Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Torsten Geist

Entwicklung eines embedded Einplatinensteuersystems für optische Batterieuntersuchung

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Torsten Geist

Entwicklung eines embedded Einplatinensteuersystems für optische Batterieuntersuchung

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. Franz Schubert

Abgegeben am 19. September 2015

Torsten Geist

Thema der Bachelorthesis

Entwicklung eines embedded Einplatinensteuersystems für optische Batterieuntersuchung

Stichworte

Lithium-Ionen-Batterie, Zyklierung, Bildaufnahme, Linux, SPI, Raspberry-Pi

Kurzzusammenfassung

Diese Arbeit stellt die Entwicklung und den Aufbau eines eingebetteten Messsystems für die Erforschung von Lithium-Ionen-Batterien vor. Deren Elektroden sollen mit einer Mikroskopkamera auf die Veränderung ihres optischen Spektrums während Lade- und Entladevorgang untersucht werden. Diese Erkenntnisse sollen genaue Aussagen über deren Ladezustand ermöglichen.

Torsten Geist

Title of the paper

Development of an embedded One-Board-Control-System for optical battery research

Keywords

Lithium-Ion-Battery, cycling, image acquisition, Linux, SPI, Raspberry-Pi

Abstract

This bachelor thesis deals with the specification and development of a measurement system for optical and electrical observation of lithium-ion-batteries for research purposes. The lithium batteries' electrodes are to be observed during charge and discharge using a microscope camera. The results from these measurements can be used to determine the State of Charge of the battery.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Motivation

Eines der größten Probleme der Elektrotechnik ist, die sehr begrenzte Möglichkeit elektrische Energie zu speichern. Deshalb geschieht dies oft in umgewandelter Form. Zum Beispiel in Pumpspeicherkraftwerken, welche sich dazu eignen, Bedarfsspitzen in der Netzversorgung abzudecken bzw. Überproduktion von Grundlastkraftwerken aufzunehmen. Deren abfließendes Wasser hat zwar eine geeignete Energiedichte, sie sind jedoch sehr von geographischen Gegebenheiten abhängig und können daher nicht als flächendeckende Lösung dienen. Es wäre wünschenswert, einen Energiespeicher zu haben, den man überall hinstellen und je nach Bedarf laden und entladen könnte. Dies würde sich positiv auf den Wirkungsgrad unserer Energieversorgung auswirken, da Kraftwerke kontinuierlich in ihren effektivsten Arbeitspunkten betrieben werden könnten.
Ein noch größeres Problem stellt die Speicherung von elektrischer Energie im Bereich der Elektromobilität dar. Kraftfahrzeuge sind darauf angewiesen, ihre Energie mit sich zu führen. Ein Ansatz dazu ist die Brennstoffzelle. Der Wasserstoff, mit der diese betrieben wird, ist nichts anderes als umgewandelte elektrische Energie, welche im Tank mitgeführt wird. Dieses Konzept hat sich jedoch, wohl aus wirtschaftlichen Gründen, bisher nicht durchsetzen können. Denn die Infrastruktur für die Wasserstoffversorgung ist in Deutschland so gut wie nicht vorhanden. Das Stromnetz hingegen kann als flächendeckend bezeichnet werden, was nahe legt, einen Akkumulator für elektrische Energie zu verwenden. Dazu standen lange Zeit nur Blei-Akkumulatoren zur Verfügung. Diese weisen zwar eine relativ hohe Energiedichte und geringe Verluste im Betrieb auf, sind jedoch sehr schwer. Deshalb waren sie bisher nur eine Option für Flurförderfahrzeuge, welche aus der Not eine Tugend machen, indem sie die Masse der Akkumulatoren als Kontermasse zur Nutzlast verwenden.
Abhilfe verspricht der Lithium-Ionen-Akkumulator, welcher neben einem geringeren Gewicht auch eine hohe Energiedichte mitbringt. Jedoch ist es bislang noch sehr schwer, zuverlässige Aussagen über den Gesundheitszustand (SoH[1]) und den Ladezustand (SoC[2]) zu machen. Da der SoH sich bei Nutzung des Akkumulators stark ändert und der SoC direkt vom SoH abhängt, müsste man, um z.B. die Restreichweite eines Elektrobusses vorherzusagen, beide im laufenden Betrieb exakt bestimmen können. Diese Bachelorthesis ist Teil einer Untersuchung, deren Ergebnisse dies in Zukunft ermöglichen sollen [35] [38] [34].

[1] SoH - State of Health
[2] SoC - State of Charge

2. Recherche

2.1. Projektzielstellung

Trotz ihrer vielversprechenden Eigenschaften können Lithium-Batterien noch nicht mit fossilen Treibstoffen konkurrieren. Deshalb wird intensiv an ihnen geforscht. An der HAW wurde hierzu, im Rahmen der Graduiertenschule „Key Technologies for Sustainable Energy Systems in Smart Grids“, ein Forschungsprojekt eingerichtet, in dessen Zuge diese und viele andere Bachelor- und Masterarbeiten entstanden sind. Ziel der Forschung ist, die Leistungsfähigkeit der Batterien zu erhöhen, ihre Zuverlässigkeit zu gewährleisten, die Lebensdauer zu steigern und die Kosten zu reduzieren.

Um die Zuverlässigkeit eines Akkumulators gewährleisten zu können, ist es von entscheidender Bedeutung, eine exakte Aussage machen zu können, wie viel Energie dieser im Moment noch liefern kann. Dazu wird u.a. die Zellspannung gemessen und versucht, aus dieser einen Rückschluss auf den SoC zu ziehen. Die Zellspannung variiert über einen kompletten Lade-/Entladevorgang (ohne Tiefentladung) über weniger als ein Volt. Daher ist eine exakte Auflösung der Abbildung des Energiegehaltes über die Spannung kaum möglich. Eine andere Möglichkeit stellt das Integrieren des Stromes über die Zeit dar. Beide Methoden sind jedoch nicht in der Lage, den SoH des Akkumulators zu berücksichtigen. Hinzu kommt die starke Temperaturabhängigkeit des Akkumulators. Ein ganz neuer Ansatz, dieses Problem zu lösen, ist die Auswertung des optischen Spektrums der Elektroden. Mit diesem Ansatz möchte man erreichen, ein direktes Messverfahren, d.h. ohne eine zeitliche Integration, zu entwickeln. Die Möglichkeiten, solche Bilder auszuwerten, werden in den Abschlussarbeiten von Grießbach [14] und Palliyaguruge [28] behandelt. In dieser Abschlussarbeit geht es um die Aufzeichnung dieser Bilder und die Bedingungen, denen die Zelle während der Untersuchung ausgesetzt ist [14] [28].

2.2. Detaillierung der Mess-, Erfassungs- und Steuergrößen

Das vorrangige Ziel des Versuchsaufbaus ist die Aufnahme von Farbbildern der Elektrode. Damit diese jedoch aussagekräftig und die Ergebnisse reproduzierbar sind, müssen auch die Bedingungen, unter denen die Bilder entstanden sind, einstellbar sein und mit aufgezeichnet werden. Dazu gehören die Zellspannung, der Lade-/Entladestrom, die Umgebungstemperatur der Zelle und vor allem die Beleuchtung der Zelle. Da die Elektrode nach ihrem optischen Spektrum untersucht werden soll (siehe [14] [28]), muss gewährleistet werden, dass die Beleuchtung während einer Untersuchung konstant ist und bei Bedarf auch mal in ihrem Spektrum kontrolliert variiert werden kann. Tageslicht kommt demnach nicht in Frage. Für die Zellspannung soll ein Bereich von ca. $U_Z = 0V$ bis $U_Z = 4,5V$ einstellbar sein um sowohl den Normalbetrieb als auch die Tiefentladung untersuchen zu können. Der Lade-/Entladestrom soll ein einstellbarer und geregelter Konstantstrom in einem Bereich von $I_Z = 1\mu A$ bis $I_Z = 1mA$ sein.

2.3. Vorhandene Hard- und Software

Die Abschlussarbeit knüpft an die Arbeit von Grießbach [14] an. Dieser hatte die Zyklierung mit einer Relaiskarte realisiert. Die Relaiskarte wurde mit einem extra dafür geschriebenen Programm, dem „Relais-Card-Controller“, gesteuert. Der „Relais-Card-Controller“ durchläuft ein Steuerskript, in dem ein Zyklierplan vorgegeben wird. Da die Auswertung dieses Skriptes durch den „Relais-Card-Controller“ auch eine Schleifenfunktionalität unterstützt, können leicht Zyklierpläne über einen Zeitraum von mehreren Monaten vorgegeben werden. Diese orientieren sich jedoch ausschließlich an der Zeit. Eine Istwert-Abfrage erfolgt mit dem ebenfalls extra entwickelten Programm „DMM4020 Reader“, welches über eine serielle Schnittstelle ein Präzisionsmultimeter ausliest und die gemessene Spannung, zusammen mit einem Zeitstempel, in einer Datei speichert. Eine Auswertung zur Beeinflussung der Zyklierung erfolgt jedoch nicht. Zusätzlich zur Spannungsmessung wird ein zweites Präzisionsmultimeter, ebenfalls durch den „DMM4020 Reader“ ausgelesen, zur Aufzeichnung des Stroms verwendet. Die Beobachtung der Elektrode erfolgt mit einer Mikroskopkamera, gesteuert von dem Open-Source-Programm „GUVCView“. Die zu beobachtende Zelle ist ein Sonderbau, der eigens an der HAW hergestellt wird und über ein Fenster den Blick auf die Elektrode zulässt. Zur Zeit werden für den Aufbau des Messplatzes zur Untersuchung von vier Lithium-Ionen-Zellen sechs PC benötigt. Zwei davon übernehmen, mit Hilfe eines Multiplexers, die elektrischen Messungen und deren Steuerung. Die vier anderen übernehmen die Verarbeitung der Bilderfassung. Im Laufe der Übernahme des Projektes wurde diese Aufteilung damit begründet, dass die Bilderfassung im Hinblick auf die zum Teil sehr langen Laufzeiten instabil seien.

Es könne z.B. die Speicherkapazität der Festplatte überschritten werden. In dem Fall würde die Steuerung ausfallen und die Zelle könne tiefentladen und somit zerstört werden [14].

2.4. Vergleichsuntersuchung und Wahl eines geeigneten Einplatinen-Computers

Zwei der Nachteile des bisherigen Messaufbaus sollen durch den Austausch der PC gegen Einplatinen-Computer beseitigt werden. Diese brauchen deutlich weniger Platz und Energie. In diesem Kapitel geht es um die Auswahl eines geeigneten Einplatinen-Computers. Dazu werden zunächst die technischen Daten verglichen und anschließend praktische Tests durchgeführt.

2.4.1. Vergleich der technischen Daten

Der Einplatinen-Computer soll in der Lage sein, alle Messaufgaben und die Aufnahme der Bilder von den Zellelektroden lückenlos und über lange Zeiträume zu bewältigen. Dabei wird die Bildbearbeitung die größten Anforderungen stellen. Deshalb liegt die erste Priorität bei der Leistung der CPU. Da die Daten auch zuverlässig und schnell auf einer Festplatte gespeichert werden müssen, wurde bei der Suche nach einem geeigneten Einplatinen-Computer auch nach einem S-ATA-Anschluss Ausschau gehalten. Ein weiterer Wunsch war ein Analog-Digital-Wandler um mit diesem die Spannungsmessung durchzuführen und ein Messgerät einzusparen. Nicht zuletzt sollte der Einplatinen-Computer nicht mehr als 100,-€ kosten. Die Tabelle 2.1 auf Seite 13 stellt die engere Auswahl gegenüber. Das Beagle-Board war lange Zeit der leistungsstärkste Einplatinen-Computer seiner Preisklasse. Mittlerweile sind die günstigeren Varianten wie der PC-Duino jedoch genauso leistungsstark. Der PC-Duino verfügt darüber hinaus als einziger über einen S-ATA-Anschluss und einen ADC. Der Raspberry-Pi 2 bringt die CPU mit der höchsten Rechenleistung mit. Da der Raspberry-Pi 1 keine konkurrenzfähige CPU besitzt, bleiben nur der PC-Duino und der Raspberry-Pi 2 um in praktischen Tests verglichen zu werden [16] [23] [11] [37].

2.4.2. Test des PcDuino

Der PcDuino wurde zuerst getestet. Das Bild 2.1 auf Seite 14 zeigt das Ergebnis eines 26-tägigen Langzeittest. In diesem Test wurden die Daten über die S-ATA-Schnittstelle auf die Festplatte geschrieben. Während des Tests liefen zwei Prozesse des „DMM4020 Reader“ für Strom- und Spannungsmessung, ein Prozess des „Relais-Card-Controller“ und ein Prozess

	R.-Pi 1	R.-Pi 2	pcDuino	Beagle-B. Black
CPU	1x700MHz	4x900MHz	2x1GHz	1x1GHz
RAM	512MB	1024MB	1024MB	512MB DDR3
Speicher	64GB	64GB	4+64GB	4+64GB
S-ATA	-	-	✓	-
LAN	✓	✓	✓	✓
HDMI	✓	✓	✓	✓
USB	4	4	2	2
Schnittst.	CSI, DSI	CSI, DSI	CSI, IR	LCD
I/O	40	40	32	2x46
SPI	✓	✓	✓	✓
I2C	✓	✓	✓	✓
UART	✓	✓	✓	✓
ADC	-	-	✓	-
Preis	<100,-€	<100,-€	<100,-€	<100,-€

Tabelle 2.1.: Vergleich von Einplatinen-Computern [16] [23] [11] [37]

des Bildverarbeitungsprogramms „fswebCam“. Die Kamera wurde an einen der zwei USB-Ports des "pcDuino“ und die anderen Geräte an einen aktiven HUB angeschlossen , welcher den zweiten USB-Port belegte. Für die Auswertung sind die Matlab-Skripte A.4.1 auf Seite 123 für das Bild 2.1 auf Seite 14 und Skript A.4.2 auf Seite 124 für die genauere Betrachtung des Ergebnisses verwendet worden. Für den Test wurden die Zeitstempel, die in die Namen der Bilddateien eingefügt worden waren, ausgewertet. Diese wurden mit Matlab separiert und die Differenz zwischen den Zeitstempeln als Plot dargestellt. Der zeitliche Abstand der Bilder sollte nicht mehr als zehn Sekunden betragen. Der größte Abstand beträgt jedoch über eine halbe Stunde. Darüber hinaus offenbarte ein Blick auf den Systemmonitor, dass die CPU von Bildaufnahme zu Bildaufnahme zwischen 80% und 100% Auslastung alternierte. Hier ein Überblick über das Testergebnis.

- Gesamtdauer = 25,8 Tage
- $\sum$ Fehlzeiten = 0,5 Tage
- Fehlzeiten in Prozent = 1,937 %
- Größter Zeitraum ohne Aufzeichnung = 0,6 Stunden

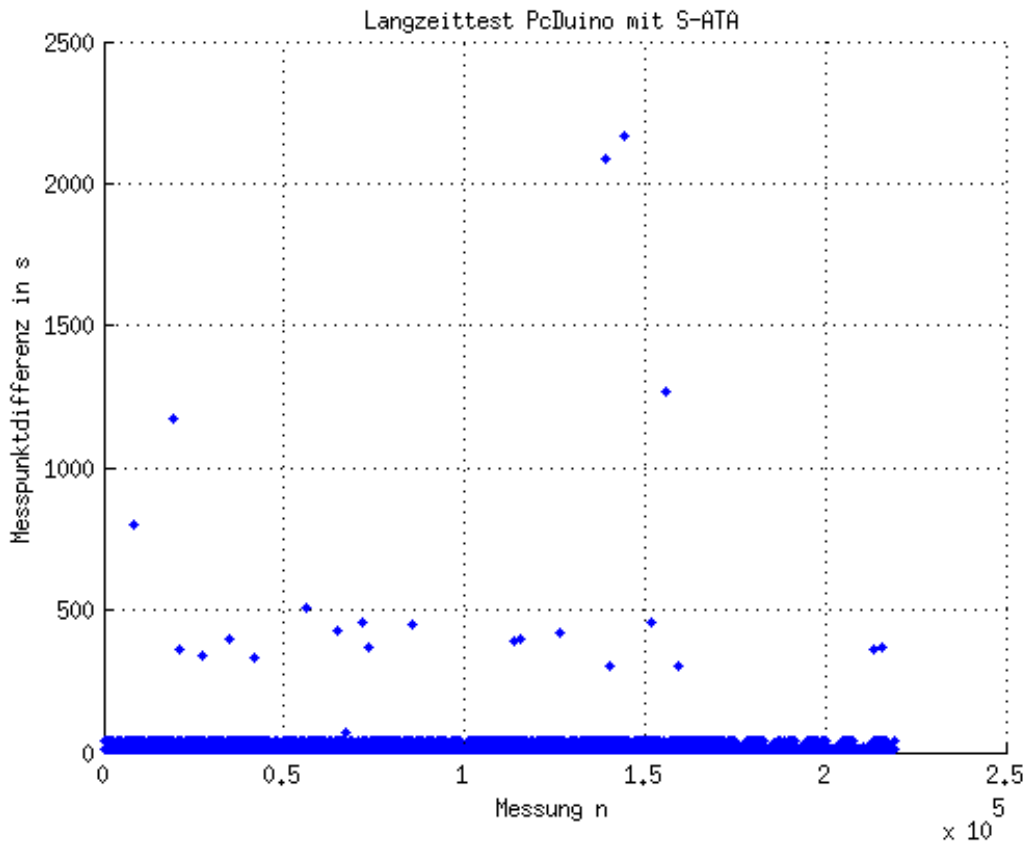

Abbildung 2.1.: Testergebnis des PcDuino: Dargestellt wird die Zeitdifferenz zwischen zwei Bildaufnahmen. Die meisten Differenzen sind nicht größer als 40 Sekunden (breites Band unten), welche noch valide Ergebnisse liefern würden. Es gibt jedoch einige Ausreißer um die 500 bis über 2000 Sekunden. Diese machen die Messung unbrauchbar.

2.4.3. Ergebnis des Vergleichs

Die Zuverlässigkeit, die der pcDuino im Test gezeigt hat, ist leider nicht ausreichend. Ganz besonders die Aussetzer mit einer Dauer von 8-30 Minuten (siehe Bild 2.1) sind ein Ausschlusskriterium. Es konnte zudem beobachtet werden, dass die CPU quasi unter Volllast lief. Selbst wenn sich herausstellen sollte, dass periphere Hardware an dem Performance-Problem beteiligt war, so würde dieser Punkt bleiben. Die Entscheidung viel deshalb schon an dieser Stelle auf den Raspberry-Pi 2, mit der deutlich stärkeren CPU.

2.5. Der Raspberry-Pi 2

Der Raspberry-Pi, der in diesem Projekt verwendet wird, gehört bereits zur zweiten Generation. Die erste Version wurde im Jahr 2006 von der gemeinnützigen Organisation „Raspberry-Pi Foundation“ in Kooperation mit der Firma „Acorn Computers Ltd.“ entwickelt. Ziel war eine Rechnerplattform, die nicht viel kostet und einfach zu verstehen ist. Diese sollte Ausbildungszwecken dienen, da die PC immer komplizierter wurden und, auch wenn das Preisleistungsverhältnis stetig besser wurde, verhältnismäßig teuer waren. Die „Raspberry-Pi Foundation“ wurde von ehemaligen Studenten der Universität Cambridge gegründet, von denen einer mittlerweile Mitarbeiter der Firma „Broadcom“ war. Diese produziert in erster Linie Multimediaprozessoren für Tablets und Smartphones sowie Datenübertragungssysteme wie z.B. Modems und Bluetooth-Chips. Exklusiv für den Raspberry-Pi entwickelte „Broadcom“ den Prozessor „bcm2836“. Dabei handelt es sich um einen Multimediaprozessor, der nicht frei gehandelt und nur auf dem Raspberry-Pi verbaut wird. Hergestellt wird dieser von den Firmen „Samsung“ und „Hynix“ [3] [37] [11].

2.5.1. Der Multimediaprozessor „bcm2836“

Der Multimediaprozessor „bcm2836“ ist Herzstück der zweiten Generation des Raspberrys. Sein Vorgänger, der Multimediaprozessor „bcm2835“, unterscheidet sich in erster Linie durch die CPU. Der Multimediaprozessor „bcm2836“ verfügt über eine „ARM Cortex-A7 CPU“. Er hat im Gegensatz zum Vorgänger einen doppelt so großen RAM[1] von 1GB, vier Kerne mit 900MHz und einen größeren L2-Cache von 1MB. Ansonsten haben die Multimediaprozessoren „bcm2835“ und „bcm2836“ keine nennenswerten Unterschiede. Auch die C-Treiber-Bibliothek, welche in dieser Arbeit verwendet wurde und ursprünglich für den „bcm2835“ gedacht war, ist absolut kompatibel.
Der „bcm2836“ ist ein Multimediaprozessor. Das bedeutet, dass er neben seiner CPU u.a. auch einen SDRAM[2] und eine GPU[3] integriert hat. Man spricht auch von einem SoC[4]. Die GPU befähigt ihn, Videofilme in HD-Qualität zu verarbeiten und über eine HDMI-Schnittstelle auszugeben. Der „bcm2836“ übernimmt auch die Funktion eines USB-Controllers. Unterstützt wird er dabei von dem Controller-Chip „LAN9514“. Bei diesem IC[5] handelt es sich um einen USB-Hub und einen Ethernet-Controller. Das bedeutet, dass sich Ethernet und USB die Bandbreite des Controller-Chip „LAN9514“ teilen [3] [37] [11].

[1] RAM - Random-Access Memory
[2] SDRAM - Synchronous Dynamic Random Access Memory
[3] GPU - Graphics Processing Unit
[4] SoC - System on Chip
[5] IC - Integrated Circuit

2.5.2. Die Portausgabe GPIO

Eine in diesem Projekt häufig genutzte Komponente des Multimediaprozessors „bcm2836“ ist die Portausgabe GPIO[6]. Diese führt Pins direkt nach draußen, wo diese zunächst als einfache Bit zur Verfügung stehen. Leider macht das den Raspberry anfällig auf Schäden durch ESD[7], denn es gibt auf der Platine keinerlei diskrete Schutzbeschaltung. Die Bit können sowohl als Eingänge als auch Ausgänge deklariert werden. Dabei muss darauf geachtet werden, dass angelegte Spannungen nicht größer als 3,3V werden.
Neben ihrer Nutzung als I/O-Bit übernehmen einige Pins auch Sonderfunktionen. Es werden die Schnittstellen I2C, SPI und UART[8] ohne Kontrollsignal für die Kommunikation mit anderen Bauteilen angeboten. Außerdem gibt es die Möglichkeit, eine Puls-Weiten-Modulation zu nutzen. Die GPIO-Schnittstelle ist kaum in der Lage, Energie zu liefern. Es wird empfohlen, einen Gesamtstrom von 16mA nicht zu überschreiten. Das schließt die 5V und 3,3V Spannungsversorgungspins mit ein [3] [37] [11] [9].

2.6. Linux als OS

Neben einem geeignetem Einplatinen-Computer, musste auch ein passendes Betriebssystem gefunden werden. Auf den bisher genutzten PC wurde die Linux-Distribution „Linux Mint“ installiert. Ob das auch für den Raspberry-Pi 2 und die Aufgabe, die er übernehmen soll, die richtige ist, soll in diesem Kapitel untersucht werden.
„Linux Mint“ ist ein Debian/Ubuntu-Derivat [36]. Es wurde in erster Linie für Desktop-PC entwickelt. Die Installation auf dem Raspberry-Pi 2 läuft problemlos und es wurde beim bisherigen Messbetrieb kein instabiles Verhalten festgestellt. Eigentlich könnte man die Suche an dieser Stelle beenden, wenn es nicht eine starke Konkurrenz gäbe.
Die Linux-Distribution „Raspbian“ [31] ist ebenfalls ein Debian-Derivat, jedoch speziell an den Raspberry-Pi (in jeglicher Version) angepasst. Man kann deshalb davon ausgehen, das Raspbian eher besser als schlechter auf dem Raspberry-Pi 2 läuft, als andere Distributionen. Hinzu kommt, dass die Community für Raspbian größer und vor allem auf „Bastler“ eingestellt ist. Es dürfte wesentlich leichter sein, für anfallende Probleme einen Blog oder Forums-Eintrag zu finden. Es ist zum Beispiel so, dass es für die Ansteuerung der GPIO (siehe Kapitel 4.4) bereits eine Bibliothek gibt, welche in erster Linie unter Raspbian getestet wurde [26]. Da der Raspberry auf jeden Fall um eine Platine erweitert werden muss, um die Zellzyklierung steuern zu können, wird das noch eine entscheidende Rolle spielen.

[6] GPIO - General Purpose Input Output
[7] ESD - Electrostatic Discharge
[8] UART -Universal Asynchronous Receiver Transmitter

Insgesamt überwiegen die Argumente für Raspbian. Es erledigt die bisherigen Aufgaben mindestens genauso gut wie „Linux-Mint“ und verspricht für zukünftige Anforderungen besser gewappnet zu sein. Die Entscheidung fällt somit ganz klar auf Raspbian [36] [31] [26].

2.7. Konzepterarbeitung

Bisher wurde geklärt, welcher Effekt untersucht werden soll und wie dies bis jetzt geschah. Es wurde auch bereits geklärt, dass die PC gegen einen Raspberry-Pi 2 mit dem Betriebssystem Raspbian ausgetauscht werden sollen. Jetzt gilt es noch zu klären, was an der Messung selbst geändert werden soll.

2.7.1. Pro und Contra des bisherigen Messaufbaus

Hier sind die Vor- und Nachteile, die beibehalten bzw. verbessert werden sollen, zusammenfassend aufgelistet [14].

Pro - Die Erfassung der Messdaten erfolgt zuverlässig.

Pro - Beliebig lange Zyklierpläne lassen sich mit wenig Aufwand realisieren.

Pro - Es gibt ein Programm welches den Zyklierplan auf seine Validität prüft.

Pro - Der Messaufbau ist sehr Modular.

Pro - Es muss keine eigene Hardware gebaut werden.

Pro - Die teuren Komponenten (z.B. Messgeräte) sind auch anderweitig einsetzbar.

Contra - Die Zyklierung orientiert sich nicht an dem Zustand der Zelle.

Contra - Der Lade-/Entladestrom lässt sich nur grob beeinflussen.

Contra - Der Messaufbau benötigt viel Platz und Material.

Contra - Der Messaufbau benötigt viel Energie.

Das Messsystem läuft sehr stabil und es lassen sich lange Zyklierpläne realisieren. Es ist jedoch nicht möglich, den Ladestrom auf einem konstanten Wert zu halten. Lediglich ein Widerstand sorgt dafür, dass ein Maximalstrom nicht überschritten wird. Der Ladestrom selbst ist jedoch immer auch vom Ladezustand der Zelle abhängig. Es wäre wünschenswert den Lade-/Entladestrom besser kontrollieren zu können.
Da eine Zelle nicht beliebig tief entladen werden kann, ohne das diese zerstört wird, sollte

die permanente Überwachung der Zellspannung dafür verwendet werden, die Zyklierung mit der Vorgabe von oberer und unterer Ladeschwelle zu beeinflussen. Dies würde auch detailliertere Untersuchungsmöglichkeiten zum Verhalten der Zelle bei dem Betrieb zwischen unterschiedlichen Spannungsgrenzen ermöglichen [14].
Das Energiebedarfs- und Platzproblem wurde bereits mit der Entscheidung zur Verwendung von Einplatinen-Computern gelöst. Für die Lösung der übrigen Nachteile muss ggf. auf die Vorteile der Modularität, des geringen Selbstbauaufwandes und die vielseitige Verwendung der Hardware verzichtet werden.

2.7.2. Betriebskonzept der Messung

Für die Zellzyklierung soll es eine über einen Zyklierplan kontrollierte Steuerung, sowie eine permanente Überwachung der Zellspannung und des Lade-/Entladestroms, geben. Für die Spannung soll eine Unter- und eine Obergrenze vorgegeben werden. Auch der Lade-/Entladestrom soll einstellbar sein und bei einer sich ändernden Zellspannung an den Sollwert angepasst werden.
Die Beobachtung der Elektrode soll mit einer Mikroskopkamera erfolgen. Um eine genaue Untersuchung des optischen Spektrums zu ermöglichen soll die Beleuchtung mit RGB-LED[9] erfolgen. Die einzelnen Farben sollen separat dimmbar sein. Da das Verhalten und die Lebenserwartung der Zelle stark temperaturabhängig ist, soll auch die Umgebungstemperatur der Zelle aufgezeichnet werden. Sämtliche Funktionen sollen mit einer Erweiterungsplatine für den Raspberry-Pi 2 realisiert werden. Für die Kommunikation zwischen Raspberry-Pi 2 und Platine soll die GPIO-Schnittstelle verwendet werden. Die parallele Nutzung der Präzisionsmultimeter mit der Anwendung „DMM4020 Reader“ soll weiter möglich sein. Das Bild 2.2 auf Seite 19 gibt einen Überblick über die Hardware. Die benötigten Schnittstellen, um alle Komponenten anzubinden, werden mit Bild 2.3 auf Seite 20 dargestellt.
Die Messdaten sollen im Zehn-Sekunden-Takt erfasst und abgespeichert werden. Dabei soll ein gespeichertes Elektroden-Bild das Ergebnis einer Integration über 5-10 Bilder sein. Aus den Bildern soll ein Video entstehen, welches eine Zeitrafferfunktion über die Zyklierung darstellt. Das kann jedoch nach der Erfassung der Messdaten geschehen.

[9] RGB-LED - Rot-Grün-Blau-Light-Emitting-Diode

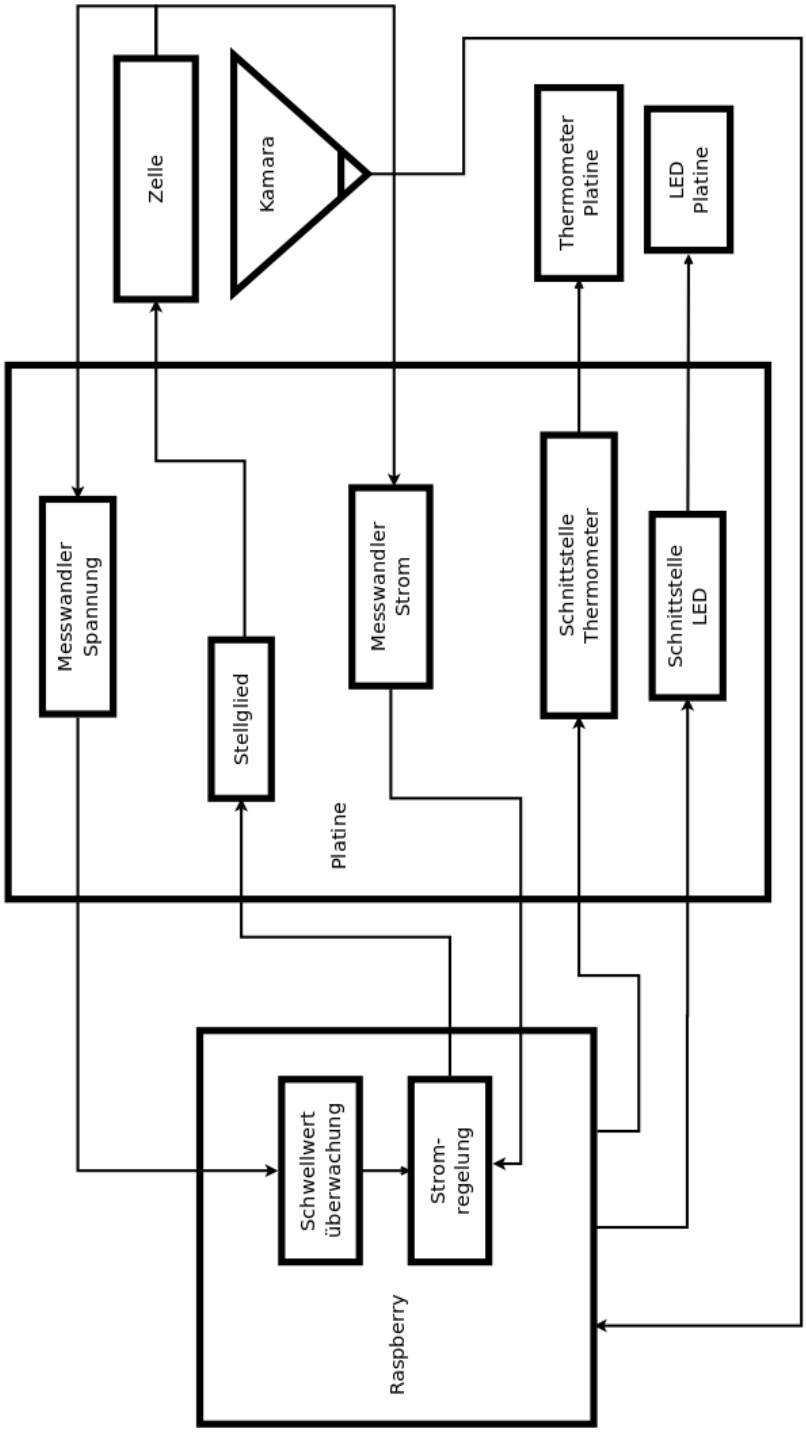

Abbildung 2.2.: Hardware Konzept des Messsystems - Auf dem Raspberry laufen die Schwellwert- und Stromüberwachung sowie die Bildverarbeitung. Auf der Erweiterungsplatine befinden sich die Messwandler, das Stellglied und die Schnittstellen zur Temperaturmessung und der Beleuchtung.

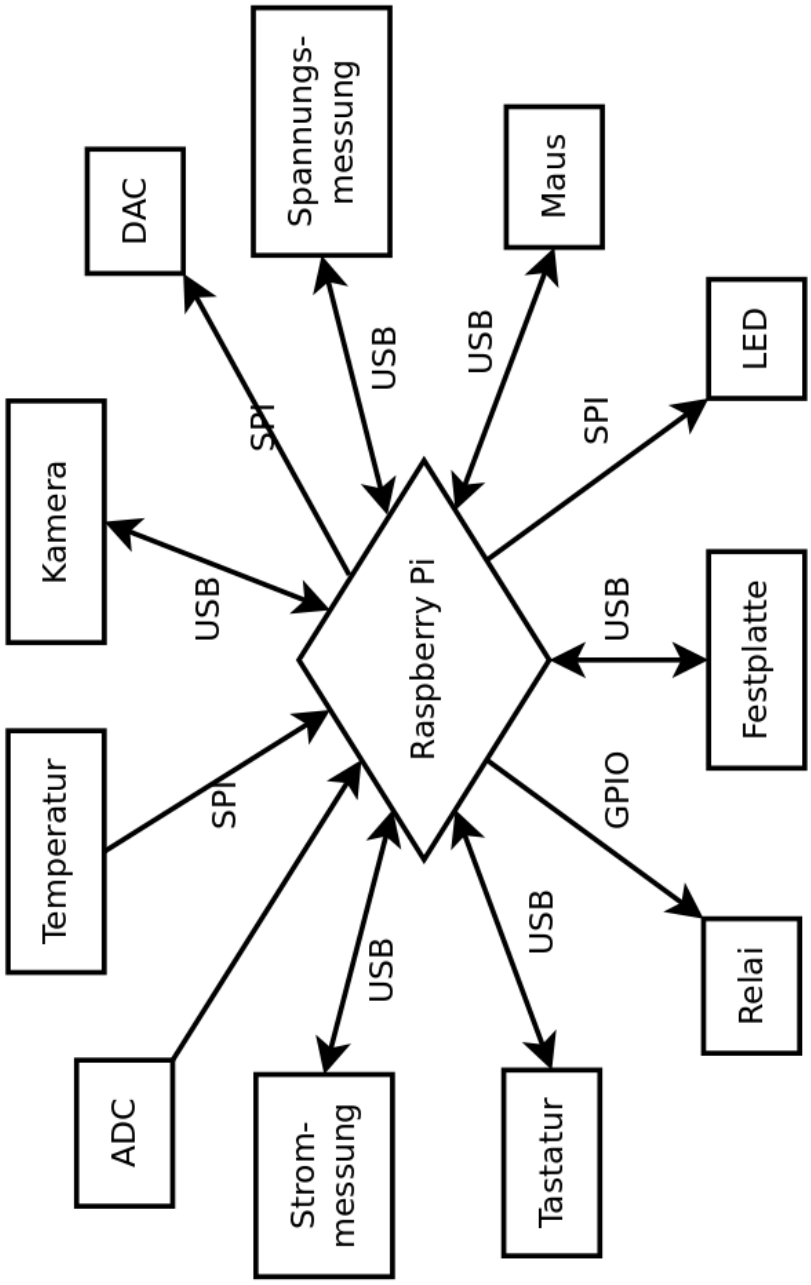

Abbildung 2.3.: Die Graphik zeigt eine Schnittstellenübersicht des Messsystems. Es kommen die USB-Ports des Raspberry, erweitert durch einen aktiven HUB und seine GPIO-Schnittstelle als binärer Ausgang und in der Sonderfunktion SPI zum Einsatz.

3. Entwurf der Hardwareerweiterung

In diesem Kapitel geht es um die Entwicklung der Messplatine. Tabelle 3.1 zeigt die Anforderungen an die Lade-/Entladeregelung. Diese soll eine Lithium-Ionen-Zelle mit einem maximalen Strom von $I_{Z_{max}} = 1mA$ zwischen $U_Z \approx 0V$ und $U_Z = 4,5V$ laden. Der Strom soll in $\Delta I_{Z_{Step}} = 1\mu A$ Schritten einstellbar sein. Es soll eine Überwachung der Zellspannung und des Lade-/Entladestroms geben. Außerdem soll die Umgebungstemperatur der Zelle gemessen werden und eine Schnittstelle für eine LED[1]-Dimmung von RGB-LED, welche die Elektrode beleuchten, implementiert werden.

Maximaler Lade-/Entladestrom	$I_{Z_{max}}$	$= 1mA$
Kleinste Zellspannug	$U_{Z_{min}}$	$\approx 0V$
Höchste Zellspannung	$U_{Z_{max}}$	$= 4,5V$
Schrittweite Stromvorgabe	$\Delta I_{Z_{Step}}$	$= 1\mu A$

Tabelle 3.1.: Anforderungen an die Lade-/Entladeregelung

Die Betrachtungen in diesem Kapitel sind reine Berechnungen. Inwieweit sich diese in der Praxis bestätigen, wird im Kapitel 5 erläutert.

3.1. Erstes Konzept der Hardwareerweiterung

Es gab zwei Konzepte für die Erweiterungsplatine, von denen eines verworfen wurde. Es soll dennoch zumindest prinzipiell vorgestellt werden, bevor das zweite, letztendlich realisierte Konzept, erläutert wird. Ein Bild der kompletten Schaltung befindet sich auf Seite 29.

3.1.1. DAC-Vorstufe

Das erste Konzept sah einen Mikrocontroller (z.B. Xmega16a4u [1]) vor, welcher vom Raspberry-Pi über einen SPI-Bus mit Befehlen versorgt wird. Diesen wäre er nachgekommen, indem er mit seinen DAC eine Analogschaltung ansteuert. Die Spannungsdifferenz

[1] LED - Light-Emitting Diode

inklusive Vorzeichen der beiden DAC (Im Bild 3.1 als Spannungsquellen dargestellt) wäre im Knoten zwischen den Widerständen R_9 und R_{10}, im Bild 3.1 auf Seite 23 als „Input CC“ bezeichnet, abgegriffen und am Eingang der Konstantstromquelle (Bild 3.2 auf Seite 24 - „U_{in}“) angelegt worden. Bei einer vollen Aussteuerung von V_4 auf 3,3V würde der zum invertierenden Verstärker beschaltete Operationsverstärker U_5 auf -6,6V aussteuern. Der DAC V_3, der auf 0V bleibt, erzeugt an seinem Ausgang eine Art virtuelle Masse, so dass der Spannungsteiler aus R_9 und R_{10} die Ausgangsspannung von U_5 am Ausgangsknoten „Input CC“ auf -3,3V halbiert. Steuert man hingegen V_3 auf 3,3V und V_4 auf 0V, befindet sich die virtuelle Masse oberhalb des Spannungsteilers. Der Ausgang des Operationsverstärkers U_6, welcher zu einem nicht invertierenden Verstärker beschaltet ist, steigt auf eine Spannung von +6,6V an. Der Spannungsteiler macht am Ausgang „Input CC“ eine Spannung von 3,3V daraus.
Das bedeutet, die DAC-Vorstufe sorgt für eine Steuerspannung von $U_{in} = -3,3V$ bis $U_{in} = +3,3V$ am Eingang der Konstantstromquelle in Bild 3.2 auf Seite 24.

3.1.2. Bipolare FET-Stromquelle

Die Konstantstromquelle [32] (Bild 3.2 auf Seite 24) hat an ihrem Eingang einen Operationsverstärker der mit seinem Ausgang der Eingangsspannung folgt und dabei einen Strom über den Widerstand R_5 fließen lässt. Dieser Strom fließt je nach Aussteuerung auch über die Widerstände R_1 und R_2, wo er einen Spannungsabfall verursacht, welcher wiederum die Operationsverstärker U_2 und U_3 ansteuert. Bei einer positiven Eingangsspannung fließt der Lastrom von R_5 auch über R_1. Damit sinkt die Spannung am positiven Eingang vom Operationsverstärker U_2 und seine Ausgangsspannung folgt. Das bewirkt ein Öffnen des MOSFET M_2 und es fließt ein Strom über R_3 in die Zelle, denn M_1 sperrt vollständig. Auf diese Weise wird die Zelle geladen. Wird die Eingangsspannung jedoch negativ, wird M_1 öffnen und M_2 sperren. Dann wird der Strom die Zelle entladen. Der Betrag des Stromes hängt dabei von der Höhe der Eingangsspannung ab. Für die Berechnung des Lade-/Entladestroms gilt $R_1 = R_2$ und $R_3 = R_4$. Der Lade-/Entladestrom berechnet sich nach Formel 3.1 [32].

$$I_Z = \frac{R_1}{R_3 R_5} \cdot U_{in} \tag{3.1}$$

Mit einer zusätzlichen Bedingung $R_1 = R_2 = R_5$ vereinfacht sich die Rechnung zu Formel 3.2 [32].

$$I_Z = \frac{U_{in}}{R_3} \tag{3.2}$$

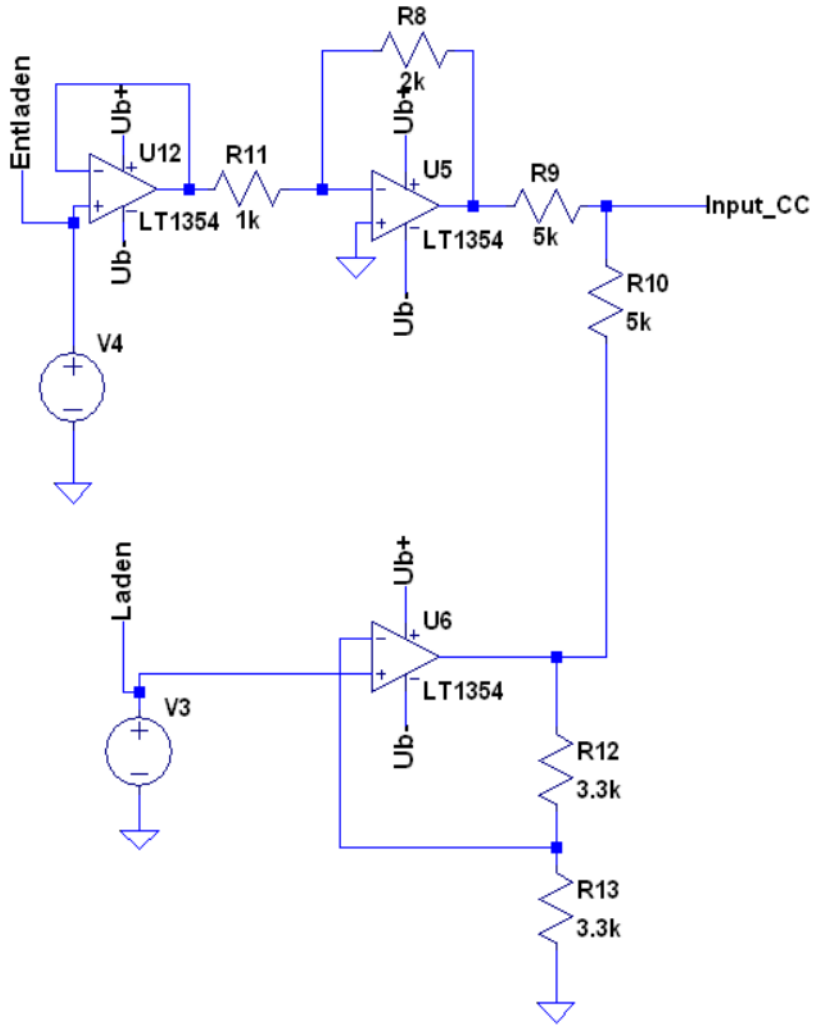

Abbildung 3.1.: DAC-Vorstufe - Erzeugt aus den Ausgangsspannungen zweier DAC V_3 und V_4 eine Steuerspannung „Input CC“ von -3,3V bis 3,3V, zur Ansteuerung einer Konstantstromquelle. Es sollten zwei DAC zum Einsatz kommen, um sowohl für Laden als auch Entladen die volle Auflösung nutzen zu können. Für mehr Details siehe Kapitel 3.1.1.

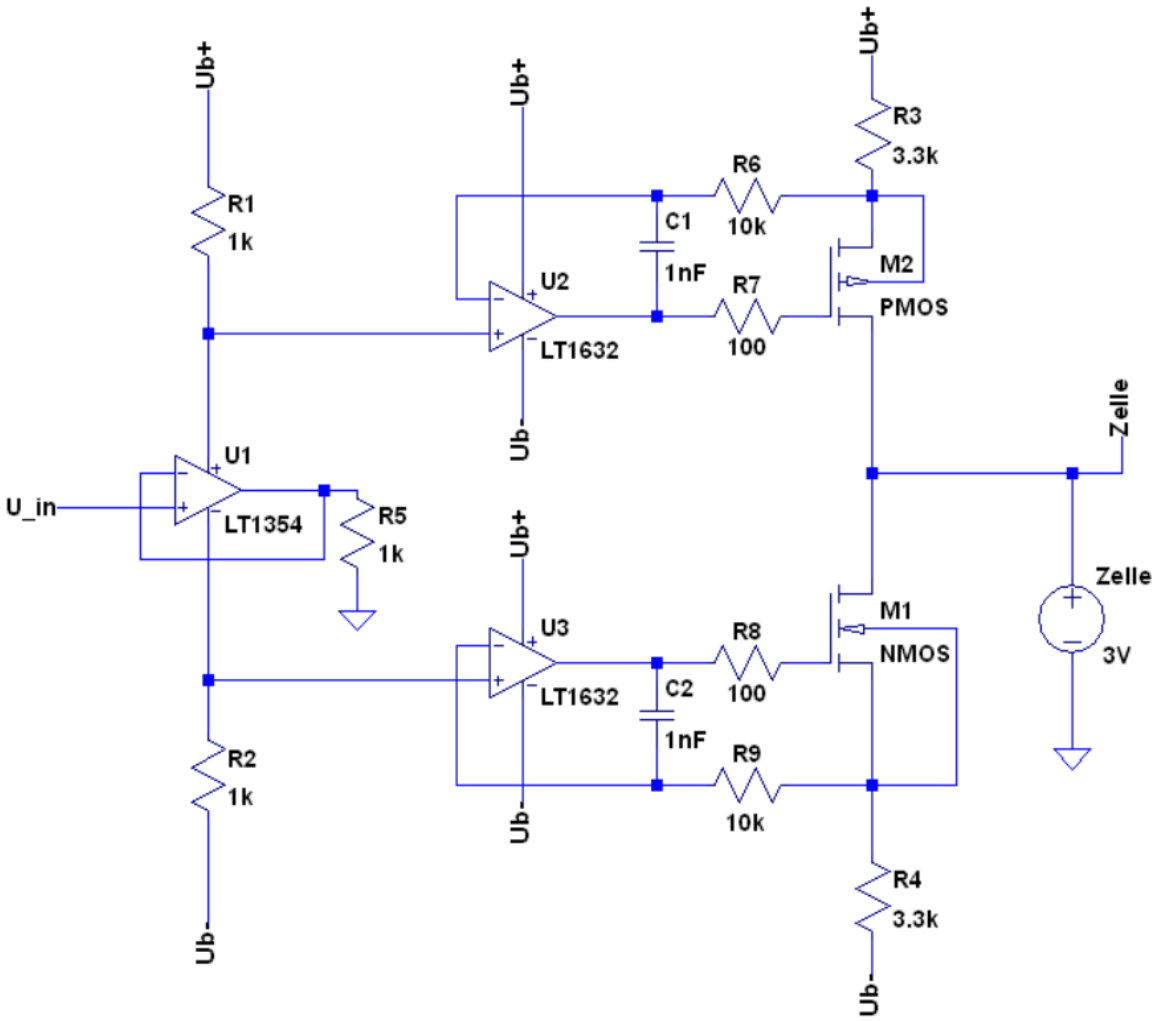

Abbildung 3.2.: Die Bipolare FET-Stromquelle hat zwei Stellglieder M_1 und M_2. M_1 steuert den Ladestrom und M_2 den Entladestrom. Für mehr Details siehe Kapitel 3.1.2 [32].

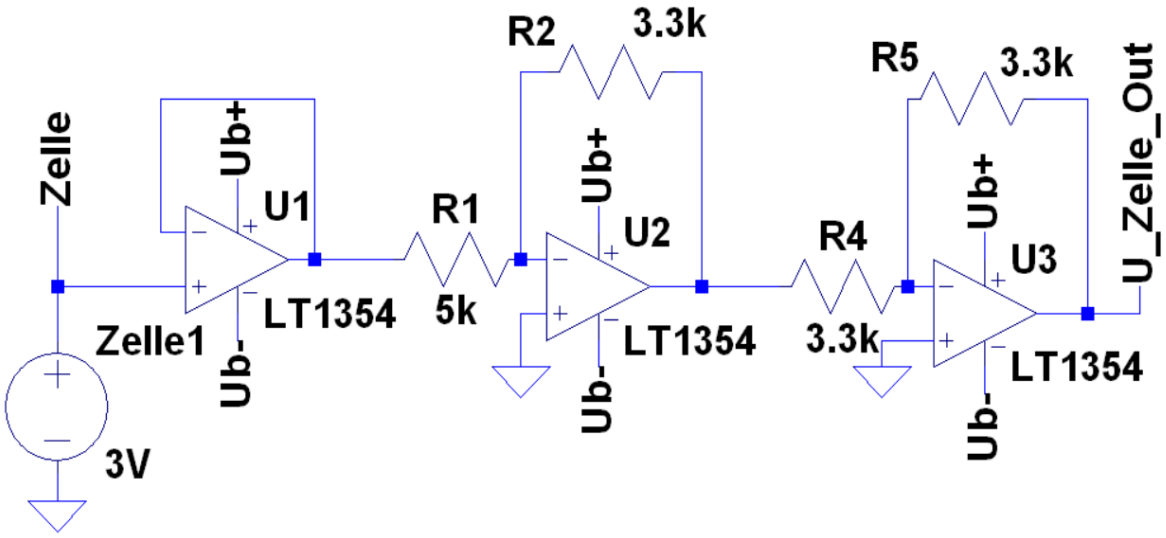

Abbildung 3.3.: Spannungsmessung mit einem Impedanzwandler, um den Ladestrom nicht zu belasten. Die Verstärker sorgen für eine Dämpfung der Spannung mit positivem Vorzeichen, um diese mit einem ADC messen zu können. Für mehr Details siehe Kapitel 3.1.3.

3.1.3. Spannungsmessung

Für die Spannungsmessung (Bild 3.3) braucht es zunächst einen Impedanzwandler U_1, denn der Ladestrom der Zelle soll möglichst nicht belastet werden. Da die Zelle auf bis zu 4,5V geladen werden soll, muss die Spannung für den ADC reduziert werden, denn dieser verträgt nur Spannungen bis 3,3V. Dafür wird ein Operationsverstärker U_2, beschaltet zu einem invertierenden Verstärker mit der Verstärkung $A_1 = -0,5$, verwendet. Ein nicht invertierender Verstärker kommt nicht in Frage, denn diese haben eine Verstärkung größer eins. Problem dabei ist das resultierende negative Vorzeichen. Der ADC möchte eine positive Spannung an seinem Eingang. Deshalb wird ein zweiter invertierender Verstärker U_3 benötigt, diesmal mit der Verstärkung $A_2 = -1$. Seine Ausgangsspannung bewegt sich jetzt zwischen 0V und 2,25V und kann von dem ADC gemessen werden [6].

3.1.4. Erdfreies Amperemeter

Für die Strommessung wurde eine Möglichkeit gesucht, die Messung so durchzuführen, dass diese die Spannungsmessung so wenig wie möglich beeinflusst. Dazu eignet sich das „erdfreie Amperemeter“ [32] (Bild 3.4 auf Seite 27). Es besteht aus zwei Eingangs-OP[2] U_4 und

[2]OP - Operationsverstärker

U_5, gefolgt von einem Subtrahierer, bestehend aus den Widerständen R_8 bis R_{11} und dem OP U_6.
Die Eingangs-OP werden versuchen, die Spannungsdifferenz zwischen ihren Eingängen zu Null zu regeln. Solange die Aussteuergrenzen nicht erreicht werden, wird die Masse am positiven Eingang des unter OP U_5 an den Verbindungsknoten der negativen Eingänge von OP U_4 und OP U_5 „weitergereicht“ und damit auch an den positiven Eingang des OP U_4. Das bedeutet, am positiven Eingang des OP U_4 entsteht eine virtuelle Masse, die als Masse für die Zelle dient. Der Subtrahierer ermittelt die Spannungsdifferenz der Austeuerungen der OP U_4 und U_5. Diese ist proportional zum Strom und bildet den Strom in einem Verhältnis von $\frac{\Delta U}{\Delta I} = \frac{2V}{1mA}$ auf die Spannung ab. Auch hier besteht das Problem, dass der ADC nur positive Spannungen bis 3,3V verträgt. Die Ausgangsspannung am Ausgang erreicht jedoch Werte von -2V bis 2V (Bild 3.5 auf Seite 28). Diese muss für den ADC angehoben und gedämpft werden. Das erledigen die OP U_7 und U_8. Der OP U_8 sorgt für einen negativen Offset, dieser wird vom OP U_7 mit der Ausgangsspannung von OP U_6 addiert und das Ergebnis gedämpft. So wird eine Spannung zwischen 0V und 2V erreicht, welche vom ADC gemessen werden kann. Bild 3.5 zeigt den Lade-/Entladestrom (grün), die Ausgangsspannung vom Subtrahierer (blau) und die für den ADC aufbereitete Spannung (rot) [32] [6].

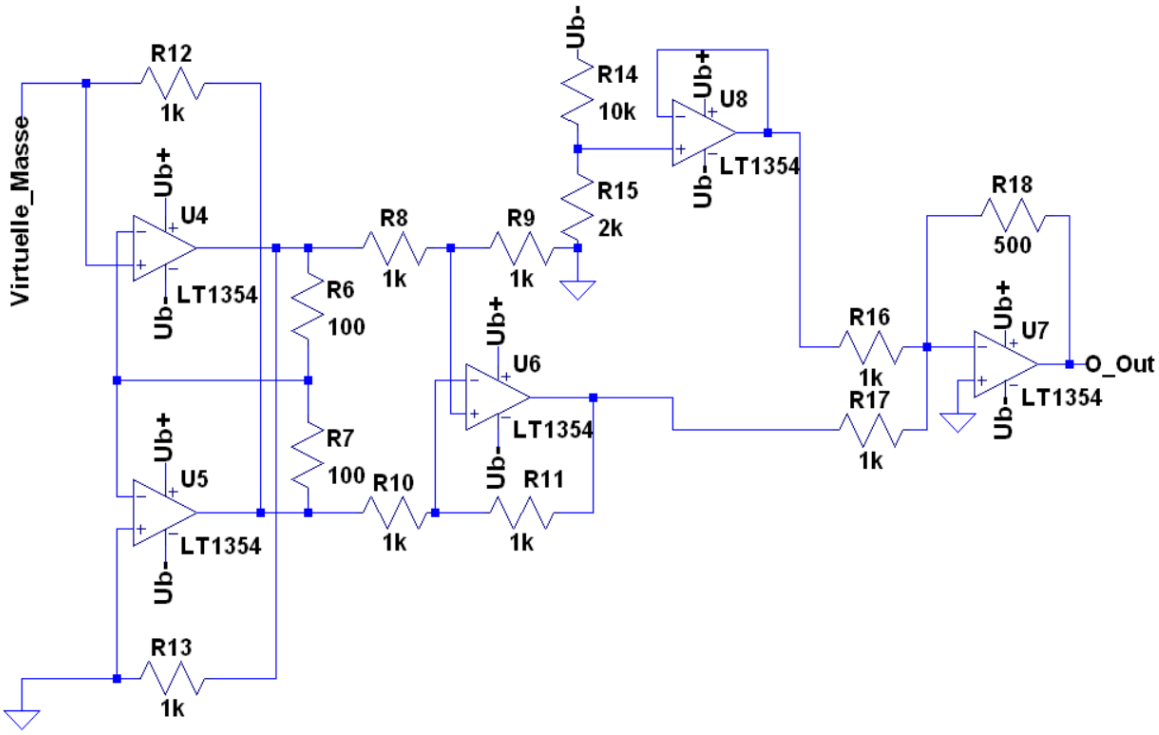

Abbildung 3.4.: Ein erdfreies Amperemeter [32], bestehend aus den OP U_4 bis U_6, mit Spannungsaufbereitung für einen ADC durch die OP-Schaltungen U_7 und U_8. Es eignet sich für Strommessungen ohne Shunt-Widerstand. Für mehr Details siehe Kapitel 3.1.4 und Bild 3.5.

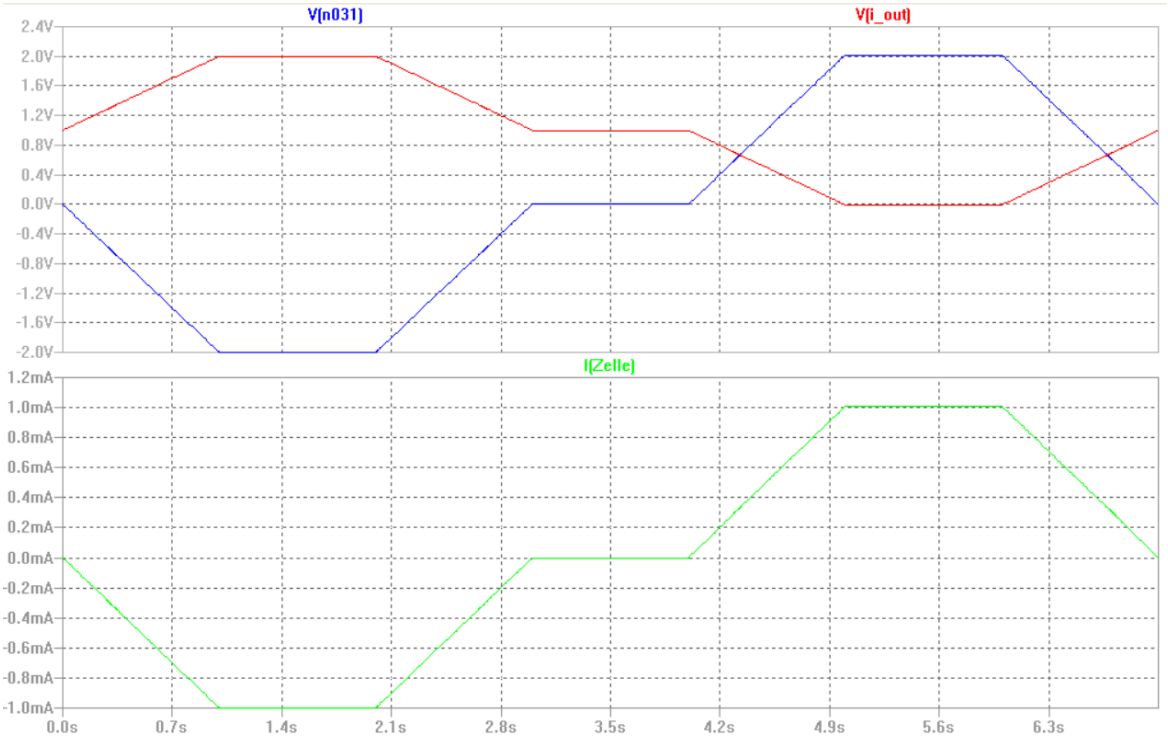

Abbildung 3.5.: Simulation der Messung des Lade-/Entladestrom (grün), und die Aufbereitung der Ausgangsspannung des erdfreien Amperemeters (blau) am Ausgang von U_6 in Bild 3.4, zu einer ADC-tauglichen Spannung (rot). Für mehr Details siehe Kapitel 3.1.4.

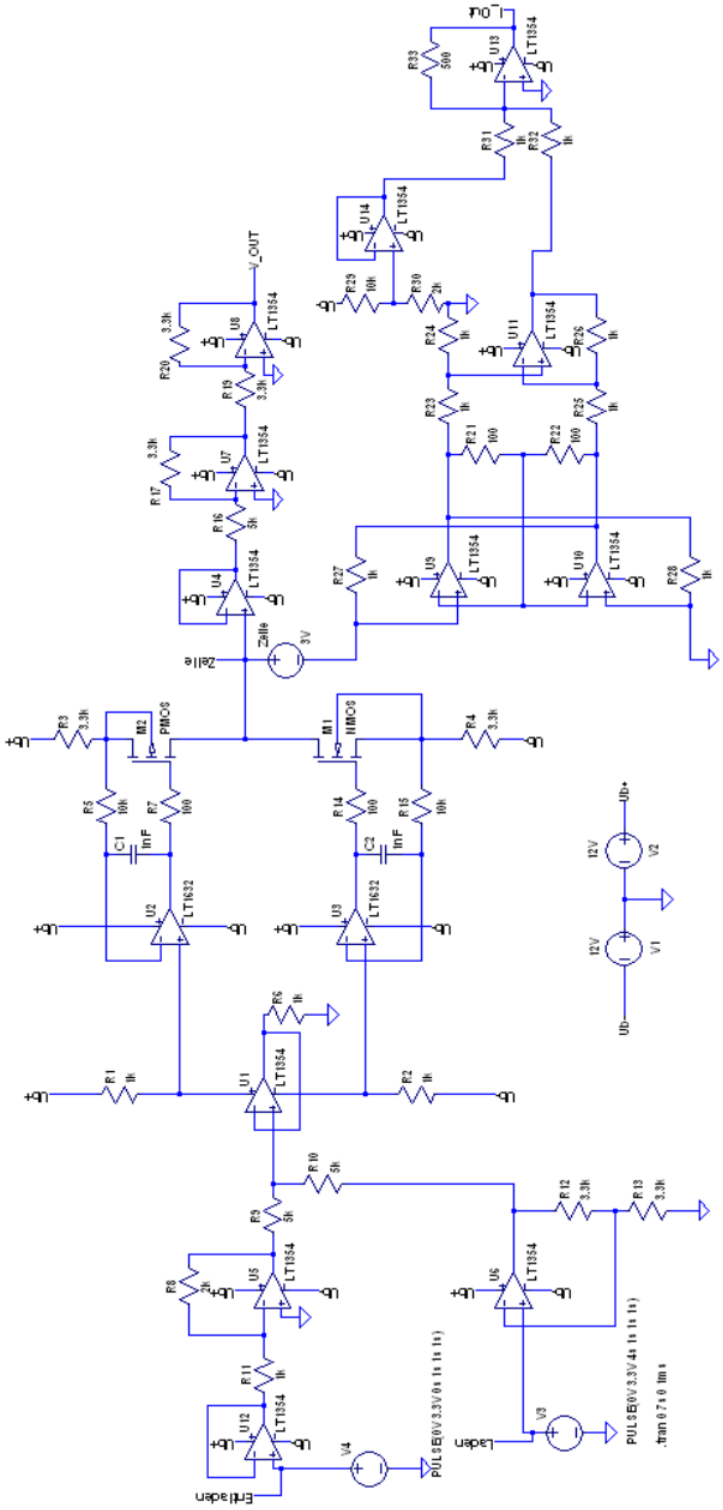

Abbildung 3.6.: Das Bild zeigt den analogen Teil des ersten Hardwarekonzeptes [32].

3.1.5. Beurteilung des Erstentwurfes

Das erste Konzept des Hardwareentwurfes konnte sich im Review nicht durchsetzen. Grund dafür war die Befürchtung, das System könne, wegen der sehr großen Anzahl von Operationsverstärkern, instabil werden. Darüber hinaus bringt jedes Bauteil eine Toleranz mit, wie z.B. Widerstände mit ihren typischen 1% Abweichung. Die hohe Anzahl der Operationsverstärker war auch deshalb ein Kritikpunkt, weil diese einen relevanten Temperaturgang haben könnten.

3.2. Zweites Konzept der Hardwareerweiterung (Release)

Das Bild 2.2 auf Seite 19 zeigt die Struktur der Hardware. Der Raspberry-Pi nimmt über ein Stellglied, welches im Wesentlichen aus einem DAC[3] und einem Treiber besteht, Einfluss auf den Lade-/Entladestrom. Das Stellglied wird der Zellspannung beim Lade-/Entladevorgang mit der Spannung vorauseilen, so dass diese kontinuierlich ihr Spannungsniveau anpasst. Der Lade-/Entladestrom wird durch die Spannungsdifferenz zwischen Stellglied und Zelle bestimmt, welcher über einem Messwiderstand abfällt. Dieser Messwiderstand ist Teil des Messwandlers für die Strommessung, der durch einen hochohmigen Instrumentenverstärker und einen ADC[4] vervollständigt wird. Der Strom-Messwandler wird vom Raspberry-Pi ausgelesen, so dass dieser entscheiden kann, ob und wie die Einstellung des Stellgliedes verändert werden muss. Der zweite Messwandler ist für die Messung der aktuellen Zellspannung zuständig und besteht aus einem Instrumentenverstärker, wie er bereits in der Strommessung zum Einsatz kommt, und einem weiteren ADC. Auch dieser ADC wird vom Raspberry-Pi ausgelesen. Diesmal um die Einhaltung der vorgegebenen Lade-/Entladeschwellen zu überwachen.
Neben der Zell-Zyklierung soll die Temperatur der Zell-Umgebung überwacht werden. Dazu wird auf der Messplatine lediglich eine Spannungsversorgung und eine SPI-Anbindung gestellt. Das Thermometer befindet sich auf einer separaten Platine, damit dieses mit Hilfe eines Kabels näher an der Zelle untergebracht werden kann. Für die LED, welche sich ebenfalls auf einer separaten Platine befinden, wird lediglich eine SPI-Anbindung gestellt. Das wurde so geplant, weil die Stromversorgung von der Mikroskop-Kamera, die mit der LED-Platine ausgestattet wird, übernommen werden soll.

[3] DAC - Digital-Analog-Converter
[4] ADC - Analog-Digital-Converter

3.3. Spannungsversorgung

Für die Versorgung der Messplatine werden mehrere Spannungsebenen benötigt. Die oberste Spannungsebene stellt das Netzteil dar, welches die Platine versorgt. Von dieser aus werden mehrere Spannungsregler kaskadiert, um die Versorgungsspannung auf die benötigten Ebenen zu reduzieren. Die 5V-Ebene wird dabei zweimal erzeugt. Einmal für die Versorgung des analogen Teils und nochmals für den digitalen Teil der Schaltung. Analog- und Digitalteil bekommen auch separate Masseflächen, damit der digitale Teil der Platine weniger Störungen in den Analogteil ausstrahlen kann. Diese werden lediglich an einer Stelle der Platine verbunden - dem Sternpunkt. Im Folgenden werden nun die Spannungsregler und ihre zugehörigen Versorgungsebenen beschrieben.

3.3.1. Die 10V-Spannungsebene

Die 10V-Spannungsebene wird mit dem LDO[5]-Spannungsregler „LM2940T-10.0“ [17] versorgt. Dieser hat ein TO-220-3-Gehäuse und kann einen Strom von 1A liefern. Seine Worst-Case Eigenstrom-Aufnahme beträgt 60mA, für diese Schaltung sind jedoch nur ca. 10mA zu erwarten. Folgende Bauteile werden vom Spannungsregler „LM2940T-10.0“ versorgt:

1. Der 5V-Analog Spannungsregler „MAX8881“
2. Der 5V-Digital Spannungsregler „MAX8881“
3. Die positive Versorgung der Instrumentenverstärker „AD8422“
4. Die positive Versorgung des Operationsverstärkers „OPA743“

Erwärmung des Spannungsreglers „LM2940T-10.0“

Der Spannungsregler „LM2940T-10.0“ muss, je nach Auslastung, einen Kühlkörper bekommen. An dieser Stelle soll berechnet werden, ob dies notwendig ist. Dazu wird zunächst der zulässige Temperaturanstieg berechnet. Für die Außentemperatur wird $\vartheta_A = 25°C$ angenommen und für die maximale zulässige Erwärmung eine Temperatur von $\vartheta_{max} = 60°C$ und die Differenz berechnet.

$$\Delta\vartheta = \vartheta_{max} - \vartheta_A = 60°C - 25°C = \underline{35°C} \tag{3.3}$$

[5] LDO - Low drop out

Die Versorgungsspannung beträgt $U_{IN} = 13V$ und es wird ein Strombedarf von $I = 85mA$ erwartet. Der Strom $I_{GND} = 30mA$ kann dem Datenblatt [17] entnommen werden. Dann wird die zu erwartende Verlustleistung berechnet.

$$P_D = (U_{IN} - U_{OUT}) \cdot I + I_{GND} \cdot U_{IN} = (13V - 10V) \cdot 85mA + 30mA \cdot 13V = \underline{\underline{0,645W}} \quad (3.4)$$

Die Wärmeabgabe des Bauteils berechnet sich als Quotient von Temperaturanstieg und Leistung [17].

$$R_{\Theta JA} = \frac{\Delta\vartheta}{P_D} = \frac{35°C}{0,645W} = \underline{\underline{54,26\frac{°C}{W}}} \quad (3.5)$$

Laut Datenblatt [17] wird ein Kühlkörper erst nötig, wenn der Wärmeableitungsquotient zwischen Bauteil und Umgebung einen Wert von $R_{\Theta JA_{min}} = 23.3\frac{°C}{W}$ unterschreitet. Sollte der Betrieb in einem Temperaturschrank erfolgen, so darf die Außentemperatur einen Wert von

$$\vartheta_{A_{max}} = (-1) \cdot (R_{\Theta JA_{min}} \cdot P_D - \vartheta_A) = (-1) \cdot (23.3\frac{°C}{W} \cdot 0,645W - 60°C) = \underline{\underline{44,97°C}} \quad (3.6)$$

nicht überschreiten.

Der hier beschriebene Rechenweg wurde im Datenblatt [17] empfohlen. Es ist auch möglich, den Wärmeabgabequotienten mit der ermittelten Leistung direkt zu verrechnen.

$$\Delta\vartheta' = \Delta R_{\Theta JA_{min}} \cdot P_D = 23.3\frac{°C}{W} \cdot 0,645W = \underline{\underline{15,03°C}} \quad (3.7)$$

Das deckt sich auch mit der ermittelten höchsten Außentemperatur (Gleichung 3.6) und der geforderten maximalen Erwärmung des Bauteils $\vartheta_{max} = 60°C$.

$$\Delta\vartheta' + \vartheta_{A_{max}} = 15,03°C + 44,97°C = \underline{60°C} \quad (3.8)$$

3.3.2. Die -10V-Spannungsebene

Die -10V-Spannungsebene wird mit einem Spannungsregler „LM7910CT“ [10] versorgt. Dieser hat ein TO-220-3-Gehäuse und kann, bei einer Eigenstromaufnahme von ca. 6mA, einen Strom von 1A liefern. Der Spannungsregler „LM7910CT“ versorgt die folgenden Bauteile:

1. Die negative Versorgung der Instrumentenverstärker „AD8422“

Erwärmung des Spannungsreglers „LM7910CT“

Die Wärmeabgabe des Spannungsreglers „LM7910CT“ wird im Datenblatt [10] mit $R_{\Theta JA} = 65\frac{^\circ C}{W}$ angegeben. Die Versorgungsspannung beträgt $U_{IN} = -13V$ und der erwartete Strom $I = 10mA$. Die Leistung beträgt

$$P_D = |(U_{IN} - U_{OUT})| \cdot I + I_{GND} \cdot |U_{IN}| = |-(13V - 10V)| \cdot 10mA + 13V \cdot 6mA = \underline{\underline{0,108W}} \tag{3.9}$$

und der zu erwartende Temperaturanstieg

$$\Delta\vartheta' = R_{\Theta JA} = 65\frac{^\circ C}{W} \cdot P_D = 65\frac{^\circ C}{W} \cdot 0,108W = 7,02^\circ C \tag{3.10}$$

Das bedeutet, es ist auch hier kein Kühlkörper nötig, um unter der geforderten maximalen Temperatur $\vartheta_{max} = 60^\circ C$ zu bleiben.

3.3.3. Die 5V-Spannungsebene

Die 5V-Spannungsebene teilt sich in zwei Bereiche, der analoge und der digitale Teil. Das dient dem Zweck, den analogen Teil nicht mit Störungen aus dem digitalen Teil zu belasten. Beide Spannungsebenen werden mit dem Spannungsregler „MAX8881“ [22] versorgt. Dieser hat ein SOT-23-6-Gehäuse, kann einen Strom von 200mA liefern und hat dabei einen Eigenstrombedarf von $10\mu A$. Der Spannungsregler für den digitalen Stromkreis versorgt:

1. Den 3,3V-Digital-Spannungsregler „TLV2217“
2. Die Relaisspulen der „HE3321A0400“-Relais
3. Den Lastkreis der Potentiometer „AD5270“

Der Spannungsregler für den analogen Stromkreis versorgt:

1. Die 2,5V-Referenzspannungsquelle „TPS79925“
2. Die 1,25V-Referenzspannungsquelle „LD39015“
3. Den Lastkreis der Analog-Digital-Wandler „AD7691“
4. Den Lastkreis des Digital-Analog-Wandlers „AD5680“

Erwärmung des Spannungsreglers „MAX8881“

Für die Erwärmung wird hier der Digital-Stromkreis exemplarisch für beide betrachtet, da dieser die größere Leistung aufbringen muss. Wenn er innerhalb der gesetzten Grenze von $\vartheta_{max} = 60°C$ bleibt, gilt das auch für den Analogteil. Die Versorgungsspannung beträgt $U_{IN} = 10V$ die Ausgangsspannung $U_{OUT} = 5V$ und es wird ein Laststrom von $I = 67mA$ bei einem Eigenbedarf von $I_{GND} = 10\mu A$ erwartet. Das Datenblatt [22] liefert einen Wärmeableitungsquotienten von $R_{\Theta JA_{min}} = 114,94\frac{°C}{W}$. Es gelten die Formeln 3.9 und 3.10.

$$P_D = (10V - 5V) \cdot 67mA + 10V \cdot 10\mu A = \underline{\underline{0,3351W}} \tag{3.11}$$

$$\Delta\vartheta = 114,94\frac{°C}{W} \cdot 0,3351W = \underline{\underline{38,51°C}} \tag{3.12}$$

Das bedeutet, dass die selbst gesetzte Grenze einer Bauteilerwärmung auf 60°C um 3,5°C überschritten wird. Das ist jedoch noch nicht wirklich problematisch, da der Spannungsregler „MAX8881“ [22] auch bei Temperaturen über 100°C keinen Schaden nimmt.

3.3.4. Die 3,3V-Spannungsebene

Die 3,3V-Ebene wird mit dem Spannungsregler „TLV2217“ [20] versorgt. Er hat ein TO-220-3-Gehäuse, kann einen Strom von 500mA liefern und hat einen Eigenbedarf von ca. 19mA. Der Spannungsregler „TLV2217“ versorgt die folgenden Bauteile:

1. Das digitale Interface des Potentiometers „AD5270“
2. Das digitale Interface des Analog-Digital-Wandlers „AD7691“
3. Das digitale Interface des Temperatursensors „LM74“

Erwärmung des Spannungsreglers „TLV2217“

Für die Berechnung der Erwärmung gilt $U_{IN} = 5V$, $U_{OUT} = 3,3V$, $I = 15mA$, $I_{GND} = 19mA$ sowie die Formeln 3.9 und 3.10. Das Datenblatt [20] liefert einen Wärmeabgabekoeffizienten von $R_{\Theta JA_{min}} = 19\frac{°C}{W}$.

$$P_D = (5V - 3,3V) \cdot 15mA + 5V \cdot 19mA = \underline{\underline{0,1205W}} \tag{3.13}$$

$$\Delta\vartheta = 19\frac{°C}{W} \cdot 0,1205W = \underline{\underline{2,2895°C}} \tag{3.14}$$

Die zu erwartende Erwärmung ist demnach unproblematisch.

3.3.5. Die 2,5V-Spannungsebene

Die 2,5V-Spannungsebene wird mit dem Spannungsregler „TPS79925“ [21] versorgt. Dieser hat ein SOT-23-5-Gehäuse, kann einen Strom von 200mA liefern und hat einen Eigenbedarf von $60\mu A$. Der Spannungsregler „TPS79925“ versorgt die folgenden Bauteile:

1. Die Referenzspannung der Analog-Digital-Wandler „AD7691“
2. Die Referenzspannung des Digital-Analog-Wandlers „AD5680“

Erwärmung des Spannungsreglers „TPS79925“

Für die Berechnung der Erwärmung gilt $U_{IN} = 5V$, $U_{OUT} = 2,5V$, $I = 200\mu A$, $I_{GND} = 60\mu A$ sowie die Formeln 3.9 und 3.10. Das Datenblatt [21] liefert einen Wärmeabgabekoeffizienten von $R_{\Theta JA_{min}} = 225,3\frac{^\circ C}{W}$.

$$P_D = (5V - 2,5V) \cdot 200\mu A + 5V \cdot 60\mu A = \underline{\underline{800\mu W}} \tag{3.15}$$

$$\Delta\vartheta = 225,3\frac{^\circ C}{W} \cdot 800\mu W = \underline{\underline{0,18024^\circ C}} \tag{3.16}$$

Die zu erwartende Erwärmung ist ebenfalls unproblematisch.

3.3.6. Die 1,25V-Spannungsebene

Die 1,25V-Spannungsebene wird mit dem Spannungsregler „LD39015“ [27] versorgt. Dieser hat ein SOT-23-5-Gehäuse, kann einen Strom von 100mA liefern und hat einen Eigenbedarf von $70\mu A$. Der Spannungsregler „LD39015“ versorgt das folgende Bauteil:

1. Die Referenzspannung des Instrumentenverstärkers „AD8422“

Erwärmung des Spannungsreglers „LD39015“

Für die Berechnung der Erwärmung gilt $U_{IN} = 5V$, $U_{OUT} = 1,25V$, $I = 50\mu A$, $I_{GND} = 70\mu A$ sowie die Formeln 3.9 und 3.10. Das Datenblatt [27] liefert einen Wärmeabgabekoeffizienten von $R_{\ominus JA_{min}} = 255\frac{^\circ C}{W}$.

$$P_D = (5V - 1,25V) \cdot 50\mu A + 5V \cdot 70\mu A = \underline{\underline{537,5\mu W}} \tag{3.17}$$

$$\Delta\vartheta = 255\frac{^\circ C}{W} \cdot 537,5\mu W = \underline{\underline{0,13706^\circ C}} \tag{3.18}$$

Auch hier ist die zu erwartende Erwärmung unproblematisch.

3.3.7. Zusammenfassung zur Spannungsversorgung

Das Bild 3.7 auf Seite 37 zeigt einen Überblick über die Spannungsversorgung. Die grünen Angaben zeigen, um welche Spannungsebene es sich handelt und die Ströme, die hier höchstens geliefert werden können. Die roten Angaben in den Kästen beziehen sich auf die zu erwartenden Ströme der jeweiligen Stromsenke. Die roten Angaben über den Knoten repräsentieren die maximal zu erwartenden Ströme, die in diesen hineinfließen könnten. Die roten Angaben über den Kästen geben die Worst-Case-Eigenstrom-Aufnahme der Spannungsregler an. Das bedeutet bei maximal zulässigem Spannungsabfall zwischen Ein- und Ausgang und dem größten zulässigen Laststrom. Dieses Szenario wird es jedoch nicht geben. Auch alle Angaben zu den Stromsenken wurden eher etwas größer angenommen, als es in der Praxis der Fall sein wird. Dennoch bleiben alle Knotenströme unterhalb ihrer maximal zulässigen Werte [10] [17] [7] [22] [19] [20] [4] [6] [18] [15] [21] [5] [27].

Die größte Erwärmung ist beim Spannungsregler für die 5V-Digitalspannung zu erwarten. Diese wird etwa 38°C betragen. Außentemperaturen von bis zu 40°C sind jedoch nicht problematisch. Die Netzspannung des externen Netzteils sollte zwischen $U_{Netz} = \pm 12V$ und $U_{Netz} = \pm 13V$ liegen. Ein Verpolungsschutz wurde vorgesehen.

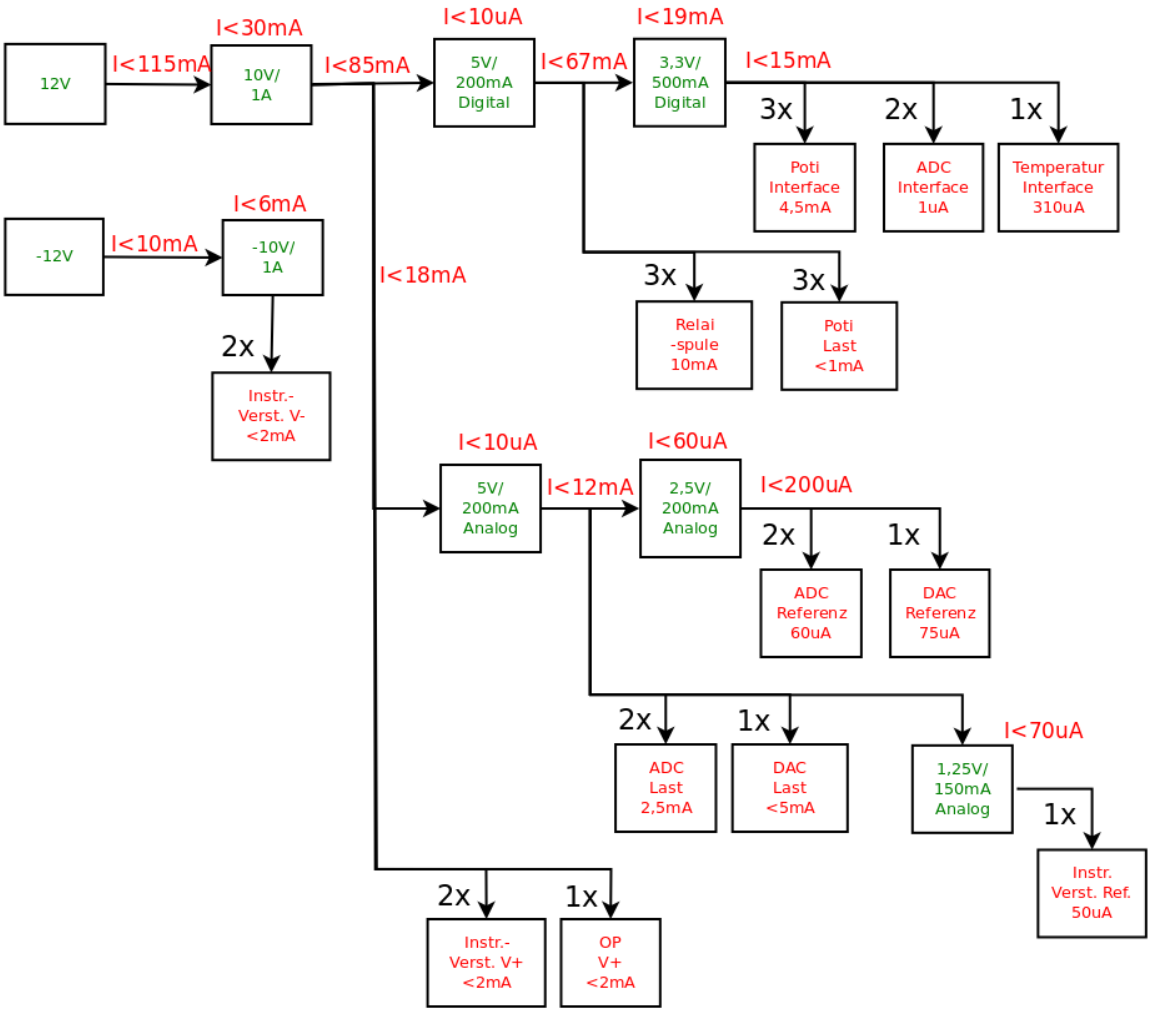

Abbildung 3.7.: Die roten Angaben zeigen die Ströme, die in die Senken und Knoten hineinfließen. Die Angaben über den Spannungsreglern, ebenfalls rot, stellen den Eigenbedarf der Regler dar. Die grünen Angaben sagen aus, welche Ströme und Spannungen von den Spannungsregler gespeist werden können. Die Werte wurden den Datenblättern der Bauteile entnommen, welche sich auf der beigefügten DVD befinden [10] [17] [7] [22] [19] [20] [4] [6] [18] [15] [21] [5] [27]. Für die Stromangaben wurden sicherheitshalber erhöhte Werte angenommen.

3.4. Entwurf einer LED-Steuerung (RGB)

Um das optische Spektrum der Elektroden untersuchen zu können, sollen die Rot-, Grün- und Blauanteile der Kamerabeleuchtung einstellbar sein. Das soll mit entsprechenden RGB-LED erreicht werden. Diese werden von einem IC gedimmt, welches aus einer variablen Gleichspannung eine PWM[6] generiert. Da die Platine mit dem PWM-IC und den LED gegeben ist [29], soll diese hier nicht weiter behandelt werden. Die Schnittstelle zwischen Beleuchtung und Messplatine ist eine variable Gleichspannung von 0V bis 5V. Diese wird über ein digitales Potentiometer realisiert.

3.4.1. Das digitale Potentiometer

Das digitale Potentiometer „AD5271“ [4] mit der Gehäuseform MSOP-10 hat eine Auflösung von 8 Bit. Das digitale Interface wird mit einer Spannung $V_{DD} = 3,3V$ versorgt. Die Spannung $V_A = 5V$ liegt am positiven Pol des $20k\Omega$ Widerstandes. Der Schleifer greift in 2^8 Schritten die Zwischenspannungen ab. Der negative Pol wird an Masse angeschlossen. Über einen SPI-Bus kann die gewünschte Schleiferstellung eingestellt werden [4].

Bemerkung: Die ersten beiden Aufbauten der Messplatine sind mit dem Potentiometer „AD5271“ ausgestattet. Alle weiteren werden die 10-Bit-Version „AD5270“ [4] erhalten.

3.5. Entwurf einer Lade-/Entladesteuerung

Die Zelle soll zwischen $U_Z \approx 0V$ und $U_Z = 4,5V$ geladen werden. Dazu wird ein DAC benötigt, welcher über einen Treiber der Zelle bei ihrem Lade-/Entladevorgang mit der Spannung vorauseilt und so über einem Messwiderstand einen Spannungsabfall zwischen Treiberausgang und Zelle erzeugt, aus dem ein entsprechender Lade-/Entladestrom resultiert.

3.5.1. Der DAC als Stellglied

Als Stellglied für die Spannung wird der DAC „AD5680“ [5] von der Firma „Analog Devices“ mit der Gehäuseform SOT-23-8 verwendet. Dieser wird mit einer Spannung von 4,5V bis 5,5V versorgt und hat dabei einen Energiebedarf von ca. 1,6mW. Da es sich um einen „Rail-to-Rail-DAC“ handelt, ist er in der Lage, bis auf wenige Millivolt an seine Versorgungsspannung heran auszusteuern.

[6] PWM - Puls-Weiten-Modulation

Der DAC „AD5680“ hat eine 18-Bit-Auflösung von denen jedoch zwei Bit eine PWM mit einem Tastgrad von 0%, 25%, 50% und 75% steuern. Da die Messung nicht mit Oberwellen belastet werden soll, kommt die Verwendung der PWM nicht in Frage, so dass eine Auflösung von 16 Bit bleibt. Diese beziehen sich jedoch nicht auf die Versorgungsspannung, sondern auf die Referenzspannung U_{Ref}, welche separat an den DAC angeschossen wird. Das resultierende LSB[7] kann mit der Formel 3.19 berechnet werden. In diesem Anwendungsfall ist $n = 16Bit$ und für die Referenzspannung wurde $U_{Ref} = 2,5V$ gewählt. Damit ergibt sich ein LSB von

$$U_{LSB} = \frac{U_{Ref}}{2^n} = \frac{2,5V}{2^{16}} = \underline{\underline{38,15\mu V}} \tag{3.19}$$

Zusammen mit dem Messwiderstand von $R_M = 100\Omega$ ergibt sich damit eine Stromauflösung von

$$I_{LSB} = \frac{U_{LSB}}{R_M} = \frac{38,14697226\mu V}{100\Omega} = \underline{\underline{381,47nA}} \tag{3.20}$$

Wenn die Referenzspannung das LSB bestimmt, resultiert aus dieser auch die maximale Ausgangsspannung $U_{DAC_{max}} = \underline{\underline{2,5V}}$. Diese muss durch einen Treiber verstärkt werden, um die Zelle bis 4,5V laden zu können (siehe hierzu Kapitel 3.5.2). Die Ausgangsspannung selbst wird während des Betriebes über einen Feedback-Pin kontrolliert. Es empfiehlt sich, diesen über eine separate Leiterbahn genau an dem Punkt anzuschließen, an dem die gewünschte Spannung anliegen soll. Denn über die Leiterbahn des Ausgangs fließt, im Gegensatz zum „Feedbackleiter“, ein Strom, welcher zu einem Spannungsabfall führt, der das Feedback verfälschen würde. Die Ausgangsspannung kann mit der Formel 3.21 berechnet werden. D ist der aktuelle Wert des Ausgangsregisters.

$$U_{DAC} = U_{Ref} \cdot \frac{D}{2^{18}} \tag{3.21}$$

Um das Ausgangsregister ändern zu können, bietet der DAC „AD5680“ eine serielle Schnittstelle. Diese unterstützt SPI, QSPI und Microwire mit einer Taktfrequenz von bis zu $f_{SCLK_{max}} = 30MHz$ [5].

[7] LSB - Least Significant Bit

3.5.2. Die Treiberstufe des Stellgliedes

Der DAC hat eine maximale Ausgangsspannung von $U_{DAC_{max}} = 2,5V$. Das reicht nicht aus, um die Zelle auf bis zu 4,5V zu laden. Deshalb wird die Ausgangsspannung des DAC über einen RC-Tiefpass mit dem Operationsverstärker „OPA743“ [19] verstärkt. Der Operationsverstärker „OPA743“ ist ein Rail-to-Rail-OP und kommt bei kleinen Lastströmen, wie den benötigten $I_{Z_{max}} = 1mA$, auf bis zu 100mV an seine Versorgungsspannung heran. Der OP kann mit einer Spannungsdifferenz von bis zu 12V versorgt werden. Da keine negativen Spannungen benötigt werden, erfolgt die Versorgung mit $V_- = 0V$ und $V_+ = 10V$. Er wird mit zwei Widerständen zu einem nicht invertierendem Verstärker mit der Verstärkung $A = 2$ beschaltet und die Ausgangsspannung des DAC somit auf $U_{OP_{max}} = 5V$ erhöht. Somit ist es möglich, die Zelle mit den gewünschten 1mA auf bis zu 4,5V zu laden. Da der OP nur bis $U_{OP_{min}} = 100mV$ aussteuern kann und am Messwiderstand bei $I_{Z_{max}} = 1mA$ eine Spannung von 100mV abfällt, kann die Zelle nur bis 200mV mit $I_{Z_{max}} = 1mA$ entladen werden [19].

3.6. Spannungs- und Strommessung im Ladekreis

Die Spannung der Zelle und der Lade-/Entladestrom sollen laufend überwacht werden. Das größte Problem stellt dabei der geringe Strom von höchstens $I_{Z_{max}} = 1mA$ im Lade-/Entladekreis, welcher nicht durch reduzierende Knotenströme verfälscht werden soll. Außerdem müssen die Zellspannung und der Strom in einen ADC-gerechten Eingangsspannungsbereich konvertiert werden.

3.6.1. Analogteil des Messwandlers

Für die Messung des Stroms und der Spannung musste ein Messaufnehmer gefunden werden, der besonders hochohmige Eingänge hat. Denn der kleine Strom von $I_{Z_{min}} = 1\mu A$, den es zu messen gilt, darf nicht belastet werden. Der Instrumentenverstärker „AD8422“ [7] von der Firma „Analog Devices“ kann mit seinem Eingangswiderstand von ca. $R_{IN} = 200G\Omega$ dieser Anforderung gerecht werden. Er hat ein SOIC-8-Gehäuse und wird mit einer bipolaren Spannung von $\pm 2,3V$ bis $\pm 18V$ versorgt. Seine Aussteuergrenzen kommen auf bis zu 1,2V an die Versorgungsspannungen heran [7].

Strommessung

Ein Vorteil des Instrumentenverstärkers „AD8422“ ist seine Gleichtaktunterdrückung von 94dB, da er für die Messung des Stromes zwischen Treiber-OP und Zelle angeschlossen wird und sein Spannungsniveau um ca. 4,5V schwanken kann. Der Instrumentenverstärker wird mit einem Widerstand R_G beschaltet, welcher die Verstärkung beeinflusst. Da für den Strom eine Spannung von $U_{R_M}(I_{max} = 1mA) = 100mV$ erwartet wird und diese für den ADC auf $U_{ADC}(I_{max} = 1mA) = 1V$ verstärkt werden soll, braucht es eine Verstärkung von $G = 10$. Der dazu passende Widerstand R_G lässt sich mit der Formel 3.22 berechnen [7].

$$R_G = \frac{19,8k\Omega}{G-1} = \frac{19,8k\Omega}{10-1} = 2200\Omega \tag{3.22}$$

Mit dieser Beschaltung bildet der Instrumentenverstärker den Lade- und Entladestrom nun auf eine Spannung von

$$U_{ADC}(-I_{max}) = -1V \tag{3.23}$$

bis

$$U_{ADC}(I_{max}) = 1V \tag{3.24}$$

ab. Der ADC würde eine negative Eingangsspannung jedoch nicht unbeschadet überstehen. Das bedeutet, das Spannungsniveau muss um mindestens 1V angehoben werden. Da zum Schutze der ADC die untere Aussteuergrenze nicht erreicht werden soll (siehe Kapitel 3.6.2), wurde das Spannungsniveau um $U_{Ref} = 1,25V$ angehoben. Der Instrumentenverstärker „AD8422“ bietet für diesen Zweck einen Referenzeingang, welcher mit den 1,25V versorgt wird. Damit wird der Strom auf ADC-passende 0,25V bis 2,25V abgebildet [7] [6].

Spannungsmessung

Der Instrumentenverstärker für die Spannungsmessung wird mit keinem Widerstand R_G beschaltet, was einer Verstärkung von $G = 1$ entspricht. Geringere Verstärkungen sind mit dem Instrumentenverstärker „AD8422“ nicht zu erreichen. Da jedoch die Spannung der Zelle auf bis zu $U_{Z_{max}} = 4,5V$ ansteigen kann und der ADC eine Spannung von höchstens $U_{ADC_{max}} = 2,5V$ erwartet, muss die Ausgangsspannung des Instrumentenverstärkers mit einem Spannungsteiler halbiert werden. Die obere Aussteuergrenze liegt damit bei $U'_{Z_{max}} = 2,25V$. Eine Anhebung des Spannungsniveaus wie bei der Strommessung ist nicht nötig, da die Zellspannung nicht negativ werden kann [7] [6].

3.6.2. Digitalteil des Messwandlers

Die Erfassung der Analogwerte übernimmt der ADC „AD7691“ [6] von der Firma „Analog Devices“ mit der Gehäuseform MSOP-10. Dieser wird mit einer einpoligen Spannung von 2.3V bis 5V versorgt. Sein Energiebedarf ist proportional zur Abtastrate, welche bei unserer Messung sehr gering ausfällt, übersteigt 4mW jedoch nicht. Das Auslesen der Daten geschieht über eine serielle Schnittstelle. Diese unterstützt SPI, QSPI und Microwire. Verwendet wird die Übertragung per SPI. Dazu muss der ADC „AD7691“ noch an seinem digitalen Versorgungspin mit einer Spannung von 3,3V versorgt werden. Diese stellt den High-Pegel für das Übertragungssignal [6].

Strommessung

Die Auflösung des ADC „AD7691“ [6] beträgt 18 Bit. Dabei ist jedoch zu berücksichtigen, dass es sich um differenzielle Eingänge handelt. Das bedeutet, dass die Eingangsspannung an den Eingängen IN+ und IN- in positiver und negativer Polung interpretiert wird. Da nur positive Spannungen gemessen werden, geht ein Bit verloren. Darüber hinaus wird ein Bit geopfert, um die Eingänge zu schützen. In diesen befinden sich ESD-Schutzdioden, die bei einer Über- bzw. Unterschreitung der Versorgungsspannung an den Eingängen sofort leitend werden. Die in diesem Fall auftretenden Ströme würden den ADC zerstören. In Zahlen sieht das folgendermaßen aus:
Die Referenzspannung beträgt $U_{Ref} = 2,5V$ mit einer Auflösung von 17 Bit berechnet sich das LSB mit

$$U_{LSB} = \frac{U_{Ref}}{2^n} = \frac{2,5V}{2^{17}} = \underline{\underline{19,073\mu V}} \tag{3.25}$$

Der maximale Lade-/Entladestrom von $I_Z = 1mA$ wird durch den Messwiderstand und einen Instrumentenverstärker mit ausgangsseitigem Offset (siehe Kapitel 3.6.1) auf

$$U(I = -1mA) = 0,25V \tag{3.26}$$

und

$$U(I = 1mA) = 2,25V \tag{3.27}$$

übersetzt. Bei einem $U_{LSB} = 19,037\mu V$ ergeben sich für unser Messung

$$\frac{U(|I_{max}|)}{U_{LSB}} = \frac{1V}{19,037\mu V} = \underline{\underline{52428,8}} \tag{3.28}$$

Zustände und

$$n'_I = \log_2(52428,8) = \underline{\underline{15,67Bit}} \tag{3.29}$$

für die Messung zur Verfügung stehende Bit. Eines der fehlenden Bit hat sich bereits durch die unipolare Messweise erklärt. Die weiteren 1,33 Bit sind eine Folge des bereits erwähnten Schutzes der Eingänge und des Umstandes, dass sowohl das Laden als auch das Entladen in dem Referenzspannungsbereich abgebildet wird. Da es nur ganze Bit geben kann muss auf 15 Bit abgerundet werden. Für die Messgenauigkeit ergibt sich damit eine Auflösung von

$$\frac{|I_{max}|}{2^{n'_I}} = \frac{1mA}{2^{15}} = \underline{\underline{30,51nA}} \tag{3.30}$$

Spannungsmessung

Für die Spannungsmessung wird ebenfalls der ADC „AD7691“ [6] verwendet. Da auch dieser mit einer Referenzspannung von $U_{Ref} = 2,5V$ versorgt wird, ergibt sich auch für die Spannungsmessung ein $U_{LSB} = 19,073\mu V$ (siehe Formel 3.25 auf Seite 42). Die Zellspannung, welche auf bis zu 4,5V ansteigen kann, wird durch einen Spannungsteiler am Ausgang des Instrumentenverstärkers halbiert (siehe Kapitel 3.6.1), so dass die zu erwartende maximale Spannung $U'_{Z_{max}} = 2,25V$ beträgt. Die minimale zu erwartende Spannung wird durch die untere Aussteuergrenze der Lade-/Entladesteuerung bestimmt. Diese kommt auf ca. $U_{Z_{min}} = 200mV$ (siehe Kapietel 3.5.2), so dass nach dem Spannungsteiler noch ca. $U'_{Z_{min}} = 100mV$ zu erwarten sind. Damit sind auch diese ADC-Eingänge vor Über- bzw Unterspannung durch Störungen geschützt. Die Berechnung der Auflösung für die Spannungsmessung erfolgt analog zur Strommessung. Der maximale Spannungshub von 4,5V wird auf eine ADC-Eingangsspannung 2,25V abgebildet.

$$\frac{\Delta U'}{U_{LSB}} = \frac{2,25V}{19,073\mu V} = \underline{\underline{117967,8}} \tag{3.31}$$

$$n'_U = \log_2(117967,8) = \underline{\underline{16,84Bit}} \tag{3.32}$$

Abgerundet ergibt sich eine Auflösung von 16 Bit und damit eine Messgenauigkeit von

$$\frac{\Delta U_Z}{2^{n'_U}} = \frac{4,5V}{2^{16}} = \underline{\underline{68,66\mu V}} \tag{3.33}$$

3.7. Temperaturmessung

Für die Temperaturmessung der Zellumgebung wird der SPI-Bus und eine 3V-Versorgungsspannung über Wago-Schraubklemmen von der Platine nach außen geführt. Hier kann ein Temperatursensor angeschlossen werden, der mit Hilfe eines Kabels in die unmittelbare Nähe der Zelle gebracht wird. Es kann im Prinzip ein beliebiger Sensor verwendet werden, für dieses Projekt wurde jedoch der LM74 von Texas Instruments ausgewählt. Dieser hat ein SOIC-8-Gehäuse, benötigt eine Versorgungsspannung von drei bis fünf Volt und nimmt weniger als $10\mu A$ Strom auf. Er hat eine Auflösung von 16 Bit, was ein LSB von $0,0625°C$ ergibt. Abgesehen von dem direkten Anschluss von Spannungsversorgung und Busanbindung, benötigt der LM74 keine weitere Beschaltung oder Kalibrierung [18].

3.8. Schutzmaßnahmen

Da es sich in diesem Projekt nur um sehr kleine Spannungen und Ströme handelt, die nicht gefährlich werden können, und auch keine Aktoren mit Gefahrenpotential angeschlossen werden, ist die Sicherheitsfrage in diesem Projekt nicht besonders relevant. Dennoch sind ein paar Maßnahmen zum Schutz der Lithium-Ionen-Zelle und der Bauteile auf der Platine nötig gewesen.

3.8.1. Schutz der Lithium-Ionen-Zelle

Die Zelle muss, wenn die Messvorrichtung abgeschaltet ist, sei es gewollt oder durch einen Stromausfall, von der Hardware getrennt werden. Damit wird verhindert, dass diese nicht schleichend entladen und durch Tiefentladung zerstört wird. Deshalb gibt es zwei „HE3321A0400“-Relais [15] welche die Zelle bei Spannungslosigkeit bzw. außerhalb des Messbetriebes allpolig von der restlichen Hardware trennen. Ein drittes „HE3321A0400“-Relais [15] ist nötig, um beim Start einer Messung erst die aktuelle Spannung der Zelle zu messen, den DAC an diese anzupassen und erst danach den Stromkreis zu schließen. Sollte die Spannungsdifferenz zwischen Treiber und Zelle zu groß sein, würde sofort ein unzulässiger Strom fließen und die Zelle eventuell beschädigt werden [15].

3.8.2. Schutz von Bauteilen

Um die Messplatine vor Schäden zu bewahren wurden einige Schutzmaßnahmen ergriffen. Die Eingänge des ADC wurden mit 3,3V-Suppressor-Dioden [33] beschaltet, um diese vor

Überspannung zu schützen. Außerdem wurden am Netzanschluss Dioden eingebaut, die bei einer Verpolung sperren und so die Eingangs-Spannungsregler vor Zerstörung schützen.

3.9. Das Routing

Beim Routing wurde darauf geachtet, Analog- und Digitalteil voneinander zu trennen. Die digitalen Signale sollen von den analogen Signalen abgeschirmt werden, um Einstrahlungen von Oberwellen in den Analogteil zu vermeiden. Die Bauteile, die sowohl analoge als auch digitale Funktionen haben, wie zum Beispiel das digitale SPI[8] und der analoge Messpfad des ADC, sind so aufgebaut, dass Analogpins auf der einen und Digitalpins auf der anderen Seite zu finden sind. So ist es möglich, diese auf die Grenze zwischen den Bereichen zu setzen und Überschneidungen zu vermeiden. Die einzige Überschneidung findet sich bei den Relais, welche an die analogen Leiterbahnen des Ladekreises angeschlossen werden müssen. Das ist jedoch nicht ganz so kritisch, da sich diese nur einmal zu Messbeginn ändern und hinterher keine Störungen mehr verursachen. Zusätzlich zur räumlichen Trennung bekommt jeder Bereich seine eigene Massefläche, diese schirmen Analog- und Digitalteil ab. Ganz besonders wurde darauf geachtet, dass die Taktleitungen des SPI-Busses nicht direkt unter Bauteilen hindurch oder über lange Strecken parallel zu anderen Signalleitungen liegen. Dem Analogteil nähern sie sich nur im rechten Winkel, wenn sie zu einem auf der Grenze liegenden Bauteil führen. Bei den Leiterbahnbreiten wurde für die Energieversorgung eine Breite von 0,3mm bei einer Kupferdicke von $35\mu m$ vorgesehen. Diese vertragen Ströme bis ca. 600mA [30] [6] [5].

[8] SPI - Serial Peripheral Interface

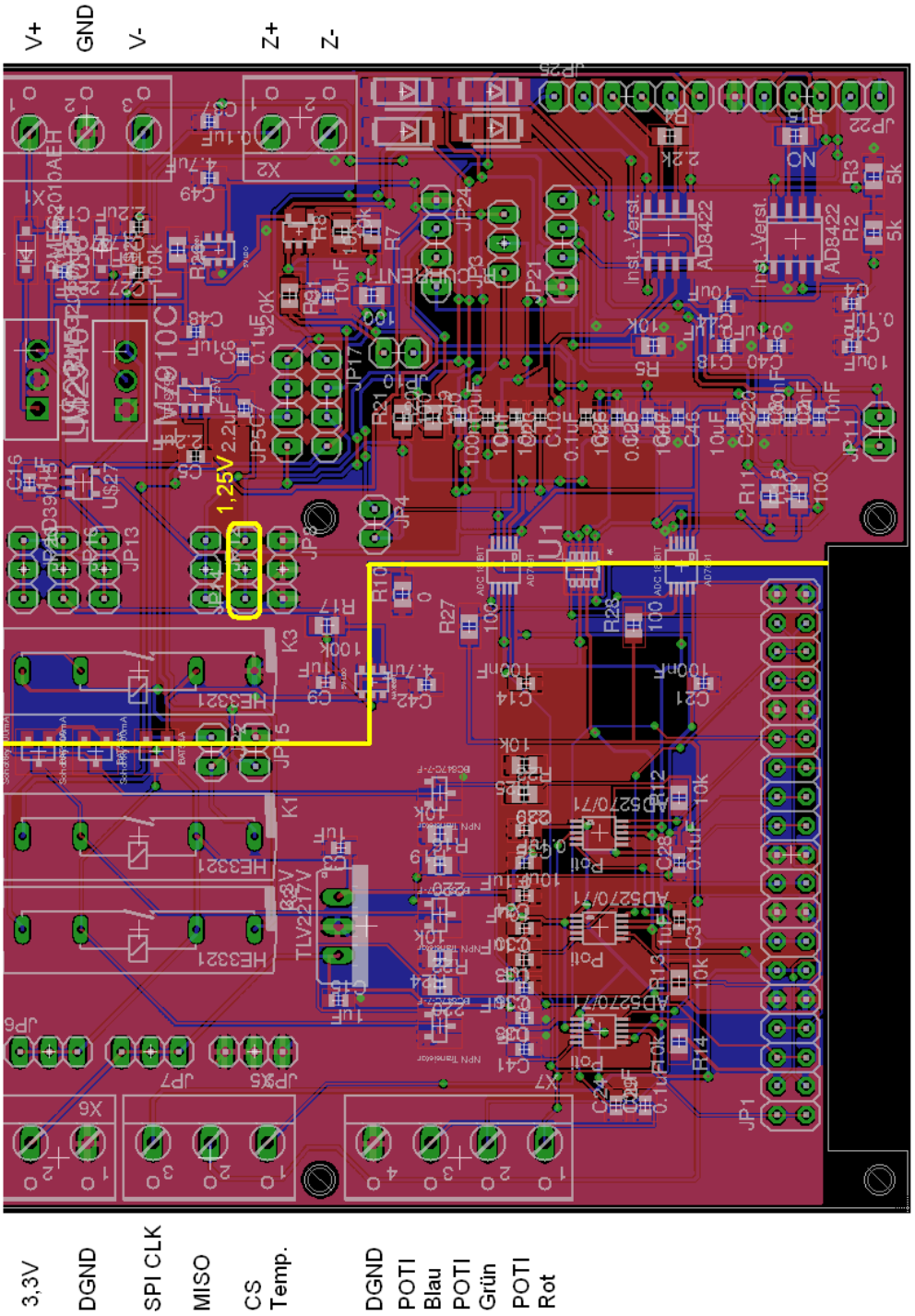

Abbildung 3.8.: Das Bild zeigt das Layout der Erweiterungsplatine. Die gelbe Linie stellt die Grenze zwischen Analog- (rechts) und Digitalteil (links) dar (Bild um 90° nach rechts drehen). Rechts oben befinden sich die Spannungsversorgung und der Zellanschluss. Darunter befindet sich die Messschaltung. Links unten befindet sich die Anschlussleiste für den Raspberry-Pi und darüber die Potentiometer. Am linken Rand befinden sich die Anschlüsse für das Thermometer (X5 und X6) und die Beleuchtungsplatine (X7). Die 1,25V-Referenzspannung wird an den markierten Pin eingespeist.

4. Entwicklung der Software

In diesem Kapitel wird beschrieben, wie das Betriebssystem zu konfigurieren ist, welche Programme benötigt werden und wie diese einzusetzen sind. Da bei dem Messsystem mehrere Programme parallel verschiedene Aufgaben übernehmen, wird ein Steuerskript benötigt, welches den Ablauf koordiniert. Hauptthema wird jedoch die Steuersoftware für die Erweiterungsplatine sein.

4.1. Konfiguration des Linux-Betriebssystems

Ausgangspunkt für die Konfiguration des Betriebssystems ist das Raspbian-Image der Version vom 11.05.2015. Das aktuellste Image kann hier [12] heruntergeladen werden. Auf der DVD dieser Abschlussarbeit wird ein Image zur Verfügung gestellt, das bereits vollständig konfiguriert ist und nur noch auf eine mindestens 4GB große Micro-SD-Karte geschrieben werden muss. Es sollen jedoch an dieser Stelle auch noch mal alle notwendigen Schritte erläutert werden.

Nach dem ersten Booten öffnet sich zunächst das Tool „Raspiconfig“. Dieses kann auch nachträglich mit dem Befehl

- sudo raspi-config

aus der Konsole heraus aufgerufen werden. Hier wird zunächst unter „Expand filesystem“ die Größe des Images an die Größe der SD-Karte angepasst. Optional kann auch ein Passwort vergeben und ein direktes booten in die Desktop-Umgebung aktiviert werden. Auf jeden Fall sollte unter „Internationalisation Options“ das deutsche Tastaturlayout aktiviert werden.

- Internationalisation Options → Change Keyboard Layout → Generic 105-key (Intl) PC → German → Right AltGR → Left Logo Key → Yes

Das deutsche Sprach- und Kodierungs-Paket wird unter „Change Locale“ ausgewählt.

- Internationalisation Options → Change Locale → de_DE.UTF-8 UTF-8 → OK

Nun noch die Zeitzone einstellen.

- Internationalisation Options → Change Timezone → Europe → Berlin

Jetzt müssen noch ein paar Einstellungen unter „Advanced Options“ geändert werden, um den GPIO und den SPI-Bus nutzen zu können.

- Advanced Options → Device Tree → Yes
- Advanced Options → SPI → Yes → OK → Yes → OK

Das System muss neu gebootet werden, um die Einstellungen zu übernehmen. Um einen reibungslosen Messablauf zu gewährleisten, sollte der Dateimanager so eingestellt werden, dass dieser für neu eingehängte Geräte keine Optionen anzeigt. Denn diese werden erst eingehängt, wenn der Nutzer eine der Optionen ausgewählt hat. Wenn während der Messung die Festplatte neu eingehängt wird, was regelmäßig vorkommt, wird die Aufzeichnung der Daten erst nach einer Benutzereingabe fortgesetzt und die Messung somit unbrauchbar.

- Dateimanager → Edit → Preferences → Volume Managment → Show availabel Options → deaktivieren

Außerdem muss die Strombegrenzung für den USB deaktiviert werden, da es sonst passieren kann, dass Geräte abgewiesen werden. Diese bleiben dann ausgehängt und die Messung fällt aus. Um die Strombegrenzung zu deaktivieren, editiert man die Datei „/boot/config.txt“.

- echo "max_usb_current=1" >> /boot/config.txt

Um die Steuer-Software der Messung auf dem System kompilieren zu können, muss noch die Bibliothek für den Multimediaprozessor „bcm2836“, der von dem Raspberry-Pi 2 verwendet wird, installiert werden. Diese wird hier [26] zur Verfügung gestellt und folgendermaßen aus dem Download-Ordner heraus installiert. Das es hier „bcm2835“ und nicht „bcm2836“ heißt, sollte nicht irritieren. Die Bibliothek ist voll kompatibel.

- wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.44.tar.gz
- tar zxvf bcm2835-1.44.tar.gz
- cd bcm2835-1.44
- ./configure
- make
- sudo make check
- sudo make install

Die Version sollte ggf. angepasst werden. Wenn man die Steuersoftware neu kompilieren möchte, muss dem Linker mitgeteilt werden, dass die Bibliothek verwendet werden soll [26] [12].

- gcc -o beispiel beispiel.c -lbcm2835

4.2. Anbindung von Open-Source-Software für die Kamerasteuerung

Für die Steuerung der Bilderfassung durch die Kamera wurde das Kommandozeilenprogramm „fswebcam“ [24] ausgewählt. Dieses gibt die Bilddaten nicht live aus. Was die CPU während der Messung entlastet, jedoch bei der Justierung der Kamera hinderlich ist. Deshalb wird für die Einrichtung einer Messung noch das Programm „luvcview“ [25] hinzugezogen. Dieses gibt die Bilddaten live aus, so dass die Kamera vor dem Messstart komfortabel scharf gestellt werden kann. Beide Programme sind nicht Bestandteil der Standardinstallation der Distribution „Raspbian“ und müssen entsprechend nachinstalliert werden.

- sudo apt-get install fswebcam luvcview

Das Programm „luvcview“ wird aus der Konsole heraus geöffnet, benötigt keine weiteren Aufrufinformationen und kann nach Justierung der Kamera geschlossen werden.

- luvcview

Der Aufruf des Programms „fswebcam“ hingegen fällt aufwändiger aus und soll an dieser Stelle genauer erläutert werden.

- fswebcam -d /dev/video0 - -loop 10 -F5 - -png 0 - -scale 1600x1200 - -set “White Balance Temperature=False“ - -save $ORDNER/’%s’.png

Nach dem Programmnamen wird mit der ersten Option „-d“ die Datei angegeben, welche die Kamera im Dateisystem repräsentiert. Die Option „- -loop“ legt die Zeit zwischen zwei Aufnahmen in Sekunden fest. Anschließend wird mit der Option „-F“ die Anzahl der Aufnahmen festgelegt, aus denen, zwecks Rauschunterdrückung, der Mittelwert für das letztendlich gespeicherte Bild gebildet wird. Im Beispiel sind es fünf. Das Bildformat wird mit der Option „--png“ und der Übergabe einer Null auf das PNG-Format ohne Kompression eingestellt, denn eine Kompression bedeutet einen Informationsverlust. Die Auflösung von 1200x1600 wird mit der Option „- -scale“ übergeben. Jetzt muss die Kamera noch davon abgehalten werden, einen Weißabgleich vorzunehmen, denn die Bilder sollen nicht auf „Schönheit-für-das-menschliche-Auge“ sondern auf ihr optisches Spektrum in harten Zahlen untersucht werden. Dafür gibt es die Option „- -set“. Diese bietet unterschiedliche Parameter, welche auch von

der verwendeten Kamera abhängen. Man kann sich einen Überblick über die, von der aktuellen Konfiguration von Hard- und Software, angebotenen Parameter verschaffen.

- fswebcam - -list-controls

Ist der Weißabgleich der Kamera abschaltbar, so erscheint in der Auflistung der Punkt „White Balance Temperature“. Diesen kann man, wie oben geschehen, der Option „- -set“ übergeben und auf „False“ setzen. Nun werden die Bilder unverfälscht in dem Verzeichnis, das mit der Option „- -save“ übergeben wird, abgelegt. Bei der Angabe des Speicherverzeichnisses empfiehlt es sich ein „%s“ einzubauen, wie es oben geschehen ist. Auf diese Weise werden die Bilder gleich mit einem Zeitstempel im Namen versehen, der eine zeitliche Zuordnung ermöglicht [24] [25].

Für die Auswertung der Bilddaten gab es bereits Tools und Matlabskripte aus vorangegangenen Arbeiten (z.B. Grießbach [14] und Palliyaguruge [28]). Sämtliche Mess- und Bilddaten sind so beschaffen, dass diese von den vorhandenen Tools und Matlabskripten ausgewertet werden können.

4.3. Steuersoftware für das Steuer- und Messmodul auf dem Einplatinen-Computer

Die Erweiterungsplatine für den Raspberry-Pi benötigt eine Software, welche die Zyklierung regelt und die gemessenen Daten in Dateien speichert. Dazu muss es eine Möglichkeit geben, eine Art Ablaufplan zu hinterlegen, an dem sich das Steuerprogramm orientieren kann. In diesem Ablaufplan muss es Informationen zu allen einstellbaren Parametern geben. Da es kaum möglich oder zumindest unkomfortabel ist, einen Plan über einen Zeitraum von mehreren Monaten händisch zu erstellen, muss es eine Schleifenfunktion geben, die Abläufe wiederholt. Die folgenden Parameter müssen dem Steuerprogramm zur Verfügung gestellt werden.

- Die aktuell zu erreichende Schwellenspannung, auf welche die Zelle ge-/entladen werden soll.
- Der Strom, mit dem die Zelle ge-/entladen werden soll.
- Die Einstellung für die LED, welche die Elektrode beleuchten.
- Die Dauer der Ruhezeit zwischen einem Lade- und einem Entladevorgang.
- Die Anzahl der Wiederholungen.
- Die zum Speichern der Messdaten zu verwendenden Dateien.

Es gibt mehre Wege, dem Programm diese Informationen zukommen zu lassen. Zum einen können diese in einer Datei hinterlegt werden, zum anderen kann das Programm mit entsprechenden Parametern in der Kommandozeile aufgerufen werden. Für die Speicherdateien und die Anzahl der Wiederholungen bietet sich die Kommandozeile an, da das Informationen sind, die sich oft von einer Zyklierung zur anderen ändern. Für die restlichen Informationen ist eine Steuerdatei besser geeignet, da diese Parameter eventuell öfter verwendet werden sollen und so auch gleich archiviert sind. Da die Messung über lange Zeiträume allein gelassen wird, könnte es passieren, dass Fehler in der Steuerdatei lange unentdeckt bleiben. Um Zeit und Materialverluste (z.B. durch ungewollte Tiefentladung) zu verhindern, sollte es einen Plausibilitätscheck geben. Dieser könnte z.B. eine Tiefentladung, welche ja grundsätzlich möglich sein soll, noch einmal hinterfragen.

Neben dem Ablauf als solchem, muss der Zellstrom überwacht und bei Bedarf angepasst werden. Dafür muss eine Abtastung gefunden werden, die diese Aufgabe ermöglicht, dabei die CPU jedoch so wenig wie möglich in Anspruch nimmt. Das bedeutet einen Kompromiss zwischen Abtastung und Ruhemodus. Um die Messgrößen zu erfassen und die Stellgrößen zu beeinflussen, muss die Software in der Lage sein, mit dem Stell- und Messgliedern kommunizieren zu können. Der Raspberry-Pi bietet zu diesem Zweck eine GPIO-Leiste an, über die u.a. auch ein SPI-Bus betrieben werden kann [3].

Im Folgenden soll nun der Programmablauf erläutert werden. Ergänzend kann der ausführlich kommentierte Quellcode im Anhang A.3.8 auf Seite 104 angeschaut werden. Das Programm besteht aus zwei Teilen, dem Initialisierungsteil sowie dem Mess- und Regelteil.

4.3.1. Initialisierungsphase

In der Initialisierungsphase werden zunächst die Übergabeparameter des Programmaufrufs ausgewertet und auf Vollständigkeit geprüft. Dazu bietet die GNU-Bibliothek die Funktion „getopt()“, welche hier [13] dokumentiert ist. Diese Funktion wertet in Kombination mit einer „Switch-Case-Anweisung“ die Übergabeparameter aus und fängt fehlerhafte Eingaben ab. Es werden die Speicherorte für die Temperatur-, Strom- und Spannungswerte, der Speicherpfad des Zyklierplans und die Anzahl der gewünschten Wiederholungen des Plans erwartet. Eine Übergabe von Speicherorten, die nicht erreichbar sind, wird beim Versuch die Dateien zu öffnen, abgefangen. Übergibt der Anwender zu wenig Parameter, bekommt er eine Betriebsanleitung angezeigt. Wenn alle Dateien geöffnet werden konnten, wird die Kommunikation mit der Erweiterungsplatine initialisiert. Hierfür wurde die Funktion „init()“ geschrieben. In dieser wird die GPIO-Schnittstelle und die SPI-Kommunikation initialisiert, dies wird im Kapitel 4.4.1 noch näher beschrieben. Außerdem werden die beiden ADC ausgelesen, die Daten jedoch verworfen, damit diese bei der ersten Messung bereits korrekte Ergebnisse liefern. Auch die Potentiometer für die Ansteuerung der Beleuchtungsdimmung müs-

sen initialisiert werden, um das Eingangsregister beschreibbar zu machen. Zum Schluss wird noch die Funktion „dacInit()“ aufgerufen, welche sowohl bei der Initialisierung als auch beim Mess- und Regelbetrieb eine wichtige Rolle spielt. Denn sie sorgt dafür, dass die DAC-Treiber-Ausgangsspannung (Ausgang des Operationsverstärkers) der Zellspannung angeglichen wird. Dafür werden zunächst nur die Relais an den Zellpolen geschlossen, so dass die Zellspannung gemessen werden und der DAC auf diese eingestellt werden kann. Ließe man diesen Schritt aus, könnte es beim Schließen des Strompfadrelais zu unzulässigen Strömen kommen. Sind alle Schritte der Funktion „init()“ erfolgreich gewesen, so gibt diese eine Eins zurück, wenn nicht, eine Null und das Programm wird beendet [6] [4].

4.3.2. Mess- und Regelbetrieb

Nach der erfolgreichen Initialisierungsphase beginnt der Mess- und Regelbetrieb. Dessen Ablauf ist nicht so linear wie bei der Initialisierung, deshalb wird mit Bild 4.1 auf Seite 55 ein Funktionslaufplan zur besseren Verständlichkeit beigefügt.

Nach der Initialisierungsphase wird als erstes die Funktion „getValue()“ aufgerufen. Diese ist dafür zuständig, die nächsten Sollwerte für die zu erreichende Spannung, den Lade-/Entladestrom, die Ruhezeit für die Zelle nach dem Lade-/Entladevorgang sowie die LED-Dimmung aus dem Steuerskript auszulesen. Anschießend wird die nächste zu erreichende Schwellspannung mit der aktuellen Zellspannung verglichen, um festzulegen, ob es sich um einen Lade- oder Entladevorgang handelt. Dann werden zwei Flags entsprechend gesetzt. Handelt es sich um eine Entladung, muss das Vorzeichen des Stroms geändert werden. Denn der Strom wird in der Steuerdatei stets positiv übergeben. Die Negation erfolgt mit der Formel 4.1. Dabei wird vom positiven Sollwert der doppelte positive Sollwert subtrahiert, um z.B. von $20\mu A$ auf $-20\mu A$ zu kommen. Die Subtraktion von 65535 dient dem Zweck, den Nulloffset nicht vom Sollwert mit zu subtrahieren. Denn sowohl positive als auch negative Ströme werden durch positive Integer repräsentiert. Dazu ist die Nulllinie auf 65535 angehoben worden (Siehe Kapitel 4.4.6 und 3.6.1) und muss für absolute Stromwerte entsprechend berücksichtigt werden.

$$I_{Integer_{soll}} = I_{Integer_{soll}} - 2 \cdot (I_{Integer_{soll}} - 65535) \tag{4.1}$$

Nachdem die aktuellen Sollwerte ausgewertet wurden, wird der Strompfad geschlossen und sofort die ersten Messdaten aufgenommen. Für die Spannungsmessung wird einfach der Spannungs-ADC ausgewertet. Denn die Spannung wird nur im hundertstel Volt-Bereich erfasst und erfährt eine Glättung durch die Kapazität der Zelle. Die Strommessung hingegen sollte möglichst eine Glättung im Nanoampere-Bereich haben. Deshalb wird für die

Strommessung ein Mittelwert gebildet[1], was nichts anderes als einen Tiefpass darstellt. Die Mittelwertbildung findet sogar auf zwei Ebenen statt. Die erste Ebene befindet sich in der Funktion „getI()“. Die Funktion „getI()“ liest den Strom-ADC aus. Sie tut dies jedoch mehrmals und verwirft dabei alle Werte, die außerhalb eines Fensters um den letzten Stromwert liegen, der beide Mittelwertbildungen durchlaufen hat. Wenn sie eine vorgegebene[2] Anzahl der Fensterbedingung entsprechenden Werte gefunden hat, bildet sie aus diesen einen Mittelwert. Eine große Anzahl zu sammelnder Werte hat jedoch den Nachteil, dass die Messdauer stark ansteigt, denn es müssen immer erst gültige Werte gefunden werden. Um trotzdem über eine große Anzahl von Werten mitteln zu können, wurde die zweite Ebene zur Mittelwertbildung eingeführt. Für diese wird der Rückgabewert von „getI()“ in ein LILO[3]-Array gespeichert, welches eine einstellbare Anzahl von Stromwerten enthält. Diese Mittelwertbildung hat keine zufällige Zeitkomponente und kann daher über viele Werte erfolgen, ohne die Laufzeit allzusehr zu verlängern. Allerdings erkauft man diese Laufzeitoptimierung mit einer Phasenverschiebung zwischen Messwert und tatsächlichem Strom. Bei großen Stromänderungen (z.B. bei Start eines Ladevorgangs) kann eine große Phasenverschiebung zum Aufschwingen des Systems führen. Um das zu verhindern, wurde die maximale Schrittweite der DAC-Änderung begrenzt.

Der gemessene Strom wird geprüft, ob dieser erhöht oder gesenkt werden muss. Zum Erhöhen des Stroms wurde die Funktion „rise()“ und zum Senken die Funktion „sink()“ geschrieben. Beide Funktionen arbeiten zunächst völlig identisch. Sie bekommen den zuvor gemessenen Strom übergeben und berechnen die Differenz von Soll- und Istwert. Diese Differenz stellt zunächst den Wert dar, um den der aktuelle DAC-Registerwert verändert werden soll. Zu große Differenzen würden jedoch dazu führen, dass die Fensterbedingung bei der nächsten Strommessung nicht mehr eingehalten werden kann. Außerdem kann das System, aufgrund der eben erwähnten Phasenverschiebung, bei zu schnellen Änderungen aufschwingen. Deshalb wird die Differenz, wenn sie einen Grenzwert übersteigt, beschränkt. Die Beschränkung ist jedoch davon abhängig, wie weit Soll- und Ist-Wert auseinander liegen. Ist die Differenz sehr groß, werden große Schritte zugelassen, ist sie kleiner, werden die Schritte kleiner. Auf diese Weise kann der Einschwingvorgang verkürzt werden. Dann kommt der Schritt bei dem sich „rise()“ und „sink()“ unterscheiden. Bei „rise()“ wird der ermittelte Stellwert mit dem aktuellen DAC-Registerwert addiert und bei „sink()“ von diesem subtrahiert. Nun erfolgt innerhalb der Funktionen eine erneute Strommessung, um zu prüfen, ob der neue Strom innerhalb einer Hysterese um den Sollwert liegt. Wenn nicht, wird der Vorgang mit dem neu gemessenen Strom wiederholt. Jedoch höchstens zehn mal, denn es muss gewährleistet werden, dass die Aufzeichnung der Messdaten rechtzeitig erfolgen kann. Für die Aufzeichnung der Messdaten wird ermittelt, wie viel Zeit seit der letzten

[1] Dies wurde aufgrund massiver Störeinstrahlungen nötig.
[2] Siehe Headerdatei „platine.h“ im Anhang A.3.7.
[3] LILO - Last In Last Out

Aufzeichnung verstrichen ist. Ist die eingestellte Abtastzeit vergangen, werden die aktuellen Strom-, Spannungs- und Temperaturwerte gespeichert.

Am Ende einer Abarbeitung von Messen, Speichern und Regeln wird geprüft, ob der aktuelle Spannungsschwellwert erreicht wurde. Wenn das der Fall ist, wird der Zelle die gewünschte Ruhezeit gegeben, geprüft, ob der Zyklierplan vollständig abgearbeitet wurde und ob noch eine Wiederholung des Plans erfolgen soll.

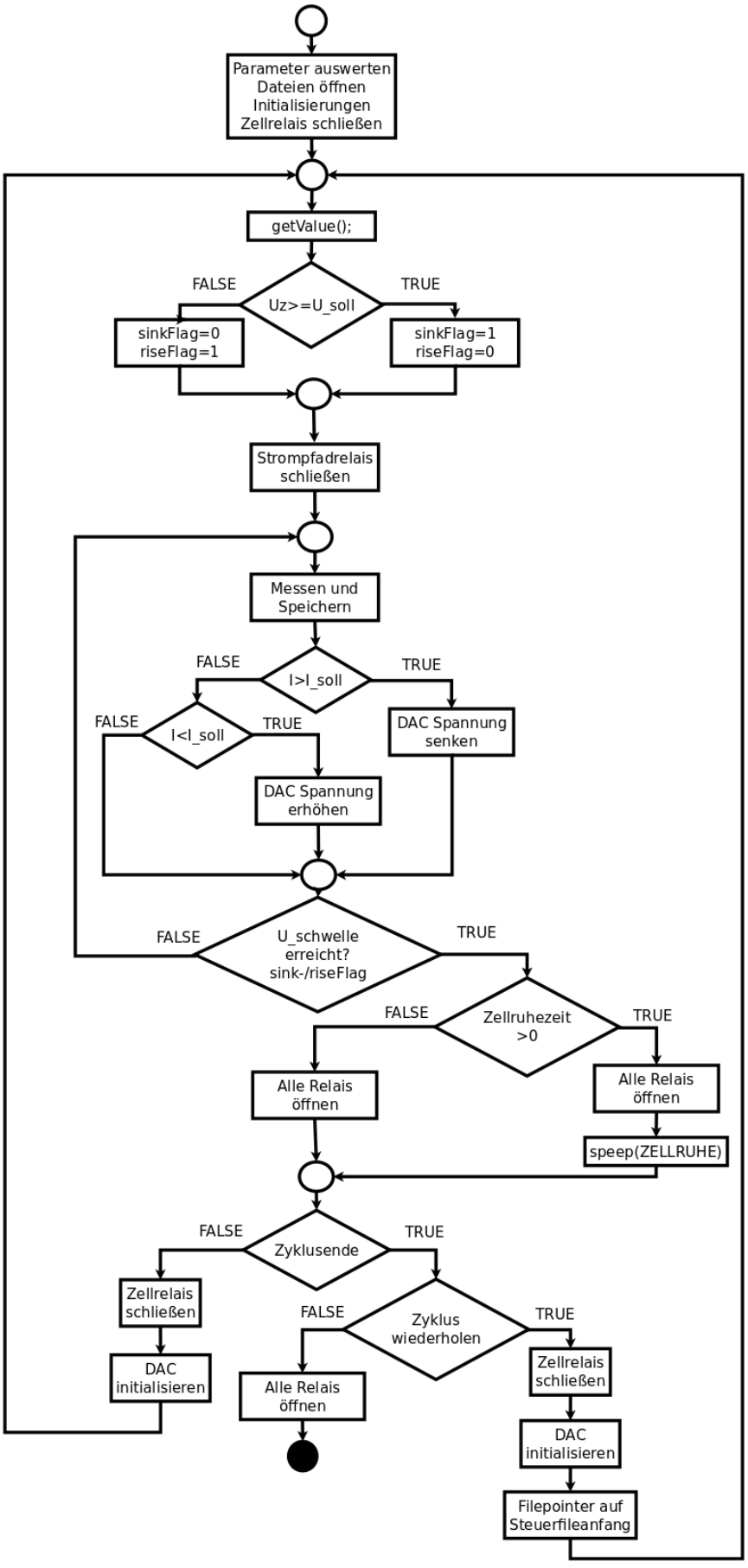

Abbildung 4.1.: Funktionslaufplan der Regel- und Messschleife

4.4. GPIO-Schnittstelle und das SPI

Nachdem der allgemeine Programmablauf dargestellt wurde, soll nun beschrieben werden, wie die Kommunikation zwischen Master und Slaves abläuft. Für die GPIO-Schnittstelle wird hier [26] eine Bibliothek bereitgestellt. Wie diese dem Raspberry-Pi zur Verfügung gestellt wird, wurde bereits im Kapitel 4.1 behandelt. Die GPIO-Schnittstelle wird für die Kommunikation zwischen dem Raspberry-Pi und der Erweiterungsplatine benötigt. Der Pin-Header hat 40 Kontakte (Bild 4.2 auf Seite 58) von denen 21 als GPIO verwendet werden können. Fünf davon sind jedoch bereits für das SPI reserviert. Einer für den seriellen Takt, einer zum Daten senden, einer zum Daten empfangen und zwei Chip-Select-Leitungen. Eine Chip-Select-Leitung wählt einen Slave aus, indem sie ihr Potential auf LOW zieht, und initiiert so die Kommunikation zwischen Master (Raspberry-Pi) und dem Slave (z.B. ADC). Das bedeutet, dass jeder Slave seine eigene Chip-Select-Leitung braucht und die zwei, die standardmäßig vorgesehen sind, nicht ausreichen. Deshalb werden die „normalen“ GPIO-Pins als Chip-Select verwendet, was die maximale Anzahl Slaves auf 18 erhöht. Tabelle 4.1 auf Seite 57 zeigt die Belegung des GPIO-Headers und die Zustände nach dem booten [8] [9].

Hinweis: Im Anhang A.6 auf Seite 128 befindet sich ein Kurzanleitung für die Verdrahtung von Raspberry-Pi und Erweiterungsplatine.

4.4.1. Initialisierung

Während der Initialisierungsphase werden, nach der Aktivierung der GPIO-Funktion des Multimediaprozessors „bcm2835“, die GPIO-Pins zunächst als Ausgänge deklariert. Da die verwendeten GPIO-Pins nach dem Booten erstmal LOW sind, müssen die Chip-Select-Pins auf HIGH gesetzt werden, denn dies ist deren Idle. Die Relais-Pins und der Konvertierungspin für die ADC werden jedoch trotz ihres LOW-Defaults im Programm nochmal kontrolliert auf LOW gesetzt, um nichts dem Zufall zu überlassen. Anschießend wird der SPI-Bus aktiviert. Dazu gehört die Festlegung der Byte-Priorität auf das MSB[4], die Taktphase, deren Idle auf LOW eingestellt wird und die Flanke, auf deren Auftreten hin die Daten übernommen werden sollen. In diesem Fall die fallende. Dann stellt man noch die Frequenz der Übertragung ein. Diese wurde auf 3,82kHz eingestellt. Die Bauteile lassen auch höhere Übertragungsraten zu, die 3,82kHz sind jedoch schnell genug und schnelle Übertragungen sind tendenziell störanfälliger. Die Initialisierung wird in der Headerdatei „init.h“ definiert welche im Anhang A.3.1 auf Seite 94 angeschaut werden kann.

[4] MSB -Most Significant Bit

GPIO	Default	Funktion	Pin
17	LOW	CS ADC1	11
27	LOW	CS ADC2	13
23	LOW	Convert ADC	16
22	LOW	CS DAC	15
10	LOW	MOSI	19
09	LOW	MISO	21
11	LOW	SCLK	23
13	LOW	LED Poti R	33
19	LOW	LED Poti G	35
26	LOW	LED Poti B	37
16	LOW	Relais Z1	36
20	LOW	Relais Z2	28
21	LOW	Relais I	40

Tabelle 4.1.: Die Tabelle zeigt, für welche Aufgabe die einzelnen GPIO-Pins verwendet werden. Man kann sehen, dass die Nummerierung der GPIO (linke Spalte) eine andere ist als die der Hardware-Pins (rechte Spalte). Für die Verwendung der „bcm2835-Bibliothek“ ist die Hardware-Pin-Nummer entscheidend. Die komplette GPIO-Leiste zeigt das Bild 4.2 auf Seite 58 [8].

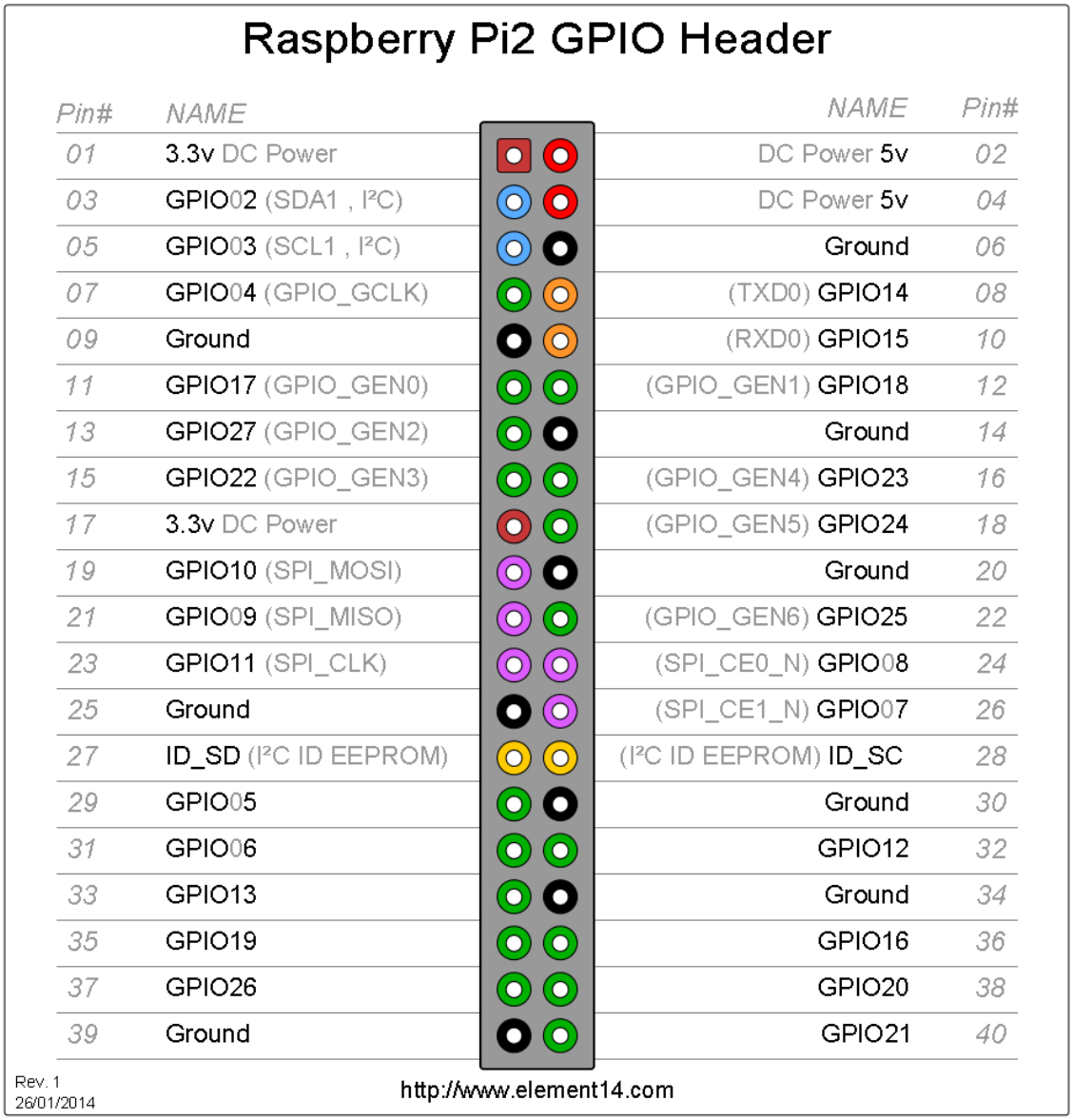

Abbildung 4.2.: Das Bild zeigt die GPIO-Leiste des Raspberry-Pi 2. Am linken und rechten Rand ist die Nummerierung der Hardware-Pins zu sehen und in der Spalte „Name“ die primäre Funktion der Pins. Die ausgegrauten Bezeichnungen zeigen die Sonderfunktionen, die einige der Pins anstelle ihrer primären Funktion übernehmen können [8].

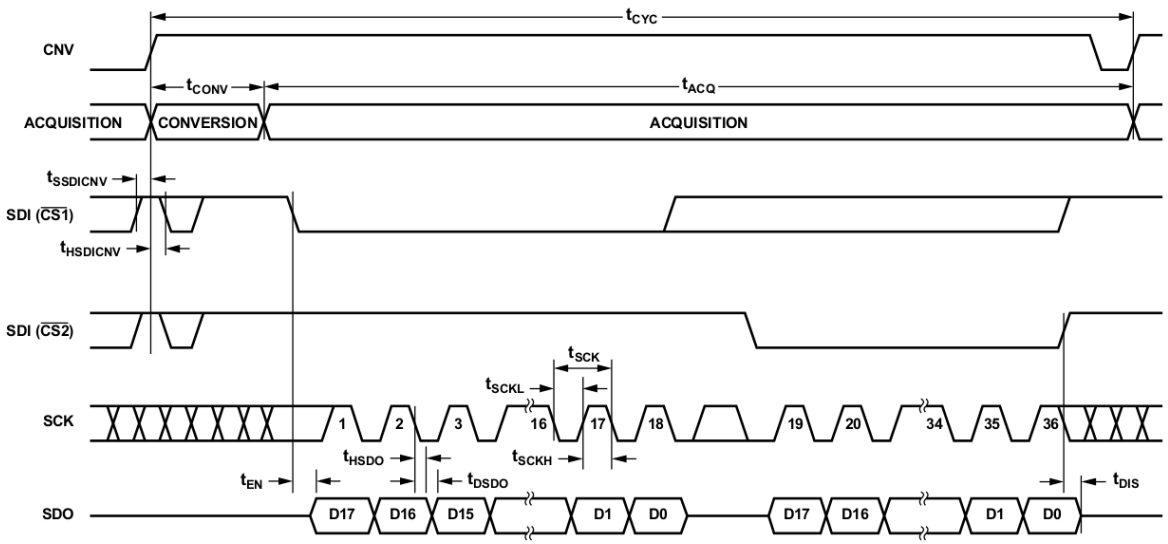

Abbildung 4.3.: Das Bild zeigt eine gültige Abfrage zweier ADC. Dazu wird zunächst der Convert-Pin auf HIGH gezogen und bis nach dem Auslesen der Daten auf HIGH gelassen. Der Convert-Pin spricht alle ADC gleichzeitig an. Die Aufforderung zur Datenübertragung kommt für jeden ADC separat über den Chip-Select-Pin, welcher dazu auf LOW gebracht wird. Dann wird mit jeder fallenden Taktflanke (SCK) ein Bit übertragen. Ist die Datenübertragung für beide ADC erfolgt, kann der Convert-Pin zurück auf LOW gebracht werden [6].

4.4.2. Auslesen des Analog-Digital-Wandlers

Für die Messung von Strom und Spannung braucht es ADC, welche die analogen Signale in digitale übersetzen. Der verwendete ADC „AD7691“ [6] wurde bereits im Kapitel 3.6.2 vorgestellt. In diesem Kapitel soll gezeigt werden, wie dieser ausgelesen wird. Für die Datenübertragung wird der in Kapitel 4.4 erwähnte SPI-Bus verwendet. Der ADC wird über eine steigende Flanke an seinem „Convert-Pin“ dazu aufgefordert, die Eingangsspannung am Eingang in einen Digitalwert zu übersetzen. Nach einer Übersetzungszeit von $3,7\mu s$ kann der Wert über den SPI-Bus ausgelesen werden. Dabei werden drei Byte, sprich 24 Bit, übertragen. Da der ADC jedoch nur eine 18-Bit-Auflösung hat, werden sechs Bit nicht benötigt. Weil es sich dabei um die niederwertigsten Bit handelt, muss der übertragene Datensatz noch um sechs Bit „nach unten“ verschoben oder, anders ausgedrückt, durch 2^6 dividiert werden. Bild 4.3 zeigt eine gültige Abfrage zweier ADC mit einem SPI-Bus. Der C-Code für den ADC kann in der Header-Datei „adc.h“ im Anhang A.3.2 auf Seite 95 angeschaut werden. Der ADC sollte zu Initialisierungszwecken einmal konvertiert und ausgelesen werden, denn der erste Wert nach dem Einschalten der Versorgungsspannung kann fehlerhaft sein [6].

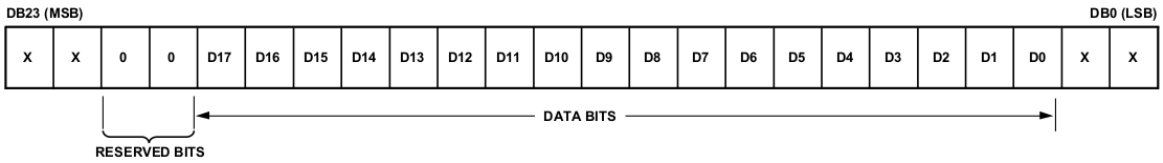

Abbildung 4.4.: Das Bild zeigt das Eingangsregister des DAC „AD5680“. Die niederwertigsten Bit (rechts) und die höchstwertigsten Bit (links) werden nicht ausgewertet. Die „Reserved Bit“ müssen mit Nullen beschrieben werden. Die Bit D0 bis D17 beinhalten die Nutzdaten [5].

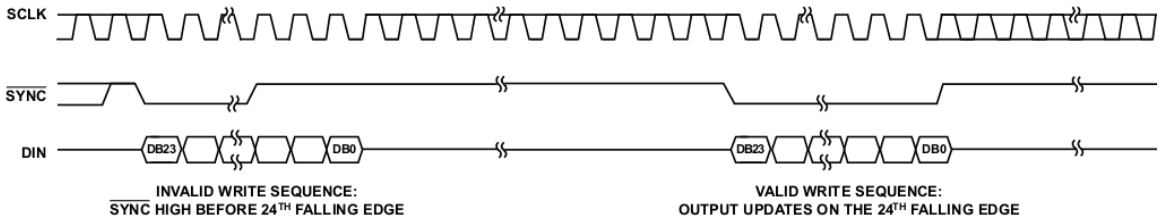

Abbildung 4.5.: Das Bild zeigt eine ungültige Aktualisierung des DAC-Eingangsregisters und eine gültige. Bei der ungültigen Aktualisierung wird der Chip-Select zu früh in den Idle gebracht [5].

4.4.3. Steuern des Digital-Analog-Wandlers

Um den Strom beeinflussen zu können, braucht es einen Digital-Analog-Wandler. Der verwendete DAC wurde bereits im Kapitel 3.5.1 vorgestellt. Hier geht es nun um die Ansteuerung das DAC. Für die Übertragung der Daten wird der „Chip-Select-Pin“ auf LOW gebracht und anschließend drei Byte an Daten übertragen. Von diesen drei Byte sind jedoch nur 18 Bit Nutzdaten. Die ersten und letzten zwei Bit sind „Don’t-care-Bit“ und die beiden vorletzten MSB sind reservierte Bit, die mit Null beschrieben werden sollen. Bild 4.4 zeigt eine Darstellung des Eingangsregisters. Bevor man die Daten sendet, müssen diese an die richtige Position geschoben werden und die nicht zu beschreibenden Bit mit Nullen „UND-verknüpft“ werden. Der Code für die Datenübertragung an den DAC kann im Headerfile „dac.h“ im Anhang A.3.3 auf Seite 96 angeschaut werden [5].

4.4.4. Auslesen des Temperatursensors

Ein Sendevorgang mit dem verwendeten Temperatursensor LM74 benötigt 32 Takte. Von diesen sind jedoch nur die ersten 16 Bit für die Übertragung der Temperaturdaten gedacht. Die anderen 16 können genutzt werden, um dem Sensor Daten zu übermitteln. So kann dieser dazu veranlasst werden, den „Shut-Down-Mode“ zu aktivieren oder Herstellerdaten auszugeben. In diesem Projekt werden nur Temperaturdaten aus dem LM74 ausgelesen. Dennoch müssen die 32 Takte angelegt werden, weil sonst keine erneute Temperaturmessung erfolgt. Dafür wird der „Chip-Select-Pin“ auf LOW gezogen und die Daten mit jeder fallenden Takt-Flanke bitweise ausgelesen. Die Daten, die ausgewertet werden sollen, stehen jetzt in den beiden MSB des integer Datentyps, in dem diese gespeichert wurden. Für eine korrekte Auswertung werden diese jetzt um 19 Bit in Richtung LSB „geshifted“. Die 19 Bit ergeben sich aus den ungenutzten 16 Bit, die für die Datenübertragung zum Sensor gedacht sind, und weiteren drei Bit, die nicht genutzt werden, da der Sensor nur eine 16 Bit Auflösung hat. Da mit den Integer-Werten nicht weiter gerechnet werden soll, werden diese in der Auslesefunktion auch gleich in anwenderfreundliche Double-Werte konvertiert. Das geschieht, indem diese mit dem Temperaturwert des LSB von 0,0625°C multipliziert werden. Die Auslesefunktion ist in der Headerdatei „temp.h“ definiert und kann im Anhang A.3.6 auf Seite 100 angeschaut werden [18].

4.4.5. Steuern des Digital-Potentiometers

Das Eingangsregister der Digital-Potentiometer zum Steuern der LED-Dimmung ist zwei Byte groß. Die ersten[5] 10 Bit des Potentiometers „AD5270“ sind „Daten-Bit“, gefolgt von vier „Befehls-Bit“. Die restlichen zwei Bit sind „Don’t-Care-Bit“. Bei dem Potentiometer „AD5271“ ist zu beachten, dass aufgrund der kleineren Auflösung die ersten beiden LSB ebenfalls „Don’t-Care-Bit“ sind. Die Digital-Potentiometer „AD5270“ und „AD5271“ besitzen noch weitere Register, in denen Werte gespeichert und bei Bedarf gesetzt werden können. Dies ist der Default-Betrieb, welcher jedoch nicht verwendet werden soll, da die Steuerung ausschließlich über das Steuerskript erfolgt. Um das Eingangsregister direkt mit Daten beschreiben zu können, muss dieser Betriebsfall aktiviert werden. Dazu ist der Befehl 0x1C02 an das Potentiometer zu senden (siehe auch [4, Seite 19-20]). Um Daten an das Digital-Potentiometer zu senden, muss dessen „Chip-Select-Pin“ auf LOW gezogen werden. Dann werden die Daten bitweise mit den fallenden Takt-Flanken übernommen [4].

[5] Vom LSB aus betrachtet.

4.4.6. Konvertierung von Double- und Integer-Werten

Um die Anwenderfreundlichkeit zu erhöhen, sollen die Spannungsschwellwerte und Lade-/Entladeströme als Gleitkommazahl über- und ausgegeben werden. Denn die Auswertung der Integer-Werte ist wenig intuitiv. Dafür wurden Funktionen zur Konvertierung geschrieben. Die Funktionen können im Anhang A.3.4 auf Seite 97 angeschaut werden.

Die Typenkonvertierung der Spannungswerte vom Typ Integer zum Typ Double erfolgt mit der Formel 4.2. Die Multiplikation der Referenzspannung mit dem Faktor zwei ist dem nicht invertierenden Verstärker geschuldet, welcher dafür sorgt, dass die Aussteuergrenze des DAC von 2,5V auf bis zu 5V angehoben wird. Der Betrag von 131071 ist der Maximalwert, den der ADC mit den zur Verfügung stehenden Bit annehmen kann .

$$U_{Double} = 2 \cdot U_{Ref} \cdot \frac{U_{Integer}}{131071} \tag{4.2}$$

Die Umrechnung der Spannung von Double- zu Integer-Werten erfolgt mit der Formel 4.3.

$$U_{Integer} = \frac{U_{Double}}{2 \cdot U_{Ref}} \cdot 131071 \tag{4.3}$$

Die Umrechnung des Stroms von Integer- zu Double-Werten erfolgt mit der Formel 4.4. Die Subtraktion des Integer-Wertes mit 65535 dient dem Zweck, den Nulloffset auszugleichen, welcher bei 1,25V liegt (siehe Kapitel 3.6.1). Der Wert 52427 entspricht der maximal zu erwartenden Aussteuerung durch den Instrumentenverstärker bei $|I| = 1000\mu A$ ohne Nullpunktoffset (siehe Seite 42, Gleichung 3.28). Das bedeutet für den Bruch einen Wertebereich von 1 bis -1. Die Skalierung mit $1000\mu A$ sorgt für die entsprechende Darstellung des Stromes mit der Einheit $[\mu A]$.

$$I_{Double} = \frac{I_{Integer} - 65535}{52427} \cdot 1000\mu A \tag{4.4}$$

Die Transformation von Double- zu Integer-Strömen erfolgt entsprechend nach Formel 4.5.

$$I_{Integer} = \frac{I_{Double} \cdot 52427}{1000\mu A} + 65535 \tag{4.5}$$

4.5. Steuerskript zur Verwaltung der Messung

Da die Messung aus dem Zusammenspiel mehrerer Programme besteht, liegt es nahe, ein Shell-Skript zu verwenden, dass einen Messvorgang startet und beendet. So muss der Anwender nicht jedesmal selbst an alle Anwendungen denken und eine zeitgleiche Ausführung aller Programme ist auch gewährleistet. Außerdem wird die Verzeichnisverwaltung mit Variablen übersichtlicher und es passiert nicht so leicht, dass eine Messung in das Verzeichnis einer anderen geschrieben wird. Sollte dies dennoch geschehen, können die Daten trotzdem gerettet werden, da die Steuersoftware die Dateien nicht neu anlegt, sondern neue Daten anhängt. Dann ist ein wenig Handarbeit gefragt. Auch das Beenden der Messung wird mit dem Skript gesteuert. Es wartet nach dem Start auf einen beliebigen Tastendruck. Wenn dieser kommt, werden die entsprechenden Signale zum Beenden an die Prozesse gesendet. Das Skript kann im Anhang A.5 auf Seite 126 angeschaut werden.

5. Test

In diesem Kapitel soll nun die Funktion des Messaufbaus überprüft, Schwachstellen gefunden und eine Aussage bezüglich der Genauigkeit und der Performance des Systems gemacht werden.

5.1. Inbetriebnahme der Erweiterungsplatine

Nach der Fertigstellung der Erweiterungsplatine wurde diese mit einem Widerstandsmessgerät zunächst auf Kurzschlüsse in der Spannungsversorgung geprüft. Nach deren Ausschluss, konnte die Platine mit Spannung versorgt werden. Danach galt es sicher zu stellen, dass alle Spannungsebenen auch wirklich das vorgesehene Spannungsniveau hatten. Das ist besonders für den Raspberry-Pi wichtig, da dieser zerstört würde, wenn seine GPIO-Pins mit mehr als 3,3V konfrontiert werden. Dabei stellte sich heraus, dass der Spannungsregler für die 1,25V Referenzspannung sehr ungenau ist. Er regelte die Spannung auf ca. 1,33V. Deutlich gravierender war jedoch, dass die 3,3V-Digital-Spannungsversorgung, eben die für die Kommunikation mit dem Raspberry-Pi, eine Spannung von 4,35V führte. Da die nächst höhere Spannungsebene die 5V-Spannungsebene ist, kam sofort der Verdacht auf, dass hier eine Diode am Fehlerfall beteiligt ist. Daraufhin wurde geprüft welche Bauteile von beiden Spannungsebenen versorgt werden. Das traf nur auf die Potentiometer für die LED-Dimmung zu. Deren digitales Interface wird mit 3,3V versorgt und der Lastkreis mit 5V. Also wurde noch mal das Datenblatt des Potentiometers [4] geprüft, welches dann auch das Problem offenbarte. Bei der Entwicklung der Platine war übersehen worden, dass es zwischen den Versorgungspins von digitalem Interface und Lastkreis eine ESD-Schutzdiode gibt. Diese wird leitend sobald die Lastkreisversorgung ein höheres Spannungsniveau erreicht als die Interfaceversorgung. Die Problemlösung war, die 5V Spannungsversorgung zu trennen und den Lastkreis des Potis mit 3,3V zu versorgen. Bild 5.1 auf Seite 65 zeigt die Stellen, an denen die Platine überarbeitet wurde. Für die Kompatibilität mit der LED-Platine, welche von den Potentiometern angesteuert werden soll, ergaben sich dadurch einige Änderungen, die sich jedoch auf Widerstandsänderungen beschränkten.

Nachdem die Spannungsversorgung ordnungsgemäß in Betrieb genommen worden war, konnten nun die einzelnen Funktionsgruppen auf ihre Funktion hin überprüft werden. Dabei

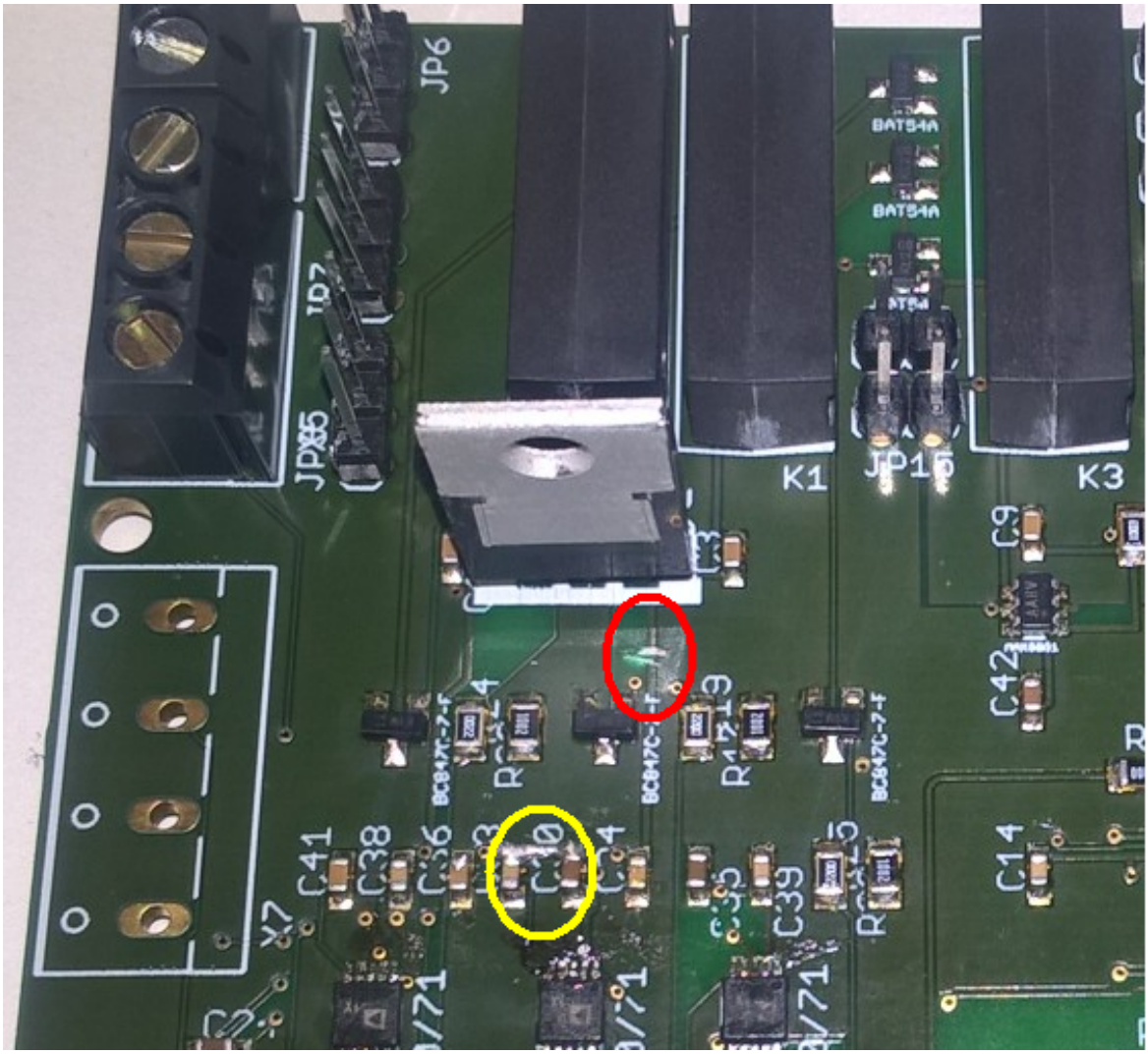

Abbildung 5.1.: Die Potentiometer-IC [4] haben eine ESD-Schutzdiode zwischen dem Versorgungspin des digitalen Interfaces und dem Lastkreis. Um ein Durchbrechen dieser zu vermeiden, musste die Spannungsversorgung der Potis überarbeitet werden. Das Bild zeigt die Stelle an der die Leiterbahn, welche den Lastkreis des Potentiometers mit 5V versorgt, durchtrennt wurde (roter Kreis). Für die Versorgung wurde stattdessen eine Brücke zur 3,3V-Spannungsversorgung eingelötet (gelber Kreis).

erwies es sich als sehr hilfreich, dass diese über Jumper miteinander verbunden werden müssen, um die Platine in ihren endgültigen Betriebszustand zu versetzen. Denn so ist es wesentlich leichter, Fehler bzw. Fehlinterpretationen zu separieren. Die Ansprache der Bauteile über den Bus erfolgte mit Programmen, die nur die für das jeweilige Bauteil benötigten Funktionen beinhalteten. Diese Funktionen wurden später identisch in die Steuersoftware übernommen.

Als erstes wurden die ADC überprüft, indem eine Spannung durch eine externes Netzteil angelegt wurde, was auch sofort ordnungsgemäß funktionierte. Danach erfolgte die Prüfung der Instrumentenverstärker. Auch hier bot es sich an, eine externe Spannung anzulegen und die Ausgangsspannung zu messen. Der Verstärker für die Spannungsmessung verhielt sich sofort wie erwartet. Der Verstärker für die Strommessung jedoch nicht. Die Spannung sollte um den Faktor zehn verstärkt werden. Das geschah auch, bis die Ausgangsspannung einen Betrag von 1,25V überstieg. Von da an verhielt sich die Verstärkung proportional zur Eingangsspannung. Da der andere Instrumentenverstärker problemlos funktionierte und der einzige Unterschied in der Beschaltung des Offset-Pin durch die Referenzspannung war, lag es nahe, hier nach dem Fehler zu suchen. Ein Blick in die Datenblätter von Instrumentenverstärker [7] und der 1,25V-Spannungsversorgung [27] sorgten schnell für Klarheit. Bei der Entwicklung der Platine war davon ausgegangen worden, dass sich der Spannungsregler als Energiequelle niederohmig verhält. Das ist jedoch nicht der Fall. Er ist zwar in der Lage, rückwärtige Ströme aufzunehmen, lässt dabei jedoch eine relevante Spannung an sich abfallen. Wenn nun der Instrumentenverstärker über seine Referenzspannung hinaus aussteuern möchte, so muss dieser einen Strom über seinen Referenzpin in die Schaltung speisen. Deshalb wird in seinem Datenblatt auch eine niederohmige Referenzspannungsquelle gefordert (z.B. Operationsverstärker). Da der Referenzspannungsregler dies jedoch wider erwarten nicht leistet, muss die Referenzspannung über ein externes Netzteil versorgt werden, so dass dieses den Strom aufnehmen kann.

Der DAC sorgte anfangs für Probleme, denn er ließ sich nicht ansteuern. Um das Problem zu identifizieren, wurde der Testcode „dacAnalyzer.c“ im Anhang A.3.9 auf Seite 120 verwendet. Dieser probiert sämtliche Kombinationen der ersten fünf und letzten sechs Bit mit allen möglichen Kombinationen von Clock-Idle und Übernahmeflanken durch. Es gelang jedoch nicht den Fehler zu rekonstruieren und einen Rückschluss zu ziehen, warum der DAC mittlerweile ordnungsgemäß funktioniert. Es wird vermutet, dass dieser zuweilen einfach „einfriert“, soll heißen, die Änderungen am Eingangsregister werden nicht umgesetzt. Das Problem tauchte, nachdem dieser plötzlich anfing, auf die Ansprache zu reagieren, noch ein einziges mal auf. Da das noch in der Programmierphase war, kam der Verdacht auf, dass es einen Buskonflikt gegeben haben könnte. Die SPI-Kommunikation muss seitens der Software aktiviert und deaktiviert werden. Das konnte, aufgrund von auftretenden Bugs, in dem Entwicklungsstadium nicht immer gewährleistet werden. Reproduktionsversu-

che scheiterten jedoch. Es empfiehlt sich daher die Messung beim Start für einen Moment im Auge zu behalten, um sich davon zu überzeugen, dass alles ordnungsgemäß funktioniert.

Bei den Potentiometern [4] sah es auch erst so aus, als würden diese einfach nicht reagieren. Die Ausgangsspannung am Schleifer änderte sich nicht. Hier konnte das Problem jedoch gefunden werden. Bei den ersten Testversuchen wurde der Schleifer-Ausgang nicht belastet. Das Potentiometer lässt intern, nicht wie zuerst gedacht, keinen Strom über seinen Widerstand fließen. Deshalb kommt es auch zu keinem Spannungsabfall, wenn es keine externe Last gibt. Dementsprechend ist die Ausgangsspannung am Schleifer ohne Last immer gleich der Versorgungsspannung des Lastkreises.

Nachdem alle Komponenten getestet worden sind, wurden diese mit den entsprechenden Jumpern verbunden. Vor dem Anschluss einer Zelle erfolgte noch eine Test mit einer externen Spannungsquelle und einem Lastwiderstand. Damit konnte der Strom besser variiert werden, um die Funktionstüchtigkeit auch über einen größeren Laststrombereich testen zu können. Nachdem auch das erfolgreich verlief, wurde die erste Zellzyklierung gestartet.

5.2. Durchführung von Modul- und Softwaretest

Nachdem nun die Hardware untersucht und in Betrieb genommen wurde, soll die Software und damit das Gesamtsystem auf die Performance und Genauigkeit hin untersucht werden.

5.2.1. Test der Performance

Bei den bisherigen Tests waren immer Lücken in der Aufzeichnung der Bilder zu beobachten. Zwar waren diese, nachdem der PcDuino aussortiert wurde, nicht mehr so gravierend, dennoch fehlten ab und zu drei bis sechs Bilder in den Aufzeichnungen. Bild 5.2 auf Seite 68 zeigt ein Beispiel für eine lückenhafte Aufzeichnung.
Der Raspberry-Pi war jedoch nie zu mehr als 50% ausgelastet, so kam der Verdacht auf, dass die Kamera für die Aussetzer verantwortlich sein könnte. Als die Platine fertiggestellt war und die Zyklierung der Zelle getestet werden sollte, ergab sich die Möglichkeit diese These zu überprüfen. Die bisher verwendet Mikroskopkamera von Conrad [2] mit 2 Mio.Pixeln wurde bei dieser Gelegenheit gegen die höher auflösende Version [2] mit 9 Mio.Pixeln ausgetauscht. Die Auswertung der ersten Zyklierung bestätigte den Verdacht. Denn mit der anderen Kamera fehlte kein einziges Bild. Das Bild 5.3 auf Seite 69 zeigt die Auswertung der lückenlosen Aufzeichnung.

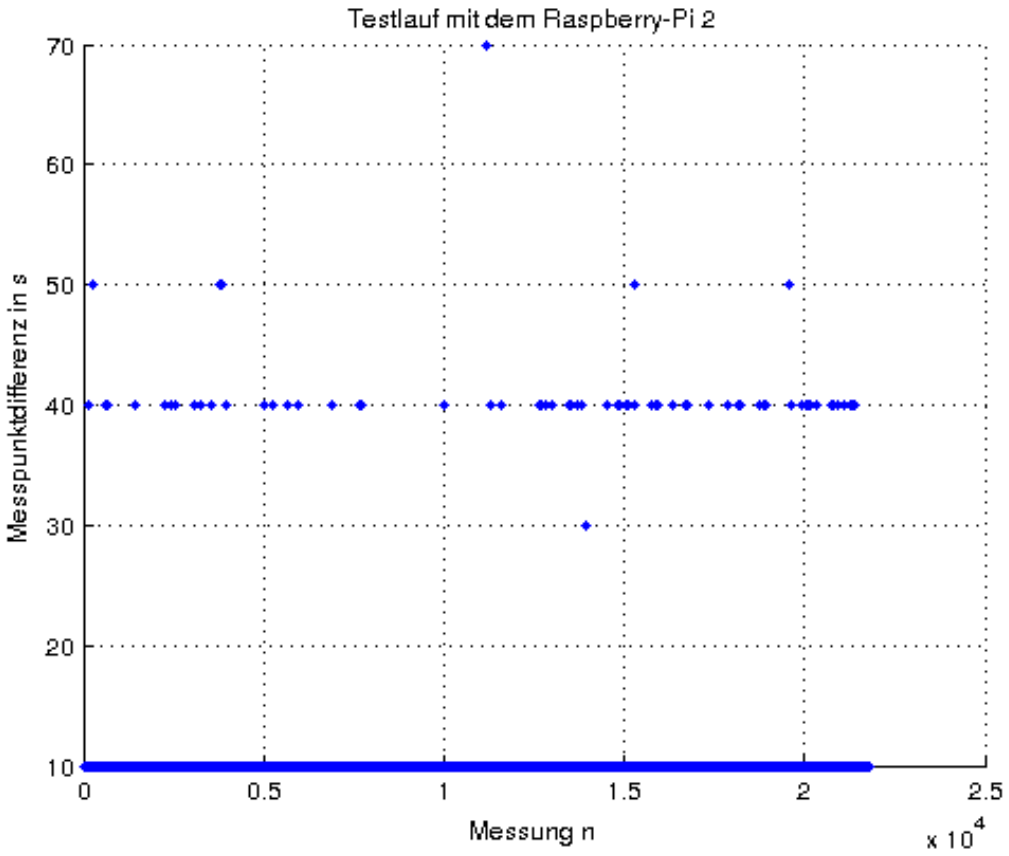

Abbildung 5.2.: Das Bild zeigt den lückenhaften Verlauf der Abstände zwischen den Zeitstempeln der, mit der ersten Kamera ([2] mit 2 Mio.Pixeln) aufgenommenen, Bilder. Es ist zu erkennen, dass einige Bilder einen Abstand von bis zu 70 Sekunden haben. Bei einer geplanten Aufnahmerate von einem Bild alle zehn Sekunden, fehlen an einigen Stellen bis zu sechs Bilder.

Die Systemlast während einer Zyklierung mit einem parallel laufenden Prozess des „DMM4020-Reader“ liegt zwischen 30% und 40%.

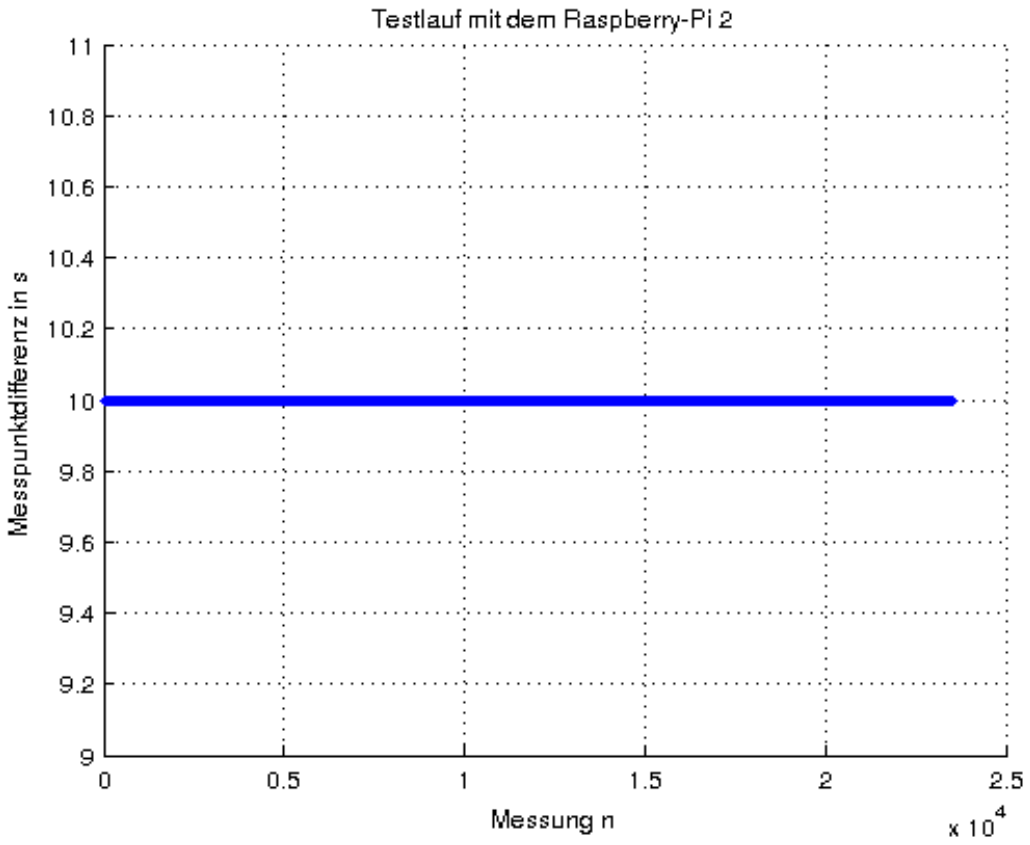

Abbildung 5.3.: Das Bild zeigt den lückenlosen Verlauf der Abstände zwischen den Zeitstempeln der, mit der höher auflösenden Kamera ([2] mit 9 Mio.Pixeln) aufgenommenen, Bilder. Es ist zu erkennen, dass alle Zeitstempel genau zehn Sekunden auseinander liegen. Es fehlt demnach kein einziges Bild.

5.2.2. Test der Genauigkeit mit $20\mu A$ Lade-/Entladestrom

Um die Genauigkeit des Systems zu untersuchen, bot es sich an, eine Zellzyklierung durchzuführen. Dazu wurde zunächst eine ältere Zelle verwendet, deren Verlust durch einen Fehlerfall nicht so tragisch gewesen wäre.
Im Laufe der Softwareentwicklung waren bereits einige Erfahrungen, bezüglich einiger Konfigurationsmöglichkeiten, gesammelt worden. Diese wurde in der Headerdatei „platine.h“ leicht zugänglich gemacht und müssen nur noch eingestellt werden. Die Tabelle 5.1 zeigt die Einstellungen für die erste Zyklierung mit einem Lade- und Entladestrom von $20\mu A$.
Die ersten beiden Makros IHYSTOBEN und IHYSTUNTEN sorgen für eine Hysterese um den Sollwert. Auf diese Weise wird die Messung etwas ruhiger, da weniger nach geregelt werden muss. Das Makro WINDOW steht für den Betrag der maximalen Abweichung, den ein neuer Strommesswert von dem letzten Messwert, der beide Mittelwertbildungen durchlaufen hat, abweichen darf. Das Makro ARRAYLENGTH repräsentiert die Länge des Arrays für die zweite Mittelwertbildung, während COLLECT für die Anzahl der Elemente der ersten Mittelwertbildung steht. Mit MAXDIFF wird die maximale Schrittweite einer DAC-Registeränderung referenziert, denn es erfolgt noch eine Skalierung proportional zu Regeldifferenz.

Makro	Wert	Funktion
IHYSTOBEN	2.0	Obere Hysterese-Grenze [μA]
IHYSTUNTEN	2.0	Untere Hysterese-Grenze [μA]
WINDOW	15.0	Zulässige Abweichung eines neuen Stromwertes [μA]
ARRAYLENGTH	100	Länge des Stromarrays zur Mittelwertbildung
MAXDIFF	0.06	Referenz für die Bestimmung der max. Änderung des DAC [μA]
COLLECT	5	Anzahl der zu sammelnden Werte für die erste Mittelwertbildung

Tabelle 5.1.: Einstellungen der Zyklierung mit $20\mu A$ über die Headerdatei „platine.h“

Auswertung

Mit den Einstellungen aus Tabelle 5.1 wurde die Testzelle 2,5 Tage mit $20\mu A$ zwischen 2V und 4V zykliert. Danach wurden die Daten mit dem Matlab-Skript aus Anhang A.4.3 auf Seite 124 ausgewertet. Die Bilder 5.4 und 5.5 auf Seite 71 und 72 zeigen das Ergebnis dieser Auswertung.
Grundsätzlich kann man sagen, dass das System wie gedacht funktioniert. Bild 5.4 zeigt, dass sie Schwellwertüberwachung einwandfrei arbeitet (unteres Diagramm). Auch die Stromregelung funktioniert gut, auch wenn es einen kleinen Offset von ca. $3\mu A$ gibt. Das Bild 5.5 zeigt den Offset, am Beispiel eines Ladevorgangs, im Detail.

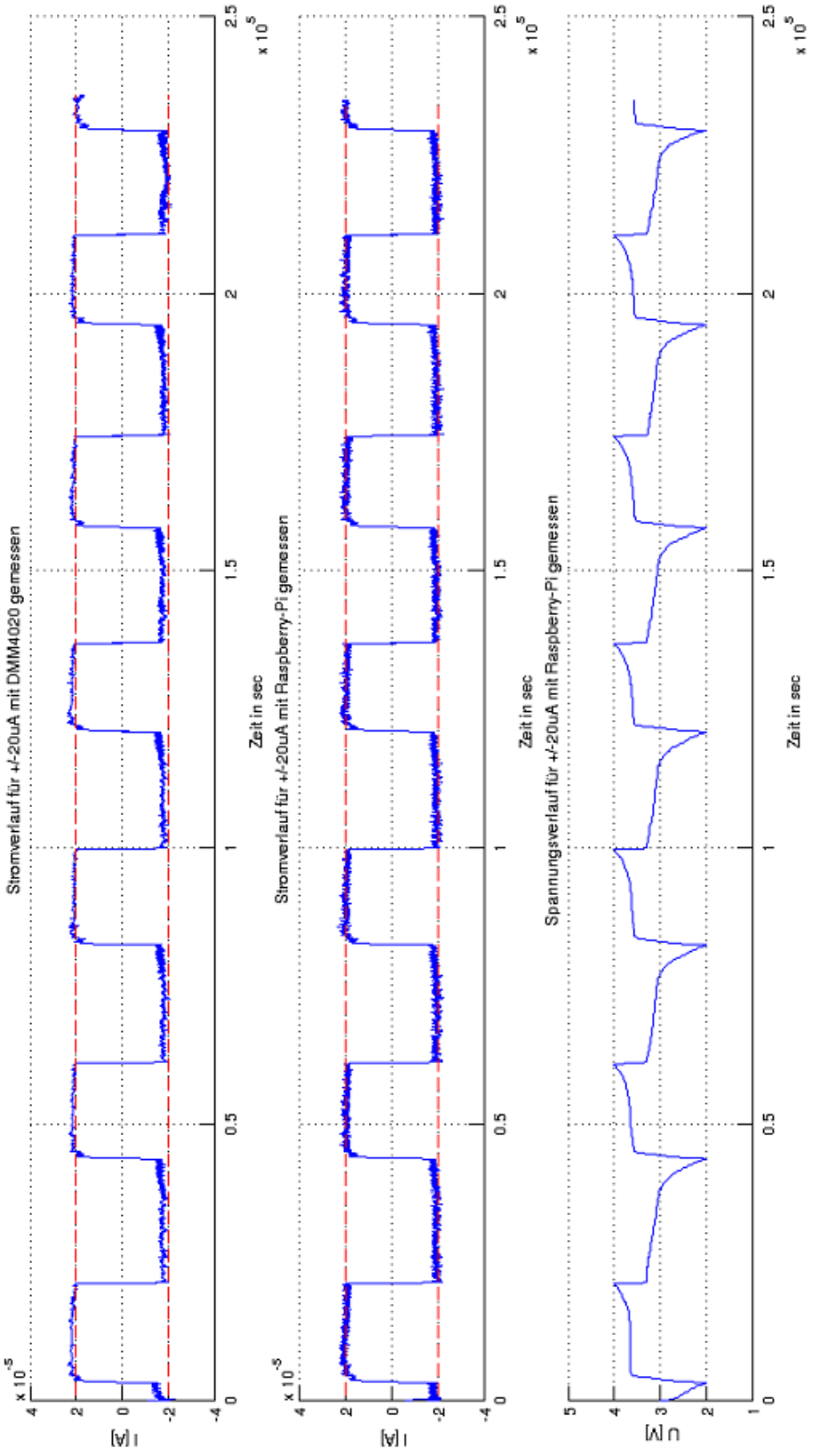

Abbildung 5.4.: Das Bild zeigt die Zyklierung einer Zelle mit einem Lade-/Entladestrom von $20\mu A$ über ca. 2,5 Tage. Während die Messung des Raspberrys auf den Sollwerten liegt, lässt das DMM4020 einen Strom-Offset erkennen.

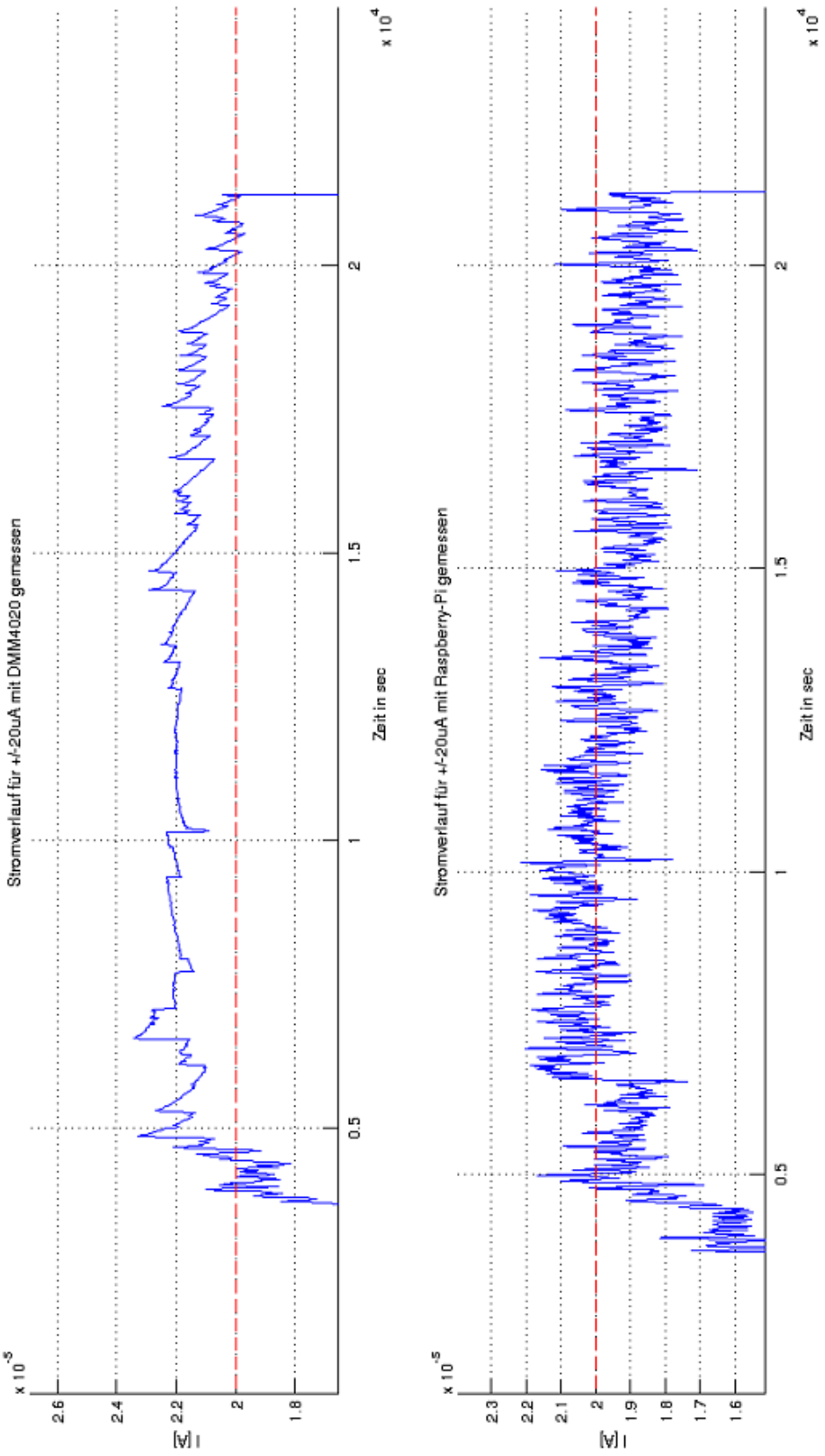

Abbildung 5.5.: Das Bild zeigt im oberen Diagramm den Strom-Offset der Zyklierung mit $\pm 20 \mu A$ im Detail. Dieser liegt bei ca. $3 \mu A$. Die Messung des Raspberry liegt, von einem Rauschen abgesehen, auf dem Sollwert (rot).

5.2.3. Test der Genauigkeit mit $100\mu A$ Lade-/Entladestrom

Im zweiten Test wurde der Lade-/Entladestrom auf $100\mu A$ erhöht. Die Einstellungen entsprechen denen aus dem ersten Versuch in Tabelle 5.1 auf Seite 70. Das Bild 5.6 auf Seite 75 zeigt die ersten beiden Zyklen des Versuchs. Man kann erkennen, dass der erste Ladevorgang mit ca. 13,5 Minuten, länger dauerte als der zweite mit ca. 10 Minuten. Diese Entwicklung der Zyklendauer setzte sich über den gesamten Versuch fort. Das Bild 5.7 auf Seite 76 zeigt die letzten Zyklen des Versuchs. Der letzte Zyklus dauerte nur noch ca. 3,5 Minuten und lässt sich fast nur noch durch den Graphen des Spannungsverlaufes abgrenzen. Denn die Regelung des Stromes schaffte es nicht mehr, den Strom auf seinen Sollwert zu bringen.
Der Ladevorgang lässt sich ganz gut an der positiven Steigung des Spannungsverlaufs erkennen, die Entladung entsprechend an der negativen Steigung. Schaut man sich den Stromverlauf eines Ladevorgangs an (Bild 5.7 Punkt 1 bis 2), kann man erkennen, dass dieser zunächst ansteigt. Der Anstieg ist sehr langsam und er erreicht nicht nur nicht den Sollwert, sondern fängt noch vor dem Ende des Ladevorgangs an zu sinken (Bild 5.7 Punkt 3). Schaut man sich den Stromverlauf an einer Stelle an, wo die obere Schwellspannung von 4V erreicht wird, erkennt man den fast nahtlosen Übergang von Lade- und Entladestrom, inmitten eines Stromverlaufs mit negativer Steigung (Bild 5.7 Punkt 2).

Auswertung

Die Beobachtung lässt sich mit der fortgeschrittenen Lebens- bzw. Zykluszeit der Zelle und der Geschwindigkeitsbegrenzung der DAC-Registeränderung erklären. Die Zelle hat mittlerweile einiges ihrer Kapazität eingebüßt und ihr Innenwiderstand dürfte sich erhöht haben. Das bedeutet, dass die Zelle, wenn diese geladen wird, ihren Ladungszustand schneller ändert, als zu der Zeit als diese noch neu war. Das lässt sich besonders gut an dem Unterschied der ersten Zykluszeit und der zweiten im Bild 5.6 auf Seite 75 erkennen. Die Zelle wird dort aus einer längeren Ruhezeit heraus gerissen, welche eine, wenn auch nur kurzlebige, Regeneration bedeutet. Sie schafft es jedoch nicht mehr dauerhaft, die selbe Energiemenge pro Spannungsanstieg auf zu nehmen. Gleichzeitig lässt ein höherer Innenwiderstand eine höhere stromabhängige Spannung an sich abfallen. Aus dem Zusammenspiel dieser Umstände ergibt sich, dass die Zelle eine so schnelle Spannungsänderung erreicht, dass die Treiberspannung es nicht mehr schafft, der Zellspannung schnell genug voraus zu eilen (Bild 5.7 auf Seite 76). Das führt dazu, dass diese den, für den Sollwert nötigen, Spannungsabfall über dem Referenzwiderstand der Strommessung nicht erzeugen kann. Der Einbruch des Stromes noch vor dem Ende des Ladevorgangs, lässt sich darauf zurückführen, dass die Zelle mit Steigender Zellspannung immer weniger Energie pro Spannungshub aufnehmen kann. Das lässt sich auch im Versuch mit einem Ladestrom von $20\mu A$ erkennen. Das Bild

5.4 auf Seite 71 zeigt deutlich, dass, bei funktionierender Stromregelung, die Spannungsanstiegsgeschwindigkeit am Ende eines Ladungsvorganges zunimmt. Diese Zunahme der Spannungsänderungsgeschwindigkeit der Zelle, sorgt dafür das die Zellspannung die Ausgangsspannung des Treibers immer weiter einholt und der Spannungsabfall über dem Messwiderstand kleiner wird. Die Folge ist ein sinkender Strom.

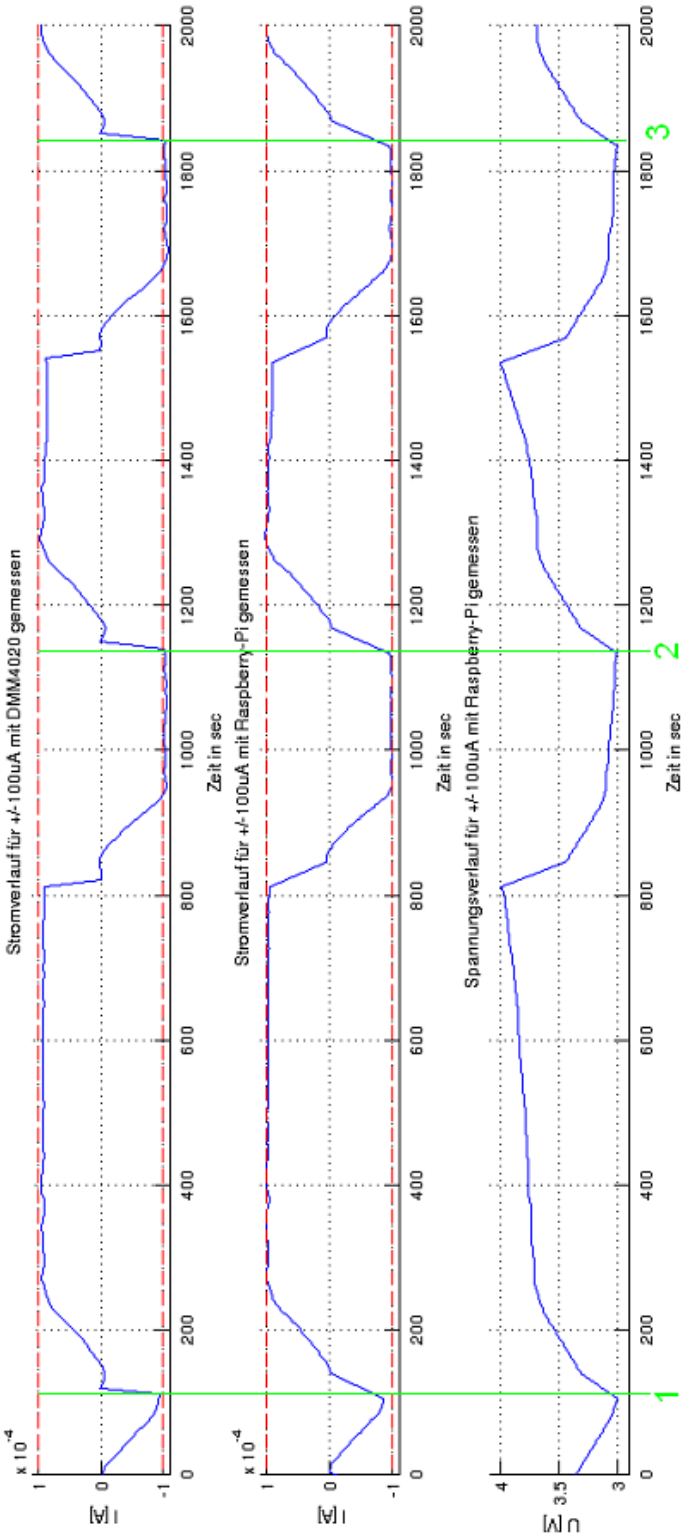

Abbildung 5.6.: Das Bild zeigt den Beginn einer Zyklierung mit $\pm 100 \mu A$. Der Intervall „Punkt 1“ bis „Punkt 2“ stellt den ersten Lade-/Entladezyklus dar, der Intervall „Punkt 2“ bis „Punkt 3“ den zweiten. Der große Unterschied der Dauer zwischen erstem und zweiten Zyklus, zeigt den altersbedingt schnellen Rückgang der Zellkapazität.

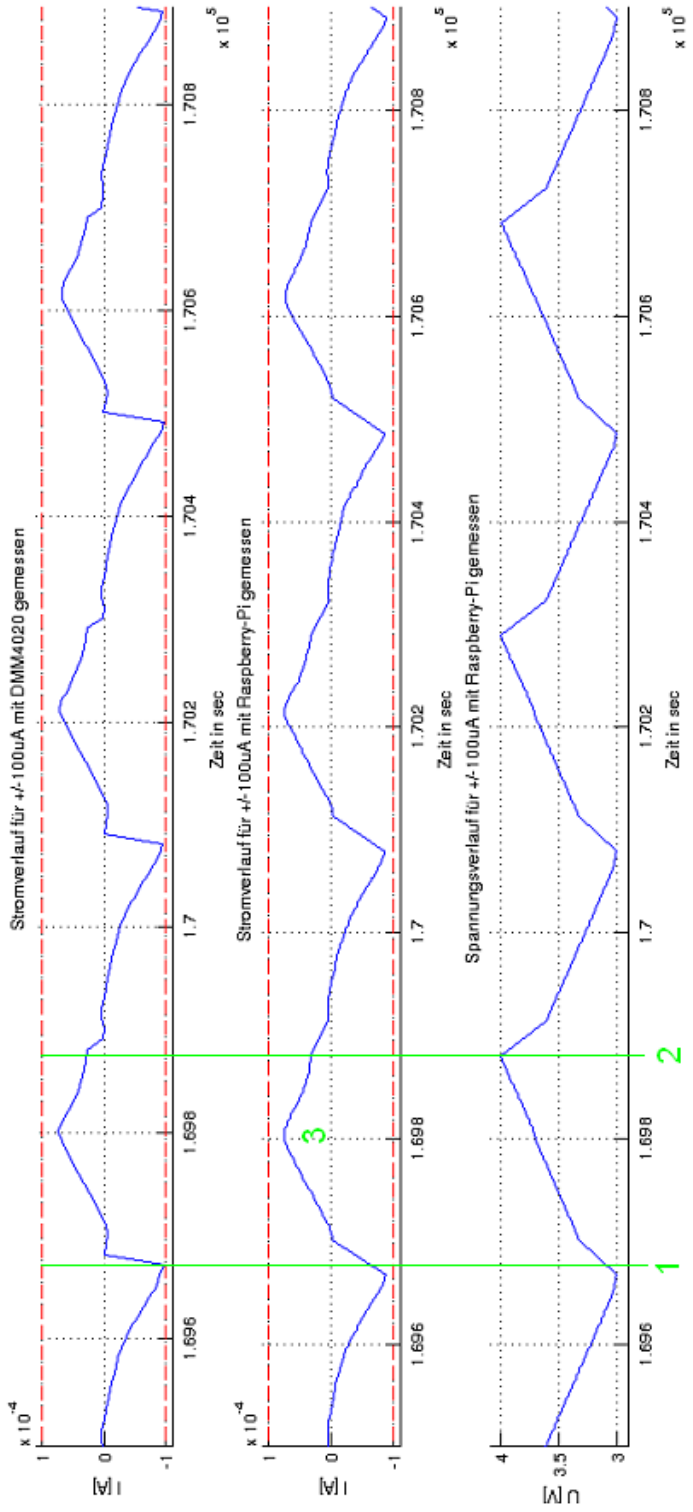

Abbildung 5.7.: Das Bild zeigt das Ende der Zyklierung mit $\pm 100 \mu A$. Man kann erkennen das die Regelung es nicht mehr schafft den Sollwert zu erreichen. „Punkt 1“ stellt den Beginn eines Ladevorgangs dar, „Punkt 2“ das Ende. An „Punkt 3“ fängt der Strom an kleiner zu werden, anstatt wie vorgesehen an „Punkt 2“.

5.2.4. Test der Genauigkeit mit $100\mu A$ Lade-/Entladestrom bei beschleunigter Regelung

Da die Stromregelung es im vorangegangenen Versuch (Kapitel 5.2.3) nicht geschafft hat, den Sollwert des Stromes zu erreichen, wurden die Einstellungen in der Headerdatei „platine.h“ angepasst. Tabelle 5.2 zeigt die Einstellungen, in denen die Referenz für die DAC-Registeränderung auf das zehnfache erhöht wurde.

Makro	Wert	Funktion
IHYSTOBEN	2.0	Obere Hysterese-Grenze [μA]
IHYSTUNTEN	2.0	Untere Hysterese-Grenze [μA]
WINDOW	15.0	Zulässige Abweichung eines neuen Stromwertes [μA]
ARRAYLENGTH	100	Länge des Stromarrays zur Mittelwertbildung
MAXDIFF	0.6	Referenz für die Bestimmung der max. Änderung des DAC [μA]
COLLECT	5	Anzahl der zu sammelnden Werte für die erste Mittelwertbildung

Tabelle 5.2.: Einstellungen der Zyklierung mit $100\mu A$ und beschleunigtem Regler

Auswertung

Das Bild 5.8 auf Seite 78 zeigt das Ergebnis der Zyklierung mit $100\mu A$ und beschleunigtem Regler. Die Zyklierung beginnt mit einer Entladung, deren Spannungsverlauf sich noch allmählich der unteren Schwellspannung annähert. Jedoch schon nach dem ersten Wechsel von Entladung zu Ladung hat die Zelle altersbedingt so viel Kapazität eingebüßt, dass beim Erreichen des Sollstromes die obere Schwellspannung bereits erreicht ist. Der entscheidende Unterschied zur Messung zuvor in Kapitel 5.2.3 ist jedoch, dass der Vorzeichenwechsel der Stromsteigung zum selben Zeitpunkt stattfindet wie bei dem Spannungsverlauf. Das bedeutet, dass die Steuerung der Zyklierung ihrer Aufgabe dieses mal gewachsen ist und ihr Spannungsanstieg mit dem der Zelle mithalten kann. Dass die Zelle bei $100\mu A$ bereits eine solch hohe Spannung an sich abfallen lässt, kann von dem Zykliersystem nicht beeinflusst werden.
Eine weitere Auffälligkeit ist das Auftreten von Schwingungen. Diese waren bei dem langsameren Regler nicht aufgetreten.

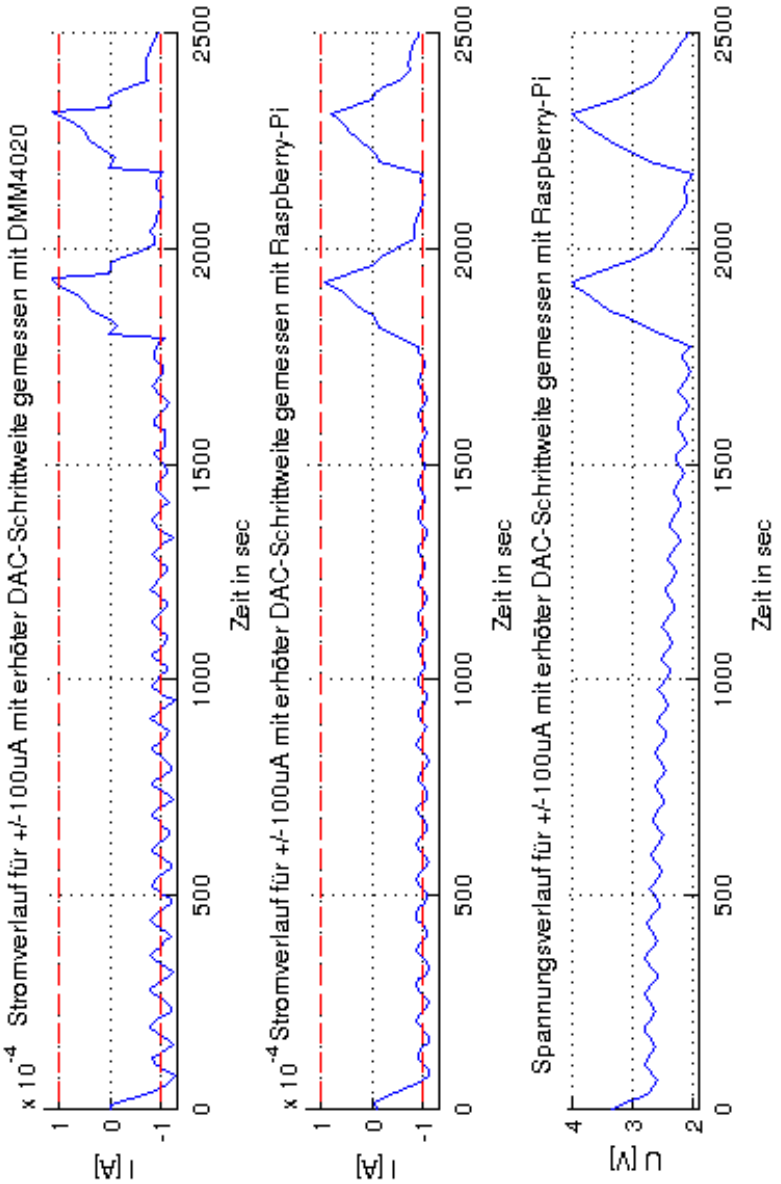

Abbildung 5.8.: Das Bild zeigt die Zyklierung mit $\pm 100 \mu A$ und beschleunigtem Regler. Die Zelle, welche zuvor eine Ruhephase hatte und somit „erholt“ ist, hat zunächst einen langsamen, wenn auch leicht schwingenden Verlauf. Fällt jedoch nach dem initialen Entladevorgang in einen steilen Spannungsverlauf, welcher dazu führt, dass ein Wechsel zwischen Ladung und Entladung erfolgt, bevor es zu einer echten Konstantstromladung kommen kann. Der zeitgleiche Wechsel des Vorzeichens von Strom- und Spannungssteigung ist ein Indikator dafür, dass die Regelung mit der Spannungsentwicklung der Zelle mithalten kann.

5.2.5. Test der Genauigkeit mit $20\mu A$ Lade-/Entladestrom bei beschleunigter Regelung

Nachdem im Kapitel 5.2.4 gezeigt wurde, dass der beschleunigte Regler die schnelle Änderung der Zellspannung bewältigen kann, soll nun untersucht werden, wie sich dieser bei einem Sollwert von $20\mu A$ verhält. Die Einstellungen sind dieselben wie im Kapitel 5.2.4 und können in Tabelle 5.2 auf Seite 77 angeschaut werden.

Auswertung

Das Bild 5.9 auf Seite 80 zeigt den Verlauf der Zyklierung mit $20\mu A$ und beschleunigtem Regler. Man kann erkennen, dass die Regelung funktioniert. Nach einem Wechsel zwischen Ladung und Entladung, kommt es jedoch zu ausgeprägten Einschwingvorgängen. Schaut man sich den Spannungsverlauf an, kann man erkennen, dass sich ein Lade-/Entladevorgang aus vielen einzelnen Zyklen zusammensetzt, die in Summe die Entwicklung des eigentlichen Lade-/Entladevorgangs ausmachen. Das ist so nicht gewollt. Bild 5.4 auf Seite 71 zeigt, dass der langsamere Regler kontinuierliche Lade-/Entladevorgänge erzeugt.

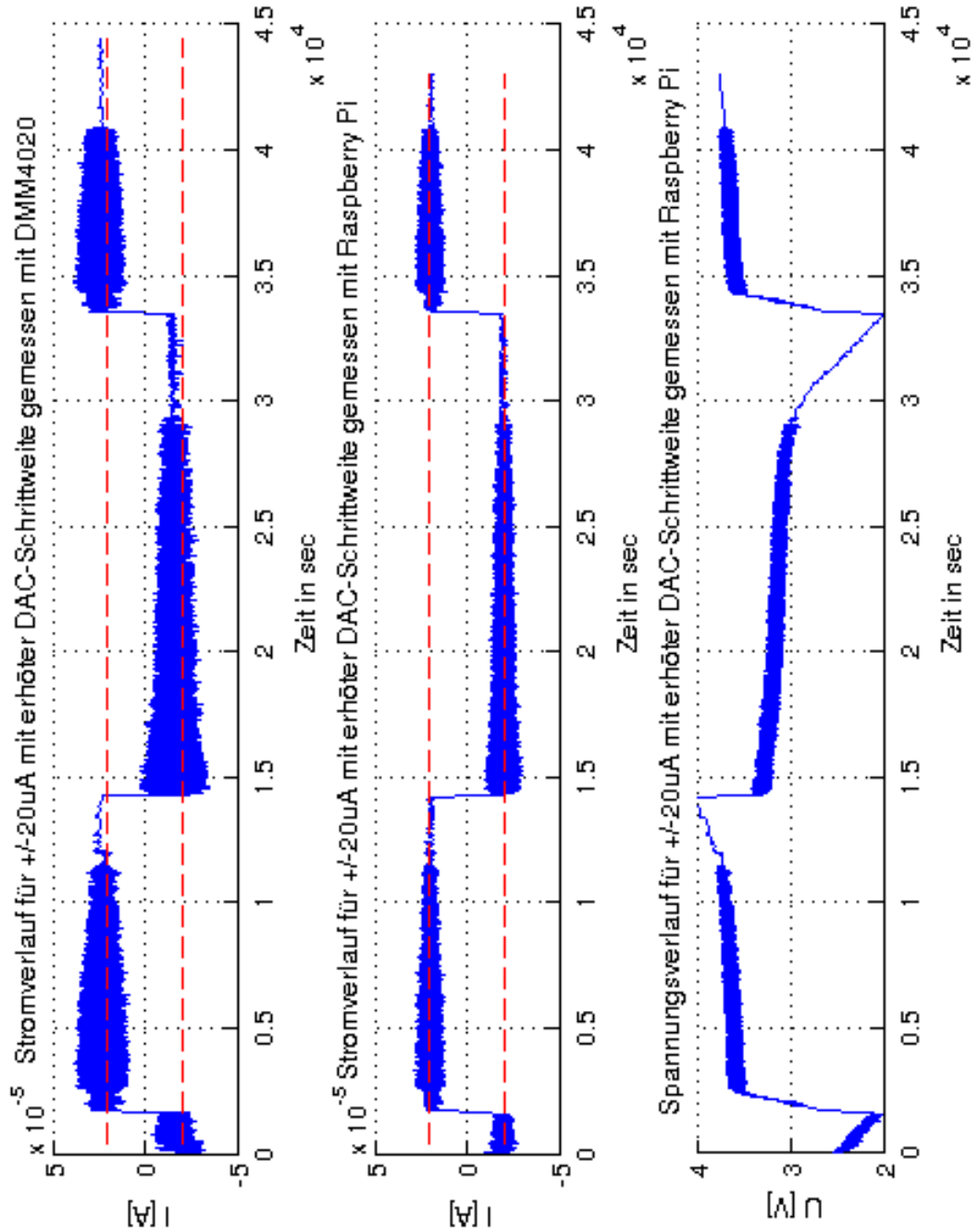

Abbildung 5.9.: Das Bild zeigt die Zyklierung mit $\pm 20 \mu A$ und beschleunigtem Regler. Der Verlauf der Zyklierung schwingt stark und die Lade-/Entladevorgänge setzen sich aus kleineren Lade-/Entladevorgängen zusammen.

5.3. Auswertung von Bilddaten

An dieser Stelle soll noch auf die Auswertung der Bilddaten eingegangen werden. Diese war Thema der Abschlussarbeiten Grießbach [14] und Palliyaguruge [28]. Bei dieser Thesis wurde lediglich dafür gesorgt, dass die Daten mit den bestehenden Tools kompatibel sind. Die Bilder 5.10 und 5.11 auf den Seiten 82 und 83 zeigen die Ausgabe der Auswertung, durch die Matlab-Skripte von Palliyaguruge [28]. Das Bild 5.11 auf Seite 83 zeigt die Intensität der Rot-, Grün- und Blauwerte im Zusammenhang mit dem Ladezustand der Zelle. Zum Zeitpunkt der Abgabe dieser Thesis stand leider nur ein defekter Datensatz zur Verfügung. Das bedeutet, dass Datenteile fehlen. Die fehlenden Datenbereiche sind im Bild 5.11 gelb hinterlegt. Die Graphen in diesen Bereichen sind nicht aussagekräftig, da die Daten entfernt werden mussten, weil die dazugehörigen Bilder überbelichtet waren. Auf der beiliegenden DVD befindet sich das Video zu den Daten in einer Originalfassung und einer Fassung, in der überbelichtet Bilder herausgenommen wurden.

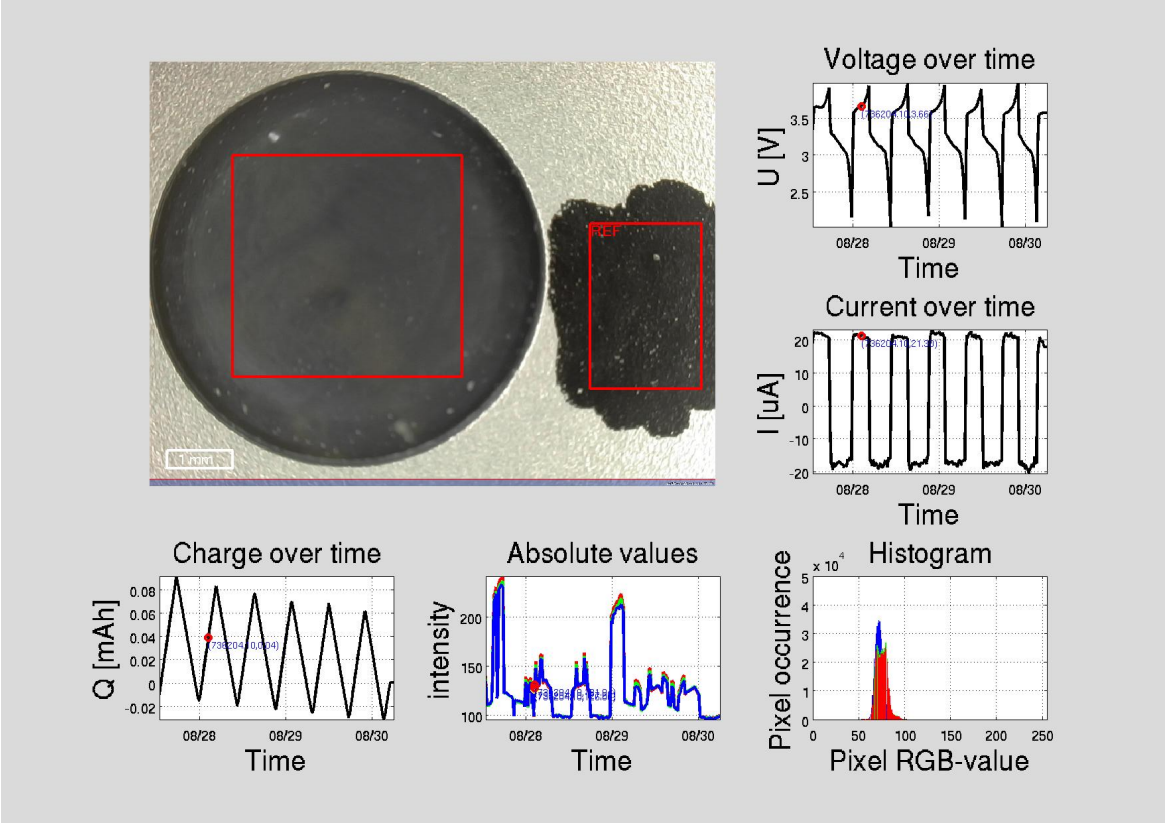

Abbildung 5.10.: Das Bild zeigt die Auswertung der Bilddaten durch Matlab [28]. Links oben ist das Bild der Zelle mit der Referenzfläche (rechts) und dem ebenfalls markierten Bereich der Zelle (links), welcher optisch ausgewertet wird. Daneben sind Strom und Spannungsverlauf zu sehen. Am unteren Rand sind der Ladezustand und die Ausgabe der RGB-Werte zu sehen. Die Ausgabe der RGB-Werte ist diesem Beispiel, aufgrund von Überbelichtung einiger Bilder, nicht aussagekräftig. Bild 5.11 auf Seite 83 zeigt die Intensität der RGB-Werte. Dort sind auch die fehlerhaften Bereiche markiert.

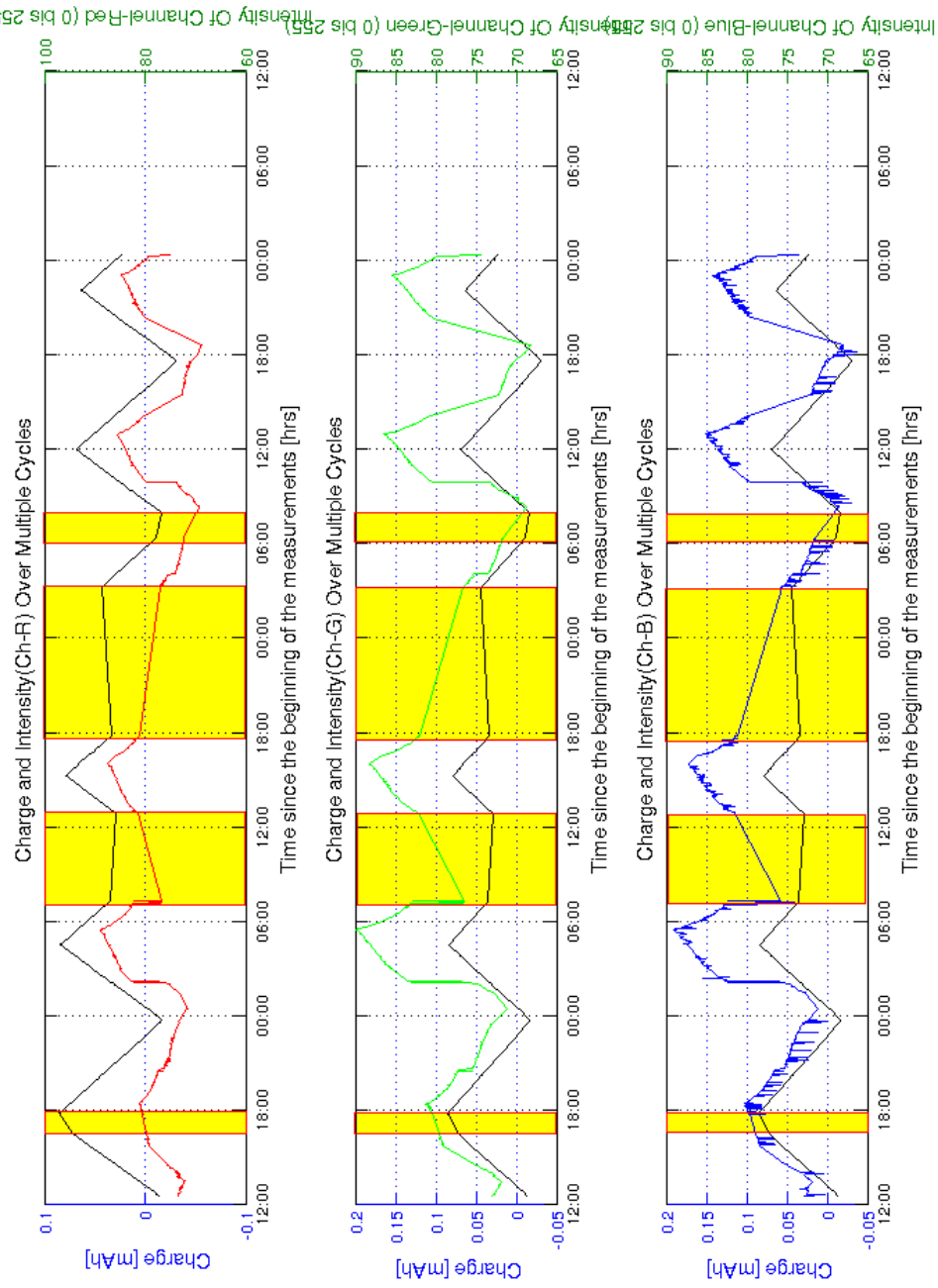

Abbildung 5.11.: Das Bild zeigt die Intensität der RGB-Werte im Zusammenhang mit dem Ladezustand der Zelle [28]. Die gelb hinterlegten Bereiche sind nicht aussagekräftig, weil die Daten hier entfernt werden mussten. Grund dafür sind überbelichtete Bilder. Ein geeigneterer Datensatz stand zum Abgabezeitpunkt leider nicht zur Verfügung. Das Bild 5.10 auf Seite 82 zeigt die Ausgabe im Gesamtüberblick.

6. Fazit

In diesem abschließenden Kapitel soll der Verlauf dieser Abschlussarbeit und ihre Ergebnisse zusammengefasst werden. Dabei wird darauf eingegangen, inwieweit die in Kapitel 2.7.2 auf Seite 18 definierten Anforderungen erfüllt werden. Anschließend gibt es einen Überblick darüber, an welchen Punkten in Zukunft angesetzt werden kann, um den Messaufbau zu verbessern.

6.1. Zusammenfassung

Aufgabe war es, ein System zu entwickeln, welches in der Lage ist, eine Lithium-Ionen-Zelle mit einem einstellbaren Konstantstrom zu zyklieren. Der Strom sollte bis zu einem Milliampere groß und in Mikroampere-Schritten einstellbar sein. Die Zyklierung sollte sich an der Zellspannung orientieren. Zusätzlich musste die Aufnahme der Bilder von der Elektrode erfolgen. Dazu brauchte es auch eine Beleuchtung mit einstellbarem Rot-, Grün-, und Blauanteil. Für belastbare Ergebnisse musste auch die Temperatur der Zellumgebung aufgezeichnet werden. Die aus vorangegangenen Arbeiten bereits bestehenden Tools zur Strom und Spannungsmessung und die Matlabskripte zur Auswertung der Mess- und Bilddaten, sollten weiterhin verwendet werden können. Deshalb musste darauf geachtet werden, dass die Speichermethoden und Namenssyntax kompatibel sind.

Als erstes galt es, eine geeignete Plattform zu finden, auf deren Basis das Zykliersystem aufgebaut werden konnte. Dazu wurden mehrere Möglichkeiten in Betracht gezogen. Letztendlich fiel die Entscheidung auf den „Raspberry-Pi 2“. Auf diesem kommt das Linux-Betriebssystem „Raspbian“ zum Einsatz. Raspbian verwaltet, mit Hilfe eines Shell-Skriptes, alle für die Messung nötigen Prozesse. Dazu gehört die Bildverarbeitung, die Messung von Strom und Spannung mit Hilfe von Präzisionsmessgeräten und der Prozess zur Steuerung der Zyklierung. Letzterer ermöglicht die Interaktion des Raspberry-Pi mit einer Zusatzplatine, welche eigens zu diesem Zweck entwickelt wurde. Die Platine wurde mit allen nötigen Sensoren und Aktoren ausgestattet, die für die Bewältigung der Aufgabe nötig sind. Die logische Verknüpfung der Komponenten und die Auswertung der Daten erfolgt jedoch ausschließlich auf dem Raspberry-Pi.

Nachdem die Hardware gebaut und die Software geschrieben war, wurde die Funktion des Aufbaus getestet. Hierfür wurde eine Lithium-Ionen-Zelle zykliert und die daraus resultierenden Daten mit Matlab-Skripten, welche im Anhang A.4 ab Seite 123 angeschaut werden können, ausgewertet. Im Fokus des Interesses stand dabei die Genauigkeit der Stromregelung und die Zuverlässigkeit des Systems. Letztere wurde durch die Systemlast und die lückenlosen Bildaufzeichnung definiert.

6.2. Bewertung des Ergebnisses

Der „Raspberry-Pi 2“ erwies sich als sehr geeignete Plattform für das Zykliersystem. Mit der Hardware, die er mitbringt, ist er bestens gerüstet, um die Anforderungen zu erfüllen. Die dabei entscheidendsten Komponenten sind die CPU und die Schnittstellen. Die CPU ist der erste Vier-Kern-Prozessor, der bei einem Einplatinencomputer dieser Preisklasse zum Einsatz kommt. Vor dem Hintergrund, dass dieser erst kurz vor Beginn dieses Projektes auf den Markt kam und die andere CPU, die getestet wurde, überfordert war, kann das als „Wie bestellt!“ bezeichnet werden. Die GPIO-Schnittstelle wird von der Community auf eine Weise unterstützt, die man für andere Systeme nur schwer finden wird. Das gilt auch für das Betriebssystem Raspbian, welches extra auf die Hardware zugeschnitten wurde und, ebenfalls aufgrund der großen Community, als ausgiebig getestet bezeichnet werden kann. Da die Auslastung im Zyklierbetrieb keine 40% beträgt, ist das Anforderungsprofil sogar noch ausbaubar.

Die Erweiterungsplatine kann im vollen Umfang genutzt werden. Sie muss dazu allerdings mit drei, statt wie geplant zwei, Spannungen versorgt werden. Denn die Versorgung der Referenzspannung für den Instrumentenverstärker der Strommessung ist, wegen des hochohmigen Verhaltens des Spannungsreglers, ungeeignet. Ist diese Versorgung gewährleistet, kann mit der Platine ein Strom mit einer Toleranz von ca. fünf Mikroampere geregelt werden. Die Platine muss nach dem Zusammenbau noch „modifiziert“ werden, da die Spannungsversorgung der Potentiometer nicht wie geplant mit 5V erfolgen kann, sondern, wegen einer ESD-Schutzdiode, nur mit 3,3V. Das schränkt jedoch die Funktionalität nicht ein, da die LED-Platine, die von hier versorgt werden soll, angepasst werden konnte. Eine Kurzanleitung für die Modifikation der Erweiterungsplatine, befindet sich im Anhang A.6 auf Seite 128.
Ein Bauteil der Platine, der DAC, ist schwer einzuschätzen. Er hat während der Entwicklungsphase zeitweise nicht richtig funktioniert, die Ursache konnte nicht geklärt werden. Der DAC hat während der Tests reibungslos seine Aufgabe erfüllt, bleibt jedoch mit einem Rest Ungewissheit behaftet. Mehr Details zur Platine sind in Kapitel 5.1 nachzulesen.

Das Softwarepaket, welches die Hardwarekomponenten zu einem Zykliersystem zusammenschweißt, erfüllt seine Aufgaben zuverlässig. Für die Bildverarbeitung wurde mit „fswebcam“ ein Programm gewählt, welches alle Einstellmöglichkeiten der Kamera ansprechen

kann, die Bilder jedoch nicht live ausgibt. Auf diese Weise konnten Systemressourcen gespart werden. Das Programm „luvcview“ ergänzt dieses mit einer Direktausgabe der Bilder für den Einstellvorgang an der Kamera. Die Bilder waren zum Abgabezeitpunkt teilweise überbelichtet. Es ist jedoch nicht zu erwarten, dass dies ein dauerhaftes Problem bleibt, sondern kalibriert werden kann.
Der modulare Aufbau des Softwarepaketes wurde bereits aus der Arbeit Grießbach [14] übernommen und beibehalten, denn dieser macht das System aus- und umbaufähig.
Die größte Änderung im Softwarepaket ist die Steuersoftware für die Erweiterungsplatine. Diese wertet die Benutzereingaben zuverlässig aus und sorgt für deren Umsetzung. Dabei traten nie Fehler auf und fehlerhafte Benutzereingaben werden abgefangen. Das Shellskript ergänzt das Programm und vereinfacht die Handhabung. Das ist, in Anbetracht der Vielzahl von nötigen Angaben, sehr hilfreich. Die Schwellwertüberwachung, die nicht zuletzt die Zelle vor Beschädigung schützt, funktioniert ebenfalls einwandfrei. Das gilt auch für die Stromregelung. Diese kann Ströme u.U. auch über die geforderten $1mA$ hinaus regeln. Allerdings haben die Test gezeigt, dass die Strecke (Zelle) dabei einen entscheidenden Einfluss hat. Eine Kombination aus geringer Zellkapazität, hohem Innenwiderstand (z.B. alte Zelle) und hohem Ladestrom kann die Regelung an ihre Grenzen bringen (siehe Kapitel 5.2.3 und 5.2.4).

Die Möglichkeit, Zellen mit einem einstellbaren Konstantstrom zu zyklieren, stellt eine große Verbesserung dar. Das Zykliersystem erfüllt alle gestellten Anforderungen und auch die Genauigkeit liegt mit $\pm 5\mu A$ sehr nahe an den angestrebten $\pm 1\mu A$. Die nötige Änderung in der Versorgung der Potis und die vierpolige Spannungsversorgung sind zwar Schönheitsfehler, welche die Funktion jedoch nicht in Frage stellen. Die Entwicklung des Zykliersystems war ein Erfolg.

6.3. Ausblick

Das Zykliersystem konnte im Rahmen dieser Arbeit nur exemplarisch getestet werden. Hier sind weitere Tests nötig. Die Kalibrierung ist zwar den aktuellen Anforderungen gewachsen, kann jedoch bestimmt noch verfeinert werden.

Die vielversprechendste Weiterentwicklung bietet wohl die aus Zeitgründen verworfene Idee, die Erweiterungsplatine mit einen Mikrocontroller als Regler auszustatten. Dieser wäre in der Lage, eine Abtastfrequenz zu gewährleisten. Das eröffnet ganz neue Möglichkeiten, das Zykliersystem nach den Methoden der digitalen Regeltechnik und Signalverarbeitung zu untersuchen, was mit einer vom Scheduler abhängigen Abtastung kaum möglich ist.
Die bestehende Hardware könnte dabei direkt übernommen werden. Es besteht die Möglichkeit, die Erweiterungsplatine nicht an den Raspberry-Pi sondern an ein Evaluation-Board

wie zum Beispiel dem „TM4C1294“ der „Tiva C Reihe“ von „Texas Instruments“ anzuschließen. Der Raspberry-Pi könnte, ggf. mit einem separaten SPI-Bus, ebenfalls an das Evaluation-Board angeschlossen werden und so seine administrativen Aufgaben übernehmen. Die Steuersoftware könnte vollständig übernommen werden und müsste lediglich mit einer Funktion für Datenaustausch und Synchronisation mit dem Evaluation-Board erweitert werden. Das übrige Softwarepaket bliebe völlig unbeeinflusst.
Auf diese Weise kann sowohl ein digitaler Regler, als auch ein FIR- oder IIR-Filter ausgelegt werden. Damit würde sich das Problem der Phasenverschiebung schlagartig erübrigen, denn ein solcher Filter ist deutlich schneller als eine Mittelwertbildung über 100 Werte, wie es zur Zeit die Methode der Wahl ist. Für die Parametrierung des Reglers könnten Zelle und Hardware systemtheoretisch untersucht werden, um aus der Regelung das Optimum heraus zu holen.

Eine weitere Möglichkeit das System auszubauen wäre eine GUI, welche das Shell-Script ablösen könnte. Das würde die Bedienung deutlich intuitiver machen. So könnte es eine Oberfläche zur Erstellung eines Zyklierplans geben und eine Auswahl bestehender Zyklierpläne angeboten werden. Auch die Darstellung der aktuellen Soll- und Istwerte wäre so viel übersichtlicher als auf der Konsole.

Literaturverzeichnis

[1] ATMEL: ATxmega16A4U. – URL http://www.atmel.com/images/Atmel-8387-8-and16-bit-AVR-Microcontroller-XMEGA-A4U_Datasheet.pdf. – Zugriffsdatum: 18.09.2015

[2] CONRAD: Digital-Mikroskop-Kamera. – URL http://www.produktinfo.conrad.com/datenblaetter/175000-199999/191377-an-01-ml-DIGITAL_MIKROSKOPKAMERA_USB_de_en_fr_nl.pdf. – Zugriffsdatum: 11.09.2015

[3] DEMBOWSKI, Klaus: Raspberry Pi Das technische Handbuch. Springer Vieweg Verlag, 2015. – ISBN 978-3-658-08711-1

[4] DEVICES, Analog: Datenblatt AD5270/71 Rev. F. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD5270_5271.pdf. – Zugriffsdatum: 15.07.2015

[5] DEVICES, Analog: Datenblatt AD5680 Rev. B. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD5680.pdf. – Zugriffsdatum: 14.07.2015

[6] DEVICES, Analog: Datenblatt AD7691 Rev. E. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD7691.pdf. – Zugriffsdatum: 14.07.2015

[7] DEVICES, Analog: Datenblatt AD8422 Rev. A. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD8422.pdf. – Zugriffsdatum: 14.07.2015

[8] ELEMENT14.COM: Raspberry Pi B+ GPIO 40 Pin Block Pinout. – URL http://www.element14.com/community/docs/DOC-68203/l/raspberry-pi-b-gpio-40-pin-block-pinout?sa=X&ved=0CBgQ9QEwAGoVChMI-ZPP76n7xgIVxpEsCh2pIQud. – Zugriffsdatum: 27.07.2015

[9] ELINUX.ORG: bcm2835 GPIO. – URL http://elinux.org/RPi_BCM2835_GPIOs. – Zugriffsdatum: 14.09.2015

[10] FAIRCHILD: Datenblatt LM7910CT. – URL https://www.fairchildsemi.com/datasheets/LM/LM7910.pdf. – Zugriffsdatum: 16.07.2015

[11] FOUNDATION, Raspberry P.: Raspberry Pi 2 Model B. – URL https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. – Zugriffsdatum: 22.07.2015

[12] FOUNDATION, Raspberry P.: Raspbian Download. – URL https://www.raspberrypi.org/downloads/. – Zugriffsdatum: 26.07.2015

[13] GNU.ORG: The GNU Library: Getopt. – URL http://www.gnu.org/software/libc/manual/html_node/Getopt.html. – Zugriffsdatum: 17.08.2015

[14] GRIESSBACH, Jan: Messaufbau mit Steuer- und Analysesoftware für die optische Zustandsbeobachtung von Lithiumbatterien. 2014

[15] HAMLIN: Datenblatt HE3321A0400 Rev. AB. – URL http://www.hamlin.com/specSheets/HE3300%20rev%20AB.pdf. – Zugriffsdatum: 17.07.2015

[16] HTTP://ELINUX.ORG: Beagleboard:BeagleBoneBlack. – URL http://elinux.org/Beagleboard:BeagleBoneBlack. – Zugriffsdatum: 22.07.2015

[17] INSTRUMENTS, Texas: Datenblatt LM2940X Rev. J. – URL http://www.ti.com/lit/ds/symlink/lm2940-n.pdf. – Zugriffsdatum: 16.07.2015

[18] INSTRUMENTS, Texas: Datenblatt LM74 Rev. K. – URL http://www.ti.com/lit/ds/symlink/lm74.pdf. – Zugriffsdatum: 16.08.2015

[19] INSTRUMENTS, Texas: Datenblatt OPA743. – URL http://www.ti.com/lit/ds/symlink/opa743.pdf. – Zugriffsdatum: 15.07.2015

[20] INSTRUMENTS, Texas: Datenblatt TLV2217 Rev. L. – URL http://www.ti.com/lit/ds/symlink/tlv2217.pdf. – Zugriffsdatum: 17.07.2015

[21] INSTRUMENTS, Texas: Datenblatt TPS79925 Rev. K. – URL http://www.ti.com/lit/ds/symlink/tps799.pdf. – Zugriffsdatum: 17.07.2015

[22] INTEGRATED, Maxim: Datenblatt MAX8881. – URL http://datasheets.maximintegrated.com/en/ds/MAX8880-MAX8881.pdf. – Zugriffsdatum: 17.07.2015

[23] LINKSPRITE: pcDuino3 Nano. – URL http://www.linksprite.com/?page_id=815. – Zugriffsdatum: 22.07.2015

[24] MANUALS ubuntu: fswebcam. – URL http://manpages.ubuntu.com/manpages/lucid/man1/fswebcam.1.html. – Zugriffsdatum: 14.09.2015

[25] MANUALS ubuntu: luvcview. – URL http://manpages.ubuntu.com/manpages/precise/man1/luvcview.1.html. – Zugriffsdatum: 14.09.2015

[26] MCCAULEY, Mike: bcm2835. – URL http://www.airspayce.com/mikem/bcm2835/. – Zugriffsdatum: 22.07.2015

[27] MICROELEKTRONIKS, ST: Datenblatt LD39015 Rev. 4. – URL http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00173477.pdf. – Zugriffsdatum: 17.07.2015

[28] PALLIYAGURUGE, Don Mithila M.: Image processing for investigation of effects in Lithium battery electrodes. 2015

[29] PEREGUDA, Sergej: Herr Pereguda entwickelte und baute die LED-Platine für die Mikroskopcamera.. – URL https://www.xing.com/profile/Sergej_Pereguda. – Zugriffsdatum: 09.09.2015

[30] POOL, PCB: Strombelastbarkeit. – URL http://www.pcb-pool.com/download/spezifikation/deu_cmso001_strombelastbarkeit.pdf. – Zugriffsdatum: 14.07.2015

[31] RASPBIAN.COM: Raspbian FAQ. – URL https://www.raspbian.org/RaspbianFAQ. – Zugriffsdatum: 22.07.2015

[32] ULRICH TIETZE, Eberhard G.: Halbleiterschaltungstechnik. Springer Vieweg Verlag, 2012. – ISBN 978-3-642-31025-6

[33] VISHAY: Datenblatt SMBJ3V3. – URL http://www.vishay.com/docs/88940/smbj3v3.pdf. – Zugriffsdatum: 17.07.2015

[34] WIKIPEDIA: Akkumulator. – URL http://de.wikipedia.org/wiki/Akkumulator. – Zugriffsdatum: 14.07.2015

[35] WIKIPEDIA: Energiespeicher. – URL http://de.wikipedia.org/wiki/Energiespeicher. – Zugriffsdatum: 14.07.2015

[36] WIKIPEDIA: Linux Mint. – URL https://de.wikipedia.org/wiki/Linux_Mint. – Zugriffsdatum: 22.07.2015

[37] WIKIPEDIA: Raspberry Pi. – URL https://de.wikipedia.org/wiki/Raspberry_Pi. – Zugriffsdatum: 22.07.2015

[38] WIKIPEDIA: Wasserstoffantrieb. – URL http://de.wikipedia.org/wiki/Wasserstoffantrieb. – Zugriffsdatum: 14.07.2015

A. Anhang

A.1. Errata

Nicht alles hat bei der Hardwareentwicklung sofort funktioniert. Hier eine Auflistung der gefundenen Fehler.

- Die Spannungsversorgung für den Lastkreis der Potentiometer (5V) musste aufgetrennt werden und eine Brücke zu 3,3V Versorgung eingelötet werden.
- Die Referenzspannung für den Instrumentenverstärker muss mit einer externen Spannungsquelle versorgt werden.

A.2. Zielsetzung

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

17. Juni 2015

Bachelorthesis Torsten Geist

Entwicklung eines embedded Einplatinensteuersystems für optische Batterieuntersuchungen

Motivation

Im Rahmen der Forschungsaktiviäten der Graduiertenschule „Key Technologies for Sustainable Energy Systems in Smart Grids“ soll der Zustand von Batterien durch neuartige Sensorik erfasst werden.
Für Lithiumbatterien ist die Verbesserung der Erkennbarkeit des Batteriezustandes wünschenswert. Die elektrisch erfassbaren Größen wie Klemmspannung, Batteriestrom und Temperatur sollen um eine optische Beobachtung der physikalisch/chemischen Prozesse im Elektrodenmaterial ergänzt werden. Hierzu ist neben dem geeigneten Batteriezellenaufbau eine Rechner-Plattform zur Bild- und Messwerterfassung sowie zur automatisierten Steuerung des Lade/Entladebetriebes erforderlich.

Aufgabe

Herr Torsten Geist erhält die Aufgabe, diese Plattform auf der Basis eine Einplatinen-Computers zu entwickeln. Die Aufgabe umfasst die Nutzung von Bilderfassungsoftware für eine angeschlossene Mikroskopkamera. Außerdem soll der Entwurf und die Implementierung von Aufzeichnungsoftware für elektrische Messwerte (Zellenspannung, Zellenstrom) sowie für die Temperatur erfolgen. Zur Bilderfassung sind Beleuchtungs-LED für die Mikroskopkamera zu steuern.
Für die Zelle soll eine Zyklier-Ablaufsteuerung entstehen. Diese besteht aus zu entwickelnder Software- und aus Hardwarekomponenten. Für die Hardware soll eine Platine entwickelt werden, die eine vom Einplatinen-Computer steuerbare Spannungsquelle, eine Spannungs- und eine Strommess-Schaltung umfasst. Ergänzend sollten auch Spannungs- und Strommessungen mit Präzisionsmessgeräten unterstützt werden. Die zu entwicklende Software soll den Bilderfassungs-, Zyklier- und Messablauf steuern. Dazu sind die Schnittstellen für die angeschlossen Geräte und die eigene Platine zu unterstützen. Die Zykliersteuerung soll durch ein Kommandoscript flexible Abläufe ermöglichen.
Durch Erprobungen und Messreihen sind die entwickelten Lösungen zu überprüfen und zu beurteilen.
Für die Abschlussarbeit sind die folgenden Arbeitspakete geplant:

1. Recherche
    - Einarbeitung in die Projektzielstellung
    - Einarbeitung in vorangegangene Bachelorarbeiten und vorhandene Hard-/Software
    - Detaillierung der Mess-, Erfassungs- und Steuergrößen (Was wollen wir beobachten?)
    - Einarbeitung in die Linux-Konfiguration und in passende Open-Source-Video-Tools
    - Vergleichsuntersuchung und Wahl eines geeigneten Einplatinen-Computers
2. Konzepterarbeitung
    - Vergleich der Plattformen PC und Einplatinen-Computer für die Messaufgabe
    - Betriebskonzept der Messung (Beleuchtung, Dauer, Abtastung, Zyklen)
    - Verbesserung des Messbetriebs (Ersatz der Schaltfunktion durch gesteuerte Zellsspannung, interne Strom- und Spannungsmessungen)

3. Entwicklung der Hardware des Steuer- und Messmoduls
    - Entwurf einer LED-Steuerung mit weißen und mehrfarbigen Leuchtdioden
    - Entwurf einer Lade-/Entladesteuerung
    - Spannung- und Strommessung im Ladekreis
    - Inbetriebnahme des Messaufbaus mit einem Einplatinen-Computer
4. Software des Steuer- und Messmoduls
    - Installation und Konfiguration einer geeigneten Linux-Variante
    - Anbindung von Open-Source-Software für Kamerasteuerung und Videogenerator
    - Entwicklung eines übergeordneten Steuerskriptes
    - Controller-Softwareentwicklung für das Steuer- und Messmodul
    - Alternative zur Controller-Software: Lowlevel-Steuerung der SPI-Schnittstelle in Linux
    - Steuersoftware für das Steuer- und Messmodul auf dem Einplatinen-Computer
5. Test, Funktionserprobung
    - Durchführung von Modul- und Softwaretest
    - Beurteilung von Performance und Stabilität
    - Visualisierung und Darstellung der Ergebnisse (Video mit Messwert-Kurven)
6. Fazit
    - Bewertung der gewählten Lösungen
    - Diskussion von erkannten Problemen
    - Aufzeigen von möglichen Verbesserungen

Dokumentation

Die entwickelte Software, die gewählten Lösungen und die Funktionsweise sind gut nachvollziehbar und für die zukünftige Nutzung zu dokumentieren.
Die Erprobungsergebnisse sind in aussagefähigem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch einordnend zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

A.3. C-Code

A.3.1. init.h

Listing A.1: Funktionen zur Initialisierung von GPIO und SPI

```
1  #ifndef STDIOH
2          #include <stdio.h>
3          #define STDIOH
4  #endif
5
6  #ifndef BCM2835
7          #include <bcm2835.h>
8          #define BCM2835
9  #endif
10
11 void spiInit(void){
12 // Initiallisierung der SPI-Schnittstelle
13 bcm2835_spi_begin();
14 // Das MSB wird zuerst gesendet
15 bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
16 // Der Idle des CS ist HIGH und die Daten werde ab der ersten Flanke
       übernommen
17 bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);
18 // Der Takt: 3,8kHz
19 bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_65536);
20 //
21 bcm2835_spi_chipSelect(BCM2835_SPI_CS0);
22 // Aktivitätspegel des CS
23 bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);
24 }
25
26 int gpioInit(void){
27 // GPIO's werden zu Ausgängen
28 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_03, BCM2835_GPIO_FSEL_OUTP);
29 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_11, BCM2835_GPIO_FSEL_OUTP);
30 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_13, BCM2835_GPIO_FSEL_OUTP);
31 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_15, BCM2835_GPIO_FSEL_OUTP);
32 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_16, BCM2835_GPIO_FSEL_OUTP);
33 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_18, BCM2835_GPIO_FSEL_OUTP);
34 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_32, BCM2835_GPIO_FSEL_OUTP);
35 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_33, BCM2835_GPIO_FSEL_OUTP);
36 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_35, BCM2835_GPIO_FSEL_OUTP);
37 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_36, BCM2835_GPIO_FSEL_OUTP);
```

```
38 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_37, BCM2835_GPIO_FSEL_OUTP);
39 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_38, BCM2835_GPIO_FSEL_OUTP);
40 bcm2835_gpio_fsel(RPI_BPLUS_GPIO_J8_40, BCM2835_GPIO_FSEL_OUTP);
41
42 // SDI- und CS-GPIO's werden HIGH gesetzt
43 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_11, HIGH);
44 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_13, HIGH);
45 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_15, HIGH);
46 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_18, HIGH);
47 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_32, HIGH);
48 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_33, HIGH);
49 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_35, HIGH);
50 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_37, HIGH);
51
52 // Relai- und Convert-GPIO's werden LOW gesetzt
53 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_03, LOW);
54 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_16, LOW);
55 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_36, LOW);
56 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_38, LOW);
57 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_40, LOW);
58 }
```

A.3.2. adc.h

Listing A.2: Funktionen zur Auswertung des ADC

```
1  #ifndef STDIOH
2          #include <stdio.h>
3          #define STDIOH
4  #endif
5
6  #ifndef BCM2835
7          #include <bcm2835.h>
8          #define BCM2835
9  #endif
10
11 int getAdc(int cs, int conv){
12 unsigned int i = 0;
13 unsigned char wbuf[3] = {0,0,0};
14
15 union buf{
16         unsigned int data;
17         unsigned char dataPart[3];
```

```
18 }dataSelect;
19
20 dataSelect.data = 0;
21
22 // Konvertieren
23 bcm2835_gpio_write(conv,HIGH);
24 bcm2835_delay(30);
25 // Chipselect
26 bcm2835_gpio_write(cs,LOW);
27 // Senden
28 for( i=3; i>0; i--){
29         dataSelect.dataPart[i-1] = bcm2835_spi_transfer(wbuf[i-1]);
30 }
31 // Konvertieren und Chipselect zurücksetzten
32 bcm2835_gpio_write(cs, HIGH);
33 bcm2835_gpio_write(conv,LOW);
34 // Daten an die richtige Stelle schieben
35 dataSelect.data = dataSelect.data >> 6;
36 // Daten zurückgeben
37 return dataSelect.data;
38 }
```

A.3.3. dac.h

Listing A.3: Funktionen zum Setzten des DAC

```
1  #ifndef STDIOH
2          #include <stdio.h>
3          #define STDIOH
4  #endif
5  #ifndef BCM2835
6          #include <bcm2835.h>
7          #define BCM2835
8  #endif
9
10 int setDac(int data, int cs){
11 unsigned int i = 0;
12
13 union buf{
14         unsigned int data;
15         unsigned char dataPart[4];
16 }dataSelect;
17
```

```
18 // Daten an die richtige Stelle schieben
19 dataSelect.data = data << 4;
20 // Die "Tabu-Bits" zu Null maskieren
21 dataSelect.data = dataSelect.data & 0b00000000000011111111111111110000;
22 //                                  xxddddddddddd00bbbbccccddddeeee0000
23
24 bcm2835_gpio_write(cs, LOW);
25 for( i=3; i>0; i--){
26         bcm2835_spi_transfer(dataSelect.dataPart[i-1]);
27 }
28 bcm2835_gpio_write(cs, HIGH);
29 return 1;
30 }
```

A.3.4. Konvertierungsfunktionen

Listing A.4: Funktionen zum Konvertieren zwischen Integer und Double

```
1  // Integer-Spannung wird in eine Doubel-Spannung übersetzt
2  double uiToUd( int ui ){
3  // Die Spannung wird dem Nutzer in V übergeben (max. 5V). Dafür
4  // muss die Spannung von den 2,5V Referenzspannung raufskaliert werden.
5  double ud = ( ui * (2.5*2) ) / 131071;
6  return ud;
7  }
8
9  // Double-Spannung wird in eine Integer-Spannung übersetzt
10 int udToUi( double ud ){
11 // Die Übergabe der Spannung erfolgt in V (max. 5V) und wird auf
12 // die Referenzspannung von 2.5V runterskaliert.
13 int ui = (ud / (2.5*2) ) * 131071;
14 return ui;
15 }
16
17 // Double-Strom wird in Integer-Strom übersetzt
18 int idToIi( double id ){
19 // Die Übergabe des Stromes erfolgt in uA (max. 1000uA)
20 int ii = ((id * 52427.0)/1000) + 65536;
21 return ii;
22 }
23
24 // Integer-Strom wird in Double-Strom übersetzt
25 double iiToId( int ii ){
```

```
26 // Der Strom wird dem Nutzer in uA übergeben (max. 1000ua)
27 double id = ( ( (ii-65536.0) / 52427) ) * 1000;
28 return id;
29 }
```

A.3.5. poti.h

Listing A.5: Funktionen zum Setzten der Potentiometer

```
1  #ifndef STDIOH
2  #include <stdio.h>
3  #define STDIOH
4  #endif
5  #ifndef BCM2835
6  #include <bcm2835.h>
7  #define BCM2835
8  #endif
9  #define AD5271 1
10 #define AD5270 2
11
12 #define POTI AD5271
13
14 void initPoti(int cs){
15
16 unsigned int i = 0;
17
18 union rbuf{
19         unsigned int data;
20         unsigned char dataPart[4];
21 }rbuffer;
22
23 union buf{
24         unsigned int data;
25         unsigned char dataPart[4];
26 }dataSelect;
27
28 dataSelect.data = 0x1C02;
29
30 bcm2835_gpio_write(cs, LOW);
31 bcm2835_delayMicroseconds(100);
32
33 for( i=2; i>0; i--){
```

```
34          rbuffer.dataPart[i-1] = bcm2835_spi_transfer(dataSelect.dataPart[
                i-1]);
35  }
36
37  bcm2835_delayMicroseconds(100);
38  bcm2835_gpio_write(cs, HIGH);
39  }
40
41  void sendPoti(unsigned int data, int cs){
42
43  unsigned int i = 0;
44
45  union rbuf{
46          unsigned int data;
47          unsigned char dataPart[4];
48  }rbuffer;
49
50  union buf{
51          unsigned int data;
52          unsigned char dataPart[4];
53  }dataSelect;
54
55  // Die Daten werden in die MSB's verschoben.
56  #if POTI == AD5271
57          dataSelect.data = data << 18;
58  #endif
59
60  #if POTI == AD5270
61          dataSelect.data = data << 16;
62  #endif
63
64  // Jetzt wird der Senden-Befehl in die Daten eigefügt
65  dataSelect.data = dataSelect.data | 0x4000000;
66
67  // Die Potis hätten gerne MODE 1
68  // bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);
69
70  bcm2835_gpio_write(cs, LOW);
71  bcm2835_delayMicroseconds(100);
72
73  for( i=4; i>2; i--){
74          rbuffer.dataPart[i-1] = bcm2835_spi_transfer(dataSelect.dataPart[
                i-1]);
75  }
```

```
76
77 bcm2835_delayMicroseconds(100);
78 bcm2835_gpio_write(cs, HIGH);
79 }
```

A.3.6. temperature.h

Listing A.6: Funktionen zum Auswerten des Thermometers

```
1  #ifndef STDIOH
2          #include <stdio.h>
3          #define STDIOH
4  #endif
5  #ifndef BCM2835
6          #include <bcm2835.h>
7          #define BCM2835
8  #endif
9
10 double getTemp(int cs){
11 unsigned int i = 0;
12 double erg = 0.0;
13
14 union wbuf{
15         int data;
16         char dataPart[4];
17 }buffer;
18
19 union buf{
20         int data;
21         char dataPart[4];
22 }dataSelect;
23
24 buffer.dataPart[2] = 0;
25 buffer.dataPart[1] = 3;
26 buffer.dataPart[0] = 196;
27
28 // Die Sensoren hätten gerne MODE 0
29         bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);
30 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_32,LOW);
31 for( i=4; i>0; i--){
32         dataSelect.dataPart[i-1] = bcm2835_spi_transfer(buffer.dataPart[i
               -1]);
33 }
```

```
34 bcm2835_gpio_write(RPI_BPLUS_GPIO_J8_32, HIGH);
35 // Zurück zu MODE 1
36 bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);
37 // Daten an die richtige Stelle schieben
38 dataSelect.data = dataSelect.data >> 19;
39 erg = dataSelect.data*0.0625;
40 return erg;
41 }
```

A.3.7. platine.h

Listing A.7: Headerfile des Steuerungsprogramms

```
1  #ifndef STRINGH
2          #include "string.h"
3          #define STRINGH
4  #endif
5  #ifndef TIMEH
6          #include "time.h"
7          #define TIMEH
8  #endif
9  #ifndef CTYPEH
10         #include <ctype.h>
11         #define CTYPEH
12 #endif
13 #ifndef STDIOH
14         #include <stdio.h>
15         #define STDIOH
16 #endif
17 #ifndef STDLIBH
18         #include <stdlib.h>
19         #define STDLIBH
20 #endif
21 #ifndef UNISTDH
22         #include <unistd.h>
23         #define UNISTDH
24 #endif
25 #ifndef STDIOH
26         #include <stdio.h>
27         #define STDIOH
28 #endif
29 #ifndef BCM2835H
30         #include <bcm2835.h>
```

```
31          #define BCM2835H
32  #endif
33  #ifndef STDLIBH
34          #include <stdlib.h>
35          #define STDLIBH
36  #endif
37  #ifndef PTHREADH
38          #include <pthread.h>
39          #define PTHREADH
40  #endif
41
42  // Parameter die eventuell mal geändert werden sollen
43  #define IHYSTOBEN       2.0     // Obere Hystersegrenze im Bezug zum
        Sollstrom in uA
44  #define IHYSTUNTEN      2.0     // Untere Hystersegrenze im Bezug zum
        Sollstrom in uA
45  #define ABTASTUNG       10      // Abtastfrequenz in Sekunden
46  #define ANZ_VALUE       6       // Anzahl der Parameter für eine
        Zyklierung (Steuerskript)
47  #define STEP            2       // Offset für Spannungskalibrierung
48  #define OFFSET          0.0     // Offset für Stromkallinbrierung
49  #define WINDOW          15.0// Fenster Mindestens 15
50  #define ARRAYLENGTH     100     // Länge des Stromarrays
51  #define MAXDIFF         0.06    // Schrittweitenreferenz für die
        Funktionen sink() und rise().
52  #define COLLECT         5       // Anzahl der von der Fuktion getI() zu
        sammelden Werte.
53
54  // Wahl des auf der Platine verwedeten Potis. Die Wahl ist entscheident
        für die Auflösung
55  // und damit für die Interpretation der Übergebenen Daten.
56  #define AD5271 1
57  #define AD5270 2
58
59  #define POTI AD5271             // Hier das Poti AD5271 oder AD5270
        auswählen!!!!
60
61  // Verteilung der Chipselects
62  #define DAC                     RPI_BPLUS_GPIO_J8_15    // Der CS für den
        Spannungs-ADC liegt auf dem GPIO22 (Pin 15)
63  #define ADC_V           RPI_BPLUS_GPIO_J8_13    // Der CS für den
        Spannungs-ADC liegt auf dem GPIO17 (Pin 13)
64  #define ADC_I           RPI_BPLUS_GPIO_J8_11    // Der CS für den Strom-
        ADC liegt auf dem GPIO27 (Pin 11)
```

```
65 #define ADC_CONV        RPI_BPLUS_GPIO_J8_16    // Der Konvertierungs-
       Trigger liegt auf dem GPIO23 (Pin 16)
66 #define TEMERATURE      RPI_BPLUS_GPIO_J8_32    // Der CS für den
       Temperatursensor liegt auf GPIO12 (Pin 32)
67 #define RELAIS_Zp       RPI_BPLUS_GPIO_J8_38    // Der CS für das Relai
       am Zellenpluspol auf den GPIO20 (Pin 38)
68 #define RELAIS_Zn       RPI_BPLUS_GPIO_J8_36    // Der CS für das Relai
       am Zellenminuspol auf den GPIO16 (Pin 36)
69 #define RELAIS_I        RPI_BPLUS_GPIO_J8_40    // Der CS für das Relai
       am Strompfad auf den GPIO21 (Pin 40)
70 #define LED_R           RPI_BPLUS_GPIO_J8_33    // Der CS für den
       Rotanteil der LED auf GPIO13 (Pin 33)
71 #define LED_G           RPI_BPLUS_GPIO_J8_35    // Der CS für den
       Rotanteil der LED auf GPIO19 (Pin 35)
72 #define LED_B           RPI_BPLUS_GPIO_J8_37    // Der CS für den
       Rotanteil der LED auf GPIO26 (Pin 37)
73
74 // Globale Variablen und Prototypen
75 struct value{
76         int current;    // Der Sollstrom
77         int voltage;    // Die Sollspannung
78         int sleep;              // Ruhezeit der Zelle
79         int led_R;              // Potiwert Rot
80         int led_G;              // Potiwert Grün
81         int led_B;              // Potiwert Blau
82         int dac;                // Aktueller DAC-Registerwert
83 } values;
84
85 int arrayI[ARRAYLENGTH];
86 int arrayRun = 0;
87 int medianRun = 0;
88 int phaseRun = 0;
89
90  int getValue( FILE*, struct value *ptrValues);
91  int checkEnd( FILE* controllFile );
92  int init(void);
93 double uiToUd( int ui );
94  int udToUi( double ud );
95  int idToIi( double id );
96 double iiToId( int ii );
97  int rise(int);
98  int sink(int);
99  void dacInit(void);
100  int getI(int);
```

```
101 void sigEnd(int);
102 void ende(void);
103 void cleanstdin(void);
```

A.3.8. platine.c

Listing A.8: Hauptprogramm

```
1  #include "platine.h"
2  #include "init.h"
3  #include "adc.h"
4  #include "dac.h"
5  #include "temperature.h"
6  #include "poti.h"
7  #include <signal.h>
8
9  // Flag wird TRUE zum beenden des Programms
10 volatile sig_atomic_t runFlag = 0;
11
12 int main(int argc, char* argv[]){
13 //Variablendeklaration
14 int c                  =  0;   // Container für die Verarbeitung der
       Aufrufoptionen. Wird bei
15                                                  // der Auswertung der
                                                       Übergabeparameter
                                                       verwendet.
16 int loop               = 0;
17 int timeA              = 0;    // Speichert den letzten
       Speicherzeitpunkt
18 int sinkFlag    = 0;    // Zeigt Entladung an
19 int riseFlag    = 0;    // Zeigt Ladung an
20 int current     = 0;    // Der aktuelle Strom
21 int voltage     = 0;    // Die aktuelle Spannung
22
23
24 // Hier stehen die Dateien in die gespeichert wird und die Steuerdatei
       drin.
25 struct messureData{
26         char* currentFileName;
27         char* voltageFileName;
28         char* controllFileName;
29         char* temperatureFile;
30         int loops;              // Wiederholungen des Zyklierplans
```

```
31 }data;
32
33 //Filepointer für die Speicherdatein von Strom und Spannung
34 FILE* currentFile;
35 FILE* voltageFile;
36 FILE* controllFile;
37 FILE* temperatureFile;
38
39 // Auswertung der Übergabeparameter
40 if(argc != 11){                 // Wurden nominal genug Parameter
       übergeben?
41         printf("Manuel\n");
42         printf("$PWD/platine -v Spannungsfile -i Stromfile -c Steuerfile
              -t Temperaturfile -l Anzahl_Loops\n");
43         return 1;
44 }
45
46 // Einrichten des Signalhandler auserplanmäßigen aber geordneten beenden
       des Programms.
47 signal(SIGINT,sigEnd);
48
49 // Auswertung der Übergabeparameter
50 while((c = getopt (argc, argv, "i:v:c:l:t:")) != -1)
51 switch(c){
52                 case 'i':
53                         data.currentFileName = optarg; // Speicherort für
                             Stromdaten OK
54                         break;
55                 case 'v':
56                         data.voltageFileName = optarg; // Speicherort für
                             Spannungsdaten OK
57                         break;
58                 case 'c':
59                         data.controllFileName = optarg;// Steuerfile
                            gefunden.
60                         break;
61                 case 'l':
62                         data.loops = atoi(optarg);
63                         break;
64                 case 't':
65                         data.temperatureFile = optarg;
66                         break;
67                 case '?':
68                 if(optopt == 'i')
```

```
69                      fprintf (stderr, "Speicherort␣␣fuer␣den␣Strom␣
                           fehlt.␣-%c␣\n", optopt);
70              else if(optopt == 'v')
71                      fprintf (stderr, "Speicherort␣␣fuer␣die␣Spannung␣
                           fehlt.␣-%c␣\n", optopt);
72          else if(optopt == 'c')
73                      fprintf (stderr, "Steuerdatei␣fehlt.␣-%c␣\n",
                           optopt);
74              else if(optopt == 't')
75                      fprintf (stderr, "Temperaturfile␣fehlt.␣-%c␣\n",
                           optopt);
76              else if(isprint(optopt))
77    fprintf (stderr, "Option␣unbekannt␣`-%c'.\n", optopt);
78              else
79    fprintf(stderr, "Paramerter␣unbekannt␣`\\x%x'.\n", optopt);
80              return 1;
81  default:
82              abort ();
83 }
84
85 // Öffnen aller benötigten Dateien
86 currentFile = fopen(data.currentFileName,"a");
87 if( currentFile == NULL ){
88         printf("Das␣Strom-File␣konnte␣nicht␣geöffnet␣werden.\n");
89         return 0;
90 }
91 voltageFile = fopen(data.voltageFileName,"a");
92 if( voltageFile == NULL ){
93         printf("Das␣Spannungs-File␣konnte␣nicht␣geöffnet␣werden.\n");
94         return 0;
95 }
96 controllFile = fopen(data.controllFileName,"r");
97 if( controllFile == NULL ){
98         printf("Das␣Steurer-File␣konnte␣nicht␣geöffnet␣werden.\n");
99         return 0;
100 }
101 temperatureFile = fopen(data.temperatureFile,"a");
102 if( temperatureFile == NULL ){
103         printf("Das␣Temperatur-File␣konnte␣nicht␣geöffnet␣werden.\n");
104         return 0;
105 }
106
107 fprintf(voltageFile,"#␣Spannung␣in␣V\n");
108 fprintf(currentFile,"#␣Strom␣in␣uA\n");
```

```
109 fprintf(temperatureFile,"# Temperatur in °C\n");
110
111 // Überprüfung des Zyklierplans
112 if( !filechecker( controllFile ) ){
113         return 0;
114 }
115
116 // Initialisierung der Hardware muss erfolgreich sein. Sonst Abbruch!
117 if( !init() ){
118         ende();
119         return 0;
120 }
121 /*****************************************************************
122 * Hier wird gemessen und gesteuert
123 * ***************************************************************/
124 // Schleifenanfang (Schleife: 1) für die Wiederholung des Zyklierplans.
125 for( loop = 0; loop < data.loops; loop++ ){
126 // Ausstiegspunkt für Programmende durch Strg-C aus der
    Wiederholungsschleife
127 if( runFlag == 1 ){
128 ende();
129 return 1;
130 }
131 // Schleifenanfang (Schleife: 2) zur Abarbeitung der einzelnen Positionen
    des Zyklierplans
132 do{
133         // Ausstiegspunkt für Programmende durch Strg-C aus der
            Zyklierplanschleife
134         if( runFlag == 1 ){
135                 ende();
136                 return 1;
137         }
138
139         // Die nächsten Sollwerte werden aus dem Zyklierplan geholt
140         getValue( controllFile, &values );
141
142         // Die Dimmung der LED wird angepasst
143         sendPoti(values.led_R, LED_R);
144         sendPoti(values.led_G, LED_G);
145         sendPoti(values.led_B, LED_B);
146
147         // Entscheidung ob es sich um einen Entlade- oder Ladevorgang
            handelt.
```

```
148         // Die Flags werden später benötigt um die Schwellspannung
                korrekt auswerten zu
149         // können.
150
151         //Wenn die zu erreichnde Spannung nidrieger ist als die aktuelle
                Zellspannung.
152         if( getAdc(ADC_V, ADC_CONV) >= values.voltage ){
153                 printf("SinkFlag\n");
154                 sinkFlag = 1;   // Es handelt sich um eine Entladung
155                 riseFlag = 0;
156                 // Also muss das Vorzeichen des positiv übergebenen
                        Stroms invertiert werden.
157                 values.current = values.current -( 2 * (values.current -
                        65535) );
158         }
159         else{
160                 printf("RiseFlag\n");
161                 sinkFlag = 0;
162                 riseFlag = 1; // Es handelt sich um eine Ladung
163         }
164
165         // Die aktuellen Sollwerte werden ausgegeben
166         printf("\tSchwell-Spannung = %f \n", uiToUd(values.voltage));
167         printf("\tSoll-Strom = %f \n", iiTold(values.current));
168
169         // Jetzt wird das Relais für den Strompfad geschlossen und ein
                Strom beginnt zu fließen.
170         bcm2835_gpio_write(RELAIS_I, HIGH);
171
172         // Da der DAC mit der Funktion "dacInit()" auf die Zellspannung
                eingestellt wurde
173         // Kann das Stromarry bei kleinen Strömen initialisiert werden.
                Auf diese weise wird
174         // der initiale Messfehler reduziert.
175         for( arrayRun = 0; arrayRun < ARRAYLENGTH; arrayRun++ ){
176                 arrayI[arrayRun] = getI( 0 );
177         }
178         arrayRun = 0;
179
180         // Schleifenanfang (Schleife: 3) für den Zyklier- und Messvorgang
181         do{
182                 // Ausstiegspunkt für Programmende durch Strg-C aus der
                        Mess- und Regelschleife
183                 if( runFlag == 1 ){
```

```
184                         ende();
185                         return 1;
186                 }
187
188                 // Dem Strom-Array wird ein neuer Stromwert hinzugefügt
                        der die erste Mittelwert-
189                 // bildung bereits durchlaufen hat.
190                 arrayI[arrayRun % ARRAYLENGTH] = getI(current);
191                 // Mit dem neuen Wert wird in der zweiten
                        Mittelwertbildungsebene eine neuer Wert
192                 // berechnet.
193                 for( medianRun = 0; medianRun < ARRAYLENGTH; medianRun++
                        ){
194                         current = current + arrayI[medianRun];
195                 }
196                 current = (current / ARRAYLENGTH);
197                 // Der nur einmal gemittelte Stromwert aus der Funktion "
                        getI()" wird durch den
198                 // Wert aus der zweiten Mittelung ersetzt.
199                 arrayI[arrayRun % ARRAYLENGTH] = getI(current);
200                 arrayRun++;
201
202                 // Jetzt wird die Spannung gemessen.
203                 voltage = getAdc( ADC_V, ADC_CONV );
204
205                 // Die Strom- und Spannungswerte werden zu Double-Werten
                        konvertiert,
206                 // mit Zeitstempeln veknüpft und in die Speicherdateien
                        und die Konsole geschrieben.
207                 if( ABTASTUNG <= (time(NULL)-timeA) ){
208                         timeA = time(NULL);
209                         fprintf(voltageFile,"%d\t%f␣\n", (unsigned)time(
                                NULL), uiToUd( voltage ));
210                         fprintf(currentFile,"%d\t%f␣\n",(unsigned)time(
                                NULL), iiToId( current ));
211                         fprintf(temperatureFile,"%d\t%f␣\n",(unsigned)
                                time(NULL), getTemp(TEMERATURE));
212                         printf("\nSchwell-Spannung␣=␣%f␣\n", uiToUd(
                                values.voltage));
213                         printf("Soll-Strom␣=␣%f␣\n", iiToId(values.
                                current));
214                         printf("%d␣%f␣V␣\n", (unsigned)time(NULL), uiToUd
                                ( voltage ));
```

```
215                     printf("%d %f uA \n",(unsigned)time(NULL), iiTold
                            ( current ));
216                     printf("%d %f °C \n\n",(unsigned)time(NULL),
                            getTemp(TEMERATURE));
217                 }
218
219                 // Nun wird geprüft ob der Strom zu groß, zu klein oder
                        auch innerhalb der Tolleranz ist.
220                 if( current > ( values.current + (idToli(IHYSTOBEN) -
                        65535) )){
221                     // Der Strom soll sinken.
222                     current = sink(current);
223                 }
224                 else if( current < (values.current - (idToli(IHYSTUNTEN)
                        - 65535) ) ){
225                     // Der Strom soll steigen.
226                     current = rise(current);
227                 }
228
229         // Ende (Schleife: 3) der Zyklierposition wenn die Spannung höher
                ist als die Schwellspannung und geladen wird
230         // oder wenn die Spannung keiner ist als die Schwellspannung und
                entladen wird.
231         }while( ((voltage < values.voltage) && riseFlag) || ((voltage >
                values.voltage) && sinkFlag) );
232         // Wurde die Schwellspannung erreicht wird der Zelle die im
                Steuerfile angegebene
233         // Ruhezeit gegönnt. Dazu werden alle Relais geöffnet und die
                Zelle ist komplett abgetrennt.
234         bcm2835_gpio_write(RELAIS_I, LOW);
235         bcm2835_gpio_write(RELAIS_Zp, LOW);
236         bcm2835_gpio_write(RELAIS_Zn, LOW);
237
238         // Die Ruhezeit wird abgewartet.
239         if( values.sleep > 0 ){
240                 sleep(values.sleep);
241         }
242         // Wenn das Ende des Zyklierplans nicht erreicht ist, wird der
                DAC wieder auf die Zellspannung
243         // hin initalisiert.
244         if( checkEnd( controllFile ) ){
245                 dacInit();
246         }
```

```
247 // Ist das Ende (Schleife: 2) des Zyklierplans erreicht. Wird die Mess-
    und Regelschleife verlassen.
248 }while( checkEnd( controllFile ) );
249
250
251 // Sind noch nicht alle Wiederholungen des Zyklusplans erfolgt, so wird
252 // der Filepointer zurückgesetzt der DAC neu initialisiert, falls die
    Zelle
253 // ihre Spannung strark verändert hat, und der Strompfad wieder
    geschlossen.
254 if( loop < data.loops ){
255         fseek(controllFile,0,SEEK_SET);
256         dacInit();
257         bcm2835_gpio_write(RELAIS_I, HIGH);
258 }
259 }// Schleifenende (Schleife: 1) für die Wiederholung der Zyklierung
260 /*****************************************************************
261 * ****************************************************************/
262 ende();
263 return 1;
264 }
265
266 // Fuktion zum Auslesen des Strom-ADC mit Fensterung der Messwerte und
    einer
267 // Mittelwertbildung über mehrer Messwerte.
268 int getI( int current ){
269 int j                   = 0;            // Schleifenläufer
270 int wert                = 0;            // Wert aus dem ADC
271 int ref                 = current;      // Referenzpunkt für die
    Fensterung
272 int acceptFlag  = 0;            // Zeigt gültigen Wert an
273
274 current = 0;
275 // Schleife die COLLECT-viele Werte zu rMittelwertbildung duchen lässt.
276 for( j=0; j<COLLECT; j++){
277         acceptFlag = 0;         // Ein neuer Wert soll gefunden werden.
278         do{
279                 wert = getAdc(ADC_I, ADC_CONV);
280                 // Der ADC-Wert muss innerhalb des zulässigen Fensters um
                    den Referenzwert
281                 // liegen, um bei der Mittelwertbildung berücksichtigt zu
                    werden.
282                 if( ((ref - idToli(WINDOW) ) < wert) && ((ref + idToli(
                    WINDOW) ) > wert)){
```

```
283                         acceptFlag = 1;
284                         current = current + wert;
285                 }
286                 // Der Eingangsspannung des ADC wird einen Moment Zeit
                        gegeben, damit diese
287                 // sich von der letzten Abtastung "erholen" kann.
288                 usleep(2000);
289         }while(acceptFlag == 0);
290 }
291 current = (current / COLLECT); // Mittelwertbildung!
292 return current;
293 }
294
295 /*****************************************************************
296 * Für die Regelung des Lade-/Entladestroms werden die Funktionen rise()
297 * und sink() benötigt. Diese passen den Strom so an das dieser wieder
298 * innerhalb einer Hysterese um den Sollwert liegt. Die Größe der
299 * Hysterese wird mit Makros festgelegt.
300 * ***************************************************************/
301 int rise(int current){
302 int maxRun = 0;
303 int diff        = 0;
304 // Schleife zur schrittweisen Erhöhung des Stroms.
305 do{
306         // Es wird ermittelt wie groß die Differenz zwischen Soll-
307         // und Istwert ist.
308         diff = values.current - current;
309
310         // Die Differenz wird zu einem Stellwert "konvertiert". Denn die
311         // Änderungsgeschwindigkeit des strom sollte nicht zu groß sein,
                um ein
312         // Aufschwingen des Systems zu verhindern.
313         if( diff > (700 * (idToli(MAXDIFF)-65535)) ){
314                 diff = 6 * (idToli(MAXDIFF)-65535);
315         }else if( diff > (150 * (idToli(MAXDIFF)-65535)) ){
316                 diff = 3 * (idToli(MAXDIFF)-65535);
317         }else if( diff > (50 * (idToli(MAXDIFF)-65535)) ){
318                 diff = 2 * (idToli(MAXDIFF)-65535);
319         }else if( diff > (idToli(MAXDIFF)-65535) ){
320                 diff = (idToli(MAXDIFF)-65535);
321         }
322         // Der DAC-Registerwert wird erhöht und gesetzt.
323         values.dac = values.dac + diff;
324         setDac( values.dac, DAC );
```

```
325
326         // Der Strom wird gemessen und durchläuft ach hier beide
                Mittelwertbildungen.
327         arrayI[arrayRun % ARRAYLENGTH] = getI(current);
328         for( medianRun = 0; medianRun < ARRAYLENGTH; medianRun++ ){
329                 current = current + arrayI[medianRun];
330         }
331         current = (current / ARRAYLENGTH);
332         arrayI[arrayRun % ARRAYLENGTH] = getI(current);
333         arrayRun++;
334
335         maxRun++;
336         // Die Schleife läuft so lange weiter bis der Strom innerhalb des
                Tolleranzbereichs lieg.
337         // Maximal jedoch 10 Durchläufe um Datenspeicherung zu
                ermöglichen.
338 }while( (current <= (values.current - (idToli(IHYSTUNTEN) - 65535)) && (
        maxRun < 10) ));
339 return current;
340 }
341
342 int sink(int current){
343 int maxRun = 0;
344 int diff        = 0;
345
346 // Schleife zur schrittweisen Verringerung des Stroms.
347 do{
348         // Es wird ermittelt wie groß die Differenz zwischen Soll-
349         // und Istwert ist.
350         diff = current - values.current;
351
352         // Die Differenz wird zu einem Stellwert "konvertiert". Denn die
353         // Änderungsgeschwindigkeit des strom sollte nicht zu groß sein,
                um ein
354         // Aufschwingen des Systems zu verhindern.
355         if( diff > (700 * (idToli(MAXDIFF)-65535)) ){
356                 diff = 6 * (idToli(MAXDIFF)-65535);
357         }else if( diff > (100 * (idToli(MAXDIFF)-65535)) ){
358                 diff = 3 * (idToli(MAXDIFF)-65535);
359         }else if( diff > (idToli(MAXDIFF)-65535) ){
360                 diff = (idToli(MAXDIFF)-65535);
361         }
362         // Der DAC-Registerwert wird verringert und gesetzt.
363         values.dac = values.dac - diff;
```

```
364         setDac( values.dac, DAC );
365
366         // Der Strom wird gemessen und durchläuft ach hier beide
                Mittelwertbildungen.
367         arrayI[arrayRun % ARRAYLENGTH] = getI(current);
368         for( medianRun = 0; medianRun < ARRAYLENGTH; medianRun++ ){
369                 current = current + arrayI[medianRun];
370         }
371         current = (current / ARRAYLENGTH);
372         arrayI[arrayRun % ARRAYLENGTH] = getI(current);
373         arrayRun++;
374
375         maxRun++;
376         // Die Schleife läuft so lange weiter bis der Strom innerhalb des
                Tolleranzbereichs lieg.
377         // Maximal jedoch 10 Durchläufe um Datenspeicherung zu
                ermöglichen.
378 }while( (current >= (values.current + (idToIi(IHYSTOBEN) - 65535)) ) && (
    maxRun < 10) );
379 return current;
380 }
381
382 /*****************************************************************
383 * Die Eingabe der Daten über das Steuerskript erfolgt mit dem Datentyp
384 * double. Die Datenerfassung und die Regelung laufen jedoch mit ganz-
385 * zahligen Integern ab. Die folgenden Funktionen übernehmen die
386 * Übersetzung.
387 * ***************************************************************/
388 // Integer-Spannung wird in eine Doubel-Spannung übersetzt
389 double uiToUd( int ui ){
390 // Die Spannung wird dem Nutzer in V übergeben (max. 5V). Dafür
391 // muss die Spannung von den 2,5V Referenzspannung raufskaliert werden.
392 double ud = ( ui * (2.5*2) ) / 131071;
393 return ud;
394 }
395
396 // Double-Spannung wird in eine Integer-Spannung übersetzt
397 int udToUi( double ud ){
398 // Die Übergabe der Spannung erfolgt in V (max. 5V) und wird auf
399 // die Referenzspannung von 2.5V runterskaliert.
400 int ui = (ud / (2.5*2) ) * 131071;
401 return ui;
402 }
403
```

```
404 // Double-Strom wird in Integer-Strom übersetzt
405 int idToli( double id ){
406 // Die Übergabe des Stromes erfolgt in uA (max. 1000uA)
407 int ii = ((id * 52427.0)/1000) + 65536;
408 return ii;
409 }
410
411 // Integer-Strom wird in Double-Strom übersetzt
412 double iiTold( int ii ){
413 // Der Strom wird dem Nutzer in uA übergeben (max. 1000ua)
414 double id = ( ( (ii-65536.0) / 52427) ) * 1000;
415 return id;
416 }
417
418 /*********************************************************************
419 * Alle Funktionen die den Zyklierablauf steuern
420 * ********************************************************************/
421 // Initialisierung des DAC um unzulässige Einschaltströme zu vermeiden.
422 void dacInit( void ){
423 int soll = 0;
424 // Schließen der Zellrelais um die Spannungsmessung zu ermöglichen.
425 bcm2835_gpio_write(RELAIS_Zp, HIGH);
426 bcm2835_gpio_write(RELAIS_Zn, HIGH);
427 sleep(1);       // Relais sind nicht immer die schnellsten.
428 // Spannungswert wird gelesen.
429 soll = getAdc( ADC_V, ADC_CONV);
430 printf("dacInit() U = %d  = %f \n",soll, uiToUd(soll));
431 // Vor dem Senden der Spannung an den DAC sollte diese halbiert werden,
432 // Denn der Ausganstreiber nach dem DAC hat eine Verstärkung von zwei.
433 values.dac = (soll/2);
434 setDac( values.dac, DAC );
435 }
436
437 // Initialisierung des Programms
438 int init(void){
439 // DER SoC des Raspoberry wird initialisiert.
440 if (!bcm2835_init()){
441         printf("Initialisirung des bcm2836 fehlgeschlage!\n");
442         printf("Eventuell \"sudo\" vergessen.");
443         return 0;
444 }
445 gpioInit();                     // Die GPIO-Pins werden initialisiert
446 spiInit();                      // SPI wird initialisiert.
447 values.current = 0;     // Alle globalen Variablen werden erst mal
```

```
448 values.dac = 0;          // Null gesetzt.
449 values.sleep = 0;
450 values.voltage = 0;
451 initPoti(LED_R);         // Die Eingangsregister der Potentiometer müssen
452 initPoti(LED_G);         // für die externe Ansprache freigeschaltet
        werden.
453 initPoti(LED_B);
454 getAdc( ADC_V, ADC_CONV );       // Die ADC liefern beim ersten Auslesen
455 getAdc( ADC_I, ADC_CONV );       // falsche Werte. Desshalb werden
        zunächst
456 getAdc( ADC_V, ADC_CONV );       // zwei davon verworfen.
457 getAdc( ADC_I, ADC_CONV );
458 usleep(500);
459 dacInit();                                  // Jetzt wird der DAC
        initialisiert.
460 return 1;
461 }
462
463
464 // getValue() holt den nächsten Datensatz aus dem Steuerskript.
465 int getValue( FILE* controllFile ,  struct value *ptrValues){
466
467 double wert = 0;
468 int i           = 0;
469 char aktion[10];
470 // Definition der Schlüsselbegriffe für die Identifizierung der
471 // Werte im Steuerskript der Zyklierung.
472 char strom[]    = "strom";
473 char spannung[] = "spannung";
474 char sleeping[] = "sleeping";
475 char led_R[]    = "led_R";
476 char led_G[]    = "led_G";
477 char led_B[]    = "led_B";
478
479 // Hier werden ANZ_VALUE-viele Werte aus dem Steuerskript ausgelesen.
480 // Der Vergleich der Schlüsselbegriffe ermöglicht die Zuordnung.
481 for( i = 0; i < ANZ_VALUE; i++ ){
482         fscanf(controllFile,"%s␣%lf␣\n",aktion, &wert);
483         if( 0 == strcmp(strom,aktion)){                  // Auswertung
            Strom
484                 ptrValues->current = idToli( wert );
485         }
486         if( 0 == strcmp(spannung,aktion) ){              // Auswertung
            Spannung
```

```
487                 ptrValues->voltage = udToUi( wert );
488         }
489         if( 0 == strcmp(sleeping,aktion) ){             // Auswertung
                Sleep
490                 ptrValues->sleep = wert;
491         }
492         if( 0 == strcmp(led_R,aktion) ){                // Auswertung
                LED_R
493                 ptrValues->led_R = wert;
494         }
495         if( 0 == strcmp(led_G,aktion) ){                // Auswertung
                LED_G
496                 ptrValues->led_G = wert;
497         }
498         if( 0 == strcmp(led_B,aktion) ){                // Auswertung
                LED_B
499                 ptrValues->led_B = wert;
500         }
501 }
502 return 1;
503 }
504
505 // Prüft ob das Ende des Zyklusses erreicht ist
506 int checkEnd( FILE* controllFile ){
507 int wert                = 0;
508 long pos                = 0;
509 char end[]              = "end";        // Schlüsselbegriff
510 char aktion[10];
511
512 // Die aktuelle Position des Filepointers wird gespeichert
513 pos = ftell(controllFile);
514 // Dann wird die nächste Zeile aus der Steuerdatei ausgelesen.
515 fscanf(controllFile,"%s %d\n",aktion, &wert);
516 // Jetzt wird geprüft ob das Schlüsselwort "end" ist.
517 if( 0 == strcmp(end,aktion)){
518         // Wenn ja zeigt die Null als Rückgabewert das Ende an.
519         return 0;
520 }
521 // Wenn nicht muss der Filepointer an seine alte Position zurück
522 // damit das Schlüsselwort an anderer Stelle ausgewertet werden kann.
523 fseek(controllFile,pos,SEEK_SET);
524 return 1;
525 }
526
```

```
527 // Signalhandler: Bei SIGINT (STRG-C) setzt er das Flag, so dass die
    Funktion
528 // "end()" ausgeführt und die main beendet wird.
529 void sigEnd( int signum ){
530 runFlag = 1;
531 }
532
533 void ende(void){
534 // Die Zelle und dr Strompfad werden getrennt.
535 bcm2835_gpio_write(RELAIS_I, LOW);
536 bcm2835_gpio_write(RELAIS_Zp, LOW);
537 bcm2835_gpio_write(RELAIS_Zn, LOW);
538
539 // Die aktivierung der bcm2836-Sonderfunktionen wird rückgängig
540 // gemacht und alle Dateien ordungsgemäß geschlossen um Datenverlust
541 // vermeiden.
542 fcloseall();
543 bcm2835_spi_end();
544 bcm2835_close();
545 printf("Ordentlich Beendet!\n");
546 }
547
548 // Diese Funktion schaut sich alle Positionen des Steurskriptes an und
    sucht
549 // unstimmigkeiten.
550 int filechecker( FILE* controllFile ){
551 double wert     = 0;
552 int i                   = 0;
553 char buf[2];
554 char c                  = 0;
555 char aktion[10];
556 // Definition der Schlüsselbegriffe
557 char strom[]    = "strom";
558 char spannung[] = "spannung";
559 char sleeping[] = "sleeping";
560 char led_R[]    = "led_R";
561 char led_G[]    = "led_G";
562 char led_B[]    = "led_B";
563
564 // Zur Zeit wird nur nach Tiefenentladung gesucht.
565 do{
566         for( i = 0; i < ANZ_VALUE; i++ ){
567                 fscanf(controllFile,"%s %lf \n",aktion, &wert);
568                 /*
```

```
569                 if( 0 == strcmp(strom,aktion)){                  //
                        Auswertung Strom
570                     current = idToli( wert );
571                 }*/
572                 if( 0 == strcmp(spannung,aktion) ){              //
                        Auswertung Spannung
573                     // Die Spannung wird untesucht ob sie unter 2V
                            liegt.
574                     if( wert < 2.0 ){
575                         // Wenn ja wird eine rot unterlegte
                                Warnung ausgegeben.
576                         printf("\033[41mSpannungsschwelle unter 2
                                V gefunden!\n");
577                         do{
578                             // Der Tastaturpuffer wird
                                    gelöscht. - Entspricht fflusch
                                    ()
579                             cleanstdin();
580                             // Der anwender wird gefragt ob
                                    er dass wirklich möchte
581                             printf("Trotzdem Fortsetzten? Y/N
                                    \n");
582                             // und seine Antwort abgewartet.
583                             fgets(buf, 2, stdin);
584                             // Konvertierung von Strin zu
                                    Char.
585                             sscanf(buf, "%c", &c);
586                             // ERs wird auf ein Y, y, N oder
                                    n gewartet.
587                         }while( (c != 'Y') && (c != 'y') && (c !=
                                'N') && (c != 'n' ) );
588                         // Die Rot-Hinterlegung wird
                                zurückgesetzt.
589                         printf("\033[m");
590                         if( (c == 'N') || (c == 'n') ){
591                             // Wenn die Tiefenentladung nicht
                                    beabsichtigt
592                             // wird Null zurückgegeben.
593                             return 0;
594                         }
595                     }
596                 }
597                 /*
```

```
598                 if( 0 == strcmp(sleeping,aktion) ){              //
                        Auswertung Sleep
599                         sleep = wert;
600                 }
601                 if( 0 == strcmp(led_R,aktion) ){                 //
                        Auswertung Sleep
602                         led_R = wert;
603                 }
604                 if( 0 == strcmp(led_G,aktion) ){                 //
                        Auswertung Sleep
605                         led_G = wert;
606                 }
607                 if( 0 == strcmp(led_B,aktion) ){                 //
                        Auswertung Sleep
608                         led_B = wert;
609                 }*/
610         }
611 }while( checkEnd(controllFile) );
612 // Der Filepointer wird zur weiteren Verwendung wieder auf den Anfang
        gesetzt.
613 fseek(controllFile,0,SEEK_SET);
614 return 1;
615 }
616
617 // Linuxersatz für die Funktin fflush() welche nur unter Windows
        funktioniert.
618 void cleanstdin(void){
619 char c = 0;
620 do{
621         c =getchar();
622 }while(c != '\n' && c!= EOF );
623 }
```

A.3.9. dacAnalyzer.c

Listing A.9: Programm zur Fehlersuche beim DAC

```
1 int main(void){
2 double adcValue = 0;
3 int bit = 0;
4 int mask = 1;
5
6 int i = 0;
```

```
7  int j = 0;
8  int y = 0;
9  int k = 0;
10 int l = 0;
11
12 FILE* ptrLogfile = NULL;
13
14 int data = 0x80000;
15
16 char fileName[] = "/media/65b55418-38c6-4990-b8ba-a1cf2dee2462/DAC_Test/
       dacLogfile.txt";
17
18
19 ptrLogfile = fopen(fileName,"w");
20 if( ptrLogfile == NULL ){
21         printf("Datei nicht geoeffnet!\n");
22 }
23
24
25 if (!bcm2835_init()){
26         printf("sudo vergessen \n");
27         return 1;
28 }
29
30 gpioInit();
31 spiInit();
32 getAdc(RPI_BPLUS_GPIO_J8_11);
33
34 for( k=0; k<3; k++){            // Alle SPI-Modi
35         for( i = 0; i<=63 ; i++){       // Die ersten 5 Bit
36         printf("I_schleife\n");
37                 for( j=0; j<=63 ; j++){  // Die letzten 6 Bit
38
39                         data = data | (i);
40                         data = data | (j << 18);
41                         sendDac(data, RPI_BPLUS_GPIO_J8_15);
42                         sleep(2);
43                         adcValue = getAdc(RPI_BPLUS_GPIO_J8_11);
44
45                         printf("Zeit: %d Mode: %d LSBs: %d MSBs: %
                               d ADC: %f Binaer: ",(unsigned)time(NULL),
                               k, i, j, adcValue);
46                         for( l=0; l<24; l++ ){
47                                 if( 0<(data & mask) ){
```

```
48                                              bit = 1;
49                                      }
50                                      else{
51                                              bit=0;
52                                      }
53                                      printf("%d",bit);
54                                      mask = mask << 1;
55                              }
56                              printf("_\n");
57                              mask = 1;
58                              data = 0x80000;
59
60                              fprintf(ptrLogfile, "Zeit: %d Mode: %d LSBs: %d
                                    MSBs: %d ADC: %f Binaer: ",(unsigned)time(NULL
                                    ), k, i, j, adcValue);
61                              for( l=0; l<24; l++ ){
62                                      if( 0<(data & mask) ){
63                                              bit = 1;
64                                      }
65                                      else{
66                                              bit=0;
67                                      }
68                                      fprintf(ptrLogfile, "%d",bit);
69                                      mask = mask << 1;
70                              }
71                              fprintf(ptrLogfile,"_\n");
72                              mask = 1;
73                              data = 0x80000;
74                      }
75                      j=0;
76              }
77
78              if( k==0 ){
79                      bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);
80              }
81              else if( k==1 ){
82                      bcm2835_spi_setDataMode(BCM2835_SPI_MODE2);
83              }
84              else if( k==2 ){
85                      bcm2835_spi_setDataMode(BCM2835_SPI_MODE3);
86              }
87      }
88
89      fclose(ptrLogfile);
```

```
90 bcm2835_spi_end();
91 bcm2835_close();
92 return 1;
93 }
```

A.4. Matlabskripte

A.4.1. Matlab-Skript zur Auswertung der Performance

Listing A.10: Performance-Auswertung

```
1  anz = 0;
2  fehlzeit = 0;
3  Fehler = 0;
4  maximum = 0;
5  fprintf('******************************************************\n');
6  fprintf('Start\n\n');
7  n2 = [1:1:length(data)-1];
8  i = 0;
9  dauer=(max(data)-min(data));
10 for j=1:length(data)-1;
11  diff = data(j+1)-data(j);
12  if ( diff > 10 )
13      anz = anz+1;
14      fprintf('%d\n%d\n',data(j),data(j+1));
15      fprintf('Differenz = %d\n\n',diff);
16      fehlzeit = fehlzeit + diff;
17      if( diff > maximum )
18          maximum = diff;
19      end
20  end
21 end
22 Fehler = (100 / dauer) * fehlzeit;
23 fprintf('Anzahl=%d\n',anz);
24 fprintf('Dauer=%d h\n',dauer);
25 fprintf('Fehlzeit=%d h\n',fehlzeit);
26 fprintf('Fehler=%d Prozent\n',Fehler);
27 fprintf('Maximum=%d sec\n',maximum);
28 fprintf('Stop\n\n');
```

A.4.2. Matlab-Skript zur graphischen Darstellung der Performance

Listing A.11: Graphische Performance-Auswertung

```
n1 = [1:1:length(data)-1];
for i=1:length(data)-1;
 diff(i) = data(i+1)-data(i);
end

figure;
hold on;
plot(n1,diff,'.');
grid on;
title('Langzeittest PcDuino mit S-ATA');
xlabel('Messung n');
ylabel('Messpunktdifferenz in s');
hold off;
```

A.4.3. Matlab-Skript zur Auswertung der Zyklierung

Listing A.12: Auswertung der Zyklierung

```
soll_dmm_h = zeros( [length(i_dmm),1] );
soll_dmm_l = zeros( [length(i_dmm),1] );
soll_pi_h  = zeros( [length(i_pi),1] );
soll_pi_l  = zeros( [length(i_pi),1] );
t_dmm = zeros( [length(time_dmm),1] );
t_pi = zeros( [length(time_pi),1] );
i_pi_2 = zeros( [length(i_pi),1] );
%%
for j=1:length(time_dmm)
    t_dmm(j)=time_dmm(j)-(min(time_dmm)-1);
end

for j=1:length(time_pi)
    t_pi(j)=time_pi(j)-(min(time_pi)-1);
end
%%
for i=1:length(i_dmm)
    soll_dmm_h(i) = 0.00002;
    soll_dmm_l(i) = -0.00002;
end

```

```
%%
for i=1:length(i_pi)
    soll_pi_h(i)= 20/1000000;
    soll_pi_l(i)= -20/1000000;
    i_pi_2(i)=i_pi(i)/1000000;
end

%%
figure;
subplot(3, 1, 1);
hold on;
plot(t_dmm, i_dmm,'b', t_dmm, soll_dmm_h,'r--', t_dmm, soll_dmm_l,'r--');
title('Stromverlauf für +/-100uA mit DMM4020 gemessen');
xlabel('Zeit in sec');
ylabel('I [A]');
% ylim([85*10^(-6) 102*10^(-6)]);
% xlim([270 780]);
grid on;
hold off;

%%
subplot(3, 1, 2);
hold on;
plot(t_pi, i_pi_2,'b', t_pi, soll_pi_h,'r--', t_pi, soll_pi_l,'r--');
title('Stromverlauf für +/-100uA mit Raspberry-Pi gemessen');
xlabel('Zeit in sec');
ylabel('I [A]');
% ylim([85*10^(-6) 102*10^(-6)]);
% xlim([270 780]);
grid on;
hold off;

subplot(3, 1, 3);
hold on;
plot(t_pi, u_pi,'b');
title('Spannungsverlauf für +/-100uA mit Raspberry-Pi gemessen');
xlabel('Zeit in sec');
ylabel('U [V]');
% ylim([2.5 4.5]);
% xlim([270 780]);
grid on;
hold off;

figure;
```

```
66 subplot(2, 1, 1);
67 hold on;
68 plot(t_dmm, i_dmm,'b', t_dmm, soll_dmm_h,'r--', t_dmm, soll_dmm_l,'r--');
69 title('Stromverlauf für +/-100uA mit DMM4020 gemessen');
70 xlabel('Zeit in sec');
71 ylabel('I [A]');
72 % ylim([85*10^(-6) 102*10^(-6)]);
73 % xlim([270 780]);
74 grid on;
75 hold off;
76
77 %%
78 subplot(2, 1, 2);
79 hold on;
80 plot(t_pi, i_pi_2,'b', t_pi, soll_pi_h,'r--', t_pi, soll_pi_l,'r--');
81 title('Stromverlauf für +/-100uA mit Raspberry-Pi gemessen');
82 xlabel('Zeit in sec');
83 ylabel('I [A]');
84 % ylim([85*10^(-6) 102*10^(-6)]);
85 % xlim([270 780]);
86 grid on;
87 hold off;
```

A.5. Steuerskript

Listing A.13: Shellskript zur Verwaltung der Prozesse

```
1  #!/bin/bash
2
3  MESSUNG="Zelle_Bemerkung_Datum_"                    # HIER DIE MESSUNG
       BENENNEN  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
4
5  STROM=$MESSUNG'Strom.txt'
6  SPANNUNG=$MESSUNG'Spannung.txt'
7  TEMPERATUR=$MESSUNG'Temperatur.txt'
8  CELL="DMM"
9
10 # Ordner für die Daten der aktuellen Messung
11 ORDNER="/media/65b55418-38c6-4990-b8ba-a1cf2dee2462/PI/MitPlatine
       /20150901"              # HIER VERZEICHNIS AKTUALLISIEREN
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
12
```

```
13 # Ordner für die Programme
14 STEUERUNG="/home/pi/Programmierung/Steuerung/"
15 DMMORT="/home/pi/Desktop/Messung"
16 # Name des Zyklierplans
17 PLANORT="/home/pi/Programmierung/Steuerung"
18 PLANNAME="steuerfile"
19
20 # Start der Zellzyklierung
21 sudo $STEUERUNG/steuerung -i $ORDNER/$STROM -v $ORDNER/$SPANNUNG -t
       $ORDNER/$TEMPERATUR -c $PLANORT/$PLANNAME -l 500 &
22 PID_PLATINE=$!
23
24 # Start der Spannungsmessung mit dem DMM4020
25 #$DMMORT/DMM4020_reader_v2/DMM4020_reader_v2 -d /dev/ttyUSB0 -f /media/
       Extern1/PI/20150618/$CELL"_spannung_".txt -i 10 -m 1 &
26 #PID_SPANNUNG=$!
27
28 # Start der Strommessung mit dem DMM4020
29 $DMMORT/DMM4020_reader_v2/DMM4020_reader_v2 -d /dev/ttyUSB0 -f $ORDNER/
       $CELL"_strom_".txt -i 10 -m 2 &
30 PID_STROM=$!
31
32 fswebcam -d /dev/video0 --loop 10 -F5 --png 0 --scale 1600x1200 --set -s
       "White␣Balance␣Temperature,␣Auto=False" --save $ORDNER/Bilder/$CELL'%
       s'.png &
33 PID_CAMERA=$!
34
35 read -n1 -s
36
37 kill -SIGINT $PID_PLATINE
38 #kill -SIGTERM $PID_SPANNUNG
39 kill -SIGTERM $PID_STROM
40 kill -SIGTERM $PID_CAMERA
```

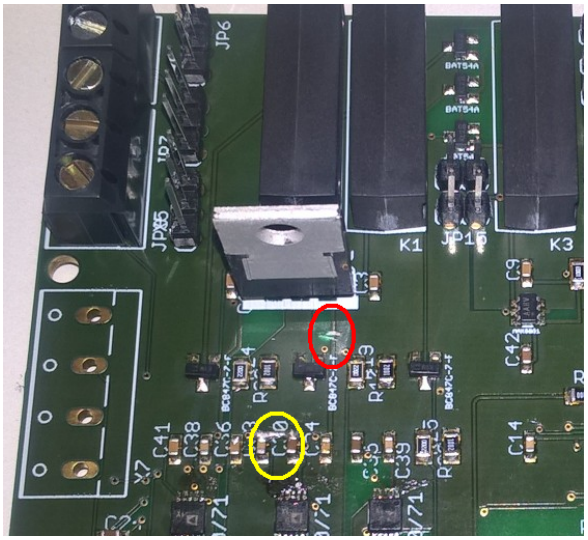

Abbildung A.1.: Änderungen an dem Platinenlayout

A.6. Kurzanleitung für die Inbetriebnahme von Erweiterungsplatine und Raspberry-Pi

- Die Platine dem Eagle-Layout (CD) entsprechend löten.
- Die 5V Versorgung der Potentiometer trennen (Bild A.1 roter Kreis).
- Die 3,3V Spannungsversorgung überbrücken (Bild A.1 gelber Kreis).
- Den Raspberry-Pi der Kurzanleitung A.2 auf Seite 129 entsprechend anschließen.
- Externe Hardware anschließen (Spannungsversorgung (Platine), USB-Netzteil (Pi), Festplatte, USB-HUB, Kamera, Tastatur/Maus, Bildschirm, Tektronix DMM4020).
- Das Image des Messsystems (siehe CD) auf eine SD-Karte (min. 4GB) kopieren und das System booten.
- Der Quellcode muss an das verwendete Potentiometer angepasst und kompiliert werden. Für die Anpassung muss nur das Makro „POTI“ in der Headerdatei „platine.h“ auf den Wert „AD5270“ oder „AD5271“ geändert werden.
- Zyklierplan vorgeben und das Shell-Skript Programmstart.sh anpassen und ausführen.

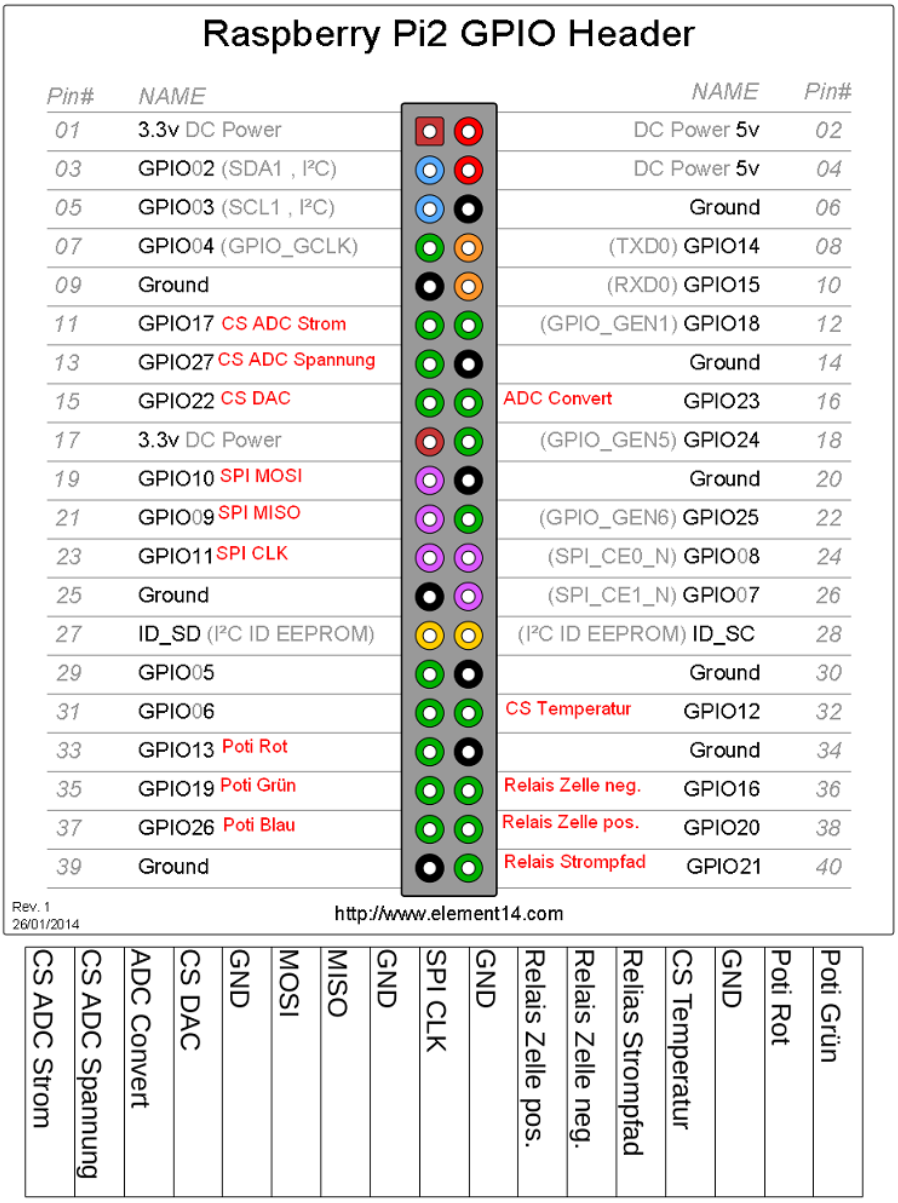

Abbildung A.2.: Oben ist der GPIO-Header des Raspberry-Pi abgebildet. Pin 1 befindet sich auf der inneren Reihe und dem USB-Port abgewandten Seite. Im unteren Teil ist die Pinbelegung der Platine abgebildet. In der Darstellung befindet sich die Pinleiste auf der dem Anwender zugewandten Seite [8].

A.7. Inhalt der DVD

Die Dateien von Soft und Hardware die für den Messaufbau nötig sind befinden sich auf der DVD. Auch die Matlab- und Shell-Skripte sind dort zu finden. Hinzu kommen die Tools von Grießbach [14] und ein Handzettel zur Verdrahtung von Raspberry-Pi und Erweiterungsplatine. Der Datenträgersatz wurde bei Erst- und Zweitprüfer hinterlegt. Der Inhalt der einzelnen Datenträger ist nachfolgend aufgelistet:

1. Die PDF-Datei der Bachelorarbeit.
2. C-Code für das Zykliersystem und seine Komponenten.
3. Datenblätter für die Komponenten der Erweiterungsplatine.
4. Die Eagle-Dateien für Erweiterungsplatine und Temperatursensor.
5. Die in der Arbeit verwendeten Matlabskripte.
6. Ein Raspbian-Image mit fertig eingerichtetem Zykliersystem.
7. Das Shell-Skript zur Steuerung der Zyklierung.
8. Die Tools von Grießbach [14].
9. Eine Bauteilliste der Erweiterungsplatine.
10. Ein Handzettel zur Verdrahtung von Raspberry-Pi und Erweiterungsplatine.
11. Video einer Elektrode während ihrer Zyklierung.

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, 19. September 2015	
Ort, Datum	Unterschrift