Institut für Elektrotechnik

Masterarbeit

Realisierung eines softwaregestützen Balancings mit Hilfe von Microcontroller

Michael Bernhard Plozner, BSc

September 2020

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 24.08.2020

Unterschrift Verfasser/in
Michael Bernhard, Plozner

Danksagung

Ich möchte an dieser Stelle meinen Dank an o. Univ.-Prof. Dipl.-Ing. Dr. techn. Helmut Weiß richten, der mir als mein Betreuer für diese Masterarbeit, stets mit Rat und viel Geduld beiseitegestanden ist. Außerdem möchte ich hiermit festhalten, dass mir von seiner Seite aus eine uneingeschränkte und angenehme Arbeitsumgebung geschaffen wurde. Die Tatsache, dass mir die Möglichkeit gegeben wurde, diese Masterarbeit auf Basis eines geplanten und umgesetzten Prototypen schreiben zu können, war für mich sehr wertvoll, da ich einen Praxisbezug sehr schätze.

Einen weiteren Dank möchte ich an alle Mitarbeiter des Institutes für Elektrotechnik richten, da sie sich bei jeder Frage ausreichend Zeit genommen haben, um eine professionelle Antwort bzw. Einschätzung liefern zu können.

Vielen Dank auch an meine Eltern und an meine Freunde, die mich dabei unterstützt haben, diese Arbeit auch in einer global schwierigen Zeit fertigzustellen.

Kurzfassung

Lithium-Ionen Akkumulatoren sind heutzutage der Standard der elektrochemischen Energiespeicherung. In der Mobilität, in transportablen netzunabhängigen Geräten oder im stationären Bereich werden sie in großer Vielfalt eingesetzt. Elektrische Fahrzeuge sowie stationäre Speicher beinhalten mehrere Lithium-Ionen Zellen. Abhängig von der Beanspruchung werden Zellen in Serie bzw. parallel geschaltet. Durch Vernetzung verschiedener Zellen können große Energiemengen gespeichert und bei Bedarf auch freigesetzt werden. Neben Sicherheitsmaßnahmen gegen Kurzschluss, Verpolung, Tiefentladung und Überhitzung soll der Ladevorgang durch einen Balancingvorgang unterstützt werden.

In dieser Arbeit wird der Prototyp eines Balancers für mehrere Zellen geplant und umgesetzt. Dieser Balancer soll in der Lage sein, neben Lithium-Ionen Akkumulatoren auch andere Zellentypen (Bleiakkumulatoren, etc.) zu laden. Im Falle eines stationären Speichers, der z.B. als Puffer in der Industrie dient, ist es erforderlich, dass mit hohen Strömen geladen werden kann, um die Ladedauer auf ein Minimum zu beschränken. Diese Ladestation soll als Prototyp das Laden bis zu 100 Ampere ermöglichen. Um ein Fehlverhalten der Zellen feststellen zu können, muss die Temperatur der einzelnen Zellen ständig überwacht werden. Deswegen wird jede Zelle mit einem Temperaturfühler ausgestattet. Um Transparenz zu schaffen, wird der gesamte Lade- und Balancingvorgang mitgeloggt, um später auf eventuell beschädigte Zellen rückschließen zu können. Da ein fehlerhaftes Verhalten des Systems nie ausgeschlossen werden kann, wird als Sicherheitsvorkehrung eine überwachende Stufe eingebaut, welche im Notfall in jeden Systemprozess eingreifen kann. Somit kann ein Überladen oder eine fehlerhafte Temperaturmessung technisch ausgeschlossen werden.

Der globale Trend der Vernetzung von Systemkomponenten zwingt Hersteller, ihre Produkte und deren Bedienung benutzerfreundlich und einfach zu gestalten. Wegen dieses Anspruchs wird der Prototyp über eine optische Darstellung aller Messwerte in Echtzeit mittels eines Touchscreens verfügen. Gespeichert werden die Messwerte auf einer MicroSD Karte im Sekundentakt. Da ein passives Balancing mit Wärmeabgabe einhergeht, wird das System aktiv von Lüftern gekühlt.

Das Ergebnis dieser Arbeit soll ein voll funktionsfähiges und sicheres Ladesystem sein, welches auch Messwerte aufzeichnet und durch eine übergeordnete Messebene gesichert wird.

Schlagwörter: Balancing, Lithium-Ionen, Messwerterfassung, Mikrocontroller

Abstract

Lithium-ion accumulators are the standard for electrochemical energy storage these days. In the mobility sector, in mobile devices and in stationary products they comprise a wide range of usage. Electric vehicles as well as stationary storage units contain several lithium-ion cells. Depending on their usage, cells are connected parallel or in series. By connecting different cells , a high amount of energy can be stored or released. Apart from safety precautions against shortcircuit, reverse polarity, deep discharge and overheating, charging is to be supported by a balancing process

In this thesis a balancing prototype is planned and implemented. This balancer should be capable of charging lithium-ion accumulators as well as other cell types (lead-acid accumulators, etc.). In case of stationary storage, like buffering storage in industry, you have to be capable of supplying a high amount of charging current to minimize charging time. This charge base prototype should make possible a charging current up to 100 ampere. The temperature of the individual cells must be constantly monitored in order to be able to detect malfunction of the cells. This is why every cellpack is fitted with a temperature sensor. Because of transparency, every data of the charging process is logged, which gives you the opportunity to detect defective cells later on. As malfunction of the system can never be precluded, a monitoring level is implemented as precautionary measure. It can intervene in any system process in case of emergency. This can eliminate false temperature measurement or overcharging technically.

The global trend to connect system components forces manufacturers to design their products user-friendly and simple. Because of this necessity, all measurements will be easily displayed on a touchscreen in realtime. The measurement data are saved on a microSD card every second. As passive balancing emits heat, the system is actively cooled by fans.

The result of this thesis should be a fully functional charging system, which records measuring data and is saved by a higher measuring level.

Keywords: balancing, lithium-ion, data acquisiton, microcontroller

Inhaltsverzeichnis

Seite

1 EINLEITUNG

Energie ist ein Begriff, der in der heutigen Zeit nicht mehr wegzudenken ist. Sie begleitet einen durch den Tag, einerseits bei der Zubereitung des Kaffees am Morgen, andererseits bei der Fahrt zum Arbeitsplatz bis zu der Nachtlampe, die einem das Lesen in der Dunkelheit ermöglicht. Energie wird abhängig von ihrem Nutzen von einer Energieform in eine andere umgewandelt. Für den häuslichen Gebrauch ist Strom eine wichtige Energiequelle, da es für fast jede Anwendung eine Technologie gibt, die Strom in den gewünschten Nutzen umwandelt. Die wachsende Industrie und der steigende Bevölkerungszuwachs sind unter anderem ausschlaggebend für den täglich größer werdenden Energiebedarf. Um diesen auch in Zukunft zu decken, werden Kraftwerke gebaut und es wird an neuen Technologien geforscht. Um möglichst ressourcensparend Energie zur Verfügung zu stellen, werden erneuerbare Energien in den Vordergrund gebracht. Neben Wind- und Wasserkraft werden Photovoltaik und Biomasseanlagen gebaut. Problematisch hierbei ist jedoch die Volatilität, die eine nicht konstante bzw. auch nicht bedarfs ausgerichtete Erzeugung mit sich zieht. Um diese Fluktuationen gering zu halten, werden Regelungen und Energiespeicher eingesetzt. In Österreich wird viel Energie in Pumpspeicherkraftwerken gespeichert, die bei Energieüberfluss Wasser auf ein höheres Niveau pumpen und bei Energiebedarf diese gewonnene potentielle Energie durch Ablassen über Turbinen wieder freigeben. Eine andere Möglichkeit bietet die Elektrolyse. Hierbei wird Wasser in seine Grundelemente, Wasserstoff und Sauerstoff aufgespaltet. In der Zukunft werden Akkumulatoren auch einen Teil der Energiespeicherung übernehmen. Einen großen Vorteil bietet die Tatsache, dass die Ladung eines Akkus sehr geringe Verluste aufweist und dieser hiermit einen großen Wirkungsgrad besitzt. Akkumulatoren sind zurzeit in fast jedem tragbaren elektronischen Gerät, als Speicher in manchen Haushalten und in Zukunft in großen Stückzahlen in der Automobilindustrie vertreten. Seit Anbeginn der Nutzung von Energie durch elektrochemische Vorgänge in Batterien oder Akkumulatoren wird nach neuen Technologien geforscht. Die weitverbreitetste Form ist die Speicherung und Nutzung von Energie auf Basis von Lithium-Ionen Zellen. Abhängig von ihrer chemischen Zellenzusammensetzung werden verschieden Spannungsniveaus und Stromstärken erreicht. Um höhere Spannungen zu erzeugen, werden Zellen in Serie-, um höhere Ströme zu erzeugen, parallel geschaltet. Da jede Zelle in der Produktion nach außen gleich aussieht, sind sie im Inneren jedoch nicht dieselben, was bedeutet, dass es zu Abweichungen der Zellstruktur im Inneren kommen kann. Das bedeutet, dass sich zusammengeschaltete Zellen beim Lade- bzw. Entladevorgang untereinander negativ beeinflussen können. Um diesem Problem entgegenzuwirken, werden in den meisten Fällen sogenannte Balancer verwendet, die die Zellenspannung der einzelnen Zellen dem Mittelwert aller Zellen angleichen, um ein einheitliches Spannungsniveau zu erhalten. Durch den Einsatz von Balancern werden einerseits die möglichen Ladezyklen erhöht und andererseits wird die Lebensdauer der einzelnen Zellen verlängert Die Umstellung auf erneuerbare Energie sowie die Tatsache, dass die Elektromobilität einen wichtigen Aspekt unserer Zukunft ausmachen wird, hat mich motiviert, einen Balancer nach meinen Vorstellungen zu planen, herzustellen und zu testen.

1.1 Aufgabenstellung

Dies Masterarbeit umfasst ein Balancing Modul, welches bis zu 6 Zellenpacks in Serie verarbeiten kann. Um im Nachhinein eine Auswertung zu ermöglichen, werden Daten wie Spannung, Temperatur und Stromstärke in einem File sekündlich mitgeloggt. Um eine rasche Ladung zu ermöglichen, soll die Einheit einen Ladestrom bis zu 100 Ampere bewältigen können. Um dem Benutzer eine Übersicht des Ladevorgangs zu geben, werden alle Daten auf einem Display dargestellt. Außerdem soll über eine Touchfunktion auch ein Eingreifen während des Betriebs ermöglicht werden.

Die Durchführung des Balancings wird einem Arduino Nano übertragen, dessen einzige Aufgabe es ist, die unterschiedlichen Balancingmodi auszuführen. Dieser “Singletask“ hat den Hintergrund, die volle Rechenleistung des Mikrocontrollers für diese Aufgabe zur Verfügung zu stellen.

Die Datenerfassung, aber auch die Festlegung, der für den Ladevorgang wichtigen Variablen (Ladeschlussspannung, Anzahl der Batterien, ...) sowie die Kommunikation zwischen den unterschiedlichen Modulen und auch die grafische Wiedergabe der Messwerte werden von einem „Raspberry Pi“ übernommen.

Sicherheit ist ein wichtiger Punkt, der wesentliche Beachtung erfordert. Um einem Ausfall des ausführenden System entgegenzusteuern, wird ein zweiter Arduino verwendet, welcher die Aufgabe besitzt, alle Messwerte zu vergleichen und im Notfall bei einer gewissen prozentuellen Überschreitung der maximalen Werte, alle Steuersignale des Balancing-Arduino auszuschalten. Diese Funktion soll mit Hilfe eines Netzwerkes, welches aus verschiedenen Logik-Gattern besteht, ermöglicht werden.

Um eine Kommunikation zwischen den technischen Einheiten ermöglichen zu können, wird die Kommunikation zwischen den Mikrocontrollern über die UART-Schnittstelle und zwischen den Analog-/ Digitalconvertern über den SPI-Bus geführt.

Damit jeder Mikrocontroller eigene Werte beziehen kann, werden zwei von drei mit Analog-/ Digitalconvertern verschaltet (10 Bit Auflösung). Der dritte Mikrocontroller bezieht seine Messwerte über die internen Analog-/ Digitaleingänge (ebenfalls 10 Bit Auflösung). Um richtige Messwerte bekommen zu können, wird als Vergleich immer eine Referenzspannung benötigt, welche über eine eigene Platine bereit gestellt werden soll.

Die für dieses Projekt notwendigen Arbeitsabschnitte werden auf verschiedene Platinen verteilt, anschließend verkabelt und in einem Gehäuse untergebracht. Dieses Gehäuse soll teilweise aus Plexiglas und auch aus 3D gedruckten Teilen bestehen. Um der durch das Balancing entstehenden Wärmeentwicklung entgegenzusteuern, sollen Axiallüfter verbaut werden. Aus praktikablen Gründen soll es nur eine einzige Stromversorgung geben, welche mittels Kaltgeräterstecker in Kombination mit einem Kippschalter zur Verfügung gestellt werden soll. Die für das Projekt notwendigen unterschiedlichen Spannungsniveaus sollen durch verschiedene AC/DC Netzteile bereitgestellt werden.

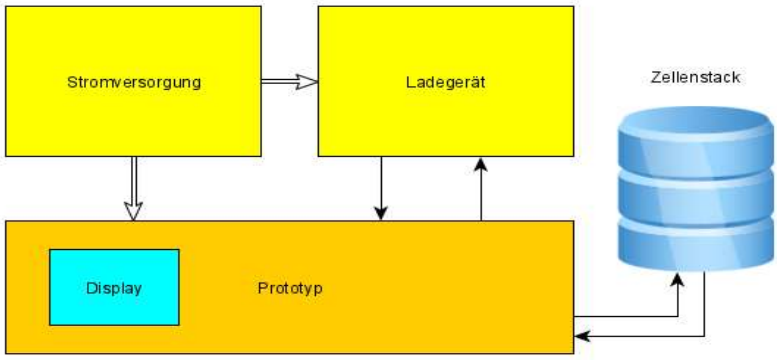


Abbildung 1-1: Blockschaltbild Funktionsweise extern

Wie in Abbildung 1-1 ersichtlich, werden das Ladegerät und der Prototyp von der Stromversorgung (230V) gespeist. Die Ladeströme werden durch zwei leistungsstarke Feldeffekttransistoren geschaltet und an den Zellenstack weitergeleitet. Die verbauten Balancingeinheiten des Prototyps werden mit den einzelnen Zellen verbunden, um einen geschlossenen Kreislauf zu ergeben.

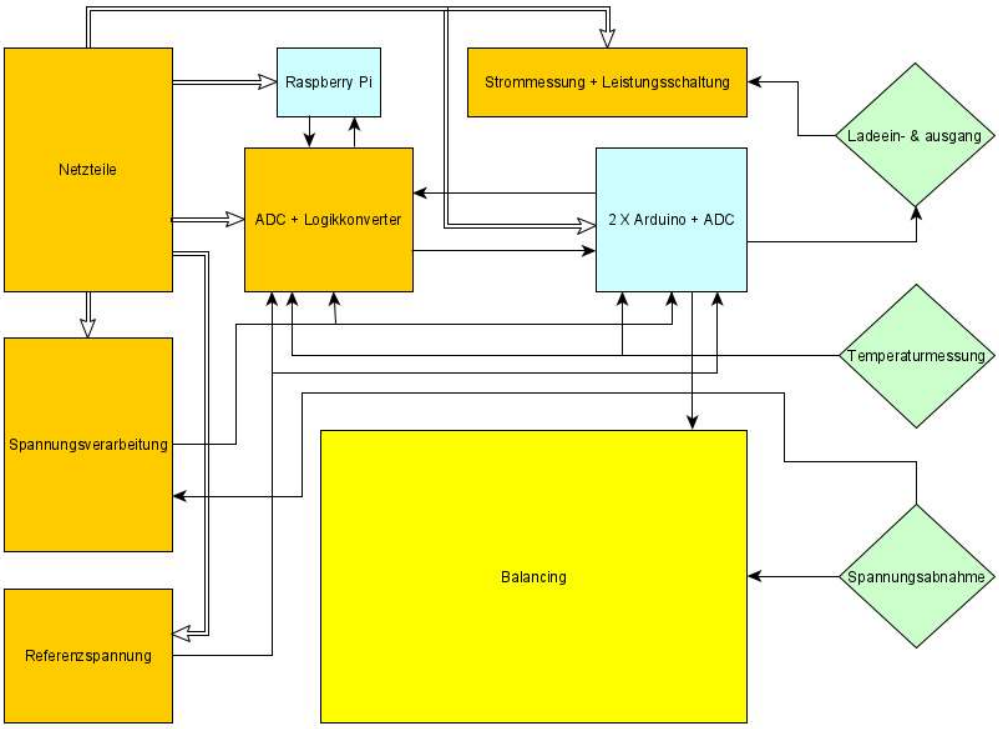


Abbildung 1-2: Blockschaltbild Funktionsweise intern

Wie in der Aufgabenstellung erklärt, beinhaltet der interne Aufbau des Projektes alle notwendigen Baugruppen. Die Balancingeinheit besteht aus sechs baugleichen Modulen, um eine schnelle Austauschmöglichkeit im Fehlerfall sicherzustellen. Die grün gefärbten Baugruppen führen extern zu dem Zellenpack oder dem Ladegerät. Die Pfeile mit doppelter Linienführung sollen Versorgungsleitungen darstellen. Die Pfeile mit einfacher Linienführung sollen den Datenfluss sichtbar machen.

2 Theoretischer Hintergrund

2.1 Galvanische Zelle

Eine galvanische Zelle ermöglicht die Umwandlung von chemischer Energie in elektrische Energie. Sie beinhaltet zwei unterschiedliche Metalle, die voneinander durch einen Elektrolyten getrennt sind. Da die Metalle verschiedene Zusammensetzungen aufweisen, ergibt sich aufgrund der elektrochemischen Spannungsreihe ein Potentialunterschied. Diesen Potentialunterschied kann man als Spannung zwischen den beiden Metallen, welche auch als Pole bezeichnet werden können, messen. Abhängig davon, welcher Pol von der positiven Messspitze eines Voltmeters erfasst wird, weist das Messgerät eine positive oder negative Spannung auf. Um den Stromkreis zu schließen, muss eine Last zwischen den beiden Polen angeschlossen werden (z.B. Widerstand, Lampe, ...). Ist die Last angeschlossen, wird am negativen Pol (Anode) eine Oxidation des Metalls stattfinden, welche dazu führt, Elektronen freizusetzen. Diese Elektronen wandern über den geschlossenen Kreis zum positiven Pol der (Kathode). Dort findet eine Reduktion statt. Um den Kreislauf zu schließen, müssen die oxidierten Ionen durch den Elektrolyten auf die andere Seite, wo die Reduktion stattfindet, wandern, um sich dort auszugleichen. Dieses System funktioniert deswegen, da nur die Ionen durch den Elektrolyten wandern können, nicht jedoch die im Oxidationsprozess bereitgestellten Elektronen.[1]

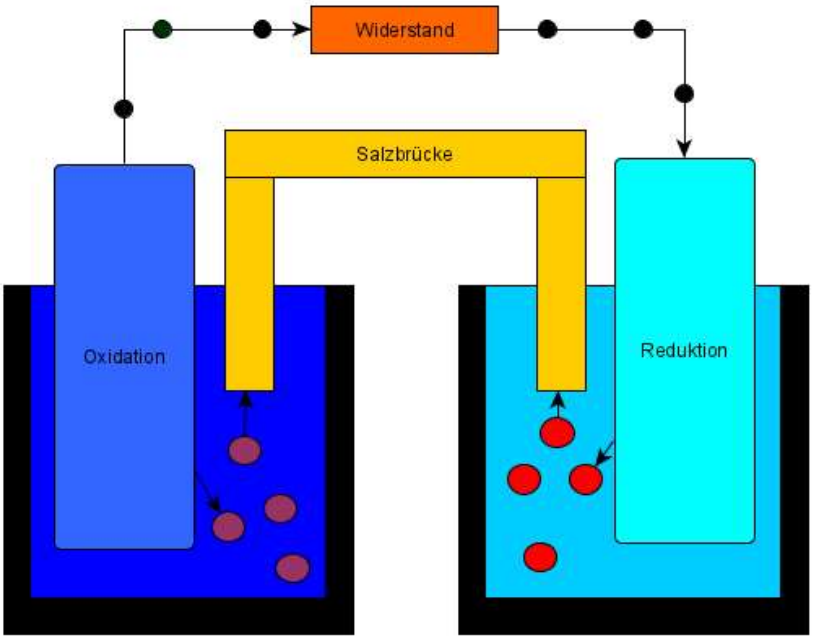


Abbildung 2-1: Daniel Element [1]

In Abbildung 2-1 wird das sogenannte Daniel Element gezeigt, welches eine spezielle Form der galvanischen Zelle darstellt. Hier sind beide Metalle von ihrer ionischen Lösung umgeben, jedoch voneinander räumlich getrennt. Die Elektronen wandern über den durch den Widerstand geschlossenen Stromkreis von Anode zur Kathode. Die durch die Oxidation gelösten Ionen wandern über eine Salzbrücke auf die andere Seite, wo sie sich mit den Ionen der Reduktion wieder vereinen. Durch diesen Prozess ist der Funktionskreis geschlossen.[1]

2.1.1 Zellenarten

Die Galvanische Zelle umfasst mehrere Zellentypen:

- **Primärzellen:** Bei dieser Art der Zellen, die auch unter dem Synonym Batterien bekannt sind, handelt es sich um einmalig verwendbare Zellen. Abhängig von ihrer Zellchemie werden diese Zellen aus unterschiedlichen Materialien zusammengesetzt und besitzen ihre volle Kapazität als fertiges Produkt. Der Ladezustand der Batterie nimmt mit zunehmender Betriebsdauer stetig ab. Schließlich ist der Punkt erreicht, an dem sich in der Zelle ein chemisches Gleichgewicht einstellt. Ist dies der Fall, ist kein Potentialunterschied erkennbar und die Zelle wird unbrauchbar. Batterien haben einen einmaligen Anwendungszeitraum und können nicht mehr wiederaufgeladen werden.[2]

- **Sekundärzellen:** Im Gegensatz zu den Primärzellen sind Sekundärzellen wieder aufladbar und können deswegen wiederverwendet werden. Die Anzahl der Lade- und Entladezyklen ist abhängig von der Zellenchemie, der Beanspruchungsart (starke oder schwache Ströme), dem Ladevorgang (Stromstärke), der Umgebung der sie ausgesetzt sind (Temperatur, Korrosion) und anderen Einflussgrößen. Der Ladevorgang ist genau gegengleich dem Entladevorgang. Hierbei wird elektrische Energie verwendet, um sie in chemische Energie umzuwandeln. Diese chemische Energie ist nun in der Zelle gespeichert und kann zu einem späteren Zeitpunkt genutzt werden. Zu beachten ist aber, dass eine Zelle sich auf Dauer selbst entlädt und dies im extremsten Fall zu ihrer Unbrauchbarkeit führen kann.[2]

- **Tertiärzellen:** Zellen dieser Art sind unter dem Begriff Brennstoffzelle bekannt. Generell ist das Funktionsprinzip ähnlich, mit dem Unterschied, dass die chemische Energie von außen zugeführt wird. Es gibt unterschiedliche Typen der Brennstoffzelle, die sich von ihren Reaktanden, von der Betriebstemperatur, ihren Elektronen, ihrer semipermeablen Membran und anderen Eigenschaften unterscheiden. Im Falle der Polymerelektrolytbrennstoffzelle wird als Brennstoff Wasserstoff verwendet, der in der Brennstoffzelle mit dem Luftsauerstoff reagiert. Da es sich hierbei um Gase handelt, werden als Elektroden Metalle wie z.B. Platin verwendet, welches bei der Reaktion als Katalysator fungiert. Der in das System eingebrachte Wasserstoff wird an der Anode zu Wasserstoffionen oxidiert, welche sich anschließend durch den Elektrolyten auf die Seite der Kathode begeben. Dort treffen sie mit durch Reduktion an der Kathode ionisiertem Luftsauerstoff zusammen und bilden als Reaktionsprodukt Wasser. Wie auch bei den anderen Zellentypen kann dies nur geschehen, sofern der Kreis durch z.B. eine Last, elektrisch verbunden ist. Da es sich hierbei um eine exotherme Reaktion handelt, wird bei der Reaktion Wärme freigesetzt, die bei üblichen Anwendungen nicht gebraucht wird, bzw. abgeführt werden muss, um die elektrische Leistung konstant zu halten. Brennstoffzellen werden zurzeit in der Mobilität als auch stationär in kleinen oder größeren Anwendungen verwendet.[2][3]

2.1.2 Akkumulatoren

Akkumulatoren werden in vielen verschiedenen Anwendungen eingesetzt. Im Gegensatz zu einer Batterie bieten sie die Möglichkeit, wieder aufgeladen zu werden. Abhängig vom Einsatzgebiet der Zellen werden verschiedene Eigenschaften verlangt. Für die meisten Anwendungen ist die Zellenspannung und deren Energieinhalt entscheidend. In speziellen Fällen wird jedoch auch viel Wert auf Sicherheit, Robustheit und das Verhalten bei unterschiedlichen Temperaturen gelegt. In Tabelle 1 wird ein Überblick der verschiedenen Eigenschaften gegeben.

Eigenschaften	Ni-Cd	Ni-MH	Li-ion
Nennspannung [V]	1,2	1,2	3,6
Temperaturbereich [°C]	-40 bis +50	0 bis +50	0 bis +50
Ladezyklen	1000	1000	1000
Überladen	Tolerierbar	Tolerierbar	Beschädigt Kathode
Energiedichte [Wh/kg]	50	100	200

Tabelle 1: Eigenschaften unterschiedlicher Akkumulatoren [4]

Für diese Arbeit wird ein Lithium-Ionen Akkumulator verwendet, welcher eine Ladeschlussspannung von 4.2 V und eine Ladungsmenge von 50 Ah pro Zelle besitzt. Diese Zellen werden zu einem Akkupack nach 6S2P verschaltet. Dieses Zellenpack ermöglicht eine konstante Stromentnahme von 180 Ampere. Um einen sicheren Ladevorgang zu gewährleisten, soll der Ladestrom 100 Ampere und die Ladetemperatur 40°C nicht überschreiten.

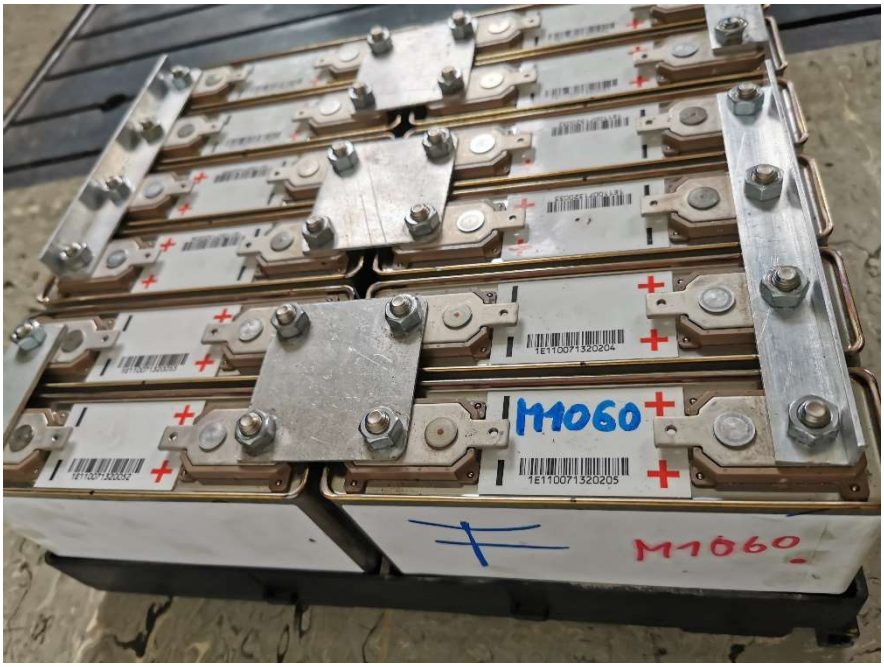


Abbildung 2-2: Zellenpack 6S2P

2.1.3 Gefahren

Sicherheit ist einer der wichtigsten Faktoren bei der Verwendung von Batterien oder Akkumulatoren. Die in der Zelle verbauten Chemikalien können bei gewissen Schadensfällen miteinander ungewollt reagieren und zu Erhitzung, Brand oder Explosion bis zur totalen Zerstörung der Zelle und deren naheliegendem Umfeld führen. Um ein Selbstverschulden ausschließen zu können, sollte man folgende Punkte beachten:

- **Mechanische Beschädigung:** Die Zellen sind abhängig von ihrer Bauform (zylindrische, prismatische oder Pouch Zelle) durch Krafteinwirkung von außen gefährdet. Jede Einwirkung von physischer Gewalt kann zu einem Schaden an der Zelle führen. Eine Beschädigung der inneren Zellform kann zu einer ungewollten Reaktion und somit zur Zerstörung der Zelle führen.[5]
- **Fehlendes Balancing:** Dieser Punkt bezieht sich nur auf den Ladevorgang von mehreren, in Serie zusammengehaltenen Zellen. Eine einfache Zelle braucht kein Balancing, weil der komplette Ladevorgang genau auf diese eine Zelle ausgerichtet ist. Wichtig ist jedoch, die Ladeschlusspannung einzuhalten sowie den Ladevorgang nicht mit einem Ladestrom zu betreiben, für den diese Zelle nicht geeignet ist. Bei verschalteten Zellen ist ein Balancing notwendig, da bei ungleicher Ladung das Gleichgewicht zwischen den Zellen nicht mehr gegeben ist. Dies führt dazu, dass die Zellen sich nach dem Ladevorgang untereinander beeinflussen, was zu einem Defekt von schwachen Zellen führen kann, was eine Kettenreaktion mit sich führen würde.[6]
- **Thermische Einwirkung:** Wie in Tabelle 1 angegeben, gibt es für die verschiedenen Zellentypen unterschiedliche Betriebstemperaturen. Wird die jeweilige Temperatur überschritten, führt dies einerseits zu einer Verkürzung der Lebensdauer als auch zu einem Brandrisiko. Ist die Temperatur unterhalb des Betriebsfensters, kann dies einen Einfluss auf den Elektrolyten nehmen, welcher dadurch weniger leitfähiger wird. Wird diese Zelle bei niedrigen Temperaturen verwendet, kann dies die Lebensdauer verringern.[6]
- **Kurzschluss:** Ein Kurzschluss bedeutet die direkte Verbindung der beiden Pole. Diese Verbindung bewirkt, abhängig vom Widerstand den die Verbindung aufweist, den maximal möglichen Strom den die Zelle liefern kann. Dieser Zustand belastet die komplette Zelle im höchsten Ausmaß. Es besteht hohe Brandgefahr.[6]
- **Tiefentladung:** Bei der Tiefentladung wird die Zelle unter die Entladeschlussspannung entladen. Abhängig vom Ausmaß der Tiefenentladung kann es einerseits zu einem dauerhaften Kapazitätsverlust bis hin zu internen Kurzschlüssen kommen, die einen Brand auslösen können.[6]
- **Überladen:** Beim Laden einer Zelle über ihre Ladeschlussspannung hinaus besteht einerseits die Gefahr der totalen Zerstörung der Zelle, andererseits wird im Falle der Überladung die Zelle intern stark erwärmt, was einen Brand zur Folge haben kann.[6]

Wie ersichtlich, ist die Handhabung von Lithium-Ionen Zellen gefährlich. In der heutigen Consumerelektronik werden in großem Rahmen Sicherheitselemente eingesetzt, um gegen z.B. Kurzschluss, Überladen und Tiefenentladung geschützt zu sein.

2.1.4 Lithium-Ionen Zellen

Die Verwendung von Lithium-Ionen Zellen ist in vielen mobilen Anwendungen Standard. Ein großer Vorteil, die diese Zellen liefern, ist die Nennspannung von ca. + 3.7 V. Diese Spannung ist um mehr als das Dreifache höher als eine Nickel-Cadmium Zelle (+1.2 V). Zu benachteiligen sind jedoch die Gefahren die so eine Zelle mit sich bringt. Lithium-Ionen Zellen sind, wenn sie in Brand geraten, schwer löschbar.

Der Lade- und Entladezyklus der Lithium-Ionen Zellen verhält sich anders als der Großteil der Primär- und Sekundärakkumulatoren. Der negative Pol (Anode) besteht aus Graphit und der positive Pol (Kathode) besteht aus Lithium in Kombination mit einem Metalloxid (z.B. $Li(NiCoMn)O_2$). Elektrolyte werden in den meisten Fällen in flüssiger Form in die Zelle eingebracht.[7][8]

Um die Zelle zu entladen, wird eine Verbindung zwischen den beiden Polen der Zelle durch einen Verbraucher hergestellt. Die einfachste Verbindung wäre eine Verbindung mit einem leitfähigen Material ohne Verbraucher. In diesem Fall spricht man von einem Kurzschluss. Die Besonderheit bei Lithium-Ionen Zellen ist die Tatsache, dass sich Lithium-Ionen in die Struktur des Kohlenstoffs einlagern können. Diesen Vorgang nennt man Interkalation.[7]

Im Falle eines Entladevorgangs wandern die im Kohlenstoff eingebundenen Lithium-Ionen durch den Elektrolyten auf die Seite der Kathode. Um ein elektrochemisches Gleichgewicht zu erreichen, werden zum Ausgleich Elektronen über den geschlossenen Stromkreis transportiert.[7]

Bei einem Ladevorgang wird diese Funktionsweise umgekehrt. Um die Lithium-Ionen dazu zu bringen, sich im Graphit einzubinden, muss von außen eine Spannung an den Zellen angelegt werden, um einen Stromfluss zu erzwingen, welcher den Kreislauf schließt.

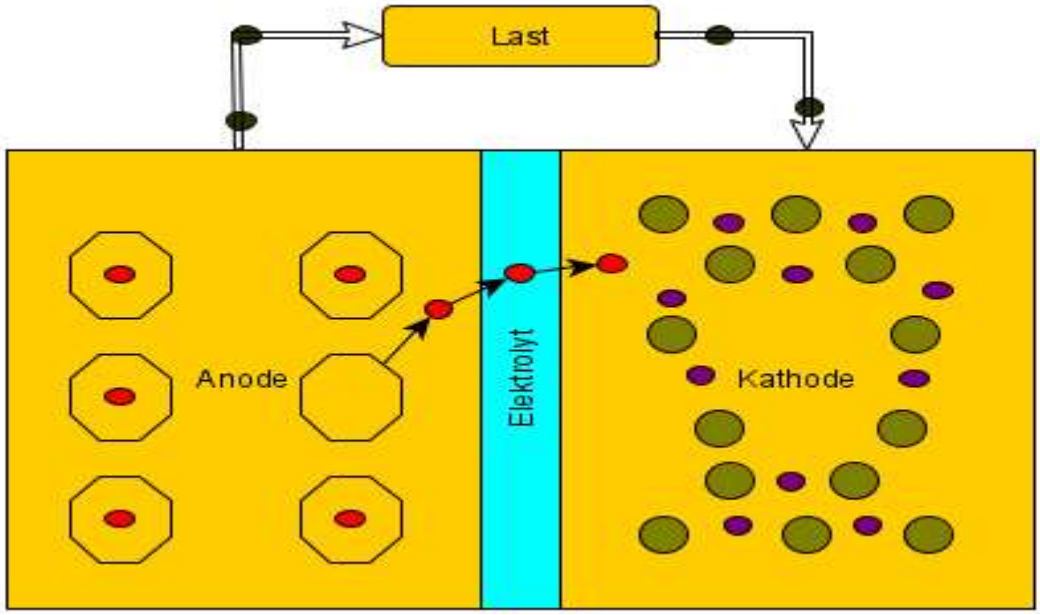


Abbildung 2-3: Entladevorgang Lithium-Ionen Zelle [7]

In Abbildung 2-3 wird der Entladevorgang einer Lithium-Ionen Zelle schematisch dargestellt. Die grünen und violetten Kreise stellen die Zusammensetzung des Kathodenmaterials dar. Auf der Anodenseite befindet sich die Graphitelektrode, die roten Punkte stellen die interkalierten Lithium-Ionen dar.

2.1.5 Ladevorgang

Um Lithium-Ionen Zellen zu laden, wird ein Netzteil an die beiden Pole der Zelle geschaltet. Abhängig von der anliegenden Zellenspannung wird mit Konstantstrom geladen, bis die Ladeschlussspannung der Zelle erreicht wird. Anschließend wird das Ladeverfahren mit konstanter Spannung fortgesetzt. Hierbei nimmt der Ladestrom so lange ab, bis keine Ladungsverschiebung mehr möglich ist. Tritt dieser Fall ein, ist die Zelle vollständig geladen.[9]

Um die Zelle nicht zu beschädigen müssen gewisse Kriterien eingehalten werden. Der Ladestrom sollte deutlich unter dem maximal möglichen Entladestrom liegen. Die Temperatur während des Ladevorganges darf typisch 40 °C nicht überschreiten, da es ansonsten zu einer Brandgefahr kommt. Das Netzteil muss richtig gepolt an die Zelle angeschlossen werden, um eine Verpolung zu verhindern. Bei Verpolung kann es einerseits zu Schäden an der Zelle kommen, andererseits kann die Sicherung des Netzteiles durchbrennen oder die ausgangsseitige Verpolungsschutz- und Serienschaltungsdiode zerstört werden sowie die gesamte ausgangsseitige Elektronik. Es muss außerdem beachtet werden, dass keine leitenden Elemente in der Nähe sind, die ungewollt die Kontakte berühren könnten und dadurch die Pole kurzschließen können.[10]

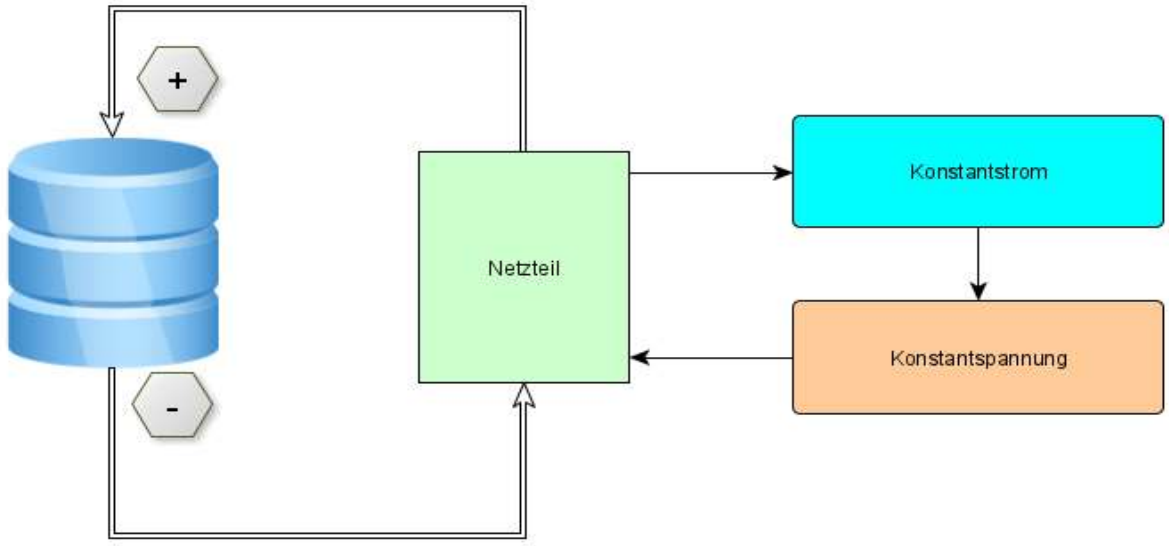


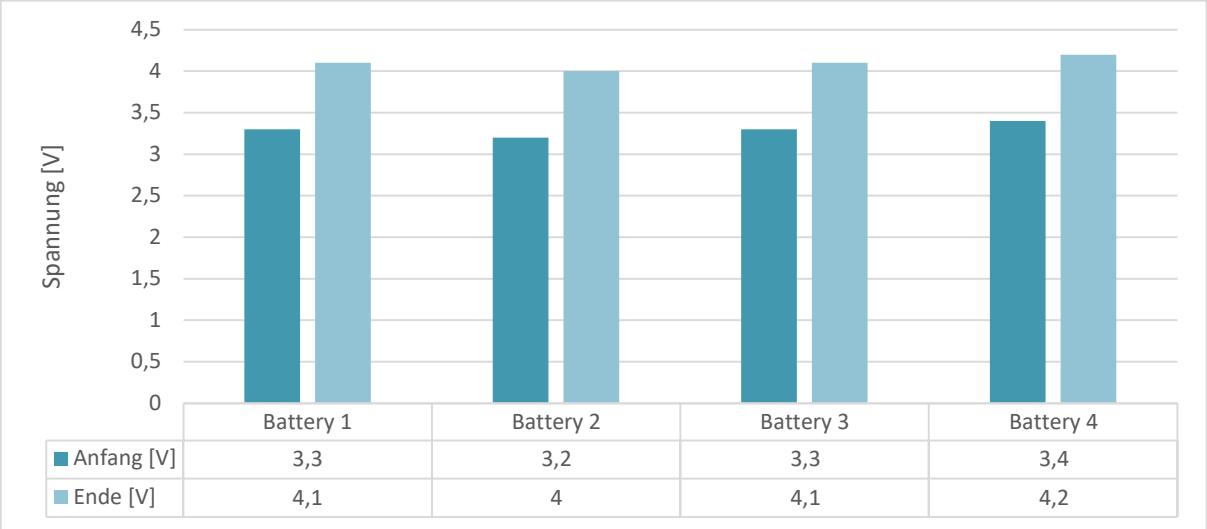
Abbildung 2-4: Ladevorgang Lithium-Ionen Zelle

Um die richtigen Einstellungen für den Ladevorgang zu wählen, muss im Datenblatt des verwendeten Zelltyps nachgesehen werden. Wurde eine Zelle tiefentladen, muss beim Ladevorgang auf das Ladeverhalten Acht gegeben werden. Aus sicherheitstechnischen Gründen soll der Ladevorgang nicht unbeaufsichtigt erfolgen. Um die Temperatur der Zellen einzuschätzen, kann man auch die Temperaturwerte mit einer Wärmebildkamera überwachen.

2.2 Balancing

Wie unter 2.1.3 beschrieben, sind viele Punkte zu beachten, wenn es um den sicheren Umgang mit Lithium-Ionen Akkumulatoren geht. Das Thema Balancing ist wichtig für Zellen, die in Serie geschaltet sind. Da eine Zelle nie komplett einer anderen Zelle gleicht, bedeutet dies, dass es Unterschiede im Ladezustand, dem zellinternen Widerstand, der möglichen Kapazitäten und vieles mehr gibt. Durch den Zusammenschluss mehrere Zellen beeinflussen sich die Zellen untereinander. In einem Beispiel werden Ladungen zum Ausgleich der Kapazitäten untereinander verschoben. Dies geht mit der Tatsache einher, dass jedes System versucht, sein chemisches Gleichgewicht zu erreichen.

2.2.1 Laden ohne Balancing



	Battery 1	Battery 2	Battery 3	Battery 4
Anfang [V]	3,3	3,2	3,3	3,4
Ende [V]	4,1	4	4,1	4,2

Abbildung 2-5: Laden ohne Balancing [11]

In Abbildung 2-5 wird der Ladevorgang von 4 in Serie geschalteten Zellen dargestellt unter der Annahme, dass die Spannung jeder Zelle überwacht wird und im Falle des Erreichens der Ladeschlussspannung der Ladevorgang beendet wird. Außerdem werden die einzelnen Zellen bis auf die unterschiedliche Energiespeicherkapazität als ideal angenommen (gleicher Innenwiderstand, Ladeverhalten, ...).

Wie ersichtlich, sind am Ende des Ladevorgangs die Zellenspannungen ungleich verteilt. Das bedeutet, dass die Summe der Spannungen (16,4 V) nicht den gewünschten Wert (16.8 V) erreicht. Im Allgemeinen bedeutet dies, dass die in den Zellen gespeicherte Energie unter dem erwarteten Wert liegt.

Um den Energieinhalt zu maximieren, wird ein Balancing, entweder aktiv oder passiv, eingesetzt. Der Wirkungsgrad sinkt beim Ladevorgang, weil bei beiden Methoden Energie dissipiert wird. Bei einem passiven Balancing wird die Energie zu Wärme an den Lastwiderständen umgewandelt. Beim aktiven Balancing wird zwar überschüssige Energie von einer Zelle zu einer schwächeren Zelle umgeleitet, doch diese Umverteilung bringt auch Leitungs- und Schaltverluste mit sich und ist durch die höhere Komponentenzahl weniger zuverlässig.

2.2.2 Laden mit Balancing

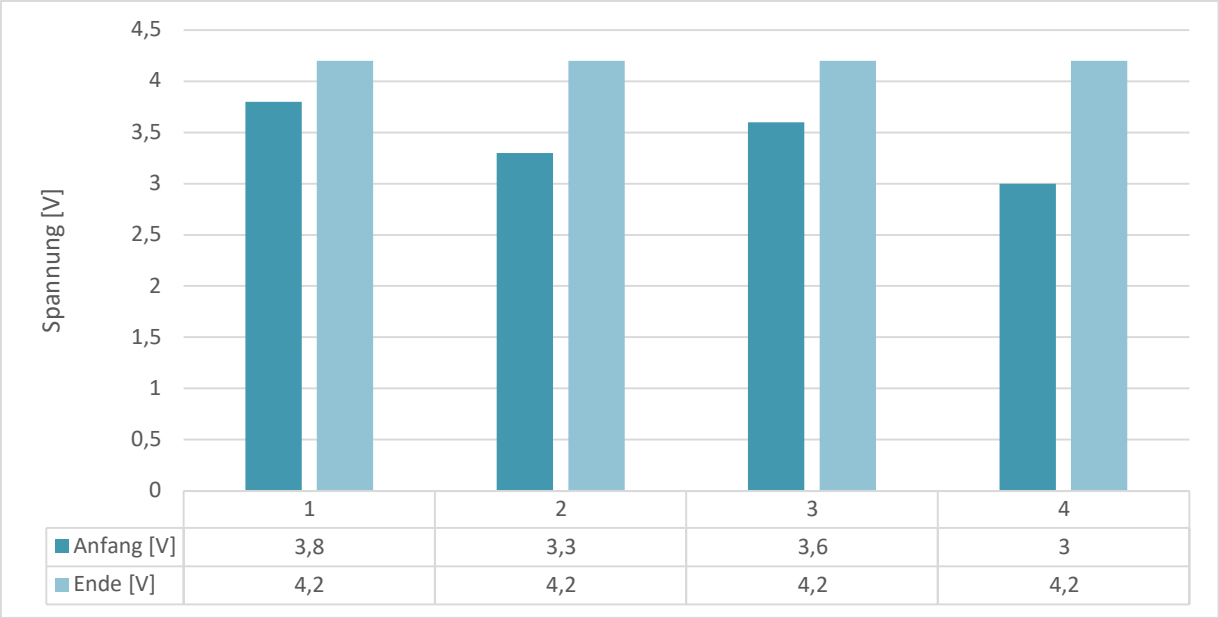


Abbildung 2-6: Laden mit Balancing

In Abbildung 2-6 sind die einzelnen Zellenspannungen am Anfang stark voneinander abweichend. Die Zellen werden auch in diesem Fall als ideal angesehen. Wie ersichtlich, sind am Ende des Ladevorganges alle Spannungen auf demselben Niveau. Es wird nun auch die errechnete Summenspannung von (16.8 V) erreicht. Dieses Ergebnis ist sowohl beim aktiven als auch beim passiven Balancing erreichbar. Abhängig vom Einsatzgebiet der Zellenpacks werden unterschiedliche Balancingmethoden verwendet, da das aktive Balancing zwar effizienter, aber demensprechend auch teurer ist.[9]

2.2.3 Aktives Balancing

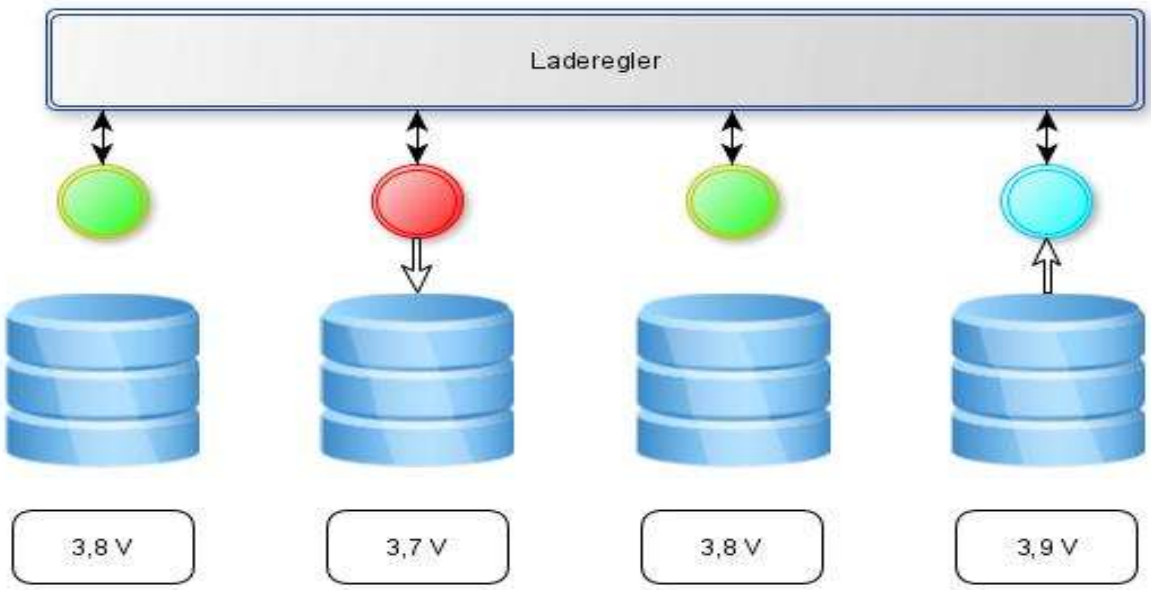


Abbildung 2-7: Ladevorgang mit aktivem Balancing

Ein aktives Balancingsystem besteht aus Laderegler, Spannungsauswertung, Leistungselektronik und eventuell auch einer Stromauswertung. Die Spannungen der einzelnen Zellen werden von der Steuereinheit überwacht. Sofern eine Differenz auftritt, wird Energie von einer stärkeren Zelle auf eine schwächere Zelle übertragen. Dies geschieht mit Hilfe eins Ladereglers und einer zwischen den Zellen angeordneten Leistungsschaltung, die bei Bedarf Verbindungen zwischen den Zellen freischaltet, um einen Stromfluss zu gewährleisten. Der Vorteil bei einem aktiven Balancing liegt in seiner Effizienz. Das System ist zwar verlustbehaftet (Leitungsverluste, Schaltverluste), jedoch sind die Verluste im Gegensatz zu einem passiven Balancing vernachlässigbar. Dieses Verfahren ist aufgrund seiner Komplexität mit höheren Kosten und höherem Betriebsrisiko verbunden.

2.2.4 Passives Balancing

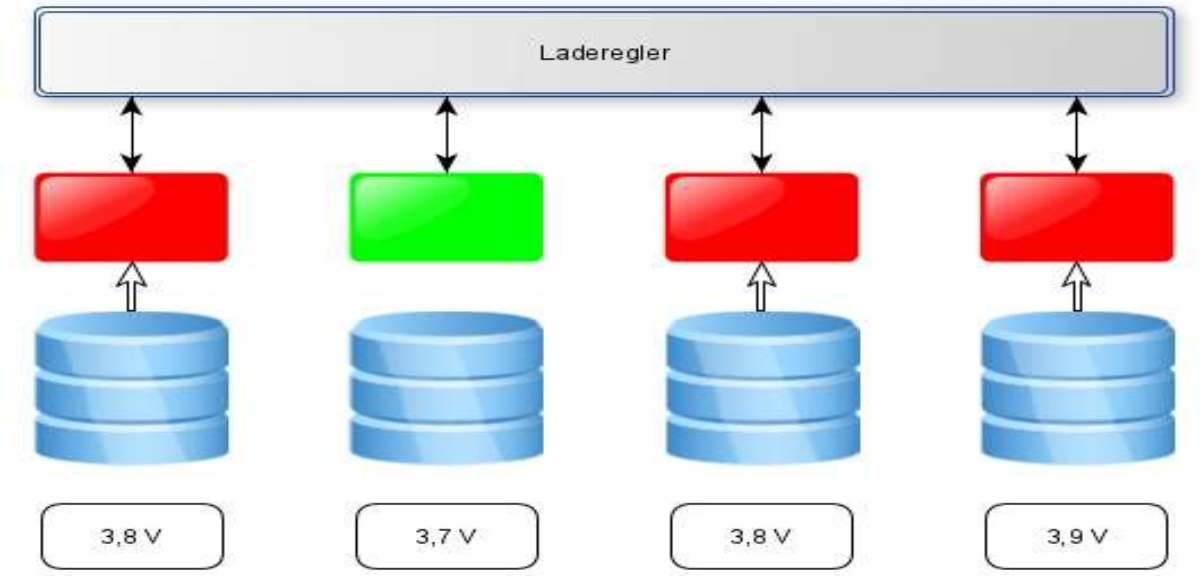


Abbildung 2-8: Ladevorgang mit passivem Balancing

Im Gegensatz zum aktiven Balancing wird Energie nicht umverteilt, sondern dissipiert. Das System besteht aus einem Laderegler, einer Spannungsüberwachung und einer Leistungselektronik, die Lastwiderstände beinhaltet. Der Laderegler überprüft die einzelnen Zellenspannungen und vergleicht sie miteinander. Zellen, die eine höhere Spannung aufweisen als die Zelle mit der niedrigsten Spannung, werden gebalanced. Hierbei wird den beiden Polen der betroffenen Zelle ein Lastwiderstand parallelgeschaltet. Nun fließt über diesen Widerstand ein Strom, der aus der Zelle entnommen wird. Dadurch sinkt die Zellenspannung so lange bis diese das Spannungsniveau der schwächsten Zelle erreicht. Anschließend wird der Widerstand wieder weggeschaltet. Wie in Abbildung 2-8 ersichtlich, wird drei von vier Zellen Strom entzogen. Um den Akkumulator voll zu laden, ist dementsprechend auch mehr Energie von Nöten. Die Dauer des Ladevorganges wird einerseits durch den Ladestrom und andererseits durch die Höhe des Entladestroms der einzelnen Zellen begrenzt. Das Ladeverhalten kann auf drei unterschiedliche Arten durchgeführt werden.

2.2.4.1 Bottom Balancing

Bei diesem Verfahren wird der Zellenpack an den Laderegler angeschlossen und die einzelnen Spannungen werden gemessen. Alle Zellen werden auf das Spannungsniveau der schlechtesten Zelle reduziert. Ist dieser Vorgang abgeschlossen, kann der Ladevorgang gestartet werden. Der Laderegler lädt das Zellenpack bis die Ladeschlussspannung erreicht wird.

2.2.4.2 Top Balancing

Hierbei wird der Anschluss gleich wie beim Bottom Balancing durchgeführt, jedoch wird der Balancingvorgang nicht gestartet. Befinden sich die einzelnen Zellenspannungen unter der Ladeschlussspannung, wird der Ladevorgang gestartet. Es wird so lange geladen, bis eine der Zellen die Ladeschlussspannung erreicht hat. Anschließend wird der Ladevorgang pausiert und das Balancing aktiviert. Sind durch das Balancing die Zellenspannungen angeglichen, wird der Ladevorgang fortgesetzt und der Kreislauf beginnt von Neuem. Der Ladevorgang gilt als abgeschlossen, wenn alle Zellenspannung denselben Wert aufweisen und die Ladeschlussspannung erreicht wurde.

2.2.4.3 Dauerhaftes Balancing

Das dauerhafte Balancing lädt und balananced zur selben Zeit. Dies hat den Vorteil, dass man Zeit spart. Problematisch hierbei ist jedoch, dass durch den Ladestrom die Zellenspannungen verfälscht werden, was dazu führt, dass die gemessen Werte von den realen Werten abschweifen. Um dies zu kompensieren, kann man für kurze Zeit den Ladevorgang unterbrechen, damit sich die einzelnen Zellen auf ihr wirkliches Spannungsniveau herabsetzen können. Der Ladevorgang wird bei den selben Kriterien wie beim Top Balancing beendet.

2.2.4.4 Systemkomponenten

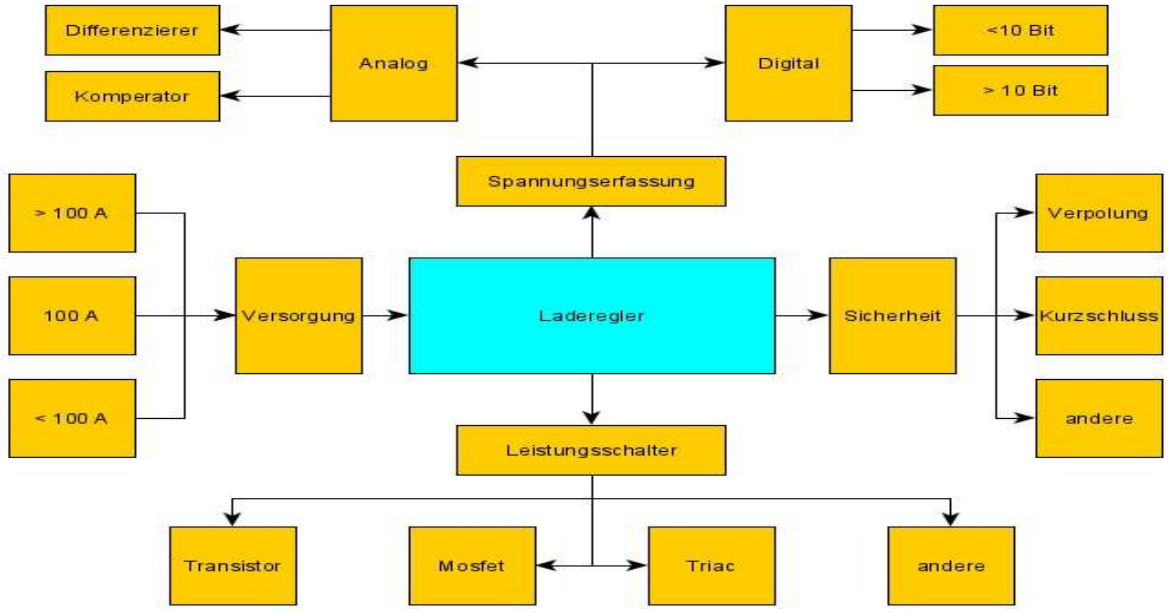


Abbildung 2-9: Systemkomponenten

3 Hardware

3.1 Operationsverstärker

Ein Operationsverstärker ist ein aktives Bauelement, das von großer Wichtigkeit ist, da es einen großen Einsatzbereich hat. Der Operationsverstärker hat die Möglichkeit, anliegende Signale in verstärkter oder abgeschwächter Form weiterzugeben. Seine Verstärkungsleistung ist abhängig von seiner Beschaltung und seiner Versorgung. Er kann als Differenzierer, Integrierer, Summierer, etc. dienen.

3.1.1 Spannungsfolger

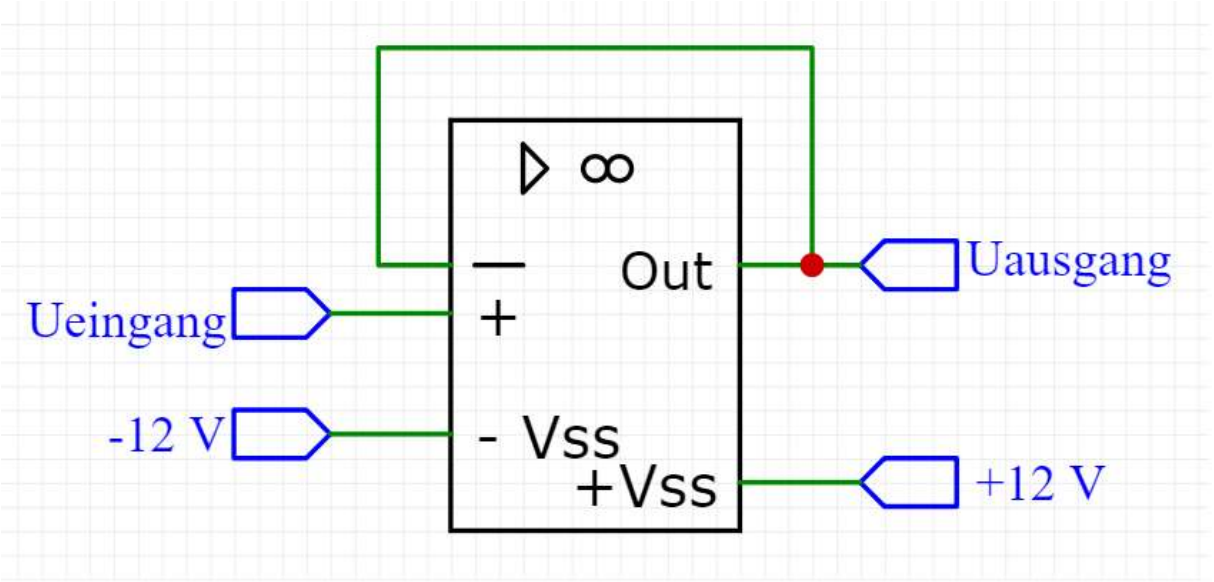


Abbildung 3-1: Spannungsfolger [12]

Ein Spannungsfolger hat die Verstärkung von 1. Das bedeutet, dass seine Ausgangsspannung gleich der Eingangsspannung ist unter der Bedingung, dass die Versorgungsspannung größer ist als die Eingangsspannung plus dem Eigenverbrauch, den der Operationsverstärker aufweist. Diese Verschaltung wird benutzt, wenn z.B. ein nicht belastbares Eingangssignal vorliegt, welches bei geringster Belastung einbrechen würde bzw. sich stark verringern würde. Im idealen Fall besitzt ein Operationsverstärker einen unendlich hohen Eingangswiderstand. In der Realität ist diese Umsetzung unmöglich, jedoch ist dieser Widerstand sehr hoch. Dies bedeutet, dass die Eingangsspannung sehr wenig Strom liefern muss und daher entsprechend gering belastet wird. Der Spannungsausgang ist durch die Versorgung des Operationsverstärkers gestützt und kann deswegen auch für stärkere Anwendungen genützt werden. Da Operationsverstärker für verschiedene Bereiche und von verschiedenen Firmen hergestellt werden, ist es ratsam, sich das passende Datenblatt dazu anzusehen. Datenblätter beinhalten alle wichtigen Eckdaten (maximal- /minimal Ratings), Testversuche und meistens auch einen Vorschlag für dessen Beschaltung, damit das Bauteil sinngemäß funktioniert. Abhängig von der Anwendung werden unterschiedliche Toleranzen gefordert, welche den Preis eines Bauteiles bestimmen können. Je genauer desto teurer, ist die Devise.[12]

3.1.2 Differenzverstärker

Für dieses Projekt ist der Differenzverstärker von großer Wichtigkeit. Da die meisten Spannungsmessungen auf Masse bezogen sind, würde das bei mehreren Zellen zu einem Problem führen, da dann nicht mehr die einzelnen Zellenspannungen ausgegeben werden würden, sondern die Summe aller bis zum Messpunkt zusammengeschalteten Zellen. Um dieses Problem zu lösen, wird ein Differenzverstärker eingesetzt, welcher als negativen Eingang dem negativen Pol der Zelle und als positiven Eingang dem positiven Pol der Zelle zugeschaltet wird. Dies ergibt, abhängig von den gewählten Widerständen, ein positives, um den Faktor des Verhältnisses der Widerstände, verstärktes bzw. abgeschwächtes Signal.[13]

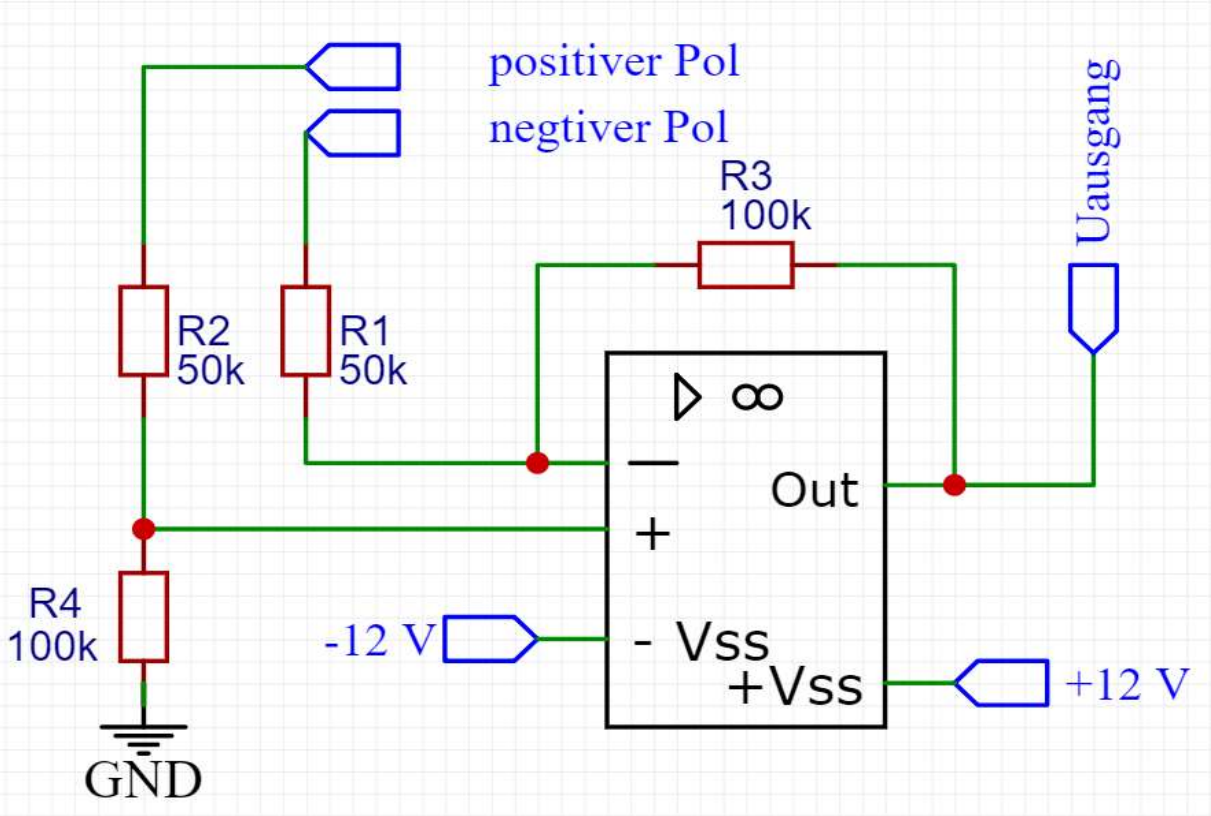


Abbildung 3-2: Differenzverstärker [12]

Der Ausgangsspannung wird folgendermaßen berechnet:

$$U_{ausgang} = (U_{positiv} - U_{negativ}) * \frac{R_3}{R_1} \quad (1)$$

In Abbildung 3-2 ist R3 mit 100 kOhm und R1 mit 50 kOhm festgelegt. Weiters ist R2 = R1 und R4 = R3. Das ergibt laut Formel (1) bei einer positiven Eingangsspannung von 1 V und einer negativen Eingangsspannung von 0 V eine Ausgangsspannung von 2 V. Da die meisten Analog-/Digitalconverter eine maximale Eingangsspannung besitzen, die nicht überschritten werden darf (Beschädigung des Bauelementes), ist eine Differenzverstärkungsschaltung von Vorteil, da man die Ausgangsspannung um einen festgelegten Faktor verkleinern kann. Somit überschreitet der Analog-/Digitalconverter seine maximale Eingangsspannung nicht.

3.2 Logik Gatter

In der Welt der Digitaltechnik werden häufig sogenannte Logik Gatter eingesetzt. Ihre Aufgabe liegt darin, Eingangssignale (logisch high oder low) nach einer gewissen Logik (und, oder, etc.) auszugeben. Mit diesen Bauelementen kann man auf Hardwareebene gewisse Ereignisse steuern.

3.2.1 Und Gatter

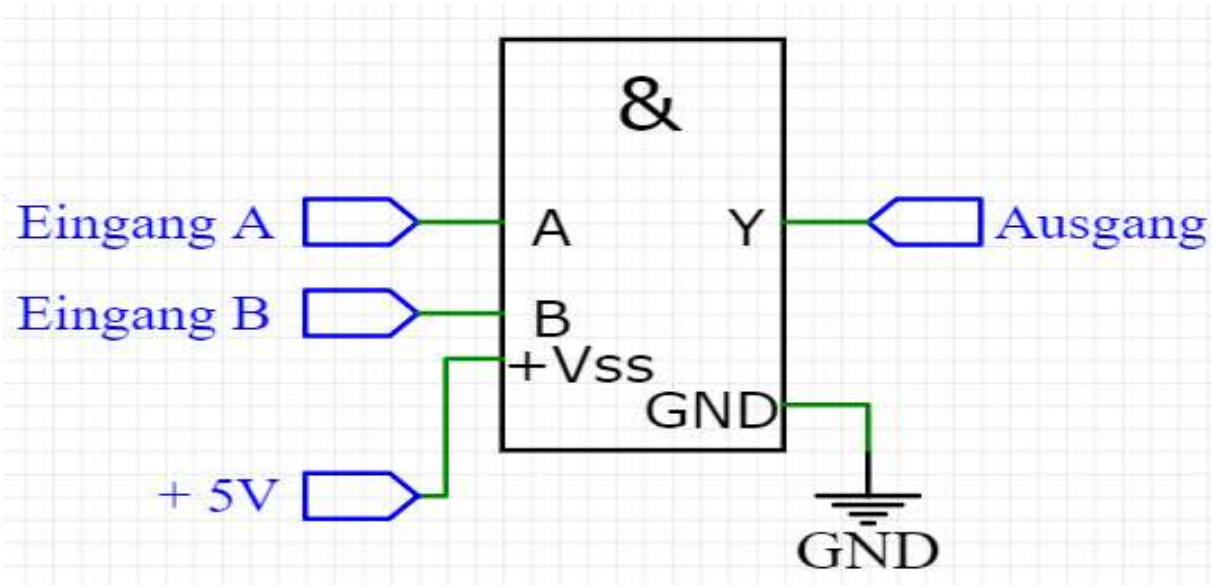


Abbildung 3-3: Ligik Gatter (AND)

Das Und Gatter verfügt in den meisten Fällen über eine Versorgung von + 5 V und einen Masseanschluss. Die zwei logischen Eingänge A&B werden anschließend mit einer & Logik verknüpft und das Ergebnis wird dann am Ausgang ausgegeben. Abhängig von der Bauform und der Herstellertype werden mehrere von diesen logischen Eingängen in einem Bauelement verbaut. Diese Logik Gatter haben im Normalfall eine sehr schnelle Schaltgeschwindigkeit (>10 Mhz) und es können auch geringe Ströme am Ausgang entnommen werden (~ 10 mA). Der Ausgang des Gatters folgt dieser Logik:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 2: Wahrheitstabelle Und Gatter [14]

3.2.2 Oder Gatter

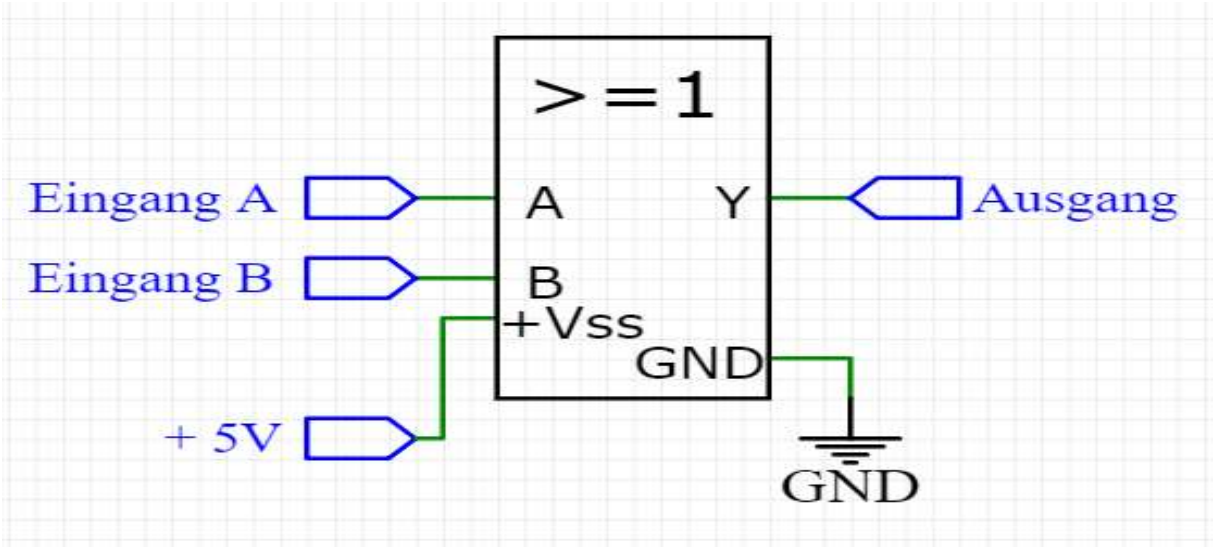


Abbildung 3-4: Oder Gatter

Das Oder Gatter ist von der äußeren Beschaltung gleich den Und Gattern. Sie unterscheiden sich nur in der internen Verschaltung. Man unterscheidet logische Bauelemente auch anhand eines negativ geschalteten Ausgangs, sogenannte NOR oder NAND, was so viel bedeutet wie “negativ or“ bzw. “negativ and“. In diesem Fall wird genau das Gegenteil von dem ausgegeben, was in der unten angeführten Wahrheitstabelle ersichtlich ist.

A	B	Y	$\bar{Y}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Tabelle 3: Logik Gatter Oder [14]

In dieser Arbeit wird eine größere Anzahl an Und Gattern verbaut, da es dem Arduino Mikrocontroller, der für die Sicherheit zuständig ist, die Möglichkeit gibt, in allen Steuerelementen des Balancing Arduinos einzugreifen. Das liegt daran, dass die meisten Ausgänge des Arduinos mit den Und Gattern verbunden sind. Somit kann bei einem Fehlverhalten des Balancing Arduinos der Sicherheits Arduino eingreifen, um das System zu stoppen.

3.3 Analog Digital Wandler

In der Digitaltechnik gibt es nur zwei mögliche Zustände: HIGH und LOW, was einer logischen 1 bzw. 0 entspricht. Um ein analoges Signal in ein digitales umzuwandeln, werden sogenannte Analog-/ Digitalconverter verwendet. Ihre Aufgabe ist es, das analoge Signal durch unterschiedliche Verfahren in ein digitales umzuwandeln. Einer der wichtigsten Punkte bei der Auswahl des Analog-/ Digitalconverters ist seine Auflösung in Bit. Die Auflösung ergibt die Aufspaltung des Messwertes nach Formel (2).[14]

$$Schritte = 2^{n_{Bit}} \tag{2}$$

Ein 10 Bit Analog-/ Digitalconverter besitzt also 1024 Schritte. Mit Formel (3) kann nun die Genauigkeit für eine Referenzspannung von +5 V berechnet werden.

$$Genauigkeit = \frac{5}{1024} = 0{,}00488\ [V] \tag{3}$$

Die Genauigkeit gibt an, wie nahe der digitale Bitwert dem analogen Signal kommt. Je höher die Auflösung, desto genauer ist der erfasste Wert an der Realität. Für dieses Projekt werden zwei unterschiedliche Analog- / Digitalconverter verwendet, welche in beiden Fällen eine 10 Bit Auflösung besitzen.

3.3.1 MCP 3008

Dieses Bauteil ist ein Analog-/ Digitalconverter mit einer 10 Bit Auflösung und dem SPI-Bus als Kommunikationsschnittstelle. Es besitzt 8 analoge Eingänge, von denen jeder einzeln oder in Zweierpärchen als Differenzschaltung eingelesen werden kann.

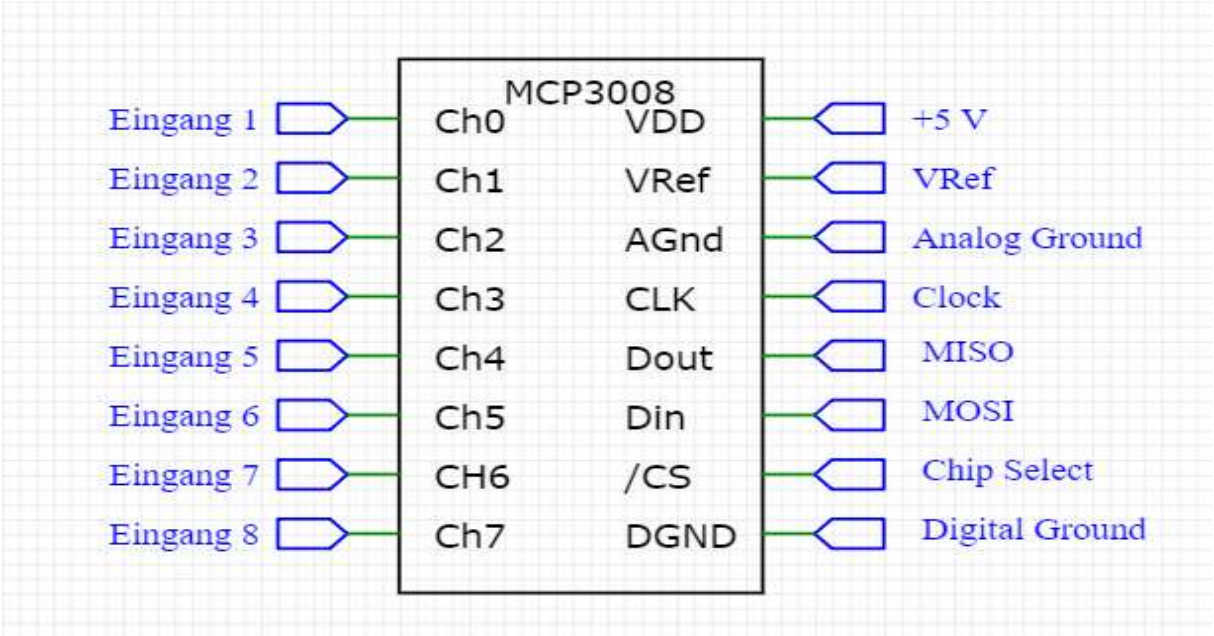


Abbildung 3-5: Darstellung MCP 3008 [15]

3.3.2 Onchip Arduino

Der verwendete Mikrocontroller, der für das Balancing zuständig ist, enthält eigene Analog-/ Digitalconverter. Der Mikrocontroller des Typs Arduino-Nano besitzt 8 analoge Eingänge (A0 – A7). Die Auflösung beträgt 10 Bit. Bei diesem Modell hat man die Möglichkeit, eine externe Referenzspannung oder unterschiedliche interne Referenzspannungen zu wählen. Die Auswahl wird über eine Softwareeinstellung “analogReference()“ festgelegt.

3.3.3 Multiplexing

Oft kommt es vor, dass man viele Ausgänge zu schalten hat, um sein Projekt umzusetzen. Die meisten Mikrocontroller besitzen eine begrenzte Anzahl von schaltbaren Ausgängen bzw. eine begrenzte Anzahl von erfassenden Eingängen. Um diese Anzahl zu erhöhen, werden sogenannte Multiplexingbausteine verwendet. Ihre Aufgabe ist es, eine gewisse Anzahl von „neuen“ Eingängen zur Verfügung zu stellen. Diese Eingänge werden auf einen oder mehrere Ausgänge nach einer gewissen Logik verschaltet. Die Anzahl der Eingänge bzw. Ausgänge ist abhängig vom gewählten Multiplexer. Geschalten wird über Schaltleitungen, die abhängig von der Ansteuerung, einen gewissen Eingang auf den Ausgang legen. In den meisten Fällen ist eine bidirektionale Kommunikation möglich.

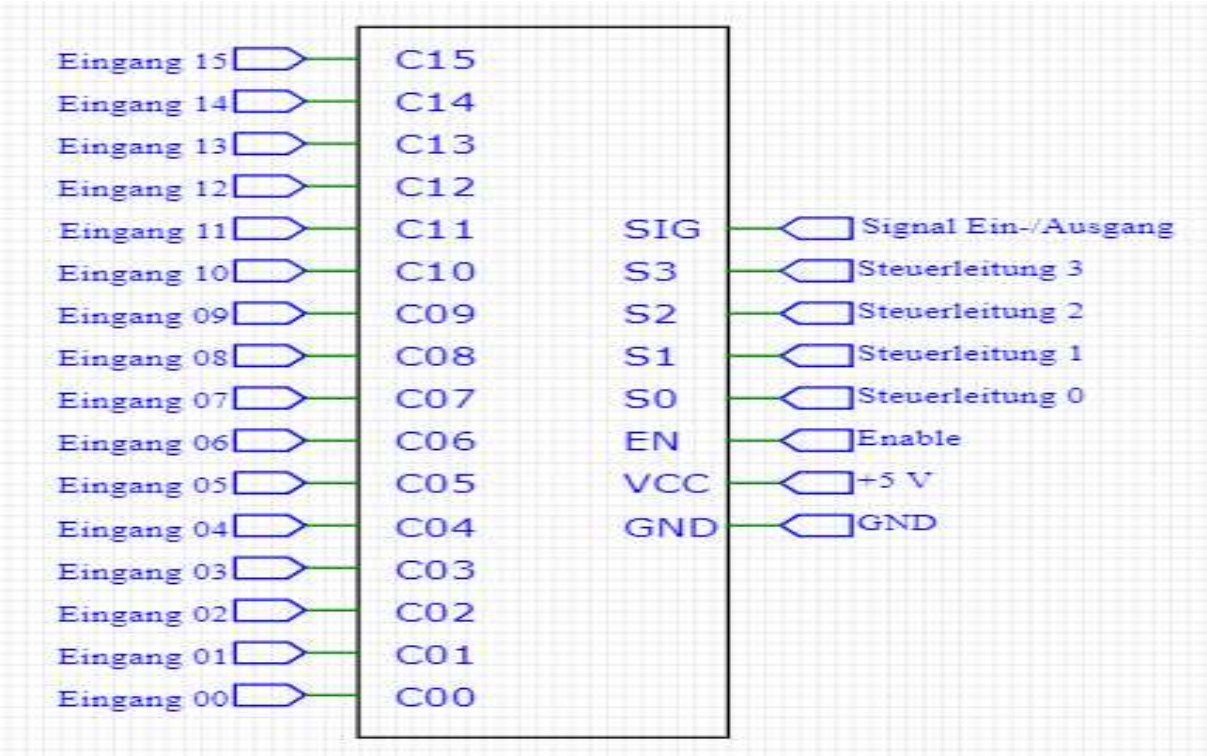


Abbildung 3-6: Multiplexer 16x [16]

Um das Schaltverhalten zu verstehen, wird in Tabelle 4 ein Überblick über alle möglichen Schaltzustände gegeben.

S0	S1	S2	S3	Ausgang
0	0	0	0	C00
1	0	0	0	C01
0	1	0	0	C02
1	1	0	0	C03

Tabelle 4: Schaltkombinationen [16]

Tabelle 4 zeigt nur die Schaltkombinationen für S0 und S1, da diese für dieses Projekt notwendig waren. Um alle 16 Eingänge zu schalten, sind auch S2 und S3 notwendig. Um die gesetzten Eingänge gültig zu schalten, ist der “Enable“ Pin auf High zu setzen. Um keine schwebenden Verbindungen zu haben, müssen auf der Schaltseite alle Eingänge, die nicht benutzt werden, auf Masse gezogen werden, um ein Fehlverhalten des Bauteils ausschließen zu können.

3.3.4 Logik-Wandler

Diese Bauelemente sind erforderlich, wenn die Schaltspannungen der einzelnen Teilnehmer unterschiedlich hoch sind. In dieser Arbeit werden Mikrocontroller der Type Arduino Nano und ein Raspberry Pi verbaut. Der Arduino besitzt ein Logiklevel von +5V. Der Raspberry Pi hingegen unterstützt nur Spannungen bis + 3.3 V. Um eine Kommunikation zwischen den beiden Systemen über die GPIO (General purpose input/ output) herzustellen, ist ein Logik-Wandler notwendig. Seine Aufgabe besteht darin, die logischen Spannungswerte an die angeschlossenen Geräte anzugleichen.

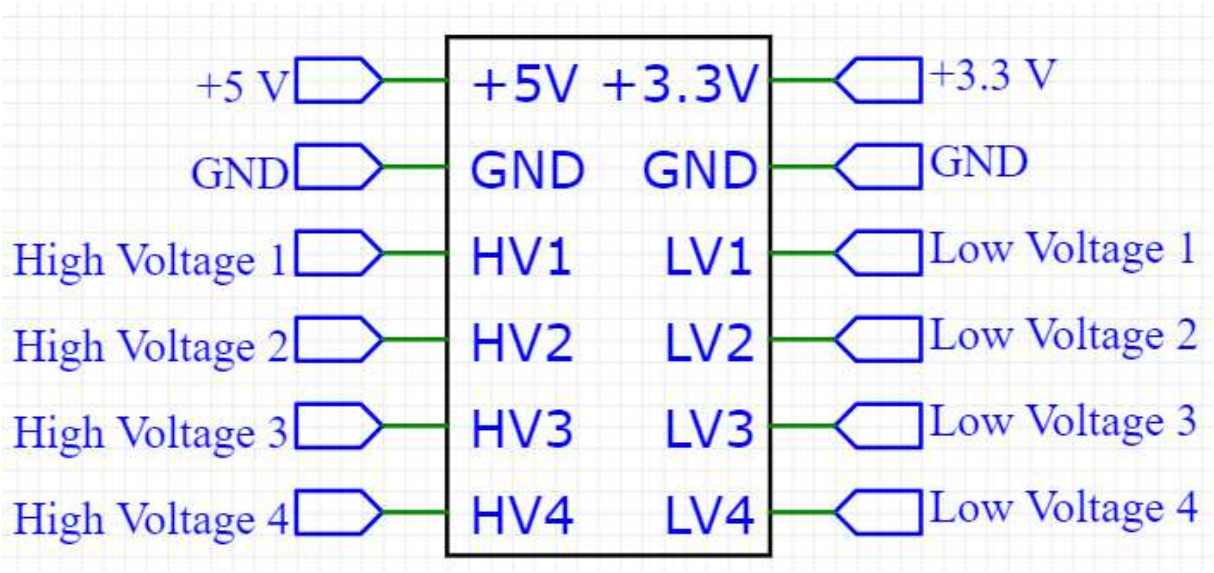


Abbildung 3-7: Logik-Wandler

3.4 Raspberry Pi

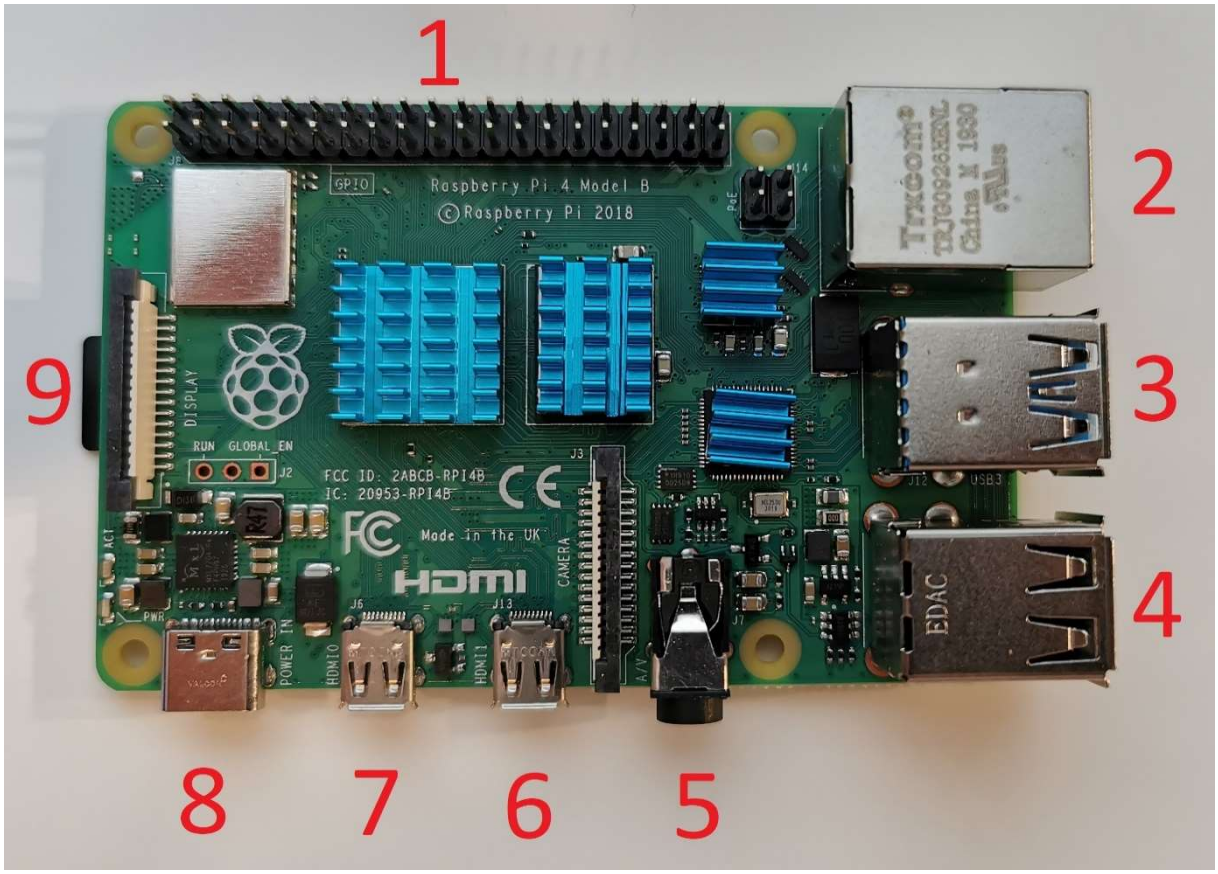


Abbildung 3-8: Raspberry Pi

Der Raspberry Pi ist ein Minicomputer, welcher mit vielen unterschiedlichen Betriebssystemen bespielt werden kann. Für dieses Projekt wird eine Linux-Distribution, Raspbian verwendet. Die Größe des Rechners ist zu vergleichen mit der einer Kreditkarte. Er besitzt vier Kerne und 40 GPIO Pins, mit denen er kommunizieren, schalten und Daten einlesen kann. Neben einer RJ45 Steckverbindung ist das System auch mit WLAN und Bluetooth ausgestattet. In Tabelle 5 werden seine Anschlussmöglichkeiten aufgelistet.

1	40 GPIO Pins
2	RJ45
3 & 4	USB vierfach
5	Headphone jack
6 & 7	HDMI (0 & 1)
8	USB C Spannungsversorgung
9	MicroSD Steckplatz

Tabelle 5: Raspberry Pi [17]

3.5 Arduino Nano

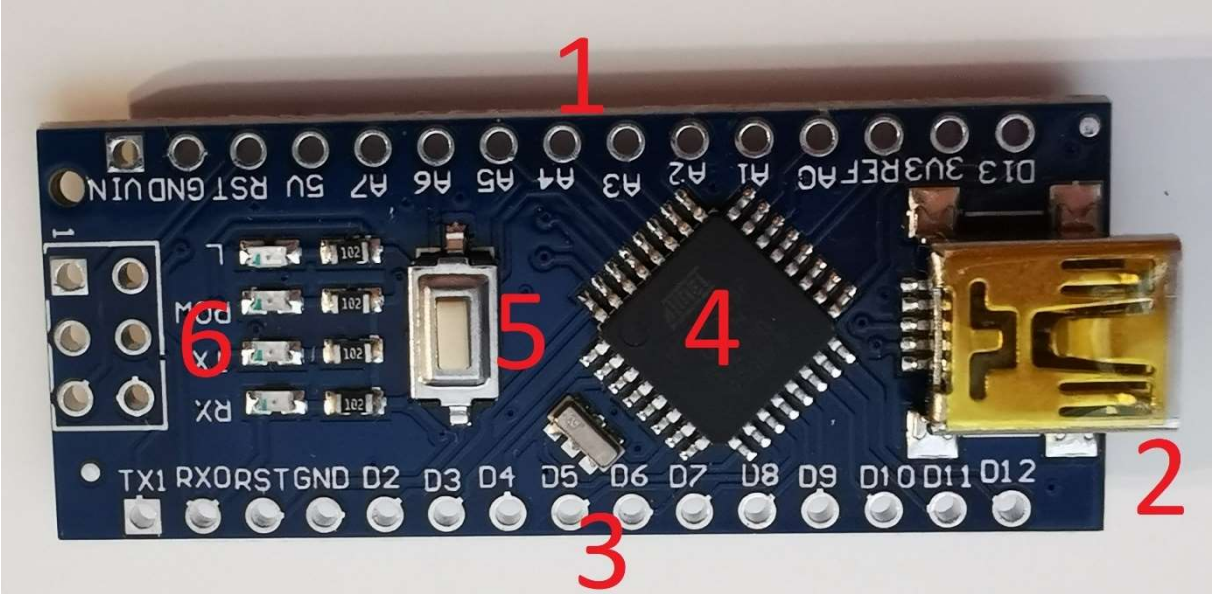


Abbildung 3-9: Arduino Nano

Der Arduino Nano ist ein Mikrocontroller mit einer Taktfrequenz von 16 Mhz. Er besitzt 8 analoge Eingänge und 22 digitale Ein- und Ausgänge. Die meisten seiner analogen Eingänge können auch als digitale Ausgänge verwendet werden (Ausnahme A6 & A7 können nur als analoge Inputs verwendet werden). Sein Speicher umfasst 32 Kilobytes. Seine Abmaße betragen 43 mm x 18 mm. Sein Logiklevel beträgt +5 V und + 3.3 V, abhängig von seiner Betriebsspannung. Er kann über folgende drei unterschiedliche Kommunikationsarten Daten austauschen : I^2C, SPI, UART. Um ihn zu programmieren, können unterschiedliche Programme verwendet werden. In dieser Arbeit wird die “Arduino IDE“ verwendet, welche den Code kompiliert und die Daten über Miniusb an den Arduino weiterleitet. Über den seriellen Monitor oder Plotter können Messwerte ausgegeben, oder Eingaben eingelesen werden. Der Arduino Nano wird von vielen Herstellern angeboten und bietet für seinen geringen Preis eine Vielfalt an Einsatzmöglichkeiten. In Tabelle 6 wird seine Hardware beschrieben.

1 & 3	GPIO Pins
2	Miniusb Steckverbindung
4	Mikroprozessor
5	Taster für Reset
6	LED Statusanzeige

Tabelle 6: Arduino Nano [18]

3.6 SPI Kommunikation

Viele Sensoren schicken ihre Werte über ein Bussystem. Abhängig von den benötigten Voraussetzungen werden verschiedene Bussysteme eingesetzt. SPI steht für “Serial Peripheral Interface“. Er funktioniert nach einem “Master – Slave Prinzip“, was bedeutet, dass es einen Masterteilnehmer gibt und einen oder mehrere Slaveteilnehmer. Ein Masterteilnehmer hat das Recht, Werte anzufordern. Ein Slave darf jedoch nicht von selbst eine Kommunikation beginnen, sondern muss auf das Ansprechsignal, dem Slaveselect, warten. Die Kommunikation ist synchron, da der Bus über ein Clocksignal verfügt. In Abbildung 3-10 wird das Funktionsschema dargestellt.[19]

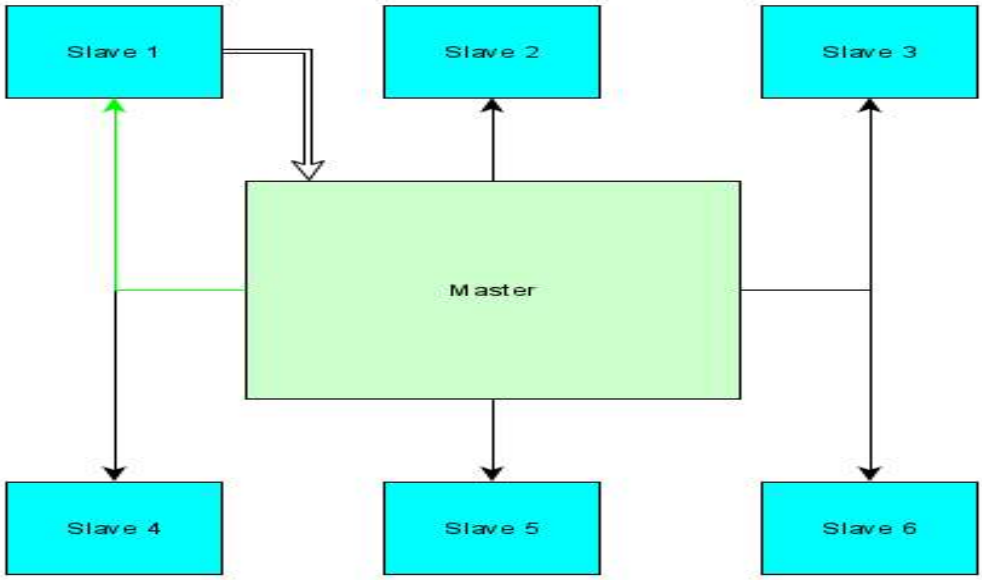


Abbildung 3-10: Funktionsblockschaltbild

Der Slave, dessen Chipselect auf logisch “Low“ geschalten wird, empfängt die Daten die der Master versendet. Meistens werden Registerdaten gesendet, damit der Slave weiß, welche Aufgabe er zu erfüllen hat. In dieser Arbeit wird der SPI-Bus verwendet, um eine Verbindung von Raspberry Pi und Arduino Nano mit dem Analog-/ Digitalconverter MCP3008 herstellen zu können. Um eine Kommunikation zwischen Master und Slave herzustellen, sind Verbindungen wie in Tabelle 7 ersichtlich, notwendig.

CS	Chipselect (oder Slaveselect)
CLOCK	Clocktakt für Synchronität
MISO	Master in slave out
MOSI	Master out slave in
GND	Gemeinsamer Massenbezug

Tabelle 7: SPI Verbindungen [19]

Abbildung 3-11 zeigt die Verbindung eines Masters mit drei Slave Teilnehmern (X, Y & Z).

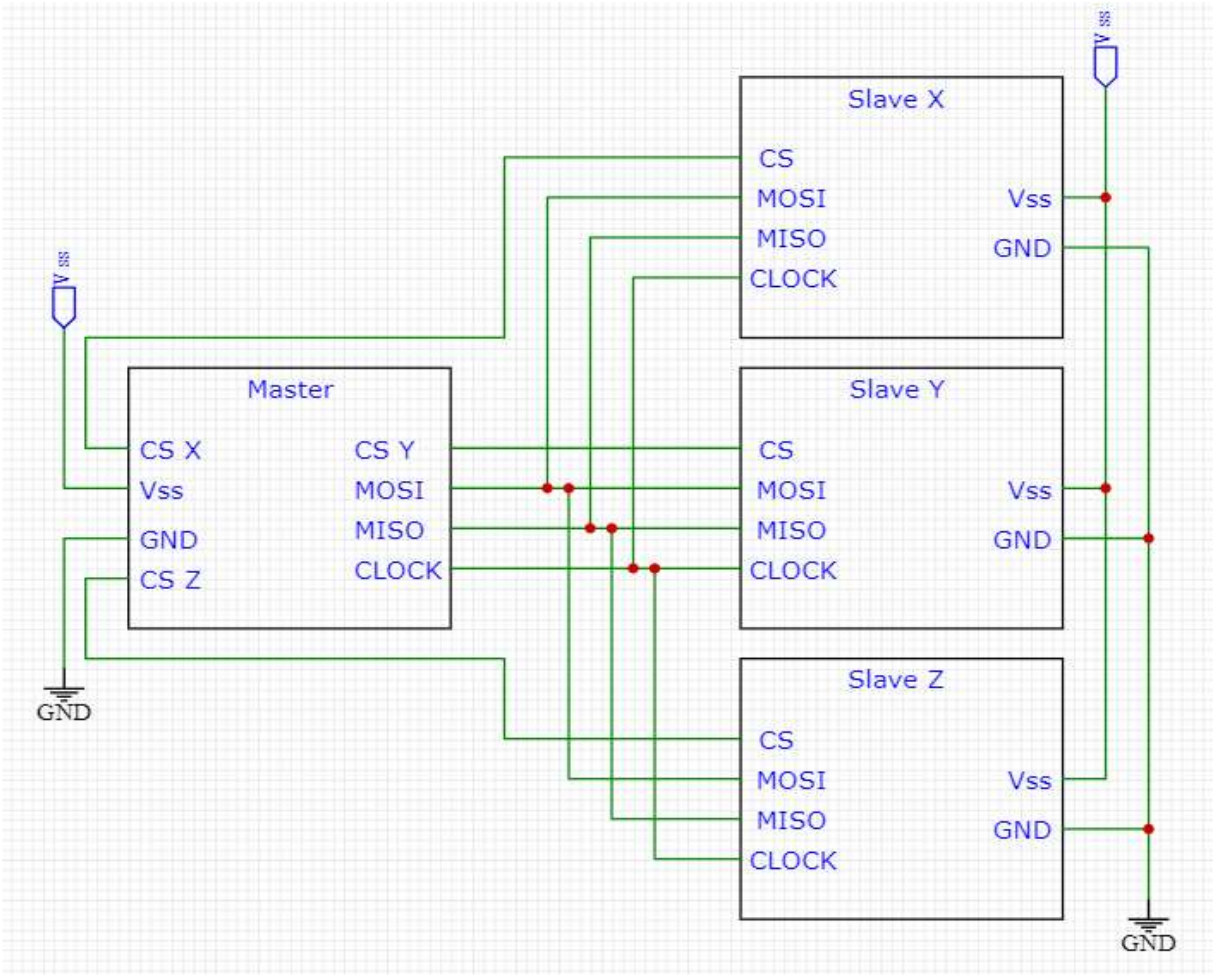


Abbildung 3-11: SPI Verschaltung

Um eine stabile Kommunikation herzustellen, ist es wichtig, ein Clocksignal zu verwenden welches auf den Slave passt, da es ansonsten zu einer fehlerhaften Datenübertragung kommen kann. Im Falle des MCP3008 darf das Clocksignal nicht zu schnell sein, da seine Funktionsweise auf einem zeitlichen Faktor beruht. Das Eingangssignal lädt mittels einer speziellen Schaltung einen Kondensator im Chip auf, welcher diese Spannung für eine vorgegebene Zeit hält, um dann mit Hilfe von Komparatoren dieses Signal abzutasten, bis ein Wert ausgegeben wird. Wird nun eine Anfrage vom Master gestellt, während das Abtasten des Slaves nicht beendet wurde, werden falsche Signale als Antwort gesendet. Dies kann bei der Messung zu großen Spannungsdifferenzen führen. Laut Datenblatt des MCP3008 wird bei Überschreiten der maximalen Clockfrequenz ein linearer Fehler eintreten.

Da verschiedene Sensoren und Aktoren an einem Bus teilnehmen können, ist es neben der Auswahl des Slaves auch wichtig, wie und welche Daten übertragen werden. Die unterschiedlichen Kommunikationsmöglichkeiten werden in Timerdiagrammen festgelegt. Diese sind in den meisten Datenblättern ersichtlich und zeigen die möglichen Funktionen der Bauteile auf.

3.6.1 Datenübertragung

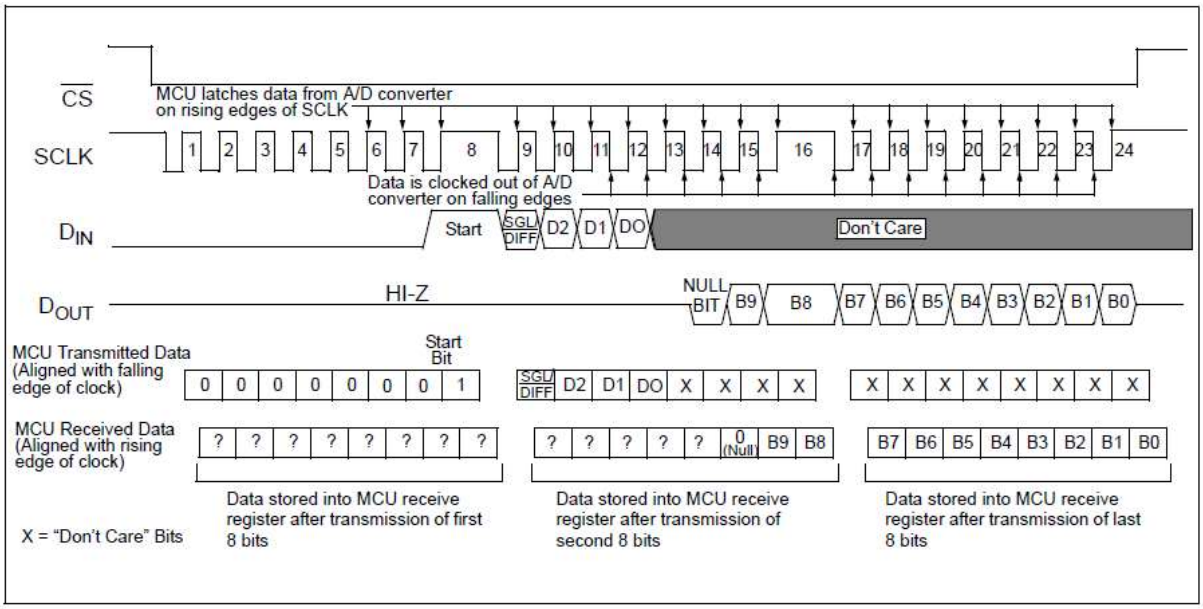


Abbildung 3-12: SPI-Bus Datenübertragung [11]

In Abbildung 3-12 wird der Kommunikationsvorgang aufgezeigt. Die oberste Zeile ist das Chipselect Signal, welches im Normalfall auf “High“ steht und erst wenn der Slave angesprochen wird, auf “Low“ gezogen wird. Somit weiß der Slave, dass mit ihm eine Kommunikation hergestellt wird. Ein Wert darunter stellt das Clocksignal dar. Es wird angegeben, dass bei steigenden Signalen das jeweilige, gesendete oder empfangene Bit zählt. Um nicht zwischen einem und dem nächsten Bit zu messen, wird das Signal immer in der Mitte des jeweiligen Bits aufgenommen. Die dritte Zeile bezieht sich auf Din (Master out slave in). Das Erste Byte hat 7 führende Nullen und ein Startbit, welches auf “High“ ist. Das nächste Byte gibt an der ersten Stelle an, ob es sich um eine “Single“-Messung handelt (Analogeingang wird mit Referenzspannung verglichen) oder um eine “Differencial“ Messung (Analogeingang 1 wird mit Analogeingang 2 verglichen usw.). Die Bit’s D2 – D0 geben an, welcher Kanal gemessen werden soll. Die Logik dahinter funktioniert ähnlich wie in Tabelle 4 ersichtlich. Alle nachfolgenden Bits haben keine Bedeutung für die restliche Übertragung. Das bedeutet, sie können jeden Wert annehmen. Ist das Byte übertragen, wird an den Dout Ausgang (Master in slave out) zuerst das “Null“ Bit übertragen und anschließend das 10te und 9te Bit. Das nächste Byte gibt die letzten 8 Bits in absteigender Reihenfolge aus. Somit sind 10 Bit Daten vom Analog-/ Digitalconverter über den SPI-Bus an den Master übertragen worden.

Um die Daten für den Master nutzbar machen zu können, muss das Programm, welches diese Daten verarbeitet, beide Bytes passend zusammenfügen (Entfernung der unnötigen Bits und ordnen der wichtigen Bits in richtiger Reihenfolge). Ist die Übertragung fertig, wird die Chipselectleitung des Slaves wieder auf “High“ gesetzt und eine neue Übertragung kann mit einem anderen Slave begonnen werden.

3.6.2 Überprüfung der Datenübertragung

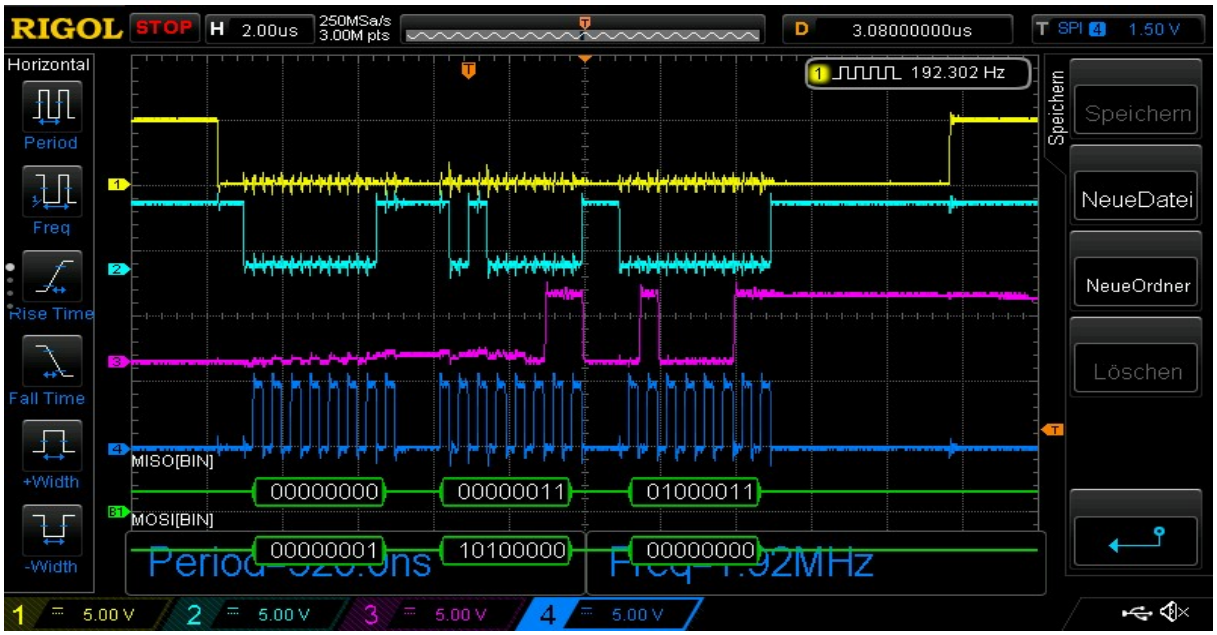


Abbildung 3-13: SPI Messung

Um das System auf eine funktionsfähige Kommunikation zu testen, wurde ein Testaufbau gemacht, mittels Spannungsquelle und Potentiometer. Die Spannung des Potentiometers wurde auf Kanal 2 des MCP3008 eingelesen. Das gelbe Signal ist Chipselect. Blau bezieht sich auf das Clocksignal, Türkis ist MOSI und Violett gibt das Verhalten von MISO an. Das verwendete Oszilloskop (Rigol DS1054Z) kann den SPI-Bus nach Bits auflösen, wie auf der unteren Hälfte von Abbildung 3-13 ersichtlich ist. MOSI gibt als erstes das Startbit aus, gefolgt von einer Singlekanalmessung und dem Kanal 2. MISO antwortet mit der Bitfolge 1101000011, was einem Wert von 835 in Dezimal entspricht. Multipliziert man diesen Wert mit Formel (3), bekommt man als Ergebnis eine Spannung von 4.0748 V, was der gemessenen Spannung von 4.075 V nahekommt. Ein Versuch mit einer anderen Spannung ist in Abbildung 3-14 ersichtlich.

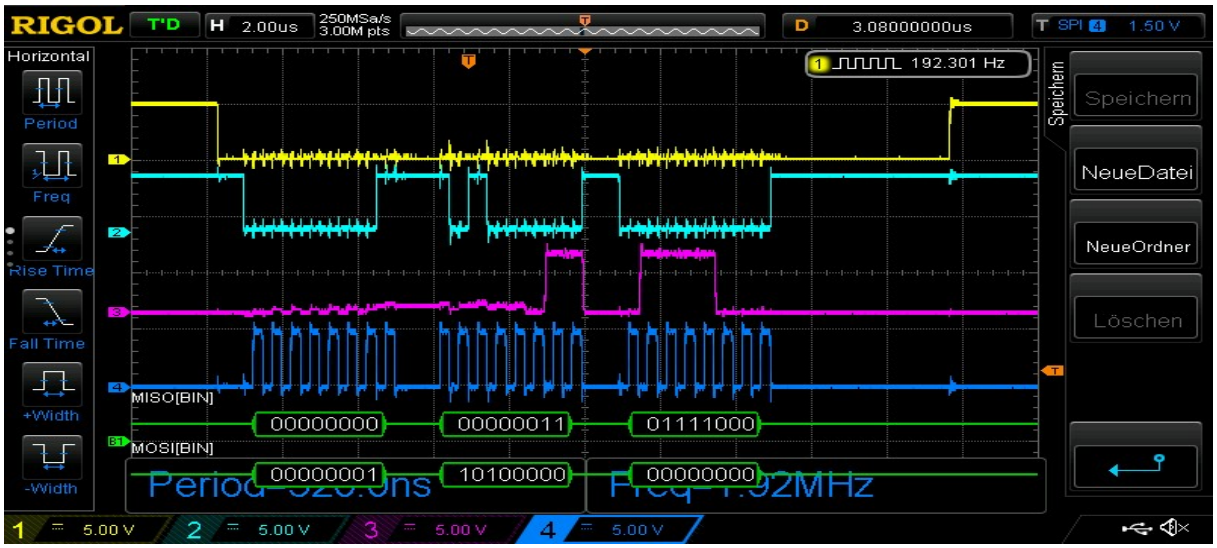


Abbildung 3-14: SPI Messung 2

3.7 UART

Die zweite Kommunikationsart, die für diese Arbeit verwendet wird, ist die Kommunikation über UART. Die Bezeichnung steht für “universal asynchronus receiver transmitter“. Da es kein Clocksignal gibt wie bei dem SPI-Bus, muss bei Sender und Empfänger eine fixe Baudrate eingestellt werden. Die Baudrate gibt an, wie viele Zeichen pro Sekunde übertragen werden. Ein Standardwert beträgt 9600 Baud. Um die Übertragung zu erhöhen, kann die Baudrate erhöht werden. Da die Baudrate aber nur fix vorgegebene Werte annehmen kann, sind die Möglichkeiten begrenzt.

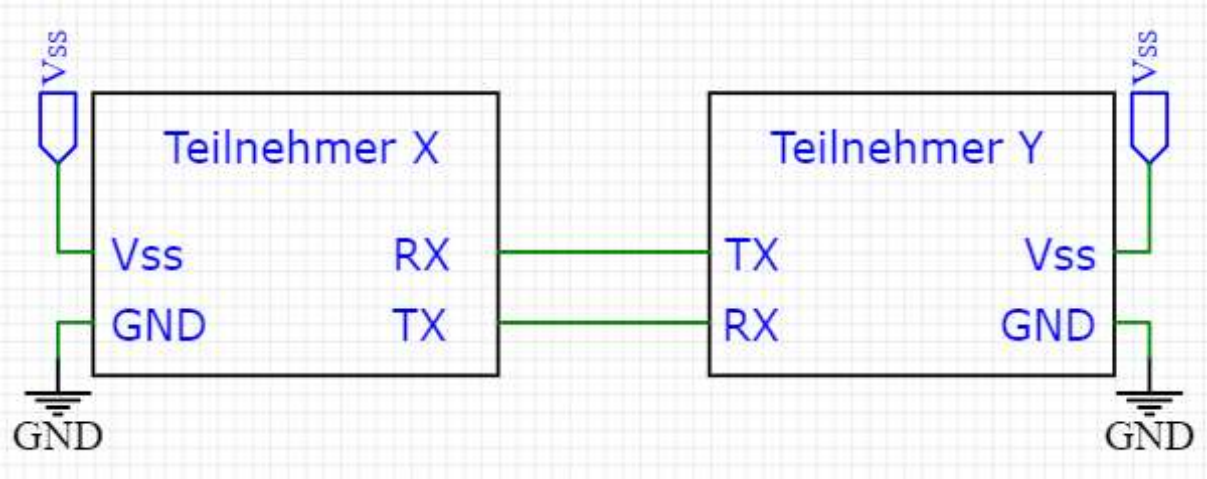


Abbildung 3-15: UART Kommunikation

Um eine Kommunikation aufzubauen, ist neben der Baudrate und einem passenden Protokoll noch eine Verbindung von zwei Signalleitungen notwendig. Außerdem müssen die Massen der beiden Teilnehmer zueinander in Beziehung stehen.

Die Signal Ein-/ Ausgänge werden mit RX (Receiver) und TX (Transmitter) bezeichnet. TX überträgt und RX empfängt die Daten. Die empfangenen Daten werden bei den meisten Mikrocontrollern in einem Buffer zwischengespeichert und sind für eine bestimmte Zeit verfügbar. Nach Ablauf der Zeit wird der Buffer gelöscht und man kann auf die Daten nicht mehr zugreifen.

Abhängig davon, welches Protokoll verwendet wird, sind verschiedene Einstellungen bei der Übertragung zu setzen. Die Übertragung wird mit einem Startbit gestartet. Hierbei wird die TX Leitung von “High“ (im Leerlaufbetrieb) auf “Low“ gezogen. Anschließend werden abhängig von der Einstellung z.B. 8 Datenbits gesendet. Um die Übertragung zu beenden, wird ein Stopbit gesendet und TX verbleibt bis zu einer erneuten Übertragung auf “HIGH“.[20]

Um die Übertragung sicherer zu gestalten, können im Protokoll Bits gesetzt werden. Eine Möglichkeit ist das Setzen eines “parity“-Bits. Dieses Bit überprüft, ob die Summe aller gesendeten Bits gerade oder ungerade ist. Somit kann der Receiver feststellen, ob ein Einzelfehler (1 Bit ist verändert) aufgetreten ist oder nicht.[20]

3.7.1 Überprüfung der Datenübertragung

Für diese Arbeit wird die Übertragung zwischen dem Raspberry Pi und beiden Arduinos über Multiplexer und Logik Wandler geprüft. Um einen Versuch durchzuführen, wird ein Testaufbau auf einem Protoboard hergestellt.

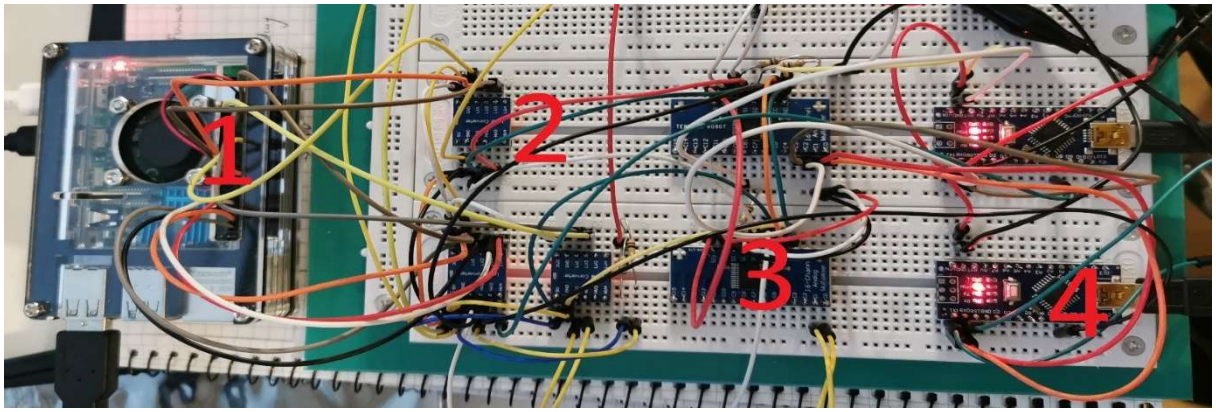


Abbildung 3-16: Testaufbau UART

1	Raspberry Pi
2	Logik-Wandler
3	Multiplexer
4	Arduino Nano

Tabelle 8: Testaufbau UART

In Abbildung 3-17 wird die Funktionsweise des Kommunikationsweges erklärt. Die dicken Leitungen sind Signalleitungen, die dünnen sind Schaltleitungen. Da der Arduino bzw. der Raspberry Pi unterschiedliche Logikpegel besitzen, müssen die verschiedenen Spannungswerte aufeinander angeglichen werden. Dies wird mit Hilfe der Logik Wandler sichergestellt. Die Multiplexer sind notwendig, um RX und TX und der einzelnen Arduinos zu schalten, da immer nur eine Kommunikationsschnittstelle hergestellt werden soll.

Im Programm des Raspberrys wird zuerst ein Zeichen über TX an das RX des ersten Arduinos gesendet. Sofern die Übertragung beendet ist, wird eine Verbindung von TX an das RX des Raspberrys hergestellt. Um zu überprüfen, ob das erste Zeichen sicher angekommen ist, wird nach einer kurzen Pause dasselbe Zeichen an den Raspberry zurückgeschickt und dort auf Gleichheit überprüft. Sofern die Überprüfung positiv war, wird dasselbe Prozedere mit Arduino zwei durchgeführt. Sind beide Übertragungen erfolgreich, kann das nächste Zeichen gesendet werden. Beendet wird die Übertragung indem ein spezielles Zeichen “Z“ gesendet wird. Dadurch wissen die beiden Arduinos, dass die Übertragung beendet ist.

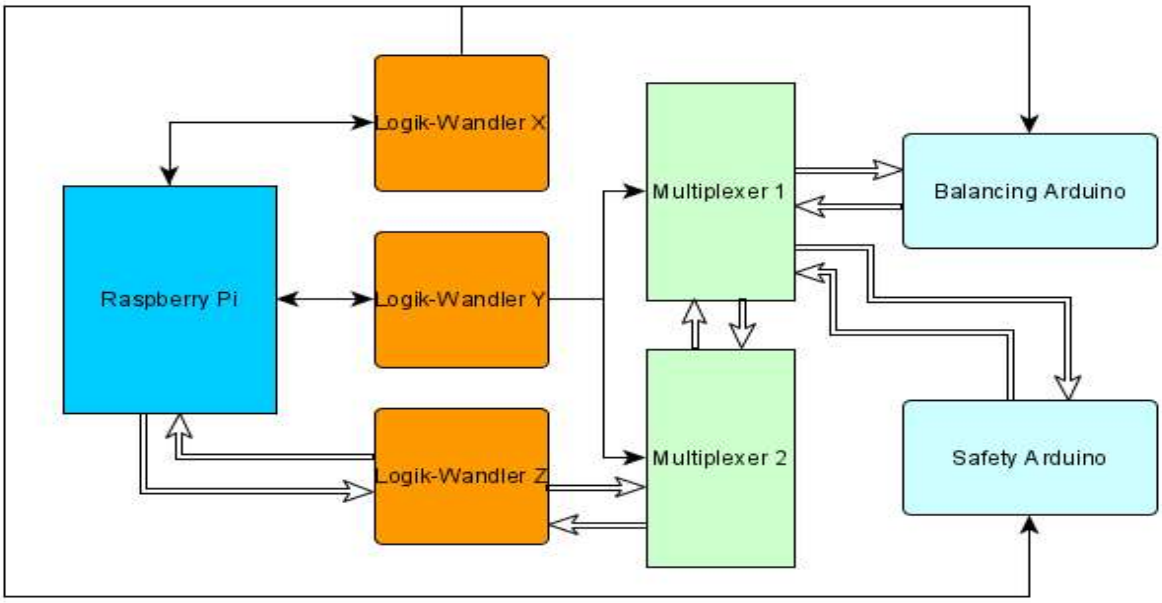


Abbildung 3-17: Funktionsweise Datenübertragung

Um zu überprüfen, ob die Signale sauber übertragen wurden, werden an den jeweiligen RX und TX Leitungen Tastköpfe für das Oszilloskop angeschlossen.

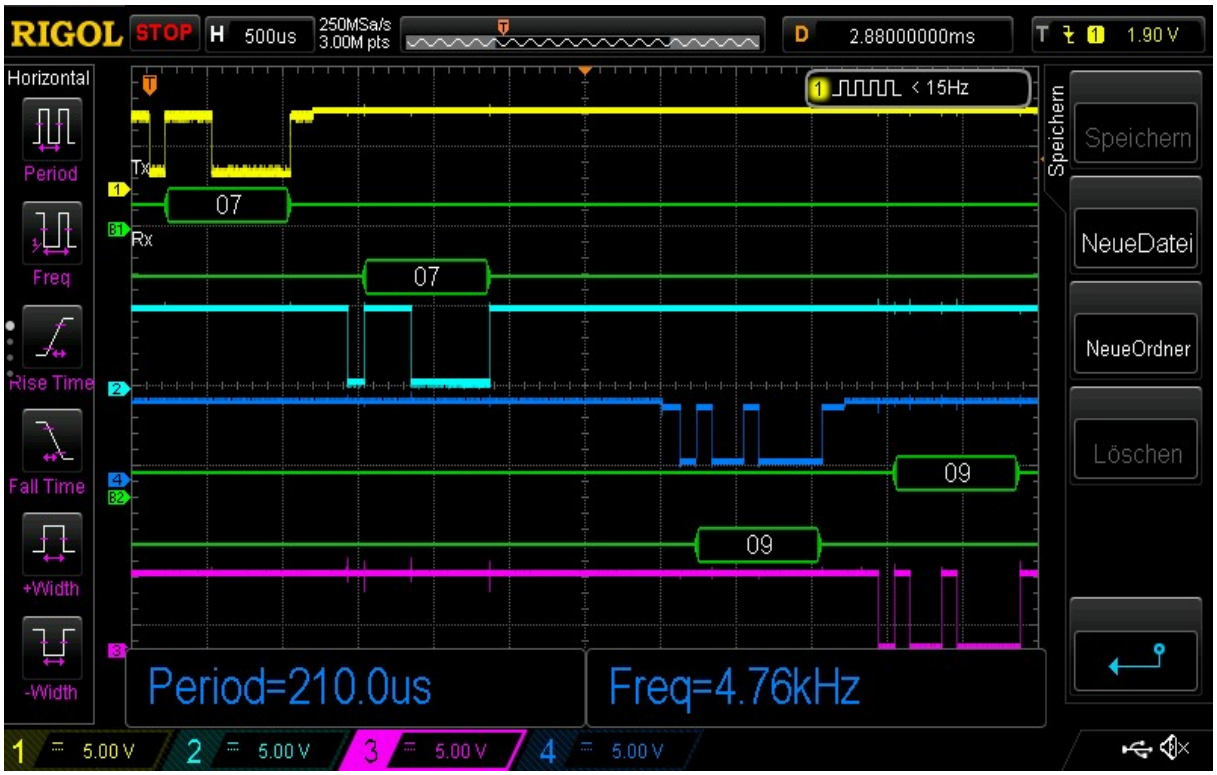


Abbildung 3-18: Signalabbildung

Im Testversuch wird Arduino 1 die Zahl 7 und Arduino 2 die Zahl 9 übertragen. Wie in Abbildung 3-18 ersichtlich, ist die Übertragung erfolgreich. Auffällig ist, dass das Signal des Raspberrys nicht so stabil ist wie jenes der Arduinos.

3.8 Logik Schaltung

Da der Prototyp von einer übergeordneten Stufe überwacht werden soll und im Fehlerfall in alle wichtigen Prozesse eingegriffen werden kann, werden die Schaltsignale durch eine Verschachtelung von Und Gattern geführt. Alle Steuervorgänge gehen vom Balancing Arduino über die Logik Gatter zu den einzelnen Systemen. Der zweite Eingang der Gatter obliegt dem Safety Arduino, der im Normalfall ein dauerhaftes “High“ Signal ausgibt. Sofern ein Fehlerfall vorliegt, zieht der Safety Arduino sein Signal auf “Low“ und die Ausgänge aller Logik Gatter sind ebenfalls “Low“. Somit wird das Schalten der einzelnen Systeme unterbunden.

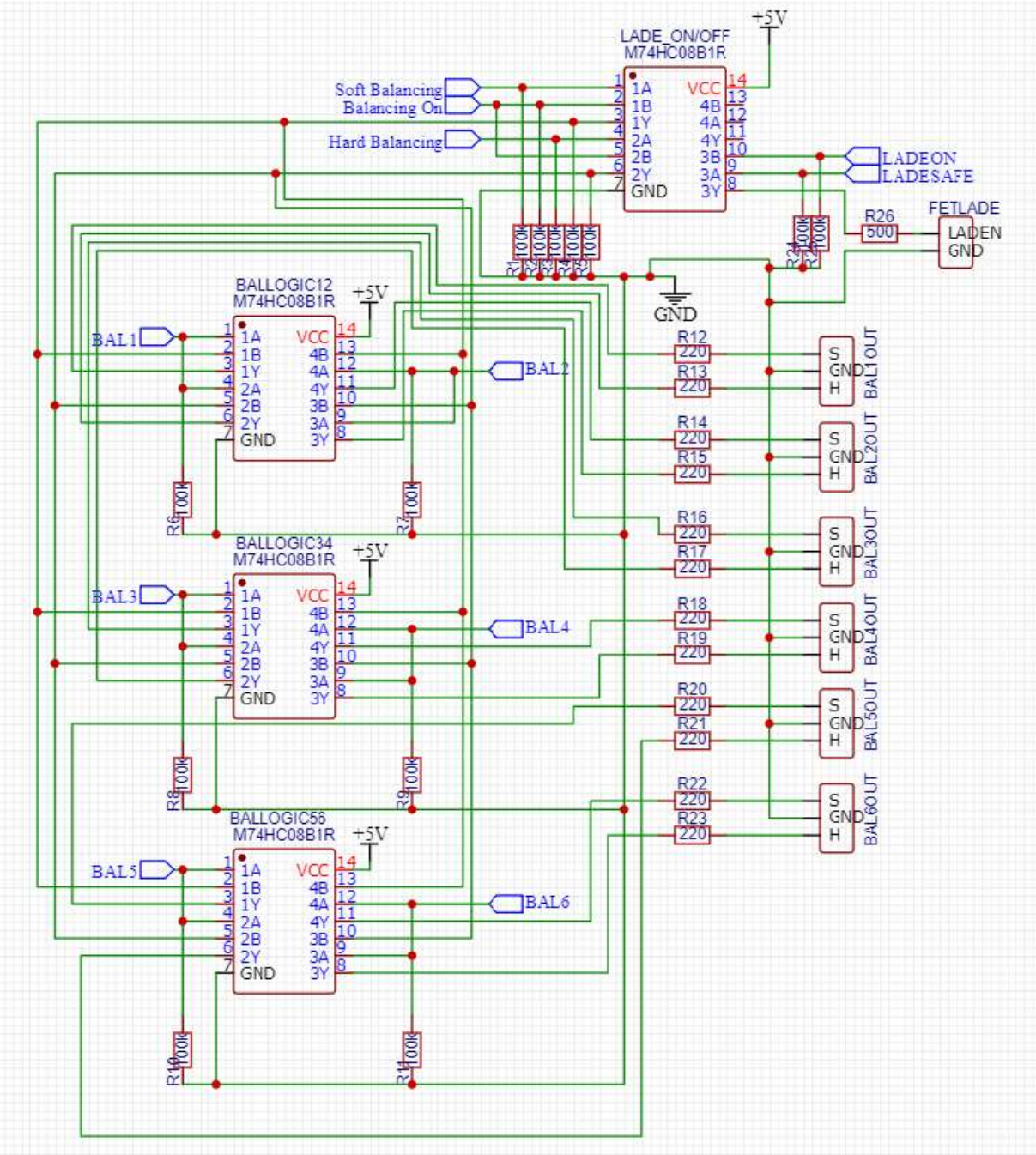


Abbildung 3-19: Logikverschaltung

4 Bauteilgruppen

Um im Fehlerfall eine Baugruppe schnell austauschen zu können, werden die einzelnen Schaltungen auf verschiedene Platinen aufgeteilt. Falls ein Fehler eintreten sollte, kann eine fehlerhafte Platine sofort durch eine funktionierende ausgetauscht werden. Um Fehler beim Löten von Verbindungen zu minimieren, werden alle Platinen im Programm “Easyeda“ erstellt, geroutet und anschließend von der Firma “Jlcpcb“ hergestellt.

4.1 Spannungsverarbeitung

Um die Spannung der einzelnen Zellen zu erfassen, werden die einzelnen Pole der verschiedenen Zellen über eine differenzierende Operationsverstärkerschaltung verschaltet. Der Ausgang dieser Schaltung gibt die Differenz der beiden Pole, also die Zellenspannung aus. Da die in der Arbeit verbauten Analog-/ Digitalconverter nur für eine maximale Spannung von + 5V zulässig sind, wird über das Widerstandsverhältnis der Differenzschaltung das ausgegebene Spannungsniveau auf 75% der Eingangsspannung herabgesetzt.

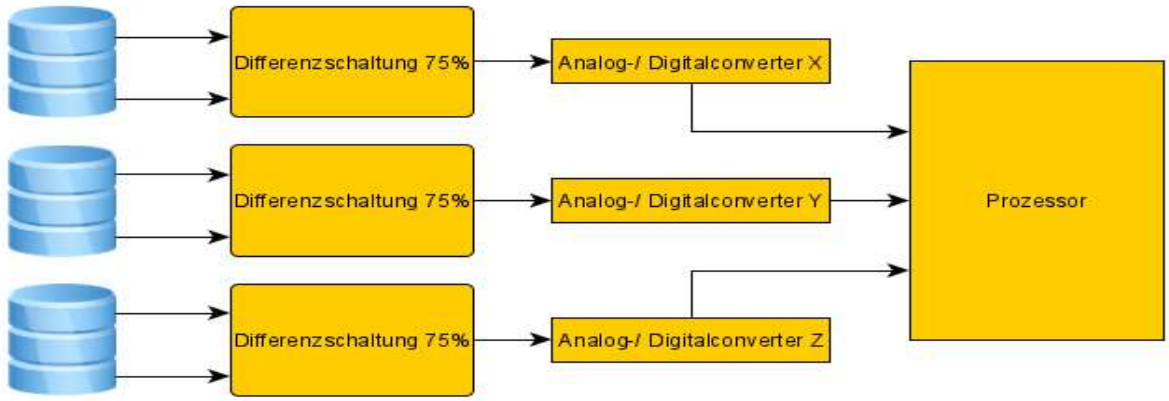


Abbildung 4-1: Funktionsweise Spannungsabnahme

4.1.1 Schaltung

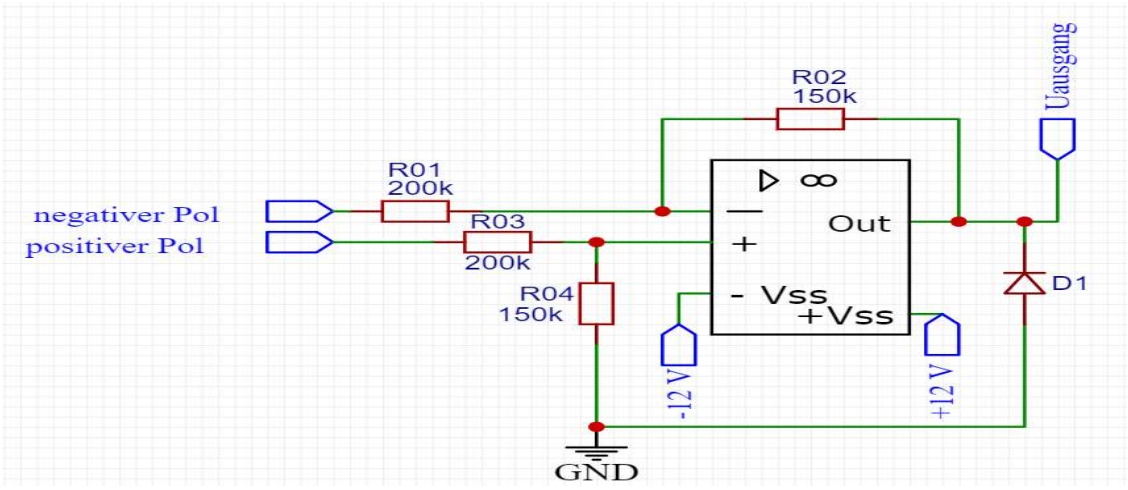


Abbildung 4-2: Differenzschaltung

Wie in Abbildung 4-2 ersichtlich, werden die beiden Zellenpole über ein Widerstandsnetzwerk mit dem Operationsverstärker verschaltet. Um eine Simulation der Schaltung zu bekommen, wird dem positive Pol ein Potential von +5 V und dem negativen Pol ein Potential von 0 V vorgegeben. Die Ausgangsspannung ist in Abbildung 4-3 ersichtlich.

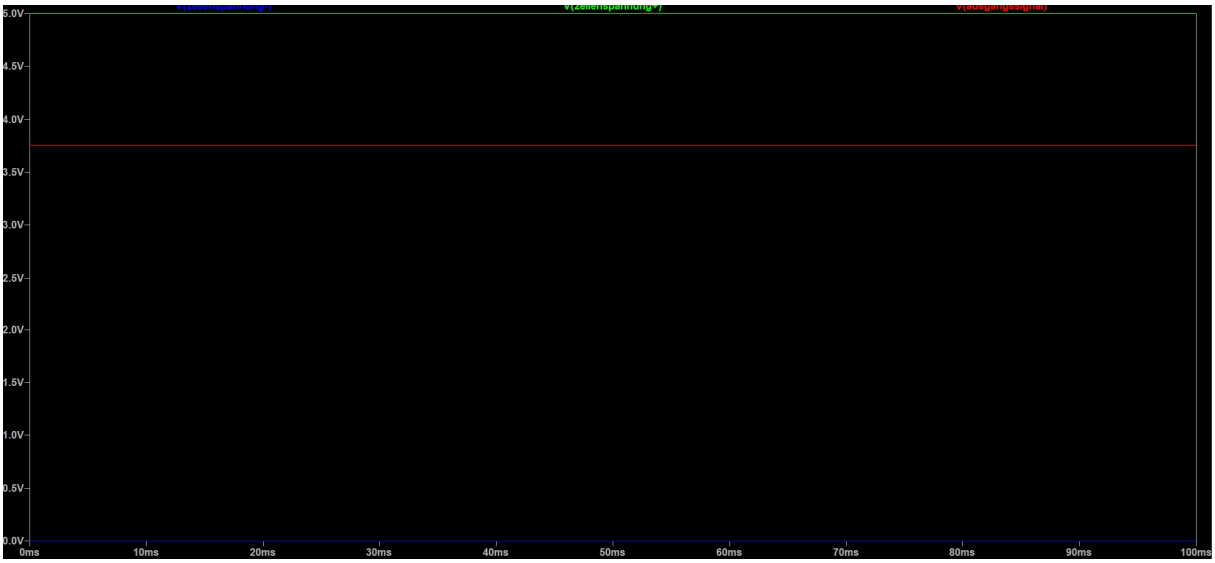


Abbildung 4-3: Simulation der Schaltung

Laut Formel (1) ergibt sich bei den Widerstandswerten R1 = 200 kOhm und R2 = 150 kOhm eine Ausgangsspannung von + 3.75 V, welche auch in Abbildung 4-3 ersichtlich ist.

4.1.2 Umsetzung auf Platine

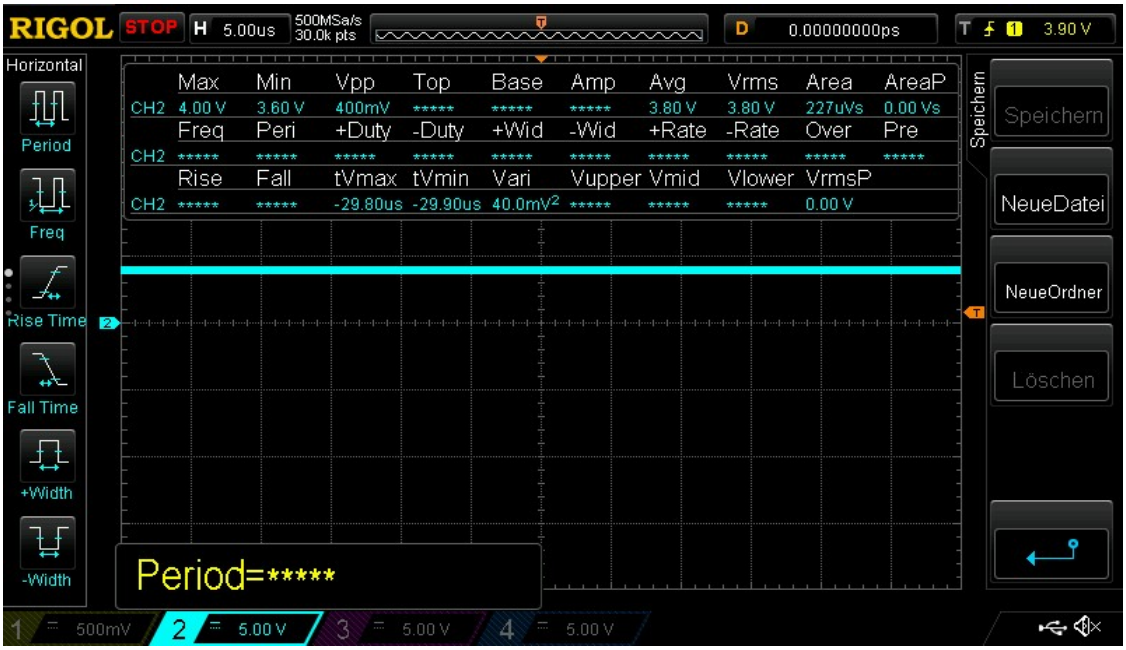


Abbildung 4-4: Messung der Schaltung

Um das Spannungsverhalten in der Realität zu überprüfen, wurde eine Testschaltung aufgebaut, welche einen ähnlichen Spannungsverlauf anzeigt. Unterschiede sind aufgrund von Toleranzen an den Widerständen zu erklären. Außerdem verhält sich ein Operationsverstärker unter realen Bedingungen nie ideal.

4.1.3 Schaltplan

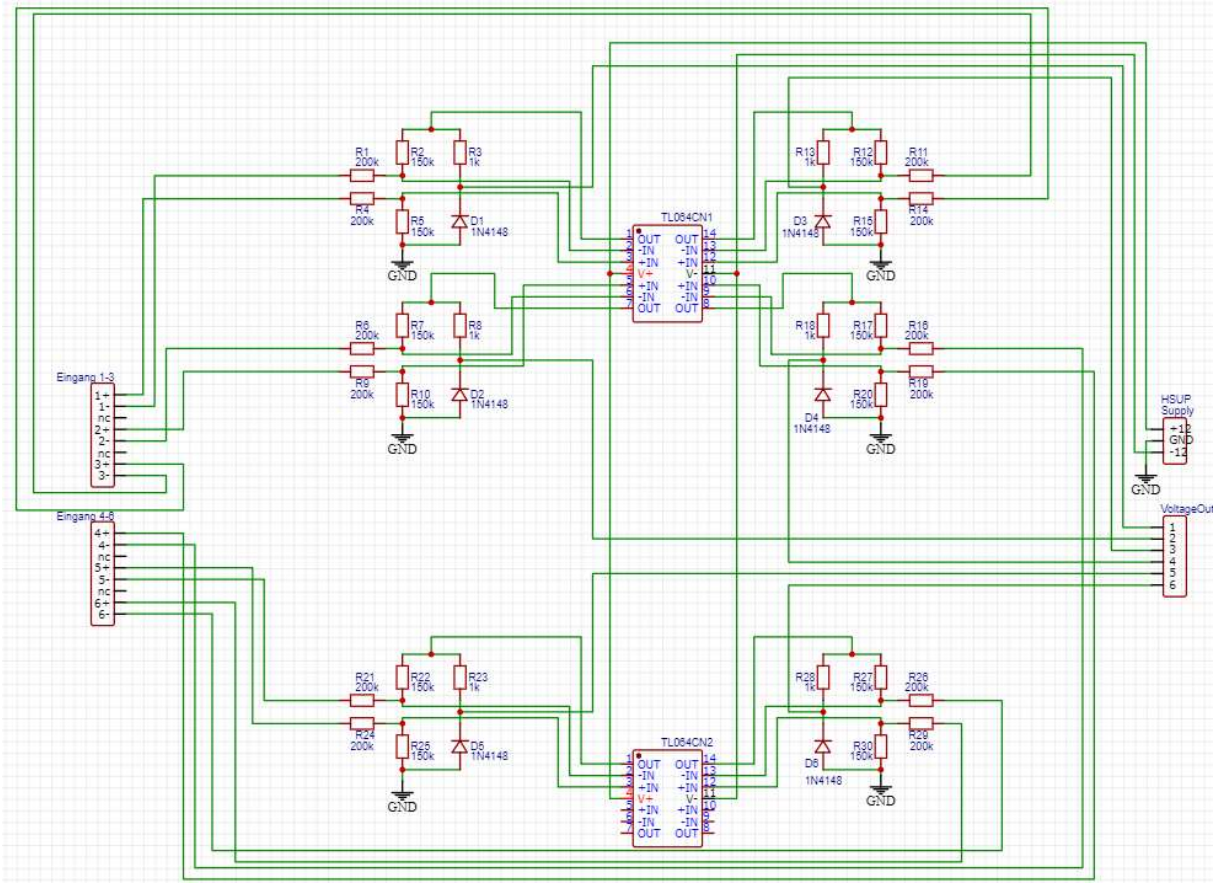


Abbildung 4-5: Schaltplan der Platine

Da in dieser Arbeit bis zu 6 Zellen verarbeitet werden können, werden die verschiedenen Zellspannungen auf zwei Operationsverstärker aufgeteilt. Die Widerstände, die auf die Platine gelötet werden, weisen eine Toleranz von kleiner als 1 % auf, um das Ausgangssignal so wenig wie möglich zu verfälschen. Die Platine bekommt ihre Messwerte über zwei 8-fach-Molexverbindungen und gibt ihre verarbeiteten Spannungswerte über einen 6-fach Molexstecker aus. Beide Operationsverstärker werden mit +/- 12 V versorgt, welche von einem Schaltnetzteil geliefert werden.[30]

4.1.4 Platinen

Der Schaltplan wurde anschließend auf eine leere Platine überspielt. Die einzelnen Bauteile wurden passend platziert und die einzelnen Verbindungen wurden anschließend miteinander durch Routing verbunden.

Um eine Unterdimensionierung ausschließen zu können, wurden die einzelnen Datenleitungen um mindestens 50% überdimensioniert. Das gleiche gilt auch für stromführende Leitungen, welche um 100% stärker ausgelegt wurden, so wie die verbauten “Vias“.

Damit für die Nachvollziehbarkeit im Nachhinein alle Verbindungen übersichtlich gestaltet sind, werden alle Steckverbinder und Bauteile eindeutig beschriftet.

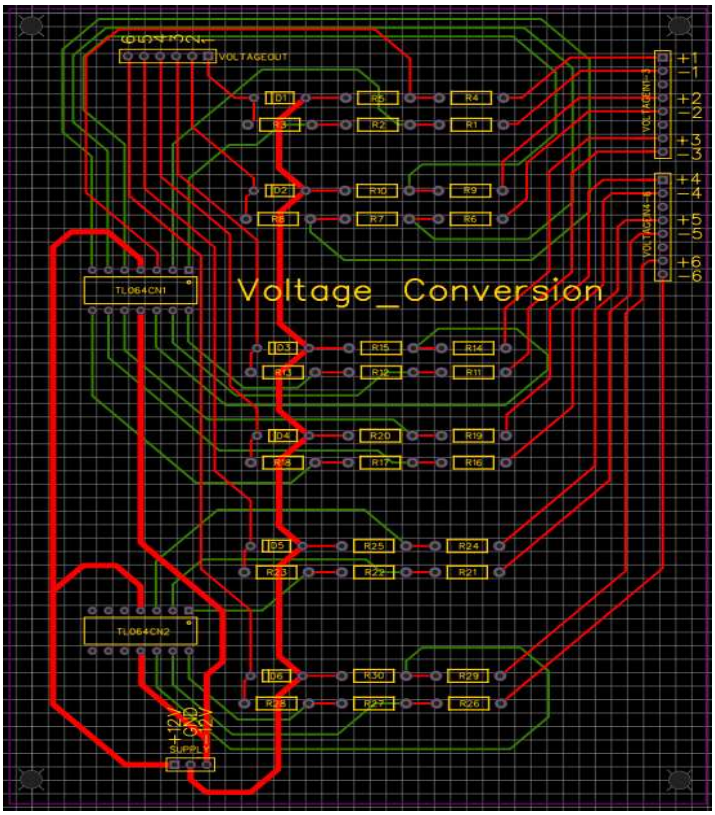


Abbildung 4-6: Routing Spannungsverarbeitungsplatine

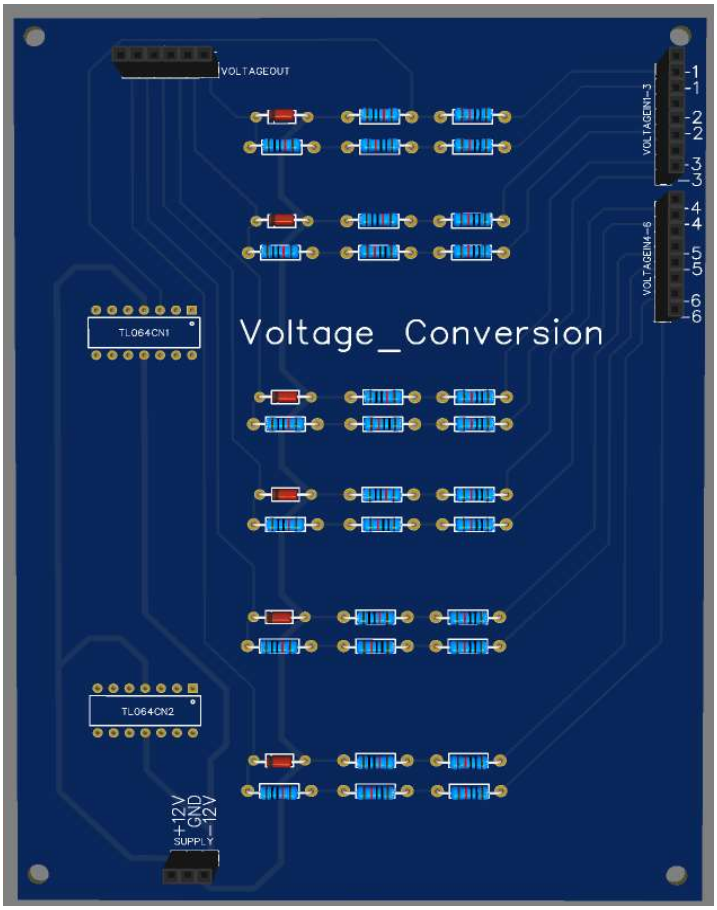


Abbildung 4-7: 3D Ansicht der Platine

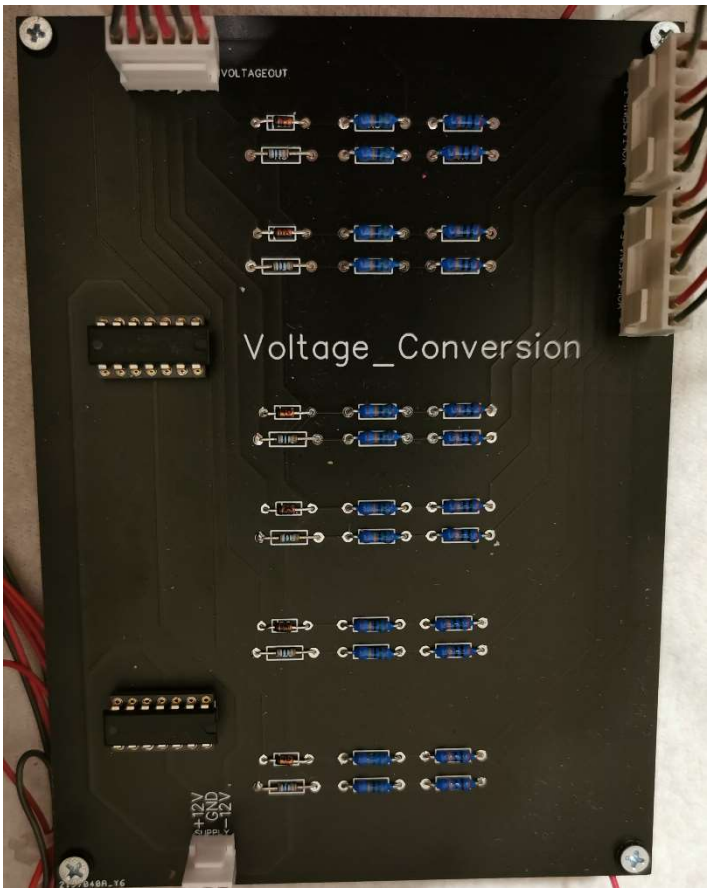


Abbildung 4-8: Platine fertig

4.2 Balancer

Die Balancingeinheit besitzt zwei unterschiedliche Modi. Man kann zwischen einem schnellen und einem langsamen Balancing wählen. Im Betrieb kann zwischen den beiden Arten gewechselt werden. Optisch ist das durch eine andere angezeigte Farbe durch die LED ersichtlich. Um den Mikrocontroller zu schützen, werden die Ansteuerung und die Leistungsschaltung voneinander durch einen Optokoppler getrennt. Für die Ansteuerung werden zwei NPN und ein PNP Transistoren verwendet. Sofern eine gewissen Spannung am Optokoppler anliegt, wird Strom an die Basis des ersten NPN Transistors gelegt. Dieser Stromfluss lässt den Transistor schalten und es kommt zu einem Stromfluss, der von der positiven Zellenspannung über die Basis des PNP Transistors in Richtung negativer Zellenspannung über den ersten NPN Transistor fließt. Da der PNP Transistor nun leitet, wird an der Basis des letzten Transistors Strom angelegt, was zu einem Durchschalten des Transistors führt. Nun ist eine leitende Verbindung zwischen Leistungswiderstand und den zwei Polen der Zelle gegeben. Die Zelle wird nun solange gebalanced bis keine Spannung mehr am Optokoppler anliegt.[21][22][23]

Abhängig davon welcher Optokoppler beschaltet wird, wird ein anderer Lastwiderstand zugeschaltet. Das schnelle Balancing wird mit einem 10 Ohm 10 Watt- und das langsame Balancing mit einem 100 Ohm 10 Watt Widerstand durchgeführt.

4.2.1 Schaltplan

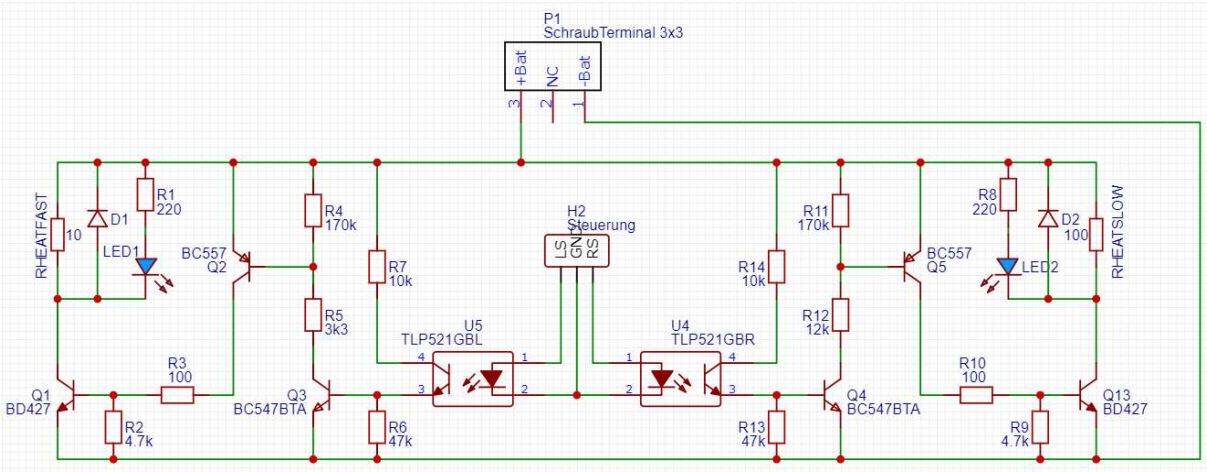


Abbildung 4-9: Schaltplan Balancingboard

4.2.2 Platinen

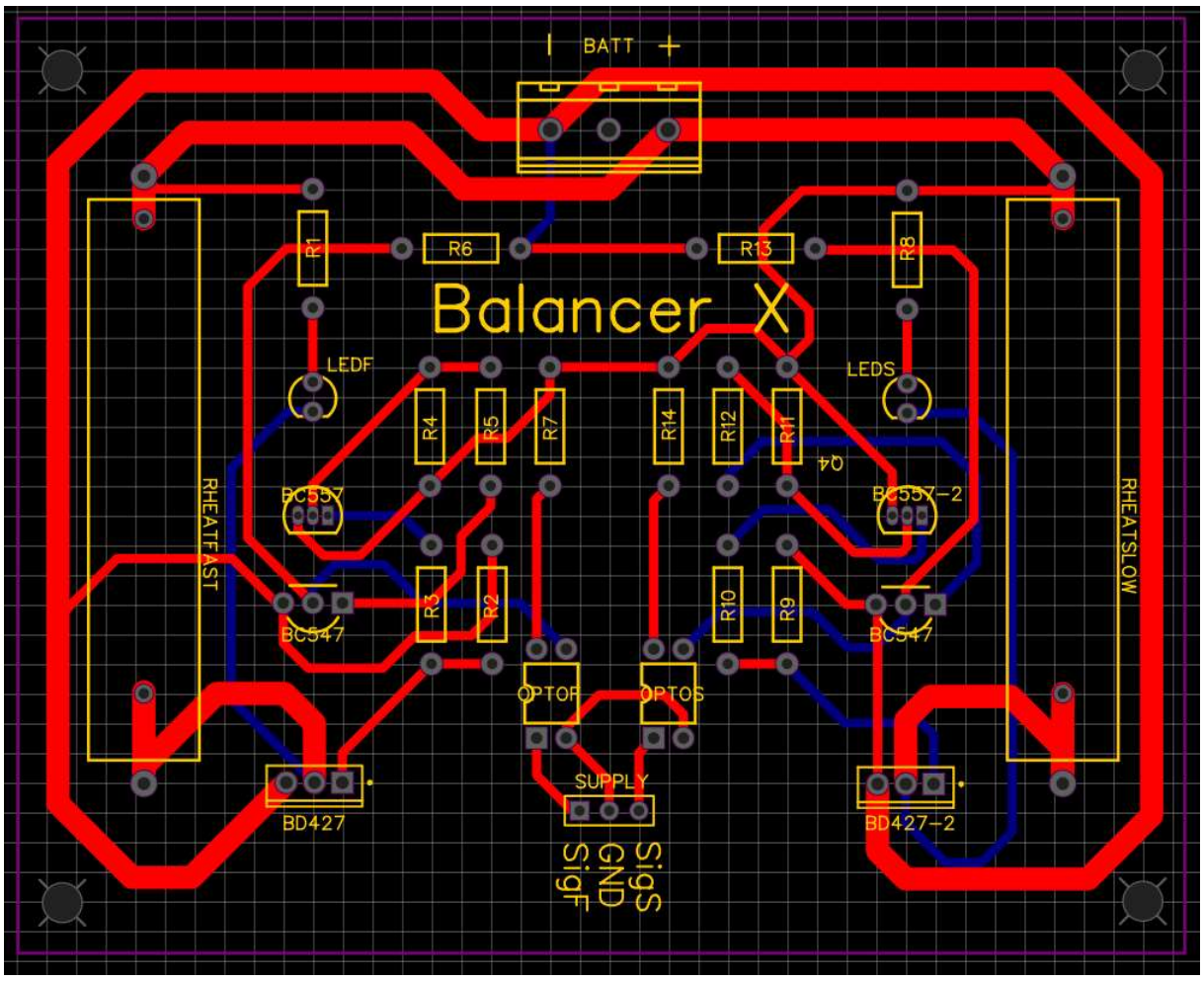


Abbildung 4-10: Gerouteter Balancer

Um beim Lötvorgang Fehler zu minimieren, wurden die Bauteile symmetrisch platziert. Außerdem wurden Löcher als Befestigungsmöglichkeiten für M3 Schrauben vorgesehen.

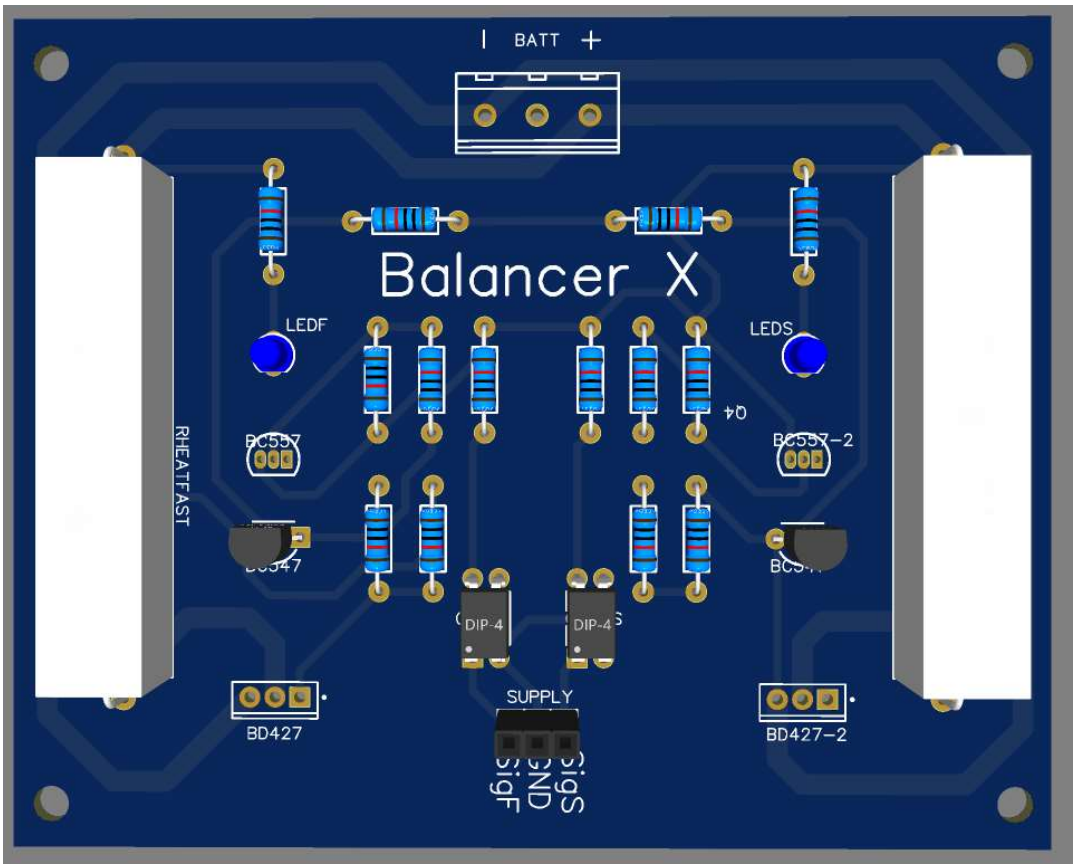


Abbildung 4-11: 3D Ansicht

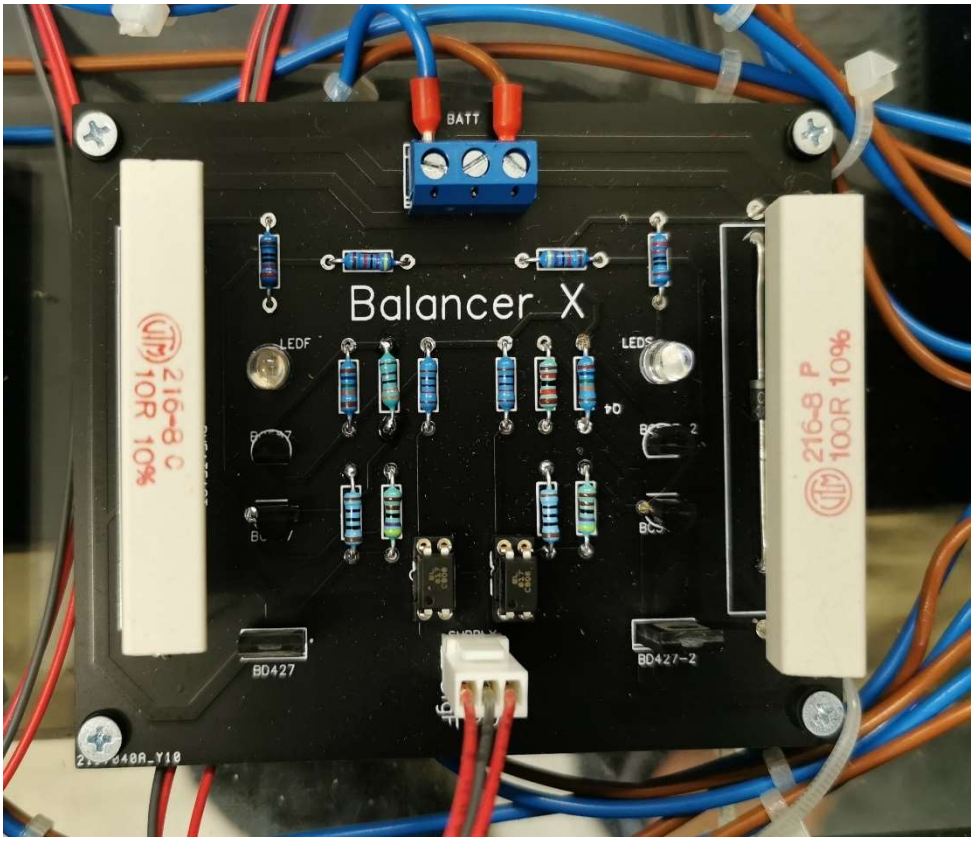


Abbildung 4-12: Fertige Platine

4.3 Referenzspannung

Damit die Analog-/ Digitalconverter genaue Ergebnisse liefern, wird eine stabile Referenzspannung benötigt. Diese Spannung wird als Bezugspunkt für die interne Elektronik der Analog-/ Digitalconverter verwendet. Je genauer und stabiler diese Spannung ist, desto genauer ist die Annäherung des Messwertes zur Eingangsspannung.

4.3.1 Schaltplan

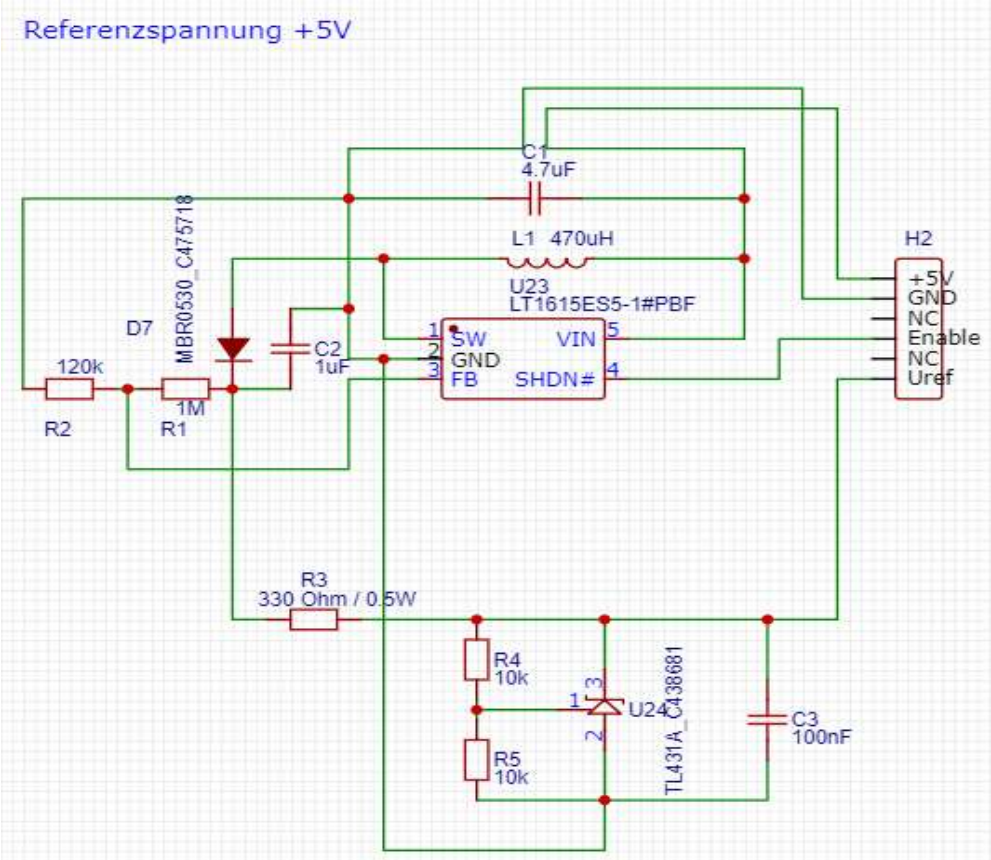


Abbildung 4-13: Schaltplan Referenzspannung

Diese Schaltung wurde nach Vorschlag der Datenblätter aufgebaut [24] und [25]. Sie gibt eine Referenzspannung von +5.046 V aus. Ein weiterer Grund, wieso eine Referenzspannung generiert wird und diese Spannung nicht gleich direkt vom versorgenden Netzteil abgenommen wird, ist der Fall, dass die Referenzspannung schneller als die Versorgungsspannung an einem Analog-/ Digitalconverter anliegt.

Es kann dabei vorkommen, dass es zu einem Fehlverhalten bzw. zur Beschädigung oder sogar zu einer Zerstörung des Bauteils kommen kann. Um diesen Fehler auszuschließen, wird eine Referenzspannungsquelle gebaut, welche erst bei geladenen Kondensatoren ihre volle Spannung ausgibt. Diese Kondensatoren haben zwar geringe Kapazitäten, trotzdem vergehen einige Mikrosekunden bis die Spannung anliegt. Dieser längere Zeitunterschied bringt einen Sicherheitsaspekt mit sich.

4.3.2 Platine

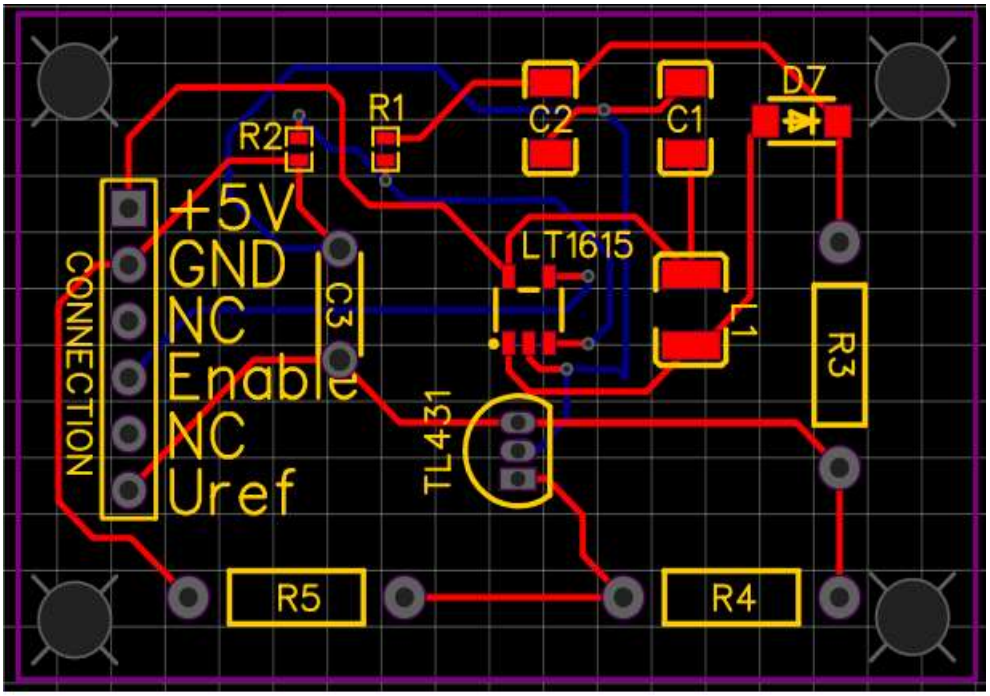


Abbildung 4-14: Routing Referenzspannung

Im Gegensatz zu “Through-hole“ Routing muss man beachten, dass SMD Bauteile keine Beine besitzen, die das Toplayer mit dem Bottomlayer verbinden können. Aus diesem Grund muss man zwischen den Schichten “Vias“ einsetzen, um eine leitende Verbindung herzustellen.

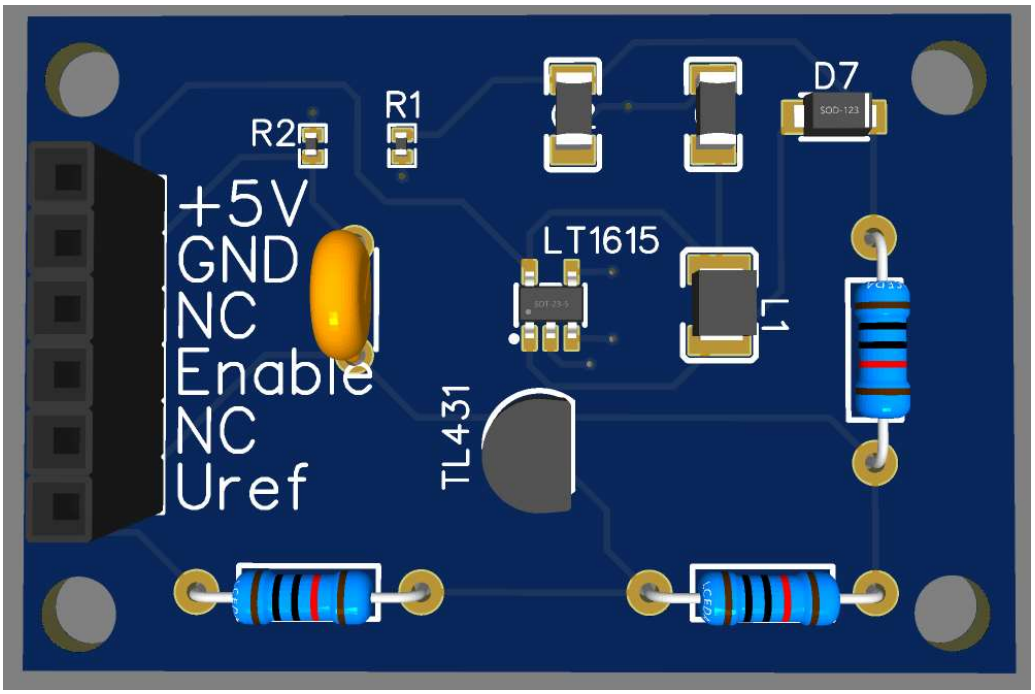


Abbildung 4-15: 3D Ansicht

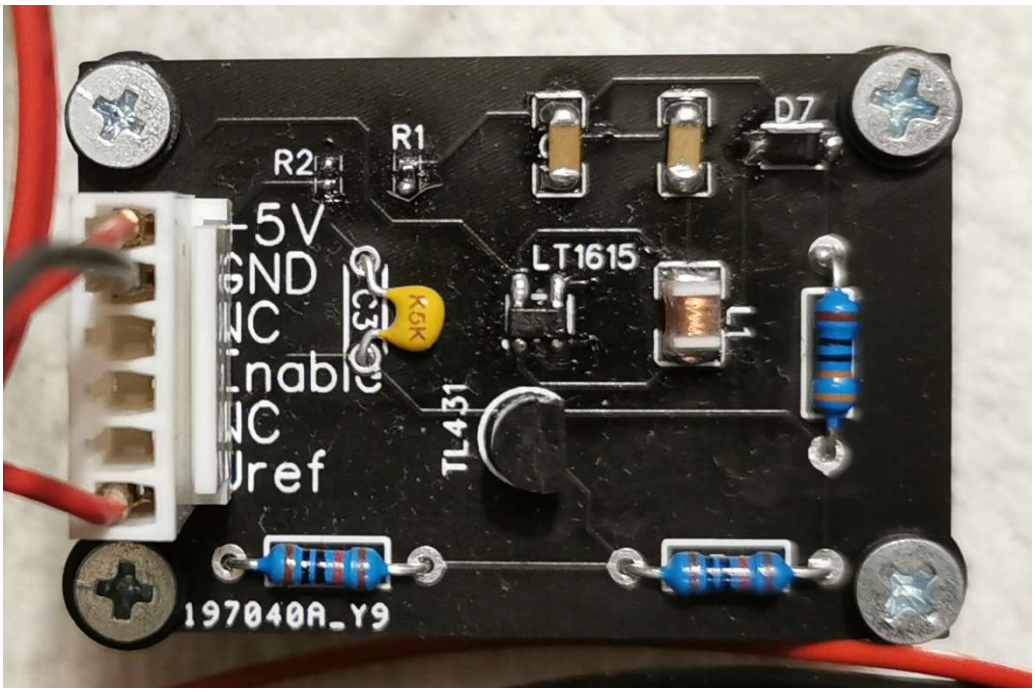


Abbildung 4-16: Fertige Platine

4.4 Verarbeitungsplatine Raspberry Pi

Der Raspberry Pi verfügt über keine analogen Eingänge und ist deswegen auf externe Bauelemente angewiesen, wenn es um eine analoge Messung geht. Durch sein Logiklevel von + 3.3 V sind auch Logik Wandler notwendig, um mit anderen logischen Spannungsniveaus kommunizieren zu können. Um alle diese Bauteile in der Nähe des Raspberry Pis anzuordnen, wird eine Verteilerplatine erstellt, welche genug Platz bietet, um Änderungen zu einem späteren Zeitpunkt der Projektstufe zu ermöglichen.

Für diesen Zweck werden extra Löcher eingefügt, um eine weitere Platine mittels Abstandshalter hinzuzufügen. Um die Stabilität zu erhöhen, wird auch die Anzahl der tragenden Löcher verdoppelt.

Um die nicht verschalteten Pins des Raspberry Pis für spätere Anwendungen verfügbar zu machen, werden die gesamten 40 GPIO mit Steckleisten auf der Platine verbunden. So hat man einerseits die Möglichkeit bearbeitete Signale auszulesen, was bei einer Fehlersuche von Vorteil ist, andererseits kann man neue Signale direkt über die Steckleisten zuschalten.

Für die Spannungsversorgung wird eine Schraubklemme auf dem Board platziert, welche die Versorgung von +5 V gewährleistet. Um die Referenzspannung für die Analog-/ Digitalconverter auf das Board zu integrieren, gibt es einen extra Steckkontakt.

Um die Referenzspannung vor Einstreuungen zu schützen, wird ein größerer Abstand zu den anderen Leitungen gewählt. Außerdem verläuft der Großteil der nebenliegenden Leitungen auf dem entgegengesetzten Layer und sind normal aufeinander ausgerichtet.

4.4.1 Schaltung

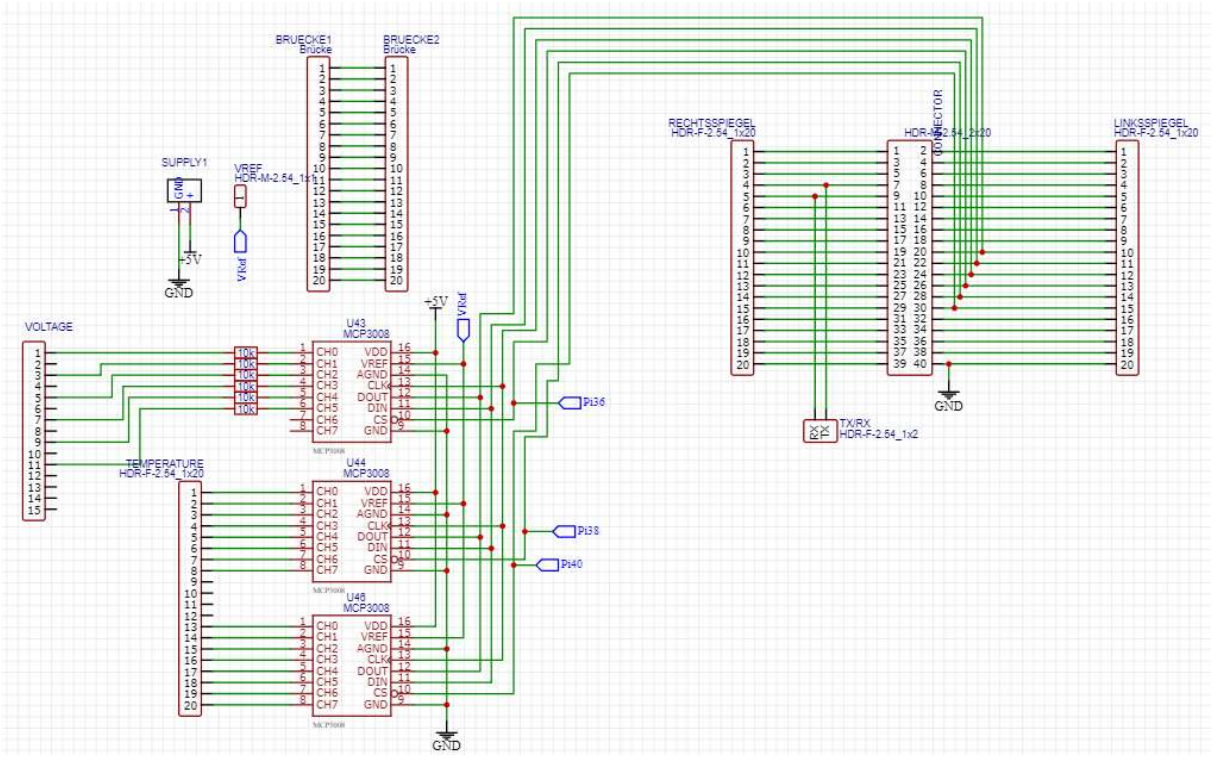


Abbildung 4-17: Schaltung Raspberry Pi Platine

4.4.2 Platine

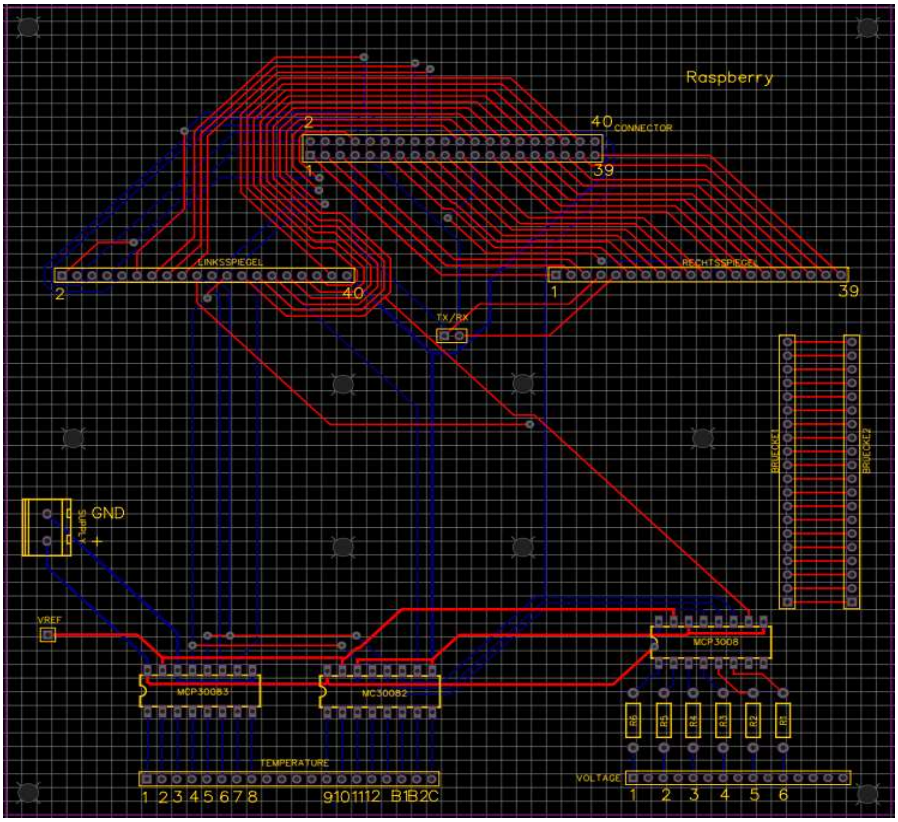


Abbildung 4-18: Routing Raspberry Pi Platine

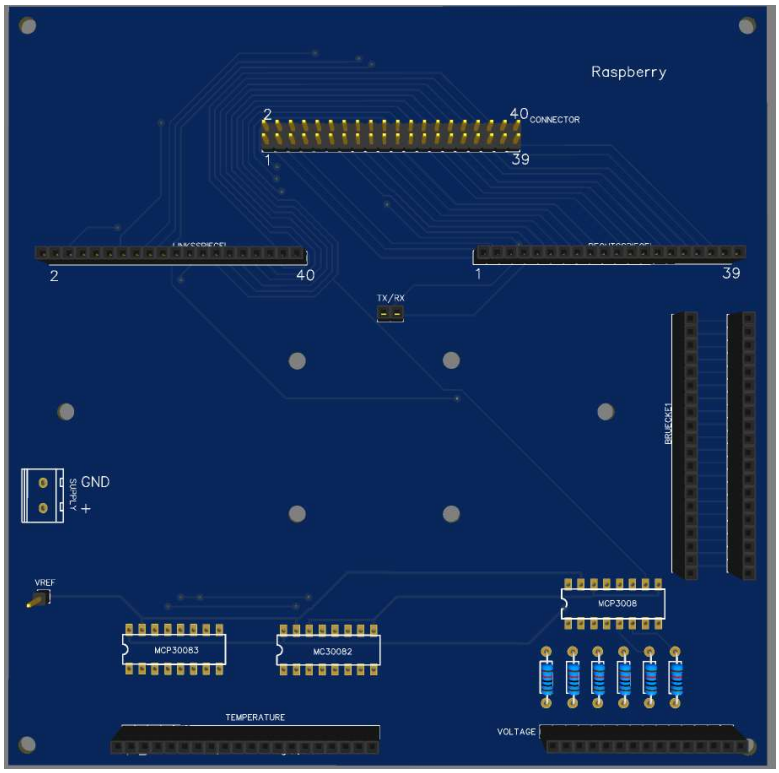


Abbildung 4-19: 3D Ansicht

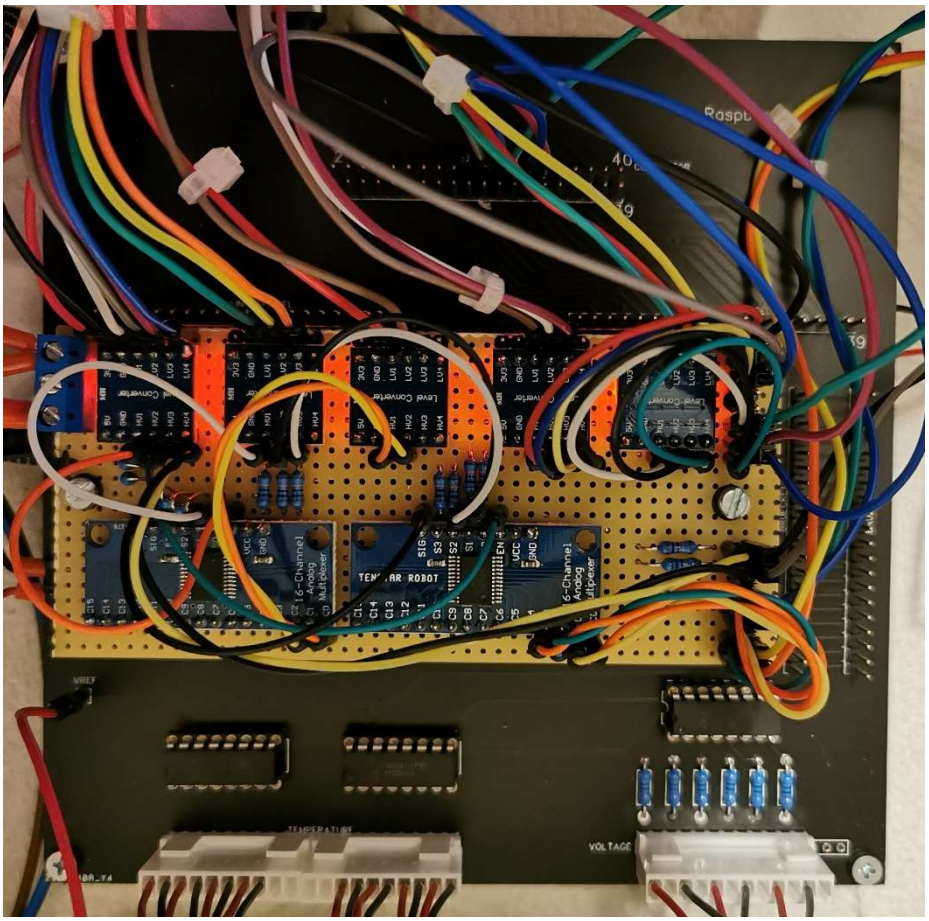


Abbildung 4-20: Fertige Platine

4.5 Balancing- & Safety Arduino

Auf dieser Platine werden beide Arduinos verbaut, sowie auch die gesammte Logikschaltung und eine eigene Baugruppe mit MCP3008 Analog-/ Digitalconverter, welche aufgrund der geringen Anzahl von Eingängen des Arduinos zugeschaltet werden. Die Analog-/ Digitalconverterbausteine werden nur vom Safety Arduino verwendet, da er mehr Messwerte zu verarbeiten hat (Temperatur, Strom, Spannung) als der Balancing Arduino (Spannung).

Die Platine wird über ein Netzteil mit + 5V versorgt. Wie bei der Raspberry Pi Platine befindet sich auch hier ein Referenzspannungseingang.

Um die Balancing-Platinen zu steuern, sind die einzelnen Leitungen für die jeweiligen Optokoppler plus Vorwiderstand mit Molexstecker verbunden.

Damit man bei Veränderung des Projekts die Möglichkeit hat, die Ein- und Ausgänge der Arduinos neu zu beschalten, sind alle ungebrauchten Pinverbindungen auf Steckleisten geführt.

4.5.1 Schaltplan

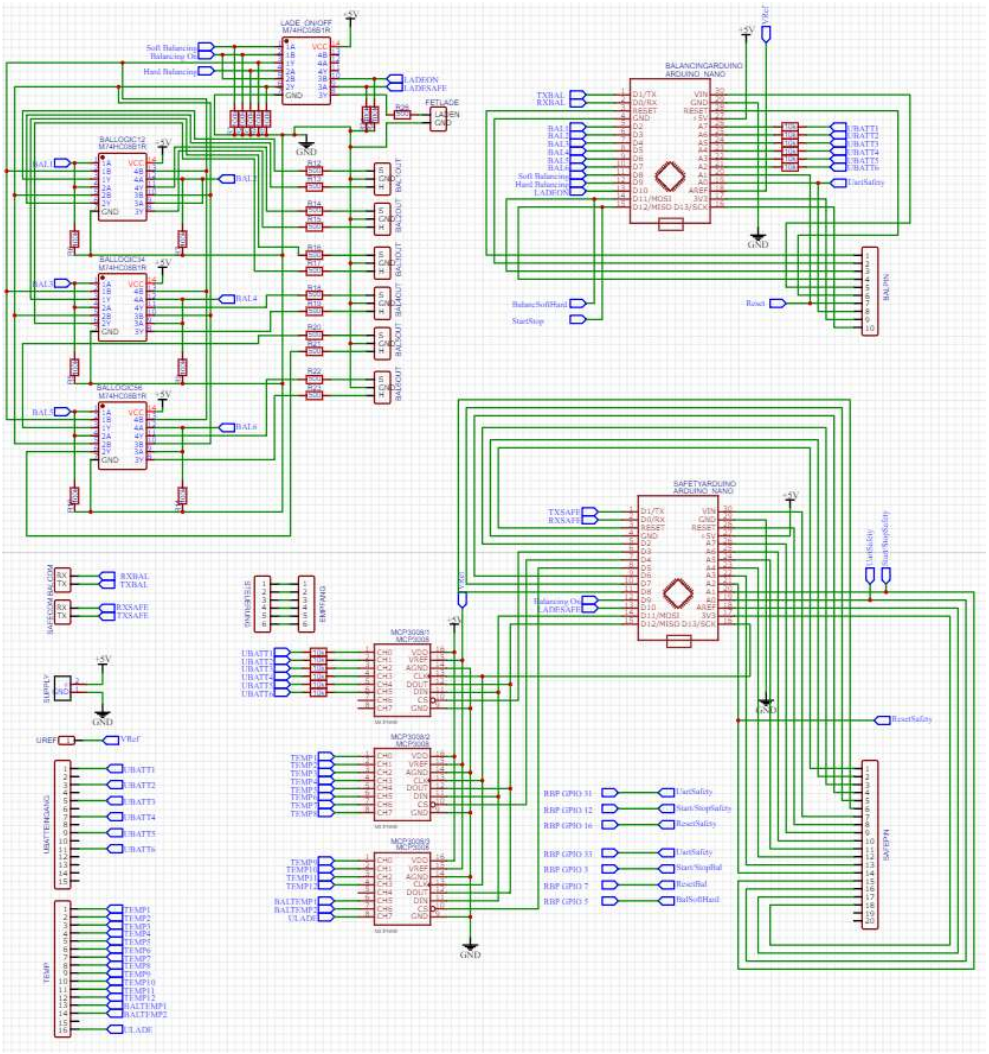


Abbildung 4-21: Schaltplan Arduino

4.5.2 Platine

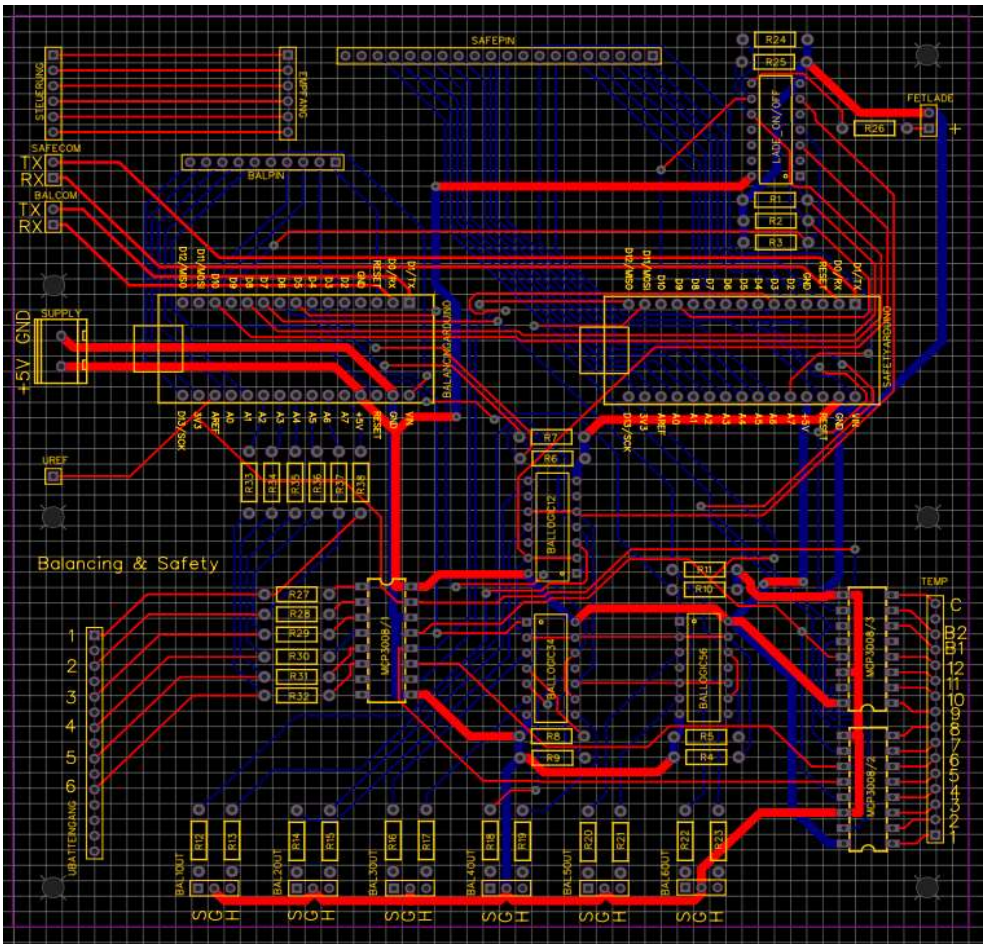


Abbildung 4-22: Routing Arduino

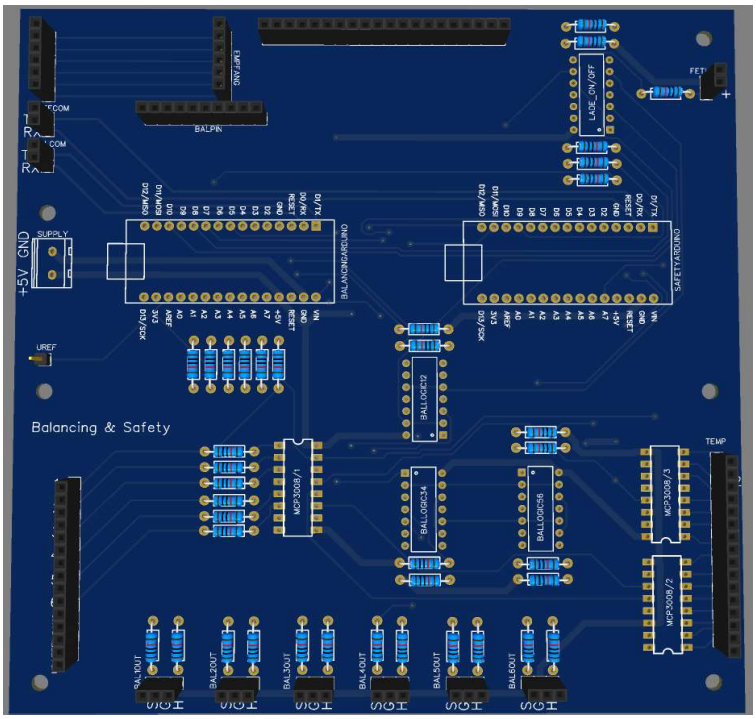


Abbildung 4-23: 3D Ansicht

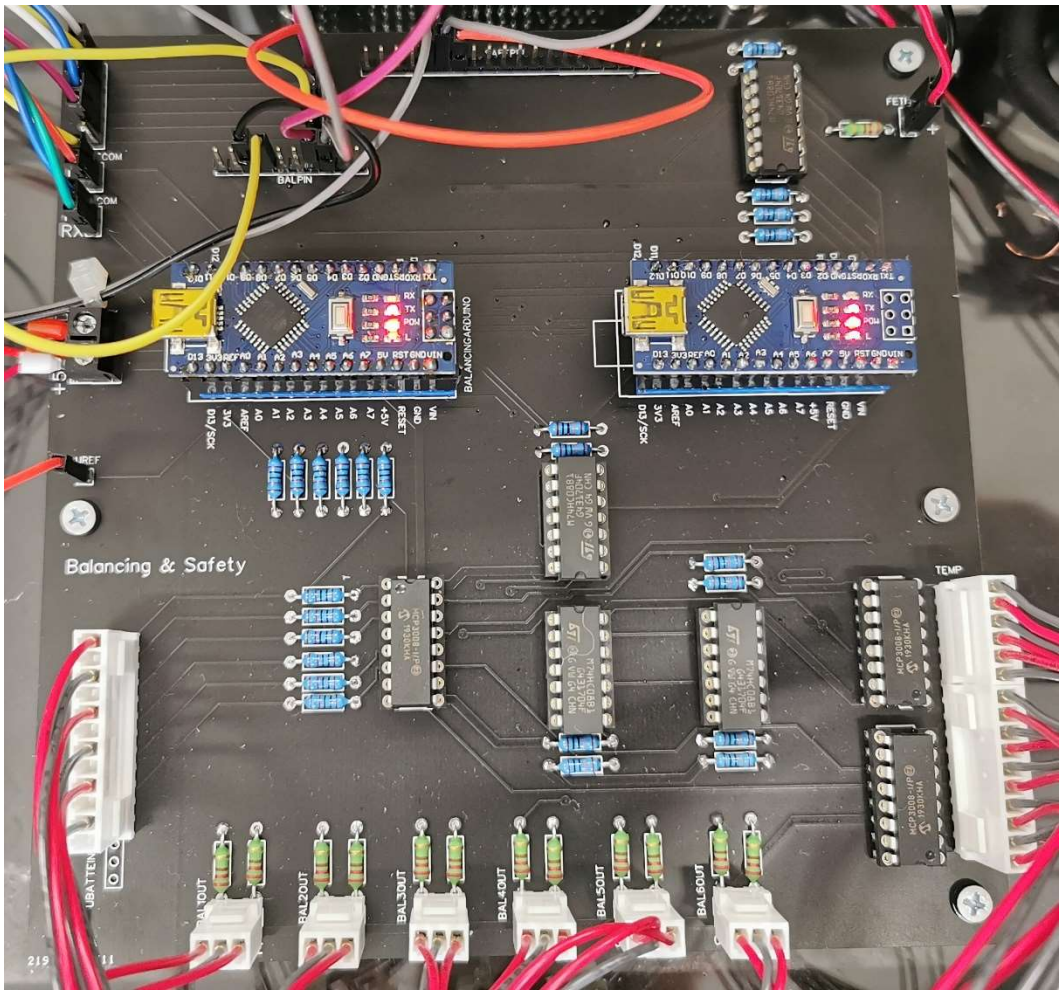


Abbildung 4-24: Reale Platine

4.6 Leistungsschaltung

Damit ein Ladestrom von 100 Ampere möglich ist, werden zwei Leistungsmosfets parallelgeschaltet. Es werden Mosfets des Typs IRFP7718 verwendet, welche laut Datenblatt über 395 Ampere schalten können. Diese Schaltung ist jedoch keinesfalls überdimensioniert. Die integrierte Schaltung kann diesen Strom schalten, jedoch ist das anliegende Gehäuse nicht in der Lage, die dabei entstandene Wärme abzugeben. Außerdem halbiert sich durch die Parallelschaltung der R_{DSON} Wert und bedingt dadurch eine noch geringere Wärmeabgabe. Um die Mosfets ausreichend zu kühlen, wird ein Kühlkörper auf Form zugeschnitten. Dieser entzieht dem Mosfet die Wärme, welche er über Konvektion an seine Umgebung abgibt.[26]

Die Schaltung wurde so ausgelegt, dass die Spannng zwischen Drain und Source unter 200 mV liegt, sofern der Mosfet eingeschaltet ist. Dies bedeutet, dass er optimal durchgeschaltet ist und sich nicht in einem Linearbetrieb befindet. Wäre dies der Fall, würde viel mehr Wärme freigegeben werden und nicht genügend Strom zwischen Drain und Source durchgelassen werden, was einen schlechteren Ladewirkungsgrad ergeben würde.[27]

4.6.1 Schaltung

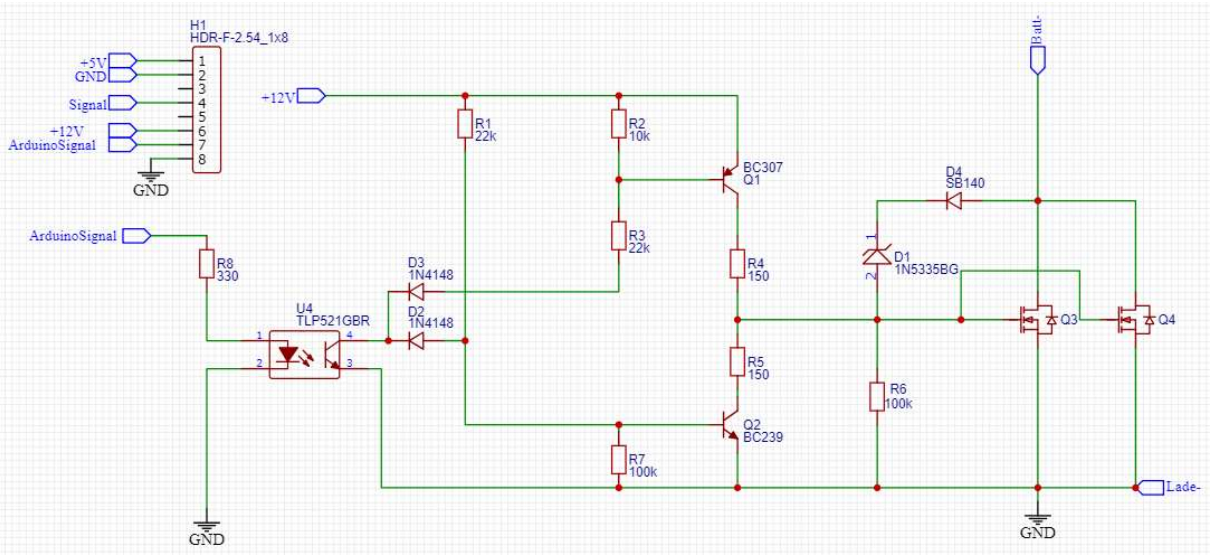


Abbildung 4-25: Schaltplan Ansteuerung Mosfets

4.6.2 Platine

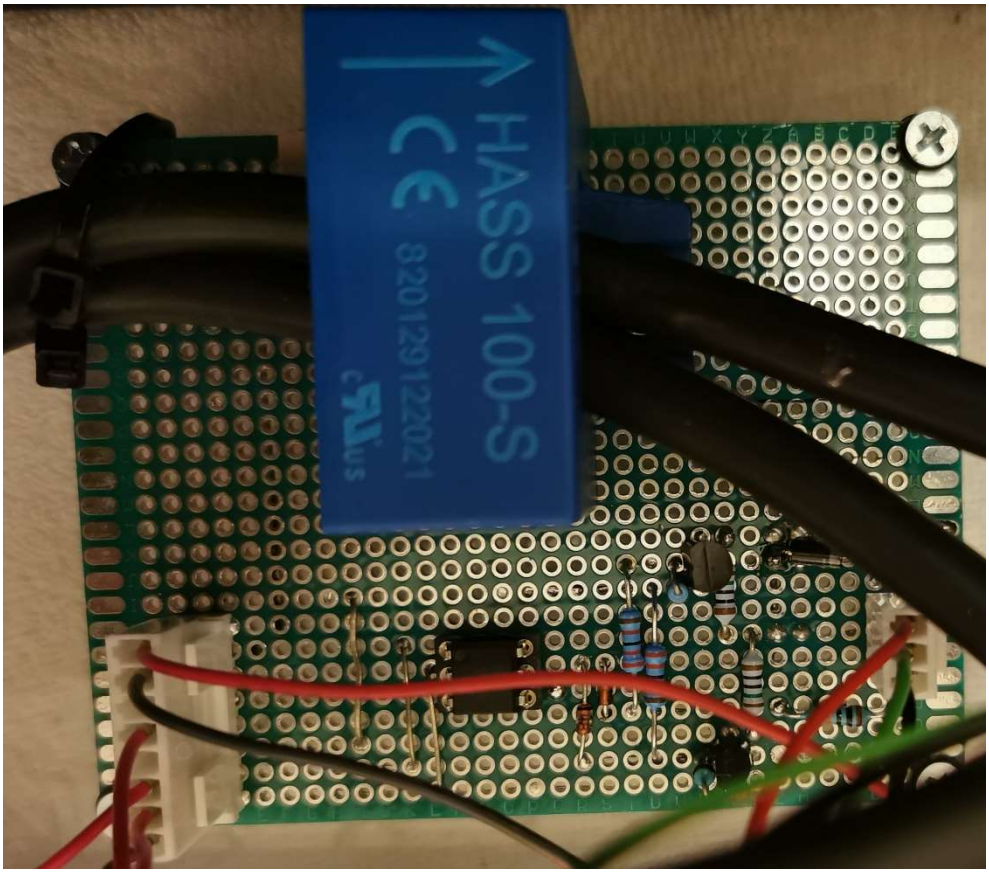


Abbildung 4-26: Reale Platine

4.7 Verteilende Platinen

Um Zellspannungen und Temperaturmesswerte entgegen zu nehmen, werden Phönix Stecker an den Platinen befestigt, um die Verbindungen mittels Plug and Play herzustellen oder zu lösen. Das hat den Vorteil, dass eine Verpolung ausgeschlossen werden kann. Um eine weitere Sicherheitsstufe zu implementieren, wird jede Verbindung von 750 mA Sicherungen gestützt.

4.7.1 Schaltungen

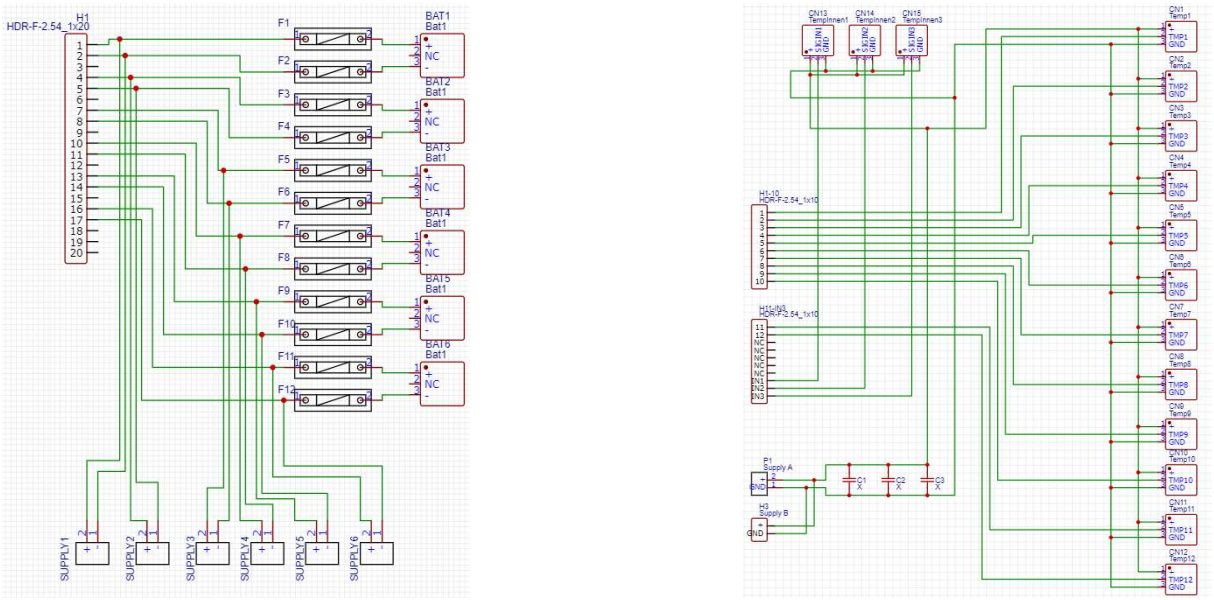


Abbildung 4-27: Zellspannung Zuleitung (links) Temperaturerfassung (rechts)

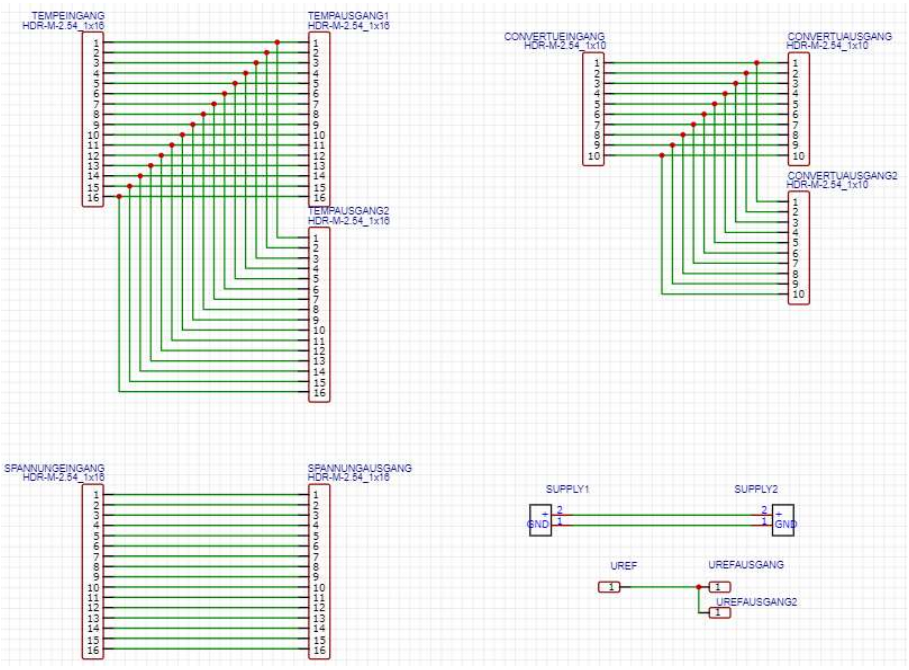


Abbildung 4-28: Verteiler Platine

4.7.2 Platinen

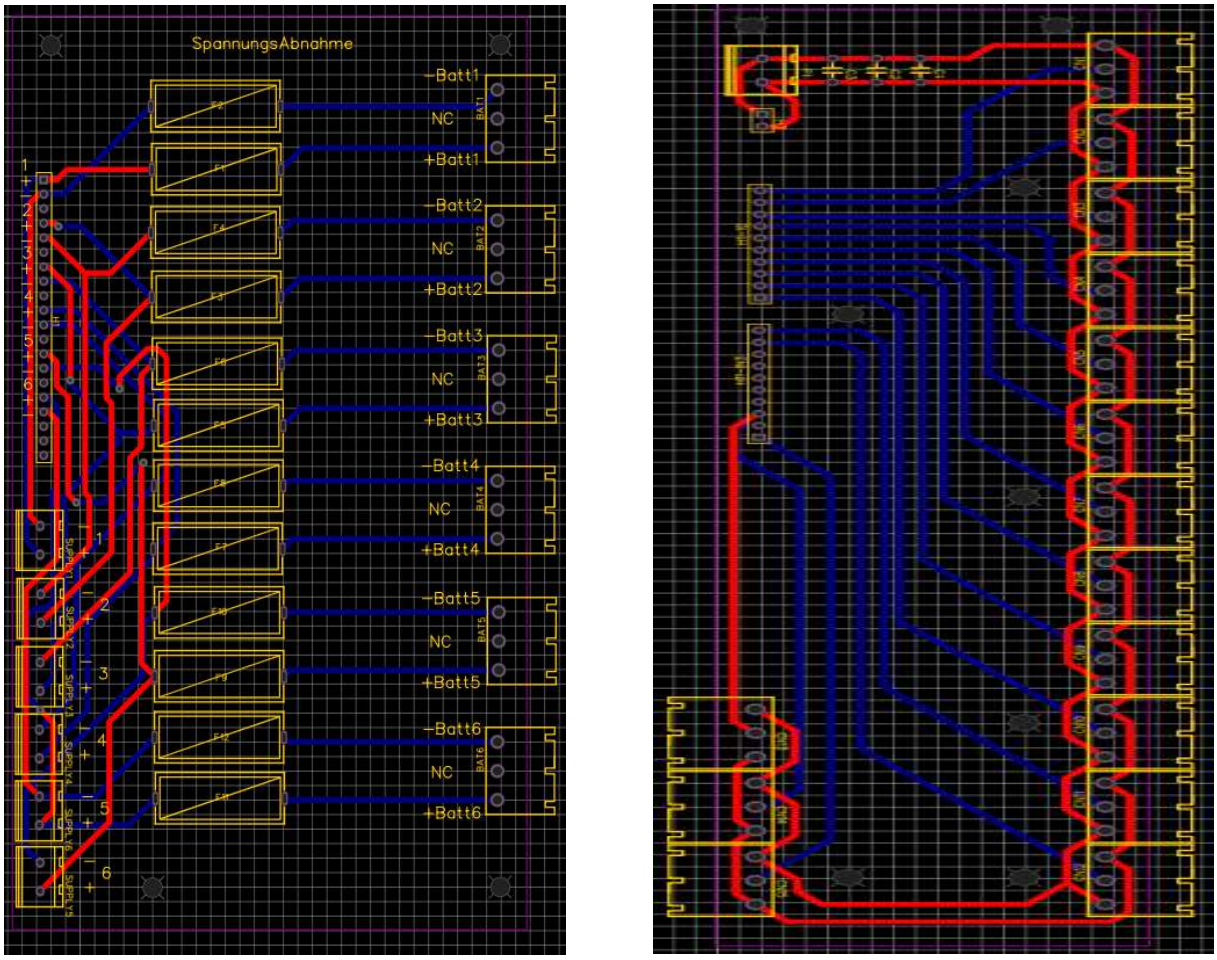


Abbildung 4-29: Routing Zellspannung Zuleitung (links) Temperaturerfassung (rechts)

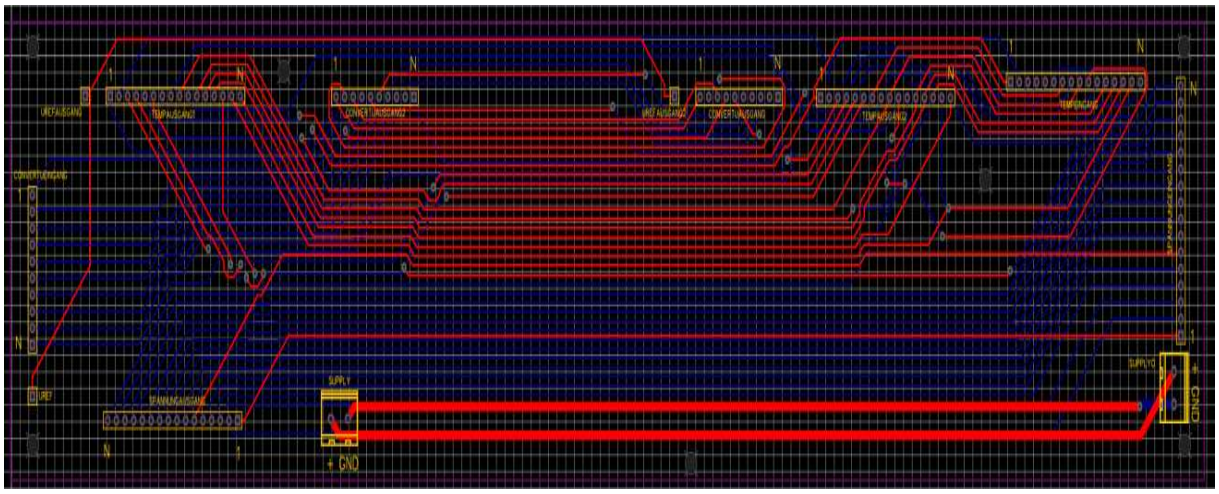


Abbildung 4-30: Routing Verteilerplatine

Abbildung 4-31: 3D Ansicht Zellenspannung Zuleitung(links) Temperaturerfassung(rechts)

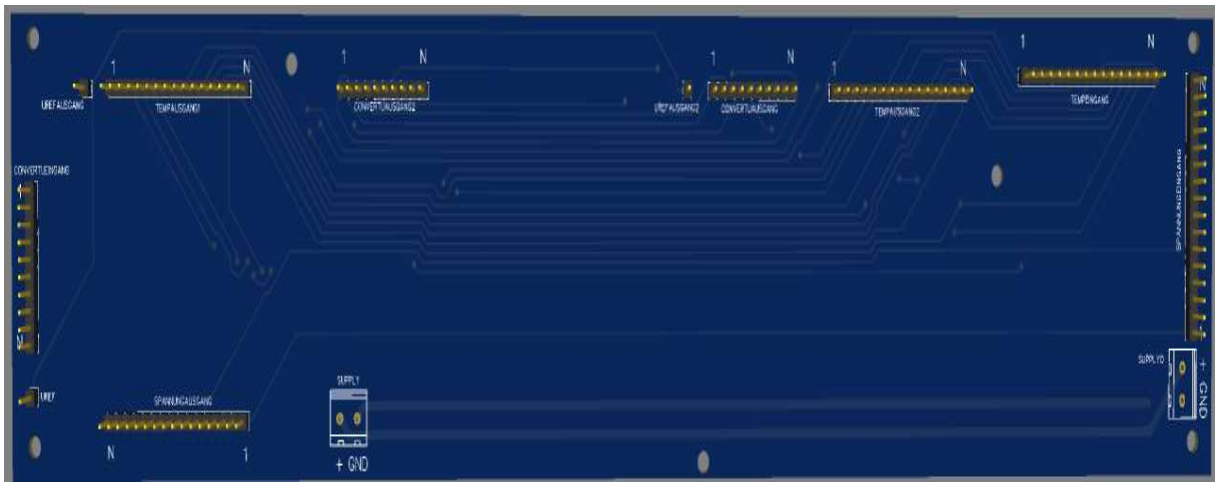


Abbildung 4-32: 3D Ansicht Verteilerplatine

Abbildung 4-33: Platine Zellenspannung Zuleitung (links) Temperaturerfassung (rechts)

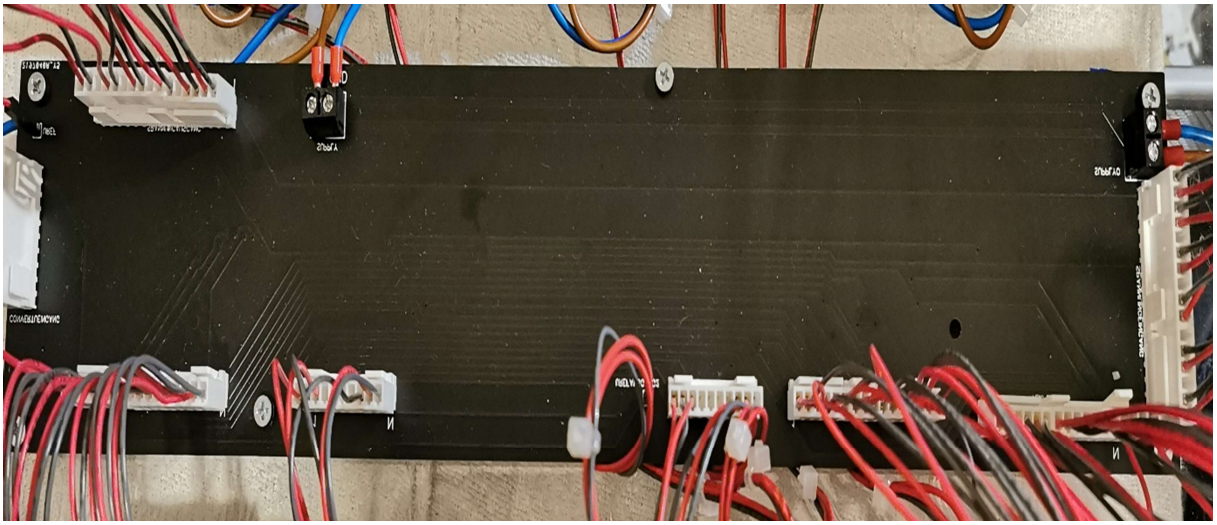


Abbildung 4-34: Platine Verteilung

5 Gehäuse

Damit die Baugruppen vor Fremdeinwirkung geschützt sind, wird ein Gehäuse aus Plexiglas in Kombination mit 3D-gedruckten Elementen konstruiert. Das Acrylglas wird als Bodenplatte und als Deckel verwendet. Die Baugruppen werden nicht verklebt. Der Zusammenhalt wird durch Schraubverbindungen garantiert. Dafür werden M3 und M4 Schrauben verwendet, welche in die dafür vorgesehenen Gewindeeinsätze geschraubt werden. Während des Balancings entsteht Hitze, welche mit Hilfe von 6 Lüftern abgeführt wird. Die Lüfter sind in einer “Push - Pull“ Konfiguration ausgerichtet. Es soll außerdem Öffnungen für Platinen geben, die Messwerte entgegennehmen und für den Stromanschluss bzw. dem Ein-/ Ausschalter. Halterungen für die beiden Netzteile werden ebenfalls gedruckt. Da die Bodenplatte viele Schraubverbindungen enthält, sollen Plastikfüße die Auflagefläche schützen. Geplant wird das Gehäuse mit “Fusion 360“ und gedruckt wird dieses mit einem Prusa i3 mk3s, welcher als Slicersoftware den Prusaslicer verwendet.

5.1 3D-Planung

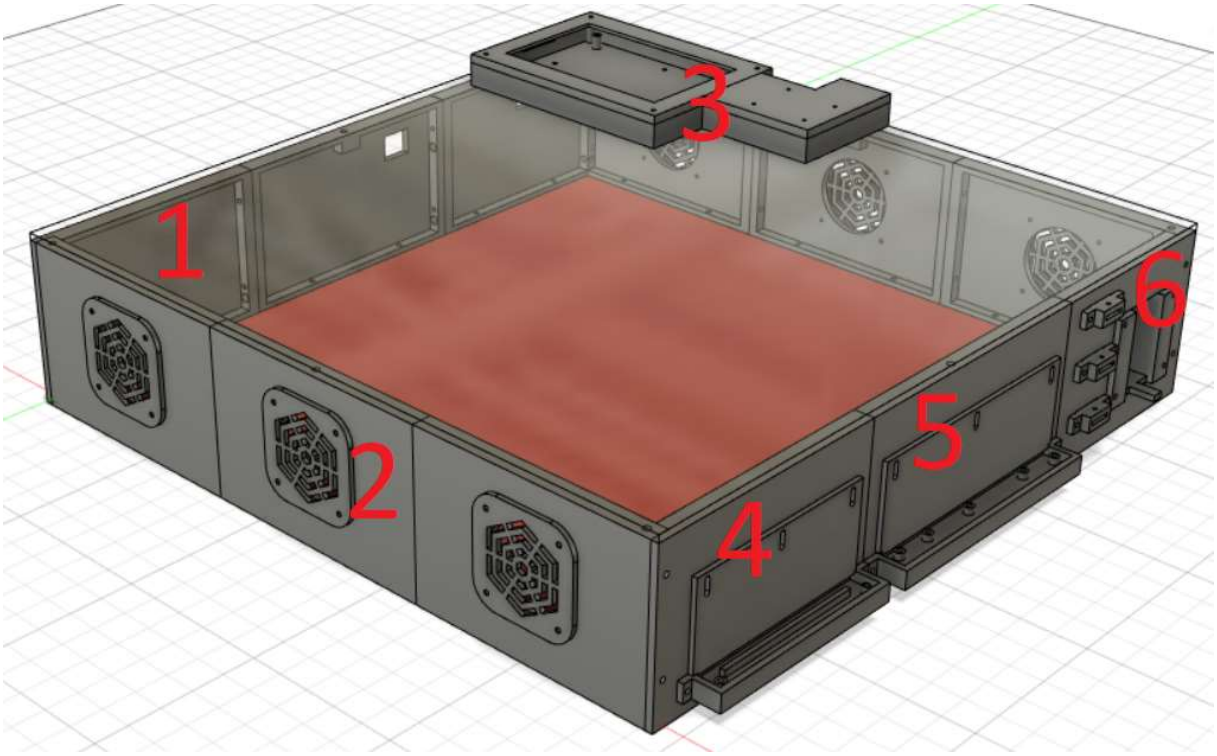


Abbildung 5-1: Gehäuse Seitenansicht

Die Maße des Projektes betragen 612 x 612 x 130 mm. Die Plastikteile werden aus schwarzem PLA (Polylactide) gedruckt. Dieses Material wird mit 210 °C aufgetragen. Im ausgehärteten Zustand ist das Plastik bis ca 60 °C stabil. Anschließend fängt das Material an, weich zu werden und es kann bei mechanischer Belastung zu bleibenden Verformungen kommen. Um das Projekt fertigzustellen, werden 2.8 kg PLA verdruckt.

1	Acrylglas Deckel (6 mm dick)
2	Lüftergitter
3	Touchdisplay Halterung
4	Platinenöffnung - Spannungserfassung
5	Platinenöffnung - Temperaturmessung
6	Kühlkörperöffnung + Schraubkontakte

Tabelle 9: Gehäuse Seitenansicht

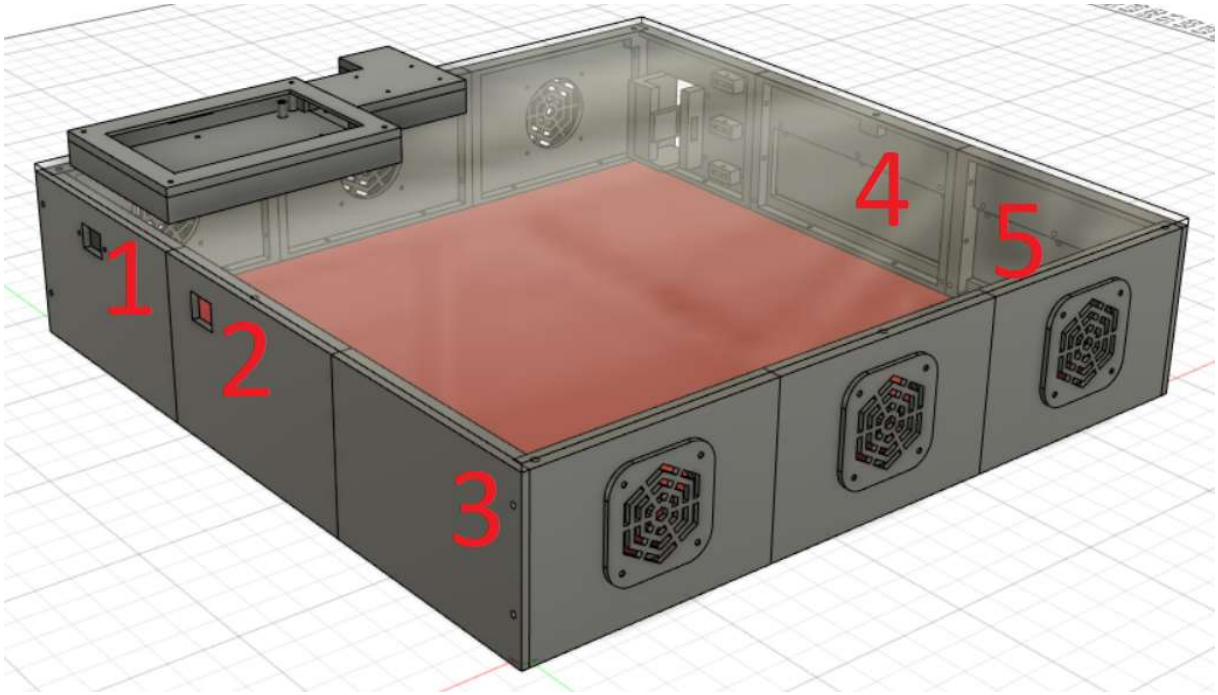


Abbildung 5-2: Gehäuse zweiter Winkel

1	Kaltgeräteanschluss
2	Ein-/ Ausschalter
3	Seitliche Schraubverbindungen
4	Abdeckung Temperaturplatine
5	Abdeckung Spannungserfassungsplatine

Tabelle 10: Gehäuse zweiter Winkel

Die Abdeckungen der einzelnen Platinen sind in der Höhe verstellbar (10 mm auf- und abwärts), um bei Bedarf eine Dichtung anbringen zu können.

5.1.1 Netzteilhalterung

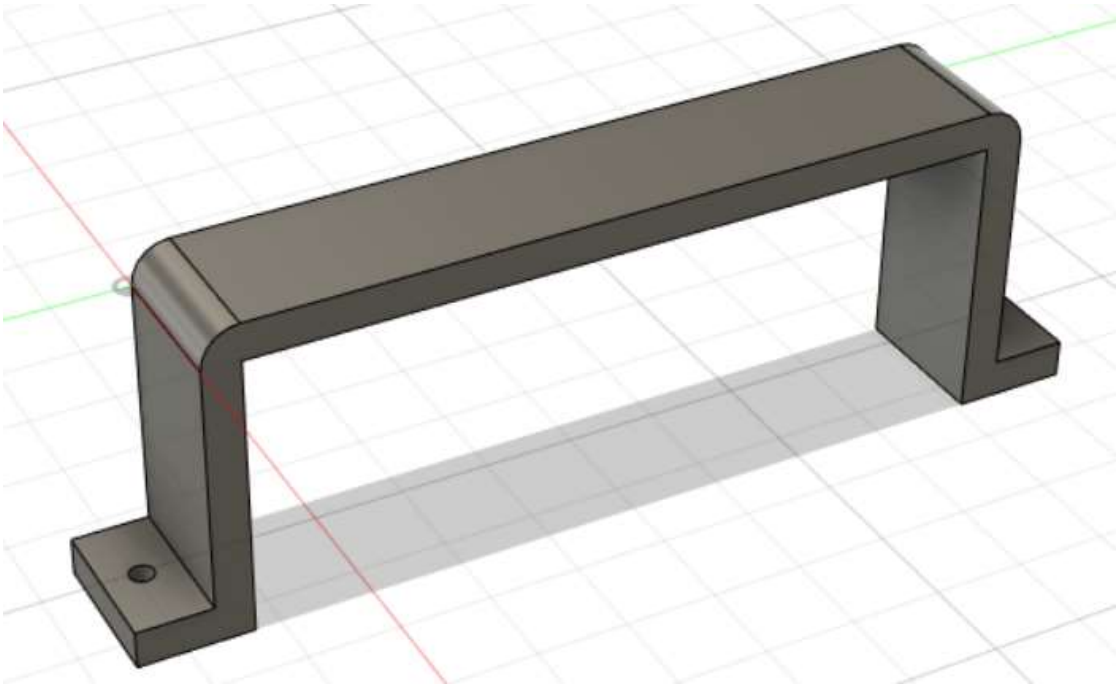


Abbildung 5-3: Netzteilhalterung

5.1.2 Bohrlehre

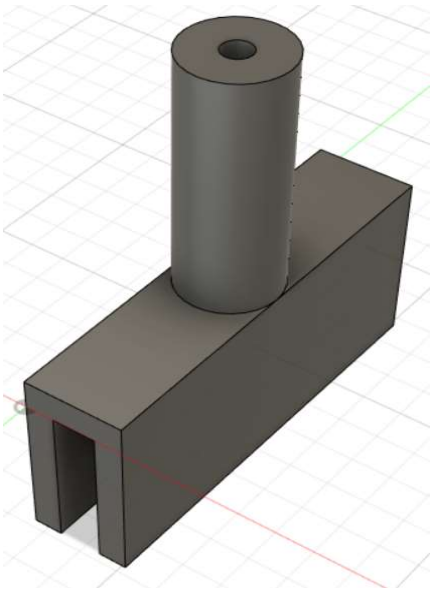


Abbildung 5-4: Bohrlehre

Um die Gewindeeinsätze am Acrylglas zu befestigen, muss ein 4 mm breites Loch in eine 6 mm dicke Fläche gebohrt werden. Da Acrylglas durchsichtig ist und eine genaue Bestimmung der Mittellinie schlecht möglich ist, wurde diese Bohrlehre entwickelt. Sie wird mit der breiten Öffnung auf die Platte gesteckt. Durch die runde Öffnung wird der Bohrvorgang gestartet. Nach meheren Bohrungen ist diese Bohrlehre unbrauchbar, da das Plastik im Führungsloch abgetragen wird und somit keine genaue Zentrierung möglich ist.

6 Software

Für die verschiedenen Mikrocontroller werden zwei unterschiedliche Programmiersprachen verwendet. Der Code für die beiden Arduinos wird in C geschrieben, für den Raspberry wird Python verwendet. Die verwendete Linux Distribution für den Raspberry ist Raspbian, wobei eigene Funktionen bei der Inbetriebsetzung aktiviert wurden (GPIO, UART, SPI). Um eine ferngesteuerte Programmierung zuzulassen, wird mit Hilfe der Windows Applikation “Remote Desktop“ auf den Raspberry zugegriffen. In der Endanwendung werden alle Eingaben über den 7 Zoll Touchscreen getätigt. Für den Notfall ist eine Bluetooth Maus-Keyboard Kombination verbaut.

6.1 Graphical User interface

Um die Bedienung des Projektes zu erleichtern, wird der Zugriff auf die verwendeten Programme auf mehrere Arten ermöglicht. Einerseits ist eine einfache Bedienung über den Touchscreen gegeben, andererseits ist ein Zugriff mittels Tastatur und Maus vorgesehen. Für eine Fernwartung wird ein “Remote Desktop Client“ verwendet.

Für die Bedienung mittels Touchscreen wird ein Graphical user interface eingerichtet, welches alle wichtigen Werte anzeigt und mehrere Möglichkeiten bereitstellt, um in das laufende Programm einzugreifen (Buttons).

Für die Programmierung der grafischen Anzeige wird “Tkinter“ verwendet. Es ist ein Programmtool von Python, welches ermöglicht verschiedene Elemente grafisch darzustellen. Dazu gehören Elemente wie Label, Skalen, Eingabefenster, Buttons und viele andere.

Die Platzierung der einzelnen Elemente wird über ein „grid“ System festgelegt. Es gibt die Möglichkeit, absolute oder relative Positionen anzunehmen. Der Vorteil von relativen Positionen ist die Tatsache, dass bei Änderung der Skalierung des Fensters die Buttons sich relativ dazu verhalten und nicht verschwinden können. Da bei einer ungewollten Bewegung alle wichtigen Buttons verfügbar sein sollen, wurde das “grid“ System verwendet.

Nach der Erstellung eines Tkinterobjektes werden die festgelegten Buttons und co. geladen und angezeigt. Um das Programm durchlaufen zu lassen, ist der Befehl “objektname.mainloop()“ notwendig. Die Problematik bei dieser Programmzeile wird bei der Verwendung von “time“ Befehlen (time.sleep(x)) ersichtlich. Es kommt zu Fehlverhalten der grafischen Oberfläche. Das liegt daran, dass die Codezeile mainloop() eine Methode aufruft, welche eine nicht endende Schleife ausführt, die alle Anzeigen und Buttons aktualisiert.

Dieses Verhalten ist im “main“ Programm nicht erwünscht, weil es die einzelnen Prozesse beeinflusst und somit keinen reibungsfreien Ablauf ermöglicht. Um diesen Prozess trotzdem funktionsfähig zu halten, wird die Methode “objektname.update_idletasks()“ bzw. “objektname.update()“ verwendet, welches eine einmalige Aktualisierung durchführt. Um eine durchgehende Aktualisierung zu bekommen, wird ein eigener Thread gestartet, welcher diese Methoden in Dauerschleife ausführt.

6.1.1 Anzeige der Grundeinstellungen

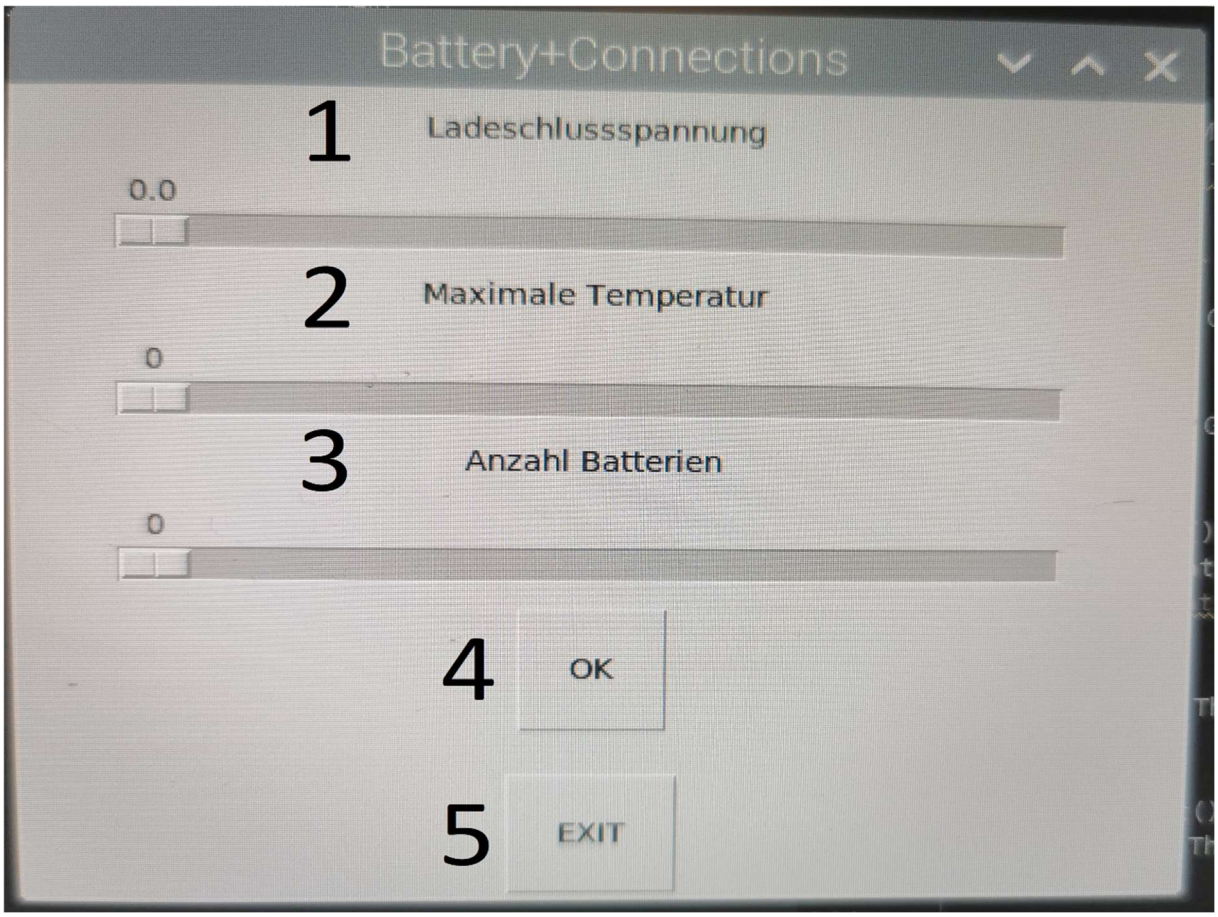


Abbildung 6-1: Tkinter Grundeinstellungen

1	Ladeschlussspannung
2	Maximal zulässige Zellentemperatur
3	Anzahl der angeschlossenen Zellen
4	Akzeptanz der Werte
5	Beendigung des Programms

Tabelle 11: Tkinter Grundeinstellungen

Im ersten Schritt wird die Ladeschlussspannug der verwendeten Zellen festgelegt. Hierfür werden Werte von 0 bis + 5 V akzeptiert und die Skala teilt sich in 0.1 V Schritten. Die maximal zulässige Temperatur kann Werte von 0 bis 100 °C annehmen.Die Temperatur kann in 5 °C Schritten gewählt werden. Die Anzahl der angeschlossenen Zellen können mit Werten von 0 bis 6 festgelegt werden und die Schrittweite beträgt 1. Exit beendet das Programm und der Button Ok führt den nächsten Programmschritt aus und es werden die ausgewählten Werte gespeichert.

6.1.2 Mainwindow

Sofern die Grundeinstellungen erledigt sind, werden die ausgewählten Daten gespeichert und an beide Arduinos geschickt. Bei erfolgreicher Übertragung wird das Hauptfenster geöffnet.

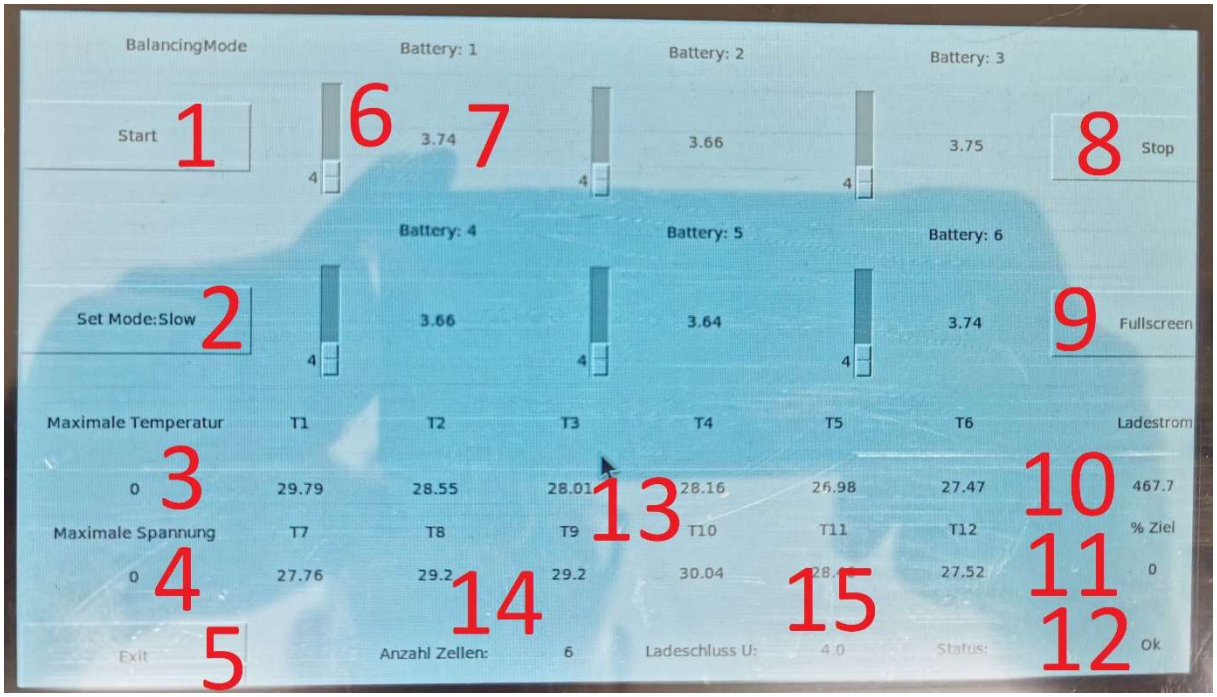


Abbildung 6-2: Hauptfenster GUI

1	Start des Balancing-/ Ladevorganges
2	Ändern Fast- / Slowmode
3	Anzeige aktueller Maximaltemperatur
4	Anzeige aktueller Maximalzellenspannung
5	Beenden des Programms
6	Anzeige Zellenspannung Skala gerundet
7	Anzeige Zellenspannung Absolutwert [v]
8	Stopp des Balancing-/ Ladevorganges
9	Ändern Windowed Mode oder Fullscreen
10	Anzeige Spannung des Stromsensors [Bit]
11	Anzeige Fortschritt des Ladevorganges [%]
12	Anzeige Zustand System
13	Temperaturen der einzelnen Zellen
14 & 15	Anzeige Grundeinstellung

Tabelle 12: Hauptfenster GUI

Bei Betätigung des Startbuttons wird ein Highsignal an einen Pin des Safetyarduino gelegt, um den Lade-/ Balancinvorgang zu starten. Um den Prozess zu stoppen, wird der Button “Stop“ gedrückt und so wird das Highsignal in ein Lowsignal geändert, was zu einem Stopp aller Vorgänge führt.

Mit dem Button “Set Mode“ wird zwischen den Modi langsames und schnelles Balancings gewechselt. Im Falle des langsamen Balancings wird der 100 Ohm Widerstand an die Zelle gelegt. Die Signalleuchte ist die rote LED und, sofern ein schnelles Balancing gewählt wird, schaltet sich stattdessen ein 10 Ohm Widerstand dazu und eine blaue LED leuchtet.

Die Angaben von “Maximale Temperatur“, “Maximale Spannung“ und “% Ziel“ sollen die jeweiligen Maximalwerte bzw. den Fortschritt des Ladevorganges zeigen.

Die einzelnen Skalen zeigen einen gerundeten Spannungswert an. Rechts davon befindet sich der Absolutwert der Zellenspannung der angegebenen Zelle.

Das Programm ist standardmäßig im Vollscreenmodus. Um auf den Windowed Modus zu wechseln, wird beim Drücken des Buttons “Fullscreen“ der Vollscreenmodus gewechselt. Dieses Feature wurde in den Code aufgenommen, um im Fehlerfall auf das Hauptprogramm zugreifen zu können, welches im Hintergrund läuft. Es bietet die Möglichkeit, die Beendigung des Programmes zu erzwingen, sofern es sich aufgehängt hat.

Das Label “Ladestrom“ gibt den Spannungswert des Stromsensors in der Einheit [Bit] aus. Dies hat den Vorteil, im Nachhinein eine passende lineare Regression durchführen zu können, um die genauen Stromwerte angeben zu können.

Die Anzeige “Status“ gibt Information über den aktuellen Zustand des Programmes. Diese Anzeige wäre im Falle einer fehlerhaften Übertragung der Grundeinstellungen an die beiden Arduinos nicht ersichtlich.

Die einzelnen Temperaturen werden mithilfe von TMP36 Sensoren erfasst und nach Konversion von Spannungswerten zu absoluten Temperaturwerten angegeben. Aufgrund der Ungenauigkeit der 10 Bit Analog-/ Digitalconverter und wegen Einstreuungen der Leitungen und Toleranzen der Sensoren, kommt es zu relativ starken Schwankungen der Temperaturwerte. Die einzelnen Werte können ± 3 °C variieren. Um jedoch eine grobe Einschätzung der Temperaturwerte zu bekommen, sind diese Messwerte absolut ausreichend. Die angegeben Temperaturwerte werden durch die zweite Sicherheitsstufe, dem Safety Arduino, ebenfalls ausgewertet. Die Werte des Arduinos sind genauer und weniger fehlerbehaftet, da weniger Einstreuungen auf das System wirken. Dies erklärt sich einerseits durch eine kürze Leitungslänge zu den Sensoren als auch durch ein kompakteres Platinenlayout.[28]

Der Bildschirm wird alle 15 Minuten in den Ruhemodus versetzt, sofern der Touchscreen keine neue Berührung wahrnimmt. Diese Funktion wurde aus Energiespargründen implementiert.

6.2 Threads

Abhängig von der Programmiersprache oder der Architektur auf der ein Programmcode läuft, werden Programmabläufe unterschiedlich abgearbeitet. Im Falle des Python Programmes auf dem Raspberrry wird Schritt für Schritt der Code durchlaufen. Unterbrechungen können nur durch sogenannte “Interrupts“ auftreten, welche durch Hardware (spezielle Pins) oder durch Software aufgerufen werden können. Im Falle der Interrupts wird das laufende Programm an seinem derzeitigen Punkt unterbrochen und abhängig von der Interruptroutine an einer anderen Stelle fortgeführt. Wurde die Interruptmethode beendet, wird das Programm dort fortgesetzt, wo es vor dem Interrupt unterbrochen wurde.

Threads bieten die Möglichkeit, mehrere Prozessschritte gleichzeitig auszuführen. Dies hat den Vorteil, Berechnungen schneller durchzuführen bzw. wichtige Prozesse gleichzeitig laufen zu lassen. Wie im Abschnitt “Graphical user interface“ bereits beschrieben, ist das Durchlaufen ohne Threads ungeeignet. Es werden insgesamt drei unterschiedliche Threads neben dem Mainthread gestartet.

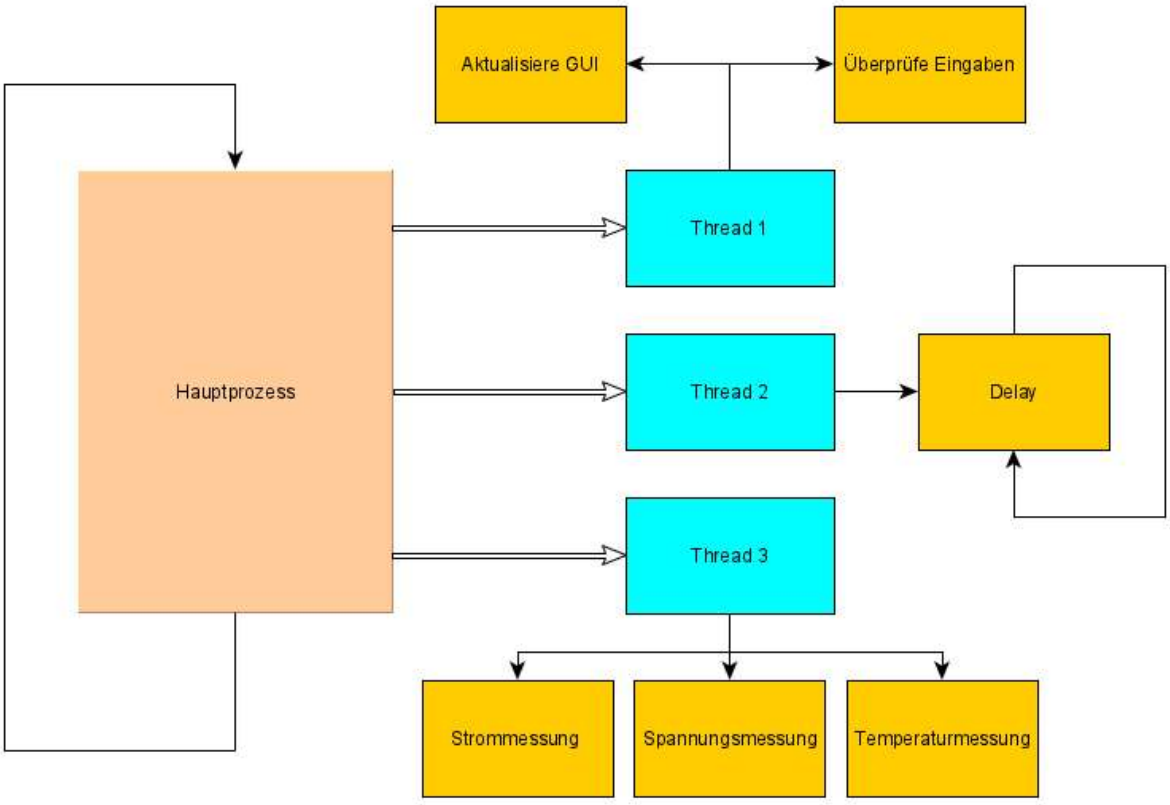


Abbildung 6-3: Threads

Die einzelnen Threads werden in Abbildung 6-3 gezeigt. Thread 1 übernimmt die Aktualisierung und Verarbeitungen von Eingaben. Thread 2 wechselt bei jeder Sekundenänderung eine Variable. Der Grund dafür liegt darin, das eine Implementierung eines Timerinterrupts in Python umständlich ist. Deswegen wird hier eine “dirty solution“ verwendet, um ein ähnliches Ergebnis zu erhalten. Thread 3 aktualisiert alle wichtigen Inputwerte und schreibt die neuen Daten in ein Array, auf das der Hauptprozess zugreifen kann.

6.3 Messwerte Glättung

Da die einzelnen Messwerte über eine Vielzahl von Kabeln geleitet werden, bleibt es nicht aus, dass Einstreuungen das Messsignal verfälschen. Neben Einstreuungen können auch eine schlechte Auslegung des Routings der Platine sowie keine Trennung zwischen digitaler und analoger Masse, instabile Referenzspannungen und eine schlechte Versorgungsspannung zu Fehlern führen.

Abbildung 6-4: Arduino analogRead() Funktion

In Abbildung 6-4 sieht man, wie die konvertierte Analogspannung in einen digitalen Wert “Bit“ ausgegeben wurde. Diese Schwankungen sind auf die oben genannten Fehler zurückzuführen. Um eine genauere Auswertung des Signals zu bekommen, gibt es mehrere Möglichkeiten. Man kann ein Filter vor den Analogeingang schalten, um gegen Störungen vorzugehen oder man schaltet nur einen Glättungskondensator dazu. Eine digitale Möglichkeit ist vorrangig die Konvertierung des “Bit“-Wertes in eine Gleitkommadarstellung.

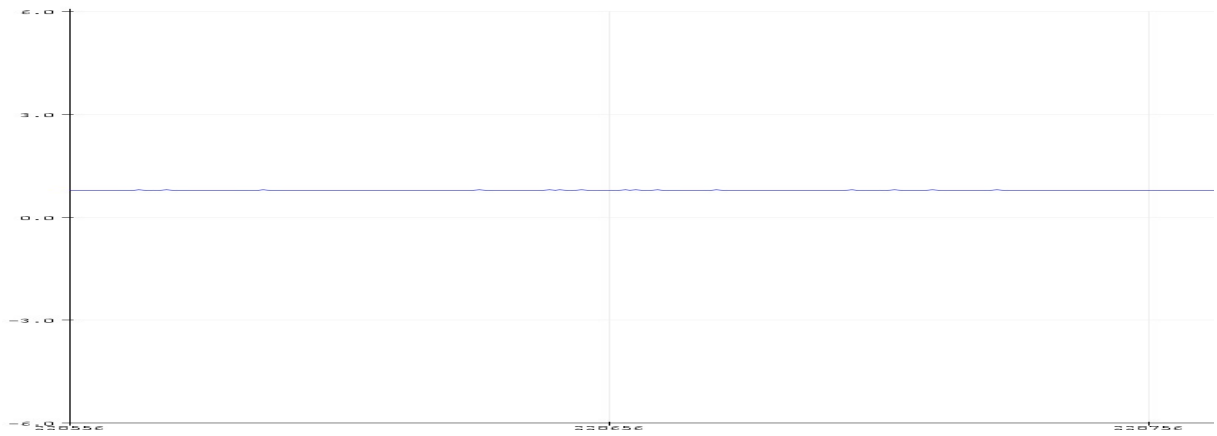


Abbildung 6-5: Umrechnen des "Bit" Wertes in reale Spannung

Wie in Abbildung 6-5 ersichtlich, ist das Signal viel ruhiger. Um noch eine bessere Ausgabe zu bekommen, kann man einen Mittelwert über mehrere Werte bilden.

6.4 Datenerfassung

Alle Messwerte, die während dem Lade-/ Balancingvorgang am Display ausgegeben werden, werden in einem Textdokument auf dem Raspberry Pi gespeichert. Diese Werte werden jede Sekunde aktualisiert und mitgeloggt, um im Nachhinein eine Analyse über den Ladeverlauf erstellen und eventuell defekte Zellen erkennen zu können.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	B1	B2	B3	B4	B5	B6	X	Y	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	TX	TY	TZ	Strom
2	3,32	3,32	3,04	3,04	3	3,01	0	0	26,43	27,17	26,63	27,12	26,23	26,24	27,12	27,32	26,04	25,79	26,48	26,48	0	0	0	506,9

Abbildung 6-6: Format des Dataloggers

Die ersten sechs Werte beziehen sich auf die einzelnen Zellspannungen in [V] mit Ausnahme von X und Y. Diese Werte sind nicht belegt, da die Analog-/ Digitalconverter jeweils 8 Eingänge haben, wovon die Batterien nur 6 Plätze einnehmen.

Im Anschluss werden die einzelnen Zelltemperaturen in [°C] angegeben. TX, TY und TZ sind wieder Ausnahmen. Diese Plätze sind zwar mit der Temperaturerfassungsplatine verbunden, jedoch sind hier keine TMP36 Sensoren angesteckt. Diese leeren drei Plätze können für eine Temperaturwerterfassung im Inneren des Prototyps verwendet werden. Aus Bequemlichkeitsgründen werden diese Kanäle softwaretechnisch auf 0 gesetzt.

Der letzte Messwert gibt den Ladestrom in “Bit“ an. Der Bitwert wurde gewählt, da für eine Umwandlung in den realen Wert eine längere Berechnung notwendig wäre. Sie würde die Rechenleistung des Prozessors unnötig beanspruchen würde. Die realen Werte werden im Anschluss bei der Datenaufbereitung umgewandelt.

Um die Textdateien im Anschluss zu sichern, gibt es mehrere Möglichkeiten. Die Daten können auf einen USB-Stick, eine externe Festplatte oder eine Cloud (z.B. Dropbox) geladen werden.

Da das Programm mit drei Threads und einem Hauptprogramm läuft, könnte man theoretisch mehrere Duzend Messwerte pro Sekunde aufnehmen, jedoch würde die Dateigröße enorm ansteigen,was bei einem externen MicroSD Speicher von 16 GB die Speicherkapazität bald aufbrauchen könnte.

Wenn man ein langsames Balancing über mehrere Tage durchführen möchte, kann man eine externe Festplatte (z.B. SSD) an den Raspberry anschließen. Der Mikrocontroller besitzt vier USB Steckplätze und unterstützt auch den Anschluss eines USB-Verteilers. Man muss nur den Speicherort ändern und überprüfen, ob Schreibrechte verfügbar sind.

7 Testaufbau

Für die Versuchsdurchführung wird ein Yuasa Zellenpack verwendet, welcher in 6S2P verschaltet ist. Die ersten Messwerte der einzelnen Zellen sind in Abbildung 6-6 erfasst worden und es ist ersichtlich, dass sich der Ladezustand des Akkupacks im unteren Bereich befindet.

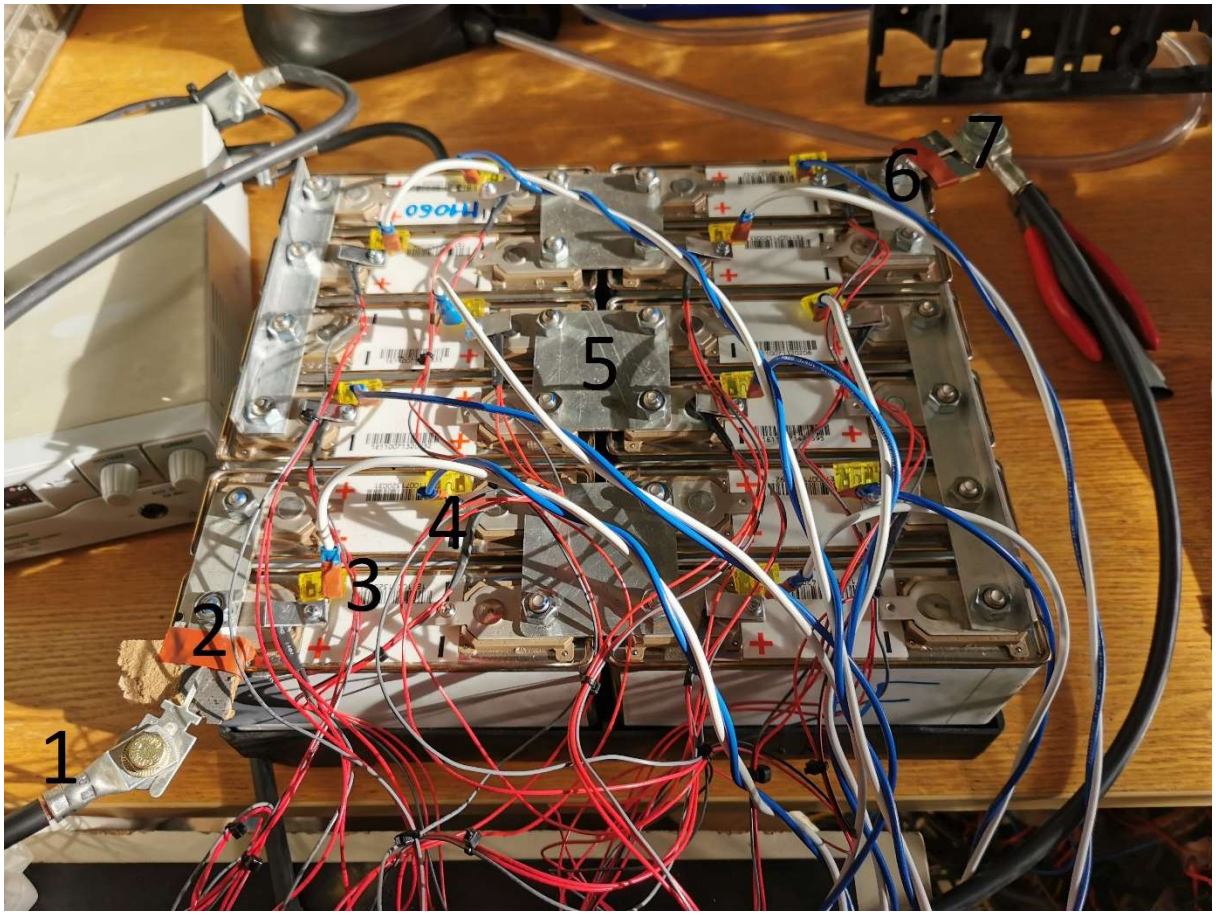


Abbildung 7-1: Testversuch Aufbau

1	Ladepol Plus (+)
2	Sicherung Metall Pluspol
3	Pluspol Zellspannung & Balancing Zelle 1
4	Temperatursensor TMP36
5	Verbindungsstück Aluminium
6	Sicherung Metall Minuspol
7	Ladepol Minus (-)

Tabelle 13: Verkabelung Akkupack

7.1 Ladevorgang 30 Ampere

Um auf Probleme schnell reagieren zu können, wurde als Anlaufphase ein Ladestrom von 30 Ampere gewählt. Das ist eine Sicherheitsmaßnahme, um den Prototypen im Erstversuch nicht voll zu belasten.

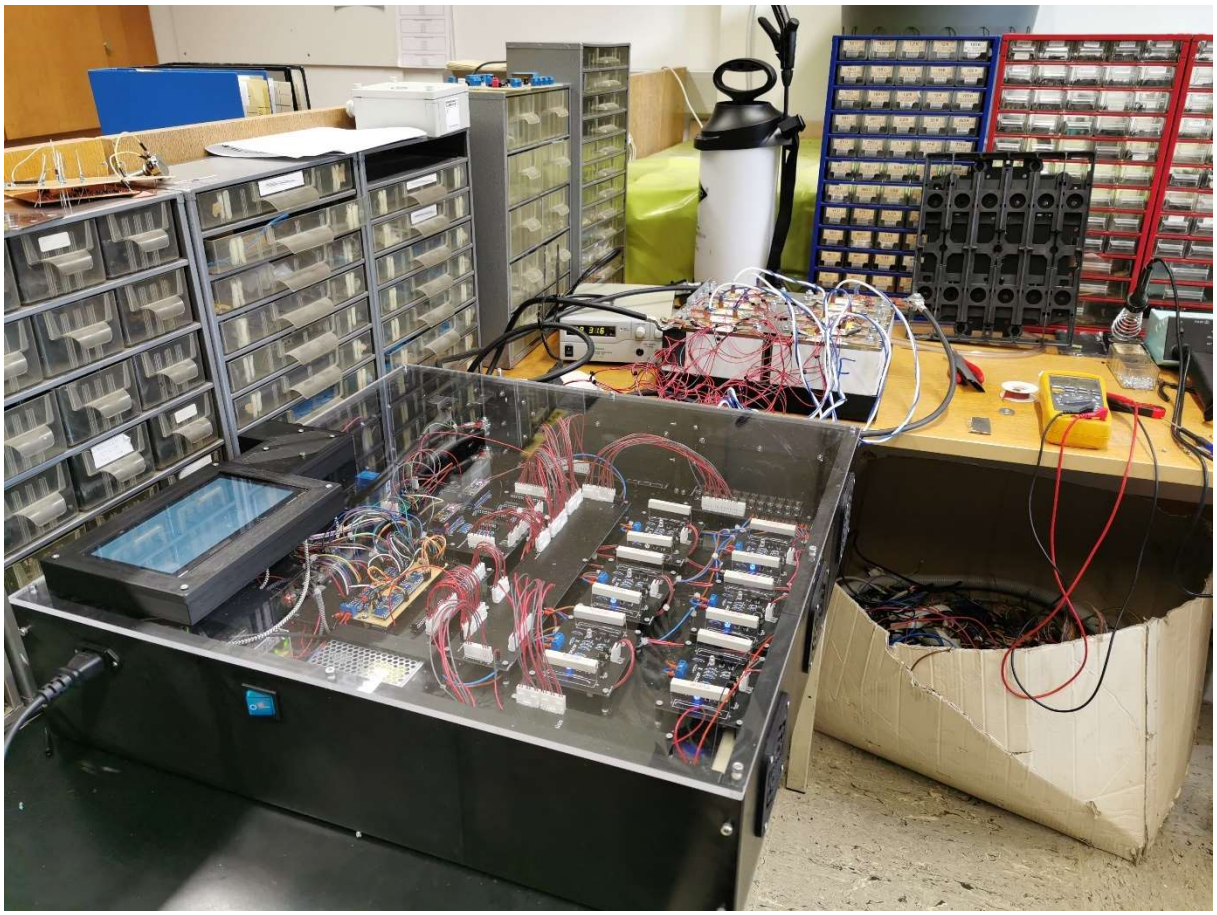


Abbildung 7-2: Versuchsaufbau

Die einzelnen Kabel- und Steckverbindungen wurden fachgerecht hergestellt, wie die folgenden Abbildungen zeigen.[29]

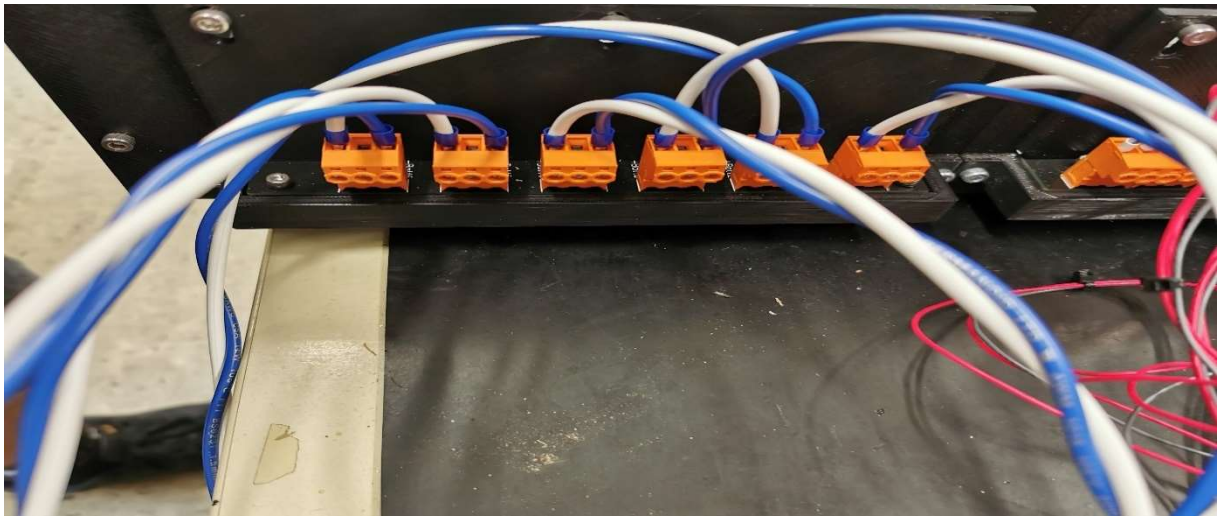


Abbildung 7-3: Verbindung der Zellenspannungen

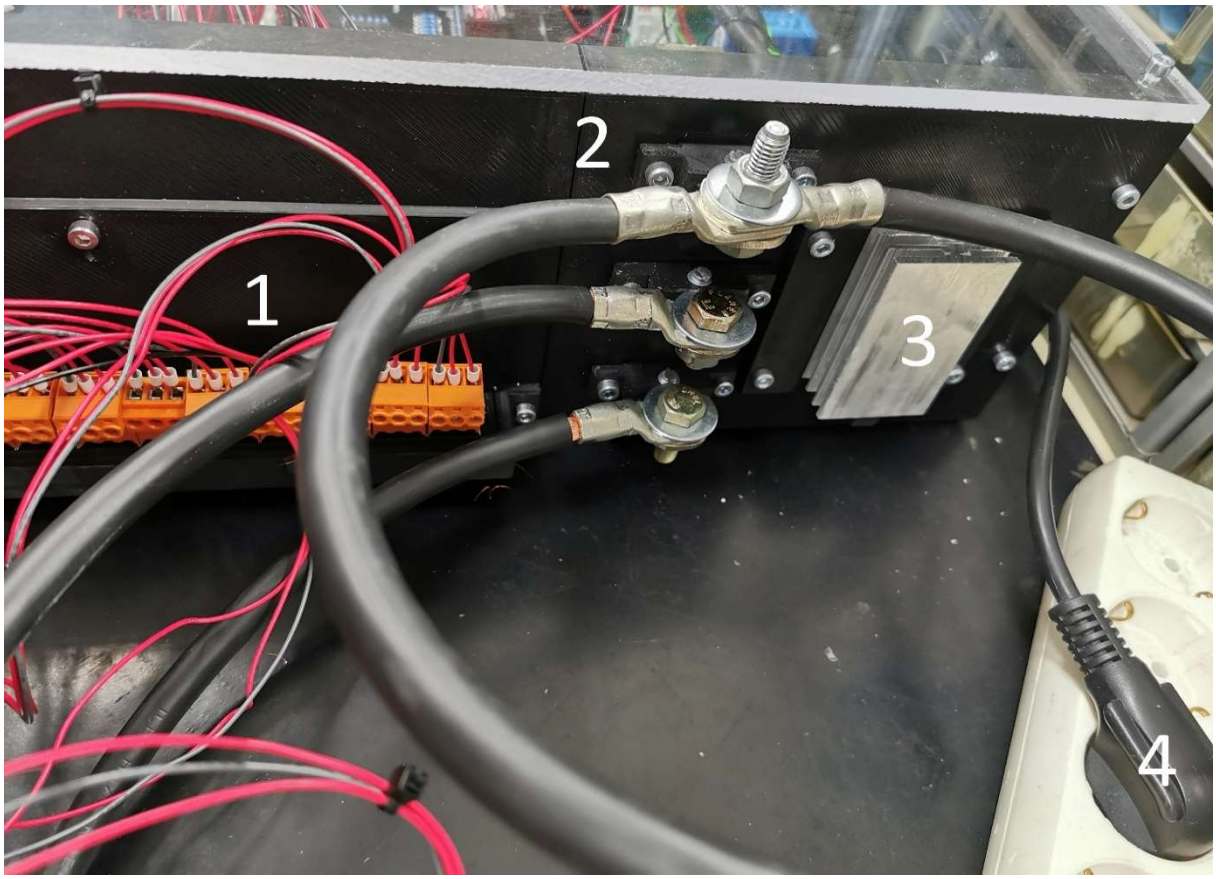


Abbildung 7-4: Restliche Verbindungen

1	Anschluss der Temperatursensoren
2	Anschluss Netzgerät & Zellenpack
3	Kühlkörper (Aluminium)
4	Stromversorgung Prototyp

Tabelle 14: Restliche Verbindungen

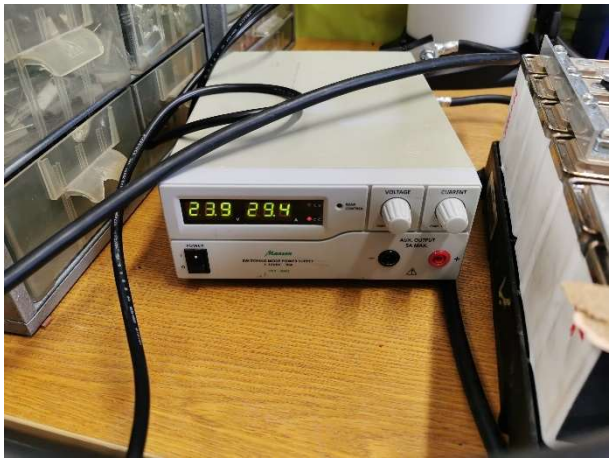


Abbildung 7-5: Netzteil in Betrieb

Wie in Abbildung 7-5 ersichtlich, beträgt der Ladestrom ungefähr 30 Ampere. Die vorderen Ausgänge des Netzgerätes konnten nicht verwendet werden, weil jene eine Strombeschränkung von 5 Ampere besitzen.

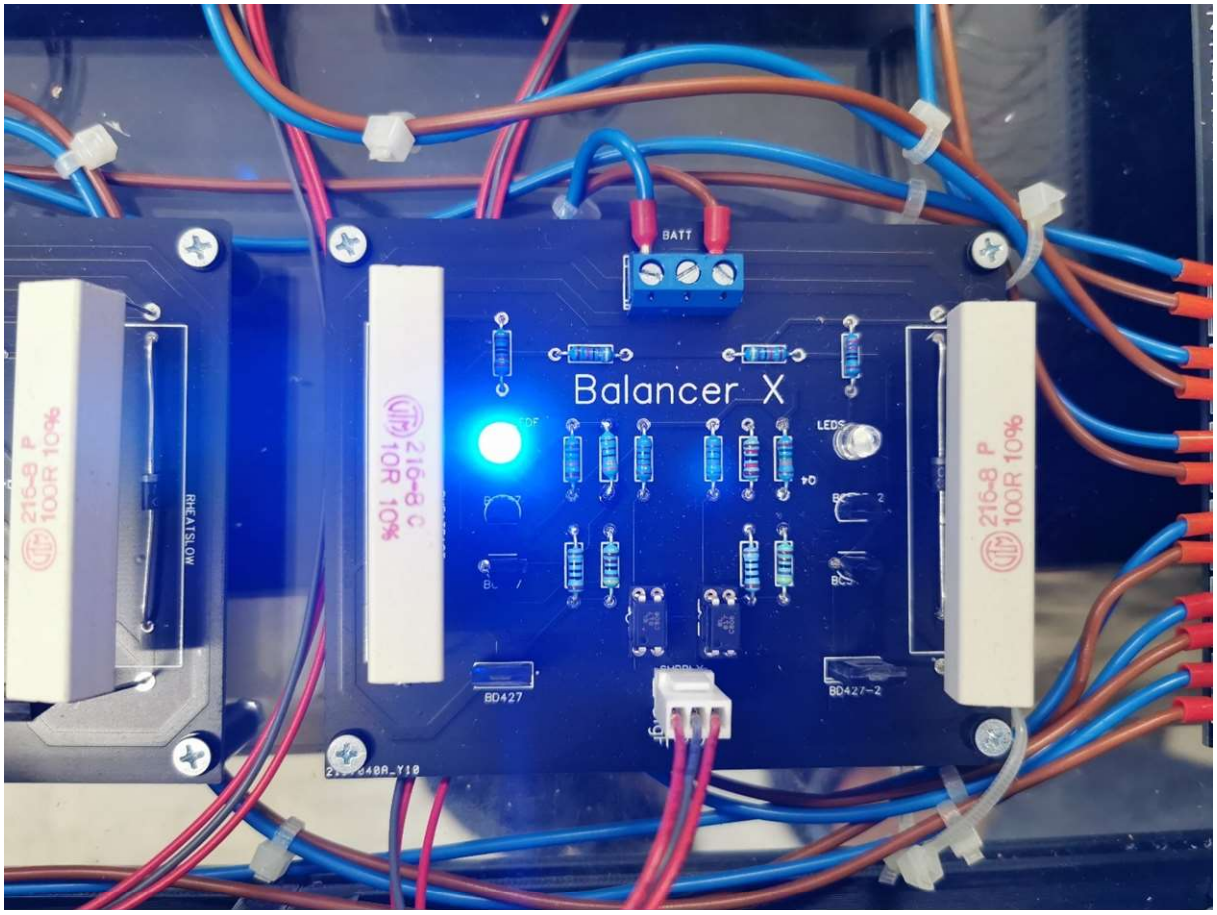


Abbildung 7-6: Balancingvorgang aktiv (schnell)

Im ersten Versuch wurde der Akkumulator im Modus 3 gebalanced und die angegebene Ladeschlussspannung wurde mit +4.0 V festgesetzt worden.

Der Ladevorgang wurde wie festgelegt beendet und die dabei gewonnen Daten wurden gespeichert und per USB-Stick archiviert, um eine anschließende Auswertung zu ermöglichen.

7.2 Ladevorgang 90 Ampere

Im Anschluss an den Ladevorgang mit 30 Ampere wurden alle Funktionen nochmals auf ihre Funktionsfähigkeit überprüft. Um die Zellen mit 90 Ampere laden zu können, waren noch zwei weitere Netzteile erforderlich.

Die Netzteile wurden parallelgeschaltet. Zwischen den einzelnen Netzteilen wurde Plastik als Isoliermaterial verwendet. Dieses Material kann bei ungewollter Lockerung der Verbindungen einen Kurzschluss verhindern.

Da der dreifache Strom eine Belastung für Zu- und Ableitungen sind, müssen die Temperaturen der Kabel optisch überprüft werden.

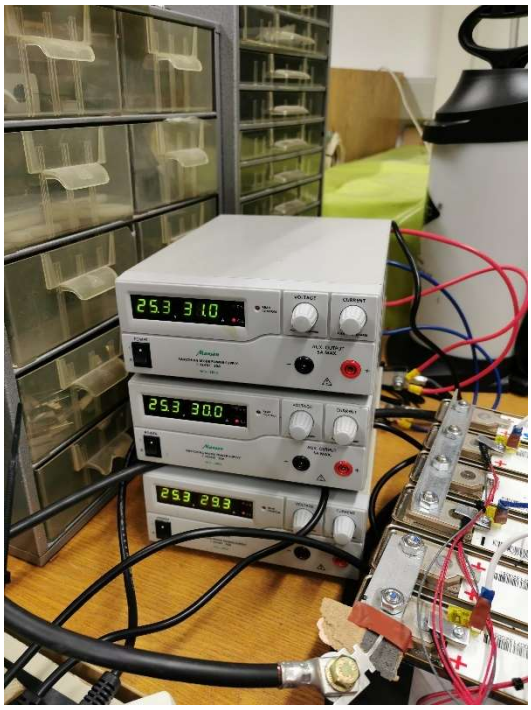


Abbildung 7-7: Ladevorgang 90 Ampere

Der Ladevorgang wurde wieder in Modus 3 durchgeführt. Es wurde von Zeit zu Zeit zwischen schnellem und langsamen Balancing gewechselt, um eventuelle Auswirkungen auf den Ladevorgang mitzuloggen.

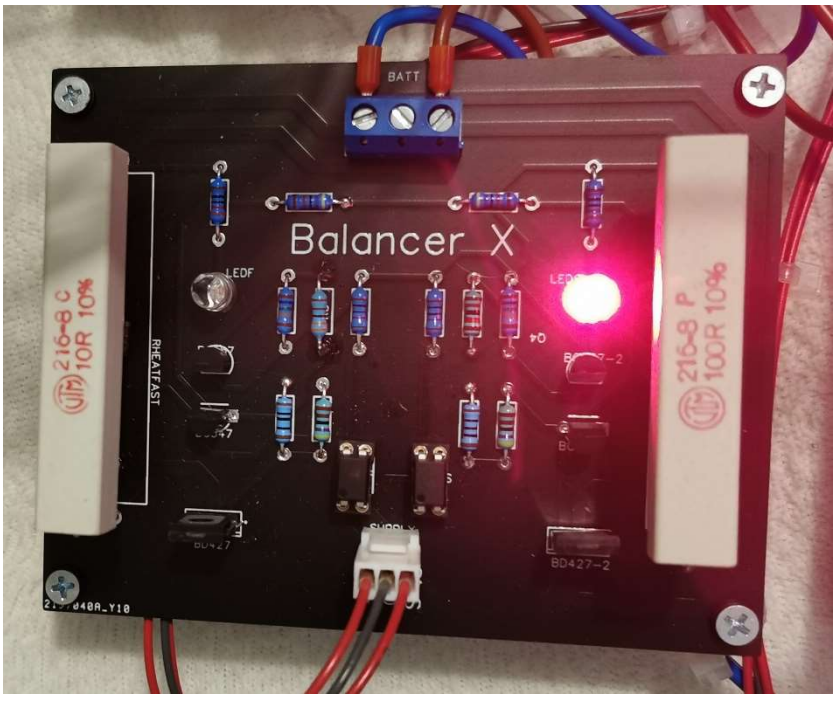


Abbildung 7-8: Balancer aktiv (slow)

Wird der Balancingvorgang auf “slow“ gestellt, wird der Balancerstrom um den Faktor 10 verringert. Als Anzeige dafür leuchtet die rote LED.

7.3 Temperaturabfrage Wärmebildkamera

Wenn man mit hohen Strömen lädt, werden Kabel und Verbingungsstücke sowie Sicherungen stärker belastet. Um ein Fehlverhalten der Materialien zu minimieren, wird regelmäßig deren Temperatur gemessen.

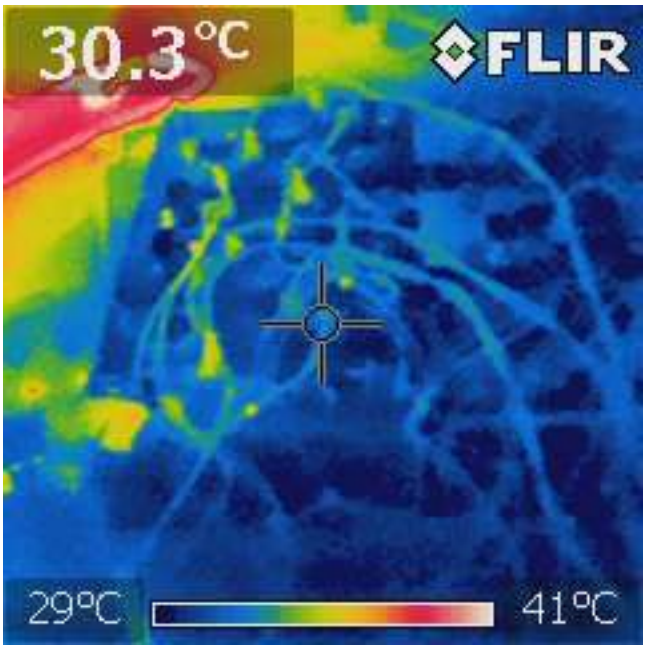


Abbildung 7-9: Temperaturmessung Akkumulator

Wie in Abbildung 7-9 ersichtlich, hat der Zellenpack eine durchgehende Temperatur von ca. 30 °C. Links oben befindet sich ein “Hotspot“, welcher auf die Erwärmung der Sicherung am Pluspol zurückzuführen ist. Auch die restlichen Temperaturerhöhungen lassen sich mit den einzelnen Autosicherungen, die für das Balancing zuständig, erklären. Sofern eine Mutter Erwärmung zeigt, kann das auf zu wenig Drehmoment bei der Montage zurückzuführen sein, da deswegen eine schlechte Verbindung zwischen den einzelnen Zellen gegeben ist.

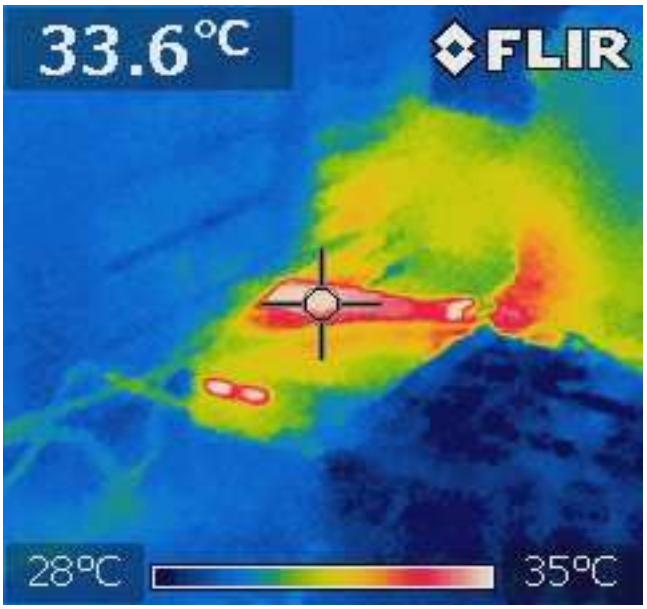


Abbildung 7-10: Temperatur Netzteil

Das Netzteil in Abbildung 7-10 weist auf der Rückseite eine leichte Erwärmung von ca 34 °C auf. Das liegt daran, dass die Leistungselektronik des Netzteiles in der Nähe der Rückseite installiert wurde.

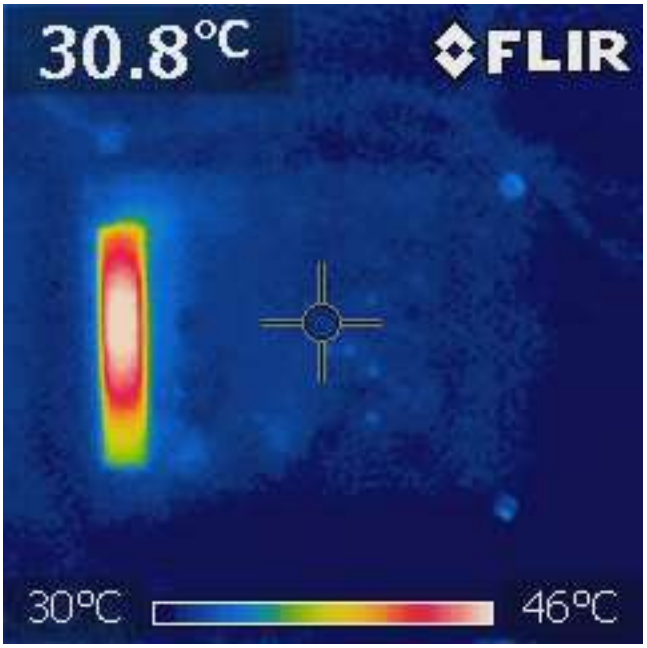


Abbildung 7-11: Balancer aktiv

Wie in Abbildung 7-11 ersichtlich, beträgt die Temperatur des Widerstandes ca 45 °C.

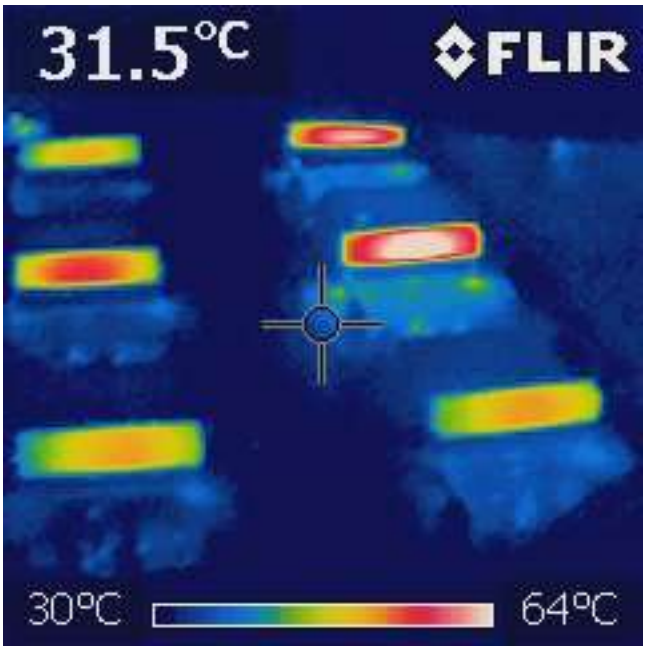


Abbildung 7-12: Temperatur aller Balancer

Abbildung 7-12 zeigt die Temperaturverteilung der einzelnen Balancer. Der Abbildung 6-6 kann entnommen werden, dass die Zellen eins und zwei höhere Zellenspannungen aufweisen als der Rest. Das wird auch in Abbildung 7-12 gezeigt. Da die Wärme stetig durch Lüfter abgeführt wird, kann sich die Innenraumtemperatur des Prototypen nicht so weit erwärmen, dass es das PLA beschädigen könnte.

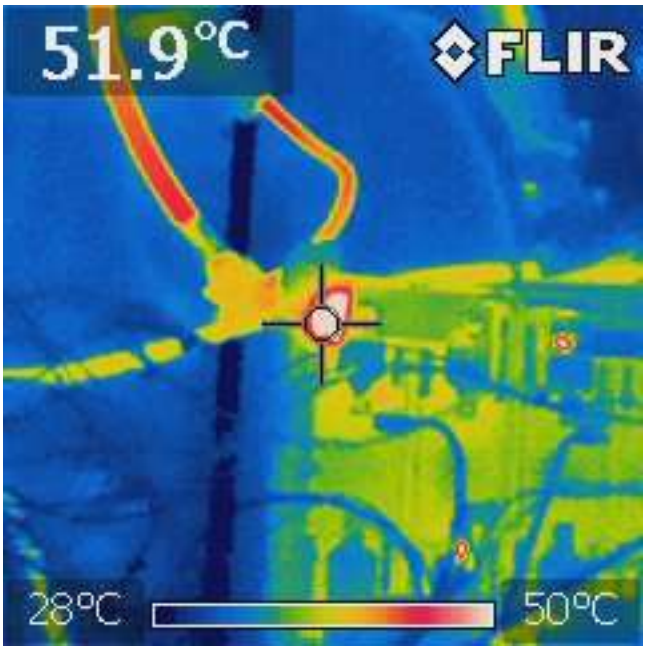


Abbildung 7-13: Temperatur Sicherungen und Kabel

Die Verbindungen, die für den Ladevorgang zuständig sind, erreichen eine Temperatur von ca. 50 °C.

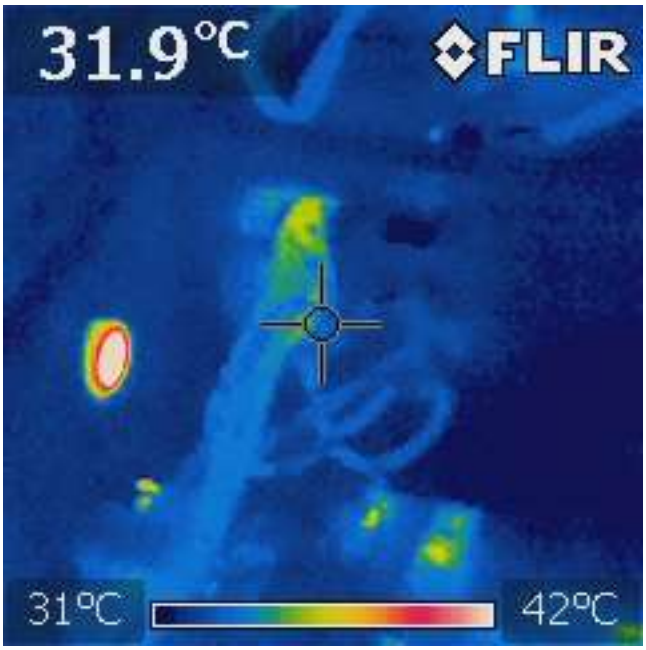


Abbildung 7-14: Temperatur Leistungsschalter

Der Kühlkörper übersteigt in der Innenseite des Prototypen 40 °C nicht. Der Hotspot auf diesem Bild ist der Lüfter. Da die Lüfter gelagert sind, kommt es zu Reibung, bzw. erzeugt die verbaute Elektronik im Lüfter Verluste, was zu einer Wärmeabfuhr führt. Die Ladeleitungen im Inneren sind mit einer Temperatur von ca. 33 °C gut dimensioniert und würden auch mehr Strom verarbeiten können. Bei einer größeren Dimensionierung müsste der Stromsensor ausgetauscht werden, da sein Arbeitsbereich 100 Ampere nicht überschreiten sollte. Einer kurzzeitigen Überbelastung würde er aber standhalten.

7.4 Daten Analyse

7.4.1 Erster Balancingschritt

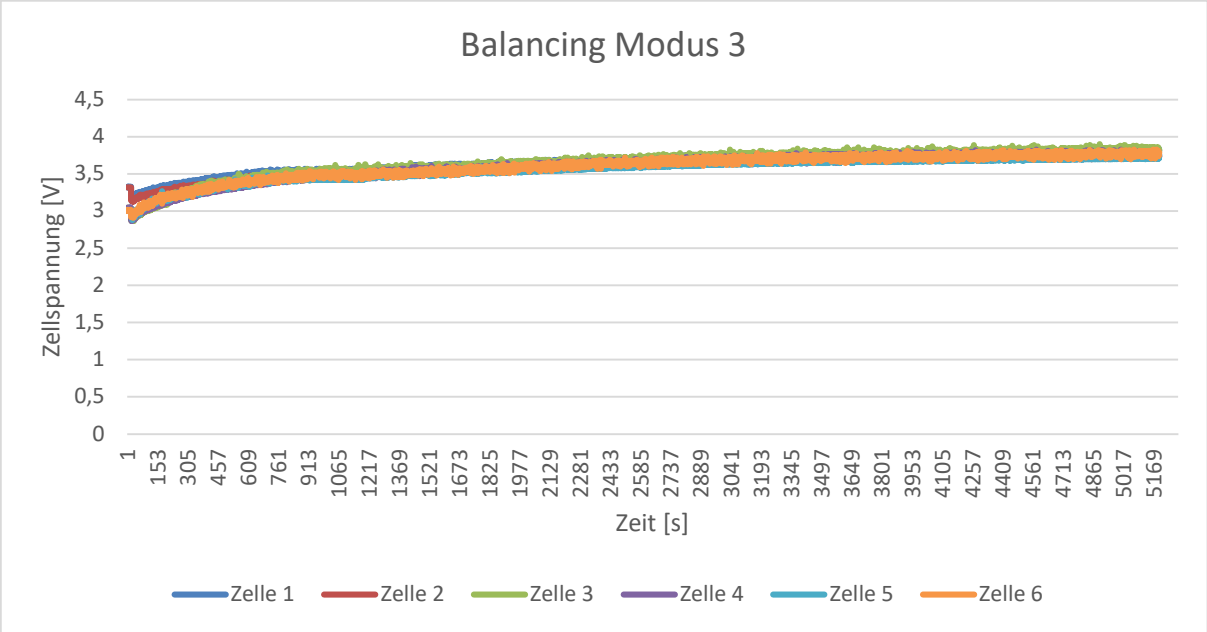


Abbildung 7-15: Zellspannungen Ladeschlussspannung 4 V

Im ersten Programmdurchlauf wurde nicht ganz bis zur Ladeschlussspannung geladen. Dieser Testversuch wurde mit 30 Ampere durchgeführt und diente nur der Funktionsüberprüfung.

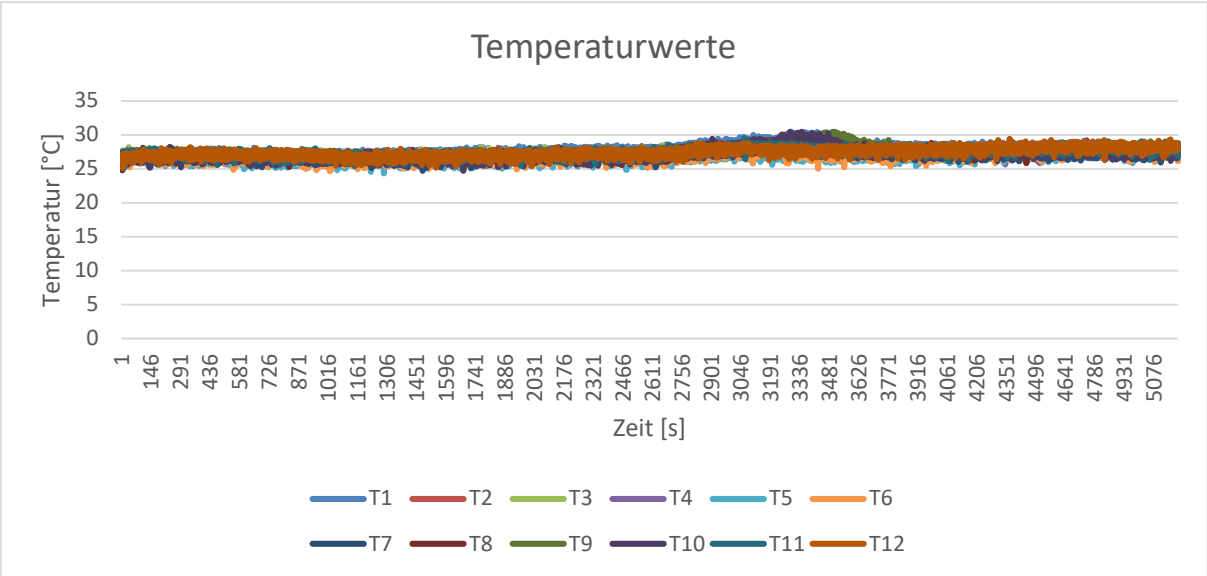


Abbildung 7-16: Temperaturen Ladeschlussspannung 4 V

Die Temperaturwerte sind über den gesamten Zeitraum durchwegs im Normalbereich.

Sofern eine Zellentemperatur den eingestellten Temperaturmaximalwert überschreitet, schaltet der Sicherheits Arduino alle stromführenden Verbindungen ab und verbleibt in einer Dauerschleife. Ein neuer Balancingvorgang ist nur bei Restart des Programmes möglich.

7.4.2 Haupttest

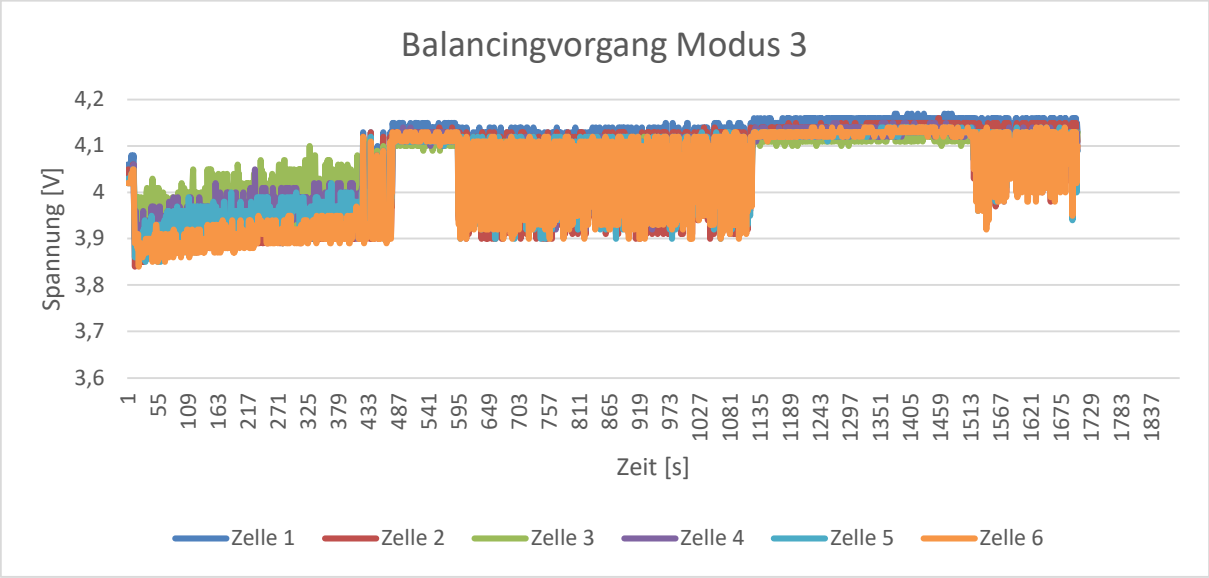


Abbildung 7-17: Zellspannungen 90 Ampere Ladestrom

In Abbildung 7-17 ist der Balancing Modus 3 dargestellt. Es ist ersichtlich, dass gleichzeitig geladen und gebalanced wird. Sofern das erste Mal die Ladeschlussspannung erreicht wird, stoppt der Ladevorgang und es wird gebalanced bis nur mehr eine geringe Abweichung zwischen den einzelnen Zellen auftritt. Da dadurch die Spannung wieder abnimmt, muss wieder geladen werden. Der Vorgang ist beendet, wenn die Ladeschlussspannung erreicht wird und alle Zellen nur mehr eine geringe Abweichung voneinander aufweisen.

Das starke Schwanken ist damit zu erklären, dass die Spannungsmessung durch den Ladestrom von 90 Ampere stark beeinflusst wird. Die Abweichungen werden sichtlich geringer, sofern der Ladevorgang pausiert. Der Testversuch musste vorzeitig gestoppt werden, da die Schwankungen das System dermaßen stark beeinflussen, dass es mehrere Stunden dauern könnte, bis der Ladevorgang beendet ist.

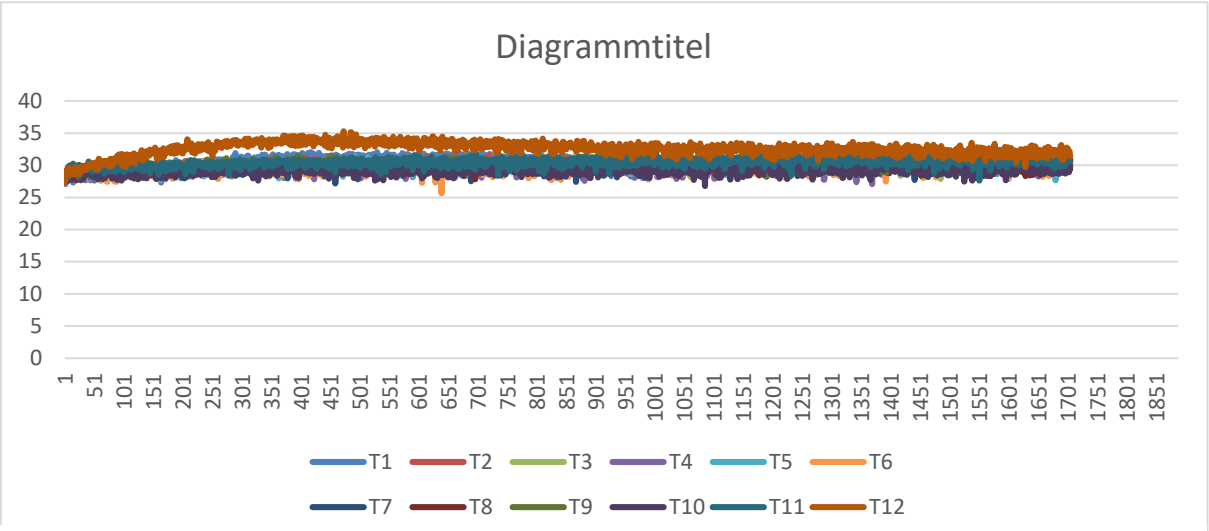


Abbildung 7-18: Temperaturverlauf 90 Ampere laden

7.4.3 Test Fehlerfall

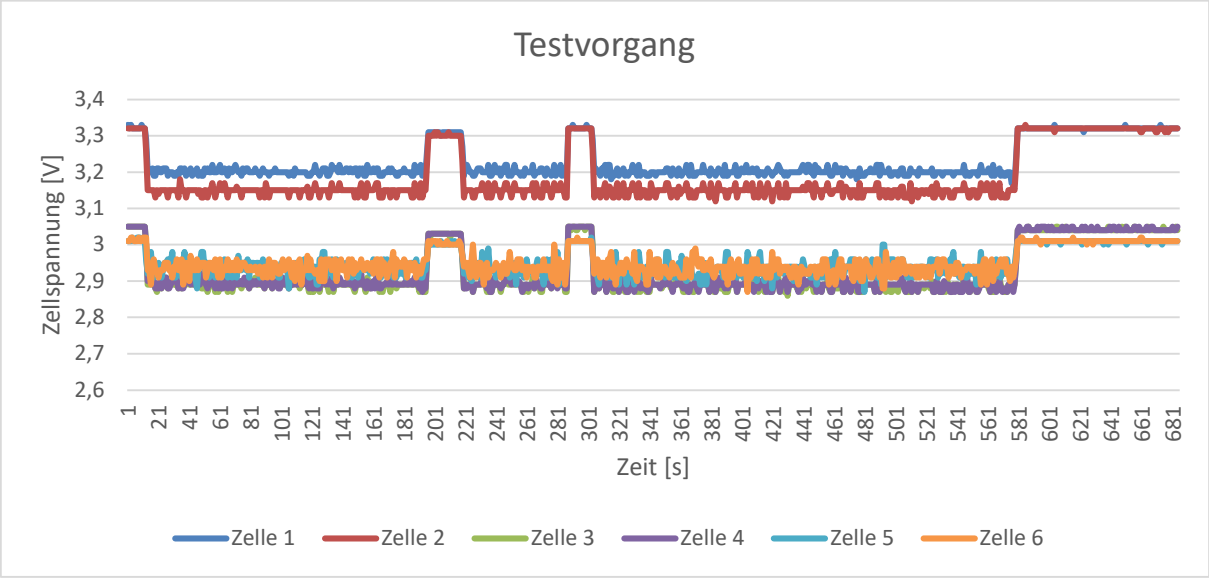


Abbildung 7-19: Balancing ohne Ladevorgang

Dieser Testversuch soll die Funktionsfähigkeit der GUI und die Arbeitsweise des Safety Arduinos überprüfen. Es wird nicht geladen sondern nur slow gebalanced. Die ersten zwei Unterbrechungen in Abbildung 7-19 zeigen ein Stoppen des Programmes mit anschließendem Start.

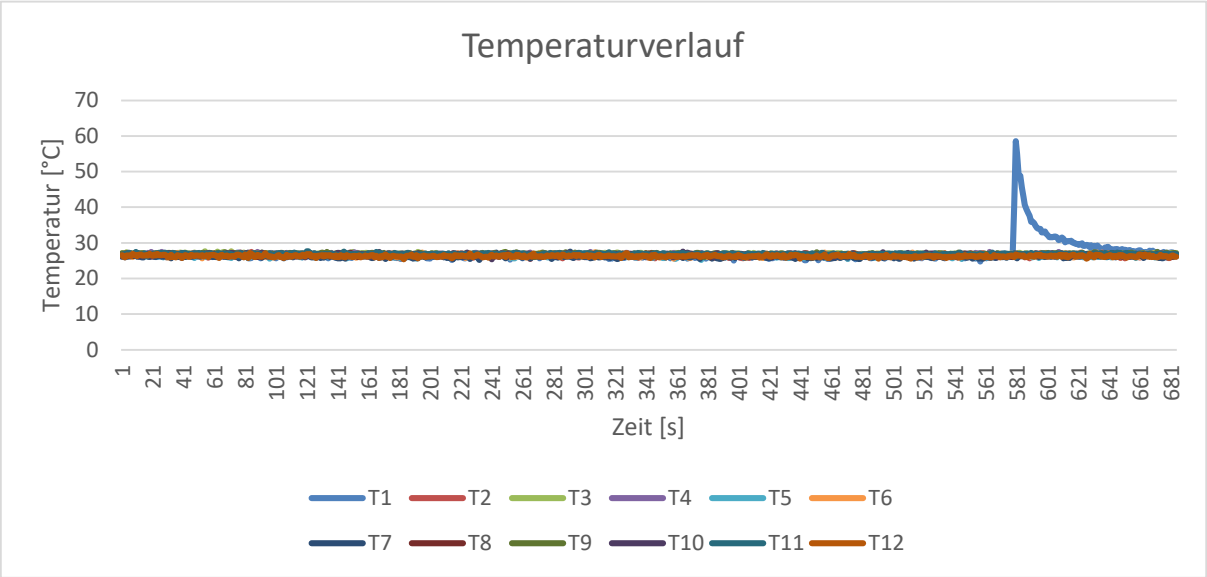


Abbildung 7-20: Temperaturverlauf

In Abbildung 7-20 wird bei einem Sensor durch Aufheizen dessen eine Temperaturüberschreitung simuliert. Wie ersichtlich ist, kühlt der Sensor ab, der Balancingvorgang wird aber nicht mehr fortgesetzt. Das Programm des Safety Arduinos schaltet keine Funktionen für den Balancing Arduino mehr frei. Die Daten werden bis zur Beendigung des Programmes mitgeloggt.

8 Ausblick

In der Zukunft wird intensiv an neuen Balancingsystemen geforscht werden, da man versucht Energie zu sparen. Somit wäre jeder Verlust von Abwärme Verschwendung. Abhängig von der Größe der Zelle können unterschiedliche Balancingarten eingesetzt werden.

Sofern die Entwicklung der Zellen voranschreitet und der Verwendungsgrad zunimmt, wird ein Balancer in den meisten Fällen erforderlich sein. Einer der wichtigsten Aspekte ist die Sicherheit. Der Umgang mit Lithium-Ionen Zellen ist gefährlich und darf nicht unterschätzt werden. Damit das Risiko minimiert wird, sollten gewisse Sicherheitsfeatures Standard werden. In den meisten Fällen ist ein Schutz gegen Verpolung, Tiefentladung, Überladung und Kurzschluss gegeben, doch welche Einheit kontrolliert die sicherheitsrelevanten Systeme?

Mikrochips sind Massenware und aus diesem Grund sinkt ihr Preis, abhängig von der Absatzzahl. Sofern ein neuer Industriestandard gesetzt wird, sinken auch die Preise der zurzeit teuren Sicherheitsfeatures.

Das Ziel soll sein, das Risiko einer Fehlfunktion soweit zu verringern, dass sie vernachlässigbar ist. Wie bekannt, ist Risiko das Produkt von Eintrittswahrscheinlichkeit und Schadensausmaß. Da das Schadensausmaß bei einer defekten Zelle sehr hoch ist, sollte man viel Innovation in den Bereich der Sicherheit investieren.

8.1 Mögliche Verbesserungen des Prototypen

8.1.1 Analog-/ Digitalconverter

Einer der wichtigsten Aspekte in der Verarbeitung von Messwerten ist es, hohe Genauigkeiten trotz schneller Abtastrate zu erzielen. Es gibt viele verschiedene Arten, analoge Signale in digitale umzuwandeln, doch jedes System hat Vor- und Nachteile. Genaue Analog-/ Digitalconverter sind teuer oder langsam. Ungenaue Systeme sind billig und schnell, aber die Messwerte können nicht für genaue Auswertungen verwendet werden.

Der nächste Schritt wäre ein Umstieg auf 16 Bit Analog/ Digitalconverter. Problematisch ist bei gewissen fertigen Modulen eine Limitierung der Adressierungsmöglichkeiten untereinander. Somit kann man nicht beliebig viele von diesen Baugruppen verbauen.

8.1.2 Platinenlayout

Kürzere Wege vom Anschluss zum Sensor sollten das Risiko von Einstreuungen minimieren. Eine Trennung von analoger und digitaler Masse wird bei hohen Abtastraten erforderlich sein. Die Verwendung von SMD Bauteilen ermöglicht eine noch kompaktere Platine. Das Einhalten von ESD Richtlinien soll gröbere Einstreuungen verhindern.

Das Verwenden einer großen Platine, auf der alle Baugruppen vorhanden sind, hilft gegen die Probleme, die Kabelverbindungen verursachen (Antenneneffekte, Masseschleife, etc.).

8.1.3 Gehäuse

Das Gehäuse des Prototyps soll gänzlich aus 3D gedruckten Teilen bestehen. Das hat den Vorteil der Reproduzierbarkeit. Sind einmal die Konstruktionsdaten vorhanden, kann man jedes Bauteil nachdrucken. Dies kann die Einzel- und Massenproduktion ermöglichen.

8.1.4 Lüfter

Der Prototyp beinhaltet 6 gleiche Lüfter, die bei einer Betriebsspannung von +12 V arbeiten. Die Lüfter sind klein und können keine hohe Drehzahl erreichen, was bedeutet, dass der Kühlprozess unterdimensioniert ist. Größere Lüfter mit eventueller Drehzahlregelung können effizienter sein und eine bessere Kühlleistung bieten. Eine Temperaturüberwachung des Innenraumes könnte den Lüfter erst nach einem gewissen Temperaturschwellwert einschalten, was Energiekosten spart.

8.1.5 Netzteil

In dieser Arbeit werden zwei unterschiedliche Netzteile verbaut. Das eine liefert Spannungen in der Größenordnung von ±12V und + 5V, das zweite ist für die Versorgung der Lüfter zuständig und gibt eine Spannung von +12 V aus. Eine Verbesserungsmöglichkeit wäre es, anstelle von zwei separaten Netzteilen ein Computernetzteil zu verwenden, welches ausreichend Leistung zur Verfügung stellen kann. Der Vorteil von modernen Computernetzteilen ist die Möglichkeit, Steckverbindungen zu lösen. Somit spart man sich eine unnötige Anzahl an Kabeln. Der zweite Vorteil bezieht sich auf die Leistungsabgabe und die Stabilität der einzelnen Spannungsniveaus. Computerinnereien (Motherboard, Grafikkarte, Ram, etc.) sind sensible Bauelemente und reagieren fehlerhaft bei schlechten Spannungsquellen. Deswegen sind Computernetzteile stabil aufgebaut und können auch viel Strom liefern.

8.1.6 Referenzspannung

Neben dem Analog-/ Digitalconverter ist die Qualität angelegte Referenzspannung wichtig. Die am Prototyp erstellte Referenzspannung hat einen langen Weg bis zum Verwendungsort. Dies bringt Einstreuungen mit sich, die in diesem Fall nicht akzeptabel sind. Eine Verbesserung wäre es, unterschiedliche Referenzspannungsmodule bauteilnahe zu implementieren, um kurze Verbindungswege zu erzielen.

8.1.7 Sicherungen

Im Testaufbau wurde bei der Überprüfung der Gatespannung des Leistungsfeldeffekttransistors ein Kurzschluss erzeugt, welcher eine Kabelverbindung in Rauch auflöste. Um gravierende Fehler auszuschließen, sollten auf jeder Platine eigene Sicherungen verbaut werden, welche knapp über dem Optimum des normalen Betriebes ausgelegt sind.

9 Literaturverzeichnis

[1] Atkins, P.W. and Paula, J. de (2013) *Physikalische Chemie*. 5th edn. Weinheim: Wiley-VCH Verlag GmbH & Co. KGaA.

[2] Brown, T.L. et al. (2018) *Chemie: Studieren kompakt*. 14th edn. (Pearson Studium - Chemie). Hallbergmoos: Pearson.

[3] Mortimer, C.E., Müller, U. and Beck, J. (2015) Chemie: *Das Basiswissen der Chemie*. 12th edn. Stuttgart: Thieme.

[4] Brodd, R.J. (2009) 'APPLICATIONS – PORTABLE | Notebooks: Batteries', in *Encyclopedia of Electrochemical Power Sources*: Elsevier, pp.22–28..

[5] Birke, P. and Schiemann, M. (2013) Akkumulatoren: Vergangenheit, Gegenwart und Zukunft elektrochemischer Energiespeicher. München: Herbert Utz Verlag

[6] Korthauer, R. (2013) *Handbuch Lithium-Ionen-Batterien*. Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] Kickelbick, G. (2008) *Chemie für Ingenieure*. (ing-maschinenbau). München: Pearson Studium.

[8] Jossen, A. and Weydanz, W. (2006) *Moderne Akkumulatoren richtig einsetzen*. Neusäß: Ubooks-Verlag

[9] Schulz, D. (2014) *Akkus und Ladetechniken: Das Praxisbuch für alle Akku-Typen, Ladegeräte und Ladeverfahren*. (Elektronik). Haar bei München: Franzis.

[10] Halaczek, T.L. (1996) *Batterien und Ladekonzepte*. s.l.: Franzis Verlag.

[11] Pistoia, G. (2014) *Lithium-ion batteries: Advances and applications*. Amsterdam: Elsevier.

[12] Schnabel, P. (2014) *Elektronik-Fibel: Elektronik Grundlagen, Bauelemente, Schaltungstechnik, Digitaltechnik*. 6th edn. Ludwigsburg: Patrick Schnabel.

[13] Vahldiek, H. (1970) *Operationsverstärker: Eigenschaften und Anwendungen in linearen und nichtlinearen Schaltungen*. (Telekosmos-Monographien zur Automation). Stuttgart: Telekosmos-Verl. Franckh.

[14] Zander, H. (1990) *Datenwandler: A/D- und D/A-Wandler - Schnittstellen der digitalen Signalverarbeitung*. 2nd edn. (Vogel-Fachbuch Nachrichtentechnik). Würzburg: Vogel.

[15] Microchip Technology Inc. (2008), *Datasheet – MCP3004/3008 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface.* Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21295d.pdf (2020, September,2).

[16] Nexperia GmbH (2020), *Datasheet – 74HC4067;74HCT4067 16-channel analog multiplexer/demultiplexer.* Available: http://assets.nexperia.com/documents/data-sheet/74HC_HCT4067.pdf (2020,September,2).

[17] Raspberry Pi Foundation (2019), *Datasheet – Raspberry Pi 4 Model B.* Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf (2020, September,2).

[18] Arduino LLC (2008), *Datasheet – Arduino Nano.* Available*:* https://arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf (2020,September,2).

[19] Fredershausen, M. (1995) *Mikrocontroller-Technik in der Praxis: 8051-Mikrocontroller-Schaltungen erfolgreich entwerfen und programmieren*. Poing: Franzis.

[20] Müller, H. and Walz, L. (1999) *Mikroprozessortechnik*. 5th edn. (Vogel-Fachbuch, 5). Würzburg: Vogel.

[21] Tietze, U., Schenk, C. and Gamm, E. (2002) *Halbleiter-Schaltungstechnik: Neuer Teil: nachrichtentechnische Schaltungen*. 12th edn. Berlin: Springer.

[22] Zach, F. (1979) *Leistungselektronik: Bauelemente, Leistungskreise, Steuerungskreise, Beeinflussungen*. Wien: Springer.

[23] Steidle, H.-G. (1995) *Transistoren-Kurz-Tabelle: Rund 9000 Transistoren mit ihren kennzeichnenden Daten und 97 Abbildungen von Gehäuseformen mit Anschlußbelegung*. 5th edn. Poing: Franzis.

[24] Linear Technology *Datasheet – LT1615/LT1615-1: Micropower Step-Up DC/DC Converters in ThinSOT.* Available: https://www.analog.com/media/en/technical-documentation/data-sheets/16151fas.pdf (2020, September,2).

[25] Texas Instruments Incorporated (2018), *Datasheet – TL431/TL432 Precision Programmable Reference.* Available: https://www.ti.com/lit/ds/symlink/tl431.pdf (2020, September,2).

[26] Infineon Technologies AG (2014), *Datasheet – IR MOSFET STRONG/RFET™IRFP7718PbF.* Available:https://www.infineon.com/cms/de/product/power/mosfet/12v-300v-n-channel-power-mosfet/irfp7718 (2020,September,2).

[27] Wupper, H. and Niemeyer, U. (1996) *Elektronische Schaltungen*. (Springer-Lehrbuch). Berlin: Springer.

[28] Analog Devices Incorporated (2015), *Datasheet – Low Voltage Temperature Sensors: TMP35/TMP36/TMP37.* Available: https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf (2020, September, 2).

[29] Heinhold, L. (1965) *Kabel und Leitungen für Starkstrom*. 2nd edn. Berlin: Siemens-Schuckertwerke.

[30] Thiel, U.L. (1998) *Schaltnetzteile erfolgreich planen und dimensionieren: Grundschaltungen, induktive Komponenten, Prüf- und Fertigungstechnik, EMV-Maßnahmen, Schaltungsbeispiele*. 2nd edn. Poing: Franzis.

10 Abbildungsverzeichnis

11 Tabellenverzeichnis

Seite

12 Abbildungsverzeichnis Anhang

```
from tkinter import *
import threading
import sys
import termios
import tty
import time
import random
import RPi.GPIO as GPIO
from TkinterBeta import tkinterClass as GUI
from TkinterBeta import Zeit as Zeiten
from TkinterBeta import WerteSchreiben as monitor
from TkinterBeta import MCP3008 as mcp3008
from DiplomarbeitAbgabe import Uart as uart

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(12, GPIO.OUT) #SafetyStartStop
GPIO.setup(16, GPIO.OUT) #SafetyReset
GPIO.setup(3, GPIO.OUT)  #BalStartStop
GPIO.setup(5, GPIO.OUT)  #BalSoftHard
GPIO.setup(7, GPIO.OUT)  #BalReset
GPIO.output(12, GPIO.LOW)
GPIO.output(3, GPIO.LOW)
GPIO.output(5, GPIO.LOW)
GPIO.output(16, GPIO.LOW)
GPIO.output(7, GPIO.LOW)

liste = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
anzahlBatt = 0
MaxTemp = 0
Ladeschlusspannung = 0
Mode = False
counter = 0
running = True
Start = False

def WerteAufnahme():
    global anzahlBatt
    global MaxTemp
    global Ladeschlusspannung
    zwischenspeicher = []
    Aufnahme = GUI.Nebenfenster()
    while True:
        if Aufnahme.Status() == False:
            Aufnahme.schleife()
        else:
            if Aufnahme.gibExit() == True:
                quit()
            zwischenspeicher = Aufnahme.GibWerte()
            Ladeschlusspannung = zwischenspeicher[0]
            MaxTemp = zwischenspeicher[1]
            anzahlBatt = zwischenspeicher[2]
            break

def HauptGui(Batt, Temp, Spannung):
    global liste
    global Mode
    global running
    global Start
    global anzahlBatt
    global MaxTemp
    global Ladeschlusspannung
    HauptFenster = GUI.Hauptfenster(Ladeschlusspannung, MaxTemp, anzahlBatt)
    while running:
        HauptFenster.getWerte(liste)
        HauptFenster.updateWerte()
        HauptFenster.aktualisiere()
        Mode = HauptFenster.gibMode()
        running = HauptFenster.gibRunning()
        Start = HauptFenster.gibstart()

def DatenAbgleich():
    global liste
    global anzahlBatt
    daten = mcp3008.mcp3008()
    while running:
        liste = daten.auslesenMittelwerteRichtig(anzahlBatt,10)

WerteAufnahme()
GPIO.output(16, GPIO.HIGH)
GPIO.output(7, GPIO.HIGH)
time.sleep(1)
GPIO.output(16, GPIO.LOW)
GPIO.output(7, GPIO.LOW)
time.sleep(2)

lo = uart.uart()
lo.uebertrageDaten(str(Ladeschlusspannung), str(anzahlBatt), str(MaxTemp), str(3))

t1 = threading.Thread(target=HauptGui, args=(Ladeschlusspannung,MaxTemp,anzahlBatt))
t1.start()
```

Abbildung i: Mainprogramm 1

```
92
93  n = Zeiten.zeit()
94  t2 = threading.Thread(target=n.dauerschleife)
95  t2.start()
96
97  t3 = threading.Thread(target=DatenAbgleich)
98  t3.start()
99
100 werteSpeicher = monitor.monitoring("testeMich.txt")
101 wechsel = False
102 ModeWechsel = False
103 StartWechsel = False
104 while running:
105     if(n.gibZustand() != wechsel):
106         wechsel = n.gibZustand()
107         werteSpeicher.schreibeDaten(liste)
108     if(ModeWechsel != Mode):
109         ModeWechsel = Mode
110         if(ModeWechsel == True):
111             GPIO.output(5, GPIO.HIGH)
112         else:
113             GPIO.output(5, GPIO.LOW)
114
115     if(Start == True and StartWechsel == False):
116         StartWechsel = True
117         GPIO.output(12, GPIO.HIGH)
118     if(Start == False and StartWechsel == True):
119         StartWechsel = False
120         GPIO.output(12, GPIO.LOW)
121
122 n.Exit()
```

Abbildung xxvii: Mainprogramm 2

```
from tkinter import *
import threading
import sys
import termios
import tty
import time
import random

class Hauptfenster():

    werte = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    MaxSpannung = 0
    MaxTemp = 0
    AnzahlBatt = 0
    fullScreenZustand = True
    Start = False
    BalancingMode = False
    running = True
    def __init__(self, Lade, Temp, AnzahlB):
        self.MaxSpannung = Lade
        self.MaxTemp = Temp
        self.AnzahlBatt = AnzahlB

        self.root = Tk()
        screenWidth = self.root.winfo_screenwidth()
        screenHeight = self.root.winfo_screenheight()
        self.root.attributes('-fullscreen', True)
        self.root.geometry("%dx%d+0+0" % (screenWidth, screenHeight))
        self.root.title("Balancer")

        self.e = Label(self.root, width=15, borderwidth=5, text="BalancingMode")  ######## ÄnderungEv
        self.e.grid(row=0, column=0, columnspan=3, padx=10, pady=10)

        self.Button1 = Button(self.root, text="Start", padx=77, pady=20, command=self.Start)
        self.Button1.grid(row=1, column=0, pady=20)
        self.Button2 = Button(self.root, text="Set Mode:Slow", padx=45, pady=20, command=self.SetMode)
        self.Button2.grid(row=3, column=0, pady=20)
        self.Button3 = Button(self.root, text="Stop", padx=80, pady=20, command=self.Stop)
        self.Button3.grid(row=1, column=8, pady=20)
        self.Button4 = Button(self.root, text="Fullscreen", padx=60, pady=20, command=self.fullScreen)
        self.Button4.grid(row=3, column=8, pady=20)
        self.Button5 = Button(self.root, text="Exit", padx=80, pady=20, command=self.Exit)
        self.Button5.grid(row=8, column=0, pady=20)

        self.ScaleB1 = Scale(self.root, from_=10000, to=0)
        self.ScaleB1.grid(row=1, column=2)
        self.ScaleB1.set(50)
        self.LabelB1 = Label(self.root, text="Battery: 1", padx=40, pady=20)
        self.LabelB1.grid(row=0, column=3)
        self.LabelB1Ausgabe = Label(self.root, text="0", padx=40, pady=20)
        self.LabelB1Ausgabe.grid(row=1, column=3)

        self.ScaleB2 = Scale(self.root, from_=10000, to=0)
        self.ScaleB2.grid(row=1, column=4)
        self.ScaleB2.set(50)
        self.LabelB2 = Label(self.root, text="Battery: 2", padx=40, pady=20)
        self.LabelB2.grid(row=0, column=5)
        self.LabelB2Ausgabe = Label(self.root, text="0", padx=40, pady=20)
        self.LabelB2Ausgabe.grid(row=1, column=5)

        self.ScaleB3 = Scale(self.root, from_=10000, to=0)
        self.ScaleB3.grid(row=1, column=6)
        self.ScaleB3.set(50)
        self.LabelB3 = Label(self.root, text="Battery: 3", padx=40, pady=20)
        self.LabelB3.grid(row=0, column=7)
        self.LabelB3Ausgabe = Label(self.root, text="0", padx=40, pady=20)
        self.LabelB3Ausgabe.grid(row=1, column=7)

        ######
        self.ScaleB4 = Scale(self.root, from_=10000, to=0)
        self.ScaleB4.grid(row=3, column=2)
        self.ScaleB4.set(50)
        self.LabelB4 = Label(self.root, text="Battery: 4", padx=40, pady=20)
        self.LabelB4.grid(row=2, column=3)
        self.LabelB4Ausgabe = Label(self.root, text="0", padx=40, pady=20)
        self.LabelB4Ausgabe.grid(row=3, column=3)

        self.ScaleB5 = Scale(self.root, from_=10000, to=0)
        self.ScaleB5.grid(row=3, column=4)
        self.ScaleB5.set(50)
        self.LabelB5 = Label(self.root, text="Battery: 5", padx=40, pady=20)
        self.LabelB5.grid(row=2, column=5)
        self.LabelB5Ausgabe = Label(self.root, text="0", padx=40, pady=20)
        self.LabelB5Ausgabe.grid(row=3, column=5)

        self.ScaleB6 = Scale(self.root, from_=10000, to=0)
        self.ScaleB6.grid(row=3, column=6)
        self.ScaleB6.set(50)
        self.LabelB6 = Label(self.root, text="Battery: 6", padx=40, pady=20)
        self.LabelB6.grid(row=2, column=7)
        self.LabelB6Ausgabe = Label(self.root, text="0", padx=40, pady=20)
```

Abbildung xlvii: Gui Klasse 1

```
92          self.LabelB6Ausgabe.grid(row=3, column=7)
93
94          self.LabelT1 = Label(self.root, text="T1", padx=20, pady=10)
95          self.LabelT1.grid(row=4, column=2)
96          self.LabelT1Ausgabe = Label(self.root, text="0", padx=20, pady=10)
97          self.LabelT1Ausgabe.grid(row=5, column=2)
98
99          self.LabelT2 = Label(self.root, text="T2", padx=20, pady=10)
100         self.LabelT2.grid(row=4, column=3)
101         self.LabelT2Ausgabe = Label(self.root, text="0", padx=20, pady=10)
102         self.LabelT2Ausgabe.grid(row=5, column=3)
103
104         self.LabelT3 = Label(self.root, text="T3", padx=20, pady=10)
105         self.LabelT3.grid(row=4, column=4)
106         self.LabelT3Ausgabe = Label(self.root, text="0", padx=20, pady=10)
107         self.LabelT3Ausgabe.grid(row=5, column=4)
108
109         self.LabelT4 = Label(self.root, text="T4", padx=20, pady=10)
110         self.LabelT4.grid(row=4, column=5)
111         self.LabelT4Ausgabe = Label(self.root, text="0", padx=20, pady=10)
112         self.LabelT4Ausgabe.grid(row=5, column=5)
113
114         self.LabelT5 = Label(self.root, text="T5", padx=20, pady=10)
115         self.LabelT5.grid(row=4, column=6)
116         self.LabelT5Ausgabe = Label(self.root, text="0", padx=20, pady=10)
117         self.LabelT5Ausgabe.grid(row=5, column=6)
118
119         self.LabelT6 = Label(self.root, text="T6", padx=20, pady=10)
120         self.LabelT6.grid(row=4, column=7)
121         self.LabelT6Ausgabe = Label(self.root, text="0", padx=20, pady=10)
122         self.LabelT6Ausgabe.grid(row=5, column=7)
123
124         self.LabelT7 = Label(self.root, text="T7", padx=20, pady=10)
125         self.LabelT7.grid(row=6, column=2)
126         self.LabelT7Ausgabe = Label(self.root, text="0", padx=20, pady=10)
127         self.LabelT7Ausgabe.grid(row=7, column=2)
128
129         self.LabelT8 = Label(self.root, text="T8", padx=20, pady=10)
130         self.LabelT8.grid(row=6, column=3)
131         self.LabelT8Ausgabe = Label(self.root, text="0", padx=20, pady=10)
132         self.LabelT8Ausgabe.grid(row=7, column=3)
133
134         self.LabelT9 = Label(self.root, text="T9", padx=20, pady=10)
135         self.LabelT9.grid(row=6, column=4)
136         self.LabelT9Ausgabe = Label(self.root, text="0", padx=20, pady=10)
137         self.LabelT9Ausgabe.grid(row=7, column=4)
138
139         self.LabelT10 = Label(self.root, text="T10", padx=20, pady=10)
140         self.LabelT10.grid(row=6, column=5)
141         self.LabelT10Ausgabe = Label(self.root, text="0", padx=20, pady=10)
142         self.LabelT10Ausgabe.grid(row=7, column=5)
143
144         self.LabelT11 = Label(self.root, text="T11", padx=20, pady=10)
145         self.LabelT11.grid(row=6, column=6)
146         self.LabelT11Ausgabe = Label(self.root, text="0", padx=20, pady=10)
147         self.LabelT11Ausgabe.grid(row=7, column=6)
148
149         self.LabelT12 = Label(self.root, text="T12", padx=20, pady=10)
150         self.LabelT12.grid(row=6, column=7)
151         self.LabelT12Ausgabe = Label(self.root, text="0", padx=20, pady=10)
152         self.LabelT12Ausgabe.grid(row=7, column=7)
153
154         self.LabelMaxT = Label(self.root, text="Maximale Temperatur", padx=20, pady=10)
155         self.LabelMaxT.grid(row=4, column=0, pady=20)
156         self.LabelMaxTAusgabe = Label(self.root, text="0", padx=20, pady=10)
157         self.LabelMaxTAusgabe.grid(row=5, column=0)
158
159         self.LabelMaxU = Label(self.root, text="Maximale Spannung", padx=20, pady=10)
160         self.LabelMaxU.grid(row=6, column=0)
161         self.LabelMaxUAusgabe = Label(self.root, text="0", padx=20, pady=10)
162         self.LabelMaxUAusgabe.grid(row=7, column=0)
163
164         self.LabelLadel = Label(self.root, text="Ladestrom", padx=20, pady=10)
165         self.LabelLadel.grid(row=4, column=8)
166         self.LabelLadelAusgabe = Label(self.root, text="0", padx=20, pady=10)
167         self.LabelLadelAusgabe.grid(row=5, column=8)
168
169         self.LabelGoal = Label(self.root, text="% Ziel", padx=20, pady=10)
170         self.LabelGoal.grid(row=6, column=8)
171         self.LabelGoalAusgabe = Label(self.root, text="0", padx=20, pady=10)
172         self.LabelGoalAusgabe.grid(row=7, column=8)
173
174         self.LabelZellen = Label(self.root, text="Anzahl Zellen:", padx=20, pady=10)
175         self.LabelZellen.grid(row=8, column=3)
176         self.LabelZellenAusgabe = Label(self.root, text="%d" % (self.AnzahlBatt), padx=20, pady=10)
177         self.LabelZellenAusgabe.grid(row=8, column=4)
178
179         self.LabelLadeschluss = Label(self.root, text="Ladeschluss U:", padx=20, pady=10)
180         self.LabelLadeschluss.grid(row=8, column=5)
181         self.LabelLadeschlussAusgabe = Label(self.root, text="%.1f" % (round(self.MaxSpannung,2)), padx=20, pady=10)
182         self.LabelLadeschlussAusgabe.grid(row=8, column=6)
```

Abbildung Iv: Gui Klasse 2

```
183
184         self.LabelStatus = Label(self.root, text="Status:", padx=20, pady=10)
185         self.LabelStatus.grid(row=8, column=7)
186         self.LabelStatusAusgabe = Label(self.root, text="Ok", padx=20, pady=10)
187         self.LabelStatusAusgabe.grid(row=8, column=8)
188
189     def getWerte(self, Daten):
190         for x in range(len(Daten)):
191             self.werte[x] = Daten[x]
192     def updateWerte(self):
193         zwischenspeicher = self.werte
194         self.LabelB1Ausgabe['text'] = zwischenspeicher[0]
195         self.LabelB2Ausgabe['text'] = zwischenspeicher[1]
196         self.LabelB3Ausgabe['text'] = zwischenspeicher[2]
197         self.LabelB4Ausgabe['text'] = zwischenspeicher[3]
198         self.LabelB5Ausgabe['text'] = zwischenspeicher[4]
199         self.LabelB6Ausgabe['text'] = zwischenspeicher[5]
200         self.LabelT1Ausgabe['text'] = zwischenspeicher[8]
201         self.LabelT2Ausgabe['text'] = zwischenspeicher[9]
202         self.LabelT3Ausgabe['text'] = zwischenspeicher[10]
203         self.LabelT4Ausgabe['text'] = zwischenspeicher[11]
204         self.LabelT5Ausgabe['text'] = zwischenspeicher[12]
205         self.LabelT6Ausgabe['text'] = zwischenspeicher[13]
206         self.LabelT7Ausgabe['text'] = zwischenspeicher[14]
207         self.LabelT8Ausgabe['text'] = zwischenspeicher[15]
208         self.LabelT9Ausgabe['text'] = zwischenspeicher[16]
209         self.LabelT10Ausgabe['text'] = zwischenspeicher[17]
210         self.LabelT11Ausgabe['text'] = zwischenspeicher[18]
211         self.LabelT12Ausgabe['text'] = zwischenspeicher[19]
212         self.LabelLadelAusgabe['text'] = zwischenspeicher[23]
213         self.ScaleB1.set(zwischenspeicher[0])
214         self.ScaleB2.set(zwischenspeicher[1])
215         self.ScaleB3.set(zwischenspeicher[2])
216         self.ScaleB4.set(zwischenspeicher[3])
217         self.ScaleB5.set(zwischenspeicher[4])
218         self.ScaleB6.set(zwischenspeicher[5])
219
220     def gibRunning(self):
221         if self.running == True:
222             return True
223         else:
224             return False
225
226     def Exit(self):
227         self.running = False
228         self.root.destroy()
229
230     def fullScreen(self):
231         if self.fullScreenZustand == True:
232             self.root.attributes('-fullscreen', False)
233             self.fullScreenZustand = False
234         else:
235             self.root.attributes('-fullscreen', True)
236             self.fullScreenZustand = True
237
238     def Start(self):
239         self.Start = True
240
241     def Stop(self):
242         self.Start = False
243
244     def SetMode(self):
245         if self.BalancingMode == False:
246             self.BalancingMode = True
247             self.Button2['text'] = "Set Mode:Fast"
248         else:
249             self.BalancingMode = False
250             self.Button2['text'] = "Set Mode:Slow"
251     def gibstart(self):
252         if self.Start == True:
253             return True
254         else:
255             return False
256     def gibMode(self):
257         if self.BalancingMode == True:
258             return True
259         else:
260             return False
261
262     def aktualisiere(self):
263         self.root.update_idletasks()
264         self.root.update()
265
266 class Nebenfenster():
267
268     AnzahlBatterys = 0
269     MaxTemperature = 0
270     LadeschlussSpannung = 0
271     ok = False
272     exit = False
273
```

Abbildung lvi: Gui Klasse 3

```
274     def __init__(self):
275         self.popup = Tk()
276 
277         self.popup.title("Battery+Connections")
278         self.popup.geometry("500x400+300+100")
279         self.w = Label(self.popup, text="Ladeschlussspannung", padx=60, pady=10)
280         self.w.grid(row=0, column=0)
281         self.w = Label(self.popup, text="Maximale Temperatur", padx=60, pady=10)
282         self.w.grid(row=2, column=0)
283         self.w = Label(self.popup, text="Anzahl Batterien", padx=60, pady=10)
284         self.w.grid(row=4, column=0)
285 
286         self.ScaleSpannung = Scale(self.popup, from_=0, to=5,resolution = 0.1, orient='horizontal', length="400")
287         self.ScaleSpannung.grid(row=1, column=0, padx=40)
288         self.ScaleSpannung.set(0.0)
289 
290         self.ScaleTemperature = Scale(self.popup, from_=0, to=100,resolution = 5, orient='horizontal', length="400")
291         self.ScaleTemperature.grid(row=3, column=0, padx=40)
292         self.ScaleTemperature.set(0)
293 
294         self.ScaleBattAnzahl = Scale(self.popup, from_=0, to=6,resolution = 1, orient='horizontal', length="400")
295         self.ScaleBattAnzahl.grid(row=5, column=0, padx=40)
296         self.ScaleBattAnzahl.set(0)
297 
298         self.ButtonOK = Button(self.popup, text="OK", padx=20, pady=20, command=self.SaveEinstellung)
299         self.ButtonOK.grid(row=6, column=0, pady=10)
300 
301         self.ButtonEXIT = Button(self.popup, text="EXIT", padx=20, pady=20, command=self.ExitPopup)
302         self.ButtonEXIT.grid(row=7, column=0, pady=10)
303 
304     def SaveEinstellung(self):
305         self.LadeschlussSpannung = self.ScaleSpannung.get()
306         self.MaxTemperature = self.ScaleTemperature.get()
307         self.AnzahlBatterys = self.ScaleBattAnzahl.get()
308         self.ok = True
309         print("Ladeschluss : %d , AnzahlBattery : %d, MaximaleTemperatur: %d" % (
310         self.LadeschlussSpannung, self.AnzahlBatterys, self.MaxTemperature))
311         self.popup.destroy()
312 
313     def gibExit(self):
314         if self.exit == True:
315             return True
316         else:
317             return False
318 
319     def ExitPopup(self):
320         self.ok = True
321         self.exit = True
322         self.popup.destroy()
323 
324     def Status(self):
325         return self.ok
326 
327     def GibWerte(self):
328         return [self.LadeschlussSpannung, self.MaxTemperature, self.AnzahlBatterys]
329 
330     def schleife(self):  # test
331         if self.ok == False:
332             self.popup.update()
333 
```

Abbildung lvii: Gui Klasse 4

```
1  import time
2
3  class zeit():
4      zustand = False
5      running = True
6
7      def Exit(self):
8          self.running = False
9      def warten(self):
10         time.sleep(1)
11     def gibZustand(self):
12         return self.zustand
13     def aendereZustand(self):
14         if self.zustand == False:
15             self.zustand = True
16         else:
17             self.zustand = False
18     def dauerschleife(self):
19         while self.running:
20             print(self.gibZustand())
21             self.warten()
22             self.aendereZustand()
```

Abbildung lviii: Zeit Klasse

```
from tkinter import *
import threading
import sys
import termios
import tty
import time
import os
#import os.path
import random
class monitoring():
    filename = ""
    def __init__(self, name):
        self.filename = name
        if os.path.exists(self.filename):
            os.remove(self.filename)
        f = open(self.filename, "w+")
        f.close()

    def schreibeDaten(self, data):
        f = open(self.filename, "a+")
        zwischenspeicher = ""
        for i in range(len(data)):
            zwischenspeicher += str(data[i]) + " "
        zwischenspeicher += "\n"
        f.write(zwischenspeicher)
        f.close()
```

Abbildung lix: Datenspeicher Klasse

```
import spidev
import time
import RPi.GPIO as GPIO

class mcp3008():

    def __init__(self):
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BOARD)

        GPIO.setup(40, GPIO.OUT)
        GPIO.setup(38, GPIO.OUT)
        GPIO.setup(36, GPIO.OUT)

        GPIO.output(40, GPIO.HIGH)
        GPIO.output(38, GPIO.HIGH)
        GPIO.output(36, GPIO.HIGH)


        self.spi = spidev.SpiDev()
        self.spi.open(0, 0)
        self.spi.max_speed_hz = 10000

    def auslesenPort(self, port):

        GPIO.output(40, GPIO.LOW)

        adc = self.spi.xfer2([0b00000001, ((8+port)<<4), 0])

        data = ((adc[1] & 3) << 8) + adc[2]
        GPIO.output(40, GPIO.HIGH)

        return data

    def auslesenKomplett(self):

        liste = []

        for i in range(3):
            for u in range(8):
                GPIO.output((40 - (i * 2)), GPIO.LOW)
                adc = self.spi.xfer2([0b00000001, ((8 + u) << 4), 0])
                data = ((adc[1] & 3) << 8) + adc[2]
                liste.append(data)
                GPIO.output((40-(i*2)), GPIO.HIGH)


        return liste
    def auslesenMittelwerteRichtig(self, anzahl, teiler):
        liste = []
        mittelwert = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
        for i in range(teiler):
            liste = self.auslesenRichtig(anzahl);
            for u in range(len(liste)):
                mittelwert[u] = mittelwert[u] + liste[u]
        for x in range(len(mittelwert)):
            mittelwert[x] = round((mittelwert[x] / teiler),2)
        return mittelwert


    def auslesenRichtig(self, anzahl):

        liste = []

        for i in range(3):
            for u in range(8):
                if i == 0:
                    if u < anzahl:
                        GPIO.output(40 , GPIO.LOW)
                        adc = self.spi.xfer2([0b00000001, ((8 + u) << 4), 0])
                        data = ((adc[1] & 3) << 8) + adc[2]
                        liste.append(round(data*5.056/(1024*0.75), 2))
                        GPIO.output(40 , GPIO.HIGH)
                    else:
                        liste.append(0)
                if i == 1:
                    if u < anzahl*2:
                        GPIO.output(36, GPIO.LOW) #38
                        adc = self.spi.xfer2([0b00000001, ((8 + u) << 4), 0])
                        data = ((adc[1] & 3) << 8) + adc[2]
                        liste.append(round(((data*5.056/(1024)-0.5)/0.01), 2))
                        GPIO.output(36, GPIO.HIGH)
                    else:
                        liste.append(0)
                if i == 2:
                    if anzahl > 4 and u < (anzahl-4)*2:
                        GPIO.output(38, GPIO.LOW) #36
                        adc = self.spi.xfer2([0b00000001, ((8 + u) << 4), 0])
                        data = ((adc[1] & 3) << 8) + adc[2]
                        #liste.append(data)
                        liste.append(round(((data*5.056/(1024)-0.5)/0.01), 2))
```

Abbildung lx: MCP3008 Klasse 1

```
92                  GPIO.output(38, GPIO.HIGH)
93              else:
94                  if len(liste) == 23:
95                      GPIO.output(38, GPIO.LOW)  # 36
96                      adc = self.spi.xfer2([0b00000001, ((8 + 7) << 4), 0])
97                      data = ((adc[1] & 3) << 8) + adc[2]
98                      liste.append(data)
99                      GPIO.output(38, GPIO.HIGH)
100                 else:
101                     liste.append(0)
102
103         return liste
104
105
106
```

Abbildung lxi: MCP3008 Klasse 2

```
import wiringpi
import time
import RPi.GPIO as GPIO

class uart():
    def __init__(self):
        GPIO.setmode(GPIO.BOARD)
        GPIO.setwarnings(False)
        self.ENL = 37
        self.ENR = 35
        self.ARDUINOL = 33
        self.ARDUINOR = 31
        self.S0L = 18
        self.S0R = 26
        self.S1R = 24

        GPIO.setup(self.ENL, GPIO.OUT)
        GPIO.setup(self.ENR, GPIO.OUT)
        GPIO.setup(self.ARDUINOL, GPIO.OUT)
        GPIO.setup(self.ARDUINOR, GPIO.OUT)
        GPIO.setup(self.S0L, GPIO.OUT)
        GPIO.setup(self.S0R, GPIO.OUT)
        GPIO.setup(self.S1R, GPIO.OUT)

        GPIO.output(self.ENL, GPIO.HIGH)
        GPIO.output(self.ENR, GPIO.HIGH)
        GPIO.output(self.ARDUINOL, GPIO.LOW)
        GPIO.output(self.ARDUINOR, GPIO.LOW)
        GPIO.output(self.S0L, GPIO.LOW)
        GPIO.output(self.S0R, GPIO.LOW)
        GPIO.output(self.S1R, GPIO.LOW)

        wiringpi.wiringPiSetup()
        self.serial = wiringpi.serialOpen('/dev/ttyS0', 9600)

    def abfrageL(self, message):
        GPIO.output(self.S0L, GPIO.LOW)
        GPIO.output(self.S0R, GPIO.HIGH)
        GPIO.output(self.S1R, GPIO.LOW)
        GPIO.output(self.ENL, GPIO.LOW)
        GPIO.output(self.ENR, GPIO.LOW)
        time.sleep(0.00004)
        wiringpi.serialPutchar(self.serial, ord(message))
        time.sleep(0.001)
        wiringpi.serialFlush(self.serial)

        GPIO.output(self.ENL, GPIO.HIGH)
        GPIO.output(self.ENR, GPIO.HIGH)
        time.sleep(0.00004)
        GPIO.output(self.S0L, GPIO.HIGH)
        GPIO.output(self.S0R, GPIO.LOW)
        GPIO.output(self.S1R, GPIO.LOW)
        GPIO.output(self.ENL, GPIO.LOW)
        GPIO.output(self.ENR, GPIO.LOW)
        time.sleep(0.00004)

        GPIO.output(self.ARDUINOL, GPIO.HIGH)
        wiringpi.serialFlush(self.serial)
        sicherheit = wiringpi.serialGetchar(self.serial)
        if sicherheit == ord(message):
            print(sicherheit)
            GPIO.output(self.ARDUINOL, GPIO.LOW)
        time.sleep(0.001)

    def abfrageR(self, message):
        GPIO.output(self.S0L, GPIO.LOW)
        GPIO.output(self.S0R, GPIO.HIGH)
        GPIO.output(self.S1R, GPIO.HIGH)
        GPIO.output(self.ENL, GPIO.LOW)
        GPIO.output(self.ENR, GPIO.LOW)
        time.sleep(0.00004)
        wiringpi.serialPutchar(self.serial, ord(message))
        time.sleep(0.001)
        wiringpi.serialFlush(self.serial)

        GPIO.output(self.ENL, GPIO.HIGH)
        GPIO.output(self.ENR, GPIO.HIGH)
        time.sleep(0.00004)
        GPIO.output(self.S0L, GPIO.HIGH)
        GPIO.output(self.S0R, GPIO.LOW)
        GPIO.output(self.S1R, GPIO.HIGH)
        GPIO.output(self.ENL, GPIO.LOW)
        GPIO.output(self.ENR, GPIO.LOW)
        time.sleep(0.00004)

        GPIO.output(self.ARDUINOR, GPIO.HIGH)
        wiringpi.serialFlush(self.serial)
        sicherheit = wiringpi.serialGetchar(self.serial)
        if sicherheit == ord(message):
            print(sicherheit)
            GPIO.output(self.ARDUINOR, GPIO.LOW)
```

Abbildung lxii: UART Klasse 1

```
92
93          time.sleep(0.001)
94
95      def durchgang(self, uebergabe):
96          for i in range(1):
97              wiringpi.serialFlush(self.serial)
98              self.abfrageL(uebergabe)
99              GPIO.output(self.ENL, GPIO.HIGH)
100             GPIO.output(self.ENR, GPIO.HIGH)
101             GPIO.output(self.ARDUINOL, GPIO.LOW)
102             GPIO.output(self.ARDUINOR, GPIO.LOW)
103             GPIO.output(self.S0L, GPIO.LOW)
104             GPIO.output(self.S0R, GPIO.LOW)
105             GPIO.output(self.S1R, GPIO.LOW)
106             wiringpi.serialFlush(self.serial)
107             self.abfrageR(uebergabe)
108             GPIO.output(self.ENL, GPIO.HIGH)
109             GPIO.output(self.ENR, GPIO.HIGH)
110             GPIO.output(self.ARDUINOL, GPIO.LOW)
111             GPIO.output(self.ARDUINOR, GPIO.LOW)
112             GPIO.output(self.S0L, GPIO.LOW)
113             GPIO.output(self.S0R, GPIO.LOW)
114             GPIO.output(self.S1R, GPIO.LOW)
115             #time.sleep(0.1)
116
117     def uebertrageDaten(self, Ladeschlussspannung, AnzahlBatt, Temperatur, BalModus):
118         for x in range(len(Ladeschlussspannung)):
119             self.durchgang(Ladeschlussspannung[x])
120         self.durchgang("X")
121         for x in range(len(AnzahlBatt)):
122             self.durchgang(AnzahlBatt[x])
123         self.durchgang("X")
124         for x in range(len(Temperatur)):
125             self.durchgang(Temperatur[x])
126         self.durchgang("X")
127         for x in range(len(BalModus)):
128             self.durchgang(BalModus[x])
129         self.durchgang("Z")
130
131
132
133
```

Abbildung lxiii: UART Klasse 2

```
#define B1 A7
#define B2 A6
#define B3 A5
#define B4 A4
#define B5 A3
#define B6 A2
#define Bal1 2
#define Bal2 3
#define Bal3 4
#define Bal4 5
#define Bal5 6
#define Bal6 7
#define SoftB 8
#define HardB 9
#define LadeOn 10
#define Reset A1
#define SoftHard 11
#define StartStop 12

#define umrechnungBatt 0.006572917
#define durchschnitt 50

double Ladeschlussspannung;
int AnzahlBatt, Temperatur, Modus;
double Batt[6];
boolean flag;
boolean botBalVar;
boolean topBalVar;
String Datenfluss;
boolean lade;
int stutter;

void(* resetFunc)(void) = 0;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10);
  pinMode(B1, INPUT);
  pinMode(B2, INPUT);
  pinMode(B3, INPUT);
  pinMode(B4, INPUT);
  pinMode(B5, INPUT);
  pinMode(B6, INPUT);
  pinMode(A0, INPUT);
  pinMode(Reset, INPUT);
  pinMode(SoftHard, INPUT);
  pinMode(StartStop, INPUT);
  pinMode(Bal1, OUTPUT);
  pinMode(Bal2, OUTPUT);
  pinMode(Bal3, OUTPUT);
  pinMode(Bal4, OUTPUT);
  pinMode(Bal5, OUTPUT);
  pinMode(Bal6, OUTPUT);
  pinMode(SoftB, OUTPUT);
  pinMode(HardB, OUTPUT);
  pinMode(LadeOn, OUTPUT);

  digitalWrite(Bal1, LOW);
  digitalWrite(Bal2, LOW);
  digitalWrite(Bal3, LOW);
  digitalWrite(Bal4, LOW);
  digitalWrite(Bal5, LOW);
  digitalWrite(Bal6, LOW);
  digitalWrite(SoftB, LOW);
  digitalWrite(HardB, LOW);
  digitalWrite(LadeOn, LOW);
  setBattZero();
  SetSoftHard();
  Ladeschlussspannung = 0.0;
  AnzahlBatt = 2;
  Temperatur = 0;
  Modus = 0;
  flag = false;
  Datenfluss = "";
  botBalVar = false;
  topBalVar = false;
  lade = false;
  stutter = 0;
}
void loop(){
   if(flag == false)
   {
      erfasseDaten();
   }
   else
   {
      if(Modus == 1) //BotBal
      {
        ResetArd();
        SetSoftHard();
        if(botBalVar == false)
        {
          if(istLade() == true)
          {
            do
            {allOff();ResetArd();}while(1);
          }
          botBal();
        }
        else
        {
          Lade();
          setBalOff();
          String s = (String)Batt[0] + " " + (String)Batt[1] + " " +(String)Batt[2] + " " + (String)Batt[3] + " " +(String)Batt[4] + " " + (String)Bat
          Serial.println(s);
```

Abbildung lxiv: Balancing Arduino 1

```
          if(lade == true)
          {
              do
              {allOff();ResetArd();Serial.println("XXX");}while(1);
          }
         }

        }//
        if(Modus == 2) //TopBal
        {
           ResetArd();
           SetSoftHard();
           if(lade == false)
           {
             Lade();
           }
           if(lade == true && topBalVar == false)
           {
             topBal();
           }
           if(lade == true && topBalVar == true)
           {
             setBalOff();
             LadeAus();
             for(int i = 0; i < 20; i++)
             {
               if(zielErreicht() == true && spannungErreicht() == true)
               {
                 do
                 {allOff();Serial.println("XXXX");ResetArd();}while(1);
               }
             }
             if(istLade() == false)
               lade = false;
             else
               topBalVar = false;
           }


        }
        if(Modus == 3) // Middl Bal
        {
           ResetArd();
           SetSoftHard();
           if(lade == false)
             Lade();
           topBal();
           if(lade == true)
           {
             setBalOff();
             LadeAus();
             for(int i = 0; i < 20; i++)
             {
               if(zielErreicht() == true && spannungErreicht() == true)
               {
                 do
                 {allOff();Serial.println("XXXX");ResetArd();}while(1);
               }
             }
             if(istLade() == false)
               lade = false;
               topBalVar = false;
           }

        }

    }
}
void Lade()
{
  bool zustand = false;
  gibMittelwert();
  for(int i = 0; i < AnzahlBatt; i++)
    if(Batt[i] > Ladeschlussspannung)
      zustand = true;
  if(zustand == false)
  {
    LadeEin();
    Serial.println("EIN");
  }
  else
  {
    LadeAus();
    lade = true;
  }
}
bool istLade()
{
  for(int i = 0; i < AnzahlBatt; i++)
    if(Batt[i] > Ladeschlussspannung)
    return true;
  return false;
}
void LadeEin()
{
  digitalWrite(LadeOn, HIGH);
}
void LadeAus()
{
  digitalWrite(LadeOn, LOW);
}
void setBattZero()
{
  for(int i = 0; i < 6; i++)
```

Abbildung lxv: Balancing Ardino 2

```
    Batt[i] = 0;
}
void ResetArd()
{
     if(analogRead(A1) >= 400)
        resetFunc();
}
void SetSoftHard()
{
    if(digitalRead(SoftHard) == true)
    {
      digitalWrite(SoftB, LOW);
      digitalWrite(HardB, HIGH);
    }
    else
    {
      digitalWrite(HardB, LOW);
      digitalWrite(SoftB, HIGH);
    }
}
boolean SStop()
{
  if(digitalRead(StartStop) == true)
    return true;
  else
    return false;
}
void gibAlleWerte()
{
  Batt[0] = analogRead(A7)*umrechnungBatt;
  Batt[1] = analogRead(A6)*umrechnungBatt;
  Batt[2] = analogRead(A5)*umrechnungBatt;
  Batt[3] = analogRead(A4)*umrechnungBatt;
  Batt[4] = analogRead(A3)*umrechnungBatt;
  Batt[5] = analogRead(A2)*umrechnungBatt;
}
void gibMittelwert()
{
   int anzahl = durchschnitt;
   double liste[] = {0,0,0,0,0,0};
   for(int u = 0; u < anzahl; u++)
   {
    gibAlleWerte();
    for(int i = 0; i < 6; i++)
      liste[i] += Batt[i];
   }
   for(int x = 0; x < 6; x++)
      Batt[x] = liste[x]/(anzahl*1.0);
}
void erfasseDaten()
{
  int i = 0;
  if(flag == false)
  {
  do
  {
    String daten = "";
    if(Serial.available())
    {
      do
      {
        daten += Serial.readString();
        Datenfluss += daten;
      }while(Serial.available());
      do
      {
        i = 0;
        if(daten[i] == 'Z')
          flag = true;
        i++;
      }while(!digitalRead(A0) && flag == false);
      for(int n = 0; n < daten.length(); n++)
        Serial.write(daten[n]);
      do
      {
        i = 0;
        if(daten[i] == 'Z')
          flag = true;
        i++;
      }while(!digitalRead(A0) && flag == false);
    }
  }while(flag == false);
  convertData(Datenfluss);
  }
}
void convertData(String data)
{
  String LSS = "", AnzahlB = "", Temp = "", Mod = "";
  int wechsel = 0;
  for(int i = 0; i < data.length(); i++)
  {
    if(data[i] == 'X')
      wechsel ++;
    else
    {
      switch(wechsel)
      {
        case 0:
          LSS += data[i];
          break;
        case 1:
          AnzahlB += data[i];
          break;
        case 2:
```

Abbildung lxvi: Balancing Arduino 3

```
          Temp += data[i];
          break;
        case 3:
          Mod += data[i];
          break;
      }
    }
  }
  Ladeschlussspannung = LSS.toDouble();
  AnzahlBatt = AnzahlB.toInt();
  Temperatur = Temp.toInt();
  Modus = Mod.toInt();
  Serial.print(Ladeschlussspannung);
  Serial.print("  ");
  Serial.print(AnzahlBatt);
  Serial.print("  ");
  Serial.print(Temperatur);
  Serial.print("  ");
  Serial.println(Modus);
}
void allOff()
{
  digitalWrite(Bal1, LOW);
  digitalWrite(Bal2, LOW);
  digitalWrite(Bal3, LOW);
  digitalWrite(Bal4, LOW);
  digitalWrite(Bal5, LOW);
  digitalWrite(Bal6, LOW);
  digitalWrite(SoftB, LOW);
  digitalWrite(HardB, LOW);
  digitalWrite(LadeOn, LOW);
}
void setBalOff()
{
  digitalWrite(Bal1, LOW);
  digitalWrite(Bal2, LOW);
  digitalWrite(Bal3, LOW);
  digitalWrite(Bal4, LOW);
  digitalWrite(Bal5, LOW);
  digitalWrite(Bal6, LOW);
}

void botBal()
{
   double minValuen = 100;
   gibMittelwert();
   for(int i = 0; i < AnzahlBatt; i++)
   {
      if(Batt[i] < minValuen)
      {
        minValuen = Batt[i];
      }
   }
   for(int u = 0; u < AnzahlBatt; u++)
   {
      if(Batt[u] > minValuen)
      {
        balOn(u);
        stutter ++;
      }
      else
      {
        balOff(u);
      }
   }
   if(stutter > 100)
   {
      setBalOff();
      stutter = 0;
      gibMittelwert();
   }
   if(Batt[0] > gibSchnitt()*0.99 && Batt[0] < gibSchnitt()*1.01)
   {
      botBalVar = true;
   }
   else
   {
      botBalVar = false;
   }
}
double gibSchnitt()
{
  double summe = 0;
  //gibAlleWerte();
  for(int i = 0; i < AnzahlBatt; i++)
    summe += Batt[i];
  return summe/(AnzahlBatt*1.0);
}
void balOn(int zahl)
{
    switch(zahl)
    {
      case 0:
        digitalWrite(Bal1, HIGH);
        Serial.println("TRUE");
        break;
      case 1:
        digitalWrite(Bal2, HIGH);
        break;
      case 2:
        digitalWrite(Bal3, HIGH);
        break;
      case 3:
        digitalWrite(Bal4, HIGH);
```

Abbildung lxvii: Balancing Arduino 3

```
        break;
      case 4:
        digitalWrite(Bal5, HIGH);
        break;
      case 5:
        digitalWrite(Bal6, HIGH);
        break;
    }
}
void balOff(int zahl)
{
    switch(zahl)
    {
      case 0:
        digitalWrite(Bal1, LOW);
        Serial.println("FALSE");
        break;
      case 1:
        digitalWrite(Bal2, LOW);
        break;
      case 2:
        digitalWrite(Bal3, LOW);
        break;
      case 3:
        digitalWrite(Bal4, LOW);
        break;
      case 4:
        digitalWrite(Bal5, LOW);
        break;
      case 5:
        digitalWrite(Bal6, LOW);
        break;
    }
}
void topBal()
{
   double minValuen = 100;
   gibMittelwert();
   for(int i = 0; i < AnzahlBatt; i++)
   {
      if(Batt[i] < minValuen)
      {
        minValuen = Batt[i];
      }
   }
   for(int u = 0; u < AnzahlBatt; u++)
   {
      if(Batt[u] > minValuen)
      {
        balOn(u);
        stutter ++;
      }
      else
      {
        balOff(u);
      }
   }
   if(stutter > 100)
   {
      setBalOff();
      stutter = 0;
      gibMittelwert();
   }
   if(Batt[0] > gibSchnitt()*0.99 && Batt[0] < gibSchnitt()*1.01)
   {
      topBalVar = true;
   }
   else
   {
      topBalVar = false;
   }
}
boolean zielErreicht()
{
  gibMittelwert();
  double schn = gibSchnitt();
  for(int i = 0; i < AnzahlBatt; i++)
    if(Batt[i] == schn)
      return false;
  return true;
}
bool spannungErreicht()
{
  gibMittelwert();
  for(int i = 0; i < AnzahlBatt; i++)
    if(Batt[i] < Ladeschlussspannung*0.99)
      return false;
  return true;
}
```

Abbildung lxviii: Balancing Arduino 4

```
#include <SPI.h>

#define CS_PIN1 3
#define CS_PIN2 4
#define CS_PIN3 5
#define CLOCK_PIN 13
#define MOSI_PIN 11
#define MISO_PIN 12
#define StartStop A1
#define Reset A2
#define LadeEin 10
#define BalEin 9

#define umrechnungBatt 0.0065716
#define umrechnungStrom 0.004882813
#define umrechnungTemp 0.01

double Ladeschlussspannung, cur, toleranzBat, toleranzTemp;
int AnzahlBatt, Temperatur, Modus;
String Datenfluss;
boolean flag;
boolean start;
boolean state;
double Batt[6];
double Temp[12];

void(* resetFunc)(void) = 0;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10);
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV32);//8
  pinMode(CS_PIN1, OUTPUT);
  pinMode(CS_PIN2, OUTPUT);
  pinMode(CS_PIN3, OUTPUT);
  pinMode(LadeEin, OUTPUT);
  pinMode(BalEin, OUTPUT);
  pinMode(A0, INPUT);
  pinMode(StartStop, INPUT);
  pinMode(Reset, INPUT);
  digitalWrite(CS_PIN1, HIGH);
  digitalWrite(CS_PIN2, HIGH);
  digitalWrite(CS_PIN3, HIGH);
  digitalWrite(LadeEin, LOW);
  digitalWrite(BalEin, LOW);
  Ladeschlussspannung = 0;
  AnzahlBatt = 0;
  Temperatur = 0;
  Modus = 0;
  flag = false;
  start = false;
  state = false;
  Datenfluss = "";
  cur = 0;
  initArray();
  toleranzBat = 1.05;
  toleranzTemp = 1.15;

}

void loop()
{
  if(flag == false)
  {
    erfasseDaten();
  }
  else
  {
    bildeMittelwertGewisseWerte(20, AnzahlBatt);
```

Abbildung lxix: Safety Arduino 1

```
    Serial.println(vergleiche());
    if(SStop() == true)
      start = true;
    else
      start = false;
    if(start == true && state == false)
    {
        schalteEin();
        state = true;
    }
    if(start ==false && state == true)
    {
        schalteAus();
        state = false;
    }
    if(vergleiche() == true)
    {
      schalteAus();
      do
      {ResetArd();}while(1);
    }

  }

}
void schalteEin()
{
  digitalWrite(LadeEin, HIGH);
  digitalWrite(BalEin, HIGH);
}
void schalteAus()
{
  digitalWrite(LadeEin, LOW);
  digitalWrite(BalEin, LOW);
}
boolean SStop()
{
    if(analogRead(StartStop) >= 400)
      return true;
    return false;
}
void ResetArd()
{
    if(analogRead(Reset) >= 400)
      resetFunc();
}
void initArray()
{
    for(int i = 0; i < 6; i++)
      Batt[i] = 0;
    for(int u = 0; u < 12; u++)
      Temp[u] = 0;
}
void erfasseDaten()
{
  int i = 0;
  if(flag == false)
  {
  do
  {
    String daten = "";
    if(Serial.available())
    {
      do
      {
        daten += Serial.readString();
        Datenfluss += daten;
      }while(Serial.available());
      do
```

Abbildung lxx: Safety Arduino 2

```
      {
        i = 0;
        if(daten[i] == 'Z')
          flag = true;
        i++;
      }while(!digitalRead(A0) && flag == false);
      for(int n = 0; n < daten.length(); n++)
        Serial.write(daten[n]);
      do
      {
        i = 0;
        if(daten[i] == 'Z')
          flag = true;
        i++;
      }while(!digitalRead(A0) && flag == false);
    }
  }while(flag == false);
  convertData(Datenfluss);
  }
}
void convertData(String data)
{
  String LSS = "", AnzahlB = "", Temp = "", Mod = "";
  int wechsel = 0;
  for(int i = 0; i < data.length(); i++)
  {
    if(data[i] == 'X')
      wechsel ++;
    else
    {
      switch(wechsel)
      {
        case 0:
          LSS += data[i];
          break;
        case 1:
          AnzahlB += data[i];
          break;
        case 2:
          Temp += data[i];
          break;
        case 3:
          Mod += data[i];
          break;
      }
    }
  }
  Ladeschlussspannung = LSS.toDouble();
  AnzahlBatt = AnzahlB.toInt();
  Temperatur = Temp.toInt();
  Modus = Mod.toInt();
  Serial.print(Ladeschlussspannung);
  Serial.print("  ");
  Serial.print(AnzahlBatt);
  Serial.print("  ");
  Serial.print(Temperatur);
  Serial.print("  ");
  Serial.println(Modus);
}
double ADCWert(int kanal, int csPin)
{
  uint8_t links = 0x00;
  uint8_t rechts = 0x00;
  switch (csPin)
  {
    case 0:
      if(kanal < 6)
      {
      digitalWrite(CS_PIN1, LOW);
```

Abbildung lxxi: Safety Arduino 3

```
      SPI.transfer(0b00000001);
      links = SPI.transfer(((0b10000000+((0b00000000 + kanal)<<4))));
      rechts = SPI.transfer(0b00000000);
      digitalWrite(CS_PIN1, HIGH);
      return ((((links & 0x03) << 8)+rechts)) * umrechnungBatt;
      }
      break;
    case 1:
      digitalWrite(CS_PIN2, LOW);
      SPI.transfer(0b00000001);
      links = SPI.transfer(((0b10000000+((0b00000000 + kanal)<<4))));
      rechts = SPI.transfer(0b00000000);
      digitalWrite(CS_PIN2, HIGH);
      return (((((((links & 0x03) << 8)+rechts)))*5.0/1024.0) - 0.5)/umrechnungTemp;
      break;
    case 2:
      if(kanal >=4 && kanal <7)
        return 0x00;
      if(kanal != 7)
      {
      digitalWrite(CS_PIN3, LOW);
      SPI.transfer(0b00000001);
      links = SPI.transfer(((0b10000000+((0b00000000 + kanal)<<4))));
      rechts = SPI.transfer(0b00000000);
      digitalWrite(CS_PIN3, HIGH);
      return (((((((links & 0x03) << 8)+rechts)))*5.0/1024.0) - 0.5)/umrechnungTemp;
      }
      else
      {
      digitalWrite(CS_PIN3, LOW);
      SPI.transfer(0b00000001);
      links = SPI.transfer(((0b10000000+((0b00000000 + kanal)<<4))));
      rechts = SPI.transfer(0b00000000);
      digitalWrite(CS_PIN3, HIGH);
      return ((((links & 0x03) << 8)+rechts)) * umrechnungStrom;
      }
      break;
    }
}
void aktualisiereAlleWerte()
{
  for(int i = 0; i < 3; i++)
  {
      if(i == 0)
      {
          for(int u = 0; u < 6; u++)
            Batt[u] = ADCWert(u,i);
      }
      if(i == 1)
      {
          for(int x = 0; x < 8; x++)
            Temp[x] = ADCWert(x,i);
      }
      if(i == 2)
      {
          for(int n = 0; n < 4; n++)
            Temp[n+8] = ADCWert(n,i);
          cur = ADCWert(7,i);
      }
  }
}
void aktualisiereGewisseWerte(int anzahl)
{
  for(int i = 0; i < 3; i++)
  {
      if(i == 0)
      {
          for(int u = 0; u < anzahl; u++)
            Batt[u] = ADCWert(u,i);
```

Abbildung lxxii: Safety Arduino 3

```
      }
      if(i == 1)
      {
         if(anzahl >= 4)
         {
          for(int x = 0; x < 8; x++)
            Temp[x] = ADCWert(x,i);
         }
         else
         {
          for(int x = 0; x < anzahl*2; x++)
            Temp[x] = ADCWert(x, i);
         }
      }
      if(i == 2)
      {
          for(int n = 0; n < (anzahl-4)*2; n++)
            Temp[n+8] = ADCWert(n,i);
          cur = ADCWert(7,i);
      }
  }
}
void bildeMittelwert(int cycle)
{
  double batSpeicher[6] = {0,0,0,0,0,0};
  double tempSpeicher[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
  double curSpeicher = 0;
  for(int i = 0; i < cycle; i++)
  {
      aktualisiereAlleWerte();
      for(int n = 0; n < 6; n++)
        batSpeicher[n] += Batt[n];
      for(int u = 0; u < 12; u++)
        tempSpeicher[u] += Temp[u];
      curSpeicher += cur;
  }
  for(int i = 0; i < 6; i++)
    Batt[i] = batSpeicher[i]/(double)cycle;
  for(int u = 0; u < 12; u++)
    Temp[u] = tempSpeicher[u]/(double)cycle;
  cur = curSpeicher/(double)cycle;
}
void bildeMittelwertGewisseWerte(int cycle, int anzahl)
{
  double batSpeicher[6] = {0,0,0,0,0,0};
  double tempSpeicher[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
  double curSpeicher = 0;
  for(int i = 0; i < cycle; i++)
  {
      aktualisiereGewisseWerte(anzahl);
      for(int n = 0; n < 6; n++)
        batSpeicher[n] += Batt[n];
      for(int u = 0; u < 12; u++)
        tempSpeicher[u] += Temp[u];
      curSpeicher += cur;
  }
  for(int i = 0; i < 6; i++)
    Batt[i] = batSpeicher[i]/(double)cycle;
  for(int u = 0; u < 12; u++)
    Temp[u] = tempSpeicher[u]/(double)cycle;
  cur = curSpeicher/(double)cycle;
}
boolean vergleiche()
{
    boolean ueberschreiten = false;
    for(int i = 0; i <AnzahlBatt; i++)
    {
        if(Batt[i] > Ladeschlussspannung*toleranzBat)
          ueberschreiten = true;
```

Abbildung lxxiii: Safety Arduino 4

```
    }
    for(int u = 0; u < AnzahlBatt*2; u++)
    {
        if(Temp[u] > Temperatur*toleranzTemp)
          ueberschreiten = true;
    }
    return ueberschreiten;

}
```

Abbildung lxxiv: Safety Arduino 5