



Hochschule München  
Fakultät für Elektrotechnik und Informationstechnik

---

ENTWICKLUNG EINER  
MIKROCONTROLLERSTEUERUNG FÜR  
HALOGENBRENNER MIT LITHIUM-IONEN  
BATTERIE MANAGEMENT

---

**Bachelorarbeit**

im Studiengang Elektrotechnik und Informationstechnik

vorgelegt von

Dennis S i t n i c

**Bearbeitungsbeginn:** 01.07.2020  
**Abgabetermin:** 02.02.2021  
**Ifd. Nr.:** 1987

Hochschule München  
Fakultät für Elektrotechnik und Informationstechnik

---

ENTWICKLUNG EINER  
MIKROCONTROLLERSTEUERUNG FÜR  
HALOGENBRENNER MIT LITHIUM-IONEN  
BATTERIE MANAGEMENT

DEVELOPMENT OF A  
MICROCONTROLLER-CONTROL FOR  
HALOGENLAMPS WITH LITHIUM-ION BATTERY  
MANAGEMENT

---

**Bachelorarbeit**

im Studiengang Elektrotechnik und Informationstechnik

vorgelegt von

Dennis S i t n i c

**Betreuer:** Prof. Dr. Christian Kißling  
**Bearbeitungsbeginn:** 01.07.2020  
**Abgabetermin:** 02.02.2021  
**Ifd. Nr.:** 1987

## **Eidesstattliche Erklärung**

Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

Datum: \_\_\_\_\_ Unterschrift: \_\_\_\_\_

Ich erkläre mein Einverständnis, dass die von mir erstellte Bachelorarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmustern selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

Datum: \_\_\_\_\_ Unterschrift: \_\_\_\_\_

## **Kurzfassung**

Diese Bachelorarbeit befasst sich mit der Hard- und Softwareentwicklung einer Mikrocontrollersteuerung für Lithium-Ionen Akkus mit Batterie Management für eine Tauchlampe. Die Aufgabe dieser Schaltung ist es eine Halogenlampe zu dimmen, die Langlebigkeit und den optimalen Betrieb der Lithiumionenbatterien zu gewährleisten und Messwerte der Zellen an den Nutzer auszugeben. Die Akkus sind im Zuge des Entladevorgangs vor Tiefenentladung und vor zu hohen Entladeströmen geschützt. Während dem Ladevorgang sind sie vor einem Überladen und zu hohen Ladeströmen geschützt. Die Schaltung gibt Auskunft über die Spannung der Zellen, den Strom der fließt, die Betriebstemperatur und die verbleibende elektrische Ladung. Die Halogenlampe wird über ein PWM Signal gedimmt, das über einen variablen Tastgrad einstellbar ist. Zur Kommunikation mit programmierbaren Bausteinen sind USB, USART, I2C, SWD und IO Schnittstellen vorhanden. Die Schaltung hat einen aktiven und einen Sleep-Modus. Die Gesamtschaltung wurde schrittweise entwickelt. Im ersten Schritt wurde ein Prototyp für das BMS entwickelt und getestet. Anschließend wurde der Steuerteil programmiert und eine Machbarkeitsstudie vorgenommen. Darauf aufbauend wurde die Gesamtschaltung entwickelt. Die Tauchlampensteuerung ist vielseitig für eine Weiterentwicklung ausgelegt und kann mit einem Bedienelement erweitert werden.

## **Abstract**

This bachelors thesis is about the hard- and softwaredevelopement of a microcontroller control for lithium-ion Accumulators with battery management for diving lights. The task of this circuit is to dim a halogen lamp, to ensure the longevity and optimal operation of the lithium ion batteries and to output measured values from the cells to the user. During the discharge process, the batteries are protected from undervoltage and from excessive discharge currents. During the charging process, they are protected against overvoltage and excessive charging currents. The circuit provides information about the voltage of the cells, the current flowing, the temperature and the remaining electrical charge. The halogen lamp is dimmed using a PWM signal that can be set using a variable duty cycle. USB, USART, I2C, SWD and IO interfaces are available for communication with the programmable components. The circuit has an active mode and a sleep mode. The final circuit was developed step by step. In the first step, a prototype for the BMS was developed and tested. Then the control part was programmed and a feasibility study was carried out. Based on the achieved informations the final ciruit was developed. The diving light control is designed with a braod set of options for further development and can be expanded with a control element.

# Inhaltsverzeichnis

Eidesstattliche Erklärung . . . . .	3
Kurzfassung . . . . .	4
<b>1. Einleitung</b>	<b>6</b>
<b>2. Grundlagen</b>	<b>7</b>
2.1. Lithiumionenbatterie . . . . .	7
2.2. Batterie Management System . . . . .	7
2.3. DS2777 . . . . .	8
2.4. STM32L031k6 . . . . .	8
<b>3. Entwicklung des BMS-Teils</b>	<b>9</b>
3.1. Konzept . . . . .	9
3.2. Hardware . . . . .	9
3.3. Software . . . . .	13
<b>4. Entwicklung des Steuerteils</b>	<b>17</b>
<b>5. Machbarkeitsstudie</b>	<b>20</b>
<b>6. Entwicklung der Gesamtschaltung</b>	<b>23</b>
6.1. BMS-Teil . . . . .	25
6.2. PWM Teil . . . . .	28
6.3. Spannungsregler . . . . .	30
6.4. USB zu UART . . . . .	31
6.5. Mikrocontroller . . . . .	32
6.6. Schnittstellen . . . . .	33
<b>7. Ergebnisdiskussion</b>	<b>35</b>
<b>8. Ausblick</b>	<b>37</b>
<b>A. Anhang</b>	<b>42</b>

# 1. Einleitung

Schon 4500 v.Chr. haben die Menschen den Meeresboden erkundet. So weit gehen die ältesten Archäologischen Funde zurück [1, S. 7]. Damals auf der Suche nach Nahrung und Perlen ohne komplizierte Ausrüstung ist Tauchen heute zu einer Freizeitbeschäftigung geworden mit modernster Technik. Ein unabdingbares Ausrüstungsstück ist die Taucherlampe. Die Notwendigkeit entsteht daher, weil das Wasser das Licht absorbiert. Lichtstrahlen mit einer längeren Wellenlänge wie rotes Licht dringen weniger Tief in das Wasser ein als blaues und violettes Licht, das eine kürzere Wellenlänge besitzt [2]. Nach wenigen Metern ist der Großteil des roten Lichtes verschwunden. Um dem entgegenzuwirken und das gesamte Farbspektrum unter Wasser wahrzunehmen greifen einige Taucher zu Halogenlampen, die im Vergleich zu herkömmlichen LEDs eine hohe spezifische Ausstrahlung von langwelligem Licht besitzen [3]. Allerdings verbrauchen Halogenlampen mehr Strom und sind weniger effizient [4]. Die Anforderung an eine Tauchlampe ist, dass sie möglichst für den gesamten Tauchvorgang Licht spendet. Für diese Aufgabe eignen sich Lithiumionenbatterien (LIB), da diese in ihrer spezifische Energie und Energiedichte andere Arten von Akkumulatoren übertrifft [5, S. 294]. Allerdings müssen LIB vor Überladung, Tiefentladung und zu hohem Stromfluss geschützt werden. Die dafür zuständige Elektronik nennt man Batteriemanagementsystem (BMS) [6, S. 35]. In einigen Taucherlampen sind nur bescheidene BMSs zu finden oder die Schaltungselektronik führt neben der Selbstentladung zu zusätzlichem Entladen bei Nichtbenutzung und die Akkumulatoren werden bei Tiefentladung gesperrt und können nicht mehr aufgeladen werden [7, S. 18]. Das Ziel dieser Arbeit ist es eine Tauchlampensteuerung mit einem BMS zu entwickeln, das die LIB vor Tiefentladung, Überladung und zu hohen Strömen schützt, den aktuellen Strom und die Zellspannung misst und eine Ladezustandsschätzung vornimmt. Die Tauchlampensteuerung soll eine Halogenlampe betreiben und dimmen. Der Gesamtstromverbrauch soll gering gehalten werden, um eine lange Laufzeit zu gewährleisten. Dabei soll die Platine einen Durchmesser von 46 mm besitzen, um in ein zylinderförmiges Gehäuse zu passen. Für die Realisierung werden zwei integrierte Schaltungen verwendet. Ein Mikrocontroller von ST für die Steuerung und Kommunikation und ein BMS-Chip von Maxim. Die Entwicklung der Tauchlampensteuerung lässt sich in drei Teile aufteilen. Im ersten Teil wird das BMS genauer untersucht. Dafür ist die Entwicklung eines Breakoutboards notwendig. Im zweiten Teil wird Steuerung und Kommunikation über den Mikrocontroller entwickelt und zusammen mit dem Breakoutboard eine Machbarkeitsstudie unternommen. Im dritten Teil wird aufbauend auf den gewonnenen Erkenntnissen eine Gesamtschaltung mit dem vorgegebenen Formfaktor entwickelt. Im Anschluss werden die Ergebnisse diskutiert und Optimierungsmöglichkeiten dargelegt. Diese Arbeit beschränkt sich lediglich auf den Steuerteil einer Tauchlampe. Ein Bedienelement oder Ladegerät kann aufbauend auf dieser Arbeit entwickelt werden.

## **2. Grundlagen**

Dieses Kapitel behandelt das Hintergrundwissen und die technischen Grundlagen, die notwendig sind, um die Entwicklung der Tauchlampensteuerung nachvollziehen zu können. Dazu gehören die grundlegenden Eigenschaften von LIB, die Erklärung von BMS und das Vorstellen der verwendeten programmierbaren Bausteine.

### **2.1. Lithiumionenbatterie**

Lithiumionenbatterien (LIB) sind „Sekundärbatterie[n] mit Elektrolyt aus einem organischen Lösungsmittel und positiven und negativen Elektroden aus Einlagerungsverbindungen, die Lithium enthalten“ [8]. Es wandern beim Entladen und Laden Lithium-Ionen zwischen den Elektroden. Für die Energieversorgung von Notebooks und Handys werden heute überwiegend von LIB benutzt [5, S. 1][9, S. 14]. Ein paar Eigenschaften kommen bei der Entwicklung dieser Arbeit zugute. Neben der hohen Energiedichte besitzen LIB eine sehr geringe Selbstentladungsrate im Vergleich zu anderen Batterien und sind deshalb gut für Anwendungen geeignet in denen Batterien für eine längere Zeit nicht geladen werden. Außerdem sind sie nicht anfällig für memory-effekte und können deshalb vom Nutzer beliebig geladen und entladen werden [6, S. 25]. Die Nominalspannung einer typischen LI-Zelle beträgt 3,6 V mit einer Entladeschlussspannung von 2,5 V und einer Ladeschlussspannung von 4,2 V. Ein Überladen der Zelle kann zu thermischem Durchgehen, Zellschwellung und Selbstentzündung führen. Ein zu tiefes Entladen führt zu Kapazitätsverlust und erhöhter Selbstentladung. [9, S. 177][6, S. 35f]

### **2.2. Batterie Management System**

Ein BMS ist eine elektronische Schaltung, die eine sichere Operation von Akkumulatoren gewährleistet und ihre Lebenszeit erhöht. Dabei besteht ein BMS meist aus einem Software und Hardware Teil. Der Hardwareteil ist im einfachsten Fall zwischen Zelle und Last angebunden und kann die Zelle von der Last trennen und die Zellspannung, die Temperatur und den Strom messen. Die Software steuert und überwacht die BMS aktivitäten. Die Aufgabe ist es unsichere Operationsbedingungen zu erkennen und entsprechend zu reagieren. [10, S. 1f][9, S. 182]

## 2.3. DS2777

Der DS2777 (Abb. 2.1) von Maxim ist ein Batteriemanagement-IC, der für zwei Lithiumionen- oder Lithium-Poly-Zellen ausgelegt ist.

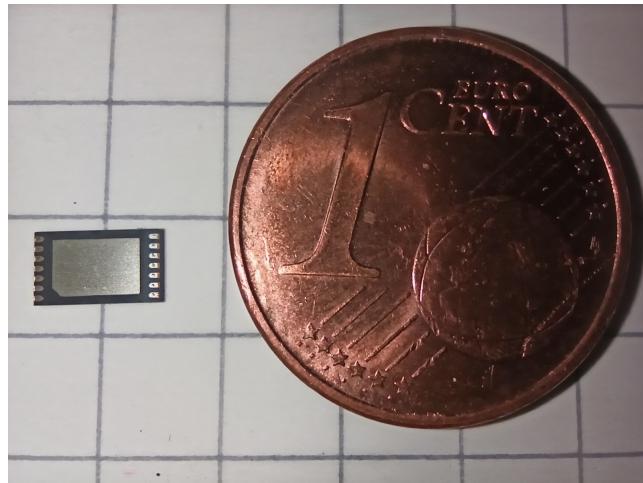


Abbildung 2.1.: DS2777 neben einer 1 Cent Münze

Der Chip hat eine Größe von  $3,1 \text{ mm} \times 5,1 \text{ mm}$ . Der DS2777 misst Zellspannungen, die Temperatur und den Strom und berechnet anhand dieser Messwerte zusammen mit den Zelleigenschaften der verwendeten Batterien und den Anwendungsparametern mit einem internen Algorithmus eine Ladezustandsschätzung. Die Zelleigenschaften und Anwendungsparameter müssen programmiert werden. Für die Kommunikation steht eine I2C Schnittstelle zur Verfügung. Um als BMS zu fungieren benötigt der Chip Peripheriebausteine. Es gibt einen aktiven und einen Sleep Mode. [11]

## 2.4. STM32L031k6

Der STM32L031k6 ist ein 32 bit ultra-low-power MC von ST. Er ist für einen geringen Stromverbrauch ausgelegt und hat verschiedene sleep modi, einen ADC, USART, SPI, I2C, eine SWD Schnittstelle und mehrere Timer. Für den Code stehen 32 kB Flash Speicher und 1 KB EEPROM zur Verfügung. [12] Im Kontext dieser Arbeit ist der MC für die Kommunikation mit dem DS2777 und dem Nutzer und der Ansteuerung des Dimmungssystems zuständig.

# 3. Entwicklung des BMS-Teils

Wegen der geringen Ausmaße des DS2777 ICs und der benötigten Peripheriebausteine ist es notwendig ein Breakoutboard zu entwickeln. Das Breakoutboard wird im Testaufbau für die Gesamtschaltung als BMS-Teil eingesetzt. Das Ziel ist es eine Platine zu entwickeln, die als Modul für den Testaufbau der Gesamtschaltung verwendet wird.

## 3.1. Konzept

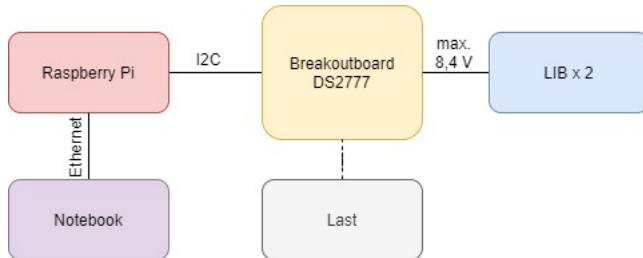


Abbildung 3.1.: Schematische Darstellung des Kontextes in dem das Breakoutboard eingebettet wird

Um die Hardware-Anforderungen für das Breakoutboard herauszuarbeiten muss der Kontext betrachten werden in welchem das Breakoutboard verwendet wird. Sämtliche Pins des DS2777 sollen zugänglich werden. Für die ersten Tests wird ein Raspberry Pi über I2C mit dem DS2777 kommunizieren. Später wird an dessen Stelle das Nucleo Board treten. Die Anbindung zweier LIB-Zellen soll vorhanden sein, wobei der Strom in beide Richtungen fließen können muss. Ausgangsseitig ist das Anlegen einer Last zu ermöglichen. Dabei sollen die beteiligten Blöcke modular ausgelegt werden.

## 3.2. Hardware

Der Großteil des Schaltbildes (siehe Abb. 3.2) wird im Datenblatt vorgegeben [11]. Um die Modularität zwischen dem Raspberry Pi, dem Nucleo Board und dem Breakoutboard zu realisieren werden Jumwires als Verbindungen zwischen den Modulen verwendet.

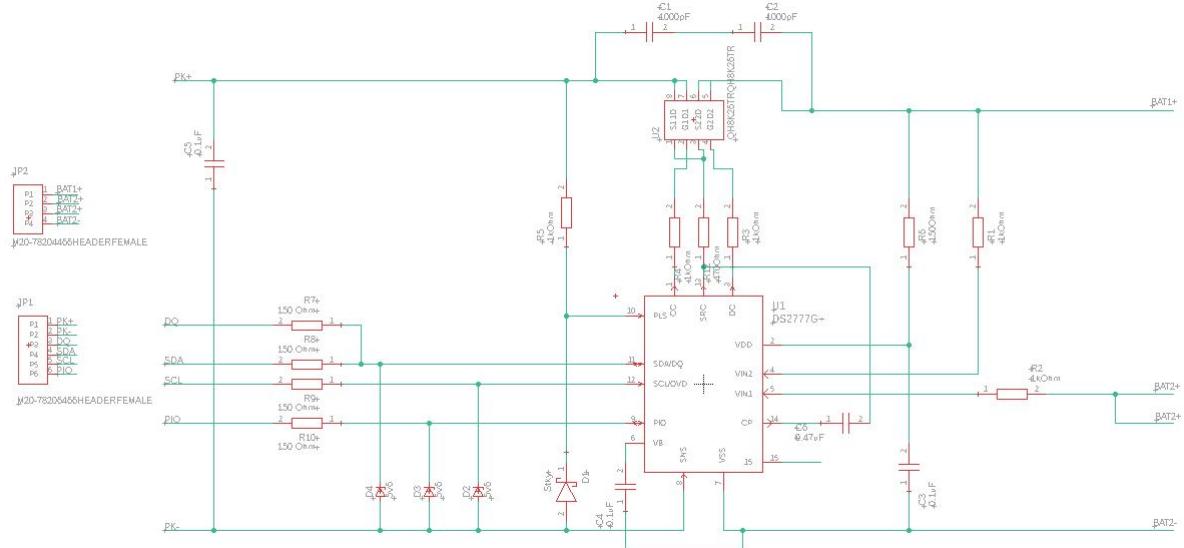


Abbildung 3.2.: Schaltbild des Breakoutboards

Es werde Steckverbinder benutzt, die den Steckverbindungen des Raspberry Pis ähneln und die passenden Abstände und Druchmesser für die Jumpwires besitzen. Weil die Leiterplatte händisch bestückt wird, werden SMD Widerstände und Kondensatoren der Baugröße 0805 verwendet, die groß genug sind, um sie mühelos per Hand zu platzieren. Die seriellen Widerstände, die zum I2C Bus und dem IO-Pin führen begrenzen den Strom, aber heben dafür den Nullpegel an [13, S. 85]. Ein I2C Bus funktioniert mit Open-Drain Verbindungen [13, S. 8]. Deshalb wird bei dieser Schaltung davon ausgegangen, dass die Pull-Up Widerstände bei dem externen Modul, das für die I2C Kommunikation angeschlossen wird, vorhanden sind. Die Z-Dioden stellen sicher, dass die Pins, die zum Verbindungsstecker führen vor zu hohen Spannungen geschützt sind. Eine Durchbruchspannung von 5,1 V wurde gewählt, da die Pins maximal 6 V vertragen und alternativ zum Raspberry Pi ein Arduino, der mit 5 V arbeitet, verwendet werden kann. Die verwendeten MOSFETS fungieren als Schalter. Einer ist für das Laden zuständig und der andere für das Entladen der LIB. Durch die MOSFETS wird ein Strom von mehreren Apere fließen. Um Verluste und Erwärmung gering zu halten ist es notwendig einen MOSFET mit geringem  $R_{DS(on)}$  zu verwenden, der von dem DS2777 angesteuert werden kann. Es wurde der QH8K26 (3.1) ausgewählt.

	Anforderung	QH8K26
$U_{DS}$	$> 8,4 \text{ V}$	40 V
$U_{GS}$	$> 4,6 \text{ V}$	12 V
$U_{GS(th)}$	$< 4,6 \text{ V}$	1 V bis 2,5 V
$R_{DSon}$	so gering wie möglich	$35 \text{ m}\Omega$
$I_D$	$\geq 4 \text{ A}$	7 A; 18 A Puls

Tabelle 3.1.: Eine Auflistung der Anforderungen an die MOSFETs (Mitte) und die Parameter [14] der gewählten MOSFETs (rechte Spalte)

Die 8,4 V ergeben sich durch die Reihenschaltung zweier LIB bei maximal möglicher Ladespannung. Bei der Auswahl des MOSFETs wurde darauf geachtet, dass die erzeugte Gate Source Spannung des Chips hoch genug ist um den MOSFET komplett durchzuschalten. Um Platz zu sparen besitzt der IC zwei Kanäle mit jeweils einem MOSFET.

## Layout

Es bestehen keine Größeneinschränkungen für die Abmessungen der Platine. Dadurch gibt es ausreichend Platz für das Platzieren der Bauteile und dem Routen der Leitungsverbindungen, wodurch 2 Layer ausreichend sind. Trotzdem sollen sich die Maße in einem handlichen Bereich erstrecken.

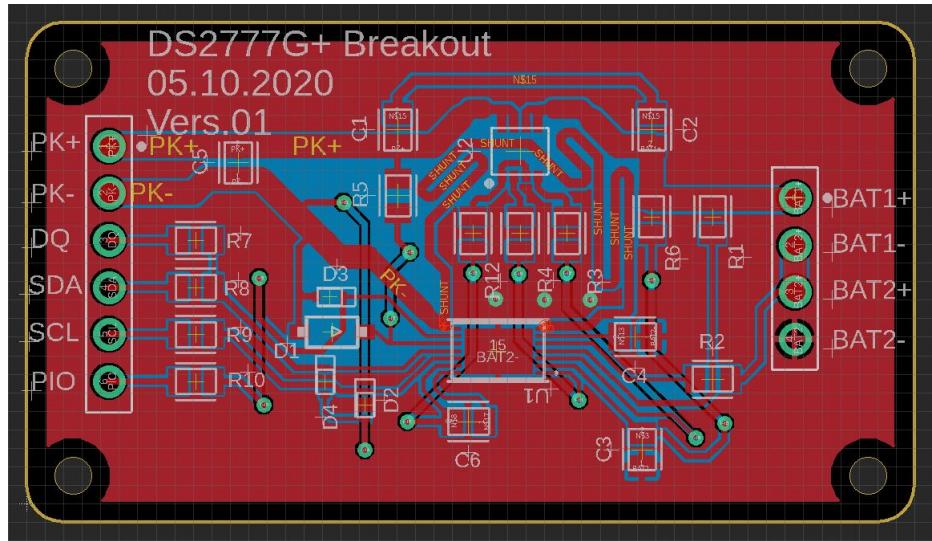


Abbildung 3.3.: Layout Breakoutboard

Die Steckverbinder werden so platziert, damit sich auf der einen Seite der Anschluss für die Zellen befindet und auf der anderen Seite die Ausgangsspannung und Kommunikations-schnittstellen zur Verfügung stehen. Zwischen Pin 7 und Pin 8 (siehe 3.2) wird der Strom

über die Spannung eines Shunt-Widerstandes gemessen. In der Schaltung ist keine Verbindung angegeben, damit die Netze voneinander getrennt werden. Im Layout wird der Shunt Widerstand als Leitungsstück realisiert. Der Strommesswiderstand soll  $25 \text{ m}\Omega$  besitzen. Die Leitung für den Strommesswiderstand muss schmal genug sein, um an die kleinen Pins angefügt zu werden und breit genug, um einen maximalen Dauerstrom von  $4 \text{ A}$  bei geringer Erwärmung zu führen. Der Dauerstrom von  $4 \text{ A}$  ergibt sich aufgrund der Einschränkungen des DS2777. Der höchstmögliche programmierbare Wert für den maximalen Dauerstrom ist  $4 \text{ A}$  [11, S. 28, Table 6]. Nach  $10 \text{ ms}$  und einem Stromfluss von  $4 \text{ A}$  sperrt der EntlademOSFET. Bei einer Kupferdicke von  $70 \mu\text{m}$  eignet sich eine Leiterbreite von  $0,5 \text{ mm}$ . Die Temperaturerhöhung beträgt bei diesen Leitungsausmaßen und dem maximalen Dauerstrom von  $4 \text{ A}$  nach Tabellenwerten:  $45^\circ\text{C}$  [15, S. 78]. Der maximale Impulsstrom, der durch den DS2777 vorgegeben wird beträgt  $12 \text{ A}$ . Nach  $t_{SCD} = 120 \mu\text{s}$  bei  $12 \text{ A}$  Entladestrom sperrt der EntlademOSFET. Die Temperaturerhöhung der Leitung beträgt bei den genannten Abmessungen und einer Impulsdauer von  $1 \text{ ms}$  nach Formel (3.1)  $0,6^\circ\text{C}$ . [15, S. 77].

$$\Delta T[K] = 5 \cdot 10^9 \frac{I^2[A] \cdot t[s]}{h^2[\mu\text{m}] \cdot b^2[\mu\text{m}]} \quad (3.1)$$

I=Stromstärke, t=Impulsdauer, h=Leiterbahnhöhe, b=Leiterbahnbreite

Die Impulsdauer von  $1 \text{ ms}$  ergibt sich, wenn man von einem PWM Signal mit einer Frequenz von  $500 \text{ Hz}$  und einem Tastgrad von  $50 \%$  ausgeht.

$$R = \rho \frac{l}{t \cdot w} \quad (3.2)$$

$\rho$  = Spezifischer Widerstand,  $l$  = Länge,  $t$  = Höhe,  $w$  = Breite

Die Länge der Leitung für den Strommesswiderstand ergibt sich durch die bekannte Formel für den Leitungswiderstand (3.2) und beträgt  $51,17 \text{ mm}$ .

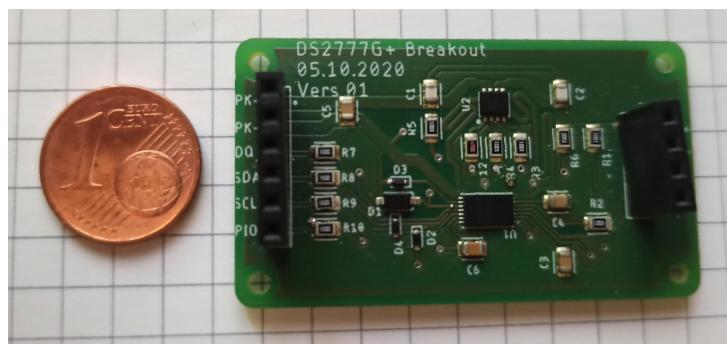


Abbildung 3.4.: Das bestückte Breakoutboard neben einer 1 Cent Münze

Die Gerber Daten wurden erstellt, dem Hersteller geschickt und die Platine zusammen mit einer Lötschablone bestellt. Die Lötpaste wurde mit Hilfe der Lötschablone aufgetragen, die Bauteile wurden per Hand bestückt und in einem Reflow Ofen verlötet. (Abb. 3.4).

### 3.3. Software

Für das initiale Programmieren des DS2777 wird der Aufbau aus Abbildung (3.1) verwendet. Damit das Breakoutboard seine Funktion erfüllt müssen die Eigenschaften der verwendeten LIB und Anwendungsparameter (Abb. 3.5) programmiert werden. Die Verbindung zwischen dem Notebook und dem Raspberry Pi wird über ein Ethernet Kabel mit PuTTy hergestellt. Für die Verbindung zwischen dem Raspberry Pi und dem Breakoutboard werden drei Jumewire verwendet, die SDA, SCL und GND miteinander verbinden. Für das Ausführen von Schreibe- und Lesevorgängen wird auf dem Raspberry Pi die Tool-Sammlung i2c-tools verwendet.

ADDRESS (HEX)	DESCRIPTION	ADDRESS (HEX)	DESCRIPTION
60h	Control Register	71h	AE Segment 3 Slope Register
61h	Accumulation Bias Register (AB)	72h	AE Segment 2 Slope Register
62h	Aging Capacity Register MSB (AC)	73h	AE Segment 1 Slope Register
63h	Aging Capacity Register LSB (AC)	74h	SE Segment 4 Slope Register
64h	Charge Voltage Register (VCHG)	75h	SE Segment 3 Slope Register
65h	Minimum Charge Current Register (IMIN)	76h	SE Segment 2 Slope Register
66h	Active-Empty Voltage Register (VAE)	77h	SE Segment 1 Slope Register
67h	Active-Empty Current Register (IAE)	78h	Sense-Resistor Gain Register MSB (RSGAIN)
68h	Active-Empty 40 Register	79h	Sense-Resistor Gain Register LSB (RSGAIN)
69h	Sense Resistor Prime Register (RSNSP)	7Ah	Sense-Resistor Temperature Coefficient Register (RSTC)
6Ah	Full 40 MSB Register		
6Bh	Full 40 LSB Register	7Bh	Current Offset Bias Register (COB)
6Ch	Full Segment 4 Slope Register	7Ch	TBP34 Register
6Dh	Full Segment 3 Slope Register	7Dh	TBP23 Register
6Eh	Full Segment 2 Slope Register	7Eh	TBP12 Register
6Fh	Full Segment 1 Slope Register	7Fh	Protector Threshold Register
70h	AE Segment 4 Slope Register	80h	2-Wire Slave Address Register

Abbildung 3.5.: Adressen und Beschreibung der Zell- und Anwendungsparameter im EEPROM [11, Tab. 8]

Zu den Zell- und Anwendungsparametern gehört beispielsweise die Ladeschlussspannung, die Entladeschlussspannung, die Größe des Strommesswiderstandes, der Kurzschlusstrom usw. Im Datenblatt wird das Verfahren für die Ermittlung der nötigen Registerwerte genauer erklärt [11, S. 13ff]. Für die initiale Programmierung werden folgende Werte benutzt:

```

1 #!/bin/bash
2 i2cset -y 1 0x59 0x60 0xee
3 i2cset -y 1 0x59 0x61 0x0
4 i2cset -y 1 0x59 0x62 0x1310 w
5 i2cset -y 1 0x59 0x64 0xd4
6 i2cset -y 1 0x59 0x65 0x28
7 i2cset -y 1 0x59 0x66 0x9A
8 i2cset -y 1 0x59 0x67 0x34
9 i2cset -y 1 0x59 0x69 0x28
10 i2cset -y 1 0x59 0x6a 0x12f8 w
11 i2cset -y 1 0x59 0x6c 0x0d

```

```

12 i2cset -y 1 0x59 0x6d 0x12
13 i2cset -y 1 0x59 0x6e 0x33
14 i2cset -y 1 0x59 0x6f 0x3b
15 i2cset -y 1 0x59 0x70 0x5
16 i2cset -y 1 0x59 0x71 0xa
17 i2cset -y 1 0x59 0x72 0x11
18 i2cset -y 1 0x59 0x73 0x27
19 i2cset -y 1 0x59 0x74 0x2
20 i2cset -y 1 0x59 0x75 0x4
21 i2cset -y 1 0x59 0x76 0x6
22 i2cset -y 1 0x59 0x77 0x16
23 i2cset -y 1 0x59 0x78 0x74
24 i2cset -y 1 0x59 0x79 0x74
25 i2cset -y 1 0x59 0x7a 0x0
26 i2cset -y 1 0x59 0x7b 0x0
27 i2cset -y 1 0x59 0x7c 0x1e
28 i2cset -y 1 0x59 0x7d 0x14
29 i2cset -y 1 0x59 0x7e 0x0a
30 i2cset -y 1 0x59 0x7f 0x58
31 i2cset -y 1 0x59 0xfe 0x44

```

Die programmierten Werte werden zunächst auf dem Shadow RAM gespeichert und sind bei einem Neustart nicht mehr vorhanden. Der Befehl `i2cset -y 1 0x59 0xfe 0x44` wird verwendet um die Daten vom Shadow Ram in den EEPROM zu schreiben. Mit dem Befehl `i2cdump -y -r 0x60-0x80 1 0x59` lassen sich alle programmierten Werte im Adressbereich 0x60 bis 0x80 auslesen.

```

1      0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
2 60: ee 00 10 13 d4 28 9a 34 00 28 f8 12 0d 12 33 3b
3 70: 05 0a 11 27 02 04 06 16 74 74 00 00 1e 14 0a 58
4 80: b2

```

## Messungen des DS2777

Der DS2777 misst alle 440 ms die Spannung der Zellen und speichert diese in die Register mit den Adressen 0x0C und 0x1C. Das gilt auch für den Strom und die Temperatur, mit anderen Zeitabständen und Registeradressen. Auf Grundlage der programmierten Werte aus dem EEPROM und den „Echtzeitmessungen“ berechnet der Chip eine Ladezustands schätzung. Um die Werte aus den Registern zu interpretieren muss man die relevanten Bits von den irrelevanten trennen, in dezimaler Form umwandeln und mit der angegebenen Unit multiplizieren. Für das Auslesen der Register wird der Befehl `i2cget -y 1 slaveadresse registeradresse bit-datentyp` bemüht.

### Auslesen der Echtzeitregister

```

1 #!/bin/bash
2 echo Spannung_1 0c:
3 i2cget -y 1 0x59 0x0c w
4 echo Spannung_2 1c:
5 i2cget -y 1 0x59 0x1c w
6 echo Strom 0x0e:
7 i2cget -y 1 0x59 0x0e w
8 echo Temp 0x0a:
9 i2cget -y 1 0x59 0x0a w
10 echo raac 0x02:
11 i2cget -y 1 0x59 0x02 w
12 echo rsac 0x04:
13 i2cget -y 1 0x59 0x04 w
14 echo rarc 0x06:
15 i2cget -y 1 0x59 0x06
16 echo rsrc 0x07:
17 i2cget -y 1 0x59 0x07

```

### Output

```

1
2
3
4 Spannung_1 0c:
5 0x604f
6 Spannung_2 1c:
7 0x404f
8 Strom 0e:
9 0xe017
10 raac 02:
11 0x0000
12 rsac 04:
13 0x0000
14 rarc 06:
15 0x00
16 rsrc 07:
17 0x00

```

Für die Messungen werden zwei Batterieeinheiten mit einer Spannung von jeweils 3,1 V angeschlossen. Als Last wird ein  $5\text{ k}\Omega$  Widerstand verwendet wodurch ein Laststrom von ca. 1,24 mA zu sehen ist. In Tabelle (3.2) ist eine Gegenüberstellung zwischen den Messwerten und den Sollwerten zu sehen.

Register	Inhalt	Ergebnis	Sollwert
Zelle 1	0x604f	3,76 V	3,1 V
Zelle 2	0x404f	2,51 V	3,1 V
Strom	0xe1ff	-0,48A	1,2 mA
Temp	0xe017	32°C	20°C
RAAC	0x0000	0 mAh	> 0 mAh
RSAC	0x0000	0 mAh	> 0 mAh
RARC	0x00	0 mAh	> 0 mAh
RSRC	0x00	0 mAh	> 0 mAh

Tabelle 3.2.: Messwerte des DS2777 bei einem Lastwiderstand von  $5\text{ k}\Omega$  in Gegenüberstellung mit den Sollwerten

Die Register können ausgelesen werden, aber die Messwerte unterscheiden sich stark von den Sollwerten. Die Messungen wurden bei gleichbleibenden Sollwerten mehrmals hintereinander mit kleinen und großen Zeitabständen durchgeführt und die Messwerte haben sich dabei ohne erkennbares Muster geändert und wurden teilweise negativ. Im Kapitel *Ergebnisdiskussion* wird näher darauf eingegangen.

## Sleepmodus

Wenn der DS2777 in den Sleepmodus eintritt, dann sperren beide MOSFETs für die Lade- und Entladekontrolle. Es gibt verschiedene Konfigurationsmöglichkeiten für Events, die das Eintreten und das Aufwachen auslösen [11, Abb. 2]. In diesem Kontext wurde als Signal für das Eintreten in den Sleepmodus ein Low-Pegel auf den SCL und SDA Leitungen ausgewählt und für das Aufwachen ein Low-Pegel an dem IO-Pin des DS2777 oder das Erkennen eines Ladestroms. So kann der MC beide I2C Leitungen auf Low ziehen und den DS2777 in den Sleep versetzen, wobei die komplette Tauchlampensteuerung bis auf den DS2777 von der Stromversorgung getrennt wird. Der IO Pin führt in der Gesamtschaltung zu einem Button und zu einem Pad (an dem man eine Leitung befestigen kann oder einen  $0\ \Omega$  Widerstand, der zum MC führt). So kann das Low-Signal von einem externen Bedienelement oder einem Button auf der Platine betätigt werden (vgl. Abb. 6.4)

## 4. Entwicklung des Steuerteils

Für die Entwicklung des Steuerteils wird ein Nucleo Board mit dem STM32L031K6 verwendet. Der MC wird in C programmiert und es wird dabei auf eine HAL-Bibliothek (Hardware Abstraction Layer) zurückgegriffen, die auf den verwendeten MC zugeschnitten ist [16]. Als Entwicklungsumgebung dient die STM32CubeIDE. Das Programm ist ziemlich simpel gehalten und besitzt die nötigsten Befehle, um die benötigten Funktionen zu erfüllen. Zu den Funktionen gehört das Erzeugen des PWM Signals mit einem einstellbaren Tastgrad. Das Auslesen der Messwerte, die der DS2777 misst und das Ausgeben der gemessenen Werte an den Nutzer.

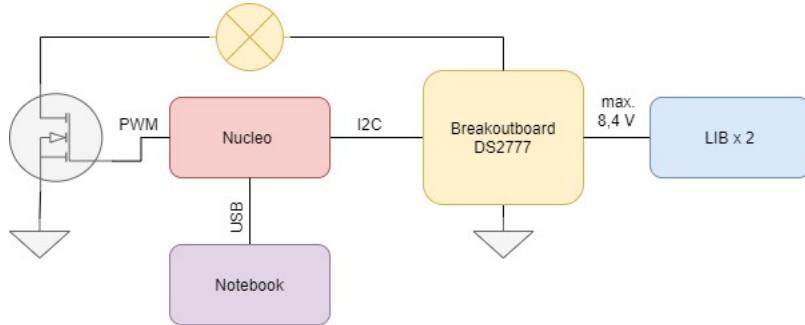


Abbildung 4.1.: Schematische Darstellung des Steuerteil Aufbaus

Der Aufbau des Steuerteils (Abb. 4.1) entspricht im Konzept der Gesamtschaltung. Die verwendete Entwicklungsumgebung ermöglicht es Initialisierungen über eine GUI einzustellen und automatisch Code zu generieren. Es werden die wichtigsten Ausschnitte vorgestellt. Das gesamte Programm ist im Anhang zu finden.

### Clock

Als Taktquelle stehen verschiedene Möglichkeiten zur Verfügung. Es wird der interne Clock MSI (Multi-speed internal RC oscillator) mit der Range 4 verwendet. Das heißt, dass das Clock Signal eine Frequenz von 1,05 MHz besitzt. Der Grund für einen internen Clock gegenüber einem externen Clock ist, dass zusätzliche Bauteile vermieden werden und der interne Clock für die Anwendungen ausreicht. Es gibt einen zweiten internen Clock namens HSI16 (High speed internal 16 MHz RC oscillator). Das Argument für den MSI ist, dass mit dem HSI16 ein Stromverbrauch von  $100 \mu\text{A}$  und mit dem MSI (in Range 4)  $4,5 \mu\text{A}$  erzielt werden. [12, S. 72ff]

## PWM-Signal

Eine Funktion der Tauchlampensteuerung ist das Dimmen einer Halogenlampe. Die Halogenlampe wird mit Hilfe eines MOSFETs, der als Schalter agiert und über ein PWM Signal angesteuert wird, gedimmt. Das Dimmen wird über einen einstellbaren Tastgrad des PWM-Signals ermöglicht. Für die Realisierung wird der 16-Bit Timer Tim22 verwendet. Als PWM Frequenz wird 500 Hz festgelegt. Der Grund dafür ist, dass bei sehr niedrigen Frequenzen Flackern mit dem Auge sichtbar wird und bei niedrigen Frequenzen Flackern mit der Kamera. Eine zu hohe Frequenz führt wegen der Gatedladung des MOSFETs zu höheren Verlusten. Bei einer höheren Frequenz wird öfter umgeladen und der MOSFET ist öfter im linearen Bereich pro Zeiteinheit, während die Umladezeit genauso lange wie bei niedrigeren Frequenzen andauert. Der Timer wird als Up-Counter konfiguriert. Um 500 Hz zu generieren wird bei einem Systemclock von 1,05 MHz das ARR (auto reload register) auf 2100 gesetzt, denn nach 2 ms hat der Zähler bis 2100 gezählt. Ein Prescaler ist nicht nötig. Das CCR (capture compare register) wird auf 1050 gesetzt für einen Tastgrad von 50 %, der nach der Initialisierung veränderbar sein soll. Die komplette Initialisierung ist im Anhang zu finden.

```

1 htim22.Instance = TIM22;
2 htim22.Init.Prescaler = 0;
3 htim22.Init.CounterMode = TIM_COUNTERMODE_UP;
4 htim22.Init.Period = 2100;
5 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
6 sConfigOC.Pulse = 1050;
```

Das PWM-Signal wird mit dem Befehl `HAL_TIM_PWM_Start(&htim22, TIM_CHANNEL_1)`; gestartet. Der Wert des CCR Registers lässt sich mit dem Befehl `TIM22->CCR1 = wert;` zuweisen. Für das Einstellen des Tastgrades wird ein Potentiometer eingelesen. Dafür wird ein der 12-bit ADC verwendet. Nach der Initialisierung wird der ADC mit einem Befehl gestartet, ein Wert gemessen und in eine float variable gespeichert. Die float Variable wird durch 4095 geteilt und mit 2000 multipliziert, sodass bei voller Spannung 2000 und bei der niedrigsten Spannung 0 in der variablen stehen. Bis zu ARR fehlen bewusst 100 Takte, damit der MOSFET beim höchsten Wert nicht permanent durchgeschaltet wird.

```

1 float raw;
2 HAL_ADC_Start(&hadc);
3 HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
4 raw = HAL_ADC_GetValue(&hadc);
5 raw = (raw/4095.0);
6 raw = raw*2000;
```

Anschließend wird der Wert, der jetzt über den Poti einstellbar in das CCR geschrieben.

```
1 TIM22->CCR1 = raw;
```

## Messwerte lesen

Die Messwerte, die der DS277 in seinen Registern speichert, werden über I2C von dem MC ausgelesen. Dafür operiert der MC als Master und der DS2777 als Slave. Die 7-bit Slave Adresse muss um ein Bit nach links verschoben werden um von der Funktion richtig verstanden zu werden und eine 1 muss an die letzte Stelle hinzugefügt werden, um einen Lesevorgang zu signalisieren.

```
1 uint8_t storage01;
2 const uint8_t ds2777_adresse = 0x59;
3 const uint8_t register_adresse = 0x80;
4 HAL_I2C_Mem_Read(&hi2c1, (ds2777_adresse<<1) | 0x01, register_adresse, 1,
&storage01, sizeof(storage01), HAL_MAX_DELAY);
```

Als Beispiel wurde das Register mit der Slave Adresse verwendet, das die Größe von einem Byte besitzt. Dieser Vorgang wird mit allen zu lesenden Registern wiederholt. Anschließend können durch Bitverschieben und Extrahieren der relevanten Bits und dem Multiplizieren mit der Unit die Werte gedeutet werden.

## Messwerte ausgeben

Eine einfache Methode für das Ausgeben der Messwerte ist durch die Verwendung der UART Schnittstelle. Dafür wird der Messwert in einen string umgewandelt.

```
1 char msg[19];
2 sprintf(msg, "Slave Adresse: %X\r\n", storage01);
3 HAL_UART_Transmit(&huart2, msg, strlen(msg), HAL_MAX_DELAY);
```

Wenn man die Messwerte über I2C ausgeben möchte gibt es ein paar Dinge zu beachten. Der Raspberry Pi kann nur als I2C Master eingesetzt werden. Dafür muss der MC als Slave operieren. Wegen dem Auslesen der Messwerte fungiert der MC bereits als Master. Es steht ein GPIO zur Verfügung, der vom MC zu den Anschlusspins für die I2C Verbindung führt. Möchte man mit dem Raspberry Pi über I2C die Messwerte aus dem MC lesen, signalisiert man mit dem GPIO beispielsweise einen High-Pegel und der MC erkennt das und schaltet in den Slave Modus um bis die Daten an den Master geschickt wurden.

## Sleep

Das Nutzen der Sleep Modi für den MC sind nicht notwendig, da wenn der DS2777 in den Sleep Modus wechselt der MC keine Stromversorgung mehr hat. Der MC kann den DS2777 in den Sleep versetzen indem SDA und SCL auf einen LOW-Pegel gezogen werden.

## 5. Machbarkeitsstudie

In diesem Abschnitt geht es um die technische Machbarkeit. Das Ziel ist es das Konzept auf seine Funktion zu testen und Anforderungen und Verbesserungen für die Gesamtschaltung zu erschließen. Im ersten Teil (*Messungen des DS2777*) wurde das Messen und Ausgeben des Stroms, der Spannung, der Temperatur und die Schätzung des Ladezustandes überprüft. In diesem zweiten Teil wird getestet, ob die LIB vor Tiefentladung, Überladung und zu hohen Strömen geschützt sind und wie stark sich die stromführenden Leitungen und Bauteile erhitzen. Für das Kontrollieren das Lade- und Entladevorgangs gibt es jeweils einen MOSFET, der entweder sperrt oder leitet.

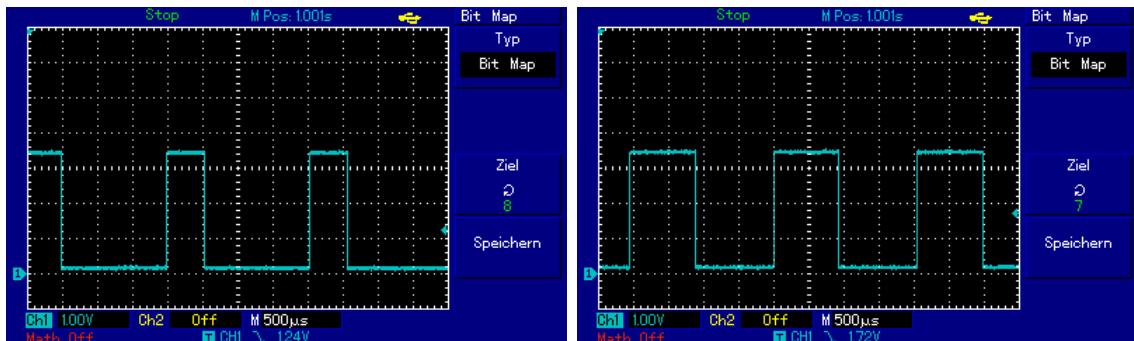


Abbildung 5.1.: Oszillogramm des PWM Signals mit einem Tastgrad von 25 % (links) und einem Tastgrad von 50 %

Der Testaufbau ähnelt Abbildung (4.1) mit dem Unterschied, dass anstatt eines MOSFETs ein Transistor verwendet wurde. Das PWM Signal (Abb. 5.1) hat wie programmiert eine Frequenz von 500 Hz und eine Amplitude von 3,3 V. Der Tastgrad, zwischen den Oszillogrammen, wurde mit dem Potentiometer verändert.

### Overvoltage

Die Ladeschlussspannung der verwendeten LIB beträgt 4,2 V. Deswegen wurde dieser Wert in das Overvoltage Threshold Register programmiert. Anstelle der LIB-Zellen werden zwei Labornetzteile benutzt. Wenn sich beide Labornetzteile im Arbeitsbereich der LIB bewegen, ist am Ausgang des Breakoutboards die Summe der beiden Spannungen zu sehen. Zu erwarten ist, dass beide MOSFETs ab einer Spannung von 4,2 V sperren. Es wurde die Gate Source Spannung beider MOSFETs gemessen. Abbildung (5.2) zeigt den gemessenen Verlauf.

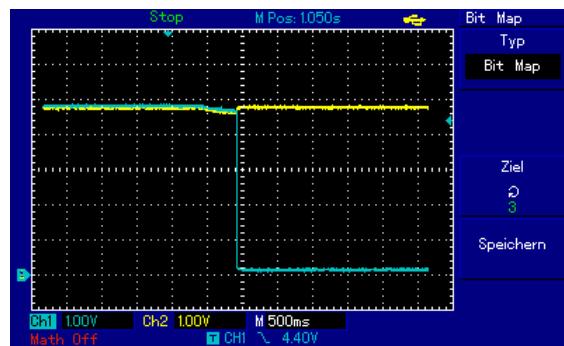


Abbildung 5.2.: Oszilloskop-Monitoring von U<sub>GSDC</sub> (gelb) und U<sub>GSOC</sub> (blau) bei Überspannung einer Zelle

Beide Labornetzgeräte befinden sich anfangs bei 3,6 V. Anschließend wurde die Spannung des Netzgerätes für die obere Zelle auf 4,4 V erhöht. Bei genau 4,2 V fällt die Gate Source Spannung des MOSFETs für die Ladekontrolle U<sub>GSOC</sub> auf 0 V und die Gate Source Spannung des MOSFETs für den Entladevorgang (U<sub>GSDC</sub>) bleibt stabil.

## Undervoltage

Als Entladeschlussspannung wurde 2,6 V programmiert. Beide Labornetzteile stehen auf 3,6 V und eines wird auf 2,4 V reduziert.

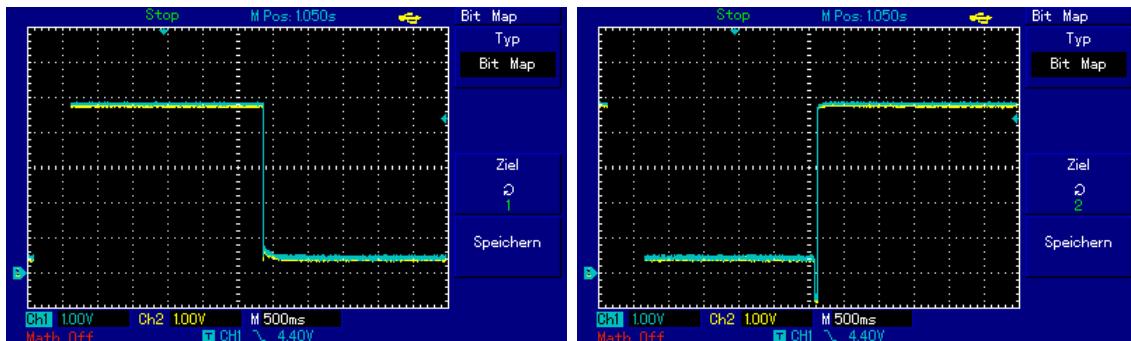


Abbildung 5.3.: Oszilloskop-Monitoring von U<sub>GSDC</sub> (gelb) und U<sub>GSOC</sub> (blau) beim Unterschreiten (links) und Überschreiten der Entladeschlussspannung

Bei 2,6 V fallen beide Gate Source Spannungen auf 0 V (siehe Abb. 5.3). Wenn der DS2777 in den Sleep Modus eintritt zeigt sich das gleiche Verhalten. Für ein undervoltage Event, kann konfiguriert werden, dass der Chip ebenfalls in den Sleep Modus eintritt und erst bei einem Wakeup Signal oder beim Erkennen eines Ladestroms aufwacht. Für diesen Test wurde diese Konfiguration nicht benutzt und man sieht in Abbildung (5.3) rechts, dass beide MOSFETs leiten, wenn man die Spannung der Zelle 1 wieder auf über 2,6 V erhöht.

## Overcurrent

Für das Testen von Strömen wurden anfangs für die kleineren Ströme die Labornetzteile mit Strombegrenzung verwendet und anschließend mit den LIB ersetzt. Die Ströme wurden schrittweise erhöht und es wurde bei 2 A nur eine sehr leichte Erwärmung an dem Rohm MOSFET spürbar. Für einen Kurzschluss am Ausgang sperrt der Entlademosfet bis der Kurzschluss gelöst wird.

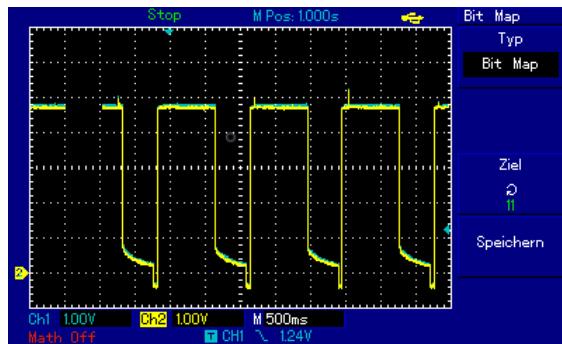


Abbildung 5.4.: Oszillogramm von  $U_{GSDC}$  (gelb) und  $U_{GSCC}$  (blau) beim einem zu hohen Dauerstrom

Für einen zu hohen Dauerstrom sperren beide MOSFETs (siehe Abb. 5.4). Wenn der PWM-MOSFET leitet, dann sperren beide MOSFETs und wenn er sperrt, dann leiten beide MOSFETs. Der Stromverbrauch ohne Last beträgt für das Breakoutboard  $85 \mu\text{A}$  und im Sleep Modus  $72 \mu\text{A}$ , unabhängig ob mit oder ohne Last, da diese im Sleep Modus getrennt wird.

# 6. Entwicklung der Gesamtschaltung

In diesem Abschnitt geht es um die Entwicklung der Gesamtschaltung anhand der gewonnenen Erkenntnisse. Die groben Anforderungen der Gesamtschaltung wurden in der *Einleitung* angeschnitten. Ein paar zusätzliche Überlegungen sind zu erwähnen. Die Hardware soll vielseitig ausgelegt werden, denn die Anpassung der Software ist leicht, aber die Anpassung der Hardware gestaltet sich im Nachhinein schwierig und bei der Erweiterung mit einem Bedienelement stehen so mehr Möglichkeiten zur Verfügung. Für Berechnungen in diesem Kapitel wird das PCB Tookit von SaturnPCB verwendet. Das Konzept der Gesamtschaltung (Abb. 6.1) ähnelt dem Konzept des Steuerteils (Abb. 4.1) mit zusätzlichen Funktionsblöcken.

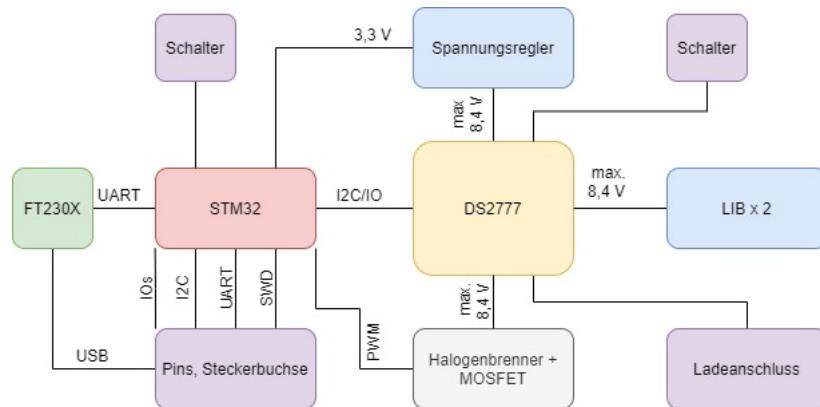


Abbildung 6.1.: Schematischer Aufbau der Gesamtschaltung

## Vorüberlegungen

Für die Platine ist keine Impedanzanpassung notwendig. Bei den Leitungsabmessungen und Bitraten ist die längste Leitung kürzer als 1/10 der kürzesten Wellenlänge. Length-Matching ist aufgrund der geringen Bitrate ebenfalls nicht notwendig. Nur bei den Datenleitungen der USB Schnittstelle wurde darauf geachtet, dass die Leitungslängen nicht zu stark voneinander abweichen.

## Kommunikation

Für die Kommunikation stehen USB, UART, I2C, SWD und IOs zur Verfügung. Um den Bootloader zu programmieren muss am BOOT0 Pin (Abb. 6.12) ein High-Pegel anliegen. Es gibt zwei Möglichkeiten den Bootloader zu Programmieren und beide werden implementiert.

Die erste Möglichkeit ist den Bootloader mit einem Programmieradapter von ST (STLink) über die SWD Schnittstelle zu programmieren. Die zweite Möglichkeit ist den Mikrocontroller mit der Software STMFlashLoader über USB zu flashen.

### Abmessungen

Die Abmessungen (Abb. 6.2) werden durch das zylinderförmige Lampengehäuse vorgegeben. Die 3 mm Bohrungen sind für Befestigungsschrauben. Die beiden 6 mm Bohrungen werden die Ladekontakte für das Laden der LIB und durch das 9 mm Loch in der Mitte werden die Leitungen des Halogenbrenners und der LIB geführt. Die kreisförmige Platine hat einen Durchmesser von 46 mm.

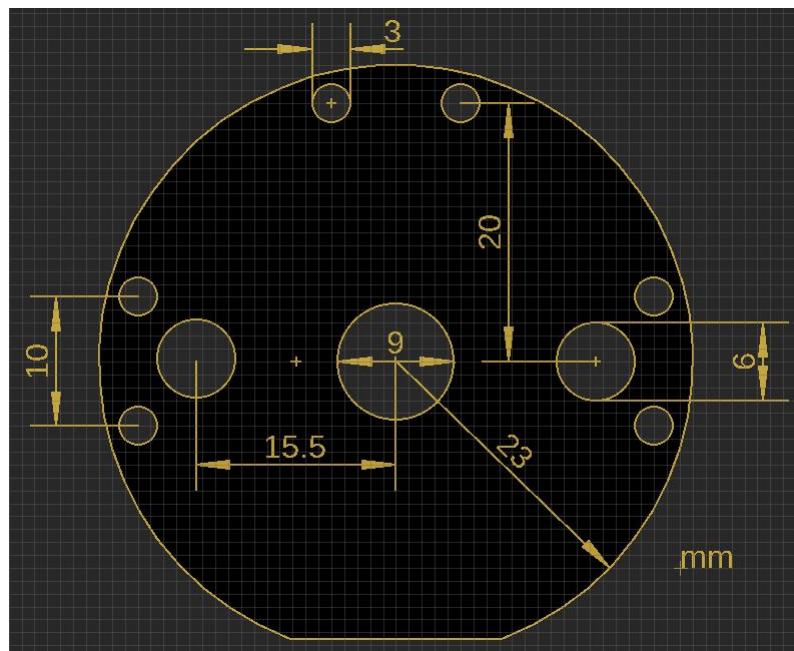


Abbildung 6.2.: Abmessungen der Platine

Der hat Platinenhersteller hat die Einschränkung, dass Bohrungen bis maximal 6,3 mm möglich sind. Dadurch kann die 9 mm Bohrung in der Mitte der Platine nicht realisiert werden. Als Lösung wurde an dieser Stelle eine 6,3 mm Bohrung beauftragt, die im Nachhinein nachbearbeitet wird. Die Kupferschichten in dem Bereich wurden mit ausreichend Abstand für eine 9 mm Bohrung ausgelegt und es wurde ein Ring mit einem 9 mm Durchmesser auf den Silkscreen Layer hinterlassen, um das Nachbearbeiten zu vereinfachen. Durch die Menge der Bauteile den geringen Abmessungen der Platine werden SMD-Bauteile der Größe 0603 verwendet, die groß genug sind, um die Platine händisch zu bestücken. Außerdem wurde nach einem erfolglosem Routingversuch mit zwei Layer eine 4 Layer Platine entworfen (Abb. 6.3).

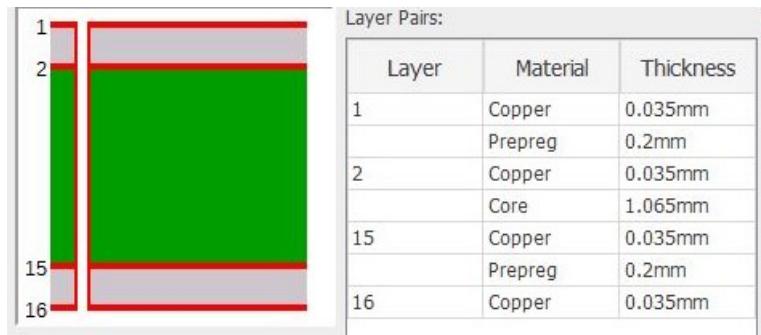


Abbildung 6.3.: Seitenansicht der verwendeten Layer und ihre Abmessungen

Die beiden äußeren Layer werden für die Signalleitungen verwendet und die beiden inneren als GND und Power Plane. Die Dicke der verschiedenen Schichten ist für die kommenden Berechnungen erforderlich und wird in diesem Fall durch den Hersteller vorgegeben.

## 6.1. BMS-Teil

Das Schaltbild des BMS-Teil der Gesamtschaltung (Abb. 6.4) ähnelt dem Schaltbild des Breakoutboards (Abb. 3.4). Es wurden jedoch ein paar Anpassungen vorgenommen. Die Änderungen ergeben sich durch die Ergebnisse der Machbarkeitstests und dem Vergleich mit einem Evaluation Board von Maxim auf dem ein DS2775 verbaut wurde. Die LIB liegen in Reihe und BAT1+ ist der Anschluss für das positive Ende und BAT2- für das negative Ende. BAT2+ ist der Anschluss für die Zwischenspannung. Die Anschlüsse werden mit einfachen, quadratischen Pads realisiert. PK+ zu GND ist die gemanagte Ausgangsspannung.

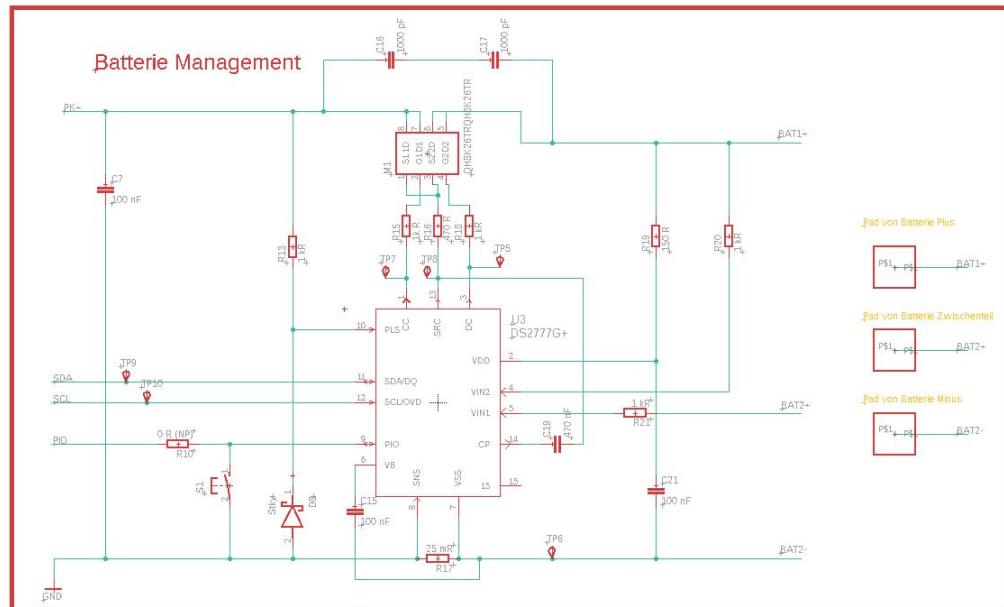


Abbildung 6.4.: Schaltbild des Batterie Management Teils mit dem DS2777

Die Z-Dioden an den Kommunikationseingängen wurden mit TVS Dioden ersetzt und in die Nähe der Steckverbinder, die nach außen führen, platziert (siehe Abbildung 6.13).

	BZX584C5V6	CPDU3V3UP
$I_R$	1 $\mu$ A	0,05 $\mu$ A
$U_R$	5,6 V	3,5 V

Tabelle 6.1.: Gegenüberstellung des Sperrstroms und der Durchbruchspannung der alten Z-Diode und der neuen TVS-Diode (rechts)

TVS-Dioden haben in der Regel eine geringere Kapazität und sind deswegen besser für Datenleitungen geeignet, auch wenn es hier nicht um sehr hohe Geschwindigkeiten geht. Die Kommunikationsleitungen sollen auf 3,3 V ausgelegt werden, deswegen wurde eine TVS-Diode mit einer Stand-Off-Sperrspannung von 3,3 V ausgewählt, bei welcher kein Strom fließt, und einer Durchbruchspannung von 3,5 V, um höhere Spannungen zu unterdrücken. Die TVS-Diode besitzt einen geringeren Sperrstrom, was zu dem Ziel eines geringen Stromverbrauchs der Schaltung beiträgt. [17] [18]

Der Strommesswiderstand in Form einer Leitung wurde mit einem tatsächlichen Strommesswiderstand ersetzt. Der Grund dafür ist, dass dadurch Platz gespart wird, breitere Leitungen und eine Kupferdicke von 35  $\mu$ m verwendet werden können und die Leitungen die viel Strom führen besser von dem DS2775 isoliert werden können. Beim Auswählen des Strommesswiderstandes wurde darauf gearichtet, dass die Abweichungstoleranz gering ist und der Widerstand genug Leistung verträgt.

Es wurde ein Schalter und ein 0  $\Omega$  Widerstand hinzugefügt, der bei Bedarf bestückt werden kann. Wird der Widerstand bestückt hat man eine Verbindung zu einem GPIO des Mikrocontrollers oder die Möglichkeit eine Leitung an das Pad anzubringen das zu einem Bedienelement führt. Der DS2777 kann so konfiguriert werden, dass er beim Betätigen des Schalters aus dem Sleep aufwacht.

Außerdem wurden Testpunkte für Steuersignale der MOSFETS, der negativen Spannung der LIB und der I2C Schnittstelle hinzugefügt.

## Berechnungen

Für das Erstellen des Layouts ist es notwendig die Leitungen, die viel Strom führen so auszulegen, dass sie sich nicht zu stark erhitzen. Dafür müssen die Leitungen einen geringen Ohmschen Widerstand besitzen bzw. breit genug und kurz genug sein. Im folgenden werden dazu Rechnungen Vorgenommen.

Die Ausgangsleitungen des BMS-Teils, die höheren Strom leiten müssen, führen zu der Halogenlampe und dem MOSFET, der über PWM angesteuert wird. Wie in Abschnitt (3.2) erwähnt wurde liegt der maximale Dauerstrom, der Chip-bedingt geführt werden kann, bei 4 A und der maximale Pulstrom bei 12 A. Für die Berechnung (Abb. 6.5) der Leitungsausmaße

## KAPITEL 6. ENTWICKLUNG DER GESAMTSCHALTUNG

zwischen PK+, Halogenbrenneranschluss, MOSFET und GND werden die Leitungslänge, die Dicke der gesamten Platine, der Abstand zwischen der Leitung und der Referenzplatte, die Höhe der Leitung und die Art des Substratmaterials benötigt.

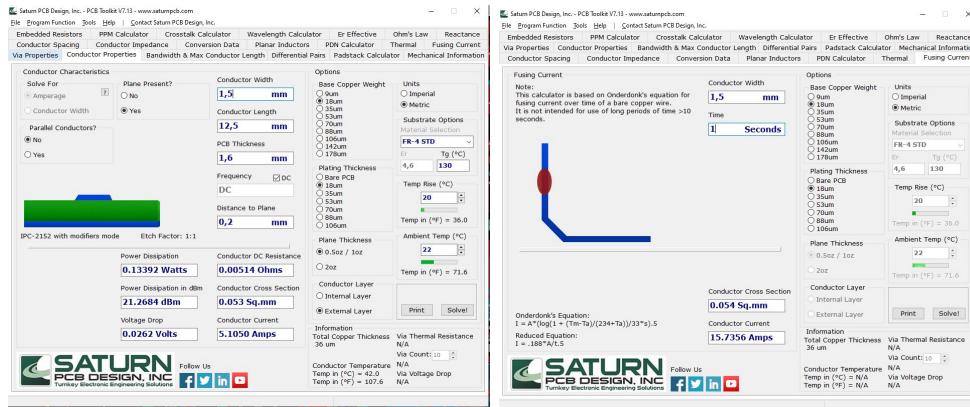


Abbildung 6.5.: Links: Berechnung der Leitungsbreite für einen Dauerstrom von 5 A.

Rechts: Berechnung der maximalen Stromstärke bei den ermittelten Leitungsabmessungen

Bei einer Breite von 1,5 mm besitzt die Leitung bei einem Dauerstrom von 5,1 A einen Temperaturanstieg von 20°C. Bei der ermittelten Leitungsbreite von 1,5 mm kann die Leitung für eine Sekunde einen Pulssstrom von 15,7 A führen bis die Leitung zerstört wird.

### Ladeanschluss

Der Ladestrom für die verwendeten LIB beträgt 2,5 A. Bei einem Leitungsdurchmesser von 1,2 mm können für eine Erwärmung von 20°C 4,5 A fließen (siehe 6.6). [19]

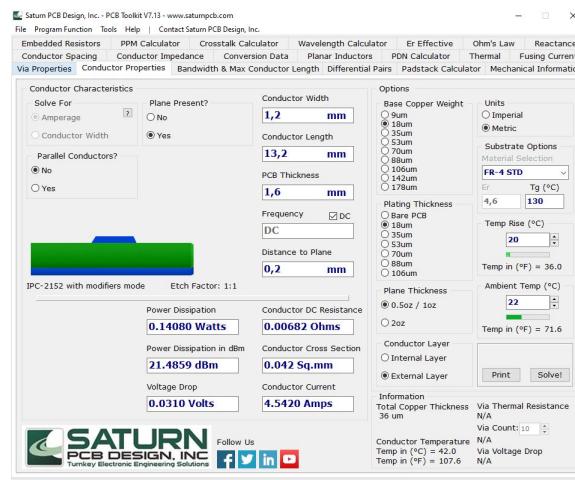


Abbildung 6.6.: Berechnung der Leitungsbreite für die Ladeleitungen

## Layout

In Abbildung 6.7 sind die Leitungen zu sehen, die höhere Ströme führen. Die Leitungen wurden kurz gehalten und entsprechend der berechneten Breite ausgelegt.

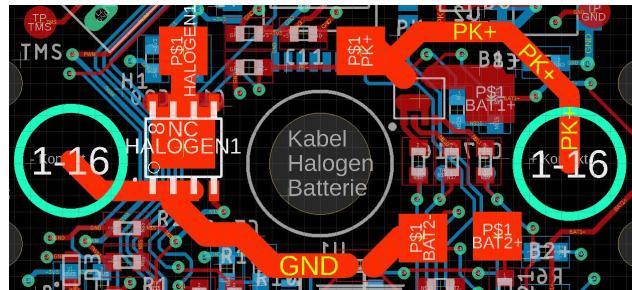


Abbildung 6.7.: Leitungen die hohen Strom führen im Layout

Die Ladekontakte für das Aufladen der LIB wurden mit Vias realisiert. Die Ströme führen im Vergleich zum Breakoutboard über definierte Leitungswege und nicht über Kupferflächen.

## 6.2. PWM Teil

Der PWM Teil besteht aus den Anschlusspads für den Halogenbrenner, einem POWER MOSFET, der über ein PWM-Signal von dem MC angesteuert wird und zwei Widerständen (siehe 6.8).

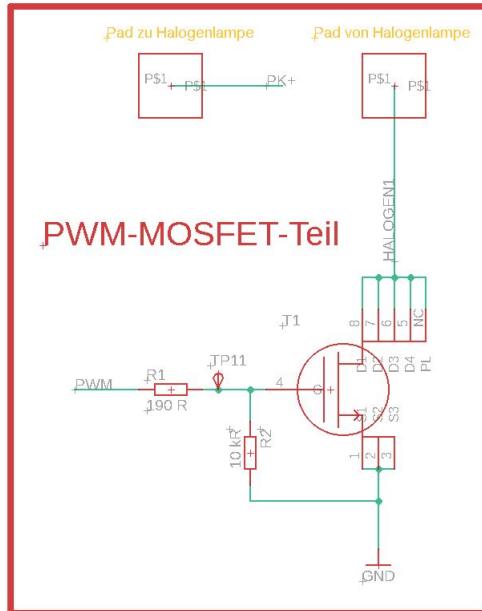


Abbildung 6.8.: Schaltbild des PWM-MOSFET-Teils

Die Anforderungen an den MOSFET sind, dass er einen geringen Innenwiderstand besitzt und von dem MC angesteuert werden kann. Beim Einschaltvorgang, wenn der Halogenbrenner kalt ist, kommt es wegen dem geringen Kaltwiderstand zu einem höheren Strom. Der MOSFET muss diesen Strom vertragen können. Ein weiterer wichtiger Aspekt ist die Gate Ladung. Wenn die Gate Ladung zu hoch ist und die Umladung zu langsam verläuft dann befindet sich der MOSFET zu lange im linearen Bereich, der zu vermeiden ist, weil Leistung verloren geht und es zu einer starken Erwärmung kommen kann. Als MOSFET wurde der SIRA90DP (Tabelle 6.2) ausgewählt.

$V_{DS}$	30 V
$R_{DS(on)}$	1,5 mΩ
$I_D$	100 A
$I_{DSS}$	1 μA
$Q_g$	28 nC
$V_{GS(th)}$	0,8 V bis 2 V

Tabelle 6.2.: Wichtige Parameter des SIRA90DP [20] für die Anwendung als PWM Schalter

Beim Umladen der Gate Ladung kann es zu hohen Strömen kommen, die den MC beschädigen können. Der Maximale Ausgangsstrom des Pins für die Erzeugung des PWM Signals beträgt 22 mA und der maximale Eingangsstrom 16 mA [12, Tab. 18]. Deshalb wird der Umladestrom durch einen seriellen Widerstand begrenzt, der groß genug sein muss, um den Strom zu begrenzen und klein genug damit die Umladung schnell genug geschieht. Für das Ermitteln der Größe des Serienwiderstandes und das Überprüfen der Parameter des eingesetzten MOSFETs in seiner Umgebung wird in LTSpice eine Simulation durchgeführt (Abb. 6.9).

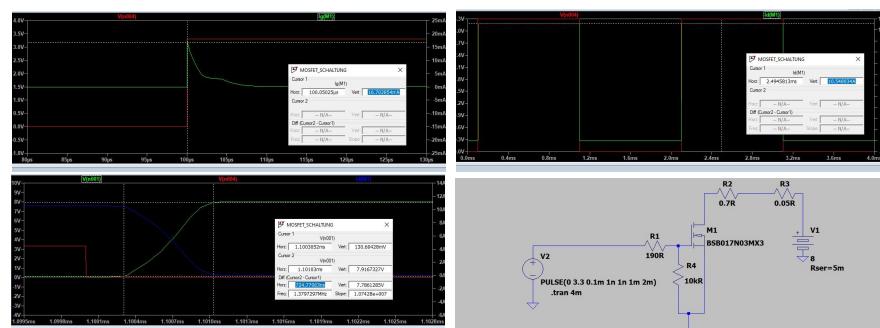


Abbildung 6.9.: LTspice Simulation des PWM-MOSFET-Teils

Für die Simulation wurde der Widerstand einer Halogenlampe im kalten Zustand gemessen. Der Widerstand betrug 0,7 Ω. Für die Ansteuerung des MOSFETs wurde ein PWM Signal mit einer Spannung von 3,3 V, einer Frequenz von 500 Hz und einem Tastgrad von 50 % verwendet. Der verwendete MOSFET hat einen anderen Namen in der Abbildung, besitzt aber annähernd gleiche Eigenschaften. In Abbildung (6.9) oben sieht man, dass sich

für einen seriellen Widerstand von  $190 \Omega$  ein Umladestrom von knapp  $16 \text{ mA}$  ergibt. Außerdem ist der Simulation zu entnehmen, dass sich ein Anfangsstrom von  $10,5 \text{ A}$  ergibt und die Zeitspanne des Umladevorgangs mit  $190 \Omega$  im Nanosekundenbereich liegt. Der parallele Pull-Down Widerstand von  $10 \text{ k}\Omega$  könnte weggelassen werden ist aber zur Sicherheit hinzugefügt worden um den Puls im LOW-Zustand auf das gleiche Potenzial zu ziehen das der MOSFET an seinem Source Anschluss sieht.

### 6.3. Spannungsregler

Die Aufgabe des Spannungsreglers ist es die Spannung PK+, die aus dem BM-Teil kommt auf eine Stabile  $3,3 \text{ V}$  Spannung zu regeln.

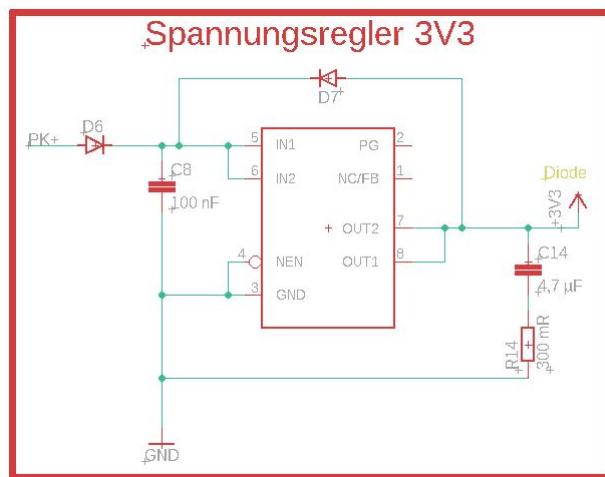


Abbildung 6.10.: Schaltbild des Spannungsreglers

Die  $3,3 \text{ V}$  werden für die Stromversorgung des MC und für die Schalter und Pull-Up Widerstände verwendet. Die Ausgangsspannung des BMS-Teils kann zwischen  $4 \text{ V}$  und  $8,4 \text{ V}$  variieren, je nach Ladezustand der LIB. Dabei soll der Spannungsregler genug Strom liefern um alle beteiligten Bauteile zu versorgen und dabei selbst wenig Strom verbrauchen. Der MC hat eine maximale Stromaufnahme von  $105 \text{ mA}$  [12, Tab. 18]. Es wurde der TPS76633DR ausgewählt.

Ausgangsstrom	$250 \text{ mA}$
Ausgangsspannung	$3,3 \text{ V}$
Eingangsspannung	$-0,3 \text{ V}$ bis $13,5 \text{ V}$
Ruhestrom	$35 \mu\text{A}$

Tabelle 6.3.: Kurzübersicht der relevanten Parameter des TPS76633DR

Die beiden Dioden wurden zusätzlich hinzugefügt. Die linke Diode schützt den IC vor negativen Spannungen am Eingang wenn der Akku aufgeladen wird. Die Diode die vom

Ausgang auf den Eingang des ICs führt schützt die Ausgangspins vor Strömen, wenn der MC über SWD programmiert wird und die 3,3 V Leitung des Programmierers mit angesteckt wird. So kann man den MC auch ohne LIB programmieren.

## 6.4. USB zu UART

Um über den PC mit der Tauchschaltung zu kommunizieren ist eine USB-Verbindung ein einfaches Mittel. Um eine USB-Verbindung zu realisieren wurde der FT230X verwendet, der zwischen USB und UART umwandelt.

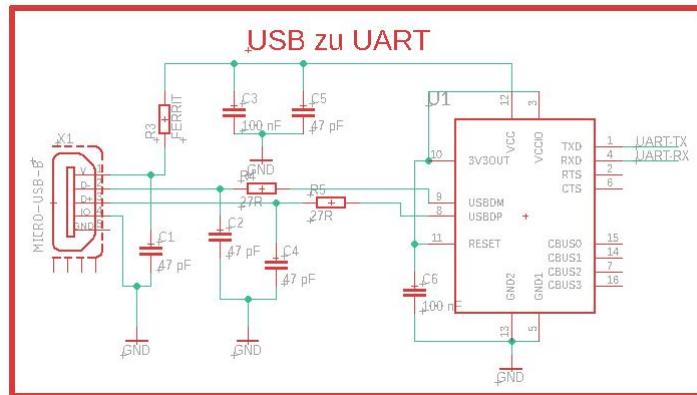


Abbildung 6.11.: Schaltbild des USB zu/von UART-Teils

Als USB Anschluss wurde ein Micro USB Buchse verwendet. Der Chip wird über die Powerleitung der USB-Verbindung versorgt, wenn ein Stecker angesteckt wird. Wenn kein Stecker angesteckt ist wird der Chip ohnehin nicht benötigt. Dadurch wird erreicht auf eine Stromversorgung durch den Spannungsregler zu verzichten und letztendlich eine längere Laufzeit der LIB ermöglicht. Der Chip kann über USB programmiert und konfiguriert werden und wird von dem Betriebssystem als COM Port erkannt.

## 6.5. Mikrocontroller

Die Aufgabe des STM32 ist die Kommunikation über I2C mit dem DS2777 und über die anderen Schnittstellen mit der Außenwelt. Eine weitere Aufgabe ist das Ansteuern des MOSFETs für das Dimmen der Halogenlampe.

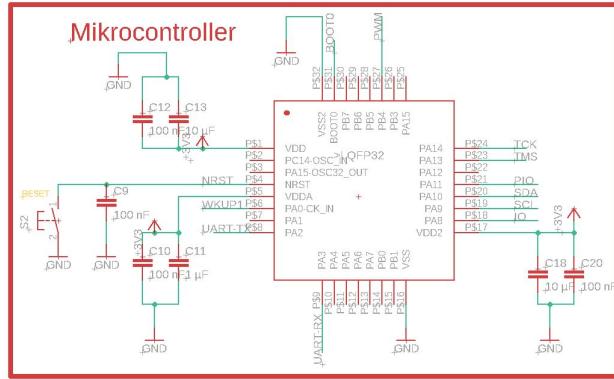


Abbildung 6.12.: Schaltbild des Mikrocontrollers

An jedem Versorgungspin des STM32 hängen Entkoppelkondensatoren, deren Kapazitätswerte im Datenblatt empfohlen werden und möglichst nah an die jeweiligen Pins angebracht werden sollten. Die Pins, die für die USART Schnittstelle verwendet werden können ebenfalls als UART und als LPUART (Low-Power-UART) konfiguriert werden. Außerdem kann der Pin PA2 auch als oder Wakeup Pin fungieren. Ein Schalter wurde an dem NRST Pin befestigt, der einen Reset verursacht, wenn er betätigt wird. Der selbe Pin führt zur Schnittstelle für den Programmierer der SWD benutzt. Ein GPIO Pin führt zum DS2777 und der andere nach draußen.

## 6.6. Schnittstellen

Die Schnittstellen ermöglichen es von Außen mit den programmierbaren Bausteinen auf der Taucherschaltung zu kommunizieren. Es wurden zwei Stiftleisten vier Pads und ein Dipschalter verwendet.

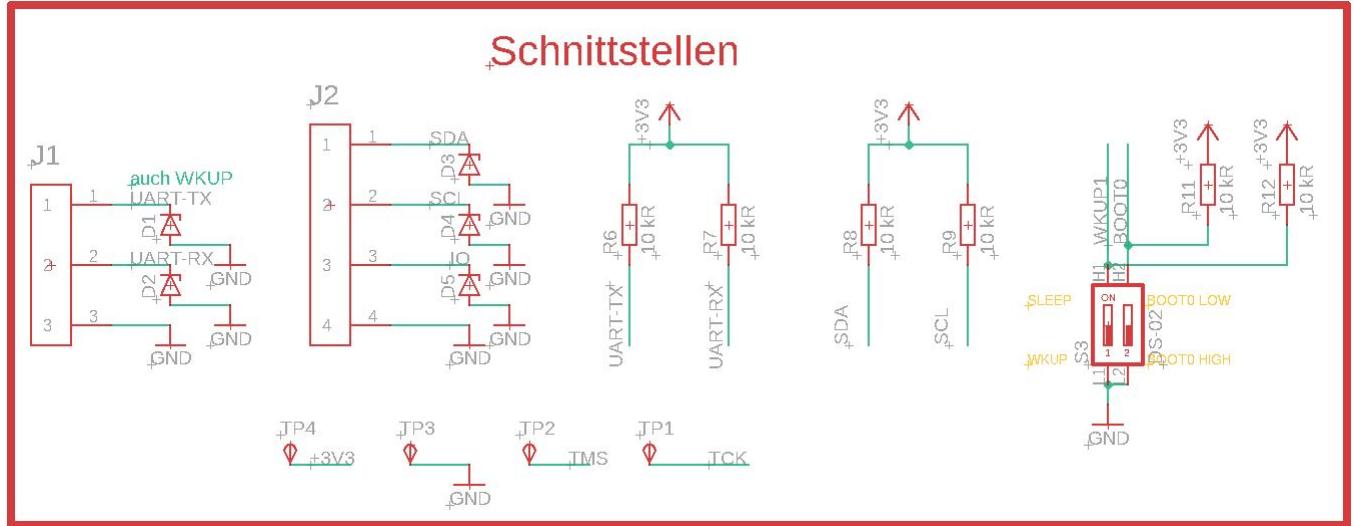


Abbildung 6.13.: Schaltbild der Schnittstellen

Die Stiftleisten führen UART, I2C und den GPIO nach außen. Wie in Abschnitt (6.1) besprochen wurden die TVS-Dioden nah an die Stiftleisten platziert, um zu hohe Spannungen schon beim Eintreten in die Schaltung abzuschneiden. Für die UART und I2C Busse wurden leichte Pull-Up Widerstände von 10 kR benutzt. Der Dipschalter ermöglicht verschiedene Einstellungen am STM32. Der linke Schalter ermöglicht es den Chip manuell in den Aktiven und in einen Schlafmodus zu versetzen. Der rechte Schalter muss für das Flashen des Bootloaders in die Position gebracht werden bei der BOOT0 einen High-Pegel sieht (in Richtung GND).

## Resultat

Die Gerberdaten wurden erstellt, dem Hersteller geschickt und die Platinen geliefert. In Abbildung (6.14) ist die Platine der Tauchlampensteuerung von beiden Seiten und eingesetzt in die Vorrichtung der Tauchlampe zu sehen.

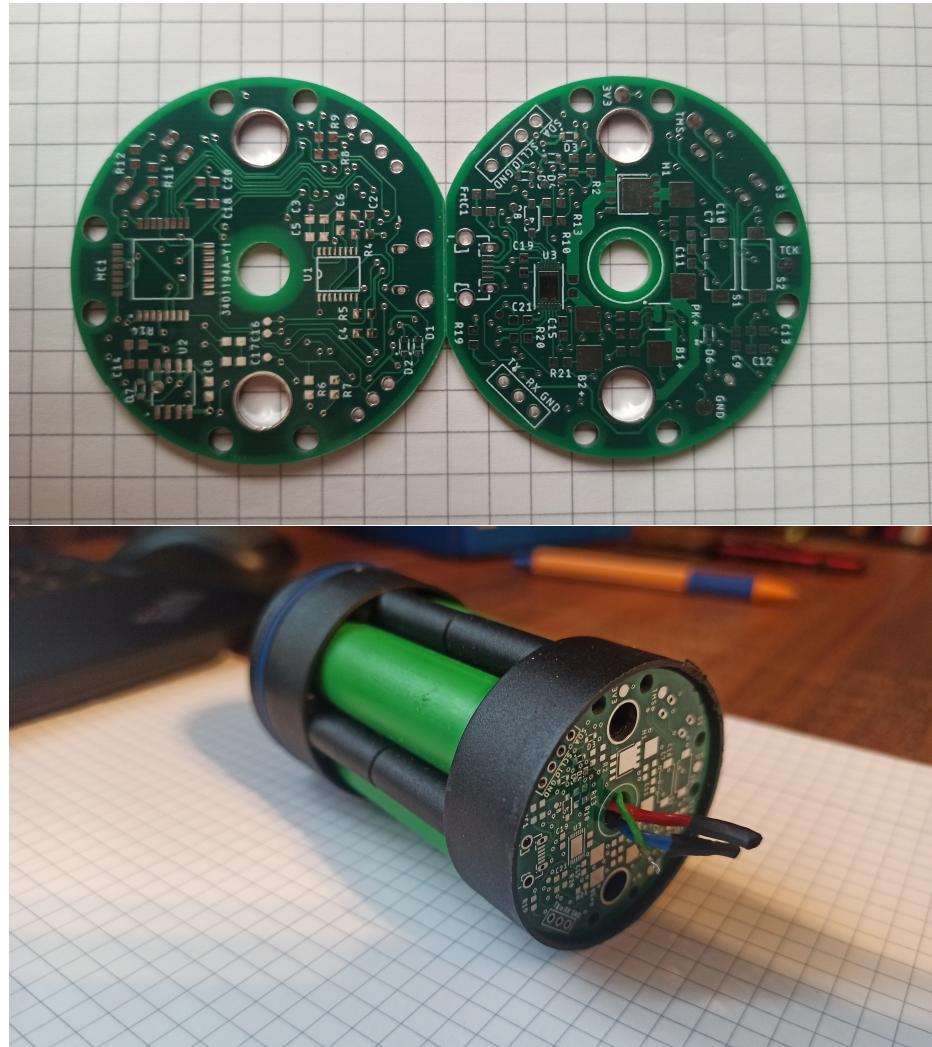


Abbildung 6.14.: Beidseitige Ansicht und Einsetzen der Platine der Tauchlampensteuerung

## 7. Ergebnisdiskussion

Die entwickelte Tauchlampensteuerung für Lithiumionenbatterien erfüllt nahezu die gesamten Zielsetzungen, die vor Beginn der Bachelorarbeit aufgestellt wurden.

Es wurde eine Platine mit der Beschaltung des DS2777 entwickelt und überarbeitet. Dafür wurde die Schaltung und das Layout designed, bestückt, mit einem Raspberry Pi initial programmiert und in einem Testaufbau getestet. Die Schaltung schützt erfolgreich bei den einprogrammierten Werten vor, Überspannung, Unterspannung und zu hohen Dauerströmen und Kurzschlusstrom.

Es wurde ein STM32L031k6 in C programmiert, der über I2C mit dem DS2777 kommuniziert, eine MOSFET über ein PWM Signal mit variablem Tastgrad für das Dimmen der Halogenlampe ansteuert und die Messdaten über UART und den Nutzer ausgibt. Für das Steuern des Tastgrades wurde ein Potentiometer eingelesen. Alternative Wege für das Steuern des Tastgrades und für das Ausgeben der Daten wurde hardwareseitig implementiert und softwareseitig angesprochen aber (softwareseitig) nicht in diesem Dokument aufgeführt.

Der Schutz vor Tiefentladung der LIB bei Nicht-Benutzen der Tauchlampe wurde durch den geringen Verbrauch des BMS-Teils mit  $72 \mu\text{A}$  im Sleepmodus bewerkstelligt, wobei die restliche Schaltung von der Stromquelle getrennt wird.

Die Schaltung wurde für einen geringen Verbrauch im Betrieb ausgelegt. Der FT230X wird nicht von den LIB versorgt, sondern durch das USB Kabel, das extern angeschlossen wird. Der verwendete MC ist ein ultra Low-Power MC und wird mit einem geringen Systemclock betrieben der  $4,5 \mu\text{A}$  verbraucht. Es wurden höhere Pull-Up Widerstände mit  $10 \text{ k}\Omega$  benutzt und TVS Dioden mit sehr geringem Sperrstrom von  $0,05 \mu\text{A}$ . Der Spannungsregler besitzt ebenfalls einen geringen Energieverbraucht mit  $35 \mu\text{A}$ .

Um eine starke Erwärmung durch die hohen Ströme zu vermeiden wurden MOSFETs mit einem geringen  $R_{DS(on)}$  ausgewählt und komplett durchgeschaltet. Die Leitungsausmaße wurden entsprechend der Ströme berechnet und ausgelegt.

Das Messen der Zellspannungen über den DS2777 funktioniert, da die MOSFETs bei den erforderlichen Schwellwerten reagieren, aber das Speichern der richtigen Werte in die erforderlichen Register funktioniert nicht. Die Registerwerte der Echtzeitmessungen sind falsch. Die Vermutung für den Fehler besteht in einem Schaden des D2777, der beim Lötprozess im Reflow Ofen entstanden ist. Da die Kupferdicke für das Breakoutboard doppelt so dick war wurde für den Reflow Ofen ein anderes Lötprogramm durchlaufen bei dem die Maximale Löttemperatur des DS2777 überschritten wurde [11, S. 2].

Die Hardware wurde vielseitig ausgelegt, um auf verschiedene Weisen programmiert zu werden zu können und mit dem Hintergrund für die Erweiterung mit einem Bedienelement zur Verfügung zu stehen. Dafür wurden SWD, USB, USART, LPUART, I2C, IOs und verschiedene Anschlussmöglichkeiten und Schalter hardwareseitig implementiert. Der Bootloa-

der lässt sich ebenfalls auf zwei verschiedene Wege programmieren. Über einen ST-Link Adapter und SWD oder über USB und dem STMFlashLoader.

Die Platine der Gesamtschaltung mit den notwendigen Ausmaßen wurde entwickelt und passt in das Tauchlampengehäuse. Auf der Platine sind Anschlüsse für die Kabel des Halogenbrenners, der LIB und Kontakte für das Ladeteil zum Aufladen der LIB vorhanden.

Die Gesamtschaltung tritt in den Sleep Modus ein, wenn an den I2C Pins des DS2777 Low-Pegel anliegen. Die Gesamtschaltung wechselt in den aktiven Modus, wenn ein Low-Pegel am IO-Pin des DS2777 vorhanden ist.

## 8. Ausblick

Die Gesamtschaltung wurde modular mit allen Modulen, die auf der Platine der Gesamtschaltung mit Formfaktor vorhanden sind, aufgebaut und getestet. Aber die fertige Platine wurde noch nicht bestückt und in Betrieb genommen. Damit die Tauchlampensteuerung in kommerziellen Tauchlampen eingesetzt werden kann muss die Platine weiteren Optimierungsverfahren unterzogen werden. Die Hardware ist vielseitig ausgelegt, was wegen der Vielzahl an Bauteilen finanzielle und verbrauchbezogene Nachteile hat. Der Sinn dahinter ist anhand der vielseitigen Hardware Tests durchzuführen und Software zu schreiben, die das Verhalten optimiert und herausarbeitet, welche Schnittstellen und Bauteile letztendlich tatsächlich notwendig sind. Anschließend kann man die überflüssigen Schnittstellen, Steckverbinder und Bauteile entfernen. Für die nächste Version der Gesamtschaltung sollten die Body-Dioden der MOSFETs für die Lade- und Entladekontrolle mit einer externen Diode zwischen Drain und Source unterstützt werden, da diese zwar für Pulssströme von 18 A ausgelegt sind aber nicht für Dauerströme über 1,5 A. Ein weiterer Verbesserungsvorschlag ist ein Pull-Up Widerstand an dem IO des DS2777, der für das Umschalten in den aktiven Zustand verantwortlich ist.

Neben der Funktion als Tauchlampensteuerung kann die Platine als Experimentierboard oder Entwicklungsboard für den STM32L031k6 dienen, da alle verwendeten Schnittstellen mit Pins nach außen geführt werden und für jeden relevanten Pin und Messwert ein Testpunkt vorhanden ist. Die nach außen geführten Pins können intern umkonfiguriert werden. Dabei ist zu beachten, dass die Funktionsbreite bei manchen Pins durch einen Pull-Up Widerstand eingeschränkt ist, der bei Bedarf entfernt werden könnte.

Die Tauchlampensteuerung kann mit einem Bedienteil erweitert werden, das über eines oder mehrere der vorhandenen Schnittstellen mit dem MC kommuniziert. Dabei kann ein Ausschalter vorhanden sein, der dem MC signalisiert den DS2777 in den sleep zu versetzen und so die Lampe ausschaltet und ein Einschalter der den IO-Pin des DS2777 auf einen Low-Pegel zieht und die Lampe anschaltet. Die Echtzeitmesswerte und Ladezustandsschätzung können über einen mini-Display ausgegeben werden. Eine zusätzliche Erweiterung wäre ein Ladegerät, das die verwendeten LIB Akkus mit der CVCC-Methode (constant current constant voltage) auflädt und auf die vorhandenen Ladekontakte passt.

# Literaturverzeichnis

- [1] MARX, Robert F.: *The history of underwater exploration*. Courier Corporation, 1990
- [2] LURIA, SM ; KINNEY, Jo Ann S.: Underwater vision. In: *Science* (1970), S. 1454–1461
- [3] MACISAAC, Dan ; KANNER, Gary ; ANDERSON, Graydon: Basic physics of the incandescent lamp (lightbulb). In: *The physics teacher* 37 (1999), Nr. 9, S. 520–525
- [4] CHENG, Yuen-Kit ; CHENG, KWE: General study for using LED to replace traditional lighting devices. In: *2006 2nd International Conference on Power Electronics Systems and Applications* IEEE, 2006, S. 173–177
- [5] ZUBI, Ghassan ; DUFO-LÓPEZ, Rodolfo ; CARVALHO, Monica ; PASAOGLU, Guzay: The lithium-ion battery: State of the art and future perspectives. In: *Renewable and Sustainable Energy Reviews* 89 (2018), S. 292 – 308. – ISSN 1364–0321
- [6] WEICKER, Phil: *A systems approach to lithium-ion battery management*. Artech house, 2013
- [7] HARTENBERGER UNTERWASSERTECHNISCHE GERÄTE GMBH (Hrsg.): *Betriebsanleitung Hartenberger High-Tech-Kleinleuchte mini compact LCD*. Hartenberger UnterwasserTechnische Geräte GmbH, 1 2016. [https://www.hartenberger.de/pdf/bet\\_mcLCD.pdf](https://www.hartenberger.de/pdf/bet_mcLCD.pdf)
- [8] ; IEC-International Electrotechnical Commission;DKE: Deutsch Ausgabe des IEV (Veranst.): *IEC 60050-482-05-07*. Mai 2016
- [9] KORTHAUER, Reiner: *Handbuch lithium-ionen-batterien*. Springer, 2013
- [10] PLETT, Gregory L.: *Battery management systems, Volume II: Equivalent-circuit methods*. Artech House, 2015
- [11] MAXIM INTEGRATED (Hrsg.): *DS2775/DS2776/DS2777/DS2778*. Maxim Integrated, 2 2017. (19-4688) . – Rev. 5
- [12] STMICROELECTRONICS N.V. (Hrsg.): *STM32L031x4 STM32L031x6*. STMicroelectronics N.V., 3 2018. (DS10668) . – Rev. 6
- [13] NXP SEMICONDUCTORS (Hrsg.): *I2C-bus specification and user manual*. NXP Semiconductors, 4 2014. (UM10204) . – Rev. 6
- [14] ROHM K.K. (Hrsg.): *QH8K26*. Rohm K.K., 5 2019. (20190527) . – Rev.003

- [15] HÄNDSCHE, Jürgen: *Leiterplattendesign*. Erste Auflage. Eugen G. Leuze Verlag KG, 2006. – ISBN 978-3-87480-329-8
- [16] STMICROELECTRONICS N.V. (Hrsg.): *UM1749*. STMicroelectronics N.V., 12 2016. (026232) . – Rev. 6
- [17] COMCHIP TECHNOLOGY Co., LTD. (Hrsg.): *SMD ESD Protection Diode CP-DU3V3UP*. Comchip Technology Co., Ltd., (QW-G7056) . – Rev. C
- [18] VISHAY INTERTECHNOLOGY, INC. (Hrsg.): *BZX584C-Series*. Vishay Intertechnology, Inc., 11 2019. (85793) . – Rev. 1.8
- [19] SONY CORPORATION (Hrsg.): *Lithium Ion Rechargeable Battery Technical Information*. Sony Corporation, 5 2015. (49934720) . – Rev. 1.0
- [20] VISHAY INTERTECHNOLOGY, INC. (Hrsg.): *N-Channel 30 V (D-S) MOSFET SiRA90DP*. Vishay Intertechnology, Inc., 4 2004. (67854) . – Rev. A

# Abbildungsverzeichnis

2.1. DS2777 neben einer 1 Cent Münze . . . . .	8
3.1. Schematische Darstellung des Kontextes in dem das Breakoutboard eingebettet wird . . . . .	9
3.2. Schaltbild des Breakoutboards . . . . .	10
3.3. Layout Breakoutboard . . . . .	11
3.4. Das bestückte Breakoutboard neben einer 1 Cent Münze . . . . .	12
3.5. Adressen und Beschreibung der Zell- und Anwendungsparameter im EEPROM [11, Tab. 8] . . . . .	13
4.1. Schematische Darstellung des Steuerteil Aufbaus . . . . .	17
5.1. Oszilloskopgramm des PWM Signals mit einem Tastgrad von 25 % (links) und einem Tastgrad von 50 % . . . . .	20
5.2. Oszilloskopgramm von $U_{GSDC}$ (gelb) und $U_{GSCC}$ (blau) bei Überspannung einer Zelle . . . . .	21
5.3. Oszilloskopgramm von $U_{GSDC}$ (gelb) und $U_{GSCC}$ (blau) beim Unterschreiten (links) und Überschreiten der Entladeschlussspannung . . . . .	21
5.4. Oszilloskopgramm von $U_{GSDC}$ (gelb) und $U_{GSCC}$ (blau) beim einem zu hohen Dauerstrom . . . . .	22
6.1. Schematischer Aufbau der Gesamtschaltung . . . . .	23
6.2. Abmessungen der Platine . . . . .	24
6.3. Seitenansicht der verwendeten Layer und ihre Abmessungen . . . . .	25
6.4. Schaltbild des Batterie Management Teils mit dem DS2777 . . . . .	25
6.5. Links: Berechnung der Leitungsbreite für einen Dauerstrom von 5 A. Rechts: Berechnung der maximalen Stromstärke bei den ermittelten Leitungsabmessungen . . . . .	27
6.6. Berechnung der Leitungsbreite für die Ladeleitungen . . . . .	27
6.7. Leitungen die hohen Strom führen im Layout . . . . .	28
6.8. Schaltbild des PWM-MOSFET-Teils . . . . .	28
6.9. LTspice Simulation des PWM-MOSFET-Teils . . . . .	29
6.10. Schaltbild des Spannungsreglers . . . . .	30
6.11. Schaltbild des USB zu/von UART-Teils . . . . .	31
6.12. Schaltbid des Mikrocontrollers . . . . .	32
6.13. Schaltbild der Schnittstellen . . . . .	33
6.14. Beidseitige Ansicht und Einsetzen der Platine der Tauchlampensteuerung . .	34

# Tabellenverzeichnis

3.1. Eine Auflistung der Anforderungen an die MOSFETs (Mitte . . . . .	11
3.2. Messwerte des DS2777 bei einem Lastwiderstand von $5\text{ k}\Omega$ in Gegenüberstellung mit den Sollwerten . . . . .	15
6.1. Gegenüberstellung des Sperrstroms und der Durchbruchspannung der alten Z-Diode und der neuen TVS-Diode (rechts) . . . . .	26
6.2. Wichtige Parameter des SIRA90DP [20] für die Anwendung als PWM Schalter	29
6.3. Kurzübersicht der relevanten Parameter des TPS76633DR . . . . .	30

# A. Anhang

```
1  /*
2  Bei der Gesamtschaltung muessen die Pins
3  aus dem Schaltplan verwendet werden
4  Hier wird die Pinverschaltung aus
5  dem Nucleo Board verwendet
6  */
7
8 #include "main.h"
9 #include <string.h>
10 #include <stdio.h>
11
12 ADC_HandleTypeDef hadc;
13 I2C_HandleTypeDef hi2c1;
14 TIM_HandleTypeDef htim22;
15 UART_HandleTypeDef huart2;
16
17 void SystemClock_Config(void);
18 static void MX_GPIO_Init(void);
19 static void MX_TIM22_Init(void);
20 static void MX_ADC_Init(void);
21 static void MX_I2C1_Init(void);
22 static void MX_USART2_UART_Init(void);
23
24 int main(void)
25 {
26
27     float raw;
28     char msg2[25];
29     uint8_t storage01;
30     HAL_StatusTypeDef ret;
31
32     //Reset aller Ausgänge
33     HAL_Init();
34
35     //Konfiguration des System Clocks
36     SystemClock_Config();
37
38     //Initialisierung der verwendeten Schnittstelle
39     MX_GPIO_Init();
40     MX_TIM22_Init();
41     MX_ADC_Init();
42     MX_I2C1_Init();
43     MX_USART2_UART_Init();
```

```

44 //Starten des PWM Signals
45 HAL_TIM_PWM_Start(&htim22, TIM_CHANNEL_1);
46
47 const uint8_t ds2777_address = 0x59;
48 const uint8_t spannung_zelle1 = 0x0c;
49 const uint8_t spannung_zelle2 = 0x1c;
50 const uint8_t strom = 0x0e;
51 const uint8_t temp = 0x0a;
52 const uint8_t raac = 0x02; //active absolute
53 const uint8_t rsac = 0x04;
54 const uint8_t rarc = 0x06;
55 const uint8_t rsrc = 0x07;
56
57 while (1)
58 {
59
60 //-----ADC-und-PWM-Teil-----
61 HAL_ADC_Start(&hadc); // ADC starten
62 HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY); // einmal abtasten
63 raw = HAL_ADC_GetValue(&hadc); // wert in raw speichern
64 //umwandeln dass man prozentualen anteil hat; 4095 weil 12 bit adc
65 raw = (raw/4095.0);
66 raw = raw*2000;
67 //Wert in das CCR schreiben
68 TIM22->CCR1 = raw;
69
70 //-----I2C-und-UART-Teil-----
71 // Fuer die beiden Funktionen HAL_I2C_Mem_Read und HAL_UART_Transmit
72 //wird eine Funktion geschrieben nach der Bachelorarbeit
73 //damit diese nicht so oft wiederholt werden muessen
74 //Register ueber I2C auslesen
75 ret = HAL_I2C_Mem_Read(&hi2c1, (ds2777_address<<1) | 0x01,
76 spannung_zelle1, 2, &storage01, sizeof(storage01), HAL_MAX_DELAY);
77 // kann auf Fehlschlag ueberprueft werden
78 if(ret != HAL_OK)
79 {
80     Error_Handler();
81 }
82 //Inhalt in einen String umwandeln und formatieren
83 sprintf(msg2, "Zellspannung 1: %X\r\n", storage01);
84 //String uebertragen
85 HAL_UART_Transmit(&huart2, msg2, strlen(msg2), HAL_MAX_DELAY);
86
87 //Das gleiche fuer die restlichen Register
88 HAL_I2C_Mem_Read(&hi2c1, (ds2777_address<<1) | 0x01, spannung_zelle2,
89 2, &storage01, sizeof(storage01), HAL_MAX_DELAY);
90 sprintf(msg2, "Zellspannung 2: %X\r\n", storage01);
91 HAL_UART_Transmit(&huart2, msg2, strlen(msg2), HAL_MAX_DELAY);
92
93 HAL_I2C_Mem_Read(&hi2c1, (ds2777_address<<1) | 0x01, strom, 2, &
94 storage01, sizeof(storage01), HAL_MAX_DELAY);

```

```

93     sprintf(msg2, "Strom: %X\r\n", storage01);
94     HAL_UART_Transmit(&huart2, msg2, strlen(msg2), HAL_MAX_DELAY);
95
96     HAL_I2C_Mem_Read(&hi2c1, (ds2777_address<<1) | 0x01, temp, 2, &
97     storage01, sizeof(storage01), HAL_MAX_DELAY);
98     sprintf(msg2, "Temperatur: %X\r\n", storage01);
99     HAL_UART_Transmit(&huart2, msg2, strlen(msg2), HAL_MAX_DELAY);
100
101    HAL_I2C_Mem_Read(&hi2c1, (ds2777_address<<1) | 0x01, raac, 2, &
102     storage01, sizeof(storage01), HAL_MAX_DELAY);
103    sprintf(msg2, "Ladezustand: %X\r\n", storage01);
104    HAL_UART_Transmit(&huart2, msg2, strlen(msg2), HAL_MAX_DELAY);
105
106 }
107
108 void SystemClock_Config(void)
109 {
110     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
111     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
112     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
113
114     //Configure the main internal regulator output voltage
115     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
116     //Initializes the RCC Oscillators according to the specified parameters
117     //in the RCC_OscInitTypeDef structure
118     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
119     RCC_OscInitStruct.MSISState = RCC_MSIS_ON;
120     RCC_OscInitStruct.MSICalibrationValue = 0;
121     RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_4;
122     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
123     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
124     {
125         Error_Handler();
126     }
127     //Initializes the CPU, AHB and APB buses clocks
128     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
129                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
130     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
131     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
132     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
133     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
134
135     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
136     {
137         Error_Handler();
138     }
139     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|
140                                         RCC_PERIPHCLK_I2C1;
141     PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_SYSCLK;
142     PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_SYSCLK;

```

```

142     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
143     {
144         Error_Handler();
145     }
146 }
147
148 //@brief ADC Initialization Function
149 static void MX_ADC_Init(void)
150 {
151     ADC_ChannelConfTypeDef sConfig = {0};
152     //Configure the global features of the ADC (Clock, Resolution, Data
153     //Alignment and number of conversion)
154     hadc.Instance = ADC1;
155     hadc.Init.OversamplingMode = DISABLE;
156     hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV1;
157     hadc.Init.Resolution = ADC_RESOLUTION_12B;
158     hadc.Init.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
159     hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
160     hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
161     hadc.Init.ContinuousConvMode = DISABLE;
162     hadc.Init.DiscontinuousConvMode = DISABLE;
163     hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
164     hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
165     hadc.Init.DMAContinuousRequests = DISABLE;
166     hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
167     hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
168     hadc.Init.LowPowerAutoWait = DISABLE;
169     hadc.Init.LowPowerFrequencyMode = ENABLE;
170     hadc.Init.LowPowerAutoPowerOff = DISABLE;
171     if (HAL_ADC_Init(&hadc) != HAL_OK)
172     {
173         Error_Handler();
174     }
175     //Configure for the selected ADC regular channel to be converted.
176     sConfig.Channel = ADC_CHANNEL_5;
177     sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
178     if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
179     {
180         Error_Handler();
181     }
182 }
183
184 //@brief I2C1 Initialization Function
185 static void MX_I2C1_Init(void)
186 {
187     hi2c1.Instance = I2C1;
188     hi2c1.Init.Timing = 0x000000202;
189     hi2c1.Init.OwnAddress1 = 20;
190     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
191     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
192     hi2c1.Init.OwnAddress2 = 0;
193     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;

```

```

193     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_ENABLE;
194     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
195     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
196     {
197         Error_Handler();
198     }
199 // Configure Analogue filter
200     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) !=
201         HAL_OK)
202     {
203         Error_Handler();
204     }
205 // Configure Digital filter
206     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
207     {
208         Error_Handler();
209     }
210
211 // @brief TIM22 Initialization Function
212 static void MX_TIM22_Init(void)
213 {
214     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
215     TIM_SlaveConfigTypeDef sSlaveConfig = {0};
216     TIM_MasterConfigTypeDef sMasterConfig = {0};
217     TIM_OC_InitTypeDef sConfigOC = {0};
218
219     htim22.Instance = TIM22;
220     htim22.Init.Prescaler = 0;
221     htim22.Init.CounterMode = TIM_COUNTERMODE_UP;
222     htim22.Init.Period = 2100;
223     htim22.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
224     htim22.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
225     if (HAL_TIM_Base_Init(&htim22) != HAL_OK)
226     {
227         Error_Handler();
228     }
229     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
230     if (HAL_TIM_ConfigClockSource(&htim22, &sClockSourceConfig) != HAL_OK)
231     {
232         Error_Handler();
233     }
234     if (HAL_TIM_PWM_Init(&htim22) != HAL_OK)
235     {
236         Error_Handler();
237     }
238     sSlaveConfig.SlaveMode = TIM_SLAVEMODE_DISABLE;
239     sSlaveConfig.InputTrigger = TIM_TS_ITR0;
240     if (HAL_TIM_SlaveConfigSynchro(&htim22, &sSlaveConfig) != HAL_OK)
241     {
242         Error_Handler();
243     }

```

```

244     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
245     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
246     if (HAL_TIMEx_MasterConfigSynchronization(&htim22, &sMasterConfig) !=
247         HAL_OK)
248     {
249         Error_Handler();
250     }
251     sConfigOC.OCMode = TIM_OCMODE_PWM1;
252     sConfigOC.Pulse = 1050;
253     sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
254     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
255     if (HAL_TIM_PWM_ConfigChannel(&htim22, &sConfigOC, TIM_CHANNEL_1) !=
256         HAL_OK)
257     {
258         Error_Handler();
259     }
260     HAL_TIM_MspPostInit(&htim22);
261 }
262 // @brief USART2 Initialization Function
263 static void MX_USART2_UART_Init(void)
264 {
265     huart2.Instance = USART2;
266     huart2.Init.BaudRate = 57600;
267     huart2.Init.WordLength = UART_WORDLENGTH_8B;
268     huart2.Init.StopBits = UART_STOPBITS_1;
269     huart2.Init.Parity = UART_PARITY_NONE;
270     huart2.Init.Mode = UART_MODE_TX_RX;
271     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
272     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
273     huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
274     huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
275     if (HAL_UART_Init(&huart2) != HAL_OK)
276     {
277         Error_Handler();
278     }
279 }
280 // @brief GPIO Initialization Function
281 static void MX_GPIO_Init(void)
282 {
283     GPIO_InitTypeDef GPIO_InitStruct = {0};
284
285     /* GPIO Ports Clock Enable */
286     __HAL_RCC_GPIOC_CLK_ENABLE();
287     __HAL_RCC_GPIOA_CLK_ENABLE();
288     __HAL_RCC_GPIOB_CLK_ENABLE();
289
290     /*Configure GPIO pin Output Level */
291     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);
292
293 }
```

```

294 /*Configure GPIO pin Output Level */
295 HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);
296
297 /*Configure GPIO pin : PA8 */
298 GPIO_InitStruct.Pin = GPIO_PIN_8;
299 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
300 GPIO_InitStruct.Pull = GPIO_NOPULL;
301 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
302
303 /*Configure GPIO pin : PA11 */
304 GPIO_InitStruct.Pin = GPIO_PIN_11;
305 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
306 GPIO_InitStruct.Pull = GPIO_NOPULL;
307 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
308 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
309
310 /*Configure GPIO pin : LD3_Pin */
311 GPIO_InitStruct.Pin = LD3_Pin;
312 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
313 GPIO_InitStruct.Pull = GPIO_NOPULL;
314 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
315 HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);
316 }
317
318 // @brief This function is executed in case of error occurrence.
319 void Error_Handler(void)
320 {
321     // Diese Funktion wird im Fehlerfall aufgerufen und kann erweitert werden
322 }
323
324 #ifdef USE_FULL_ASSERT
325 /**
326 * @brief Reports the name of the source file and the source line number
327 */
328 void assert_failed(uint8_t *file, uint32_t line)
329 {
330     /* User can add his own implementation to report the file name and line
331      number,
332      tex: printf("Wrong parameters value: file %s on line %d\r\n", file,
333      line) */
334 }
335 #endif

```