**Danksagung**

**Inhaltsverzeichnis**

0 Allgemeine Beschreibung

1 Mechanik

2 Elektronik

2.1 Allgemeines

2.2 Funktionsprinzip

2.3 Platinen

2.3.1 Platinen allgemein

2.3.2 Hauptplatine

2.3.3 Unterplatine

2.3.4 Nebenplatine

2.4 Schaltung und Bauteile

2.4.1 Spannungsversorgung

2.4.2 LiPo-Akku und Ladeschaltung

2.4.3 Modi-Anzeige

2.4.4 Dehnmessstreifen

2.4.5 Differenzenverstärker

2.4.6 RC-Tiefpassfilter

2.4.7 Analoger Multiplexer

2.4.8 Analog Digital Wandler

2.4.9 Logik-Level Konverter

2.4.10 Testverfahren

2.5 Berechnungen

3 Programmierung

4 Graphical User Interface

4.1 GUI Allgemein

4.2 GUI Oberfläche

4.3 Struktur des Codes

4.3.1 Fenster & Allgemeines

4.3.2 Einstellgrad

4.3.3 Graph

5 Bluetooth

6 Infrarot Datenbank

6.1 Infrarot allgemein

6.2 Infrarot-Schaltung

6.3 Datenspeicher

7 Quellen & Verwendete Programme

**2 Elektronik**

**2.1 Allgemeines**

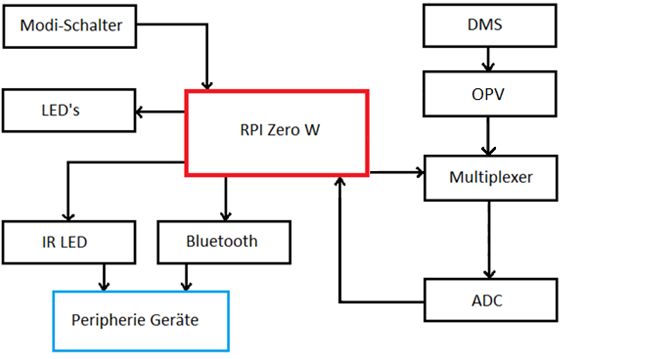
Um das Ziel einer funktionierenden Elektronik zu erreichen, musste zuerst eine Übersicht erstellt werden, um einen reibungslosen Ablauf des Geräts zu gewährleisten. Diese wurde von allen Teilnehmern der Diplomarbeit verfasst. In diesem Teil der Diplomarbeit geht es um die vertiefende Erklärung zu den einzelnen elektronischen Komponenten, sowie die Lösungsansätze, um eine fehlerfreie und sichere Funktionalität unseres Projektes zu gewährleisten.

Zuerst musste die Frage beantwortet werden, welche Funktionen die Elektronik überhaupt erfüllen muss. Bei unserem ersten Konzept wurden folgende Aufgaben der Elektronik definiert. Es müssen fünf Dehnmessstreifen (wird in diesem Text als DMS abgekürzt) möglichst gleichzeitig ausgewertet und die gemessenen Werte mit einem Operationsverstärker auf gut einlesbare Werte verstärkt werden. Anschließend werden sie mithilfe eines Raspberry Pi Zero W verarbeitet. Die Auswertung erfolgt mittels Microcontroller. Ein Schalter regelt zwischen den verschiedenen Peripherie-Steuerungen. Der verwendete Modus wird durch LED’s visualisiert. Das komplette Gerät soll mit 2 AAA Batterien, die je 1,5 Volt liefern, betrieben werden. Diese 3V werden mit einer DC-DC Step Up Schaltung auf 5V gewandelt. Die Anzeige der Aktivschaltung wird ebenfalls durch eine LED visualisiert. Die Infrarot-LED soll mit 3,3V versorgt werden und über einen GPIO Ausgang des Raspberry, der einen NPN „Bipolar-Junction-Transistor“ ansteuert und so ein Infrarot-Signal erzeugt.

Beim zweiten Konzeptionierungsansatz - nach einer Besprechung mit unserem Betreuer - wurde der Microcontroller durch einen Analog-Digital-Wandler (wird in diesem Text als ADC abgekürzt) und einen analogen Multiplexer (wird in diesem Text als MUX abgekürzt) ersetzt. Es wurde ein 12-Bit ADC gewählt, der eine SPI Schnittstelle und einen Shutdown Pin besitzt und damit eine simple Steuerung durch den Raspberry ermöglicht. Außerdem wurde ein Multiplexer mit einer Übersetzung von 8:1 gewählt, der durch eine 3-Bit Logik-Schnittstelle mit dem Raspberry verbunden ist. Es wurde unter anderem auch die Messmethode für die DMS, auf eine Wheatstone Messbrücke festgelegt. Im Detail

wurde eine Viertelbrücke gewählt, da es bei der mechanischen Konstruktion einfacher ist, nur einen DMS pro Finger zu montieren. Dazu wurde ein Tiefpassfilter geplant, der hohe Frequenzen und damit ein Messrauschen filtert. Auch wurden die Batterien durch Lithium-Polymer Akkus ersetzt, da sie bei diesem Projekt mehr Sinn ergeben. Durch ihre platzsparende Bauweise und die Möglichkeit, sie durch eine Ladeschaltung wieder aufzuladen, haben sie bei einem mobilen Gerät klare Vorteile gegenüber Batterien. Tests ergaben, dass die Operationsverstärker für klare Messergebnisse eine negative Versorgungsspannung benötigen. Es wurde ein zweites DC-DC Step Up Modul in die Schaltung integriert und mit einem Spannungswandler IC auf negativ 5V gewandelt.

**2.2 Funktionsprinzip**

****

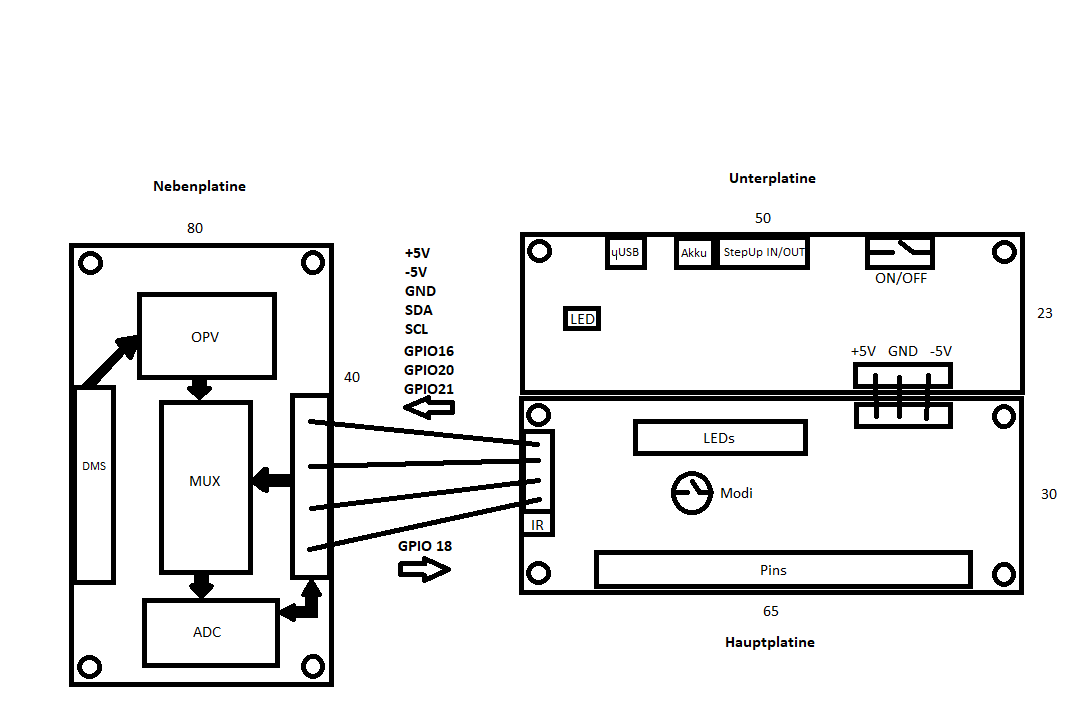
Es werden durch den Grad der Abknickung der DMS auf jedem Finger Signale erzeugt, die je durch eine Differenzenverstärkende OPV-Schaltung auf gut messbare Werte verstärkt werden. Anschließend wird durch einen RC-Tiefpass das Rauschen hoher Frequenzen gefiltert und die Signale in kurzen Abständen, nacheinander durch einen Multiplexer geschickt, dessen Schaltlogik durch den Raspberry gesteuert wird. Der ADC wandelt dieses analoge Signal in ein digitales Signal um, welches anschließend in den Raspberry eingelesen wird.   
  
Durch den Modi-Schalter können fünf verschiedene Module im Programm des Raspberry aufgerufen werden. Durch die Betätigung des Knopfes wird auf das nächste Steuermodul umgeschaltet. Mit diesem können dann z.B. über die Infrarot LED, periphere Geräte mit einer IR-Schnittstelle bedient werden.

**2.3 Platinen**

**2.3.1 Platinen Allgemein**

Unser erstes Konzept sah vor, dass es nur eine Platine gibt, die über dem Raspberry in einem Gehäuse fixiert ist. Dabei gab es jedoch Probleme mit der Mechanik und den Dehnmessstreifen, da die Platine sehr weit von den zu messenden Stellen entfernt war und wir sehr lange Kabel brauchten.

Es wurden also ein neues Prinzip entwickelt, welches aus 3 getrennten Platinen besteht, da so die Auswertung der Dehnmessstreifen auf geringstem Abstand stattfinden kann und es für den Bediener des Handschuhs angenehmer zu benutzen ist.



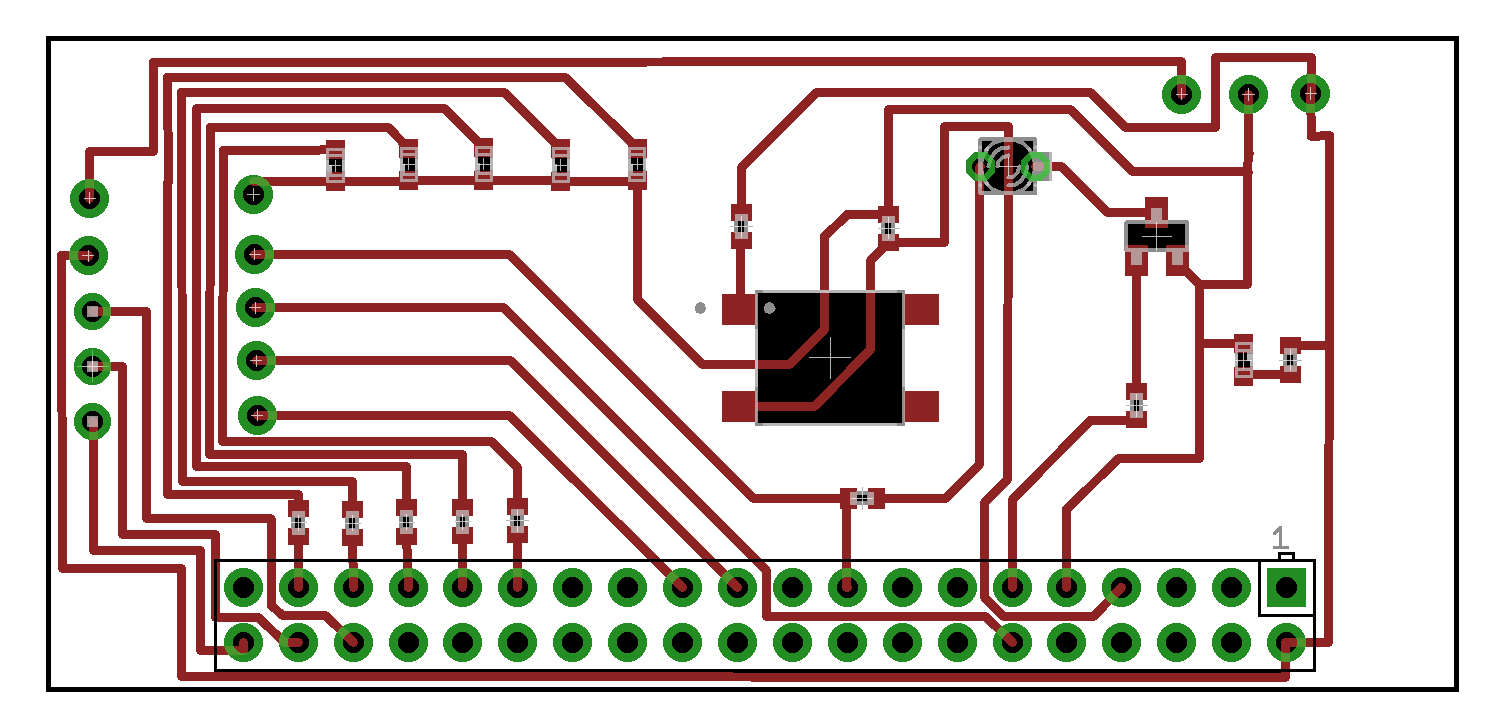
Die Schaltung wurde nach ihren funktionellen Baugruppen getrennt, da es so einen besseren Überblick und eine leichtere Fehleranalyse ermöglicht. Die Hauptplatine liegt dabei direkt über dem Raspberry, dessen Pins direkt durch Löcher in der Platine gesteckt und angelötet werden. Die Platinen werden über angelötete Jumperkabelgruppen miteinander verbunden, da so eine flexible Länge wählbar ist und sie bei Problemen leicht auszutauschen sind.

Die einzelnen IC’s auf jeder Platine wurden im SMD-Format gewählt, da es sonst platztechnisch zu Problemen gekommen wäre. Bis auf ein paar besondere Bauteile, wie die Infrarot LED, wurden auch alle anderen elektrischen Bauteile in SMD-Baugrößen gewählt.

Alle Widerstände wurden standardisiert auf SMD-Baugröße 0402 festgelegt, da es das kleinste SMD-Format ist, welches man noch ohne Probleme mit der Hand bzw. einer Pinzette löten kann. Die einzige Ausnahme machen dabei die 348Ω Widerstände der Messbrücken, da sie in der Baugröße 0402 bei unseren gewählten Händlern nicht in dieser Größe zu kaufen waren. Stattdessen wurden sie im 0603 Format bestellt, da es die nächstgelegene Größe war, die noch mit der Hand zu löten war.

Die drei Platinen wurden in der Elektrotechnik Werkstätte der HTL Mödling gefertigt, da wir dort die Platinen kostengünstiger ätzen konnten, als wenn wir sie bei einem externen Unternehmen bestellt hätten. Außerdem besitzt die Werkstätte einen Reflow-Oven, der das Löten von SMD-Bauteilen deutlich einfacher macht, als die konventionelle Art erlaubt.

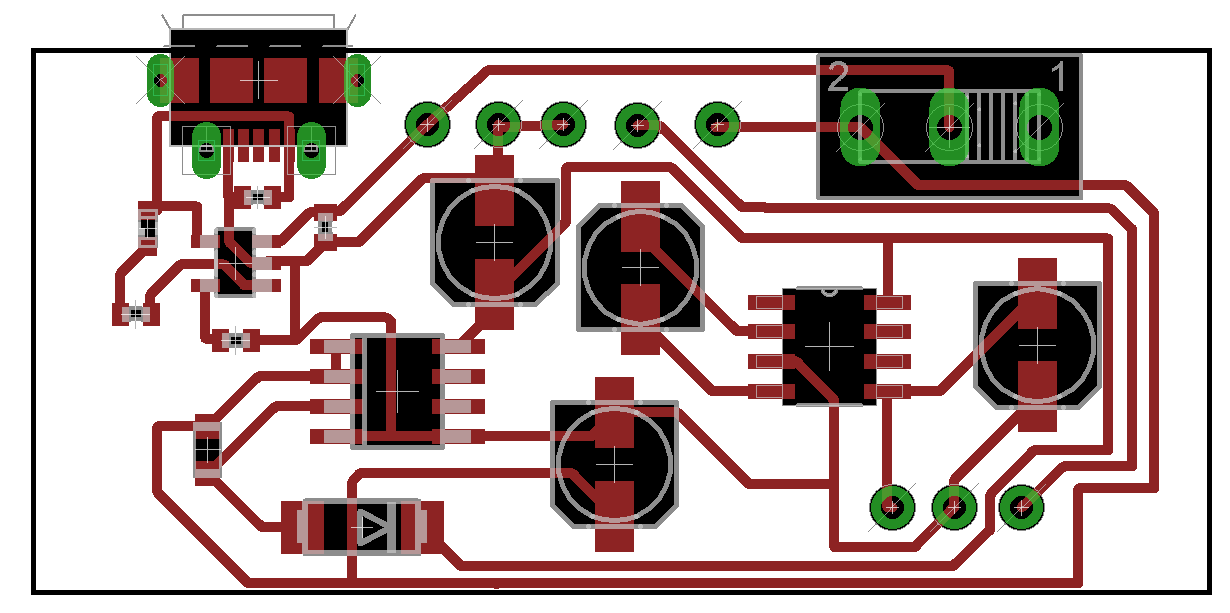
**2.3.2 Hauptplatine**



Die Hauptplatine wurde mit den Maßen 65x30 gefertigt, da diese der Baugröße eines Raspberry Pi Zero W entspricht. Die Hauptplatine liegt über dem Raspberry Pi und dessen GPIO-Pinleiste wird direkt durch Bohrungen auf der Platine gesteckt und angelötet. Die vier Ecken der Platine wurden nicht mit Kupferbahnen belebt, da dort Löcher gebohrt wurden, um die Platine im Gehäuse zu befestigen. Die Platine dient außerdem als Verbindung zwischen Unter- und Nebenplatine, da sie die Spannungsversorgung von der Unterplatine, also -5V, +5V und GND, für sich selbst und den Raspberry nützt, aber sie auch durch Leiterbahnen an Pins weiterleitet, die mit der Nebenplatine verbunden sind.

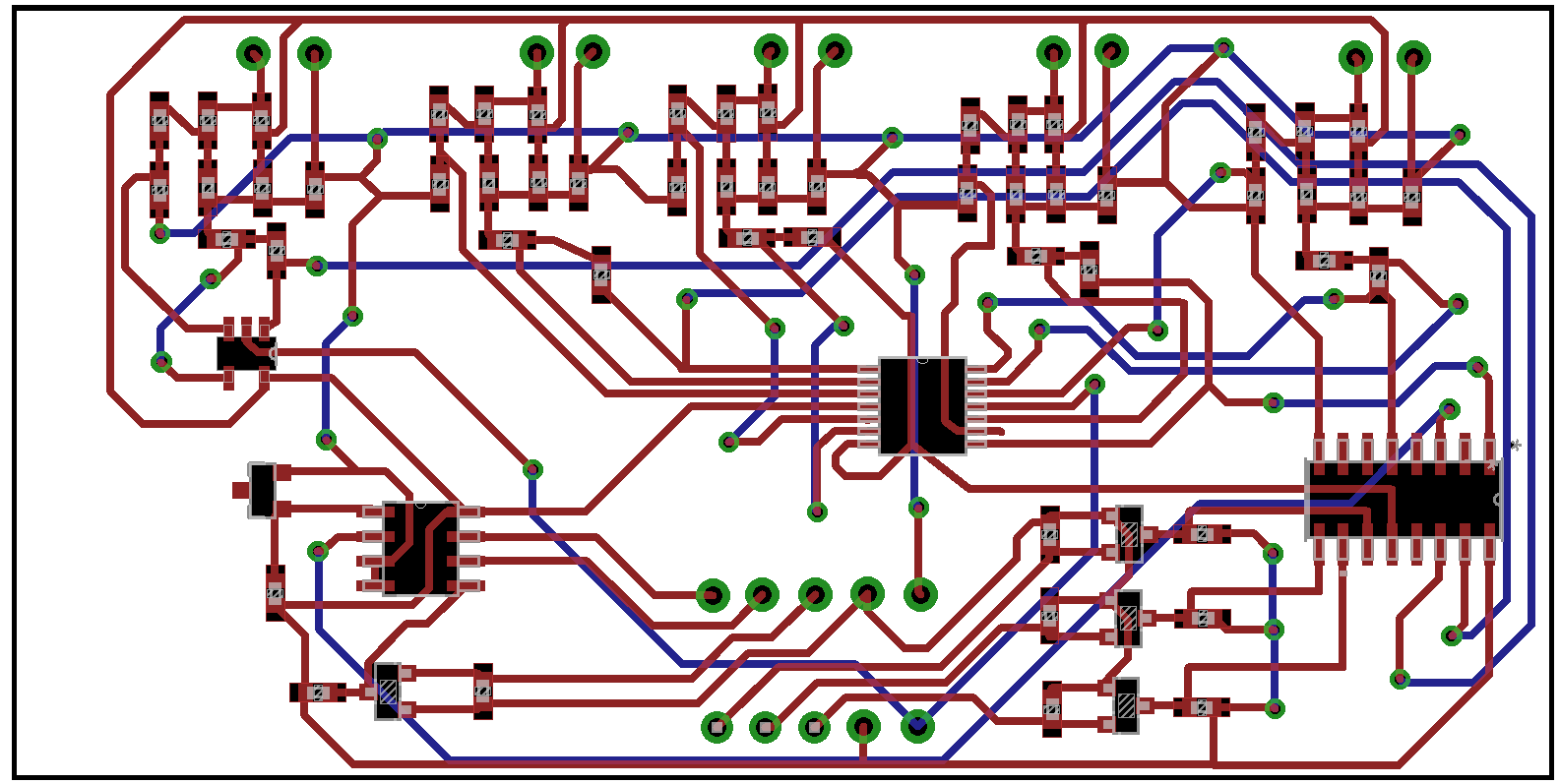
Die Platine verbindet auch die GPIO-, sowie die SPI-Ausgänge des Raspberry mit der Nebenplatine, da sich ein Großteil der zu steuernden IC’s auf jener Platine befinden. Die Hauptplatine beinhaltet einen SMD Druckknopf für die Modi-Auswahl, sowie die LED’s für die Visualisierung der einzelnen Module. Außerdem besitzt sie eine LED, die zeigt, ob das Gerät ein- oder ausgeschaltet ist. Die letzte verbaute Baugruppe ist die Infrarot-LED, sowie ein NPN-Transistor und deren Vorwiderstände, die als einziger Bauteil, neben den verlöteten Pins, in der THT-Baugröße ausgeführt ist, da der eigentliche LED-Kopf im Außengehäuse befestigt ist und nicht auf platinennahem Niveau wie die anderen Bauteile. Diese Platine wurde in der HTL Mödling gefertigt und auch dort bestückt.

**2.3.3 Unterplatine**

****

Die Platine wurde mit den Maßen 50x23 gefertigt, da das Seitengehäuse so genügend Platz für die externe DC-DC Step Up Platine und den Lithium-Ion-Polymer-Akku hat. Die Unterplatine ist rechts neben der Hauptplatine am Unterarm befestigt. Es wurden wieder die Ecken der Platine freigelassen, um dort zu bohren und sie mit dem Gehäuse zu fixieren. Die Platine hat angelötete Pins, um eine Verbindung zu der externen Platine und dem LiPo-Akku zu schaffen. Die Platine hat drei Pins (+5V, -5V, GND), die mit der Hauptplatine verbunden sind und das gesamte System mit Spannung versorgen. Ob diese Spannung zu anderen Platinen gelangt, wird durch einen Schiebeschalter geregelt. Die Platine beinhaltet außerdem einen Micro-USB-Port, der mit einem sogenannten „Battery-Charge-Management-Chip“ den Akku laden kann. Zusätzlich befindet sich dort auch noch die Baugruppe für die negative Spannungsversorgung, bestehend aus einem DC-DC Step Up Modul und einem Spannungswandler. Die Platine besteht, bis auf den Schalter und den Pins, nur aus SMD-Bauteilen, da andere Baugrößen mit diesen Platzvorgaben nicht ausgekommen wären.

**2.3.4 Nebenplatine**



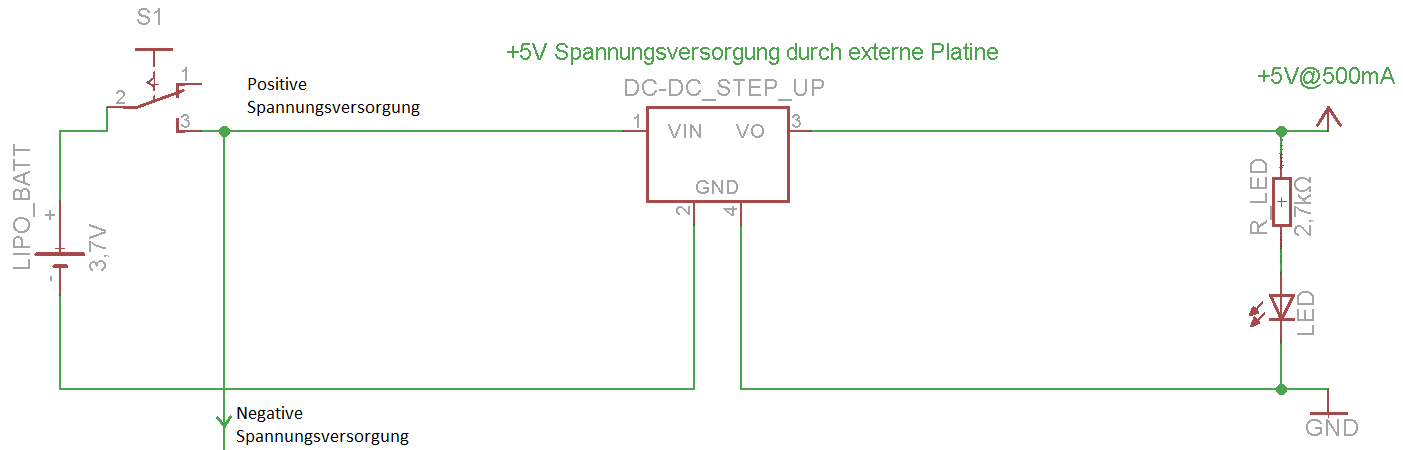
Die Platine wurde mit den Maßen 80x40 gefertigt, da dieser Bereich sich angenehm auf dem Handrücken ausgeht und die Dehnmessstreifen sich damit nahe bei den Anschlusspins befinden. Die Ecken der Nebenplatine wurden ebenfalls freigelassen, damit sie durch eine Bohrung am Handschuh befestigt werden kann. Dies ist die einzige Platine mit zwei Schichten, da die Platine sonst nicht zu fertigen wäre. Auf der Nebenplatine befinden sich die Anschlüsse, sowie die Messbrücken und der Tiefpassfilter zu jedem Dehnmessstreifen. Außerdem sind der Single-Rail- sowie der Quad-Rail-Operationsverstärker in SOT-23 und SOIC-14 neben den Messbrücken platziert. Die Ausgänge der Operationsverstärker führen in den Multiplexer, der in TSSOP-Format auf der rechten Seite der Platine ausgeführt ist. Das Ausgangssignal des Multiplexers wird weiter in den Analog-Digital-Wandler geführt, der in SOIC-8 Baugröße auf der linken Seite der Platine platziert wurde. Damit eine korrekte Referenzspannungsversorgung für den ADC anliegt, wurde der LM4040 physisch knapp neben dem Analog-Digital-Wandler platziert.

Die verlöteten Pins, unten mittig auf der Platine, führen die Spannungsversorgung +3.3V, +5V, -5V & GND in das System und geben die Steuersignale, die der Raspberry der Hauptplatine übertragen hat, auf die jeweiligen zu steuernden Bauteile ab. Die zu steuernden Pins sind die SPI-Schnittstelle am ADC (MISO & SCLK Pins am Raspberry), sowie der Shutdown-Pin (GPIO18 am Raspberry) und die Pins der 3-Bit-Schnittstelle des Multiplexers, die durch die Pins GPIO16, GPIO20 & GPIO21 des Raspberry gesteuert werden.

**2.4 Schaltung und Bauteile**

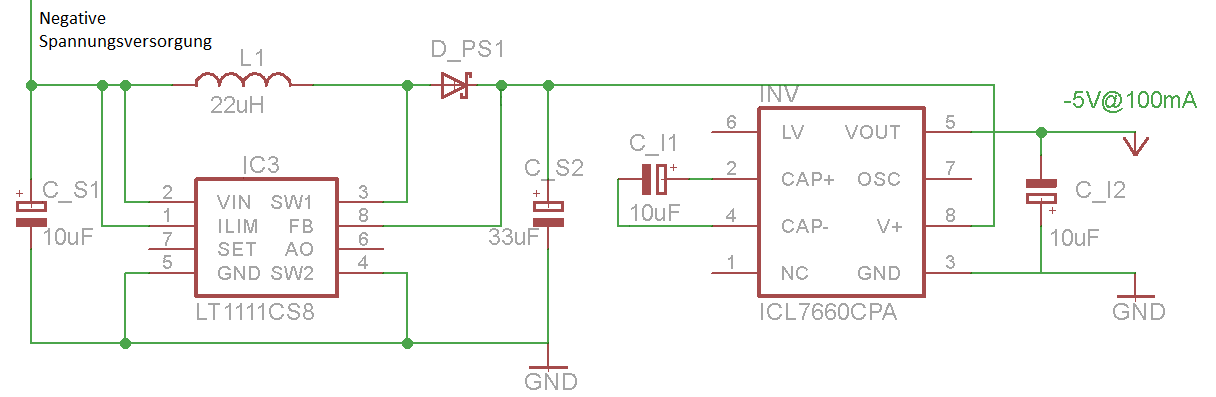
**2.4.1 Spannungsversorgung**

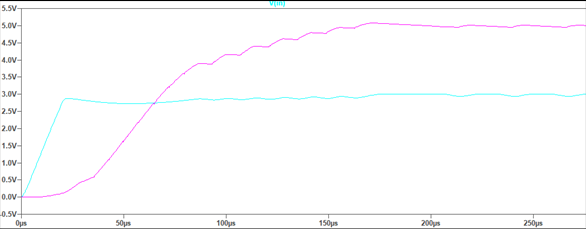
Sowohl die positive als auch die negative Spannungsversorgung werden beide über einen Lithium-Polymer-Akku betrieben, dessen Stromfluss mit einem Kippschalter ein- und ausgeschalten werden kann. Der LiPo-Akku liefert jedoch nur 3,7V. Da aber ein Großteil der IC’s 5V als VDD benötigen, werden zwei DC-DC-Step Up Module verwendet.

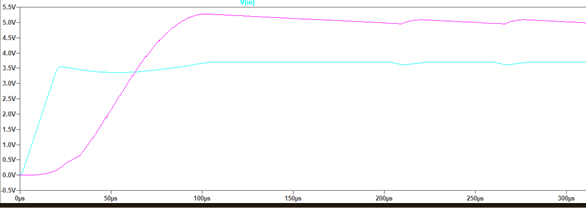
****

Für die positive Spannungsversorgung wurde eine fertige Platine von „Banggood“ zugekauft, da es Spannungen von 1V-5V konstant auf 5V mit max. 500mA, mit hoher Genauigkeit (+0 - +20mV) wandeln kann. Das führt bei 5V zu einer Genauigkeit von Abweichung, welche für unsere Zwecke vollkommen ausreicht. Dass das Modul Eingangsspannungen von 1-5V umwandeln kann hat vor allem den Vorteil, dass das Step Up Modul trotzdem konstant 5V liefert ohne große Änderungen bei der Ausgangsspannung feststellen zu können. Da der Akku je nach Ladezustand, 3V-3,7V liefert, war dies eine nötige Maßnahme, damit die IC’s nicht mit weniger als 5V versorgt werden können. Das Modul wurde auch wegen seinem hohen Wirkungsgrad gewählt, der laut Hersteller 96% betragen sollte, was durch Belastungstests überprüft wurde. Durch den billigen Preis der Platine, war dies die beste Option für dieses Projekt. Da der aktive Stromverbrauch des Projekts ca. 230mA beträgt, ist diese Platine vollkommen ausreichend für unsere Zwecke.

Weiters ist noch eine Anzeige LED nach der positiven Spannungsversorgung angebracht, die jedoch nur max. 2mA benötigt. Der Vorwiderstand wurde mit dem Ohm’schen Gesetz berechnet. Der Widerstand wurde auf den nächsten sinnvollen, in 0402 erhältlichen, Wert gerundet => RLED = 2,7kΩ



Die negative Spannungsversorgung muss bei diesem Projekt nur die Operationsverstärker versorgen, weshalb der LT1111-CS8 Chip von „Linear Technology“ gewählt wurde, der zwar nur 100mA liefern kann, aber dafür einen Wirkungsgrad von bis zu 99% hat, welcher sich hervorragend für ein Akku-betriebenes System eignet. Da der Hersteller seinen Chip auf einer LT-Spice Datenbank hochgeladen hat, wurde die StepUp-Schaltung mit verschiedenen Eingangsspannungen simuliert. 



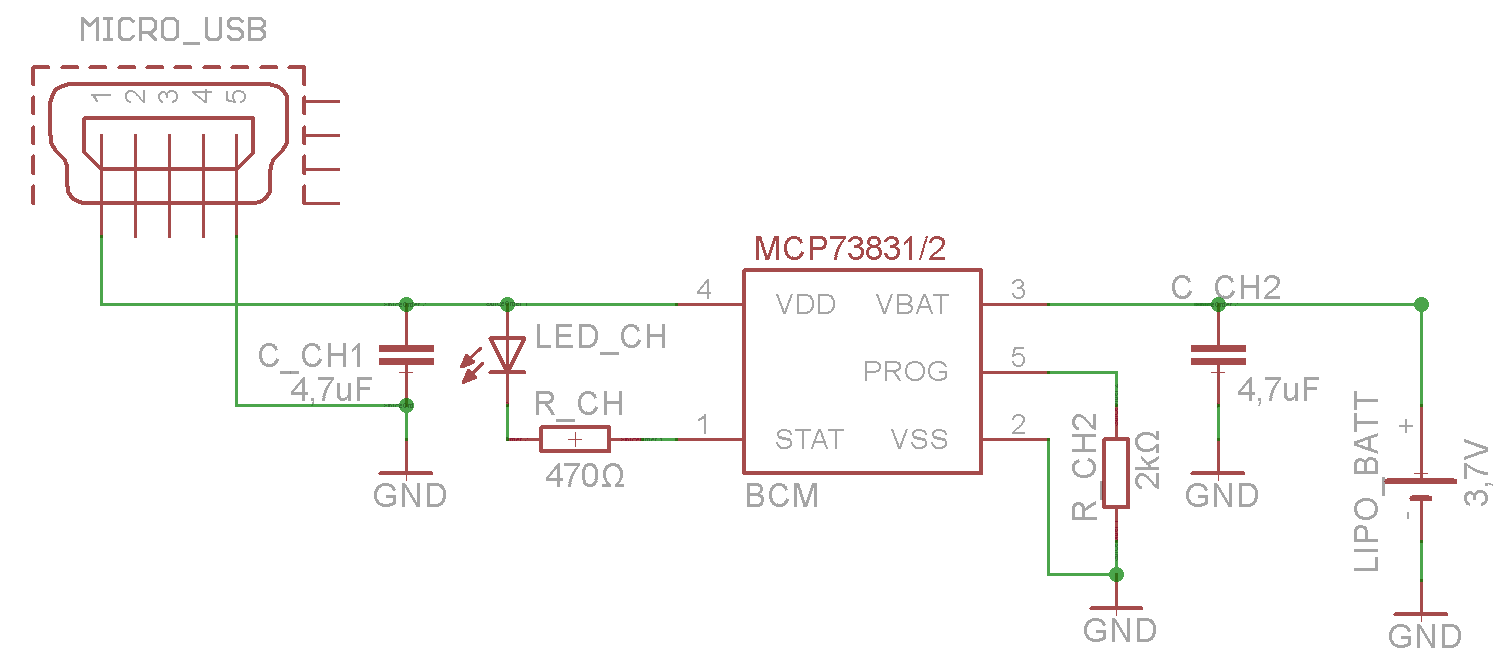
Obwohl die Ausgangsspannung bei 3,7V Eingangsspannung ein kleines Überschwingen aufweist, regelt der Chip diesen Wert jedoch nach ca. 200µs wieder ein.

Die beiden Elektrolyt-Kondensatoren CS1 und CS2 werden dabei als sogenannte   
„Ripple-Filter“ eingesetzt, da ohne sie starke Schwankungen bei der Ausgangsspannung entstehen würden. Die Kondensatoren wurden dabei nach Herstellerangaben vorgegeben.

Da der LT1111-CS8 jedoch nur positive Spannungen liefern kann, wurde ein Spannungswandler-Chip in die Schaltung integriert. Der ICL7660A wurde wegen seiner hohen Effizienz von 99% und wegen seines simplen Designs gewählt. Mit diesem Chip werden nur noch zwei externe Elektrolyt-Kondensatoren benötigt, um positive 5V in negative 5V umzuwandeln. Mit einem Stromverbrauch von gerade einmal 20µA ist dieser Chip perfekt für ein Akku-betriebenes Design, bei dem der Stromverbrauch der einzelnen Bauteile kritisch ist.

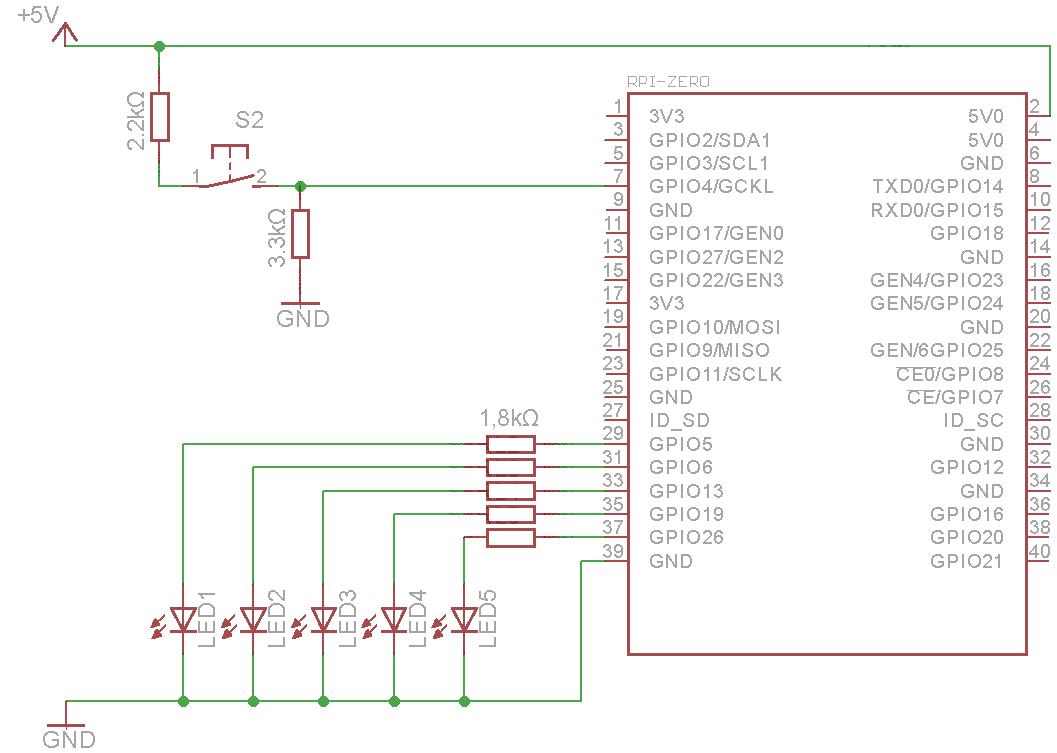
**2.4.2 LiPo-Akku und Ladeschaltung**

Dieses Projekt sollte laut dem ersten Konzept mit AAA Batterien versorgt werde, welche jedoch den großen Nachteil haben, dass sie nicht wieder aufladbar sind. Es wurde schnell ein Lithium-Polymer Akku gefunden, der den Anforderungen des Projekts entsprach. Der Vorteil eines LiPo-Akkus ist, dass er deutlich mehr Speicherkapazität als z.B. ein Lithium-Ionen-Akku hat. Deshalb konnte ein Akku gewählt werden, der 1320mAh Kapazität beinhaltet, aber trotzdem eine Baugröße von 37,5x61x6 mm aufweist, was bei einem mobilen Gerät essenziell ist. Auch wiegt der Akku nur 26g, was dem Benutzer des Handschuhs fast nicht auffallen wird.



Als sogenannten „Battery-Charge-Manager“ wurde der MCP73831/2 von „Microchip“ gewählt, da er in der platzsparenden Baugröße SOT-23-5 zu einem sehr günstigen Preis zu erwerben ist. Außerdem waren viele Individualisierungsmöglichkeiten, sogenannte „Presets“, wie die Ausgangsspannung, „End-of-Charge-Control“, sowie der „Charge-Status-Output“ auf dieses Projekt abstimmbar. Da der LiPo-Akku genau 4,2V als Ladespannung benötigt, wurden 4,2V bei 500mA am Chip eingestellt. Durch den Kondensator CCH1 entsteht am Beginn der Ladezeit ein „Drop-Down“ welcher dem Ladevorgang jedoch nicht schadet. Der zweite Kondensator CCH2 dient nur als „Rippl-Filter“, damit der Ladevorgang konstant auf 4,2V stattfindet. Die Widerstände, die für den „Preset“ verwendet werden, wurden aus dem Datenblatt entnommen. Weiters wurde noch eine herstellerempfohlene LED eingebaut, die den Ladestatus anzeigen soll. Das Gerät wird über einen Micro-USB Stecker Type-B geladen, der an einem Standard 5V bei min. 500mA Netzteil angeschlossen werden kann.

**2.4.3 Modi-Anzeige**

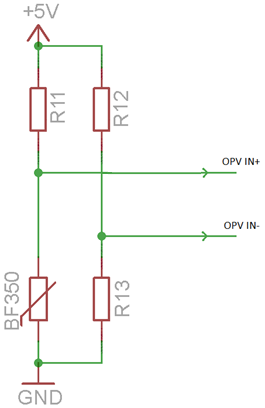


Die Modi-Anzeige besteht aus einem Druckknopf in einem Spannungsteiler. Durch den Spannungsteiler werden nur 3V in den GPIO4-Pin eingelesen, welche vollkommen für ein „Logic High“, also eine „1“ in der Booleschen Algebra reicht. Da der Raspberry Pi auf CMOS Schaltlogik basiert, reicht ein Eingangssignal von 2V für einen „logisch hohen“ Input. Ein weiterer Vorteil dieses Spannungsteilers ist, dass bei Nichtbetätigung des Schalters der 3,3kΩ Widerstand als „Pull-Down-Resistor“ wirkt. Das heißt, wenn der Schalter nicht verbunden ist, wird der Eingang automatisch auf „Ground“ gezogen wird.

Die Anzeige-LED’s sind je mit einem GPIO-Ausgang verbunden, der 3,3V liefert. Je nach gewähltem Modus leuchtet eine LED auf, die immer nur max. 2mA zieht. Um zu sichern, dass die LED nur mit dem passenden Strom beliefert werden, wurden sogenannte „Current-Limiting“-Widerstände eingesetzt, die durch das Ohm’sche Gesetz berechnet wurden.

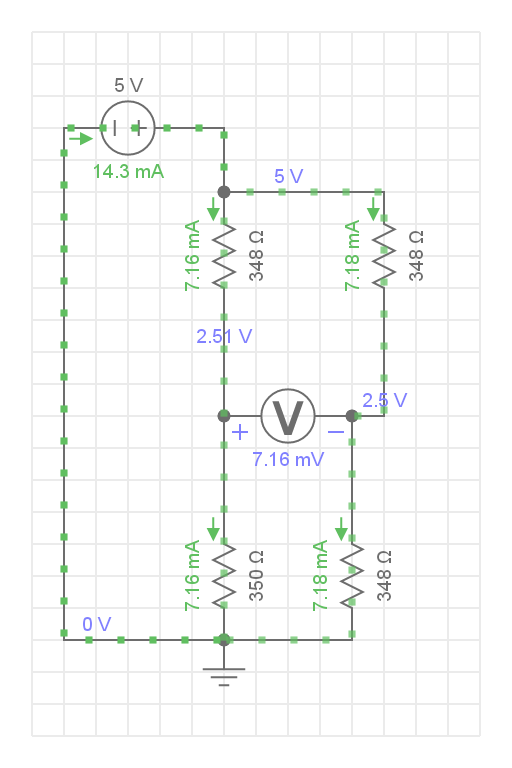
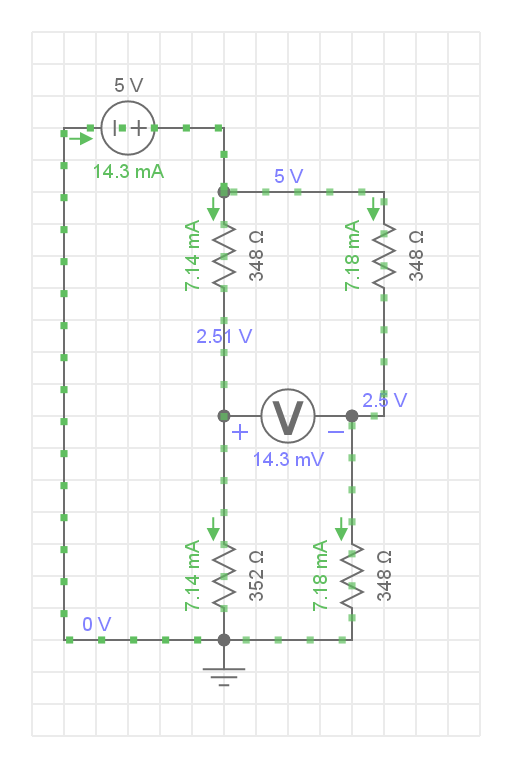
Gewählt wurde ein 1,8kΩ Widerstand, da er in 0402-Baugröße erhältlich ist.

**2.4.4 Dehnmessstreifen**



Die bei diesem Projekt wurden BF350 verwendet, welche genormte Präzisions-Dehnmessstreifen sind, die einen Standardwiderstand von 350Ω und als Grundmaterial Modifiziertes Phenolharz haben. Der DMS hat einen Abweichungsgrad unter ±0.1Ω, was bei einer genauen Messung wie bei diesem Projekt notwendig war. Die Widerstandsänderung der Dehnmessstreifen wurden nicht durch eine Zugbeanspruchung, sondern nur durch den Abknickungswinkel herbeigeführt, daher waren die Angaben des „Gauge Faktors“ nicht relevant für das Projekt. Durch Tests wurde eine maximale Widerstandsänderung von +3Ω festgestellt, bei der der Finger vollkommen angezogen war.

Die Dehnmessstreifen wurden durch eine sogenannte Wheatstone’sche Messbrücke ausgewertet, deren Widerstände nicht auch 350Ω besitzen, da es keine SMD Widerstände am Markt gab, die auf diesen Wert passten. Stattdessen wurden 348Ω Widerstände im 0603-Format gewählt, da sie die ähnlichsten Werte erzielten, die es in einer Baugröße unter 1206 gegeben hat. Die Messbrücke wurde durch Nachbauen der Schaltung und durch Programme wie LTSpice und EveryCircuit getestet und auf Fehler überprüft. Die Berechnung des gezogenen Stroms wurde durch EveryCircuit kontrolliert und wird einfachheitshalber nur als Screenshot dargestellt.

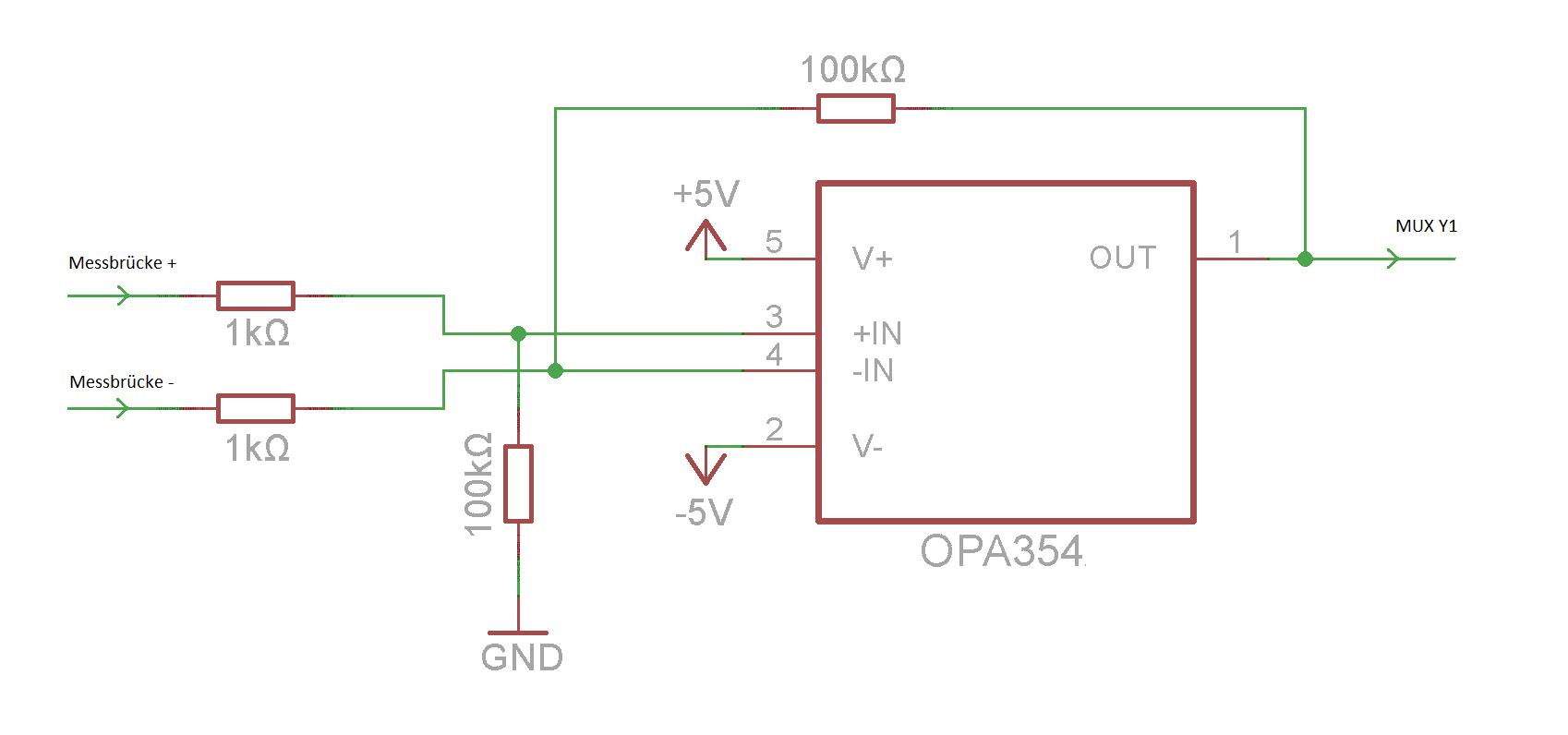


Die

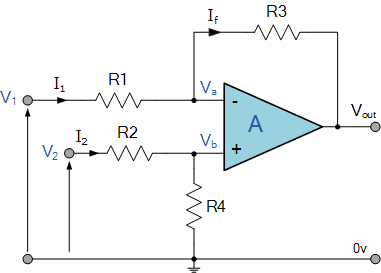
Schaltung auf der linken Seite beschreibt die Messbrücke im Normalzustand, also ohne abgeknickten DMS. Auf ihr ist erkennbar, dass durch die Differenzen der Widerstandswerte zwischen DMS und den anderen Widerständen, auch ohne Belastung ein Strom von ca. 7,2mV anliegt.

Auf der rechten Graphik wird der maximale Belastungszustand des DMS dargestellt, bei dem er einen Wert von 352Ω besitzt. Durch Berechnungen wurde eine Steigerung der Spannung von ca. 3,6mV/ΔΩ erzielt. Durch einen Vergrößerungsfaktor von 100, mittels Differenzenverstärker, wird ein Wert von 0,7V bis 1,5V an den ADC abgegeben, welcher Werte in dieser Größenordnung gut verarbeiten kann.

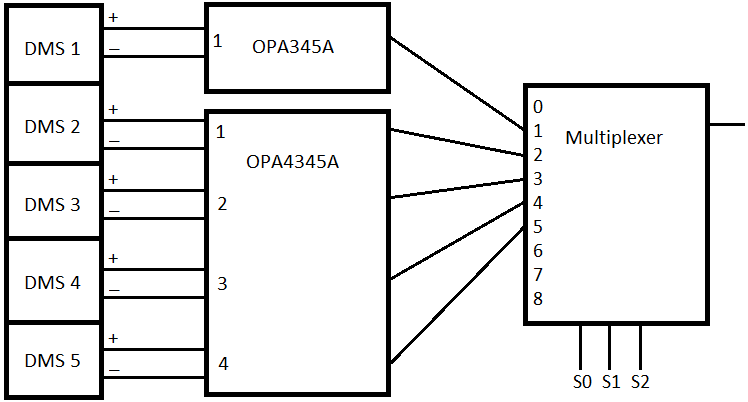
**2.4.5 Differenzenverstärker**



Für die Auswertung der Signale der Dehnmessstreifen, wurden Operationsverstärker mit einer differenzverstärkenden Schaltung gewählt. Durch dieses Auswertungsprinzip wird die Differenz zwischen den beiden Eingangssignalen ermittelt und gleich mit einer wählbaren Verstärkung multipliziert. So werden die Differenzen, die sich im mV Bereich befinden, in diesem Projekt mit dem Faktor 100 verstärkt. Die gewählten Widerstände wurden ausfolgender Formel ermittelt.



Da dieses Projekt aber fünf Dehnmessstreifen benötigt, wurden zwei OPV-Chips gewählt. Der OPA354A und der OPA4354 sind „Low Noise“ Operationsverstärker mit einem niedrigen Stromverbrauch von ca. 5mA im aktiven Zustand, die in kompakten SOT-23-5 und TSSOP-14 Packungen erhältlich sind. Der OPA354A ist ein Single Rail-OPV, der das erste Messbrückensignal erhält und der OPA4354 ein Quad Rail-OPV, der die Signale 2 bis 5 verstärkt.



Die obere Schematik beschreibt, wie die einzelnen Dehnmesstreifen mit den Operationsverstärkern zusammenhängen und wie die Ausgänge der OPV’s mit dem Multiplexer verbunden sind. Es mussten dabei 2 Chips gewählt werden, da es keinen effektiv verwendbaren OPV-Chip gab, der 5 oder mehr Ausgänge eingebaut hat.

**2.4.6** **RC-Tiefpassfilter**

Bei Internetrecherchen wurde oft empfohlen, einen Tiefpassfilter nach Messbrücken einzubauen, da die genauen Messwerte durch höhere Frequenzen gestört werden können. Es wurde ein RC-Tiefpass gewählt, da ein LR-Tiefpass zu viel Platz auf der Platine verbraucht hätte und spezifische Spulen allgemein schlechter erhältlich sind als Kondensatoren.

Die Wheatstonebrücke wurde als Testschaltung aufgebaut und eine Spektrum-Analyse wurde mittels eines Oszilloskops im Bereich von 1Hz-10kHz durchgeführt. Daraus hat sich ergeben, dass eine „Cutoff-Frequency“ bei ca. 16Hz die beste Lösung für die Messbrücke ist.

Durch die Formel: wurde eine Top-Down Umrechnung durchgeführt, um geeignete Werte für den Kondensator und den Widerstand zu ermitteln.

Da ist, wurde daraus die Formel: .

Durch weiteres Umformen konnte die Formel erstellt werden.

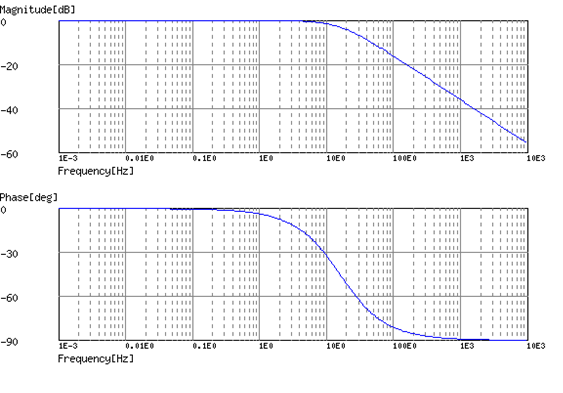
Durch Einsetzen entstand .

Es wurde ein Kondensator von 0,1µF gewählt, damit so ein großer, gut definierbarer Widerstand genommen werden konnte. Aus diesen Annahmen ergaben sich die Werte von:

R= 100kΩ und C=0,1µF

Die durch diese Werte erreichte „Cutoff-Frequency“, wurde durch erneutes Einsetzen in die Grundformel ermittelt.

Daraus wurde mit einem Online Plotter von „Okawa“ folgendes Bodediagramm erstellt:

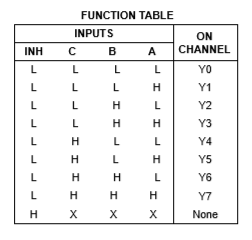


**2.4.7 Analoger Multiplexer**

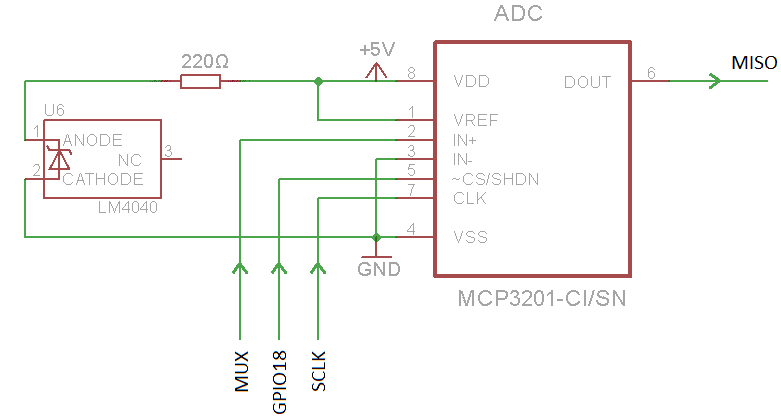


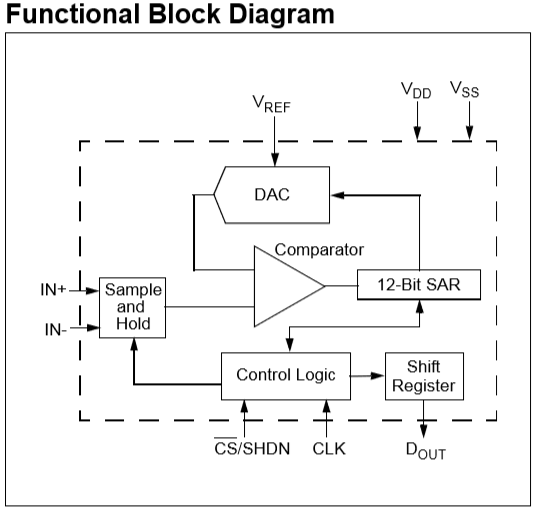
Der analoge Multiplexer wird für die Signalvereinzelung verwendet, die durch eine 3-Bit Schnittstelle kontrolliert werden. Für diese Aufgabe wurde der SN74HC4851-Q1 von „Texas Instruments“ gewählt, da er der kostengünstigste Multiplexer war, den es in TSSOP-PW Bauform gab und der analoge Signale verwerten kann. Mit seiner geringen Schaltzeit von ca. 9ns bei einem Signalwechsel, war er optimal für ein zeitabhängiges Steuersignal von dem Raspberry.

Der Multiplexer erhält seine Signale durch die Ausgänge der OPV’s, wie in Abbildung 34 gezeigt, und leitet dann das gewählte Signal an den Analog Digital Wandler weiter.

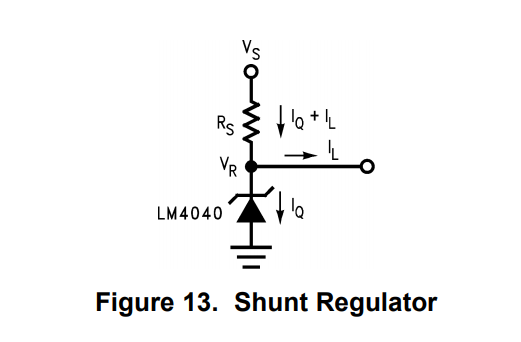
Welches Signal weitergegeben wird, hängt von der Kombination der Logischen Eingänge A, B & C ab. Wie in der Funktionstabelle beschrieben, werden bei diesem Projekt die Eingänge von Y1 – Y5 benutzt, da fünf Signale verarbeitet werden müssen. Der Eingang Y0 wurde dabei absichtlich frei gelassen, da der Raspberry nach dem Einschalten erst Zeit zum Hochfahren braucht und so alle Eingangssignale auf 0 wären. Dadurch würde der Multiplexer aber schon Werte an den ADC weitergeben, was zu Fehlern führen könnte. Der Inhibitor Eingang INH, der einen „normally low“-Zustand hat, wurde bei diesem Projekt auf Ground gelegt, da er das Programm nur komplizierter und unübersichtlicher machen würde.

**2.4.8 Analog Digital Wandler**



Als Analog-Digital-Wandler (ADC) wurde der MCP3201 von Microchip gewählt, da er ein günstiger 12-Bit ADC mit SPI-Schnittstelle ist und in SOIC-8 Baugröße erhältlich ist. Die Schwankung der Messwerte liegt bei ±2 LSB („Least-Significant-Bit), was bei diesem Projekt vollkommend ausreichend ist. Mit einer Sample Rate von 100ksps bei 5V Versorgung und einer Anpassungszeit von nur 12 Clock Cycles, kann der ADC akkurate Einsichten in den aktuellen Spannungszustand geben. Der ADC besitzt außerdem einen CS/SHDN Eingang, der „normally open“ ist, also standardmäßig ein Logisches „HIGH“ annimmt und den Eingang des ADC automatisch blockiert. Dieser PIN wird durch den GPIO18-Pin des Raspberry, dieser kann je nach Belieben „HIGH“ oder „LOW“ ausgeben und so den CS/SHDN-Pin ein- oder ausschalten, was einen Eingang am ADC ermöglicht. Der Positive Sample Eingang am ADC (IN+) wird mit dem Spannungswert des Multiplexers zum Zeitpunkt des Messvorganges versorgt. Der ADC wird durch den Raspberry gesteuert, der die SPI-Pins MISO & SCLK nutzen kann um dem ADC einen Takt vorzugeben (durch SCLK Leitung) und die, durch den ADC, erhaltenen Digitalen Werte in den Raspberry einzulesen (durch MISO Leitung). Die innere Schaltung des ADC wird in diesem Funktionsdiagramm dargestellt.

Die Referenzspannung des Analog-Digital-Wandlers muss sehr genau sein, um ein korrektes Umwandeln zu ermöglichen. Da die allgemeine Spannungsversorgung jedoch nicht immer exakt 5V ist, wurde ein LM4040-5.0 eingebaut, der die Spannung am Referenzeingang regeln soll. Die Spannungsreferenz hat eine Genauigkeit von ±0,1% was einem A-grade nach Texas Instruments Standard entspricht und ist als SOT-23-3 Package erhältlich.

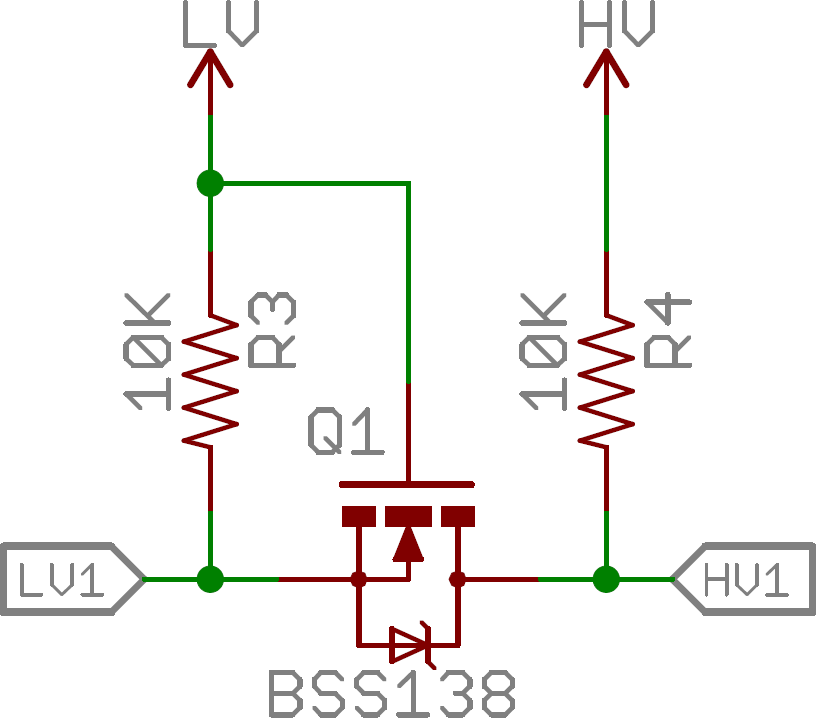
Der LM4040 wird dabei als sogenannter   
„Shunt Regulator“ eingesetzt, dessen Vorwiderstand RS durch folgende Formel berechnet werden konnte.

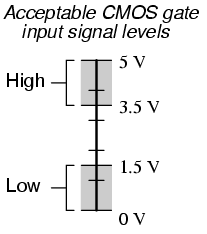
Gewählt wurde ein 220Ω Widerstand, da diese Rechnung die Maximalwerte der Versorgungsspannung und des gezogenen Lastenstroms sind und dieser Wert als 0402-Baugröße erhältlich war.

Durch die genaue Referenzspannung von 5V kann der LSB-Wert bestimmt werden.

Somit kann der einstellbare Grenzwert für die DMS-Auswertung leicht im Programm verwertet werden, da man die kleinste Spannungsstufe kennt. Da der Ausgabewert des ADC nur um ±2 LSB schwanken kann, ist es ein geringer Störfaktor bei der Auswertung, der bei dem Programm in Betracht gezogen wurde.

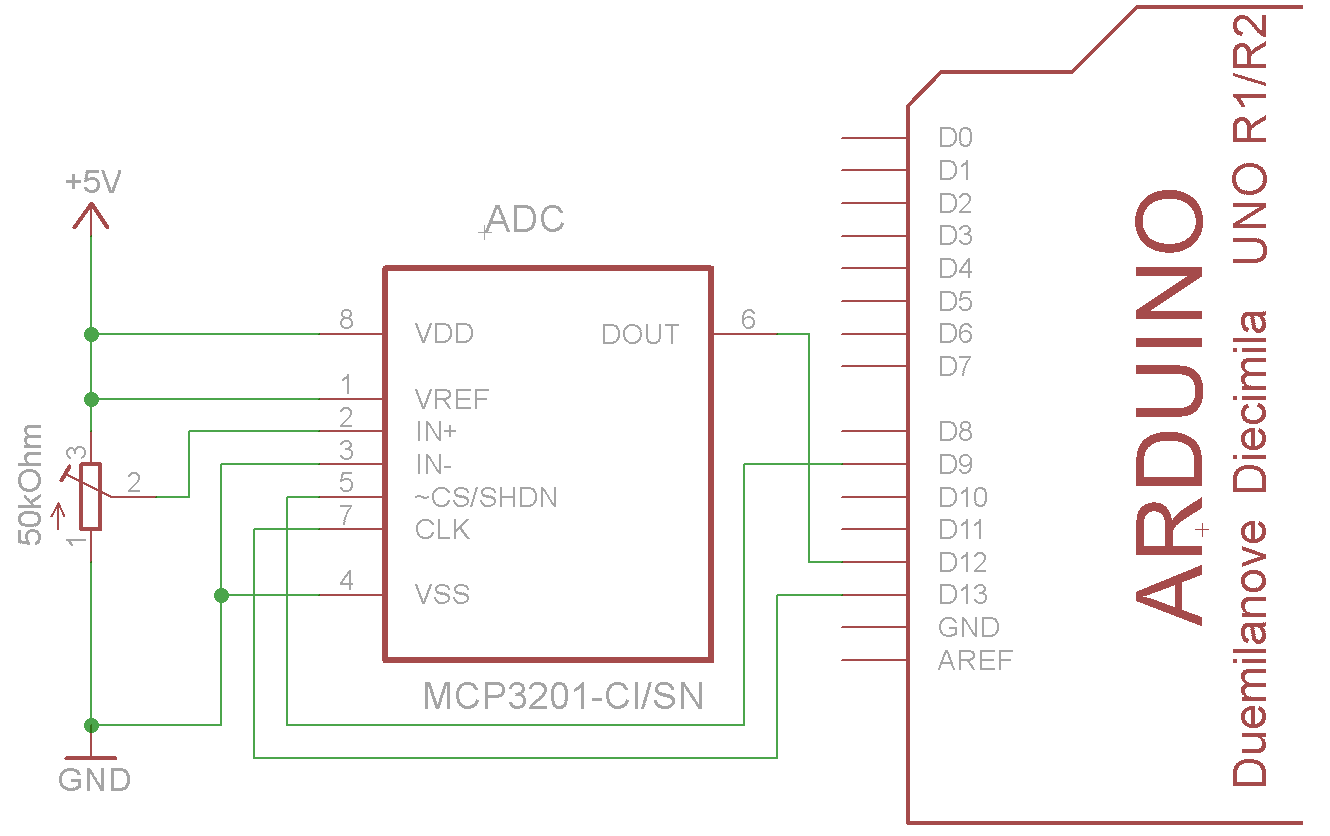
**2.4.9 Logik-Level-Konverter**



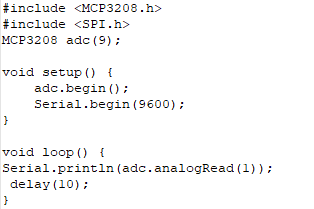
Da die Logischen Eingänge des Multiplexers A, B & C und der CS/SHDN Eingang des Analog-Digital-Wandlers auf 5V CMOS-Logik basieren und die GPIO-Pins des Raspberry nur auf 3,3V CMOS-Logik basieren, können sie ohne Logik-Level-Konverter keine „HIGH“ Signale empfangen, da 5V CMOS-Logik erst ab 3,5V ein „HIGH“ einliest. Deswegen wurde an den benötigten Stellen eine Konverter- Schaltung implementiert, die diese Differenzen ausgleichen soll. Durch den Logik-Level-Konverter werden 5V an den High Voltage Output ausgegeben, wenn der Low Voltage Input ein Signal ausgibt. Durch dieses System können Signale mit geringem Stromverbrauch und durch nur 3 externe Bauteile 5V-Logik mit einem Raspberry ansteuern.

**2.4.10 Testverfahren**

ADC:

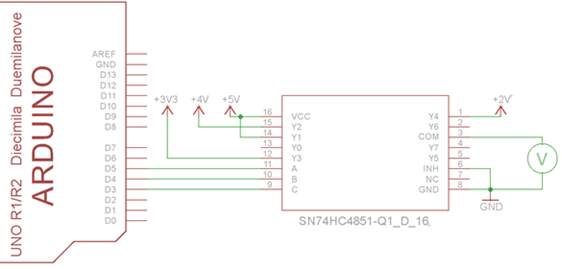
****

Der ADC wurde mithilfe eines Arduino UNO und einem 50kΩ Potentiometer durchgeführt. Mithilfe einer SPI- und MCP32xx-Library wurde die Funktionalität des ADC und dessen LSB-Schwankung überprüft.

Durch dieses Programm wurden die Binärausgaben am Serial Plotter ausgegeben.

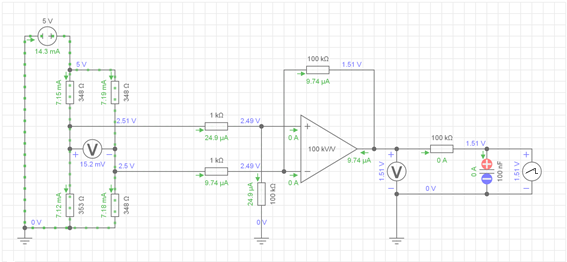
Es wurde überprüft, ob eine maximale Schwankung von ±2 LSB vorhanden war.

Multiplexer:



Der Multiplexer wurde dadurch getestet, dass verschiede Spannungen, erzeugt durch verschiedene Netzgeräte, an den verwendeten Eingängen des MUX angelegt wurden.  
Die 3-Bit-Schnittstelle wurde durch den Befehl **digitalWrite(5, HIGH);** gesteuert, indem verschiedene Eingangskombinationen getestet wurden. Am Ausgang des Multiplexers wurde mittels eines Multimeters die aktuelle Spannung gemessen, die ausgegeben wurde. Durch Umschalten der Steuersignale wurde mittels Wahrheitstabelle und Ausgangsspannung überprüft, ob der Multiplexer ein exaktes Signal ausgibt. Dies konnte damit kontrolliert werden, dass verschiedene Spannungen an den Eingängen anliegen.

Messbrücke Differenzenverstärker & Tiefpassfilter



Diese Schaltung wurde in EveryCircuit getestet und danach in natura aufgebaut, wobei der verwendete OPV wieder ein LM741-Chip war. Der Ausgang dieser Schaltung wurde mit einem Oszilloskop gemessen und auf Fehler bzw. eine Fehlerminderung durch den Tiefpass überprüft.

Folgende, nicht spezifisch erwähnten, Schaltungsteile oder Baugruppen

* Spannungsversorgung
* LED’s bei Modi-Anzeige
* Battery-Charge-Control
* Druckknopf mit Spannungsteiler
* Referenzspannungsquelle
* Logik-Level-Konverter

wurden direkt nach Schaltungsvorgaben mit einem Netzgerät als Spannungsversorgung (mit der je benötigten Spannung) getestet und mit EveryCircuit überprüft.

**2.5 Berechnungen:**

|  |  |  |
| --- | --- | --- |
| **Stromverbrauch passiv/aktiv:** | | |
|  |  |  |
| **//** | **aktiv** | **passiv** |
| **[mA]** | **[mA]** |
| **Messbrücken** | **71,5** | **0** |
| **RPI Zero W** | **130** | **120** |
| **Spannungswandler** | **0,08** | **0,08** |
| **Anzeige LED** | **2** | **2** |
| **Modi Schalter** | **2,3** | **0** |
| **ADC** | **0,002** | **0,4** |
| **LM4040** | **0,06** | **0,06** |
| **Logik-Konverter** | **3,32** | **0** |
| **Single OPV** | **4,9** | **0,000003** |
| **Quad OPV** | **4,9** | **0,000003** |
| **MUX** | **0,002** | **0,0001** |
| **//** | **Effizienz** | **Effizienz** |
| **[%]** | **[%]** |
| **DC-DC StepUp** | **0,96** | **0,96** |
| **Gesamtverbrauch:** | **227,82656** | **127,44171** |

Der Stromverbrauch des gesamten Projektes im aktiven und passiven Zustand wurde aus Werten zusammengetragen, die der Hersteller der jeweiligen Bauteile als typischen Verbrauch angegeben hat, und aus Berechnungen wie z.B. die Stromaufnahme der Messbrücke.

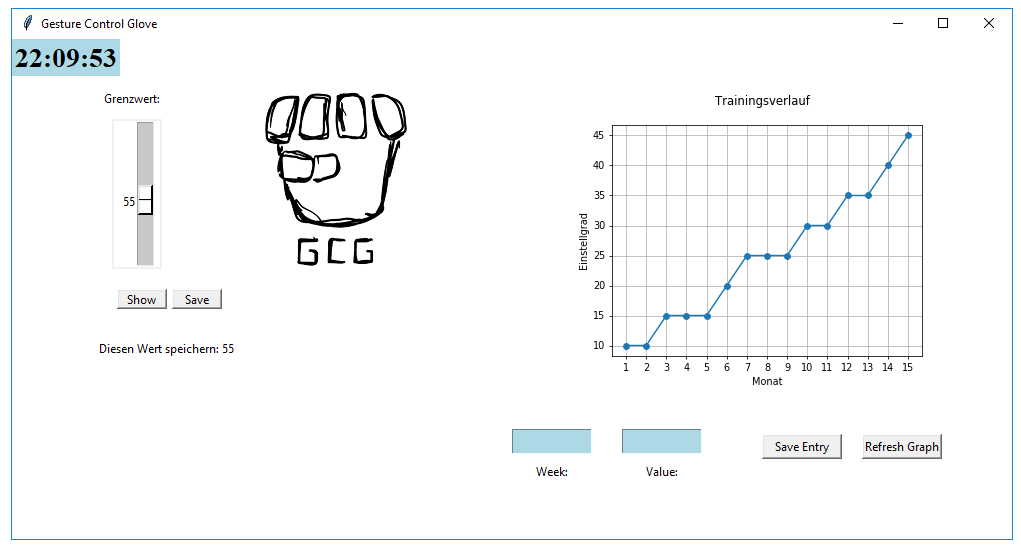
|  |  |  |
| --- | --- | --- |
| **Akku-Betriebsdauer:** | | |
|  |  |  |
| **Dauer aktiv** | **Dauer passiv** | **Kapazität** |
| **[h]** | **[h]** | **[mAh]** |
| **1** | **1,5** | **190** |
| **2** | **4** | **500** |
| **4,5** | **8** | **1000** |
| **6** | **10,5** | **1320** |

Die Akku-Betriebsdauer wurde mit verschiedenen Kapazitäten berechnet, da ein geeigneter LiPo-Akku für dieses Projekt essenziell ist. Es wurde für dieses Projekt ein 1320mAh Akku gewählt, da sechs Stunden aktive Benutzungsdauer für einen Tag optimal ausreichen. Die Betriebsdauer wurde durch folgende Formeln berechnet:

**4 Graphical User Interface**

**4.1 GUI allgemein**

Die Aufgabe der Graphischen Benutzerfläche oder auch kurz GUI ist, dass der Benutzer die Möglichkeit hat, das Projekt ~~auf~~ an individuelle Bedürfnisse anzupassen. Das Programm wurde mit Python 3.7 erstellt, da der Raspberry Pi diese Sprache schon installiert hat und damit die Kommunikationsprobleme wegfallen, die bei anderen Sprachen vorhanden wären. Außerdem bietet Python „Tkinter“, welche ein einfaches Erstellen eines GUI ermöglicht, sowie weitere nützliche vorgefertigte Bibliotheken an, die bei einem solchen Projekt essenziell sind. Weiters bietet diese Sprache die Möglichkeit, sehr leicht andere Dateien zu beschreiben bzw. sie auszulesen, was eine Kommunikation mit dem Hauptprogramm ~~sehr~~ einfach gestaltet. Durch einen im GUI implementierten Schieberegler kann der Grenzwert des Dehnmessstreifens festgelegt werden, bei dem ein logisches „high“ ausgegeben wird. Da die maximale Widerstandsänderung des DMS 2Ω beträgt, kann durch einen prozentuellen Teiler, bestimmt durch den Status des Schiebereglers, der Grenzwert in eine Textdatei gespeichert werden, welche leicht durch das Hauptprogramm aufgerufen werden kann. Die graphische Benutzeroberfläche bietet außerdem die Möglichkeit, dass der Benutzer des Handschuhs seinen Trainingsverlauf eingeben kann, welcher durch eine, mit „matplotlib“ erzeugte Graphik, visualisiert werden kann. Die vom Nutzer eingelesenen Werte werden in einer csv-Datei („comma seperatet values“) gespeichert, sodass ein erneutes Eingeben nicht nötig ist.**4.2 GUI Oberfläche**

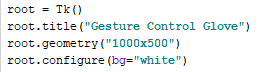


Die Benutzeroberfläche wird auf ein Fenster beschränkt, welches in zwei funktionelle Gruppen gegliedert wurde. Auf der linken Seite befindet sich ein Schieberegler, mit welchem der Benutzer in Fünferschritten den Grenzwert festlegen kann. Mit dem Knopf „Show“ wird noch einmal nachgefragt, ob der Benutzer wirklich den gerade eingestellten Wert einlesen will. Durch den „Save“-Knopf wird die Eingabe bestätigt und der aktuelle Status des Schiebereglers wird in die „Value.txt“ geschrieben. Auf der rechten Seite befindet sich eine Graphik, die den Trainingsverlauf des Benutzers darstellen soll. Mit den Eingabefeldern „Week“ und „Value“ soll der verwendete Trainingsgrad und die derzeitige Woche nach Trainingsbeginn eingetragen werden. Mit dem „Save Entry“-Knopf wird die Benutzereingabe auf Fehler überprüft und anschließend in eine csv-Datei eingelesen, welche alle vorherig eingetragenen Werte bereits gespeichert hat. Durch den „Refresh Graph“-Knopf hat der Benutzer die Möglichkeit, den Graphen zu aktualisieren und die neu eingetragenen Werte mit den vorherigen zu vergleichen. Das Ganze wird mit einer Echtzeituhr in der oberen linken Ecke und dem Logo der Diplomarbeit abgerundet, um das GUI anschaulicher zu gestalten.

**4.3 Struktur des Codes**

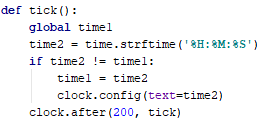
**4.3.1 Fenster & Allgemeines**

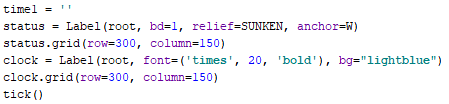
Das Fenster wurde mit „Tkinter“ als „root“ erstellt. Der Titel wurde auf den Diplomarbeitstitel umbenannt und die Maße wurden auf 1000x500 Pixel festgelegt, da so jeder Display das GUI gut anzeigen kann. Der Hintergrund wurde Weiß gewählt, da so die anderen Funktionen gut zu erkennen sind.

Das Fenster wurde durch Befehle so eingestellt, dass sich das Fenster immer ~~auf~~ an den aktuellen Bildschirm anpasst und immer zentriert ist. Dies wurde mit der „os“-Bibliothek bewerkstelligt, da sie die Bildschirmweite und -höhe des verwendeten Displays auslesen kann.



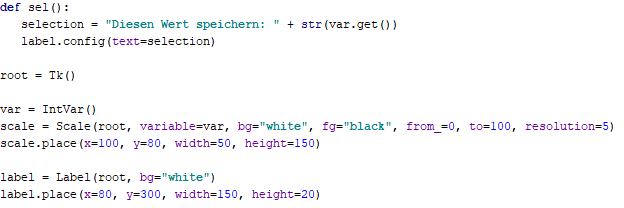
Weiters wurde das Diplomarbeitslogo als Label eingefügt und mit dem „label.place“-Befehl in Position gebracht. Als extra Funktion des GUI wurde eine Echtzeituhr integriert, welche durch die „time“-Bibliothek erstellt wurde. Die Uhr besteht aus einer Funktion, die alle 200 Ticks eine Sekunde zu der aktuellen Zeit, welche als globale Variable definiert wurde, dazurechnet und die alte Zeit überschreibt. Das Ganze wird in einem Label dargestellt, welches mit einem hellblauen Hintergrund unterlegt wurde.



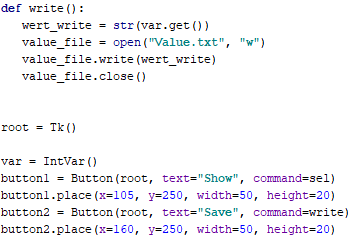


**4.3.2 Einstellgrad**

Der Einstellgrad ist ein prozentualer Wert des Grenzwertes, der durch eine Skala einstellbar ist. Die Skala wurde auf den Bereich von 0 bis 100%, bei einer Schrittweite von 5% ausgelegt. Der Ausgabewert der Skala wurde mittels einer Funktion gelöst, die beim betätigen des „Show“-Knopfes, den aktuellen Status der Skala ausliest und mittels eines Labels anzeigt.

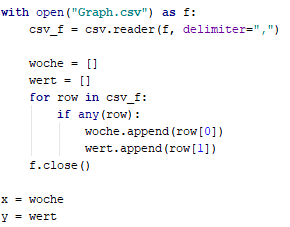


Durch das Betätigen des „Save“-Knopfes wird eine weitere Funktion aufgerufen, die den aktuellen Wert ausliest, die „Value.txt“ öffnet und den aktuellen Status darin speichert. Die Textdatei wurde dabei mit einem sogenannten „write“-Befehl beschrieben, welcher alle vorherigen Daten löscht und nur noch den aktuellen Wert als „Integer“ speichert, welcher dann leicht vom Hauptprogramm übernommen werden kann.

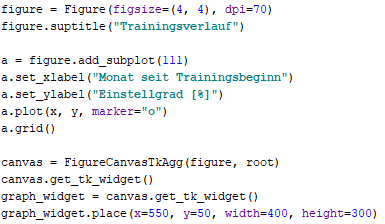


**4.3.3 Graph**

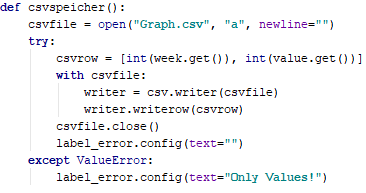
Der Graph wurde mit den „Figure“ und der „FigureCanvasTkAgg“-Funktionen erstellt. Diese wurden dabei gewählt, weil damit sehr viele Individualisierung-Optionen bei einer simplen Grundstruktur erstellt werden können. Als erstes wird die „Graph.csv“-Datei von einem Algorithmus ausgelesen und die einzelnen Spalten in zwei verschiedene Listen verpackt. Bei dieser Auslesung wird durch einen „If-Befehl“ kontrolliert, ob leere Spalten in der Datei sind und lässt Sie gegebenenfalls aus. Von diesen Werten wurden die x- und y-Achse des Graphen erstellt, welche als Variablen x und y eingespeichert werden.



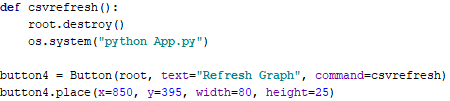
Danach wird die eigentliche Figur (in diesem Fall eine Linie) erstellt, die Parameter des Graphen festgelegt und diverse Beschriftungen hinzugefügt. Mit dem „plot“ Befehl wird ein Bild erstellt, welches den Graphen darstellt. Da dieses Bild aber nicht einfach so im GUI angezeigt wird, wurde ein sogenannter „Canvas“ erstellt, in dem das Bild integriert und somit sichtbar ist. Als letztes wurden noch die Parameter des „Canvas“ eingestellt, sodass der Graph sich an der korrekten Stelle befindet.



Da der Graph veränderbar sein muss, wurden zwei Eingabefelder eingebaut, mit denen neue Werte in die csv-Datei schreiben kann. Die beiden Felder werden durch die „csvspeicher“ Funktion als erstes durch „try-except“ auf korrekte Eingabe überprüft und dann durch den „csv.writer“ Befehl zu der „Graph.csv“ Datei hinzugefügt. Die Eingabekontrolle wird auf den „ValueError“-Fehler getestet, der erkennt ob entweder verwertbare Zahlen oder unverwertbare „Strings“ und Leerzeichen eingegeben wurden. Wenn keine Zahlen eingegeben wurden, zeigt das GUI eine Fehlermeldung an, welche bei einer korrekten Eingabe wieder verschwindet. Danach wird eine Liste erstellt, welche aus beiden eingelesenen Werten besteht. Mit dem „csv.writer“-Befehl wird diese Liste als eine neue Zeile in der csv-Datei hinzugefügt. Die „csvspeicher“-Funktion wird über das Betätigen des „Save Entry“-Knopfes ausgeführt.



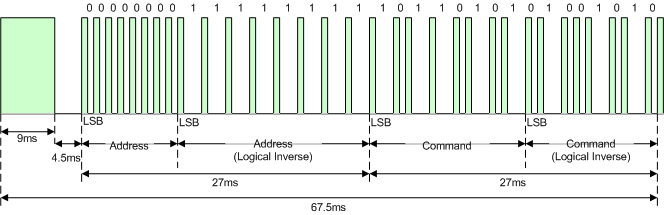
Der Graph wird bei dieser Aufbaustruktur nicht automatisch aktualisiert, weswegen ein weiterer Knopf eingebaut wurde, der dies bewerkstelligen soll. Durch Betätigen des „Refresh Graph“ Knopfes wird die „csvrefresh“ Funktion aufgerufen, welche durch schnelles Schließen und Wiederaufrufen des GUI-Fensters eine Aktualisierung des Graphen durchführt. Um dies bewerkstelligen zu können, wurde ~~die~~ der „os“ Bibliothek hinzugefügt, welche Daten innerhalb des Computers öffnen kann.



**6 Infrarot Datenbank**

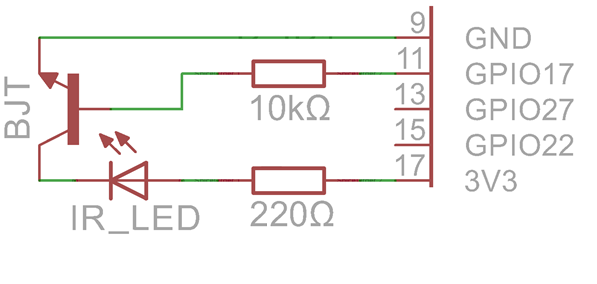
**6.1 Infrarot allgemein**

Da dieses Projekt Geräte mit einer Infrarot-Schnittstelle ansprechen soll, wurde ~~eine~~ nach einer Möglichkeit gesucht, diese Steuermethode in die Schaltung zu integrieren. Da der Raspberry Zero W keine standardmäßig eingebaute Infrarot-Schnittstelle besitzt, wurde das Ganze über eine externe Schaltung betrieben, die sich die sogenannte „LIRC“-Bibliothek zunutze macht. „LIRC“ also „Linux Infrared Remote Control“ ist ein Paket, das mithilfe eines IR-Ermitter Infrarot-Signale durch „pulsing“ emulieren kann.



Da das Infrarot-Übertragungsprotokoll im Microsekunden-Bereich stattfindet, kann es nicht durch die gewöhnliche „timer“-Funktionen ausgeführt werden. Das „pulsing“ des IR-Ermitters wird in diesem Fall direkt über die „Base-Clock“ des Raspberry, der mit 1GHz den Takt vorgibt, geregelt. Das oben abgebildete Übertragungsprotokoll wird dabei aus einem 9ms Aktivierungsbefehl, gefolgt von der 8-Bit-Empfängeradresse, der invertierten Empfängeradresse und danach mit einem 8-Bit-Befehl und am Ende mit dem invertierten Befehl zusammengesetzt. Ein logisches „high“ wird bei Infrarot durch einen 562,5µs Puls, gefolgt von 1,6875ms Pause, zusammengestellt. Ein logisches „low“ wird durch einen 562,5µs Puls gefolgt von 562,5µs Pause, gesendet. Durch LIRC werden diese genau definierten Signale durch simple Befehle im Hauptprogramm ersetzt, welche nur durch Hexadezimal-Codes angegeben werden können. Die LIRC-Bibliothek wandelt diesen simplen HEX-Befehl in ein vollständiges Übertragungsprotokoll um und schickt dieses über einen Infrarot-Ermitter an externe Geräte.

**6.2 Infrarot-Schaltung**



Die Infrarot-Ermitter ist eine handelsübliche IR-LED mit einer Wellenlänge von 940nm, was Industriestandard ist, die in THT ausgeführt ist, da sie am äußeren Gehäuse befestigt ist. Der Takt, mit welcher der Infrarot-Ermitter blinkt, wird durch einen „Bipolar-Junction-Transistor“ in NPN-Ausführung vorgegeben. Der gewählte Transistor ist dabei ein BC847, der in einer SOT-23 Verpackung ausgeführt ist. Der BJT hat eine „transistion frequency“ von , was durch die Formel eine Schaltdauer von 0,01µs ergibt, welche für Takte von 562,5µs vollkommen ausreicht. Vor dem „Base“-Kontakt des Transistors wurde ein Vorwiderstand von 10kΩ eingebaut, der den Strom, durch das Ohm’sche Gesetz, auf 330µA begrenzt. Vor der Infrarot-LED wurde ebenfalls ein Vorwiderstand eingebaut, um die LED zu schützen. Durch den GPIO17-Pin kann nun der Raspberry mithilfe der LIRC-Bibliothek einen Takt ausgeben, der den Transistor über den „Base-Pin“ ein- und ausschalten und so das Infrarot-Übertragungsprotokoll ausführen kann.

**6.3 Datenspeicher**

Da die LIRC-Bibliothek Hexadezimalbefehle braucht, um Geräte damit steuern zu können,

wurde eine Datenbank in Form eines Python-Tuples erstellt, da die Daten so nicht durch einen Programmfehler verändert werden können und einzelne Befehle damit leicht aufzurufen sind. Die gespeicherten Befehle wurden mithilfe eines IR-Empfängers durch einen Arduino aufgezeichnet, indem mit verschiedenen Fernbedingungen die Befehle:

"On/Off", "Enter/Ok", "Channel Up", "Channel Down", "Volume Up", "Volume Down" auf dem „Serial Plotter“ des Arduinos gespeichert wurden. Diese Befehle waren jedoch in dezimal und mussten erst auf hexadezimal umgewandelt werden.

Für den Prototypen wurden nur Befehle von der MKJ42519618 Fernbedienung von „LG“ in das Programm integriert, da es sonst zu komplex werden würde. Es gibt jedoch immer die Möglichkeit, das Programm mit Befehlen von anderen Fernbedienungen zu erweitern, um verschiedene Geräte anzusprechen, die über andere Befehle gesteuert werden.



Die in diesem Bild dargestellten Tuples beinhalten die kompletten Informationen der

Infrarot-Befehle, die durch „Control\_List“ als String, durch „IR\_controls“ als Dezimalzahl

und durch „IR\_controls\_HEX“ als fertige Hexadezimalzahlen, gespeichert.

Durch den Befehl: „command = IR\_controls\_HEX(1)“ wird z.B. der erste Befehl im

Tuple unter der Variable „command“ gespeichert, die dann in das Programm einfließen

und über die LIRC-Befehle zu einem Infrarot-Datensatz umgewandelt werden kann.

Quellen:

Infrarot:

<https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol>

<http://www.lirc.org/>

Bodediagramm RC-Tiefpass:

<http://sim.okawa-denshi.jp/en/CRtool.php>

Logik-Level Konverter

<https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/>

<https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all>

Negative Versorgungsspannung:

<https://www.instructables.com/id/Create-A-Negative-Power-Supply-For-An-Analog-Circu/>