerstand 47 kΩ für R_1 verwendet. Dies geht auf Kosten der Auflösung des AD-Umsetzers. Unter Berücksichtigung des Worst Case der 1% Toleranzen der SMD-Widerstände ergibt sich somit eine maximale Messspannung von

$$U_{2,max} = U_{RE} \cdot \frac{R_2}{R_1 + R_2} = 4,7\,V \cdot \frac{1,01 \cdot 100k\Omega}{0,99 \cdot 47k\Omega + 1,01 \cdot 100k\Omega} = 3,22\,V\,. \qquad (2.29)$$

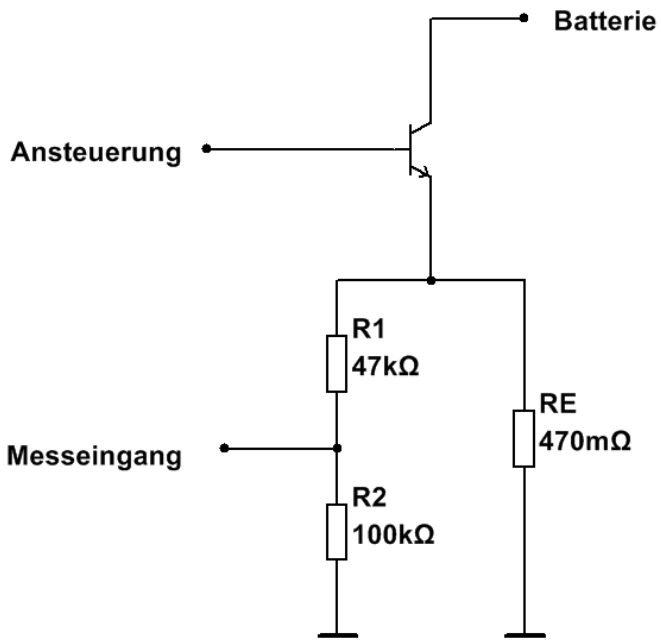

Abbildung 2.28.: Dimensionierter Spannungsteiler für die Strommessung mittels AD-Umsetzers.

2.4.3. Impedanzwandlung gegen Messbeeinflussung

Abbildung 2.28 zeigt den Aufbau des Spannungsteilers zur Strommessung. Dabei dient R_2 als Messwiderstand. Messungen der Spannung über R_2 mit diesem Aufbau zeigen starke Schwankungen der Spannungen bei verschiedenen Messdurchläufen. Dabei zeigt eine Messung des Emitterstroms keine Veränderung. Grund könnte ein sich ändernder Eingangswiderstand des AD-Umsetzers sein. Das würde bedeuten, Messungen am bestehenden Spannungsteiler würden durch den Eingangswiderstand des AD-Umsetzters zusätzlich belastet. Das Datenblatt gibt einen konstanten Eingangswiderstand von 2,5 kΩ an. Dies widerspricht der Annahme. Ausgehend von einem dynamischen Eingangswiderstand wird ein Impedanzwandler zwischen Messeingang und DAU-Eingang geschaltet (Abbildung 2.29). Erneute messungen zeigen, dass das Problem behoben ist. Die Änderung wird deshalb beibehalten.

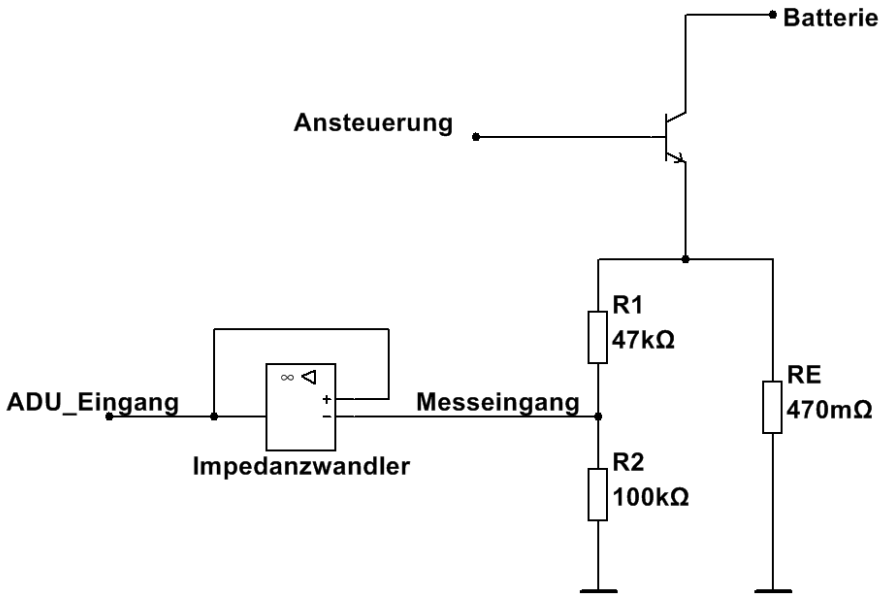

Abbildung 2.29.: Messspannungsteiler für die Strommessung mittels AD-Umsetzer. Dazwischen gepuffert durch einen Impedanzwandler.

2.4.4. DA-Umsetzer

Für die Ausgabe der Steuerspannung wird für jeden der zwei Kanäle ein DA-Umsetzer benötigt. Dieser muss keine nennenswerten Funktionen neben den üblichen Funktionen eines Wandlers haben. Eine hohe Auflösung ist von Vorteil. Die Mindestauflösung sollte der Auflösung der AD-Umsetzer des Controllers entsprechen, da diese die gemessene Ist-Größe des Stromes zurückführen, nach der der Strom am Ausgang eingestellt werden soll.

Verwendet werden kostengünstige 16 Bit-Umsetzer *AD5061* von *Analog Devices* [22] die mittels Seriellem Datenbus SPI angesprochen werden können. Sie können mit einer Betriebsspannung von 2,5 V bis 5,5 V betrieben werden. Diese können vom Controller geliefert werden. Auf die Ansteuerung des Umsetzers wird in Kapitel 3 näher eingegangen. Der Digital-Analog-Umsetzer braucht auch eine Referenzspannung, die den maximalen Aussteuergrad und damit die Stufenhöhe pro Bit bestimmt. Zwar könnten auch hier die 5 V des Mikrocontrollers verwendet werden, allerdings sollte die Referenzspannung so stabil wie möglich sein. Für die Messung im Laboraufbau wird zunächst eine externe Versorgung aus einem Labornetzteil verwendet. Für den Prototypen-Aufbau wird ein Spannungsregler (LDO) mit 5 V Ausgangsspannung an den Referenzpin geschaltet. Hierzu wird ein Regler des Typs *ADR445ARZ* von *Anlalog Devices* [23] verwendet. Sein Eingang liegt an einem weiteren Regler mit 8 V Ausgangsspannung, siehe Kapitel 2.4.7.

2.4.5. Berechnung der Verlustleistung

Wie bereits in Kapitel 2.3.3 erwähnt, muss von einer hohen Verlustleistung ausgegangen werden. An dieser Stelle wird die genaue Verlustleistung berechnet und geklärt, inwieweit eine zusätzliche Kühlung der Transistoren notwendig ist.

Die maximale Verlustleistung kann mit einem Emitterwiderstand von 470 mΩ (bestimmt in Kapitel 2.3.3) berechnet werden. Der Transistor ist in der Lage, bis zu 10 A Kollektorstrom zu treiben. Sollte es einmal zu einer Aussteuerung über den geforderten 6 A kommen, muss die Schaltung trotzdem in der Lage sein, die entstehende Wärme abzuführen, um Beschädigungen auszuschließen. Ist dies der Fall, so kann ggf. auch in Betracht gezogen werden, eine Aussteuerung über die geforderten 6 A zu nutzen.

Für die maximale Leistung am Emitterwiderstand ergibt sich mit den maximal möglichen 10 A pro Kanal eine umgesetzte Leistung von 47 W.

$$P_E = I_E^2 \cdot R_E = (10\,A)^2 \cdot 470\,m\Omega = 47\,W \tag{2.30}$$

Bei 470 mΩ wird folglich ein großer Leistungswiderstand benötigt.

Für die maximale Leistung am Transistor P_T ergibt sich unter Verwendung von Formel 2.14 und unter Berücksichtigung einer Ladeschlussspannung einer vollen Batterie von etwa $4 \cdot 3,6\,V = 14,4$ V der folgende Wert für P_T:

$$P_T = (U_{Bat} - U_{RE}) \cdot I_C = (U_{Bat} - R_E \cdot I_E) \cdot I_C = (14,4\,V - 470m\Omega \cdot 10\,A) \cdot 10\,A = 97\,W. \tag{2.31}$$

Ein Auszug aus dem Datenblatt des Transistors, Abildung 2.30, zeigt die safe-operating-area, in der festgehalten wird, wie sehr der Transistor bei 25 °C belastet werden darf. Wird der Transistor mit dem vorgesehenem Strom von 6 A Gleichstrom belastet, so ist keine weitere Kühlung vonnöten. Wird er aber mit dem maximalen angegebenen Strom von 10 A oder womöglich mit noch größerem Strom übersteuert, so wird ein Kühlkörper benötigt.

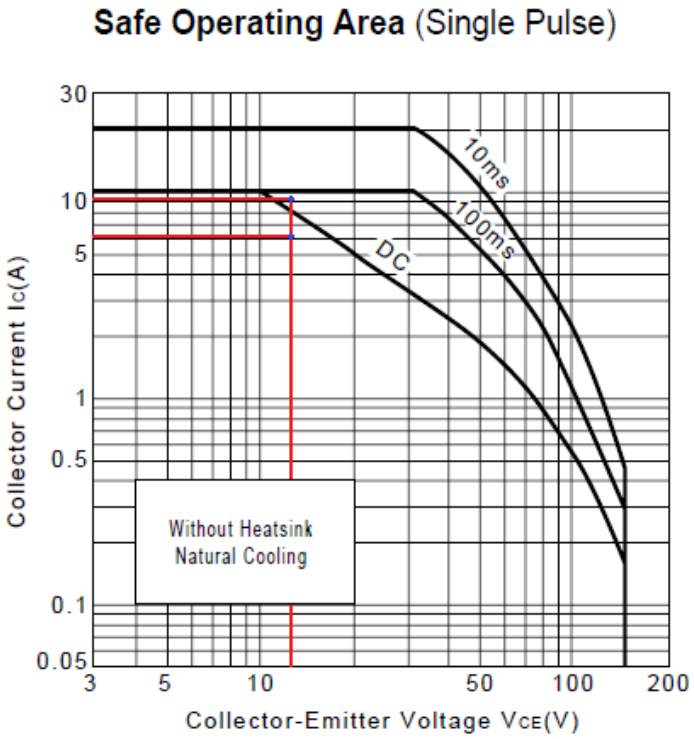

Abbildung 2.30.: Safe-operating-area des Transistors 2SD2390 nach [20]. Der Leistungspunkt 6 A, 14,4 V Gleichstrombelastung liegt innerhalb der SOA. Der Leistungspunkt 10 A, 14,4 V liegt außerhalb der SOA. Für den Fall solcher Belastungen muss der Transistor gekühlt werden.

2.4.6. Kühlung

Für eine effektive Kühlung wird die Berechnung der benötigten Kühlleistung für den maximal im Datenblatt angegebenen Strom 10 A bei vier voll geladenen 3,3 V Zellen mit Entladeschlussspannung 3,6 V durchgeführt. Der Emitterwiderstand hat einen Wert von 470 mΩ.

$$RE||(R_1+R_2) = \frac{R_E \cdot (R_1+R_2)}{R_E+R_1+R_2} = \frac{470\,m\Omega \cdot (47\,k\Omega + 100\,k\Omega)}{470\,m\Omega + 47\,k\Omega + 100\,k\Omega} = 469,9985\,m\Omega \approx 470\,m\Omega \tag{2.32}$$

Bei einem Emitterstrom von 10 A fällt ein Teil der umgesetzten Verlustleistung an ihm ab.

$$P_{RE} = I_E^2 \cdot R_E = (10\,A)^2 \cdot 470\,m\Omega = 47\,W \tag{2.33}$$

der Transistor muss in der Lage sein, die restliche Verlustleistungswärme abzuführen. Über die Restspannung, die über der Kollektor-Emitter-Strecke abfällt, ergibt sich diese Leistung zu

$$P_T = U_{CE} \cdot I_C = (U_{Bat} - R_E \cdot I_E) \cdot I_C = (14,4\,V - 470\,m\Omega \cdot 10\,A) \cdot 10\,A = 97\,W \tag{2.34}$$

Es wird von einer Umgebungstemperatur T_A von 25 °C ausgegangen. Aus dem Datenblatt des Transistors lässt sich entnehmen, dass die Sperrschichttemperatur T_j 150 °C nicht überschritten werden darf. Aus diesen Werten lässt sich nun ermitteln, welche Kühlleistung ein Kühlkörper haben muss, um die vorliegende Wärme soweit abzuführen, dass die maximale Sperrschichttemperatur nicht überschritten wird.

$$\text{Temperaturdifferenz } \Delta T = T_j - T_A = 150\,^\circ C - 25\,^\circ C = 125\,K \tag{2.35}$$

$$\text{Thermischer Widerstand des Kühlkörpers } R_{th} = \frac{\Delta T}{P_T} = \frac{125\,K}{97\,W} = 1,32\frac{K}{W} \tag{2.36}$$

Der thermische Widerstand des Kühlkörpers für jeden der beiden Transistoren darf nach Formel 2.36 jeweils höchstens $1,405\frac{K}{W}$ haben. Für den Fall von höheren Umgebungstemperaturen und anderen vernachlässigten Effekten sollte ein entsprechender Kühlkörper immer etwas überdimensioniert werden. Für die Kühlung des Prototypen-Aufbaus wird ein Prozessorkühler mit 100 W Kühlleistung, Abbildung 2.31 verwendet. Dieser ist also in der Lage, die Wärme bei 100 W Verlustleistung zu kühlen. Der Kühler ist stark überdimensioniert. Dies führt zu weniger Temperaturänderung beim Transistorbetrieb und sorgt so für bessere Ausgangsstromstabilität.

Abbildung 2.31.: Der Lüfter für den Prototypenaufbau [24].

2.4.7. Spannungsversorgung

Da neben dem Mikrocontroller auch die Batterie als oberste Versorgung verwendet wird, wird die Batterie für diese erste Version einer Anregung als stabile, harte Spannungsversorgung angenommen, bei der Spannungseinbrüche aufgrund von verstärkter Belastung nicht auftreten. Zur Versorgung der Bauteile nach Abbildung 2.36 wird ein Spannungsregler eingesetzt, der 10 V am Ausgang zur Verfügung stellen soll. Verwendet werden dazu Regler des Typs *LM1777* von *Texas Instruments* [25]. Er wird mit zwei Widerständen beschaltet, um die gewünschte Ausgangsspannung zu erhalten. Im Folgenden werden diese dimensioniert: Für die Ausgangsspannung gilt Formel 2.37.

$$V_a = V_{REF}\left(1 + \frac{R_2}{R_1}\right) + I_{ADJ}R_2 \tag{2.37}$$

Der Strom I_{ADJ} beträgt 60 μA und kann daher vernachlässigt werden. V_{REF} beträgt 1,25 V. Die Ausgangsspannung soll 10 V betragen. Der Strom durch R_1 soll etwa 2 mA betragen. So ergibt sich R_1:

$$R_1 = \frac{1,25\,V}{2mA} = 625\,\Omega \tag{2.38}$$

Mit der angestrebten Ausgangsspannung lässt sich R_2 berechnen.

$$U_a = U_{REF} \cdot \left(1 + \frac{R_2}{R_1}\right) + I_{ADJ} \cdot R_2 \tag{2.39}$$

$$R_2 = \frac{U_a - U_{REF}}{\left(\frac{U_{REF}}{R_1} + I\right)} = \frac{10\,V - 1,25\,V}{\frac{1,2\,V}{620\,\Omega} + 60\,\mu A} = 4,2\,k\Omega \tag{2.40}$$

Aus den Berechnungen folgend wird $R_1 = 620\,\Omega$ und für $R_2 = 4,3\,k\Omega$ eingesetzt. Ein Bild der Verschaltung zeigt Abbildung 2.32

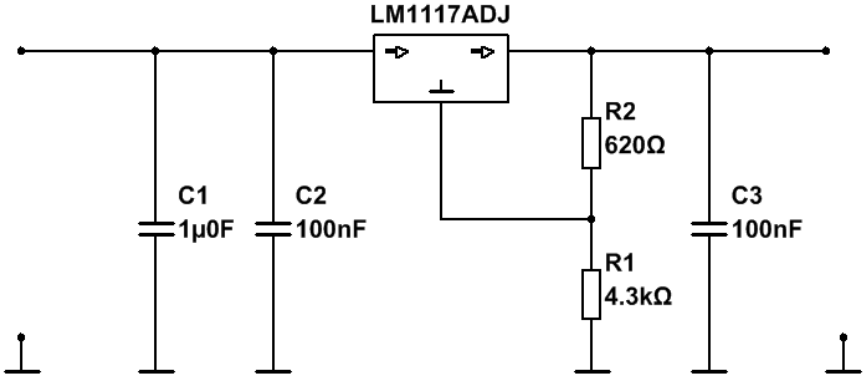

Abbildung 2.32.: Schaltung des verwendeten Reglers LM1117ADJ für eine Ausgangsspannung von 10 V.

2.5. Zusammenfassung

Zur Verifikation der Schaltungsfunktionalität wird ein Laboraufbau in Betrieb genommen. Nachdem an diesem die Auswahl und Dimensionierung der Bauteile sowie die Softwarefunktionalität bestätigt werden kann, wird eine erste Version einer Gesamtschaltung aufgebaut, die in der Lage ist, möglichst erste Messungen durchzuführen, aber besonders das Ziel hat, Schlüsse für die Weiterentwicklung des Anregesystems zu ziehen.

2.5.1. Laboraufbau

Der Laboraufbau ist eine provisorische Form des Gesamtschaltungsaufbaus. Er verfügt nicht über Spannungsregler und wird stattdessen von Laborquellen versorgt. So werden die Lüfter sowie die beiden verwendeten Verstärker (siehe Kapitel 3.2.1) mit externen 12 V versorgt. Die verwendeten Impedanzwandler werden vom Mikrocontroller versorgt. Der Aufbau ist mit Labormesskabeln verschaltet. Diese sind störanfällig, da sie Antenneneigenschaften und messbare Widerstände im zweistelligen Milliohm-Bereich aufweisen. Zudem können Kabelverbindungen mit Lamellensteckern dieser Art Kapazitäten erzeugen. Es kommt hinzu, dass mechanischer Einfluss auf den Aufbau Änderungen in der Signalübertragung hervorrufen kann. Für eine genaue Messung ist ein solcher Aufbau deshalb ungeeignet. Der Aufbau bestätigt aber dennoch, dass das Konzept in der Umsetzung funktioniert. Die Abbildungen in 2.33 zeigen die Elemente des Laboraufbaus.

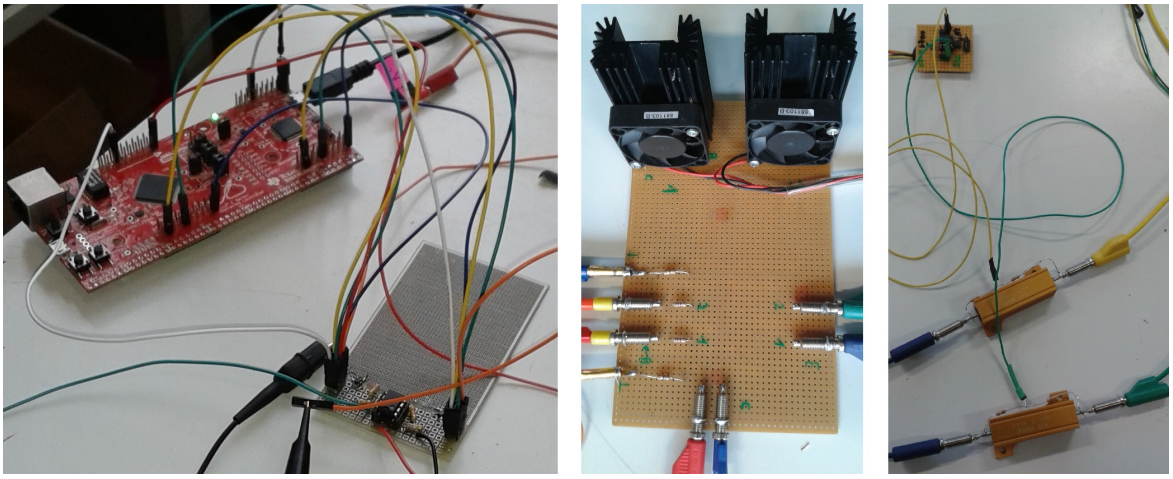

Abbildung 2.33.: Elemente des Laboraufbaus: Links Mikrocontroller und DA-Umsetzer sowie Verstärker. In der Mitte die Transistoren mit Kühlung und Vorspannungsteiler. Rechts Leistungswiderstände und Impedanzwandler.

2.5.2. Aufbau des Prototyps

Die fertige Gesamtschaltung des Prototypen-Aufbaus ist in Abbildung 2.36 abgebildet. Im Unterschied zum Laboraufbau enthält der Prototyp-Aufbau Spannungsregler und Blockkondensatoren zu Glättung eventueller Störungen. Eine, in Kapitel *DA-Umsetzer* erläuterte Erweiterung sind die Operationsverstärker zur Ansteuerung der Transistoren und die Spannungsregler, die diese versorgen. Damit möglichst störungsfreie Messungen durchgeführt

werden können, soll die Menge an Kabeln reduziert werden. Zudem soll der Aufbau mechanisch möglichst störunanfällig sein. Deshalb wird ein Teil der Schaltung in Form einer Platine gefertigt.

Auf einer Platine befinden sich die notwendigen Bauteile für die Ansteuerung und die Rückführung eines Transistors. Für einen Aufbau mit zwei Transistoren werden deshalb zwei Platinen benötigt. Unabhängig von der Platine liegen die beiden Transistoren sowie die Emitterwiderstände, die in der Lage sein müssen, 47 W abzuführen. Verwendet werden dafür 50 W, 470 mΩ Widerstände, die zur Sicherheit zusätzlich auf einen Kühlkörper montiert werden.

Abbildung 2.34.: 1 Ω und 500 mΩ Varianten der verwendeten Leistungswiderstände auf dem verwendeten Kühlkörper.

Als Aktivlüfter für die Transistoren wird jeweils ein 100 W CPU-Lüfter von Arctic verwendet. Die Überdimensionierung der Kühlung führt zu geringerer Erwärmung der Transistoren, was wiederum mehr Stabilität der Übertragungskennlinie zur Folge hat. Durch die Stromgegenkopplung der Emitterschaltung wird der Einfluss der Temperatur reduziert und durch die Kühlung wird die Temperaturänderung selbst verringert. Ein Teil der Schaltung wird auf einer Platine realisiert. Dadurch wird Leitungsführung eingespart und der Aufbau wird kompakter. Auf der Platine befinden sich Operationsverstärker, Spannungsregler und DA-Umsetzer. Die Platine ist so ausgelegt, dass sie zusammen mit einem Transistor und einem Emitterwiderstand einen Entladestrang bildet. Abbildung 2.35 zeigt das Design dem Programm EAGLE und eine fertig bestückte Platine. Ein Transistor wird direkt in die Platine gelötet. Dadurch ist ein stabiler mechanischer Aufbau möglich. Der Entladestrom fließt über die breiten Leiterbahnen der Platine, die durch Silberdraht und Lötzinn verstärkt werden, um die nötige Lei-

terdicke für den Entladestrom zu gewährleisten. Die Realisierung der Platine in Form eines Stranges ermöglicht die Kaskadierung von Entladepfaden. Die Umsetzung mit zwei parallelen Transistoren kann auch durch weitere parallele Transistoren erweitert werden. Mehr dazu in Kapitel 5.2.3. Für die Umsetzung der geplanten Schaltung werden zwei Platinen, zwei Emitterwiderstände und zwei Transistoren zu einer Schaltung zusammengefasst.

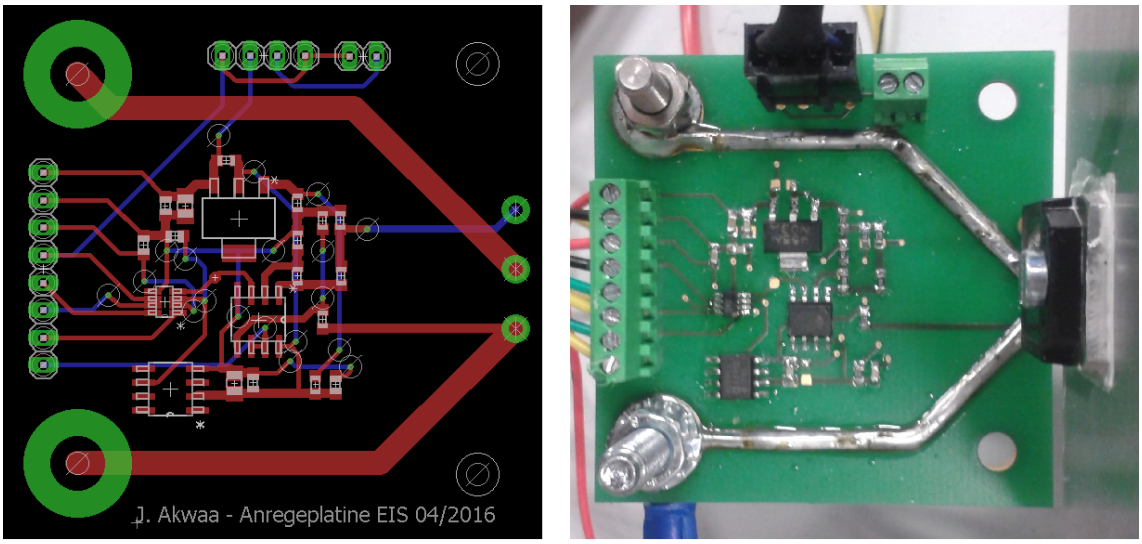

Abbildung 2.35.: Design der Anregeplatine im Design-Programm EAGLE und die bestückte Platine im Betrieb.

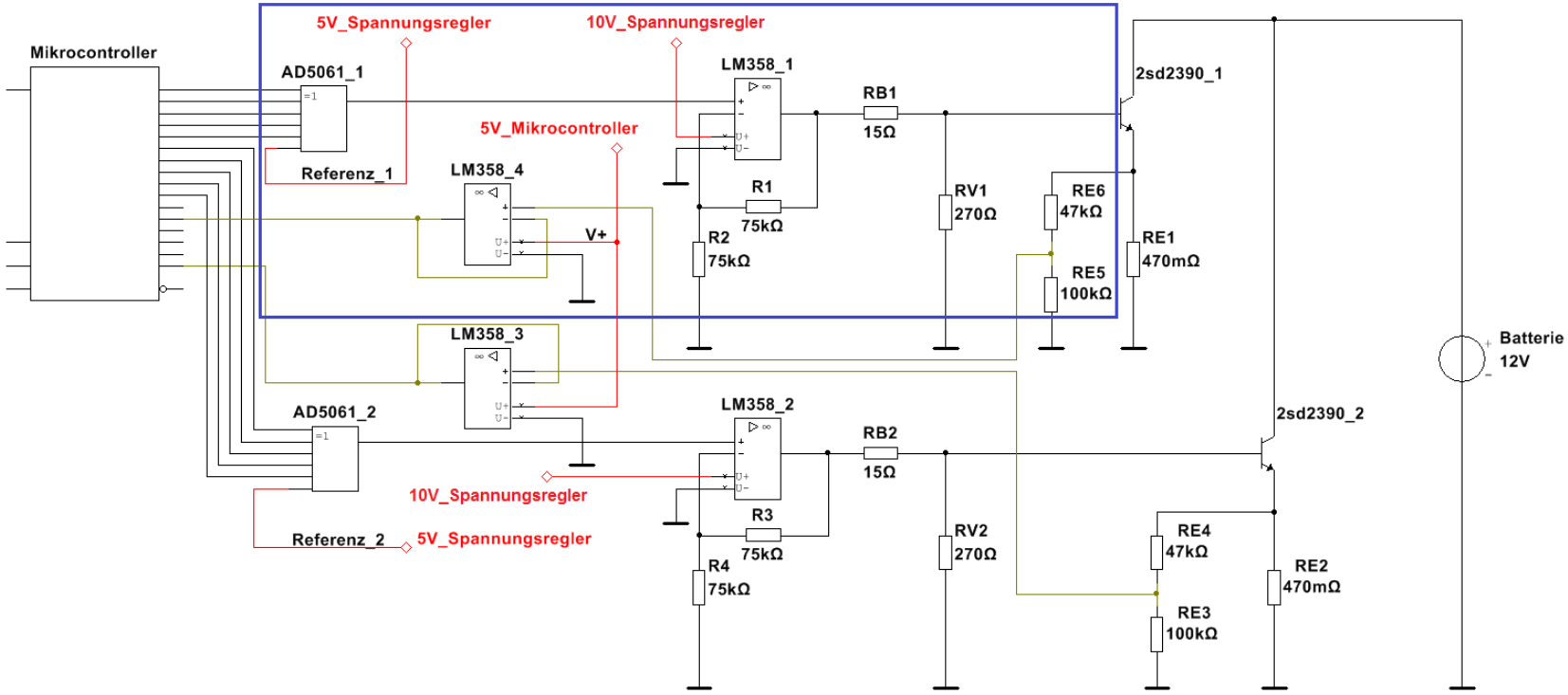

Abbildung 2.36.: Gesamtschaltung wie sie im Prototypen realisiert ist. Dabei sind die Ausgangsspannungen der Regler nur referenziert. Der blaue Rahmen beinhaltet die Bauteile, die auf einer Platine zusammenkommen. Dazu gehört jeweils auch ein 5 V Spannungsregler und ein 10 V Spannungsregler.

3. Ansteuerung

Die Ansteuerung der Transistorhardware ist die Schnittstelle zwischen Hardware und Software. Die Software muss die gewünschten Daten an den DA-Umsetzer schicken. Von ihm aus muss das Signal in der gewünschten Form den Transistor erreichen. Im Folgenden wird deshalb auf die Einstellungen und Modifikationen an der Ansteuerungshardware für den Transistor eingegangen. Weiterhin wird die Funktion der Software beschrieben. Als Grundgerüst für die Software, wurden UART, SPI und Timer-Funktionen sowie Header aus [1] verwendet. Die wesentlichen Funktionalitäten der Anregung finden in den Interrupt-Handlern und den Funktionen für die Berechnungen und Stromeinstellungen statt.

3.1. Anforderungen an die Ansteuersoftware

Unter Kenntnis der Funktionalität können Anforderung und Umsetzung der Software erfolgen. Ziel ist es, die Software so zu gestalten, dass das Anregesystem einfach mit dem Batteriesteuergerät zusammenarbeiten kann. Dabei soll die Software folgende Funktionalitäten aufweisen:

- Spannungen in korrespondierende Bit-Wert umrechnen, um die Transistorschaltung anzusteuern
- Entsprechende Spannung an die Transistorschaltung legen, um die Batterie mit dem gewünschten Strom zu entladen
- Aus der Batterie fließenden Strom einlesen, um den gewünschten Strom einzustellen
- Kommunikation mit Batteriesteuergerät
- Ausgangswerte entsprechend einer gewünschten Funktion (Sinusverlauf) einstellen
- Frequenzänderung des eingestellten Zeitsignals durch das Batteriesteuergerät ermöglichen
- Starten und Stoppen der Senke-Funktion durch das Batteriesteuergerät
- Starten und stoppen der kompletten Anregung durch entsprechende Befehle des Batteriesteuergeräts

Ziel ist es, dem Nutzer über das Batteriesteuergerät die Möglichkeit zu geben, eine komplette EIS-Messung mit einem einzigen Start-Befehl durchzuführen. Zunächst wird dabei von einer Anregung mit einem Sinussignal ausgegangen. Es muss geklärt werden, wie die Parameter des Sinus eingestellt und die Messung durch das Steuergerät initiiert und begleitet wird.

3.2. Einstellen der Hardware

Um die Ansteuerung von der Hardware-Seite korrekt durchzuführen, müssen einige Bauteildimensionierungen vorgenommen werden, die erst nach dem Testen der Softwareausgabe möglich sind.

3.2.1. DA-Umsetzer

Der Digital-Analog-Umsetzer AD5061 ist ein 16-Bit SAR-Umsetzer, der in der Schaltung vom Evaluation Board mit 5 V versorgt wird. Daraus ergibt sich die Stufenhöhe

$$\frac{5\,V}{2^{16}\,Bit} = 76,296\,\frac{\mu V}{Bit}\,. \tag{3.1}$$

Der DA-Umsetzer muss neben dem benötigten Basisstrom auch den Strom durch die Widerstände des Vorspannungsteilers treiben können. In Kapitel 3.2.2 ergeben sich diese zu 15 Ω und 270 Ω. Unter Vernachlässigung des Basisstroms ergibt sich ein zu treibender Strom von

$$I = \frac{5\,V}{15\,\Omega + 270\,\Omega} = 52,6\,mA \tag{3.2}$$

Der Hersteller des verwendeten Umsetzers gibt keinen maximalen Ausgangsstrom unter Last an, der Kurzschlussstrom bei analoger Null am Ausgang beträgt aber nur 45 mA [22]. Die Forderung des Stroms kann somit nicht erfüllt werden. Bei einer Aussteuerung von 5 V Basis-Emitter-Spannung beträgt der Kollektorstrom bei den verwendeten Transistoren etwa 5,8 A. Bei einem angestrebten Kollektorstrom von 6 A reicht dieser Wert nicht aus. Die Basis-Emitter-Spannung muss erhöht werden. Deshalb wird ein Operationsverstärker zwischen Umsetzer und Ansteuerung geschaltet. Diese Pufferstufe ist in der Lage, sowohl die Ansteuerspannung, als auch den notwendigen Strom zu liefern. Er wird über einen Spannungsregler von der zu messenden Batterie versorgt. Dabei wird seine Betriebsspannung

auf 10 V geregelt. Verwendet werden handelsübliche Operationsverstärker des Typs *LM358* [26]. Die Aussteuerbarkeit bei diesem Typen geht bis 1,5 V unter die Versorgungsspannung. Es ist also eine Eingangsspannung an den Vorspannungsteiler bis 8,5 V möglich. Da der Operationsverstärker das Ausgangssignal des DA-Umsetzers verstärken soll, wird er als nicht-invertierender Verstärker beschaltet. Dabei soll seine Verstärkung das maximale Eingangssignal auf sein maximales Ausgangssignal verstärken.

$$G = \frac{8,5\,V}{5\,V} = 1,7 \tag{3.3}$$

Die Verstärkung wird auf 2 aufgerundet. Dies vereinfacht die Beschaltung und stellt sicher, dass die maximale Aussteuerspannung trotz Widerstandstoleranzen erreicht werden kann. Lediglich die Softwarekonfiguration muss angepasst werden.

$$G = 1 + \frac{R_N}{R} \mid_{R=R_N} = 2 \tag{3.4}$$

Die Widerstände R und R_N werden jeweils mit 75 kΩ beschaltet.

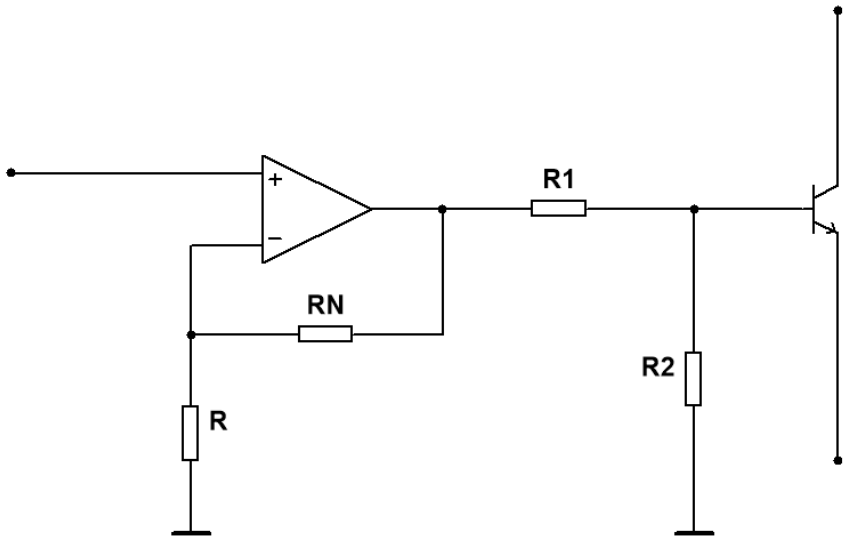

Abbildung 3.1.: Vorverstärker und Vorspannungsteiler zwischen DAU und Transistor. Sie dienen dazu, die Aussteuerung korrekt einzustellen.

3.2.2. Spannungsteiler zur Einstellung der Aussteuerstärke

Die Ansteuerung der Transistoren geschieht nach der Planung in Kapitel 2 zum Aufbau der Hardware direkt durch die DA-Umsetzer. Diese haben eine Aussteuerung von 0...5 V. In Kapitel 2.3 wurde bereits erläutert, dass die Ansteuerspannung durch die Nutzung von Darlington-Transistoren höher ist als bei Einzeltransistoren. Des Weiteren sollen die Transistoren durch die Ansteuerung mindestens bis 6 A durchgesteuert werden können. Der Vorspannungsteiler zur Arbeitspunkteinstellung muss deshalb entsprechend eingestellt werden.

Die Ströme durch R_1 und R_2 sind nicht identisch, da der Steuerstrom über die Basis abfließt. Die in ihnen umgesetzte Leistung und somit auch die an ihnen erzeugte Wärme unterscheiden sich folglich ebenfalls. Die unterschiedliche Erwärmung führt zu einer ungleichen Änderung der Widerstandswerte der beiden Widerstände. Das Spannungsteilerverhältnis ändert sich also durch veränderlichen Ansteuerstrom. Bei einer Stromverstärkung der Transistoren von etwa 9000 (siehe Kapitel 2.3.3) würden sich bereits kleine Änderungen des Ansteuerstroms deutlich auf den Kollektorstrom auswirken. Bei der Auslegung des Spannungsteilers kann bei richtigem Verhältnis von Basisstrom zu Widerstand R_2 (siehe Abbildung 3.2) dieser Temperatureinfluss reduziert werden. Dies verringert die Veränderung des Spannungsteilerverhältnisses, die sich auf den Steuerstrom auswirken würde [17]. Für gute Temperaturstabilisierung des Vorspannungsteilers wird ein Stromverhältnis von $I_{R2} \approx 5...10 \cdot I_B$ angestrebt [9].

Die maximale Spannung, die am Eingangspotential vor dem Widerstand R_1 (Abbildung 3.2) anliegt, ergibt sich aus der Ansteuerbeschaltung. Der vorgeschaltete Operationsverstärker wird über einen der beiden verwendeten Spannungsregler mit 10 V versorgt. Er verliert 1,5 V, die intern an Halbleitern abfallen. Er kann deshalb eine maximale Ausgangsspannung von 8,5 V ausgeben. Dies ist die maximale Ansteuerspannung des Vorspannungsteilers. Bei einer Verstärkung von 10000 bei einem Kollektorstrom von 10 A nach Abbildung 2.20 ergibt sich mit $I_B = \frac{I_C}{B}$ ein Basisstrom von 1 mA.

Wird der Basisstrom bei beliebiger Vorspannungsteiler-Kombination gemessen, so ergibt sich bei Vollaussteuerung durch die Stromdifferenz $I_2 - I_1$ ein Basisstrom von etwa 2,3 mA (Es sei angemerkt, dass diese Messung sehr ungenau ist, da dieser relativ kleine Strom während der Messung mitunter starken Schwankungen unterliegen kann).

Wird der Basisstrom zu Einhaltung der genannten Stabilisierungsbedingung auf 3 mA aufgerundet, so ergibt sich für den Strom I_2 von 30 mA. Eine Messung der Basis-Emitter-Spannung bei Vollaussteuerung ergibt 8 V. Sie entspricht der Spannung U_2. Für die Aussteuerung gilt: Ist eine maximale Aussteuerung möglich, auch wenn diese nicht gefordert ist, so stellt dies kein Problem dar. Kann jedoch nicht ausreichend durchgesteuert werden, so

können die benötigten Ströme mitunter nicht erzeugt werden. Deshalb wird in diesem Sinne dimensioniert. Zur Einhaltung der oben genanten Bedingung wird von einem maximalen Basisstrom von 3 mA ausgegangen. Mit diesem kann, unter Berücksichtigung der oben genannten Bedingung zur Temperaturstabilität, der Widerstand R_2 und daraus R_1 berechnet werden.

$$R_2 = \frac{U_1}{10 \cdot I_B} = \frac{8\,V}{30\,mA} = 266,67\,\Omega \tag{3.5}$$

$$R_1 = \frac{U_1}{U_2} \cdot R_2 = \frac{8,5\,V - 8\,V}{8\,V} \cdot 266,67 = 16,67\,\Omega \tag{3.6}$$

In der verwendeten E12-Reihe können diese Widerstände mit $R_1 = 15\,\Omega$ und $R_2 = 270\,\Omega$ angenähert werden. Dabei führt das Verkleinern von R_1 und das Vergrößern von R_2 dazu, die Basis-Emitter-Spannung tendenziell noch zu erhöhen.

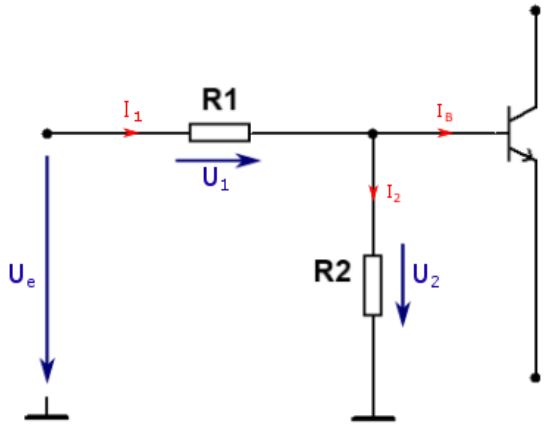

Abbildung 3.2.: Vorspannungsteiler der Leistungstransistoren. Die Dimensionierung der Widerstände legt die Aussteuerstärke bzw. die benötigten Ansteuerspannungen fest.

3.3. Konzeption und Umsetzung der Software

Die Software wartet zum Start und nach abgeschlossener Anregung auf einen Befehl. Mögliche Befehle sind

- Parameter zur Änderung des Gleichanteil des Sinussignals
- Parameter zur Änderung des Wechselanteil des Sinussignals
- Anregung starten
- Anregung stoppen

Bei Änderung des Zustands wird vom Anregesystem eine entsprechende Information an das Batteriesteuergerät gesendet. Die möglichen Zustände sind ein Bereit-Zustand, ein Initialisierungszustand und ein Anregezustand. Im Bereit-Zustand werden sowohl Parameteränderungen angenommen als auch einen Befehl zum Starten empfangen. Dieser führt zu einem Zustandswechsel. Ist dies geschehen, so wird der Initialisierungszustand eingenommen, in dem das System den gewünschten Gleich- und Wechselanteil einstellt. Nachdem dies geschehen ist, wird automatisch die Anregung gestartet und das entsprechende Signal an das Batteriesteuergerät gesendet. In diesem Zustand bleibt das System, bis ein Beendet-Befehl bzw. eine Parameteränderung empfangen wird. Abbildung 3.3 fasst diese Zusammenhänge der Zustände in einem vereinfachtem Zustandsdiagramm zusammen.

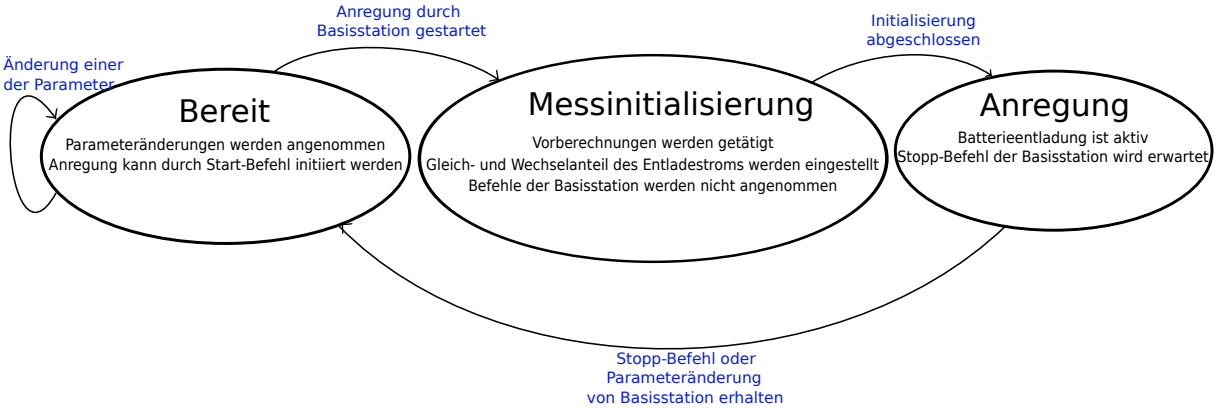

Abbildung 3.3.: Vereinfachter endlicher Zustandsautomat der Steuersoftware. Eventuelle Fehler werden in diesem Diagramm nicht berücksichtigt.

Abbildung 3.4 zeigt den Programmablaufplan des geschriebenen Programms. Dabei werden die über UART übertragenen Befehle hexadezimal übertragen. Die hexadezimalen Codierungen sind wie folgt:

- Parameter zur Änderung des Gleichanteil des Sinussignals - 0x2
- Parameter zur Änderung des Wechselanteil des Sinussignals - 0x3

- Anregung starten 0x1
- Anregung stoppen 0x0

Nachrichten, die nicht einem dieser Werte entsprechen, werden ignoriert. Wird die Anregung gestartet bzw. gestoppt, wird eine entsprechende Nachricht an den Master gesendet.

- Anregung gestartet - 0x1
- Anregung gestoppt - 0x0

Sie sollen das Batteriesteuergerät über den Zustand der informieren.

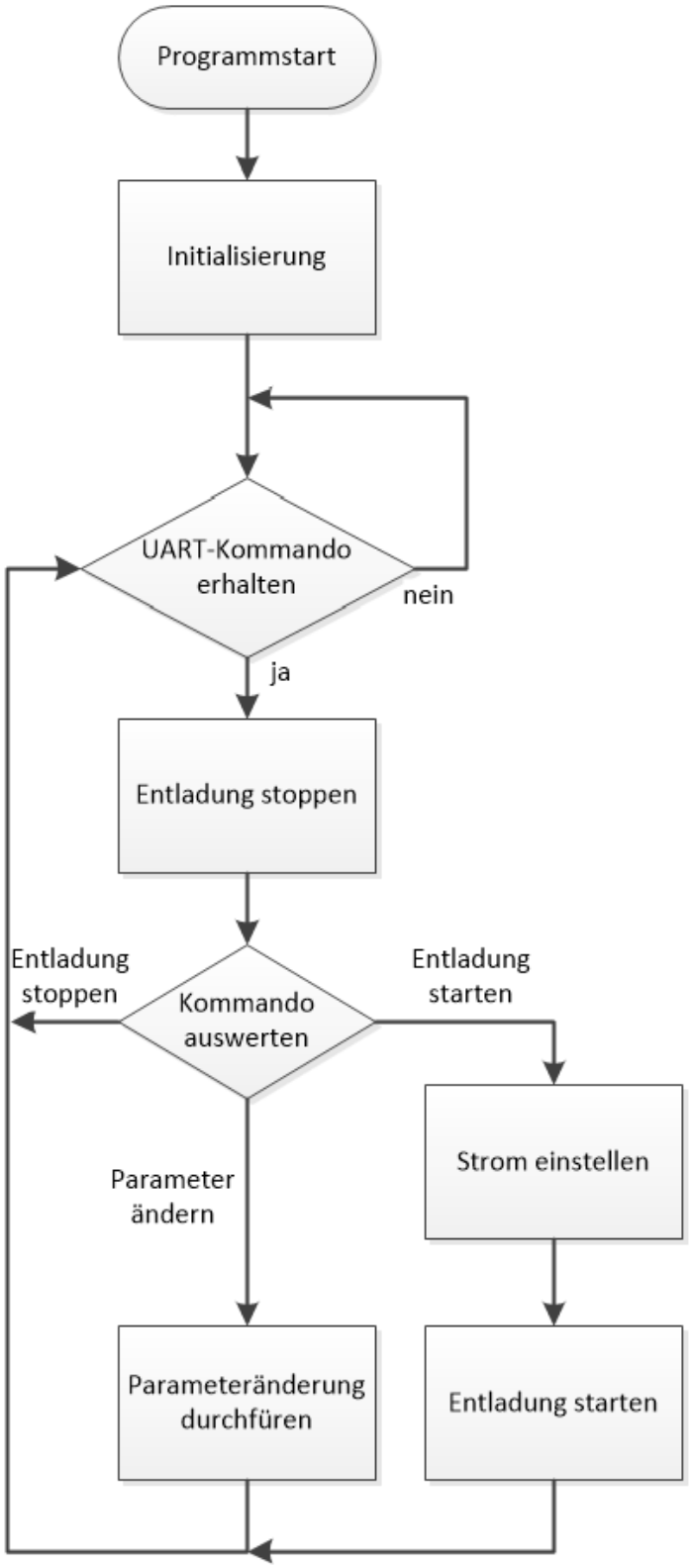

Abbildung 3.4.: Programmablaufplan der geschriebenen Ansteuersoftware. Der Ausgangszustand ist eine leere Endlosschleife, in der auf Befehle vom Steuergerät gewartet wird. Je nach Befehlt wird weiter agiert.

3.3.1. Aufteilung des Gleichstromanteils auf Gleich- und Wechselstromstrang

Sowohl der Gleich- als auch der Wechselanteil werden vor der Aktivierung der Anregung einmalig auf den Sollwert eingestellt.

Zunächst wird der Gleichanteil behandelt. Bis zu 4 A des gesamten Gleichanteils werden immer von dem Transistor übernommen, der den Wechselanteil erzeugt. Ist der Gesamtgleichanteil größer als 4 A, so wird der restliche Gleichanteil vom Transistor für den Gleichanteil erzeugt. Grund hierfür ist, dass der Sinus immer mindestens einen Offset von einem Wert gleich seiner Amplitude und dem Mindeststromwert haben muss. Als Bedingung gilt „der Wechselanteil darf nicht größer als der Gleichanteil sein". Dies ist entsprechend in die Software integriert. Ist der Gleichstromanteil kleiner als die Hälfte des maximalen Stroms, übernimmt deshalb der Transistor des Wechselstromstrangs den kompletten Gleichstrom. Ist der Gleichstromanteil größer als die Hälfte des maximalen Stroms, übernimmt der Transistor des Wechselstromstrangs die Hälfte des maximalen Gleichstroms. Die fehlende Stromdifferenz übernimmt dann der Transistor des Gleichstromstrangs. Die Abfrage nach der Hälfte des maximalen Stroms ist im Programm auf 4 A festgelegt. Es wird also von einem maximalen Strangstrom im Wechselstrang von 8 A, 4 A Gleich- und 4 A Wechselstrom ausgegangen. Messungen am Transistor und spätere Messungen zeigen, dass 8 A noch innerhalb des linearen Bereichs liegen. Die folgenden Beispiele verdeutlichen die Funktionalität noch einmal:

- Gewünschter Strom: 2 A Gleichstrom mit 500 mA Wechselstromamplitude.

2 A liegen innerhalb des halben maximalen Stroms 4 A. Der Gleichanteil wird deshalb komplett vom Wechselstromstrang übernommen. Die Amplitude von 500 mA wird zusätzlich einfach dazu moduliert.

- Gewünschter Strom: 10 A Gleichstrom mit 2 A Wechselstromamplitude.

10 A liegen ausserhalb des halben maximalen Stroms 4 A. Deshalb wird eine Aufteilung des Gleichanteils vorgenommen.

$$Aufteilung\ 10\,A - 4\,A = 6\,A \tag{3.7}$$

Es werden 4 A vom Wechselstromstrang übernommen und die restlichen 6 A vom Gleichstromstrang. Auf den Wechselstromstrang wird zusätzlich die Wechselstromamplitude moduliert.

- Gewünschter Strom: 4 A Gleichstrom mit 5 A Wechselstromamplitude.

Es kommt zu einer Fehlermeldung und die Ausführung der Anregung wird verweigert, da die Bedingung „Wechselstromamplitude nicht größer Gleichstrom” nicht eingehalten wird.

3.3.2. Einstellen des gewünschten Stroms

Nach dem Anrege-Startbefehl mit korrekten Parametern werden Gleich- und Wechselanteil auf die korrekten Stromwerte eingestellt. Dazu wird der Istwert von Null an den Sollwert angenähert, bis dieser überschritten wird. Nach der Überschreitung des Sollwerts wird der aktuelle Wert um den doppelten vorherigen geringeren Wert reduziert. Nun wird die Wert um die zuvor erhöhte Menge verringert. Anschließend wird der Istwert wieder erhöht. Dieser Vorgang wird wiederholt bis der Istwert dem Sollwert plus/minus einer bestimmten Toleranz entspricht. Diese Einstellregelung geschieht zunächst für den Gleichstromstrang, danach für den Gleichanteil des Wechselstromstrangs und dann für den Wechselanteil. Diese Vorgehensweise wird also sowohl für den Gleichstromstrangs als auch für Gleichanteil und Wechselanteil des Wechselstromstrangs verwendet. Nach erfolgreichem Abschluss der Regelung von Gleich- und Wechselanteil wird die Anregung gestartet und die entsprechende Statusnachricht wird an das Batteriesteuergerät gesendet.

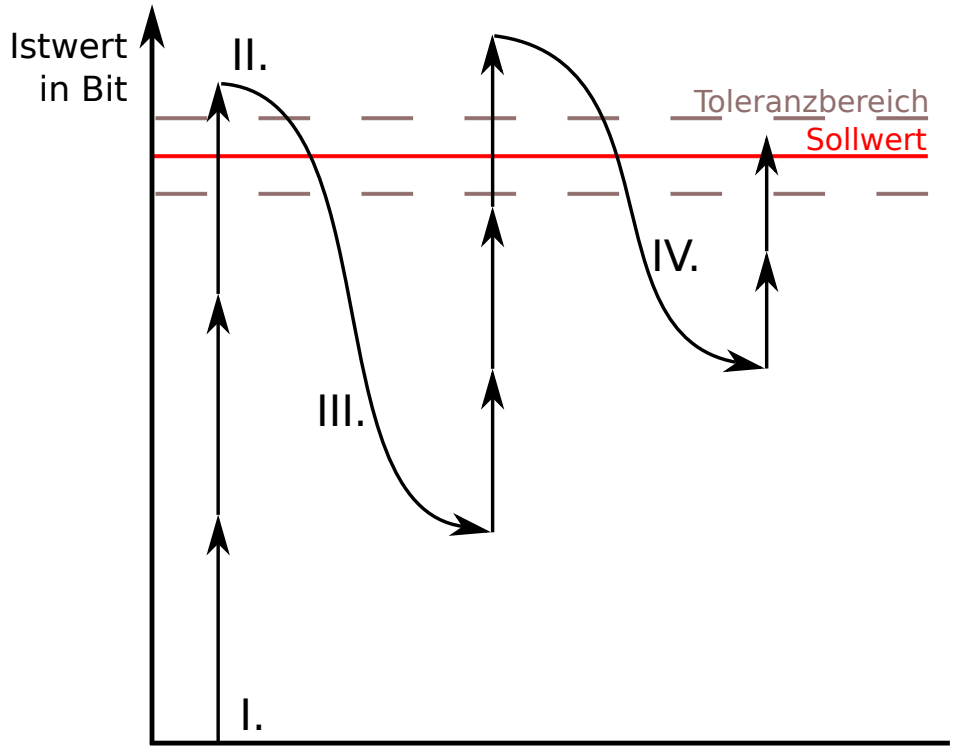

Abbildung 3.5.: Veranschaulichung des Regelvorgangs zur Stromeinstellung. Ziel ist ein Bit-Wert, der dem Sollwert innerhalb der Toleranzen entspricht. 1. Der Istwert wird solange um einen Wert erhöht bis der Sollwert erreicht oder überschritten wird. 2. Der Sollwert wurde überschritten. Es wird um zwei Werte zurück gesprungen. 3. Der Wert, um den erhöht wird, wird reduziert. Dieser Vorgang wird wiederholt bis 4. eintritt.

3.3.3. Das Sinussignal in Software

Stützstellen

Für die EIS-Messungen sollen Sinuswerte bei verschiedenen Frequenzen gemessen werden. Die 13 zu messenden Frequenzen sind nach [1] an dem bestehenden System gewählt. Der Sinus der Anregung soll den Frequenzvorgaben so gut wie möglich entsprechen. Der erzeugte Sinus wird digital durch Stützstellen angenähert. Ein hohe Zahl an Stützstellen N pro Sinusperiode reduziert spektral die vorhandenen Oberwellen und führt zu einem „saubererem" Sinus. Daraus resultiert mehr Leistung in der, durch den Goertzel-Algorithmus berechneten Spektrallinie. Das verändert auch das Ergebnis der EIS-Messung.

So benötigt ein Sinussignal mit 100 Punkten N (Stützstellen) bei einer Frequenz von 1 Hz, 100 Ausgaben des DAUs in einer Sekunde, eine Ausgabefrequenz von 100 Hz. Bei einer Signalfrequenz von 2 Hz werden 200 Ausgaben benötigt. Wird die Zahl der Punkte auf 200 erhöht, so werden 400 Ausgaben benötigt. Der Zusammenhang wird durch Formel 3.8 beschrieben.

$$f_{DAU} = f_{sin} \cdot N \tag{3.8}$$

Das Ergebnis der EIS ist der charakteristische Ast im Nyquist-Plot. Der Real- und der Imaginärteil, aus denen jeder Punkt des Astes geplottet wird, werden in den Zellensensoren des Messsystems durch Widerstand und Strom-Spannungs-Phasenversatz ermittelt und mittels Goertzel-Algotithmus berechnet. Der Goertzel-Algorithmus berechnet die Spektralkomponente bei einer bestimmten, vorgegebenen Frequenz. Liegt die erzeugte Signalfrequenz spektral nicht auf der vom Algorithmus beobachteten Frequenz, sondern daneben, wird der Hauptanteil der Signalleistung kaum oder gar nicht in der errechneten Spektrallinie auftauchen. Je weiter sich die Frequenz von der Goetzel-Frequenz entfernt, desto weniger Signalleistung wird in den Algorithmus in der Spektrallinie auftauchen. Aus diesem Grund muss die Frequenz der Anregung so exakt wie möglich mit der, über die Zeit und die Abtastrate definierten, selektierten Goertzel-Frequenz übereinstimmen [27].
Die Anforderungen an die Frequenzerzeugung sind folglich eine möglichst hohe Ausgabefrequenz des DAUs bei einer hohen Anzahl an Punkten. Nach Formel 3.8 ist also die maximale Anzahl der Punkte, mit einer maximalen Signalfrequenz von 2 kHz nach [1] limitiert durch die maximale Ausgabefrequenz.

$$N = \frac{f_{DAU}}{f_{sin}} = \frac{f_{DAU}}{2\,kHz} \tag{3.9}$$

Die maximale Ausgabefrequenz (Clock Rate) des verwendeten DA-Umsetzers beträgt 30 MHz [22]. Die Ausgabe eines Wertes alle 33,4 ns kann durch den Umsetzer realisiert werden. Praktisch ist diese Zeit nicht einhaltbar, da der Interrupt-Handler, in dem die Ausgabe der Werte durchgeführt wird, eine Ausführungszeit von etwa 2,77 μs hat (Abbildung 3.6). Die höchst mögliche Ausgabefrequenz ist also $\frac{1}{2{,}77\,\mu s} \approx 361\,kHz$. Mit Formel 3.8 ergibt sich dann eine Stützstellenzahl von N = 180 für die maximal geforderte Frequenz von 2 kHz (3.10).

$$N = \frac{f_{DAU}}{f_{sin}} = \frac{361\,kHz}{2\,kHz} \approx 180 \tag{3.10}$$

Eine Periode kann folglich maximal 180 Stützstellen beinhalten, um die maximale Frequenz noch ausgeben zu können. Es ist möglich, die Zahl der Stützstellen an die Signalfrequenz anzupassen, um jeweils das Maximum an Stützstellen zu erhalten. Die Entscheidung, dies anzuwenden, wird durch die Wahl der Art der Frequenzerzeugung getroffen.

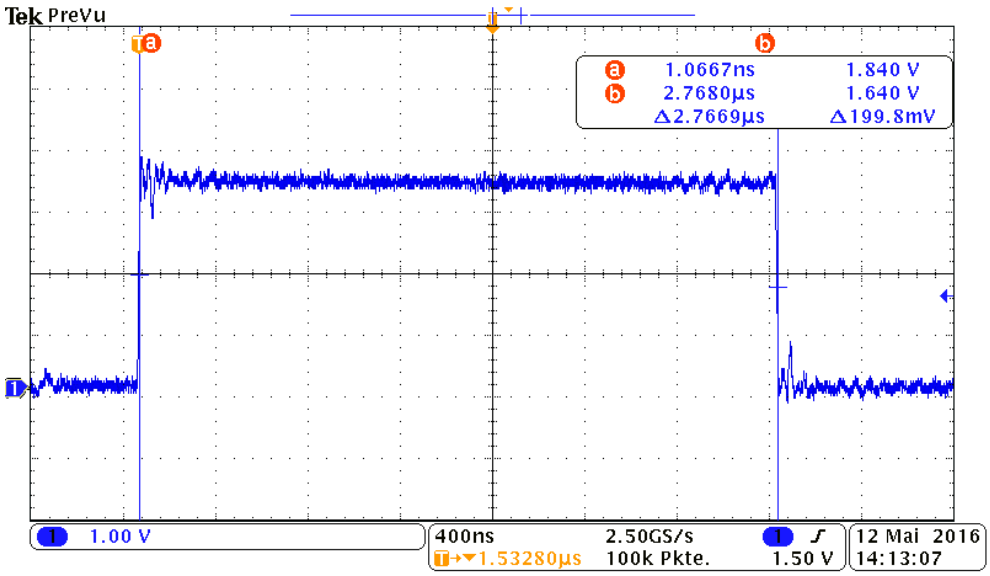

Abbildung 3.6.: Messung der Laufzeit des Interrupt-Handlers zur Ausgabe der Sinuspunkte. Für die Messung wird am Anfang des Handlers ein Mess-Pin auf High gesetzt und am Ende auf Low zurückgesetzt. Die Dauer des High-Zustands wird gemessen. Sie beträgt etwa 2,77 μs.

Erzeugung der Sinus-Daten

Daten in Form eines Sinussignals sollen für die Anregung über die DA-Umsetzer an die Transistoren übertragen werden. Dazu müssen diese erzeugt und gespeichert werden.
Eine Möglichkeit zur Ausgabe der Sinuswerte ist das Abrufen von vorgespeicherten Werten aus einer Tabelle. Die Tabelle liegt im Programm vor und die benötigten Daten werden bei Ausführung des Programms abgerufen. Die errechneten Werte Datenwerte des entsprechenden Sinus für jede vordefinierte Anforderung an Gleich- und Wechselanteil werden in

einer Tabelle, zum Beispiel einem mehrdimensionalen Array, abgelegt. Dadurch ist das Programm in der Lage, den gewünschten Sinus ohne Rechenzeit auszugeben, da er vorher aus einer Tabelle ausgelesen wurde. Diese Methode ist auf Kosten von Speicher dafür besonders rechen-effizient. Ein Nachteil dieser Vorgehensweise ist, dass alle möglichen Parameter vorher definiert und eingespeichert werden müssen. Sollen in der Anwendung Stromwerte oder Kombinationen haben, die nicht in der Tabelle festgehalten werden, können diese nicht ohne Berechnungen ausgegeben werden. Der Vorteil des ausbleibenden Rechenaufwands ist damit nicht mehr gegeben. Eine weitere Möglichkeit ist die Berechnung des Sinus innerhalb des laufenden Programms. Die ermöglicht das dynamische Errechnen der geforderten Werte. Das bedeutet Amplituden- und Gleichanteilwerte können (innerhalb des Gültigkeitsbereichs der Variablen) beliebig gewählt und berechnet werden. Ein weiterer Vorteil ist, dass die Anzahl der Stützstellen beliebig variiert werden kann, wenn dies gefordert ist. Die errechneten Werte werden dann in einem Array gespeichert und bei Bedarf abgerufen. Bei dieser Variante bietet es sich an, eine Normierung des Signals auf die Amplitude û = 1, die Frequenz f = 1Hz und den Gleichanteil G = 0 als Ausgangspunkt der Berechnung zu wählen. Sollen nun Veränderungen vorgenommen werden, kann das Array durch Multiplikation mit dem Faktor A, die Amplitudenhöhe verändert werden, mit der Veränderung von N können die Punkte innerhalb einer Periode angegeben werden und mit einer Addition von G kann der gewünschten Gleichanteil addiert werden.

$$\hat{u} \cdot sin(2\pi f t) \tag{3.11}$$

$$A \cdot sin\left(\frac{2\pi f t}{N}\right) + G \tag{3.12}$$

Nachteil dieser Methode der Ausgabe der Sinuswerte ist die Rechenzeit, die benötigt wird, um die Sinuswerte zu ermitteln.

In der Struktur des Programms wird nach Aktivierung der Anregung zuerst der Strom eingestellt. Ist die abgeschlossen, beginnt die Anregung. In Abbildung 3.3 ist dieser Vorgang in dem Zustand „Messinitialisierung” zusammengefasst. Dieser Zustand ist nicht zeitkritisch und wird einmal komplett durchlaufen. Aus diesem Grund kann die dynamische Berechnung der Sinuswerte zur Laufzeit geschehen, ohne Einschränkungen in der Funktionalität einzubüßen. Bei der Berechnung handelt es sich um ein einfache Schleife mit N Durchläufen. Die CPU des Controllers ist mit 120 MHz getaktet. So läuft das Programm in Echtzeit. Für den Anwender ist die zusätzliche Wartezeit nicht wahrnehmbar.

Frequenzerzeugung

Wie bereits erwähnt müssen, Anrege-Signalfrequenz und selektierte Frequenz des Goertzel-Algorithmus übereinstimmen. Es soll deshalb gelingen, die beiden Frequenzen so weit wie möglich aneinander anzupassen. Theoretisch ist es möglich, die Ausgabe des Sinussignals zu einer beliebigen Frequenz einzustellen. Um eine Anregefrequenz in Abhängigkeit der Eingabe eines Nutzers einzustellen, muss sowohl die Stützstellenzahl als auch die Handlerlaufzeit berücksichtigt werden. Es kann dann eine Verzögerungszeit zwischen den Ausgaben des DA-Umsetzers integriert werden. Es ergibt sich eine Stufenbreite (Stützstellenabstand), die der Summe der Laufzeit des Handlers und der veränderbaren Verzögerungszeit entspricht (Abbildung 3.7). In der Software kann dies durch einen Hardware-Timer im Output-Compare Mode realisiert werden. Dabei wird bei Ablauf des Timers ein Handler aufgerufen, der die Ausgabe der Werte durchführt.

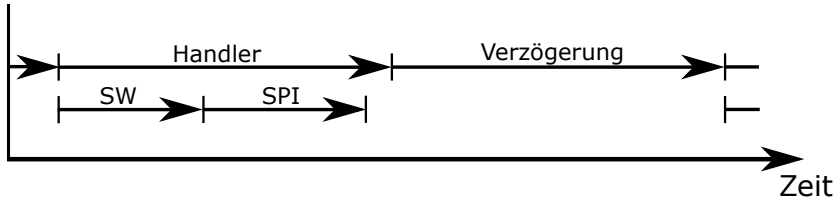

Abbildung 3.7.: Zeitlicher Ablauf zwischen den DA-Umsetzer-Ausgaben. Der Timer-Handler Aufruf initiiert die Ausgabe. Nach der Ausführung einiger Software (SW), werden die Daten über die SPI-Schnittstelle an den DA-Umsetzer gesendet. Nach diesem Vorgang wird der Interrupt-Handler beendet. Danach tritt die, je nach Signalfrequenz, veränderliche Zeitverzögerung durch einen Hardware-Timer ein. Läuft dieser ab, so wird der Timer-Handler erneut aufgerufen. Der Ablauf wiederholt sich.

Das Einstellen des Parameters Signalfrequenz soll für das Batteriesteuergerät und für den Nutzer einfach gestaltet werden. Die Angabe einer Frequenz soll diese Frequenz erzeugen. Soll zum Beispiel eine Signalfrequenz von 1 kHz erzeugt werden, so beträgt die Handlerlaufzeit t_{ISR} inklusive Ausführung, Sprünge des Programms in und aus dem Handler sowie SPI-Sendung und die addierte Timer-Verzögerung

$$t_{ISR} = \frac{N}{f} = \frac{100}{1\,kHz} = 100\,ms\,. \tag{3.13}$$

Die externe Taktfrequenz, die vom Batteriesteuergerät vorgegeben werden muss, um eine gewünschte Frequenz zu erzeugen, die Timer-Verzögerung dieser Zeit entsprechen:

$$f_{CLK} = \frac{f}{N} = t_{ISR} = 100\,ms \tag{3.14}$$

Die Erproung dieser Lösung der Frequenzerzeugung hat gezeigt, dass es praktisch äußerst aufwendig, bis hinzu unmöglich ist, die gewünschte Frequenz exakt zu erzeugen. Es kommt hinzu, dass die Quarzoszillatoren der beiden Controller eine Toleranz haben. Diese liegt bei etwa 30 ppm, ist also sehr gering [21]. Das bedeutet, dass von einer minimalen Taktabweichung der beiden Controller zueinander ausgegangen werden muss. Da beide Controller bei Angabe einer Mess- oder Anregefrequenz die Zeit (Sekunden) als Referenz benutzen, ist eine exakte Frequenzanpassung des Anregesignals an die geforderte Frequenz unmöglich, wenn die genaue Taktabweichung voneinander nicht bekannt ist. Denn selbst wenn die geforderte Frequenz in Hertz exakt erzeugt wird, entspricht dieses bei Abweichung der Taktrate nicht exakt der Selektionsfrequenz des Messystems. Die zeitliche Bezugsgröße muss identisch sein.

Dies führt zu einer Lösung, die prinzipiell von der Zeit in Sekunden und entsprechend von Frequenzen in Hertz unabhängig ist. Die Idee ist, dass der Takt das Batteriesteuergerät verwendet wird, um die für die Messungen vorgegebenen Frequenzen grob einzuhalten. Wird nun, anstelle des abweichenden Systemtakts des Anrege-Controllers, ein Takt von das Batteriesteuergerät vorgegeben, so ist die Referenzgröße dieser externe Takt, der für die Basistation der Vorgabefrequenz entspricht. Wenn die unbekannte Taktfrequenz das Batteriesteuergerät einen Sinus erzeugt, welcher der Selektionsfrequenz des Goertzel-Algorithmus entspricht, so spielt es keine Rolle, um welche tatsächliche Frequenz es sich hierbei handelt.

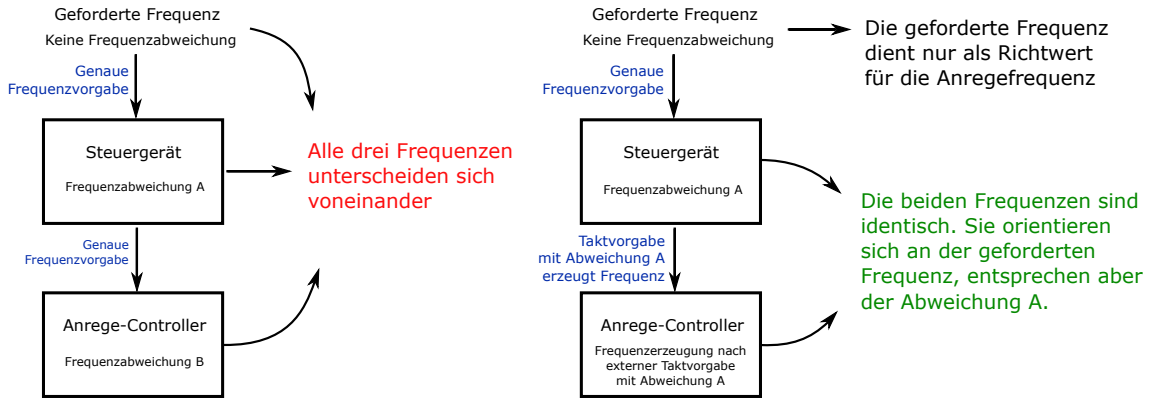

Abbildung 3.8.: Varianten zur Erzeugung der Anregefrequenz. Links: Variante nach genauer Vorgabe einer Frequenz von außen. Die resultierende Anregefrequenz entspricht nie genau der gewünschten. Rechts: Das Steuergerät versucht die Frequenzvorgabe von außen zu erfüllen und gibt einen entsprechenden Takt an den Anrege-Controller weiter. Die resultierende Anregefrequenz entspricht der Vorgabe des Steuergeräts.

Lediglich das Messergebnis, die Impedanz der Batterie, verändert sich in Abhängigkeit der wahren Frequenz mit der Referenz Zeit in Sekunden. Liegt die erzeugte Frequenz nahe an der Wunschfrequenz, verschiebt sich auch der Punkt im Nyquist-Plot nur geringfügig. Das Ergebnis unterliegt ohnehin einer Regression, das heißt es wird die Kurve durch die Punkte skizziert. Aus den Messpunkten wird eine Kurve ermittelt (Ast-Form). Abweichungen von einigen Hertz spielen deshalb keine Rolle.

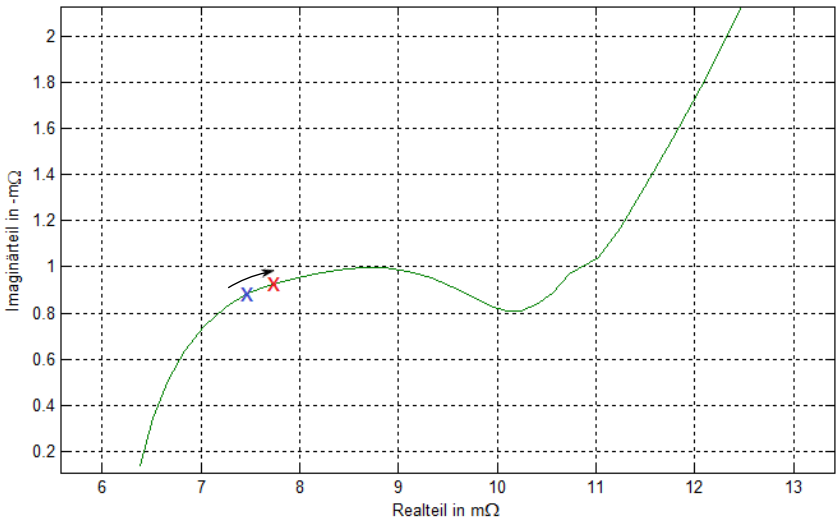

Abbildung 3.9.: Das Ergebnis einer EIS-Messung. Wenn das rote Kreuz den 100 Hz Punkt markiert, so könnte das blaue Kreuz den von das Batteriesteuergerät angenäherten 100 Hz Punkt darstellen. Zwar liegen die beiden Punkte nicht aufeinander, qualitativ ist das Gesamtergebnis aber gleich, da die durch die Punkte gelegte Kurve in beiden Fällen gleich verläuft.

Die zeitliche Begrenzung wird für hohe Frequenzen so ebenfalls durch die Handlerlaufzeit definiert, kann aber von außen durch Taktänderung auch während der Anregung beliebig geändert werden.

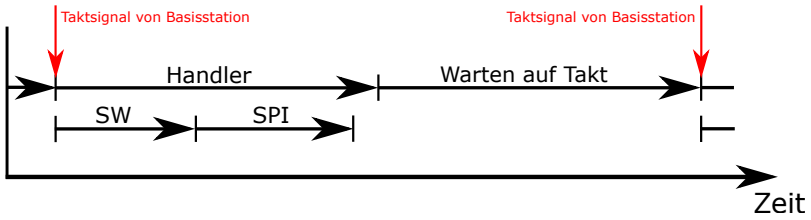

Abbildung 3.10.: Frequenzerzeugung durch externen Takt des Batteriesteuergeräts. Der zeitliche Ablauf ist prinzipiell identisch mit dem der Frequenzerzeugung durch Timer Interrupt. Mit der Methode der Frequenzerzeugung durch externen Takt kann die gewünschte Goertzel-Frequenz jedoch exakt und während der laufenden Anregung eingestellt werden wenn Handlerlaufzeit und Stützstellenzahl bekannt sind.

Für die Methode der Frequenzerzeugung mit Hilfe eines externen Takts kann nun festgelegt werden, welche Anzahl an Stützstellen für den Sinus geeignet ist. In weiter oben wird die maximale Stützstellenzahl 180 berechnet, um eine Signalfrequenz von 2 kHz noch auszugeben. Da sich die Frequenz jedoch nach Formel 3.15 berechnet, ist dieser Wert ungünstig. Wird die Stützstellenzahl 100 gewählt, lässt sich die Abhängigkeit der Frequenz f des Anregestroms weit leichter berechnen.

$$f = \frac{f_{CLK}}{N}|_{N=100} \tag{3.15}$$

Die Zusammensetzung der Handlerlaufzeit nach 3.7 und Ausgabe der Datenwerte des Sinussignals in einem Handler sehen grundsätzlich in etwa wie folgt aus:

```
// sinus[LENGTH] sei ein Integer-Array der Länge LENGTH
// mit auszugebenden Sinusdaten
// i sei eine globale  Zählvariable

    DAC_write(sinus[i]); //Beschreiben des SPI-Senderegisters für den DAU
    i++;                 //i inkrementieren
    if(i >= LENGTH){     //wurde das Array einmal durchlaufen,
        i = 0;           //wird die Zählvariable auf 0 zurückgesetzt.
    }                    //Die Ausgabe beginnt vom ersten Element.
```

Bei beiden Lösungen zur Frequenzerzeugung tritt ein Problem auf, welches es erschwert, die gewünschte Frequenz exakt zu erzeugen. Nach einem Komplettdurchlauf des Arrays wird die Zählvariable auf 0 gesetzt. Dieser Schritt nimmt Rechenzeit in Anspruch. Dies verhindert eine äquidistante Ausgabe zwischen der Ausgabe des ersten und des letzten Datenpunkts. Es ist nicht möglich, eine Abhängigkeit der Frequenz von der einzustellenden Verzögerungszeit und der Handlerlaufzeit in Form einer linearen Funktion zu beschreiben. Selbst wenn es gelingt, diese Abhängigkeit zu beschreiben und in die Frequenzerzeugung einzubinden, ist es praktisch nicht möglich, die gewünschte Frequenz beliebig einzustellen und den gewünschten Frequenzen über die gesamte Frequenzbandbreite bei der Anregung genau zu entsprechen. Die ungewollte Verzögerung erhöht sich verhältnismäßig, je größer die Stützstellenzahl beziehungsweise je kleiner die Stufenbreite ist.

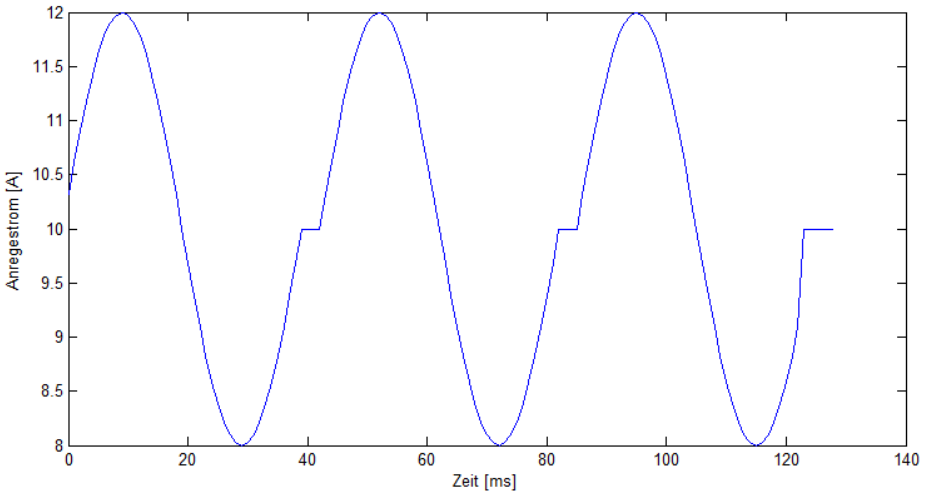

Abbildung 3.11.: Qualitative Simulation der Verzögerung der Ausgabe des Sinus an den DA-Umsetzer. Es entsteht aufgrund des Zurücksetzens der Laufvariable eine ungewollte Verzögerung zu Beginn jeder Periode.

Dieses problem wirkt sich einerseits auf die erzeugte Frequenz und andererseits auf das Oberwellenverhalten des Signalspektrums aus. Die Messungen in Kapitel 4 zeigen jedoch, dass dieser Einfluss bei N = 100 Stützstellen vernachlässigbar ist.

3.3.4. Korrektur des Ausgangsstroms

Ideal ergibt der maximale Vorgabewert in Bit, den maximalen Kollektorstrom. Der Zusammenhang von Bit-Wert und Kollektorstrom verhält sich dabei linear. Die Messung der Ausgangsströme bei verschiedenen Vorgabewerten, zeigt Abweichungen des zu erwartenden Werts. Diese verhalten sich im Gleich- und Wechselstrang unterschiedlich, sind aber in beiden Fällen linear. Werden die sich ergebenden Stromwerte in Abhängigkeit der Ausgabe-Bit-Werte aufgenommen, so ergibt sich ein nahezu linearen Verlauf, der sich aber von den zu erwartenden Werten unterscheidet. Die Ausgabe soll so korrigiert werden, dass der tatsächliche Kollektorstromwert, der Vorgabe entspricht. Dazu werden die Werte für jeden Bit-Wert, in Tausender-Schritten, sechs mal aufgenommen und dann gemittelt. Es ergibt sich der Verlauf der Balken in Abbildung 3.12. Durch die Berechnung der linearen Regressionsgeraden durch die ermittelten Punkte, ergibt sich ihre Steigung und ihr Ordinatenabschnitt. Diese Korrekturfunktion der Form $y = m \cdot X + b$ kann umgestellt zu $x = \frac{y-b}{m}$ verwendet werden, um die tatsächlichen Ausgabe-Bit-Werte für einen Strom zu ermitteln und auszugeben. Die Funktion wird in der geschriebenen Software auf den Ausgabewert angewendet. Auf diese Weise wird

der Wert x vorgegeben, der für den angestrebten Strom y benötigt wird. Diese Korrektur wird jeweils für den Gleichstromstrang und für den Wechselstromstrang durchgeführt. Abbildung 3.12 zeigt beispielhaft das Ergebnis für die Korrektur des Wechselstromstrangs von 0 bis 17000 Bit. Das Ergebnis des Gleichstromstrangs sieht vergleichbar aus.

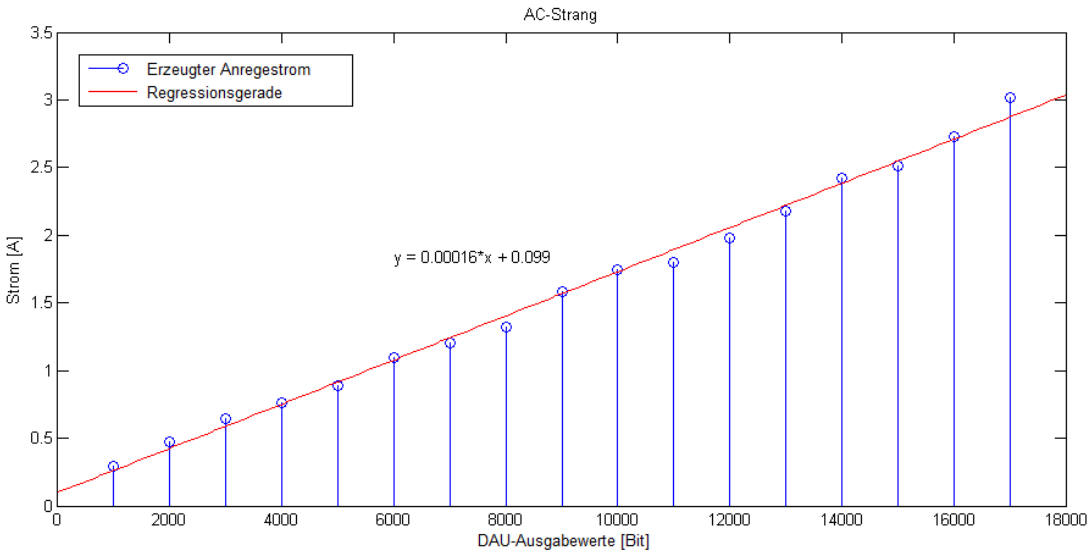

Abbildung 3.12.: Aufnahme der tatsächlichen Kollektorströme in Abhängigkeit des Bit-Werts. Die rote Gerade ist die lineare Regressionsgerade der Stromwerte. Sie dient der Korrektur des Stroms.

3.4. Hinweise zur Einbindung an das Batteriesteuergerät

Um das Anregesystem an das bestehende System anzugliedern, sollten einige Punkte berücksichtigt werden. Die entwickelte Steuerstruktur des Anregesystems ist so ausgelegt, dass es die manuelle Anregung mittels PC, aber auch die automatisierte Anregung über das Batteriesteuergerät erlaubt. Für die Automatisierung sollten die Kontrollbefehle und Rückmeldungen der Anregesoftware verwendet werden. Diese werden in Kapitel 3.3 genannt und erläutert.

Eine Anregung bei einem gewünschtem Strom erzeugt, aufgrund der Verlustleistung, Wärme in den Transistoren. Diese Wärme erreicht nach einiger Zeit eine Temperatur, die abhängig vom eingestellten Arbeitspunkt ist. Sie variiert bei der Anregung zwar aufgrund des Wechselanteils besonders bei geringen Anregefrequenzen, ist aber im Mittel entsprechend der Wärmeentwicklung des Gleichstromanteils. Deshalb sollte vor einer EIS-Messung aus

dem Ruhezustand der Transistoren eine Aufwärmphase eingeplant werden, bei der die Wechselamplitude auf 0 A gesetzt wird. So können die Transistoren sich auf die Arbeitstemperatur einstellen, bevor die Messung stattfindet. Dies verhindert die einseitige Änderung des Kollektorstroms und somit auch des Entladestroms während der Messung.

Abbildung 3.13 zeigt den möglichen Ablauf einer gesamten automatisierten EIS-Messung. Dabei werden die einzelnen Schritte vom Batteriesteuergerät initiiert.

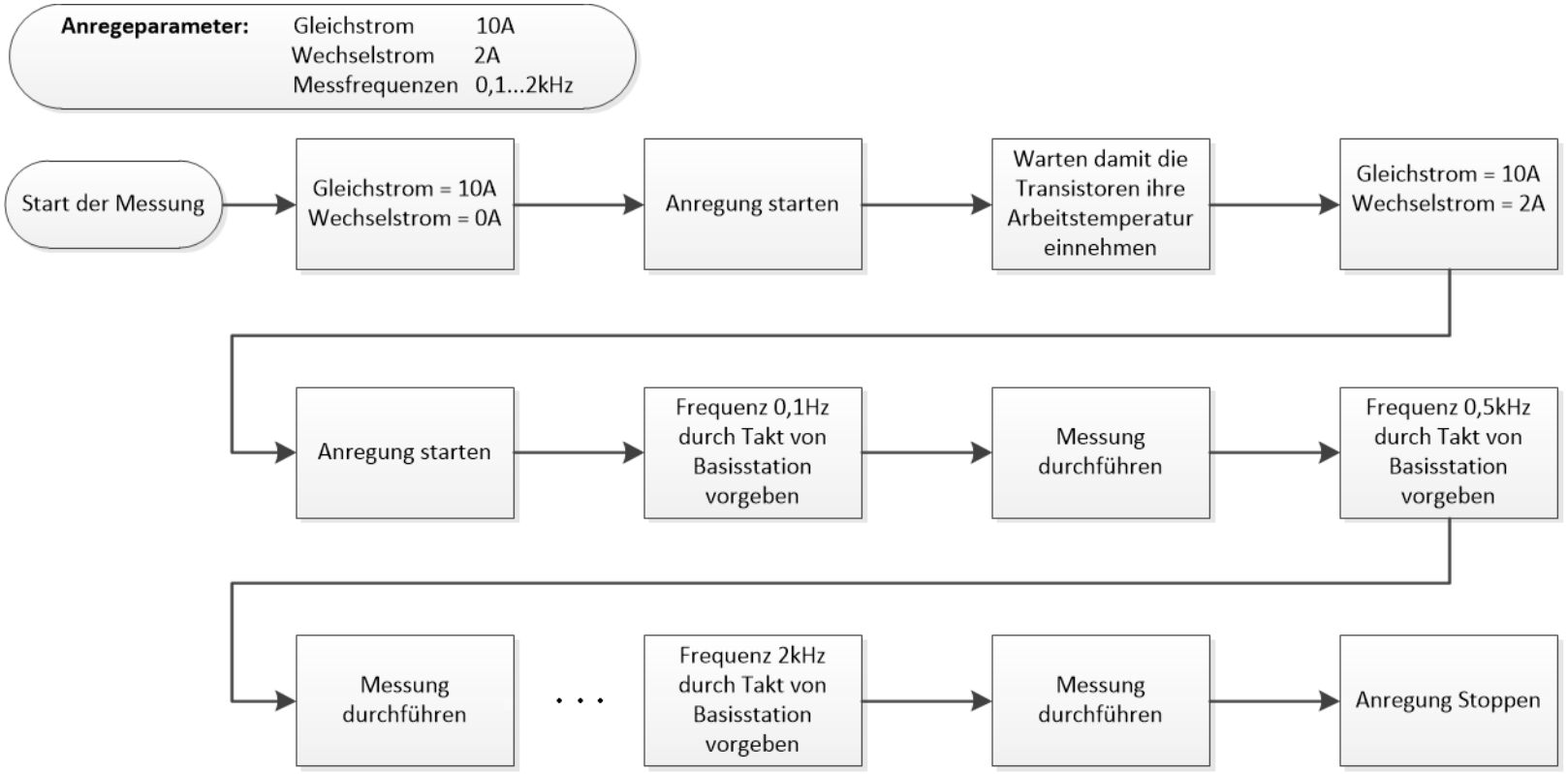

Abbildung 3.13.: Ablauf einer automatisierten Messung am Beispiel der angegebenen Parameter.

4. Erprobung

Um die Eignung des entwickelten Systems zu bewerten und die Qualität des Anregesignals zu beurteilen, werden einige Messungen mit Anregung des Anregesystems durchgeführt. Dabei kommen verschiedene Messmethoden zum Einsatz, die im Folgenden kurz erläutert werden.

4.1. Methoden zur Auswertung des Signalqualität

4.1.1. Spektrum

Das Spektrum eines periodischen Signals zeigt den Frequenzgehalt, aus dem sich das Signal zusammensetzt bzw. die Leistungsdichte der verschiedenen Frequenzanteile des Signals. Zur schnellen digitalen Errechnung des zu messenden Signalspektrums kann die diskrete Fourier-Transformation auf das Signal angewendet werden. Das mittels schneller Fourier Transformation FFT ermittelte Signal des Entladestroms kann mit dem idealen Signal eines Sinus verglichen werden. Das ideale Signal entspricht einem Impuls-Peak bei der Signalfrequenz. Real muss von einem Sinussignal ausgegangen werden, das verzerrt ist und deshalb Oberwellen aufweist. Spektral sind folglich neben dem Signal-Peak weitere Peaks zu erwarten.

4.1.2. Klirrfaktor

Der Klirrfaktor ist ein Maß für die Verzerrung eines Signals [28]. Er beschreibt, welchen Anteil überlagerte Oberwellen an einem sinusförmigen Signal haben. Dabei werden die Effektivwerte angegeben. Der Begriff Klirrfaktor stammt ursprünglich aus der Audiotechnik. Er beschreibt das Geräusch, dass zu hören ist, wenn ein Audiosignal nicht oberwellenfrei ist. Real ist bei der Messung eines sinusförmigen Signals ein Wert zwischen 0 und 1 zu erwarten. Ideal ist ein Klirrfaktor von 0. Er beschreibt einen reinen Sinus, ohne Oberwellen. Besteht ein Signal ausschließlich aus Oberwellen, ergibt sich ein Klirrfaktor von 1. Der Klirrfaktor wird üblicherweise in Prozent, in einigen Fällen aber auch in Dezibel angegeben.

$$k_i = \sqrt{\frac{U_2^2 + U_3^2 + U_4^2 + ...U_n^2}{U_1^2 + U_2^2 + U_3^2 + U_4^2 + ...U_n^2}} = \sqrt{\frac{U_2^2 + U_3^2 + U_4^2 + ...U_n^2}{U}} \leq 1 \qquad (4.1)$$

4.2. Messungen zur Ermittlung der Signalqualität

4.2.1. Maximale Werte

Für die Vollständigkeit der Dokumentation werden die maximalen Ströme sowie die maximale Frequenz zum Software- und Hardwarestand nach der Entwicklung ermittelt.
Der maximale einstellbare Gesamtstrom für die Anregung beträgt 21,38 A. Dabei kann eine maximale Wechselamplitude von 5 A eingestellt werden. Die Anzahl der Stützstellen des Sinus ist im Programm durch eine entsprechende Präpozessordirektive zu 100 definiert. In Kapitel 3.3.3 wird erläutert, wie sich die Stützstellenzahl auf die Frequenzerzeugung auswirkt. Sie ist neben der Handlerlaufzeit des taktgebenden Port-Handlers, einer der limitierenden Faktoren der maximalen Frequenz. Die Folge ist, dass eine Erhöhung des externen Stützstellentakts ab der maximalen Frequenz nicht zu einer weiteren Erhöhung der Anregefrequenz führt. Die Messung ergibt eine maximale Anregefrequenz von etwa 3,12 kHz.

4.2.2. Verzerrungen im unteren Aussteuerbereich

In Kapitel 2.3 wird erläutert, wie sich der Grad der Aussteuerung eines Transistors (Linearität), auf das Ausgangssignal auswirkt. Es wird mit einem *Rohde & Schwarz UPV Audio-Analyzer* überprüft, wie stark der Verzerrungsgrad bei verschiedenen Aussteuerungen ist. Wesentlich hierfür ist die Aussteuerung des Transistors, der die Wechselgröße überträgt. Sein Gleichanteil beträgt maximal 4 A. Da der Gleichanteil-Transistor keine Wechselgrößen überträgt, verursacht dieser auch keine Verzerrungen. Es wird der Wechselstromtransistor zunächst an der unteren Aussteuergrenze und dann auf der oberen Aussteuergrenze ausgesteuert, um den Grad der Verzerrung aufgrund der Kennlinien-Nichtlinearität zu bewerten. Die Messungen werden bei einer Signalfrequenz von 1 kHz, Ausgabetaktfrequenz 100 kHz, durchgeführt. Dabei wird die Spannung, wie in Abbildung 4.1 dargestellt, über dem 470 mΩ-Emitterwiderstand gemessen, während die Batteriespannung durch eine *Philips PE1516* Leistungslaborspannungsquelle zur Verfügung gestellt wird.

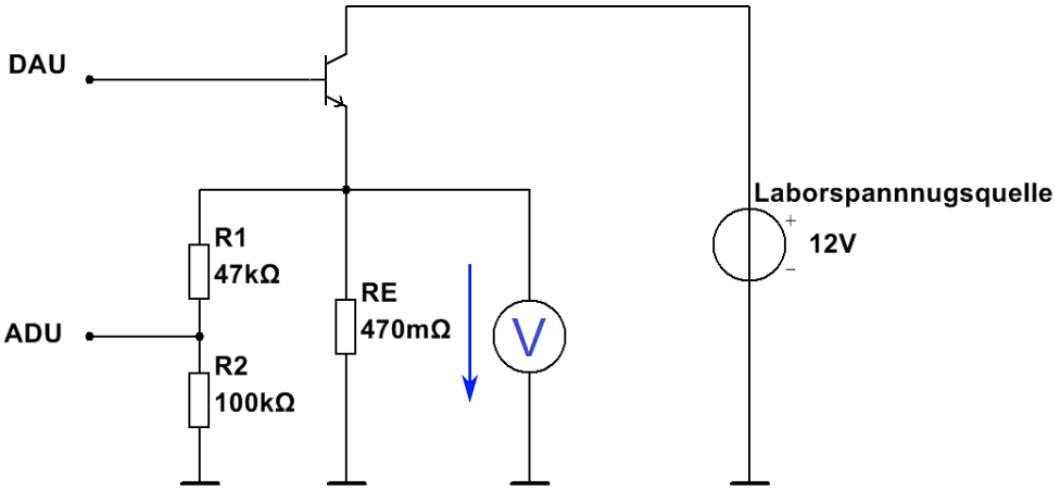

Abbildung 4.1.: Über dem Emitterwiderstand wird die Spannung gemessen. Aus dieser werden Spektren und Klirrfaktoren ermittelt.

Bei der ersten Messreihe wird versucht, eine Aussage über die entstehende Verzerrung aufgrund des unteren Teils der Übertragungskennlinie zu treffen. Dazu wird bei einer Aussteuerung für einen Kollektorstrom von 1 A Gleichanteil der Wechselanteil stetig erhöht.

Begonnen wird mit einer 1 A Wechselamplitude. Das Spektrum in Abbildung 4.3 zeigt deutlich Oberwellen, Harmonische bei ganzen Vielfachen der Signalfrequenz. Der Signalpegel liegt bei -5,76 dBm. Der Pegel der zweiten Harmonischen beträgt -21 dBm. Somit beträgt die Pegeldifferenz 15,24 dB. Der bei diesem Spektrum gemessene Klirrfaktor beträgt 16,4 %.
Bei Erhöhung des Gleichanteils auf 2 A Gleichanteil mit 1 A Wechselamplitude, zeigt das Spektrum weniger stark ausgeprägte Oberwellen. Der Signalpegel beträgt 9,73 dBm und die zweite Harmonische -46,3 dBm. Die Pegeldifferenz beträgt 36,57 dB. Die weiteren Oberwellen verschwinden zunehmend im Rauschen. Der Klirrfaktor beträgt nur noch 1,78 %.
Wird der Gleichanteil auf 3 A erhöht, so zeigt das Spektrum keine Oberwellen, die sich vom Rauschen abheben. Der Signalpegel sinkt auf -21,68 dBm. Die Messung ergibt einen Pegel der ersten Harmonischen von -70 dBm. Die Pegeldifferenz beträgt 48,32 dBm. Die Messung des Klirrfaktors, der das Rauschen nicht berücksichtigt, ergibt 1,07 %.
Bei der letzten Messung der Messreihe mit 4 A Gleichanteil beträgt der Signalpegel -16,77 dBm. Bei einem Pegel der zweiten Harmonischen von -59,7 dBm ergibt sich eine Pegeledifferenz von 42,93 dB. Der Klirrfaktor sinkt auf 0,8 %. Zum Verglich sind die Spektren der Messungen mit 1 A Gleichanteil und mit 4 A Gleichanteil in den Abbildungen 4.2 und 4.3 dargestellt.

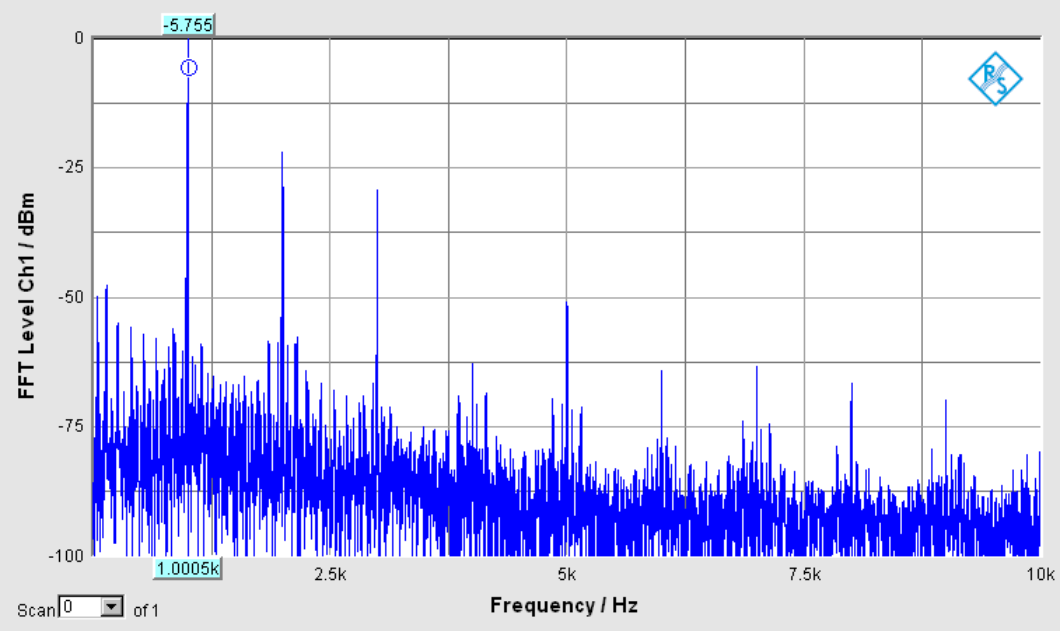

Abbildung 4.2.: Das Spektrum zeigt die Spannung über dem Emitterwiderstand bei 1 A Wechselamplitude mit einem Gleichanteil von 1 A. Oberwellen sind deutlich erkennbar. Das Signal ist stark verzerrt. Es hat einen Klirrfaktor von 16,4 % Der Cursor markiert die Amplitude des Signals in dBm.

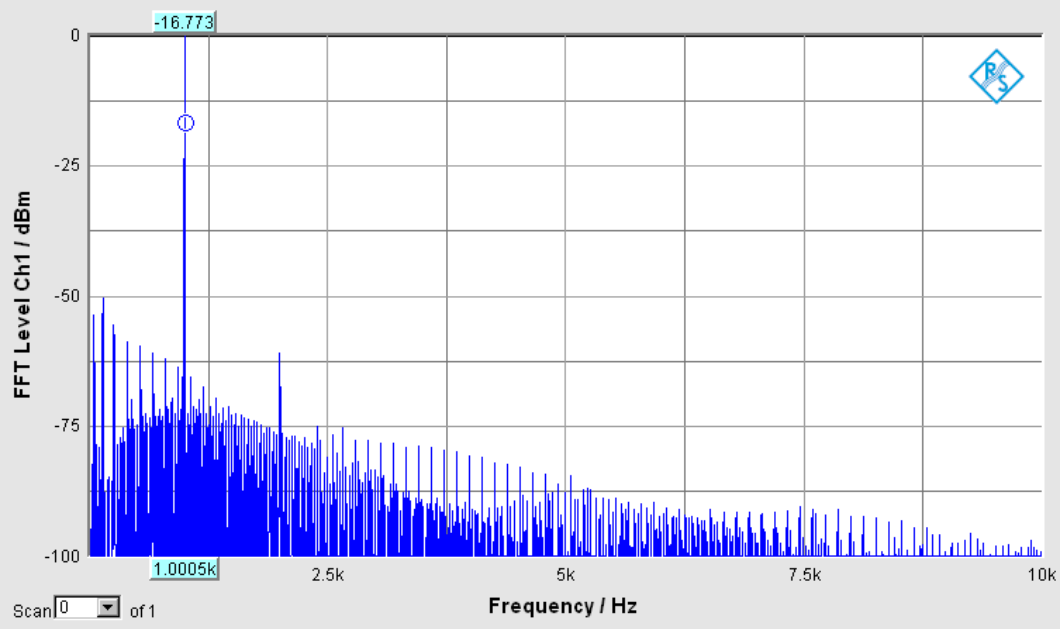

Abbildung 4.3.: Das Spektrum zeigt die Spannung über dem Emitterwiderstand bei 1 A Wechselamplitude mit einem Gleichanteil von 4 A. Oberwellen heben sich nicht vom Rauschen ab. Das Signal ist weitgehend oberwellenfrei bzw. sie beinhalten nur einen sehr geringen Teil an der Signalleistung. Der Klirrfaktor des Signals beträgt 0,8 %. Das Signal ist nur geringfügig verzerrt. Der Cursor markiert die Amplitude des Signals in dBm.

Die Tabelle 4.1, fasst die Ergebnisse dieser Messreihe noch einmal zusammen. Wesentlich ist die Abnahme der Oberwellen bei Erhöhung des Gleichanteils. Dies wird durch die Zunahme der Pegeldifferenz des Signalpegels und der zweiten Harmonischen bestätigt. Einzig die vierte Messung weist eine Abnahme des Signalpegels und der Differenz des Signalpegels zur zweiten Harmonischen auf. Grund ist die in Abbildung 4.3 sichtbare Abnahme des Rauschens. Die Aussteuerung des Transistors wird erhöht. Resultat ist eine Abnahme des Klirrfaktors. Die Verzerrung des Signals nimmt ab. Der Arbeitspunkt des Transistors wird zunehmend in die Mitte der Kennlinie verschoben und entfernt sich aus dem nichlinearen Bereich.
Eine Aussteuerung mit 3 A Gleichanteil abzüglich Amplitude ist deshalb der Mindestspannungswert, wenn Verzerrungen vermieden werden sollen.

Tabelle 4.1.: Zusammenfassung der Messreihe des unteren Aussteuerbereichs. Dabei beträgt die Wechselstromamplitude immer 1 A.

Gleichanteil	Signalpegel	Abstand Pegel,1. Oberwelle	Klirrfaktor
1A	-5,78dBm	15,24dB	16,40%
2A	-9,73dBm	36,57dB	1,78%
3A	-21,5dBm	48,32dB	1,06%
4A	-16,7dBm	42,93dB	0,80%

4.2.3. Verzerrungen im oberen Aussteuerbereich

Es soll festgestellt werden, ab welchem Entladestrom sich eine deutliche Verzerrung aufgrund der Begrenzung des Transistors einstellt. Wie bereits in Kapitel 3 erläutert, wird der Wechselspannungstransistor nur bis 4 A Gleichstrom ausgesteuert. Der restliche Strom wird vom Gleichstromtransistor übernommen. Deshalb wird der Gleichanteil für den Wechseltransistor auf den Maximalwert 4 A gestellt. Nun wird die Aussteuerung der Amplitude nach oben durchgeführt.

Bei einer Wechselstromamplitude von 2 A hat das gemessene Signal einen Klirrfaktor von 2,16 %. Nach Erhöhung der Amplitude auf 3 A, liegt der Klirrfaktor des Signals bei 2,32 %. Bei der Maximalaussteuerung des Transistors von 4 A Wechselstromamplitude bei 4 A Gleichstrom liegt der Klirrfaktor bei 2,6 %.
Tabelle 4.2 fasst die Ergebnisse der Messung zusammen. Die Vergrößerung der Aussteuerung bringt die Amplitude an ein höheres Maximum, aber auch an ein geringeres Minimum. Aufgrund der Messungen in Kapitel 4.2.2, ist deshalb davon auszugehen, dass die ansteigende Verzerrung von der Aussteuerung im unteren Bereich herrührt. Da die Amplitude softwareseitig nicht größer als Gleichanteil eingestellt werden kann, ist es folglich auch bei maximaler Aussteuerung 8 A nicht möglich, den Transistor in Sättigung zu fahren. Es entstehen bei Aussteuerung bis zu 8 A also keine wesentlichen Verzerrungen durch den Eintritt in Sättigung.

Tabelle 4.2.: Zusammenfassung der Messreihe des oberen Aussteuerbereichs. Dabei beträgt die Gleichstromamplitude immer 4 A.

Wechselanteil	Klirrfaktor
2A	2,16%
3A	2,32%
4A	2,60%

4.2.4. Einflüsse der Frequenz

Um die Linearitätseigenschaften der Transistoren im Betrieb zu ermitteln, werden die Oberwellen des Entladestroms bei veränderlicher Frequenz betrachtet. Nimmt die Anregefrequenz ab, so steigt die Zeit zwischen den Ausgabeänderungen des DA-Umsetzers. Die Folge ist eine größere Stufenbreite zwischen den Sinus-Stützstellen, die zu einer Änderung an Oberwellen führen könnte. Zu prüfen ist in diesem Zusammenhang, ob diese Änderung erkennbar ist und sich die Oberwellen verstärken oder verringern, wenn die Frequenz reduziert wird. Gegen eine Veränderung durch hochfrequente Anteile spricht die Trägheit, die Batterien aufweisen. Möglicherweise werden hochfrequente Stromänderungen heraus geglättet. Bei der Messung wird ein Gleichanteil von 4 A eingestellt. Dieser legt den Arbeitspunkt in einen wenig verzerrenden Bereich (siehe Kapitel 2.3.1). Die Wechselstromamplitude wird gegenüber der vorausgegangenen Messungen auf 3 A gestellt, damit sich die Signalpegel deutlich vom Rauschen abheben.

Bei 1 kHz ergibt sich ein Klirrfaktor von 2,2 %. Bei 100 Hz beträgt der Klirrfaktor ebenfalls 2,2 %. Bei 10 Hz beträgt der Klirrfaktor 2,5 %.
Bei niedriger Frequenz erhöht sich der Klirrfaktor geringfügig. Die Verzerrung des Signals hat aber nach wie vor ein Maß, welches ein ausreichen unverzerrtes Signal für die EIS-Messung bildet. Die Frequenz, bei der die Anregung stattfindet ist folglich nicht wesentlich für die Qualität der Anregung.

4.3. Durchführung einer EIS-Messung

Nachdem bestätigt wurde, dass die Anregung grundsätzlich für eine EIS-Messung geeignet ist, kann eine Messung mit Batteriesteuergerät, Zellensensoren und Anregesystem durchgeführt werden. Dabei soll die Frage beantwortet werden, ob eine Messung mit dem entwickelten System, zu einem befriedigendem Ergebnis der EIS führt. Für die Messung werden 4 ANR-Zellen [7] in Reihe geschaltet. Es wird eine Messreihe mit Frequenzen von 100 mHz bis 2 kHz durchgeführt. Tabelle 4.3 zeigt die Frequenzen, bei denen die Zellen angeregt wurden. Dabei nimmt jeweils ein Zellensensor an jeder der vier Zellen die Spannungswerte auf. Das Batteriesteuergerät misst den Strom. Die Zellen werden mit 2 A Gleichanteil und 1 A Wechselanteil entladen.

Tabelle 4.3.: Anregefrequenzen der EIS-Messung mit dem Gesamtsystem.

0,1Hz	0,5Hz	1Hz	5Hz	10Hz	25Hz	50Hz	100Hz	250Hz	500Hz	750Hz	1kHz	2Hz

Nach der Messung werden die Messergebnisse mittels Matlab-Skript verarbeitet, um auf den Nyquist-Plot sowie das zugehörige Spektrum zu plotten. Im Folgenden werden die Ergebnisse der Messung einer der vier Zellensensoren dargestellt.

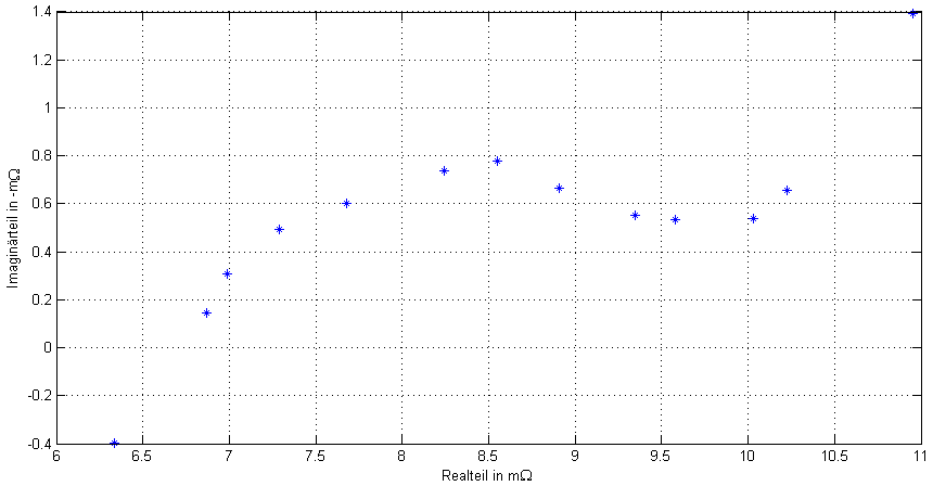

Abbildung 4.4.: Ergebnis der EIS-Messung mittels Gesamtsystem im Nyquist-Plot. Der Graph zeigt die erwartete Ast-Form der Batteriezellenimpedanz.

Das Messergebnis des Nyquist-Plots unterscheidet sich quantitativ von Messungen in [1]. Dies begründet sich auf der Tatsache, dass sich eine gemessene Batterie nach jeder Messung in einem anderem Zustand befindet. Im Ruhezustand weist sie geringe Selbstentladung auf. Bei einer EIS-Messung wird sie entladen, ihr Ladezustand verändert sich folglich recht signifikant. Die Belastung der Messung hat auch einen geringen Einfluss auf den SoH. Diese Faktoren, besonders aber der veränderte Ladezustand, führen dazu, dass Messungen nie identisch sind. Es ist weiterhin zu beachten, dass die Batteriezellen Einfluss auf die Signalqualität und das Rauschen haben. Der Nyquist-Plot der Zellenimpedanz zeigt allerdings qualitativ die typische erwartete Ast-Form. Die Werte für die reellen und die komplexen Widerstände liegen im zu erwartenden Milliohm-Bereich. Die Messung ist folglich gültig und aussagekräftig.
Mit den aufgenommenen Daten der gemessenen Signale kann auch deren Spektrum ermittelt werden. Abbildung 4.5 zeigt das Spektrum des gemessenen Signals an einem der Zellensensoren bei 500 Hz Signalfrequenz. Es bestätigt die Messungen des Spektrums an der Laborquelle. Der Signal-Oberwellen-Abstand ist so groß, das der Signalpeak deutlich überwiegt.

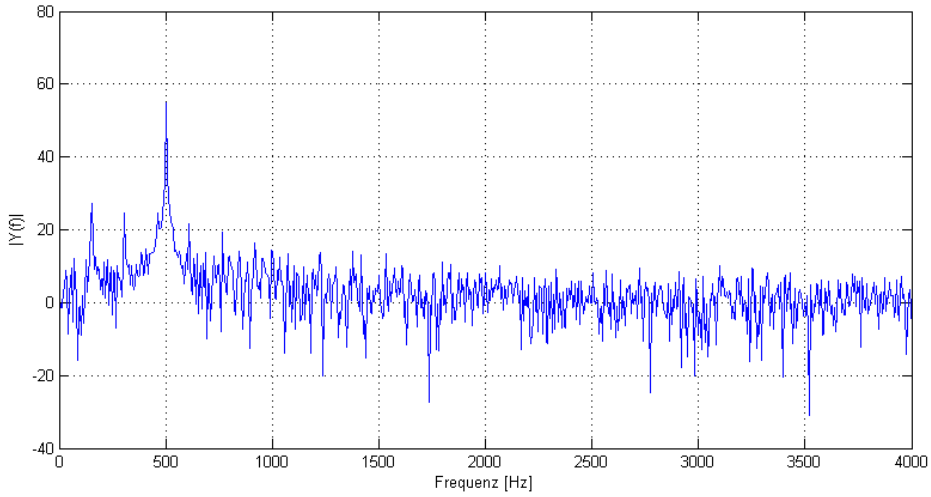

Abbildung 4.5.: Das sich ergebende Spektrum des Signals mit 500 Hz, aufgenommen durch eine EIS-Messung.

5. Bewertung, Ausblick und Fazit

5.1. Zusammenfassung und Bewertung

In der vorliegenden Arbeit wurde ein System entwickelt, das in der Lage ist, eine Batterie mittels eines sinusförmigen Entladestroms zu enltladen (anzuregen). Das System ist Teil eines Messsystems für die Messmethode der elektrochemischen Impedanzspektroskopie (EIS), welches im Batterieprojekt „Drahtlose Zellensensoren für Fahrzeugbatterien" definiert wurde [2]. Das System kann den notwendigen Gleich- und Wechselanteil sowie die benötigten Frequenzen des Entladestroms für eine Messung erzeugen, die für kleine und große, im Labor verwendete Batterien benötigt werden. Für die entwickelte Schaltung des Systems wurden Transistortypen und Schaltungsvarianten gegenübergestellt, um die für die Anforderungen beste Schaltungsvariante zu ermitteln. Die geschriebene Software zur Ansteuerung der Schaltung ist auf einem Mikrocontroller umgesetzt und gibt die Ausgabewerte für den Strom mittels Digital-Analog-Umsetzer an die Ansteuerschaltung, die den gewünschten Strom mittels Leistungstransistoren erzeugt.

Nach erfolgreicher Entwicklung der Schaltung im Labor, wurde eine Platine entworfen, auf der die Ansteuerelektronik zusammengefasst ist. Der Kern der Schaltung sind zwei Leistungstransistoren. An dem Gesamtaufbau mit Platine wurden anschließend Messungen zur Bewertung der Signalqualität durchgeführt. Diese haben ergeben, dass die Signalqualität der erzeugten Signale, abhängig von der Transistoraussteuerung variieren, jedoch im Großteil des Aussteuerbereichs, für die EIS-Messung geeignet sind. Die Schaltung ist in der Lage, die Anregung mit Strömen von 2 A Gleichanteil und 1 A Wechselanteil für kleine Batterien mit einem Innenwiderstand von etwa 6 mΩ bei 1 kHz ausreichend unverzerrt durchzuführen. Sie kann aber auch die Anregung für größere Zellen mit einem Innenwiderstand von 1 mΩ bei 1 kHz mit Strömen von 10 A Gleichanteil und 2 A Wechselanteil durchführen. Nach der Bewertung der Signalqualität wurde eine EIS-Messung mit Anregung durch das entwickelte System durchgeführt. Das Ergebnis der Messung entsprach dem typischen Ergebnis solch einer Messung.

Die EIS-Messung mit Anregung durch eine Transistorschaltung wie der entwickelten ist möglich. Um das Messsystem autonom einsetzbar zu machen, muss das entwickelte Anregesystem in das Gesamtsystem integriert werden. Es bleiben zudem noch viele Optionen zur Verbesserungen des bestehenden Systems.

5.2. Ausblick

Die Entwicklung der Anregeschaltung hat gezeigt, dass die Realisierung einer solchen Schaltung mit dem gewählten Ansatz möglich ist. Es gibt in Folge der Ergebnisse der Entwicklung und der Auswertung einige Punkte, die bedeutsamen Einfluss auf die Weiterentwicklung des Anrege- und Messsystems haben könnten.

Verbesserung des Sinussignals durch Anpassung der Stützstellenzahl

Das erzeugte Sinussignal hat 100 Stützstellen, das heißt 100 Ausgabewerte pro Periode. Dieser Zahl an Stützstellen ermöglicht eine maximale Anregefrequenz von knapp über 3 kHz. Die Laufzeit des Handlers zur Ausgabe der Werte ist hierbei der limitierende Faktor. Bei geringen Frequenzen ist es möglich die Zahl der Stützstellen zu erhöhen und dadurch auch die Stufenhöhe der Ausgabe zu verringern. Dies würde zu verbesserten spektralen Eigenschaften des Sinussignals führen. Soll immer die höchstmögliche Zahl an Stützstellen für die verwendete Anregefrequenz verwendet werden, so muss eine Möglichkeit implementiert werden, durch die die Zahl der Stützstellen dynamisch, bei Änderung der Frequenz mit geändert wird. Eine andere Möglichkeit wäre eine vorberechnete Sinustabelle mit verschieden langen Sinus-Vektoren, die je nach Anregefrequenz, abgerufen werden.

5.2.1. Verbesserung des Sinussignals durch softwareseitige Vorverzerrung

Da das Eingangssignal des Transistors zuvor digital erzeugt wird, ist es möglich, die auftretenden Verzerrungen bereits vor der Übertragung über den Transistor kompensieren. Dazu könnte unter Einbeziehung des Übertragungsverhaltens des Transistors, das gewünschte Ausgangssignal softwareseitig so verzerrt werden, dass sich nach der Übertragung, über den Transistor, die gewünschte Signalform ergibt. Es müsste vor der Messung bei verschiedenen Ansteuerwerten, beispielsweise bei jedem DAU-Bit-Wert, der Ausgangsstrom gemessen werden, um die Gesamtkennlinie eines Transistors in Form einer Abhängigkeit von Ausgangs- zu Eingangsgröße aufzunehmen. Wird die Umkehrfunktion der Kennlinie nun mit dem Eingangssignal multipliziert, so ergibt sich ein vorverzerrtes Signal, welches durch das Übertragungsverhalten der Anregeschaltung ein sauberes Sinussignal am Ausgang hervorruft. Um dies mathematisch umzusetzen, muss ein Algorithmus entwickelt werden, der in der Lage ist, aus aufgenommenen Datenwerten eine Funktion Ausgangswerte (Strom) in Abhängigkeit von Eingangswerten (Bit-Werte) zu approximieren. Je besser die Approximation einer solchen Funktion, desto weniger Verzerrungen im Ausgangssignal.

Um die gesuchte Umkehrfunktion zu erhalten, muss aus den gemessenen Werten eine Funktion gebildet werden. Aus dieser lässt sich dann die Umkehrfunktion bilden, indem X-Werte und Y-Werte vertauscht werden. Um die Übertragungsfunktion zu erhalten, können verschiedene Methoden angewandt werden. Zu den simpelsten Möglichkeiten gehört die Approximation mittels Taylorreihenentwicklung. Mit ihr lässt sich eine Funktion $f(x)$ um einen Arbeitspunkt x_0 approximieren. Unter der Voraussetzung „$f(x)$ ist in der Umgebung von x_0 beliebig oft differenzierbar und das Restglied $R_n(x)$ in der Taylorschen Formel verschwindet für $n \to \infty$" gilt

$$f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{5.1}$$

Dabei gilt, je größer n, desto genauer die Approximation. Das Ergebnis bildet approximiert die gesuchte Funktion, die dann noch invertiert auf das Ausgangssignal multipliziert werden muss [29].
Auch wenn diese Methode der Vorverzerrung theoretisch zu einer starken Verbesserung der Ausgangsverzerrung führen sollte, ist unklar, wie gut sich eine derartige Methode eignet, denn Verzerrungen treten nicht nur aufgrund des Übertragungsverhaltens der Transistoren und der weiteren Elemente des Ansteuerpfads auf. Auch das zuvor erwähnte Temperaturverhalten der Transistoren, die Änderung der Temperatur der Transistoren, führt zu einer veränderlichen Übertragungsfunktion) und womöglich auch anderer Bauteile sowie mögliche Einstreuungen von außen usw. können Einfluss auf das Übertragungsverhalten haben.

Theoretisch müsste dazu die Übertragungsfunktion des Transistors $|h(t)|$ ermittelt und diese mit dem Eingangssignal $e(t)$ verrechnet werden. Dabei beinhaltet $h(t)$ in dieser Beschreibung nur das nichtlineare Verhalten, aber nicht die Verstärkung und die Phasenverschiebung. Es würde sich ein neues Eingangssignal $u(t)$ ergeben, das der Umkehrfunktion $h(t)^{-1}$ entsprechen und durch Übertragung über den Transistor wiederum $e(t)$ ergeben würde (unter Einbeziehung einer Verstärkung G und unter Vernachlässigung einer Phasenverschiebung).

$$Y(s) = \frac{U(s)}{|H(s)|} \tag{5.2}$$

Wird U(s) nun auf den Transistor gegeben, so ergibt sich an seinem Ausgang

$$Y(s) \cdot H(s) = U(s) \cdot G \ , \tag{5.3}$$

wobei G der Verstärkung entspricht.

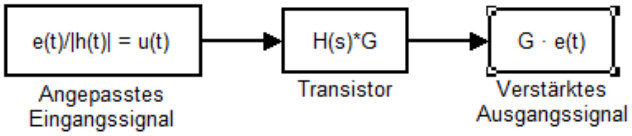

Abbildung 5.1.: Übertragungsstrecke – Linearisiertes Eingangssignal auf den Transistor. Am Ausgang entsteht das gewünschte verzerrungsfreie und verstärkte Eingangssignal.

Das Verfahren kann recht aufwendig werden, da die Übertragungsfunktion des verwendeten Transistors genau ermittelt werden muss. Wird dieser Ansatz praktisch umgesetzt, so kann der Einfluss der Nichtlinearität kompensiert und dieses Kriterium zur Auswahl des Transistortyps ausgeschlossen werden. Es ermöglicht, das digital erzeugte Signal verzerrungsfrei am Transistorausgang zu erhalten.

5.2.2. Verbesserung des Sinussignals durch Regelung

Die Auswertung des erzeugten Signals in Kapitel 4 hat gezeigt, dass die „Reinheit" des Sinus, die Oberwellen im Spektrum des erzeugten Signals, messbar sind. Erzeugte Signale enthalten, je nach Aussteuerung stark ausgeprägte Oberwellen. Zwar reicht der Pegelabstand zwischen Signalpegel und Pegel der ersten und weiterer Oberwellen für EIS-Messungen, es ist jedoch möglich, diese Pegeldifferenz durch weitere Reduzierung der Verzerrungen zu erhöhen. Unter Vernachlässigung der Verzerrung durch die digitale Erzeugung (Treppenstufen des DA-Umsetzers) rührt die Verzerrung in erster Linie daher, dass die Übertragungsfunktion des Hardwareteils, genauer der Transistoren, nichtlinear ist (siehe Kapitel 2.3). Um die Nichtlinearität der Transistoren zu umgehen, ist es möglich, eine Regelung in die Anregesoftware zu integrieren, die in der Lage ist, während der Anregung den aktuellen Ausgangsstromwert so gegenzuregeln, dass dieser den Sollwert des nächsten Sinuspunktes annimmt. Diese Form der Regelung ist theoretisch möglich, trifft aber praktisch auf einige Hindernisse. Diese Art der Regelung muss im Gegensatz zur Signalfrequenz verhältnismäßig schnell im sein, denn es muss mehrfach pro Treppenstufe nachgeregelt werden (siehe Abbildung 5.2). Die Zeit, die dabei für das Messen, das Regeln und zur Korrekturausgabe vorhanden

wäre, müsste kleiner sein als die Dauer zwischen zwei Abtastpunkten. Die Wahl der Abtastfrequenz für die Regelungsabtastung ist dabei abhängig von der Breite des in Software generierten Sinus multipliziert mit einer maximalen Signalfrequenz, aus der sich die Ausgaberate (blaue Markierungen in Abbildung 5.2) des DA-Umsetzers ergibt. Ergebnis dieser Softwareerweiterung ist ein saubereres Sinussignal, welches zu einem besseren Messergebnis führt.

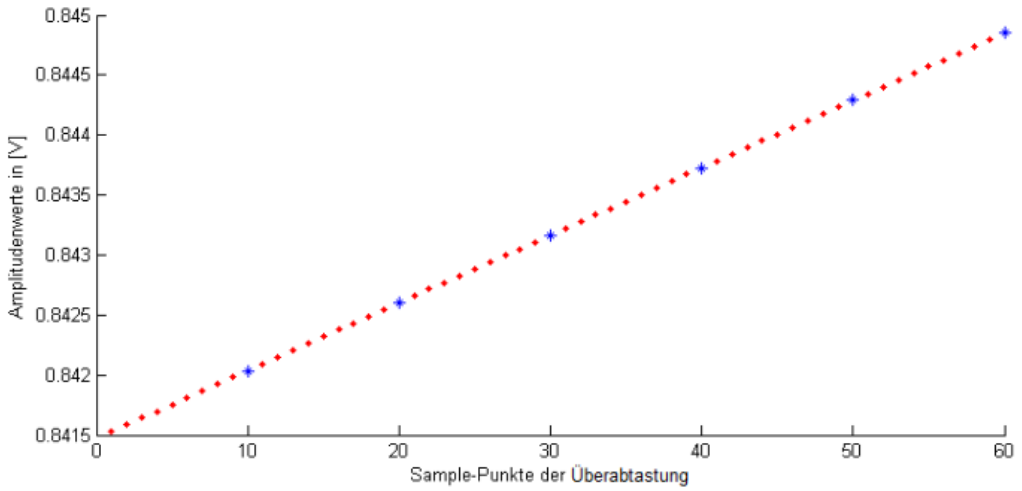

Abbildung 5.2.: Das Prinzip der Überabtastung zur Regelung des erzeugten Signals. Der Kurvenverlauf zeigt einen Ausschnitt des Anstiegs eines Sinussignals. In blau sind die Punkte, an denen der DA-Umsetzer über die Transistoren den Wert verändert. In rot sind die Punkte, an denen der fließende Strom im Hauptstrang gemessen werden könnte. Hier ist das Verhältnis zehn Messungen pro einer Änderung.

5.2.3. Verwendung weiterer Transistoren bei synchroner Aussteuerung

In Kapitel 2.3.3 wird begründet, weshalb die Aussteuerung in Gleich- und Wechselanteil unterteilt wird. Grund hierfür ist die Tatsache, dass dies bei zwei Transistoren des gewählten Typs erlaubt, die maximale Linearität für den Wechselanteil zu nutzen und trotzdem einen Gesamtstrom von 12 A und mehr zu erreichen. Die Lösung ist mit zwei Transistoren durchaus sinnvoll. Bei genauer Untersuchung des im Labor vorhandenen EIS-Meters *TrueData-EIS*

von FuelCon [30], welches als Referenz für die Ergebnisse der EIS-Messungen der Zellensensoren verwendet wird, kann festgestellt werden, dass für die EIS-Anregung drei Transistoren verwendet werden. Zwar ist nicht klar, wie diese verschaltet sind, es führt allerdings zum Lösungsansatz der Anregung mit drei oder mehr parallel geschalteten Transistoren. Dies hat bei synchroner Ansteuerung folgende Vorteile:

- Geringere Verlustleistung an den einzelnen Transistoren, da sich diese auf alle Transistoren etwa gleich verteilt.
- Eine Änderung des Gleichanteils verändert den Arbeitspunkt aller Transistoren geringfügiger, da die Verschiebung auf alle Transistoren aufgeteilt wird. Das gleiche gilt für die Amplitudenaussteuerung. Das bedeutet, dass sich der lineare Bereich der Kennlinien mit der Anzahl der verwendeten Transistoren verlängert.
- Durch synchrone Aussteuerung können alle Transistoren von einem steuernden Kanal betrieben werden, sofern die Summe der benötigten Basisströme zur Verfügung gestellt werden kann.
- Wird die Anforderung an den maximalen Entladestrom erhöht, so können weitere Transistoren parallel kaskadiert werden.
- Das Einlesen des fließenden Stroms zur Einstellung oder Regelung kann über die Spannungsdifferenz eines gemeinsamen Kollektorwiderstands, eines Differenzverstärkers und eines AD-Umsetzers geschehen.

Nachteil bei der Verwendung weiterer Transistoren und synchroner Aussteuerung könnte lediglich die Anforderung an die Ansteuerung für die Basis sein, den nötigen Basisstrom für alle Transistoren zu liefern. Allerdings verringert sich der Basisstrom, wenn der Kollektorstrom reduziert wird.

5.2.4. Automatische Anpassung der Anregung an die aktuelle Zellenspannung

Das Anregesystem im bestehenden Zustand ist so ausgelegt, dass durch das Batteriesteuergerät ermöglicht werden kann, durch einen Befehl des Benutzers, eine komplette EIS-Messung durchzuführen. Dabei werden die veränderlichen Größen der Messung, je nach eingegebenem Batterietyp, angepasst. Diese Änderungen sind jedoch statisch, das heißt die Einstellungen sind für einen jeweiligen Batterietypen fest eingestellt. Der Innenwiderstand einer Batterie ist allerdings veränderlich, wodurch sich je nach SoC auch der Spannungsabfall über diesem verändert. Eine Erweiterung im Zuge des Anregesystems wäre es, die statischen Größen, die den Strom betreffen, dahingehend dynamisch zu gestalten, dass die Größe des Anregestroms, Gleich- sowie Wechselanteil, sich wie in Kapitel 2.1.2, an dem

notwendigem Spannungsabfall am Innenwiderstand der Zelle einstellt. Das würde bedeuten, dass der Entladestrom am Anfang einer Messung, je nach gemessener Klemmspannung, für den notwendigen Spannungsabfall des Zellentyps an den Innenwiderstand angepasst würde. Ziel wäre ein genaueres Messergebnis bei jedem SoC der Batterie.

5.2.5. Zusammenfassen von Anrege- und Steuersoftware auf einem Mikrocontroller

In Kapitel 2.2 wird begründet, warum zur Entwicklung des Prototyps eines Ansteuersystems, neben dem Mikrocontroller für das Batteriesteuergerät, ein zweiter Mikrocontroller verwendet wird. Um das Gesamtsystem kompakt zu halten, damit es problemlos Anwendung in Kraftfahrzeugen finden kann, ist es naheliegend, die bestehende Software beider Controller auf einem zu realisieren. Die bereits erwähnten Schwierigkeiten bezüglich der Interrupt-Prioritäten erfordern es, einen Überblick über die bestehende Software und die kritischen zeitlichen Abläufe der Systeme zu schaffen, um daraus eine entsprechende Lösung zu entwickeln.

5.2.6. Erfassen des Energieverbrauchs des Messsystems

In dieser Arbeit liegt der Kern auf der Verifikation der Funktionalität der Anregung. Der Energiebedarf des Systems wird vernachlässigt. Soll das Gesamtsystem autonom Anwendung finden, so ist es unerlässlich, den Energiebedarf der Anregeschaltung näher zu betrachten. Er sollte, besonders wenn keine Messungen durchgeführt werden, möglichst gering sein, um die versorgende Batterie so wenig wie möglich zu belasten. Eine Abschaltfunktion für die Steuerelektronik sowie gegebenenfalls verwendete Lüfter, sollte diskutiert werden. Auch sollte eine ausführliche Dokumentation des Energiebedarfs eines weiterentwickelten Anregesystems erfolgen.

5.2.7. Erweiterung des Systems durch Live-Tracking während der Messung

Das entwickelte Gesamtsystem soll als Prototyp auch für Vorführungen der Funktionalität einer kompakten, mobilen und kabellosen EIS-Messung dienen. Deshalb wird ein funktionierendes Gesamtsystem interessierten Vertretern aus Wirtschaft und Forschung präsentiert werden. In diesem Zusammenhang wäre eine Weiterentwicklung interessant, die nicht nur den genauen Fortschritt einer Messung dokumentiert, sondern dem interessiertem Zuschauer veranschaulicht, was während des Messprozesses an Ergebnissen generiert wird.

Das Gesamtsystem ist bereits in der Lage, über USB-Schnittstellen und Batterie versorgt zu werden. Da eine Verbindung zu einem PC im Falle einer Vorführung ohnehin besteht, wäre es möglich, die Ergebnisse der EIS-Messung, die in Form von komplexen Zahlen übermittelt werden, zum Beispiel über MATLAB, live in einem Plot darzustellen. So würde sich während der Messung Stück für Stück der charakteristische Nyquist-Plot der EIS aufbauen.

5.2.8. Analyse der Anregung mit anderen Signalformen

Bisher wurde die EIS-Messung mit Sinussignalform durchgeführt. Bei ihr ist das ideale Spektrum und die spektrale Verteilung der Signalleistung verständlich. Weitere Möglichkeiten zur Anregung bieten beliebige andere periodische Signale. Für die praktische Nutzung eines Systems zur EIS-Messung kann es unter Umständen ungünstig sein, dass ein Messsystem große Kühlkörper und aktive Lüftung benötigt. Es wäre von Vorteil, wenn platzraubende Hardware vermieden werden könnte. Im Zuge dessen würde es sich anbieten, bereits bestehende Systeme in Kraftfahrzeugen für eine passive Anregung zu verwenden. Interessant scheint besonders das Rechtecksignal. Diese Art der Anregung ist wesentlich einfacher zu realisieren und kann unter Umständen durch bestehende Systeme durchgeführt werden. Als Beispiel könnte hier die in einigen Fahrzeugtypen bestehende Heckscheibenheizung als Stromsenke dienen. Wesentlich für diese Art der Lösung der Anregung ist die Verwendbarkeit eines Rechtecksignals oder anderer periodischer Signale für die EIS-Messung. Im Falle des Rechtecksignals hat dieses ein Spektrum in Form einer si-Funktion. Wesentliche spektrale Anteile bestehen also in mehreren Frequenzen. Es ist zu klären, ob sich das Signal trotzdem eignet, um in EIS-Messungen aussagekräftige Ergebnisse zu erhalten. Sollte dies möglich sein, so könnten sich die Eigenschaften des Signals zu Nutze gemacht werden. Es könnte möglich sein, mehrere der auftretenden Frequenzen gleichzeitig zu nutzen, um Mehrere Impedanzwerte bei einer Rechteckfrequenz zu ermitteln. In diesem Falle lässt sich die Gesamtmessdauer möglicherweise reduzieren.

5.3. Fazit

Die Bedeutung von Batterien im Bereich von Kraftfahrzeugen nimmt stetig zu. Steigende Anforderungen durch aufwendigere Elektronik und steigende Popularität der Elektromobilität erfordern nicht nur zunehmend leistungsfähige Batterien, sondern auch Batterie-Management-Systeme, die in der Lage sind, diese zu überwachen. Das Projekt „Drahlose Zellensensoren für Autobatterien”, entwickelt ein System, dass als Teil eines Batterie-Management-Systems in der Lage ist, die elektrochemische Impedanzspektroskopie an Batterien durchzuführen. Dazu muss das System Strom und Spannung an dem Prüfling, bei verschiedenen Frequenzen messen. Das System besteht aus zwei elementaren Bestandteilen. Dem Messsystem

und der Anregung. Das Messsystem wurde bereits erfolgreich entwickelt. Mit dieser Arbeit wurde ein erster Schritt getan, die Anregung zu entwickeln. Das Prinzip und ihre Funktionalität konnten getestet und bestätigt werden. Erste Messungen mit dem entwickelten Anregesystem und dem Messsystem ergaben vielversprechende Ergebnisse. Für die Durchführung einer autonomen elektrochemischen Impedanzspektroskopie haben sich in dieser Arbeit viele Punkte zur Verbesserung und Weiterentwicklung ergeben. Das entwickelte System kann dabei als Ausgangspunkt dienen.

Literaturverzeichnis

[1] Nico Sassano. *Entwicklung eines Messsystems zur funksynchronisierten elektrochemischen Impedanzspektroskopie an Batterie-Zellen*. Master Thesis - HAW Hamburg, 2015.

[2] Karl-Ragmar Riemschneider Nico Sassano, Valentin Roscher. *Automobil-Sensorik - Ausgewählte Sensorprinzipien und deren automobile Anwendung*. Thomas Tille, 2016.

[3] European Association of Research and Technology Organisations EARTO. The TRL Scale as a Research and Innovation Policy Tool, EARTO Recommendations. http://www.earto.eu/fileadmin/content/03_Publications/The_TRL_Scale_as_a_R_I_Policy_Tool_-_EARTO_Recommendations_-_Final.pdf.

[4] Phillip Durdaut. *Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz*. Bachelor Thesis - HAW Hamburg, 2013.

[5] Martin Kiel. *Impedanzspektroskopie an Batterien unter besonderer Berücksichtigung von Batteriesensoren für den Feldeinsatz - 1. Auflage*. Shaker, Juni 2013.

[6] Matthias Schöllmann. *Energiemanagement und Bordnetze - 1. Auflage*. Expert, 2005.

[7] A123-Systems. Datenblatt: ANR26650 - MD100113-01. http://www.a123systems.com/Collateral/Documents/English-US/A123%20Systems%20ANR26650%20Data%20Sheet.pdf.

[8] ECC-Repenning-GmbH. Datenblatt: LFPP 45Ah. http://www.eccbatteries.com/files/kunden-_technische_daten_ecc_batteries_gmbh.pdf.

[9] Eberhard Gamm Ulrich Tietze, Christoph Schenk. *Halbleiter-Schaltungstechnik - 14. Auflage*. Springer, 2012.

[10] J. Ross Macdonald Evgenij Barsoukov. *Impedance Spectroscopy: Theory, Experiment, and Applications - 2nd Edition*. Wiley, Apri 2005.

[11] Alexander Angold. *Verfahren zur Aufwandsreduzierten Elektrochemischen Impedanzspektroskopie für Starterbatterien*. Bachelor Thesis - HAW Hamburg, 2014.

[12] FH-Strahlsund Antriebstechnik. Der Bipolartransistor. http://antriebstechnik.fh-stralsund.de/1024x768/Dokumentenframe/Kompendium/Fachvorlesungen/Halbleiter/Bauelemente-40.pdf.

[13] Iris Strassacker. lautsprechershop.de - elektronenröhren. http://www.lautsprechershop.de/index_hifi_de.htm.

[14] Fairchild Semiconductors. Datenblatt: BC547 - Rev. 1.1.1. https://www.fairchildsemi.com/datasheets/BC/BC547.pdf.

[15] International-Rectifier. Datenblatt: IRFR/U024N - April 1998. http://www.irf.com/product-info/datasheets/data/irfr024n.pdf.

[16] On-Semiconductor. Datenblatt: 2N3055 - Rev. 6, Dezember 2005. http://www.onsemi.com/pub_link/Collateral/2N3055-D.PDF.

[17] Stefan Goßner. *Halbleiter, Bauelemente und Schaltungen - 8. Auflage*. Shaker Verlag, 2011.

[18] On-Semiconductor. Datenblatt: MJ15003 Rev. 16, September, 2013. http://www.onsemi.com/pub_link/Collateral/MJ15003-D.PDF.

[19] On-Semiconductor. Datenblatt: MJ11012 Rev. 5, September 2008. http://www.onsemi.com/pub_link/Collateral/MJ11012-D.PDF.

[20] SankenElectric. Datenblatt: 2SD2390. http://www.semicon.sanken-ele.co.jp/sk_content/2sd2390_ds_en.pdf.

[21] Texas Instruments. Datenblatt: Tiva TM4C1294NCPDT Microcontroller. http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf.

[22] AnalogDevices. Datenblatt: AD5061 Rev. C, August, 2015. http://www.analog.com/media/en/technical-documentation/data-sheets/AD5061.pdf.

[23] Analog Devices. Datenblatt: ADR445ARZ - Rev. E, November 2010. http://www.analog.com/media/en/technical-documentation/data-sheets/ADR440_441_443_444_445.pdf.

[24] NixOnlineStore. http://www.nix.ru/autocatalog/arctic_cooling/Arctic-Cooling-Alpine-64-PLUS-4pin-754-AM2-AM3-FM1-600-2000-ob-min-135580.html.

[25] Texas Instruments. Datenblatt: LM1117 - Rev. Januar 2016. http://www.ti.com/lit/ds/symlink/lm1117.pdf.

[26] Texas Instruments. Datenblatt: LM325 - Rev. Dezember 2014. http://www.ti.com/lit/ds/symlink/lm158-n.pdf.

[27] Donald Reay Rulph Chassaing. *Digital Signal Processing - Second edition*. Wiley Interscience, 2008.

[28] Wolfgang Nerreter Arnold Führer, Klaus Heidemann. *Grundgebite der Elektrotechnik 2 - 8. Auflage*. Hanser, 2007.

[29] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler Band 1 - 14. Auflage*. Springer, Juni 2014.

[30] FuelCon. Produktseite TrueData-EIS. http://www.fuelcon.com/produkte/brennstoffzellentest/testzubehoer/truedata-eis/daten-zum-truedata-eis.html.

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

EIS	Elektrochemische Impedanzspektroskopie
UART	Universal Asynchronous Receiver Transmitter
SoH	State of Health
SoC	State of Charge
ppm	parts per million
SPI	Serial Peripheral Interface
BMS	Batterie-Management-System
TRL	Technology Readiness Level
NASA	National Aeronautics and Space Administration

A. Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg
Department Informations- und Elektrotechnik
Prof. Dr.-Ing. Karl-Ragmar Riemschneider

22. Februar 2016

Bachelorarbeit: Julian Akwaa

Anregeschaltung und Controllersoftware für die Elektrochemischen Impedanzspektroskopie für Lithium-Fahrzeug-Batterien

Motivation

In der Forschungsgruppe 'Zellensensoren für Fahrzeugbatterien - BATSEN' werden an der HAW Hamburg Sensoren für Batteriezellen entwickelt, die den Ladezustand und die Lebensdauer bestimmen sollen. Ein bekanntes Verfahren zur Analyse einer Batteriezelle unter Laborbedingungen ist die Elektrochemische Impedanzspektroskopie (EIS). Dabei werden der komplexe Widerstand der Batterie erfasst und auf der Basis von Batterie-Modellen verschiedene Parameter ermittelt. Das Verfahren der EIS soll mit Hilfe der entwickelten Zellensensoren auch bei Batterien nutzbar werden, die sich bereits im Fahrzeugeinsatz befinden.

Aufgabe

Für die EIS ist eine Anregung mit Wechselströmen mit Frequenzen ab 10 mHz bis zu einigen kHz, sowie ein einstellbarer Gleichstromanteil erforderlich. Weil die großen Batteriezellen Innenwiderstände bis unter 1 mΩ aufweisen, sind für messtechnisch nutzbare Spannungsantworten hohe Anregeströme bis in den Amperebereich notwendig.
Herrn Akwaa erhält die Aufgabe zu untersuchen, welche Anforderungen an eine Anregungschaltung für die Impedanzspektroskopie zu stellen sind. Dazu sind ein Konzept und eine Lösung in Hard- und Software zu entwickeln, die sich grundsätzlich für den Fahrzeugeinsatz eignen würden und zu den Zellensensoren passen. Die Anregung der Batterie soll mit Strömen in Sinusform erfolgen, wobei die diskreten Werte von einem Mikrocontroller vorgeben werden. Dabei sind Amplituden, Frequenzen und Offsets einstellbar. Mit einer Leistungsschaltung sind diese Vorgaben auf den Amperebereich zu verstärken.
Die erforderlichen Parameter (z.B. AC, DC-Stromwerte, Frequenzen) und die Signalqualität (z.B. Verzerrungen, Quantisierung) sind zu betrachten. Ein entsprechender messtechnischer Funktionsnachweis wird angestrebt.
Das Ergebnis ist mit einem verfügbaren Labor-Impedanzspektrometer zu vergleichen.
Die Aufgabe der Bachelorarbeit gliedert sich wie folgt:

1) **Erfassung der Vorarbeiten und Analyse der Rahmenbedingungen**

   - Einarbeitung in die Vorarbeiten im der Arbeitsgruppe BATSEN
   - Recherche der Fachliteratur zum Thema Impedanzspektroskopie
   - Diskussion von Lösungskonzepten am Beispiel eine LiFePO4-Starterbatterie mit 4 Zellen (Lade- bzw. Entladeschaltung, Nutzung im Bordnetz, Lastmodulation u.a.)
   - Abschätzen der erforderlichen Parameter unter Berücksichtigung der Sensor-Messauflösung, der Messraten und der Zelleninnenwiderstände u.a.

2) **Aufbau der Leistungstufe der Anregung**

- Diskussion und Auswahl von Schaltungsvarianten der Leistungstufe (z.B. MOSFET bzw. bipolarer Leistungstransistoren)
- Schaltungsimulation der Leistungstufe
- Laboraufbau der Leistungstufe und Inbetriebnahme
- optional: Platinenentwicklung und -aufbau

3) **Ansteuerung der Anregung**

- Controlleransteuerung auf Basis eines Evaluation-Board
- Auswahl und Ansteuerung eines geeigneten D/A-Umsetzers
- Controller-Software für die Wertvorgabe der Ansteuerung, Berücksichtigung des Zeitverhaltens für verschiedene Frequenzen
- Anbindung der Zellensensoren und eines Batteriesteuergerätes (weitere Controller)
- optional: Rückführung aus dem Steuergerät z.B. für den Offsetwert

4) **Erprobung**

- Aufstellung eines Test- und Messplans
- Aufzeichnen der Messwerte mittels Oszilloskop
- Erfassung und Dokumentation von Messreihen, Visualisierung mit Matlab-Scripten
- Exemplarische Zyklierung der Batterie verbunden mit EIS-Messungen bei verschiedenen Ladezuständen
- Referenzmessungen mit einem Labor-Impedanzspektrometer

5) **Gesamtbewertung und Fazit**

- Bewertung der entwickelten Anregeschaltung
- Analyse hinsichtlich Genauigkeit, Reproduzierbarkeit und systematischen Messfehlern
- Darstellung offener Punkte und weitergehender Verbesserungsmöglichkeiten

Dokumentation

Die Vorarbeiten, die Fachliteratur und Datenblätter sind zielgerichtet zu recherchieren. Die gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungen, die Grundkonzeption, auftretende Probleme und wesentliche Entwurfsentscheidungen sollen beschrieben werden. Die Erprobungsergebnisse sind in exemplarischem Umfang zu erfassen und auszuwerten. Passende Unterstützungsunterlagen für die Nutzung und für weitere Arbeiten sind zu erstellen.

B. Schaltplan

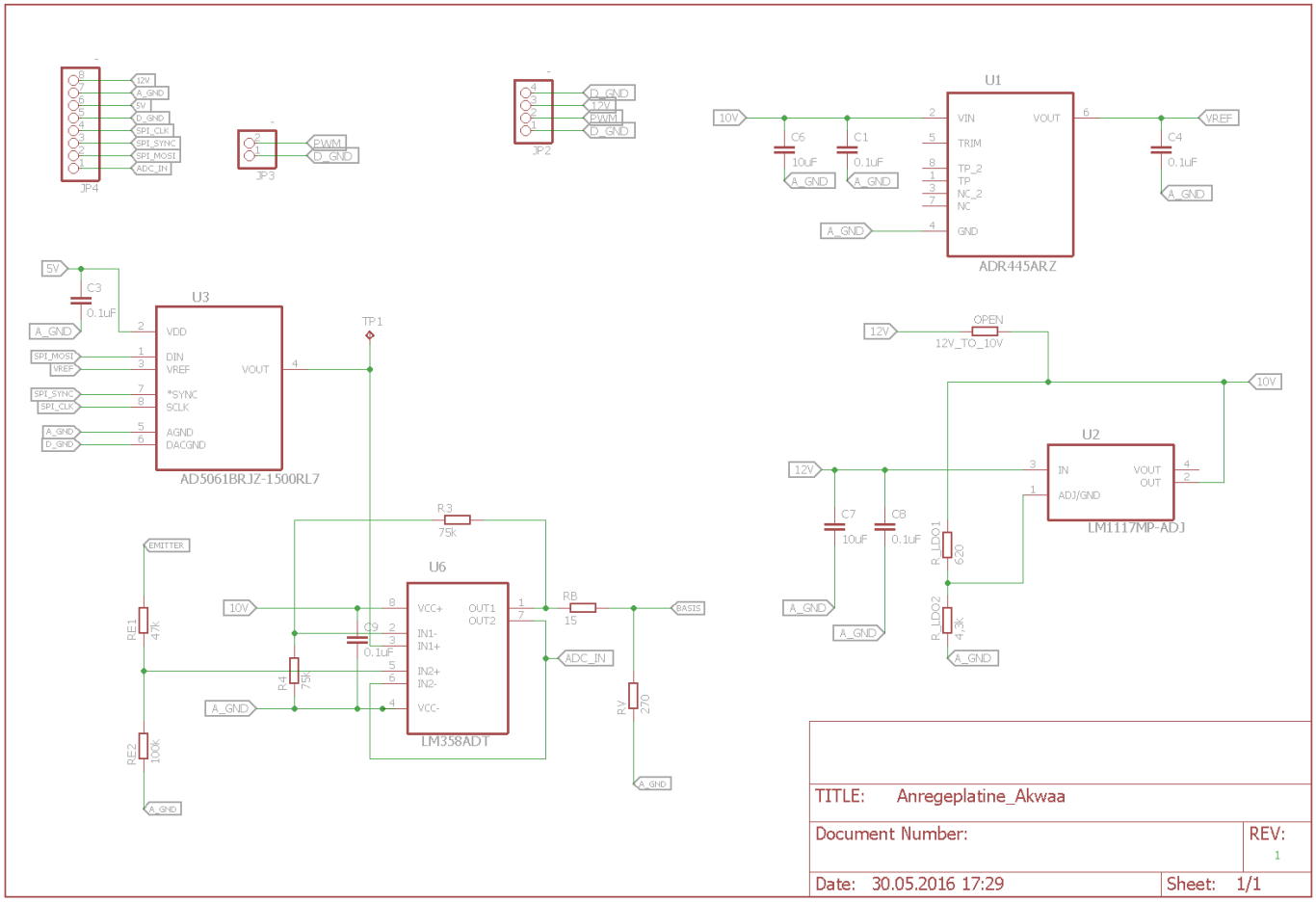

Abbildung B.1.: Schaltplan des EAGLE-Designs.

C. Platinenlayout

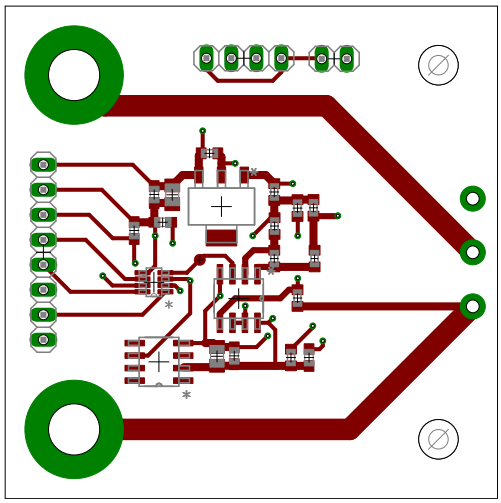

Abbildung C.1.: Anregeplatine in EAGLE - Front

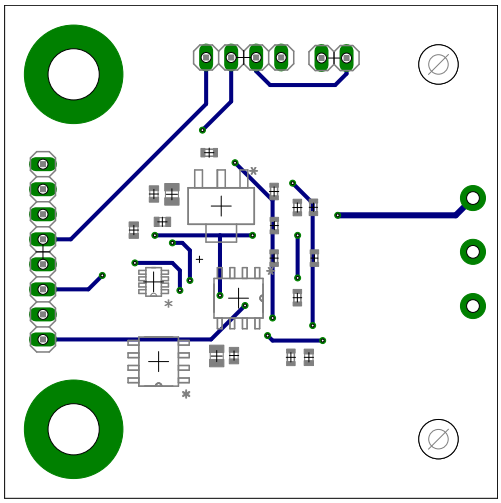

Abbildung C.2.: Anregeplatine in EAGLE - Rückseite

D. MATLAB Code

```
%%Aufnahme der Wechselstromwerte
ac(1)=mean([.24 .29 .31 .28 .32 .31 ]);     % 1000 Bit
ac(2)=mean([.51 .45 .47 .47 .46 .46]);      % 2000 Bit
ac(3)=mean([.63 .62 .66 .67 .63]);          % 3000 Bit
ac(4)=mean([.73 .73 .73 79 .79 .78]);       % 4000 Bit
ac(5)=mean([.87 .87 .91 .9 .91]);           % 5000 Bit
ac(6)=mean([1.13 1.13 1.05 1.05]);          % 6000 Bit
ac(7)=mean([1.18 1.18 1.17 1.24 1.23]);     % 7000 Bit
ac(8)=mean([1.31 1.3 1.35 1.35 1.34]);      % 8000 Bit
ac(9)=mean([1.62 1.64  1.53 1.53 1.53]);    % 9000 Bit
ac(10)=mean([1.77 1.77 1.72 1.72 1.72]);    %10000 Bit
ac(11)=mean([1.79 1.8 1.78 1.81 1.81]);     %11000 Bit
ac(12)=mean([1.96 1.95 2 2 2]);             %12000 Bit
ac(13)=mean([2.15 2.15 2.22 2.20 2.21]);    %13000 Bit
ac(14)=mean([2.26 2.28 2.57 2.58 2.58]);    %14000 Bit
ac(15)=mean([2.46 2.47 2.56 2.56 2.56]);    %15000 Bit
ac(16)=mean([2.57 2.58 2.87 2.88 2.89]);    %16000 Bit
ac(17)=mean([3.01 3.02 3.03 3.02]);

bit=1000:1000:17000;

stem(bit,ac);
title('AC-Strang');
xlabel('Bit');
ylabel('Strom [A]');
```

E. C-Quellcode

Listings

E.1. main.c

```
1   /*******************************************************************************
2   **    File name     : main.c
3   **    Hardware      : ARM Cortex M4 Stimulus Controller
4   **    Date          : 16/05/2014
5   **    Last Update   : 10/05/2016
6   **    Author        : Nico Sassano, modified by Julian Akwaa
7   **    Description   : Main routine
8   **    State         : Final state
9   *******************************************************************************/
10  #include <stdint.h>
11  #include <stdbool.h>
12  #include <math.h>
13  #include "inc/hw_ints.h"
14  #include "inc/hw_memmap.h"
15  #include "inc/hw_types.h"
16  #include "driverlib/debug.h"
17  #include "driverlib/fpu.h"
18  #include "driverlib/ssi.h"
19  #include "driverlib/gpio.h"
20  #include "driverlib/interrupt.h"
21  #include "driverlib/pin_map.h"
22  #include "driverlib/rom.h"
23  #include "driverlib/rom_map.h"
24  #include "driverlib/sysctl.h"
25  #include "driverlib/timer.h"
26  #include "driverlib/uart.h"
27  #include "utils/uartstdio.h"
28  #include "header/main.h"
29  #include "header/uart.h"
30  #include "header/spi.h"
31  #include "header/timer.h"
32  #include "header/interrupts.h"
33  #include "header/CC1101.h"
34  #include "header/gpio.h"
35  #include "header/convert.h"
36  #include "header/adc.h"
37
38
39  // System setting - Clock, UART, Timer, Interrupts
40
41  uint32_t g_sw_version                   = 150717;
42  uint32_t g_ui32SysClock;
43  uint32_t g_ui32Flags;
44  uint32_t spiBitRate                     =4700000;
45  uint32_t spiBit                         = 8;
46  uint32_t spiConfig                      = SSI_FRF_MOTO_MODE_1;//SSI_FRF_MOTO_MODE_0;
47  uint32_t rx_timeout                     = 3000;
48  uint32_t tx_timeout                     = 3000;
49  uint32_t ui32Baud                       = 56000;
50  uint32_t ui32Baud_FUELCON               = 9200;
51  uint32_t ui32PortNum                    = 0;
52  uint32_t ui32PortNum_FUELCON            = 4;
53  volatile uint32_t ui32SrcClock                  = 0;
54
55  volatile uint8_t rx_uart                = 0;                // RX command
56  volatile uint8_t uart_rx_flag           = 0;                // Flag for the UART
57  volatile unsigned char rx_char[31];                         // UART data buffer
58
59  volatile uint8_t irq_timer0_mode        = DELAY_MODE;       // Flag for the timer mode
60  volatile uint8_t irq_delay_flag         = 0;                // Flag for the delay timer
61  volatile uint8_t irq_timeout            = 0;                // Flag for the time out timer
62
63  void initStimulus();
64  void runStimulus();
65
66
67  int main(void) {
68
69      ROM_IntDisable(INT_WATCHDOG);                           // Disable watchdog-timer
70
71
72      g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
73              SYSCTL_OSC_MAIN |
74              SYSCTL_USE_PLL |
75              SYSCTL_CFG_VCO_480), 120000000);                // Set the clocking to run directly from the crystal at 120
                    MHz.
76
77      ui32SrcClock = g_ui32SysClock;
78
79
80      GPIOinit();
81      SPIinit(spiBitRate, spiBit, spiConfig);                 // Initialize SPI
```

```
82      initStimulus();
83      UARTinit(ui32PortNum, ui32Baud, ui32SrcClock);          // Initialize SPI
84
85      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);            // Enable for external clk interrupt
86      GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_5);      // Output pin for external clk interrupt
87      GPIOIntTypeSet(GPIO_PORTA_BASE, GPIO_PIN_5, GPIO_RISING_EDGE);  //set interrupt PA5, rising edge
88
89      IntPrioritySet(INT_UART0, 0x40);                        // UART - highest prio
90      IntPrioritySet(INT_GPIOA, 0x60);                        // Port A CLK interrupt
91      IntPrioritySet(INT_TIMER0A, 0x80);                      // Timer0
92
93      GPIOIntClear(GPIO_PORTA_BASE,GPIO_PIN_5);               // Clear interrut register of PA5
94
95      CS1_ENABLE;                                             // Write 0 to both DACs
96          SSIDataPut(SSI3_BASE, 0x00);        while(SSIBusy(SSI3_BASE));
97          SSIDataPut(SSI3_BASE, 0x0);         while(SSIBusy(SSI3_BASE));
98          SSIDataPut(SSI3_BASE, 0x0);         while(SSIBusy(SSI3_BASE));
99      CS1_DISABLE;
100     CS2_ENABLE;
101         SSIDataPut(SSI2_BASE, 0x00);        while(SSIBusy(SSI2_BASE));
102         SSIDataPut(SSI2_BASE, 0x0);         while(SSIBusy(SSI2_BASE));
103         SSIDataPut(SSI2_BASE, 0x0);         while(SSIBusy(SSI2_BASE));
104     CS2_DISABLE;
105
106
107     while(1)    ;
108 }
```

E.2. stimulus.c

```
1   /*******************************************************************************
2   **    File name     : stimulus.h
3   **    Hardware      : ARM Cortex M4 Stimulus Controller
4   **    Date          : 26/03/2016
5   **    Last Update   : 12/05/2016
6   **    Author        : Julian Akwaa
7   **    Description   : Output and calculation routines
8   **    State         : Final state
9   *******************************************************************************/
10  #include <stdint.h>
11  #include <stdbool.h>
12  #include <math.h>
13  #include "inc/hw_ints.h"
14  #include "inc/hw_memmap.h"
15  #include "inc/hw_types.h"
16  #include "driverlib/debug.h"
17  #include "driverlib/fpu.h"
18  #include "driverlib/ssi.h"
19  #include "driverlib/gpio.h"
20  #include "driverlib/interrupt.h"
21  #include "driverlib/pin_map.h"
22  #include "driverlib/rom.h"
23  #include "driverlib/rom_map.h"
24  #include "driverlib/sysctl.h"
25  #include "driverlib/timer.h"
26  #include "driverlib/uart.h"
27  #include "utils/uartstdio.h"
28  #include "header/main.h"
29  #include "header/uart.h"
30  #include "header/spi.h"
31  #include "header/timer.h"
32  #include "header/interrupts.h"
33  #include "header/CC1101.h"
34  #include "header/gpio.h"
35  #include "header/convert.h"
36  #include "header/adc.h"
37  #include "header/stimulus.h"
38
39  long DC = DEFAULT_DC;                                       // Default DC value
40  long AC = DEFAULT_AC;                                       // Default AC value
41  long AcOffset = 4;                                          // Default AC offset value
42  volatile uint16_t sinus[SIN_POINTS];                        // Stores the sine values of the stimulation
43  int DCinBit, ACinBit;
44  volatile int DCset = 0, ACset = 0;
45  extern uint32_t ui32SrcClock;
46
47
48  void initStimulus(){
49      TIMER0init();                                           // Initialization Timer 0
50      TIMER2init();                                           // Initialization Timer 2
51      init_adc();                                             // Initialization of the ADCs
```

```
52  }
53
54  // Calls all functions necessary for preparation of the current stimulation
55  void runStimulus(){
56          setCurrent();
57          adjustDC();
58          adjustAC();
59          GPIOIntEnable(GPIO_PORTA_BASE, GPIO_PIN_5);             // Enables the interrupt wich allows the AC values to be
                written to the DAC
60          IntEnable(INT_GPIOA);
61          GPIOIntClear(GPIO_PORTA_BASE,GPIO_PIN_5);               // Clear Iterrupt PA5
62  }
63
64  // Splits the DC current and calculates the bit values accordingly
65  void setCurrent(){
66
67      DCset=DC;
68      AcOffset = 4;                                               // Default offset
69      if(DCset > 4){                                              // If the DC is greather than 4, split it
70          DCset -= 4;
71          AcOffset = getCh1CurrentInBit(AcOffset);                //Calculate current to bit values
72          DCinBit = DCset;
73      }else{
74          AcOffset = getCh1CurrentInBit(DCset);                   //Calculate current to bit values
75          DCset = 0;
76          DCinBit = 0;
77      }
78
79      DCinBit = (DCset != 0)? getCh2CurrentInBit(DCinBit):0;
80
81      DCinBit = (DCinBit < MIN_CURRENT) ? MIN_CURRENT:DCinBit;    // Limit current values to avoid errors
82      DCinBit = (DCinBit > MAX_CURRENT) ? MAX_CURRENT:DCinBit;
83
84
85      ACinBit = getCh1CurrentInBit(AC);
86      ACinBit = (ACinBit < MIN_AMPLITUDE) ? MIN_AMPLITUDE:ACinBit;// Limit current values to avoid errors
87      ACinBit = (ACinBit > MAX_AMPLITUDE) ? MAX_AMPLITUDE:ACinBit;
88  }
89
90  // Calculates the sine values according to amplitude and ofsset values and stores them in an array
91  void calculate(int amplitude){
92      int n;
93
94      if(AC == 0) amplitude = 0;
95
96      for(n = 0;n < SIN_POINTS;n++){                              // Calculate sine values and store them
97          sinus[n] = amplitude/2 * sin(2*PI*n*1/SIN_POINTS) + ACset;
98      }
99  }
100
101 // Sets the output value of the DC DAC so that the desired DC current results
102 void adjustDC(){
103     int set=0, v=0, change = CHANGE;
104     signed short lock = -3;                                     // Initialization error lock buffer
105
106     do{
107         set = (set >= 65535)? 0:set;                            // Reset set if it exceeds maximum
108         if(!DCset) break;                                       // If desired DC part is zero, no adjustment is needed
109
110         CS1_ENABLE;                                             // Set the DAC1 Output to the value in variable set
111             SSIDataPut(SSI3_BASE, 0x00);
112             while(SSIBusy(SSI3_BASE));
113
114             SSIDataPut(SSI3_BASE, set >> 8);
115             while(SSIBusy(SSI3_BASE));
116
117             SSIDataPut(SSI3_BASE, set & 0xFF);
118             while(SSIBusy(SSI3_BASE));
119         CS1_DISABLE;
120
121         TIMER0config_ms(1);                                     // Delay so the DAC can engage the voltage modified above
                (DA switching times are a few ns)
122         v = read_adc1() * 16;                                   // Get the ADC1 value and convert to 2^16 bit -> 2^12 *
              2^4 = 2^16
123         set = (v > 65000) ? 0:set;                              // If current value exceeds maximum, reset to zero
124         (lock++ < 0)? v=0 : lock++;                             // Protection from initialisation errors
125
126         if(v <= DCinBit - DC_TOLERANCE){                        // If desired value isn't reached, increase variable set
127             set += change;
128         }
129
130         if(v >= DCinBit + DC_TOLERANCE){                        // If ADC value is greater than desired value
131             set = (set - 2*change > 0) ? set - 2*change : 0;    // Decrease set if set isn't less than zero
132             change *= .5;                                       // Half the increase amount
133             if(change < 1){                                     // If the increase amount is less than 1 reset it
```

```
134                 change = CHANGE;
135             }
136         }
137     }
138     while(v >= DCinBit + DC_TOLERANCE || v <= DCinBit - DC_TOLERANCE); // Repeat until desired Curret value is measured
            by the ADC
139     DCset = set;
140 }
141
142 // Sets the output value of the AC DAC
143 void adjustAC(){
144     int set = 0, v = 0, change = CHANGE, offset = 0;
145     signed short lock = -3;                                     // Initialization error lock buffer
146
147     do{                                                         // Set DC offset of the AC part
148         set = (set >= 65535)? 0:set;                            // Reset the variable set if it exceeds maximum
149         CS2_ENABLE;                                             // Set the DAC2 Output to the value in variable set
150             SSIDataPut(SSI2_BASE, 0x00);
151             while(SSIBusy(SSI2_BASE));
152
153             SSIDataPut(SSI2_BASE, set >> 8);
154             while(SSIBusy(SSI2_BASE));
155
156             SSIDataPut(SSI2_BASE, set & 0xFF);
157             while(SSIBusy(SSI2_BASE));
158         CS2_DISABLE;
159
160         TIMER0config_ms(100);                                   // Delay so the DAC can engage the voltage modified above
                (DA switching times are a few ns)
161         v = read_adc2() * 16;                                   // Get ADC2 value and convert to 2^16 bit -> 2^12 * 2^4 =
                2^16
162         set = (v > 65000) ? 0:set;                              // If current value exceeds maximum, reset to zero
163         (lock++ < 0)? v=0 : lock++;                             // Protection from initialisation errors
164
165
166         if(v <= AcOffset - DC_TOLERANCE){                       // If desired value isn't reached, increase variable set
167             set += change;
168         }
169
170         if(v >= (int)AcOffset + DC_TOLERANCE){                  // If ADC value is greater than desired value
171             set = (set - 2*change > 0) ? set - 2*change : 0;    // Decrease set if set isn't less than zero
172             change *= .5;                                       // Half the increase amount
173             if(change < 1){                                     // If the increase amount is less than 1 reset it
174                 change = CHANGE;
175             }
176         }
177     }
178     while(v >= (int)AcOffset + DC_TOLERANCE || v <= (int)AcOffset - DC_TOLERANCE || set == 0); // Repeat until desired
            Curret value is measured by the ADC
179     ACset = set;
180     change = CHANGE;                                            // Reset the modified change value
181     offset = read_adc2() * 16;                                  // Get ADC2 value and convert to 2^16 bit -> 2^12 * 2^4 =
            2^16
182
183     do{                                                         // Set the maximum amplitude value
184         set = (set >= 65535)? 0:set;                            // Reset the variable set if it exceeds maximum
185         CS2_ENABLE;                                             // Set the DAC2 Output to the value in variable set
186             SSIDataPut(SSI2_BASE, 0x00);
187             while(SSIBusy(SSI2_BASE));
188
189             SSIDataPut(SSI2_BASE, set >> 8);
190             while(SSIBusy(SSI2_BASE));
191
192             SSIDataPut(SSI2_BASE, set & 0xFF);
193             while(SSIBusy(SSI2_BASE));
194         CS2_DISABLE;
195         TIMER0config_ms(1);                                     // Delay so the DAC can engage the voltage modified above
                (DA switching times are a few ns)
196         v = read_adc2() * 16;                                   // Get adc2 value and convert to 2^16 bit -> 2^12 * 2^4 =
                2^16
197         set = (v > 65000) ? 0:set;                              // If current value exceeds maximum, reset to zero
198         (lock++ < 0)? v=0 : lock++;                             // Protection from initialisation errors
199
200         if(v <= offset + ACinBit - AC_TOLERANCE){               // If desired value isn't reached, increase variable set
201             set += change;
202         }
203
204         if(v >= offset + ACinBit + AC_TOLERANCE){               // If ADC value is greater than desired value
205             set = (set - 2*change > 0) ? set - 2*change : 0;    // Decrease set if set isn't less than zero
206             change *= .5;                                       // Half the increase amount
207             if(change < 1){                                     // If the increase amount is less than 1 reset it
208                 change = CHANGE;
209             }
210         }
211     }
```

```
212       while(v >= offset + ACinBit + AC_TOLERANCE || v <= offset + ACinBit - AC_TOLERANCE); // Repeat until desired Curret
              value is measured by the ADC
213       calculate(set - AC_OFFSET);
214   }
215
216   #define DC_M 126e-6
217   #define DC_B 0.16
218   #define AC_M 147e-6
219   #define AC_B 0.104
220
221   // Correct function for output precision
222   unsigned int getCh1CurrentInBit(long I){
223       return (I - AC_B) / AC_M;
224   }
225
226   // Correct function for output precision
227   unsigned int getCh2CurrentInBit(long I){
228       return (I - DC_B) / DC_M;
229   }
230
231   // Correct function for output precision
232   long getCh1CurrentInAmps(int bit){
233       return AC_M * bit + AC_B;
234   }
235
236   // Correct function for output precision
237   long getCh2CurrentInAmps(int bit){
238       return DC_M * bit + DC_B;
239   }
240
241   // Output Disable for both DACs
242   void disableOutput(){
243       CS1_ENABLE;
244           SSIDataPut(SSI3_BASE, 0x00);    while(SSIBusy(SSI3_BASE));
245           SSIDataPut(SSI3_BASE, 0x00);    while(SSIBusy(SSI3_BASE));
246           SSIDataPut(SSI3_BASE, 0x00);    while(SSIBusy(SSI3_BASE));
247       CS1_DISABLE;
248       CS2_ENABLE;
249           SSIDataPut(SSI2_BASE, 0x00);    while(SSIBusy(SSI2_BASE));
250           SSIDataPut(SSI2_BASE, 0x00);    while(SSIBusy(SSI2_BASE));
251           SSIDataPut(SSI2_BASE, 0x00);    while(SSIBusy(SSI2_BASE));
252       CS2_DISABLE;
253   }
```

E.3. interrupts.c

```
  1   /*****************************************************************************
  2   **    File name     : interrupts.c
  3   **    Hardware      : ARM Cortex M4 Stimulus Controller
  4   **    Date          : 16/05/2014
  5   **    Last Update   : 10/05/2016
  6   **    Author        : Nico Sassano, modified by Julian Akwaa
  7   **    Description   : Interrupt Handler
  8   **    State         : Final state
  9   *****************************************************************************/
 10   #include <stdint.h>
 11   #include <stdbool.h>
 12   #include "inc/hw_ints.h"
 13   #include "inc/hw_memmap.h"
 14   #include "inc/hw_types.h"
 15   #include "driverlib/debug.h"
 16   #include "driverlib/adc.h"
 17   #include "driverlib/gpio.h"
 18   #include "driverlib/ssi.h"
 19   #include "driverlib/interrupt.h"
 20   #include "driverlib/pin_map.h"
 21   #include "driverlib/rom.h"
 22   #include "driverlib/rom_map.h"
 23   #include "driverlib/sysctl.h"
 24   #include "driverlib/uart.h"
 25   #include "driverlib/timer.h"
 26   #include "utils/uartstdio.h"
 27   #include "header/main.h"
 28   #include "header/interrupts.h"
 29   #include "header/gpio.h"
 30   #include "header/timer.h"
 31   #include "header/adc.h"
 32   #include "header/CC1101.h"
 33   #include "header/uart.h"
 34   #include "header/stimulus.h"
 35
```

```
36
37  extern volatile uint8_t irq_timer0_mode;               // Flag for the timer mode
38  extern volatile uint8_t irq_delay_flag;                // Flag for the delay timer
39  extern volatile uint8_t irq_timeout;                   // Flag for the time out timer
40
41  extern volatile uint16_t sinus[];
42  uint16_t sin_index = 0x00;
43
44  extern uint32_t g_ui32SysClock;
45  extern long DC, AC;
46  extern int DCset, ACset ;
47
48
49
50  #define ONE_CHAR 1
51
52  // define receive commands
53  #define OUTPUT_STOP 0x00
54  #define OUTPUT_START 0x01
55  #define CHANGE_DC 0x02
56  #define CHANGE_AC 0x03
57
58  // define transfer commands
59  #define OUTPUT_STOPPED 0x00
60  #define OUTPUT_STARTED 0x01
61
62  // define min-/max input current values
63  #define MIN_DC_INPUT 0
64  #define MAX_DC_INPUT 12
65  #define MIN_AC_INPUT 0
66  #define MAX_AC_INPUT 8
67
68
69  // on PA5 interrupt writes sine data to the AC DAC
70  void PortAIntHandler(void){
71
72      GPIOIntClear(GPIO_PORTA_BASE,GPIO_PIN_5);          // Clear interruot register PA5
73      CS2_ENABLE;                                        // Write sine value to DAC2
74          SSIDataPut(SSI2_BASE, 0x00);
75          while(SSIBusy(SSI2_BASE));                     // Wait until SSI3 is done transferring all the data in the
                transmit FIFO.
76
77          SSIDataPut(SSI2_BASE, sinus[sin_index] >> 8);
78          while(SSIBusy(SSI2_BASE));                     // Wait until SSI3 is done transferring all the data in the
                transmit FIFO.
79
80          SSIDataPut(SSI2_BASE, sinus[sin_index] & 0xFF);
81          while(SSIBusy(SSI2_BASE));                     // Wait until SSI3 is done transferring all the data in the
                transmit FIFO.
82      CS2_DISABLE;
83
84      if(++sin_index == SIN_POINTS){                     // Incrase or reset sinus index variable
85          sin_index = 0;
86      }
87  }
88
89
90  // Will read the UART buffer register
91  // According to the received command will start, or stop the signal output
92  // or change voltage parameters into received values.
93  void UARTIntHandler(void) {
94
95      uint32_t ui32Status;
96      long c;
97
98
99      ui32Status = UARTIntStatus(UART0_BASE, true);      // Get interrrupt status.
100     UARTIntClear(UART0_BASE, ui32Status);              // Clear the asserted interrupts.
101     GPIOIntDisable(GPIO_PORTA_BASE, GPIO_PIN_5);       // Disable Interrupt PA5
102     IntDisable(INT_GPIOA);                             // Disable Interrupt Port 5
103     disableOutput();
104
105
106     c = UARTCharGetNonBlocking(UART0_BASE);            // Read char from buffer register
107     if(c == -1){                                       // Exit function if buffer is empty
108         return;
109     }
110     UARTprintf("\nCurrent DC value: %dA. Current AC value: %dA.\n", DC, AC);
111
112     switch(c){                                         // Switch read character
113
114         case OUTPUT_STOP:
115             UARTSend(OUTPUT_STOPPED, 1);               // Send stop confirm signal to Battery Control Unit
116             UARTprintf("Output disabled.\n");
117             break;
```

```
118
119         case OUTPUT_START:
120             if(AC > DC){
121                 UARTprintf("\n\nAC amplitude cannot be greater than DC offset. Change values accordingly.\nOutput
                        disabled.\n");
122             }else{
123                 UARTprintf("Setting current...\n");
124                 runStimulus();                          // Start Stimulation Initialization
125                 UARTSend((uint8_t *)OUTPUT_STARTED, ONE_CHAR); // Send start conform signal
126                 UARTprintf(" Output enabled.\n");
127             }
128             break;
129
130         case CHANGE_DC:             //DC change
131             UARTprintf("Enter desired DC value\n");
132             while(!ROM_UARTCharsAvail(UART0_BASE));     // Wait for char in buffer register
133
134             c = UARTCharGetNonBlocking(UART0_BASE);     // Get char from buffer register
135             if(c < MIN_DC_INPUT || c > MAX_DC_INPUT){
136                 UARTprintf("DC input value out of bounds. Values between %d and %d are valid.\nDC value remains unchanged
                        (%dA).\n",MIN_DC_INPUT, MAX_DC_INPUT, DC);
137             }else{
138                 DC = c;
139                 UARTprintf("DC value set to %dA. Output disabled.\n",DC);
140             }
141             break;
142
143         case CHANGE_AC:                                 // Change AC current parameter
144             UARTprintf("Enter desired AC value\n");
145             while(!ROM_UARTCharsAvail(UART0_BASE));     // Wait for char in buffer register
146
147             c = UARTCharGetNonBlocking(UART0_BASE);     // Read char from buffer register
148             if(c < MIN_AC_INPUT || c > MAX_AC_INPUT){
149                 UARTprintf("DC input value out of bounds. Values between %d and %d are valid.\nAC value remains unchanged
                        (%dA).\n",MIN_AC_INPUT, MAX_AC_INPUT, DC);
150             }else{
151                 AC = c;
152                 UARTprintf("DC value set to %dA. Output disabled.\n",AC);
153             }
154             break;
155
156         default:                                        // Invalid char read
157             UARTprintf("Invalid Command.\n",AC);
158             break;
159     }
160 }
161
162 // Interrupt handler for the delay and the timeout function.
163 void Timer0IntHandler(void) {
164
165
166     ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt.
167     ROM_TimerDisable(TIMER0_BASE, TIMER_A);             // Disable the timer interrupt.
168
169     if(TIMER_MODE_IS_TIMEOUT) {                         // Check whether this is the delay timer or the timeout timer
170         timeout_stop();                                 // Stop the timeout mode
171         TIMEOUT_SET;
172     } else if(TIMER_MODE_IS_DELAY){
173         timeout_stop();                                 // Stop the delay mode
174         irq_delay_flag = 1;
175     }
176 }
```

E.4. adc.c

```
1  /*******************************************************************************
2  **    File name     : adc.c
3  **    Hardware      : ARM Cortex M4 Basestation
4  **    Date          : 17/09/2014
5  **    Last Update   : 02/04/2016
6  **    Author        : Nico Sassano, modified by Julian Akwaa
7  **    Description   : ADC
8  **    State         : Final state
9  *******************************************************************************/
10 #include <stdbool.h>
11 #include <stdint.h>
12 #include "inc/hw_memmap.h"
13 #include "driverlib/adc.h"
14 #include "driverlib/gpio.h"
15 #include "driverlib/pin_map.h"
16 #include "driverlib/sysctl.h"
```

```
17  #include "driverlib/uart.h"
18  #include "utils/uartstdio.h"
19
20  #define SAMPLE_SEQUENCE_NUMBER 3
21
22
23  //Initialize 12bit 3.3V range ADCs 0 & 1
24  void init_adc() {
25      // The ADC0 peripheral must be enabled for use.
26      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
27      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);//ADC1
28
29      // Using PE3 -> This pin is AIN0
30      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
31      GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
32      GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);//AIN1
33
34      // Enable sample sequence 3 with a processor signal trigger.  Sequence 3
35      // will do a single sample when the processor sends a signal to start the
36      // conversion.  Each ADC module has 4 programmable sequences, sequence 0
37      // to sequence 3.  This example is arbitrarily using sequence 3.
38      ADCSequenceConfigure(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER, ADC_TRIGGER_PROCESSOR, 1);
39      ADCSequenceConfigure(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER, ADC_TRIGGER_PROCESSOR, 0);
40
41      // Configure step 0 on sequence 3.  Sample channel 0 (ADC_CTL_CH0) in
42      // single-ended mode (default) and configure the interrupt flag
43      // (ADC_CTL_IE) to be set when the sample is done.  Tell the ADC logic
44      // that this is the last conversion on sequence 3 (ADC_CTL_END).  Sequence
45      // 3 has only one programmable step.  Sequence 1 and 2 have 4 steps, and
46      // sequence 0 has 8 programmable steps.  Since we are only doing a single
47      // conversion using sequence 3 we will only configure step 0.  For more
48      // information on the ADC sequences and steps, reference the datasheet.
49      ADCSequenceStepConfigure(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
50      ADCSequenceStepConfigure(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER, 0, ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_END);
51
52
53      // Since sample sequence 3 is now configured, it must be enabled.
54      ADCSequenceEnable(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER);
55      ADCSequenceEnable(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER);
56
57      // Clear the interrupt status flag.  This is done to make sure the
58      // interrupt flag is cleared before we sample.
59      ADCIntClear(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER);
60      ADCIntClear(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER);
61
62  }
63
64  uint32_t read_adc1() {
65      uint32_t pui32ADC0Value[1];
66
67      // Trigger the ADC conversion.
68      ADCProcessorTrigger(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER);
69
70      // Wait for conversion to be completed.
71      while(!ADCIntStatus(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER, false));
72
73      // Clear the ADC interrupt flag.
74      ADCIntClear(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER);
75
76      // Read ADC Value.
77
78      ADCSequenceDataGet(ADC0_BASE, SAMPLE_SEQUENCE_NUMBER, pui32ADC0Value);
79
80      return pui32ADC0Value[0];
81  }
82
83
84  uint32_t read_adc2() {
85      uint32_t pui32ADC1Value[1];
86
87      // Trigger the ADC conversion.
88      ADCProcessorTrigger(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER);
89
90      // Wait for conversion to be completed.
91      while(!ADCIntStatus(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER, false));
92
93      // Clear the ADC interrupt flag.
94      ADCIntClear(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER);
95
96      // Read ADC Value.
97      ADCSequenceDataGet(ADC1_BASE, SAMPLE_SEQUENCE_NUMBER, pui32ADC1Value);
98
99      return pui32ADC1Value[0];
100 }
```

E.5. gpio.c

```
/*******************************************************************************
**    File name     : gpio.c
**    Hardware      : ARM Cortex M4 Basestation
**    Date          : 08/08/2014
**    Last Update   : 02/04/2016
**    Author        : Nico Sassano, modified by Julian Akwaa
**    Description   : GPIO Functions
**    State         : Final state
*******************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "header/uart.h"
#include "header/spi.h"
#include "header/timer.h"
#include "header/interrupts.h"
#include "header/gpio.h"

// Enables used pins and sets their directions
void GPIOinit(){
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);

    // Current Sensor
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_4);
    ROM_GPIOPinWrite(GPIO_PORTM_BASE, GPIO_PIN_4, 0xFF);

    // Testpin
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTQ_BASE, GPIO_PIN_4);
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);

    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_3); // For ssi2 clk DA2
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTQ_BASE, GPIO_PIN_1); // For sii3 clk DA1

}
```

E.6. spi.c

```
/*******************************************************************************
**    File name     : spi.h
**    Hardware      : ARM Cortex M4 Stimulus Conroller
**    Date          : 16/05/2014
**    Last Update   : 02/05/2016
**    Author        : Nico Sassano, modified by Julian Akwaa
**    Description   : SPI
**    State         : Final state
*******************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
```

```
22  #include "driverlib/rom_map.h"
23  #include "driverlib/sysctl.h"
24  #include "driverlib/timer.h"
25  #include "driverlib/uart.h"
26  #include "utils/uartstdio.h"
27  #include "header/uart.h"
28  #include "header/interrupts.h"
29  #include "header/CC1101.h"
30
31  // Initializes the SPI
32  void SPIinit(uint32_t spiBitRate, uint8_t spiBit, uint32_t spiConfig) {
33
34      SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);         // Enable SSI2
35      SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI3);         // Enable SSI3
36
37      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);        // Enable GPIOQ
38      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);        // Enable GPIOD
39      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);        // Enable GPION
40
41      GPIOPinConfigure(GPIO_PQ0_SSI3CLK);                 // Configure pin muxing.
42      GPIOPinConfigure(GPIO_PD3_SSI2CLK);
43      GPIOPinConfigure(GPIO_PQ2_SSI3XDAT0);
44      GPIOPinConfigure(GPIO_PQ3_SSI3XDAT1);
45      GPIOPinConfigure(GPIO_PD1_SSI2XDAT0);
46      GPIOPinConfigure(GPIO_PD0_SSI2XDAT1);
47
48
49      // Configure the GPIO settings for the SSI pins.
50      //          DA1                                         DA2
51      //      PQ2 - SSI3Tx    MOSI                        D1      green
52      //      PQ3 - SSI3Rx    MISO                        D0
53      //      PQ1 - SSI3Fss   Chip select                 N3      yellow
54      //      PQ0 - SSI3CLK   CLK                         D3      white
55      GPIOPinTypeSSI(GPIO_PORTQ_BASE, GPIO_PIN_0 | GPIO_PIN_2 | GPIO_PIN_3);
56      GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_3);
57
58      // Configure and enable the SSI port for SPI master mode.  Use SSI0,
59      // system clock supply, idle clock level low and active low clock in
60      // freescale SPI mode, master mode, 1MHz SSI frequency, and 8-bit data.
61      // For SPI mode, you can set the polarity of the SSI clock when the SSI
62      // unit is idle.  You can also configure what clock edge you want to
63      // capture data on.  Please reference the datasheet for more information on
64      // the different SPI modes.
65      SSIConfigSetExpClk(SSI3_BASE, SysCtlClockGet(), spiConfig, SSI_MODE_MASTER, spiBitRate, spiBit);
66      SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(), spiConfig, SSI_MODE_MASTER, spiBitRate, spiBit);
67
68
69      SSIEnable(SSI3_BASE);                               // Enable SSI modules.
70      SSIEnable(SSI2_BASE);
71  }
```

E.7. uart.c

```
1   /*****************************************************************************
2   **    File name     : uart.c
3   **    Hardware      : ARM Cortex M4 Basestation
4   **    Date          : 14/09/2014
5   **    Last Update   : 29/03/2016
6   **    Author        : Nico Sassano, modified by Julian Akwaa
7   **    Description   : UART function
8   **    State         : Final state
9   *****************************************************************************/
10  #include <stdint.h>
11  #include <stdbool.h>
12  #include "inc/hw_ints.h"
13  #include "inc/hw_memmap.h"
14  #include "driverlib/debug.h"
15  #include "driverlib/gpio.h"
16  #include "driverlib/interrupt.h"
17  #include "driverlib/pin_map.h"
18  #include "driverlib/rom.h"
19  #include "driverlib/rom_map.h"
20  #include "driverlib/sysctl.h"
21  #include "driverlib/uart.h"
22  #include "utils/uartstdio.h"
23  #include "header/uart.h"
24  #include "header/stimulus.h"
25
26
27  // extern variable declarations
28  extern uint32_t g_ui32SysClock;
```

```
29  extern long DC, AC;
30  extern int DCset, ACset ;
31
32
33  // Send a string to the UART.
34  void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count) {
35      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 1);
36
37      // Loop while there are more characters to send
38      while(ui32Count--)
39      {
40          // Write the next character to the UART
41          ROM_UARTCharPutNonBlocking(UART0_BASE, *pui8Buffer++);
42
43          // Delay for 1 millisecond, necessary because UART Baud rate is slow
44          SysCtlDelay(g_ui32SysClock / (1000 * 3));
45
46      }
47      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0);
48  }
49
50  void UARTinit(uint32_t port_num,uint32_t data_rate, uint32_t clock_rate){
51
52      // Enable the GPIO Peripheral used by the UART
53      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
54
55      // Enable UART0
56      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
57
58      // Configure GPIO Pins for UART mode
59      ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
60      ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
61      ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
62
63      // Enable the UART interrupt
64      ROM_IntEnable(INT_UART0);
65      ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
66
67      UARTFIFODisable(UART0_BASE);
68
69      // Initialize the UART
70      UARTStdioConfig(port_num, data_rate, clock_rate);
71
72      UARTprintf("Ready!\n");
73
74  }
```

E.8. timer.c

```
1   /*****************************************************************************
2   **    File name     : timer.c
3   **    Hardware      : ARM Cortex M4 Basestation
4   **    Date          : 16/05/2014
5   **    Last Update   : 25/04/2016
6   **    Author        : Nico Sassano, modified by Julian Akwaa
7   **    Description   : Timer function
8   **    State         : Final state
9   *****************************************************************************/
10  #include <stdint.h>
11  #include <stdbool.h>
12  #include "inc/hw_ints.h"
13  #include "inc/hw_memmap.h"
14  #include "driverlib/debug.h"
15  #include "driverlib/gpio.h"
16  #include "driverlib/interrupt.h"
17  #include "driverlib/pin_map.h"
18  #include "driverlib/rom.h"
19  #include "driverlib/rom_map.h"
20  #include "driverlib/sysctl.h"
21  #include "driverlib/uart.h"
22  #include "driverlib/timer.h"
23  #include "utils/uartstdio.h"
24  #include "header/timer.h"
25
26  //*****************************************************************************
27  // System clock rate in Hz.
28  //*****************************************************************************
29  extern uint32_t g_ui32SysClock;
30  extern volatile uint8_t irq_delay_flag;
31
32  extern volatile uint8_t irq_timer0_mode;
```

```
33  extern volatile uint8_t irq_timer1_mode;    // Flag for the timer1 mode
34  extern volatile uint8_t irq_timeout;        // Flag for the time out timer
35
36  extern volatile uint32_t burst_config;
37
38  void TIMER0init(){
39      // Enable the peripherals used by this example.
40      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
41
42      //UARTprintf("TIMER0 initialization done...\n");
43  }
44
45  void TIMER1init(){
46      // Enable the peripherals used by this example.
47      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
48
49      UARTprintf("TIMER1 initialization done...\n");
50  }
51
52  void TIMER2init(){
53      // Enable the peripherals used by this example. SINUS TIMER
54      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
55
56  }
57
58  void TIMER2config(uint32_t config){
59      // Configure the two 32-bit one shot timers.
60      TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
61      TimerLoadSet(TIMER2_BASE, TIMER_A, (g_ui32SysClock/120000000*config) ); // (load value) Config the clocktime ->
            g_ui32SysClock / config
62
63      // Setup the interrupts for the timer timeouts.
64      ROM_IntEnable(INT_TIMER2A);
65      ROM_TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
66      TIMER2_STOP;
67  }
68
69
70
71
72  //*****************************************************************************
73  // Function to config the Timer0
74  // Timer0 is the counter for the delay function
75  // Timer is in ms
76  //*****************************************************************************
77  void TIMER0config_ms(uint32_t config){
78      // Configure the two 32-bit one shot timers.
79      ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
80      ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, (g_ui32SysClock/1000)*config );  ////Config the clocktime -> g_ui32SysClock /
            config
81
82      // Setup the interrupts for the timer timeouts.
83      ROM_IntEnable(INT_TIMER0A);
84      ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
85
86      // Enable the timer 0.
87      TIMER0_START;
88  }
89
90  //*****************************************************************************
91  // Function to config the Timer0
92  // Timer0 is the counter for the delay function
93  // Timer is in us
94  //*****************************************************************************
95  void TIMER0config_us(uint32_t config){
96      // Configure the two 32-bit one shot timers.
97      ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
98      ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, (g_ui32SysClock/1000000)*config );   ////Config the clocktime ->
            g_ui32SysClock / config
99
100     // Setup the interrupts for the timer timeouts.
101     ROM_IntEnable(INT_TIMER0A);
102     ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
103
104     // Enable the timer 0.
105     TIMER0_START;
106 }
107
108 // Delay function in ms
109 void delay_ms(uint32_t ms){
110     TIMER_MODE_SET_DELAY;
111     irq_delay_flag = 0;
112     TIMER0config_ms(ms);
113
114     while(!irq_delay_flag);
```

```
115 }
116
117 // Delay function in us
118 void delay_us(uint32_t us){
119     TIMER_MODE_SET_DELAY;
120     irq_delay_flag = 0;
121     TIMER0config_us(us);
122
123     while(!irq_delay_flag);
124 }
125
126
127 // Timeout function
128 void timeout_start(uint32_t ms){
129     TIMER_MODE_SET_TIMEOUT;
130     TIMEOUT_UNSET;
131     TIMER0config_ms(ms);
132
133     //while(!TIMEOUT_IS_SET);
134 }
135
136 void timeout_stop() {
137     TIMER_MODE_SET_DELAY;
138
139     // Disable the timers.
140     ROM_TimerDisable(TIMER0_BASE, TIMER_A);
141
142 }
```

E.9. startup _ccs.c

```
  1 //*****************************************************************************
  2 //
  3 // startup_ccs.c - Startup code for use with TI's Code Composer Studio.
  4 //
  5 // Copyright (c) 2013-2014 Texas Instruments Incorporated.  All rights reserved.
  6 // Software License Agreement
  7 //
  8 // Texas Instruments (TI) is supplying this software for use solely and
  9 // exclusively on TI's microcontroller products. The software is owned by
 10 // TI and/or its suppliers, and is protected under applicable copyright
 11 // laws. You may not combine this software with "viral" open-source
 12 // software in order to form a larger program.
 13 //
 14 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
 15 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
 16 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 17 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
 18 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
 19 // DAMAGES, FOR ANY REASON WHATSOEVER.
 20 //
 21 // This is part of revision 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
 22 //
 23 //*****************************************************************************
 24
 25 #include <stdint.h>
 26 #include "inc/hw_nvic.h"
 27 #include "inc/hw_types.h"
 28
 29 //*****************************************************************************
 30 //
 31 // Forward declaration of the default fault handlers.
 32 //
 33 //*****************************************************************************
 34 void ResetISR(void);
 35 static void NmiSR(void);
 36 static void FaultISR(void);
 37 static void IntDefaultHandler(void);
 38
 39 //*****************************************************************************
 40 //
 41 // External declaration for the reset handler that is to be called when the
 42 // processor is started
 43 //
 44 //*****************************************************************************
 45 extern void _c_int00(void);
 46
 47 //*****************************************************************************
 48 //
 49 // Linker variable that marks the top of the stack.
 50 //
```

```
51  //*****************************************************************************
52  extern uint32_t __STACK_TOP;
53
54  //*****************************************************************************
55  //
56  // External declaration for the interrupt handler used by the application.
57  //
58  //*****************************************************************************
59  extern void PortAIntHandler(void);
60  extern void UARTIntHandler(void);
61  extern void UART2IntHandler(void);
62  extern void Timer0IntHandler(void);
63  extern void Timer1IntHandler(void);
64  extern void Timer2IntHandler(void);
65
66  //*****************************************************************************
67  //
68  // The vector table.  Note that the proper constructs must be placed on this to
69  // ensure that it ends up at physical address 0x0000.0000 or at the start of
70  // the program if located at a start address other than 0.
71  //
72  //*****************************************************************************
73  #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
74  void (* const g_pfnVectors[])(void) =
75  {
76      (void (*)(void))((uint32_t)&__STACK_TOP),
77                                              // The initial stack pointer
78      ResetISR,                               // The reset handler
79      NmiSR,                                  // The NMI handler
80      FaultISR,                               // The hard fault handler
81      IntDefaultHandler,                      // The MPU fault handler
82      IntDefaultHandler,                      // The bus fault handler
83      IntDefaultHandler,                      // The usage fault handler
84      0,                                      // Reserved
85      0,                                      // Reserved
86      0,                                      // Reserved
87      0,                                      // Reserved
88      IntDefaultHandler,                      // SVCall handler
89      IntDefaultHandler,                      // Debug monitor handler
90      0,                                      // Reserved
91      IntDefaultHandler,                      // The PendSV handler
92      IntDefaultHandler,                      // The SysTick handler
93      PortAIntHandler,                       // GPIO Port A
94      IntDefaultHandler,                      // GPIO Port B
95      IntDefaultHandler,                      // GPIO Port C
96      IntDefaultHandler,                      // GPIO Port D
97      IntDefaultHandler,                      // GPIO Port E
98      UARTIntHandler,                         // UART0 Rx and Tx
99      IntDefaultHandler,                      // UART1 Rx and Tx
100     IntDefaultHandler,                      // SSI0 Rx and Tx
101     IntDefaultHandler,                      // I2C0 Master and Slave
102     IntDefaultHandler,                      // PWM Fault
103     IntDefaultHandler,                      // PWM Generator 0
104     IntDefaultHandler,                      // PWM Generator 1
105     IntDefaultHandler,                      // PWM Generator 2
106     IntDefaultHandler,                      // Quadrature Encoder 0
107     IntDefaultHandler,                      // ADC Sequence 0
108     IntDefaultHandler,                      // ADC Sequence 1
109     IntDefaultHandler,                      // ADC Sequence 2
110     IntDefaultHandler,                      // ADC Sequence 3
111     IntDefaultHandler,                      // Watchdog timer
112     Timer0IntHandler,                       // Timer 0 subtimer A
113     IntDefaultHandler,                      // Timer 0 subtimer B
114     Timer1IntHandler,                       // Timer 1 subtimer A
115     IntDefaultHandler,                      // Timer 1 subtimer B
116     Timer2IntHandler,                      // Timer 2 subtimer A
117     IntDefaultHandler,                      // Timer 2 subtimer B
118     IntDefaultHandler,                      // Analog Comparator 0
119     IntDefaultHandler,                      // Analog Comparator 1
120     IntDefaultHandler,                      // Analog Comparator 2
121     IntDefaultHandler,                      // System Control (PLL, OSC, BO)
122     IntDefaultHandler,                      // FLASH Control
123     IntDefaultHandler,                      // GPIO Port F
124     IntDefaultHandler,                      // GPIO Port G
125     IntDefaultHandler,                      // GPIO Port H
126     UART2IntHandler,                       // UART2 Rx and Tx
127     IntDefaultHandler,                      // SSI1 Rx and Tx
128     IntDefaultHandler,                      // Timer 3 subtimer A
129     IntDefaultHandler,                      // Timer 3 subtimer B
130     IntDefaultHandler,                      // I2C1 Master and Slave
131     IntDefaultHandler,                      // CAN0
132     IntDefaultHandler,                      // CAN1
133     IntDefaultHandler,                      // Ethernet
134     IntDefaultHandler,                      // Hibernate
135     IntDefaultHandler,                      // USB0
```

```
136         IntDefaultHandler,                      // PWM Generator 3
137         IntDefaultHandler,                      // uDMA Software Transfer
138         IntDefaultHandler,                      // uDMA Error
139         IntDefaultHandler,                      // ADC1 Sequence 0
140         IntDefaultHandler,                      // ADC1 Sequence 1
141         IntDefaultHandler,                      // ADC1 Sequence 2
142         IntDefaultHandler,                      // ADC1 Sequence 3
143         IntDefaultHandler,                      // External Bus Interface 0
144         IntDefaultHandler,                      // GPIO Port J
145         IntDefaultHandler,                          // GPIO Port K
146         IntDefaultHandler,                      // GPIO Port L
147         IntDefaultHandler,                      // SSI2 Rx and Tx
148         IntDefaultHandler,                      // SSI3 Rx and Tx
149         IntDefaultHandler,                      // UART3 Rx and Tx
150         IntDefaultHandler,                      // UART4 Rx and Tx
151         IntDefaultHandler,                      // UART5 Rx and Tx
152         IntDefaultHandler,                      // UART6 Rx and Tx
153         IntDefaultHandler,                      // UART7 Rx and Tx
154         IntDefaultHandler,                      // I2C2 Master and Slave
155         IntDefaultHandler,                      // I2C3 Master and Slave
156         IntDefaultHandler,                      // Timer 4 subtimer A
157         IntDefaultHandler,                      // Timer 4 subtimer B
158         IntDefaultHandler,                      // Timer 5 subtimer A
159         IntDefaultHandler,                      // Timer 5 subtimer B
160         IntDefaultHandler,                      // FPU
161         0,                                      // Reserved
162         0,                                      // Reserved
163         IntDefaultHandler,                      // I2C4 Master and Slave
164         IntDefaultHandler,                      // I2C5 Master and Slave
165         IntDefaultHandler,                      // GPIO Port M
166         IntDefaultHandler,                      // GPIO Port N
167         0,                                      // Reserved
168         IntDefaultHandler,                      // Tamper
169         IntDefaultHandler,                      // GPIO Port P (Summary or P0)
170         IntDefaultHandler,                      // GPIO Port P1
171         IntDefaultHandler,                      // GPIO Port P2
172         IntDefaultHandler,                      // GPIO Port P3
173         IntDefaultHandler,                      // GPIO Port P4
174         IntDefaultHandler,                      // GPIO Port P5
175         IntDefaultHandler,                      // GPIO Port P6
176         IntDefaultHandler,                      // GPIO Port P7
177         IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
178         IntDefaultHandler,                      // GPIO Port Q1
179         IntDefaultHandler,                      // GPIO Port Q2
180         IntDefaultHandler,                      // GPIO Port Q3
181         IntDefaultHandler,                      // GPIO Port Q4
182         IntDefaultHandler,                      // GPIO Port Q5
183         IntDefaultHandler,                      // GPIO Port Q6
184         IntDefaultHandler,                      // GPIO Port Q7
185         IntDefaultHandler,                      // GPIO Port R
186         IntDefaultHandler,                      // GPIO Port S
187         IntDefaultHandler,                      // SHA/MD5 0
188         IntDefaultHandler,                      // AES 0
189         IntDefaultHandler,                      // DES3DES 0
190         IntDefaultHandler,                      // LCD Controller 0
191         IntDefaultHandler,                      // Timer 6 subtimer A
192         IntDefaultHandler,                      // Timer 6 subtimer B
193         IntDefaultHandler,                      // Timer 7 subtimer A
194         IntDefaultHandler,                      // Timer 7 subtimer B
195         IntDefaultHandler,                      // I2C6 Master and Slave
196         IntDefaultHandler,                      // I2C7 Master and Slave
197         IntDefaultHandler,                      // HIM Scan Matrix Keyboard 0
198         IntDefaultHandler,                      // One Wire 0
199         IntDefaultHandler,                      // HIM PS/2 0
200         IntDefaultHandler,                      // HIM LED Sequencer 0
201         IntDefaultHandler,                      // HIM Consumer IR 0
202         IntDefaultHandler,                      // I2C8 Master and Slave
203         IntDefaultHandler,                      // I2C9 Master and Slave
204         IntDefaultHandler                       // GPIO Port T
205     };
206
207     //*****************************************************************************
208     //
209     // This is the code that gets called when the processor first starts execution
210     // following a reset event.  Only the absolutely necessary set is performed,
211     // after which the application supplied entry() routine is called.  Any fancy
212     // actions (such as making decisions based on the reset cause register, and
213     // resetting the bits in that register) are left solely in the hands of the
214     // application.
215     //
216     //*****************************************************************************
217     void
218     ResetISR(void)
219     {
220         //
```

```
221         // Jump to the CCS C initialization routine.  This will enable the
222         // floating-point unit as well, so that does not need to be done here.
223         //
224         __asm("    .global _c_int00\n"
225               "    b.w     _c_int00");
226     }
227
228     //*****************************************************************************
229     //
230     // This is the code that gets called when the processor receives a NMI.  This
231     // simply enters an infinite loop, preserving the system state for examination
232     // by a debugger.
233     //
234     //*****************************************************************************
235     static void
236     NmiSR(void)
237     {
238         //
239         // Enter an infinite loop.
240         //
241         while(1)
242         {
243         }
244     }
245
246     //*****************************************************************************
247     //
248     // This is the code that gets called when the processor receives a fault
249     // interrupt.  This simply enters an infinite loop, preserving the system state
250     // for examination by a debugger.
251     //
252     //*****************************************************************************
253     static void
254     FaultISR(void)
255     {
256         //
257         // Enter an infinite loop.
258         //
259         while(1)
260         {
261         }
262     }
263
264     //*****************************************************************************
265     //
266     // This is the code that gets called when the processor receives an unexpected
267     // interrupt.  This simply enters an infinite loop, preserving the system state
268     // for examination by a debugger.
269     //
270     //*****************************************************************************
271     static void
272     IntDefaultHandler(void)
273     {
274         //
275         // Go into an infinite loop.
276         //
277         while(1)
278         {
279         }
280     }
```

E.10. main.h

```
 1   /*****************************************************************************
 2   **    File name     : main.h
 3   **    Hardware      : ARM Cortex M4 Basestation
 4   **    Date          : 14/05/2014
 5   **    Last Update   : 02/08/2015
 6   **    Author        : Nico Sassano
 7   **    Description   : main
 8   **    State         : Final state
 9   *****************************************************************************/
10   #ifndef MAIN_H_
11   #define MAIN_H_
12
13   #define HELP                    0x00
14   #define GET_VOLTAGE             0x01
15   #define BURST                   0x02
16   #define GET_BURST               0x03
17   #define GET_CONFIG_BS           0x04
18   #define SEND_HEX                0x05
```

```
19  #define SET_BRATE               0x06
20  #define CALI_BURST              0x07
21  #define CALI_CLK                0x08
22  #define MATLABMODE              0x09
23  #define LTC2376MODE             0x0A
24  #define AD7691MODE              0x0B
25  #define AD7176MODE              0x0C
26  #define SET_AD5270_BS           0x0D
27  #define SET_AD5270_ZS           0x0E
28  #define GET_CONFIG_ZS           0x0F
29  #define SET_PROPROCESSING       0x10
30  #define GET_GORTZEL             0x11
31  #define SEND_SPI                0x12
32  #define BURST_AUTO              0x13
33  #define WHY                     0xEE
34
35  #endif /* MAIN_H_ */
```

E.11. stimulus.h

```
 1  /*******************************************************************************
 2  **    File name     : stimulus.h
 3  **    Hardware      : ARM Cortex M4 Stimulus Controller
 4  **    Date          : 26/03/2016
 5  **    Last Update   : 12/05/2016
 6  **    Author        : Julian Akwaa
 7  **    Description   : Header for output and calculation routines
 8  **    State         : Final state
 9  *******************************************************************************/
10
11  #ifndef STIMULUS_H_
12  #define STIMULUS_H_
13
14  void initStimulus();
15  void runStimulus();
16  void setCurrent();
17  void calculate(int);
18  void adjustDC();
19  void adjustAC();
20  unsigned int getCh1CurrentInBit(long);
21  unsigned int getCh2CurrentInBit(long);
22  long getCh1CurrentInAmps(int);
23  long getCh2CurrentInAmps(int);
24  void disableAcOutput();
25  void disableOutput();
26
27  #define DEFAULT_DC 2
28  #define DEFAULT_AC 1
29  #define PI 3.141592653589793
30  #define DC_TOLERANCE 50
31  #define AC_TOLERANCE 100
32  #define CHANGE 400
33  #define MIN_CURRENT 0 //0A
34  #define MAX_CURRENT 65500 //
35  #define AC_OFFSET 23102 //! =~ 4A AC path
36  #define MIN_AMPLITUDE 0 //min value offset - û
37  #define MAX_AMPLITUDE 23102 //! =~ 4A
38
39  #endif /* STIMULUS_H_ */
```

E.12. interrupts.h

```
 1  /*******************************************************************************
 2  **    File name     : interrupts.h
 3  **    Hardware      : ARM Cortex M4 Basestation
 4  **    Date          : 16/05/2014
 5  **    Last Update   : 02/05/2016
 6  **    Author        : Nico Sassano, modified by Julian Akwaa
 7  **    Description   : interrupts
 8  **    State         : Final state
 9  *******************************************************************************/
10  #define SIN_POINTS 100      //length of the DAC sine
11
12  #ifndef INTERRUPTS_H_
13  #define INTERRUPTS_H_
14
```

```
15  void GDO2IntHandler(void);
16  void UARTIntHandler(void);
17  void Timer0IntHandler(void);
18  void Timer1IntHandler(void);
19
20  #endif /* INTERRUPTS_H_ */
```

E.13. adc.h

```
 1  /*****************************************************************************
 2  **    File name     : adc.h
 3  **    Hardware      : ARM Cortex M4 Basestation
 4  **    Date          : 14/09/2015
 5  **    Last Update   : 02/08/2015
 6  **    Author        : Nico Sassano
 7  **    Description   : ADC12
 8  **    State         : Final state
 9  *****************************************************************************/
10  #ifndef ADC_H_
11  #define ADC_H_
12
13  void init_adc(void);
14  uint32_t read_adc1(void);
15  uint32_t read_adc2(void);
16
17  #endif /* ADC_H_ */
```

E.14. gpio.h

```
 1  /*****************************************************************************
 2  **    File name     : gpio.h
 3  **    Hardware      : ARM Cortex M4 Basestation
 4  **    Date          : 08/08/2014
 5  **    Last Update   : 02/08/2015
 6  **    Author        : Nico Sassano
 7  **    Description   : convert
 8  **    State         : Final state
 9  *****************************************************************************/
10  #ifndef GPIO_H_
11  #define GPIO_H_
12
13  void GPIOinit(void);
14  void LED_test(void);
15
16  #define B4_ON       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0xFF)
17  #define B4_OFF      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0x00)
18
19  #define LED0_ON     GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0xFF)
20  #define LED0_OFF    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0x00)
21
22  #define LED1_ON     GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0xFF)
23  #define LED1_OFF    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0x00)
24
25  #define LED2_ON     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0xFF)
26  #define LED2_OFF    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0x00)
27
28  #define LED3_ON     GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0xFF)
29  #define LED3_OFF    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x00)
30
31  #define LED_ALL_ON  GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0xFF);\
32                      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0xFF);\
33                      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0xFF);\
34                      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0xFF)
35
36  #define LED_ALL_OFF GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0x00);\
37                      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0x00);\
38                      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0x00);\
39                      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0x00)
40
41  #endif /* GPIO_H_ */
```

E.15. spi.h

```
1   /*******************************************************************************
2   **    File name     : spi.h
3   **    Hardware      : ARM Cortex M4 Basestation
4   **    Date          : 06/06/2014
5   **    Last Update   : 02/08/2015
6   **    Author        : Nico Sassano
7   **    Description   : spi
8   **    State         : Final state
9   *******************************************************************************/
10  #ifndef SPI_H_
11  #define SPI_H_
12
13  void SPIinit(uint32_t, uint8_t, uint32_t);
14  void SPI_send_data(int length, uint32_t *);
15
16  #endif /* SPI_H_ */
```

E.16. uart.h

```
1   /*******************************************************************************
2   **    File name     : uart.h
3   **    Hardware      : ARM Cortex M4 Basestation
4   **    Date          : 16/05/2014
5   **    Last Update   : 02/08/2015
6   **    Author        : Nico Sassano
7   **    Description   : uart
8   **    State         : Final state
9   *******************************************************************************/
10  #ifndef UART_H_
11  #define UART_H_
12
13  void UARTReceive();
14  void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count);
15  void UARTinit(uint32_t,uint32_t,uint32_t);
16  void UARTinit_FUELCON(uint32_t,uint32_t,uint32_t);
17  void UARTSend_FUELCON(const uint8_t *, uint32_t );
18
19  #endif /* UART_H_ */
```

E.17. timer.h

```
1   /*******************************************************************************
2   **    File name     : timer.h
3   **    Hardware      : ARM Cortex M4 Basestation
4   **    Date          : 16/05/2014
5   **    Last Update   : 02/08/2015
6   **    Author        : Nico Sassano
7   **    Description   : timer
8   **    State         : Final state
9   *******************************************************************************/
10  #ifndef TIMER_H_
11  #define TIMER_H_
12
13  #define PERIOD1000ms    1
14  #define PERIOD500ms     2
15  #define PERIOD250ms     4
16  #define PERIOD125ms     5
17  #define PERIOD64_5ms    6
18  #define PERIOD32_25ms   7
19  #define PERIOD16_125ms  8
20  #define PERIOD8_645ms   9
21  #define PERIOD4_32125ms 10
22
23  void TIMER0init(void);
24  void TIMER1init(void);
25  void TIMER2init(void);
26  void TIMER0config_ms(uint32_t);
27  void TIMER0config_us(uint32_t);
28  void TIMER1config(uint32_t);
29  void TIMER1config_LTC2376(uint32_t);
30  void TIMER1config_AD7691(uint32_t);
31  void TIMER1config_AD7176(uint32_t);
32  void TIMER2config(uint32_t);
33  void delay(unsigned long);
34  void delay_ms(uint32_t);
35  void delay_us(uint32_t);
```

```
36  void timeout_start(uint32_t ms);
37  void timeout_stop(void);
38
39
40  /*****************************************************
41  ** Timeout IRQ Macros
42  *****************************************************/
43  #define TIMEOUT_SET                 (irq_timeout = 1)
44  #define TIMEOUT_IS_SET              (irq_timeout == 1)
45  #define TIMEOUT_UNSET               (irq_timeout = 0)
46  #define TIMEOUT_IS_UNSET            (irq_timeout == 0)
47
48  #define TIMER_MODE_SET_DELAY        irq_timer0_mode = DELAY_MODE
49  #define TIMER_MODE_SET_TIMEOUT      irq_timer0_mode = TIMEOUT_MODE
50  #define TIMER_MODE_IS_DELAY         irq_timer0_mode == DELAY_MODE
51  #define TIMER_MODE_IS_TIMEOUT       irq_timer0_mode == TIMEOUT_MODE
52
53  #define TIMER1_MODE_SET_BURST       irq_timer1_mode = BURST_MODE
54  #define TIMER1_MODE_SET_LTC2376     irq_timer1_mode = LTC2376_MODE
55  #define TIMER1_MODE_SET_AD7691      irq_timer1_mode = AD7691_MODE
56  #define TIMER1_MODE_SET_AD7176      irq_timer1_mode = AD7176_MODE
57  #define TIMER1_MODE_IS_BURST        irq_timer1_mode == BURST_MODE
58  #define TIMER1_MODE_IS_LTC2376      irq_timer1_mode == LTC2376_MODE
59  #define TIMER1_MODE_IS_AD7691       irq_timer1_mode == AD7691_MODE
60  #define TIMER1_MODE_IS_AD7176       irq_timer1_mode == AD7176_MODE
61
62  #define TIMER0_START                ROM_TimerEnable(TIMER0_BASE, TIMER_A)
63  #define TIMER0_STOP                 ROM_TimerDisable(TIMER0_BASE, TIMER_A)
64
65  #define TIMER1_START                ROM_TimerEnable(TIMER1_BASE, TIMER_A)
66  #define TIMER1_STOP                 ROM_TimerDisable(TIMER1_BASE, TIMER_A)
67
68  #define TIMER2_START                ROM_TimerEnable(TIMER2_BASE, TIMER_A)
69  #define TIMER2_STOP                 ROM_TimerDisable(TIMER2_BASE, TIMER_A)
70
71  /*****************************************************
72  ** Interrupt Modes
73  *****************************************************/
74  #define DELAY_MODE                                  0x00
75  #define TIMEOUT_MODE                                0x01
76  #define BURST_MODE                                  0x02
77  #define LTC2376_MODE                                0x03
78  #define AD7691_MODE                                 0x04
79  #define AD7176_MODE                                 0x05
80
81
82
83  #endif /* TIMER_H_ */
```

E.18. convert.h

```
 1  /*****************************************************************************
 2  **    File name     : convert.h
 3  **    Hardware      : ARM Cortex M4 Basestation
 4  **    Date          : 14/09/2015
 5  **    Last Update   : 02/08/2015
 6  **    Author        : Nico Sassano
 7  **    Description   : convert
 8  **    State         : Final state
 9  *****************************************************************************/
10  #ifndef CONVERT_H_
11  #define CONVERT_H_
12
13  uint8_t char_to_int(char);
14  uint8_t char_to_int2(char, char);
15  uint8_t char_to_int3(char, char, char);
16  uint8_t char_to_hex(char, char);
17  uint16_t char_to_hex3(char, char, char);
18
19  #endif /* CONVERT_H_ */
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 8. Juni 2016	
Ort, Datum	Unterschrift