Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Henrik Klockmann

Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Henrik Klockmann

Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 27. September 2013

Henrik Klockmann

Thema der Bachelorthesis

Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern

Stichworte

Lithium-Eisenphosphat-Akkumulator, optische Sensorik, Lichtwellenleiter, Lichtsensoren, Mikrocontroller

Kurzzusammenfassung

In dieser Arbeit wurden Voruntersuchungen bezüglich optischer Sensorik innerhalb von Lithiumbatterien durchgeführt. Dafür fanden erste Spektralanalysen an Lithium-Eisenphosphat statt, welche das optische Verhalten bei unterschiedlicher Lithinierung zeigen. Zur Messung der optischen Spektraleigenschaften wurde eine Messplatine entworfen, welche eine Grundlage für weiterführende Untersuchungen darstellen soll.

Henrik Klockmann

Title of the paper

Optical spectral analysis of lithium battery electrodes with optical fibers

Keywords

Lithium iron phosphate battery, optical sensor technology, optical fibers, light sensors, microcontroller

Abstract

This thesis describes preliminary investigations about the use of optical sensors inside of lithium batteries. First spectral analysis of lithium iron phosphate were accomplished for determining its optical behavior. It was possible to show changes of the spectral properties of lithium iron phosphate through the intercalation of lithium. A PCB-Layout was designed for measuring this optical behavior. Based on this layout, further investigations can be performed.

Danksagung

An dieser Stelle möchte ich mich zunächst bei den Mitarbeitern des BATSEN-Projektes bedanken, insbesondere bei Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider und Herrn Prof. Dr.-Ing. Jürgen Vollmer. Als Leiter des Forschungsvorhabens und als meine Erst- und Zweitprüfer ermöglichten sie mir diese Bachelorarbeit. Auch gegenüber den Herren Dipl.-Ing. Günter Müller, Dipl.-Phys. Valentin Roscher und Dipl.-Ing. Matthias Schneider möchte ich aufgrund ihres großartigen Engagements bei Problemen fachlicher und formaler Natur meinen Dank aussprechen.

Gleichermaßen bedanke ich mich bei meinen Eltern Siw und Jürgen Klockmann, welche mich in all den Jahren meines Studiums moralisch und auch finanziell erheblich unterstützt haben.

Der letzte Dank gehört meiner Freundin Janina Lentföhr für ihren außerordentlichen Rückhalt, speziell in den letzten Wochen dieser Arbeit.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einführung

In der heutigen Zeit gewinnen Batterien, zunehmend auch in Fahrzeugen, an Bedeutung. Besonders bei großen und teuren Akkupacks, welche aus vielen zusammengeschalteten Zellen bestehen, ist es sinnvoll, diese zu überwachen. Bei vielen Lithium-Ionen-Akkumulatoren ist es sogar aus Sicherheitsgründen zwingend erforderlich, um diese stabil betreiben zu können. Durch die Überwachung lassen sich die Zuverlässigkeit und Lebensdauer und somit auch die Wirtschaftlichkeit von Akkus erheblich steigern. Zum Batteriemanagement können, je nach Zelltyp, folgende Methoden zählen [1]:

- Messdatenerfassung
- Sicherheitsmanagement/Überwachung
- Überwachung wichtiger Größen, wie z.B. Ladungsumsatz und Zyklenzahl
- Batteriezustandsbestimmung (Lade- und Alterungszustand)
- Ladealgorithmus
- Entladeüberwachung
- Thermisches Management
- Kommunikation mit übergeordneten Systemen

Im Allgemeinen werden Größen wie Spannung, Strom und Temperatur erfasst. Diese Messungen werden herangezogen, um mithilfe von statistischen Grundlagen unter anderem auf den Ladezustand (SOC = state of charge) und Alterungszustand (SOH = state of health) einer Zelle oder Batterie zu schließen. Auf Grundlage dieser Messungen und Berechnungen kann dann in Lade- und Entladevorgänge eingegriffen werden, um zum Beispiel Überlastungen und Beschädigungen der Batterie zu vermeiden.

1.1. BATSEN-Projektbeschreibung

In dem Projekt "BATSEN – Drahtlose Sensoren für Fahrzeug-Batterien" wird die Möglichkeit des Einsatzes von Funksensoren innerhalb von Batterien untersucht. Diese Sensoren sollen an jeder einzelnen Zelle eines Batteriepacks angebracht werden und dessen Spannung und Temperatur aufnehmen. Durch den Einsatz von drahtloser Kommunikation sollen negative Eigenschaften, welche verdrahtete Lösungen mit sich bringen, vermieden werden. Eine Gegenüberstellung von verdrahteter und drahtloser Lösung ist in der Tabelle 1.1 aus einer Veröffentlichung des BATSEN-Projektes dargestellt.

	Verdrahtet	Drahtlos
Aufwändige Messkabel und Stecker	- -	+
Kompatible Konstruktion	-	+
Robustheit	-	++
Potentialtrennung	- -	+++
Modularität für unterschiedliche Zellenanzahl	-	++
Chemisch beständige Werkstoffe/Kapselung	-	-
Synchronisation zur Bordnetzfunktion	+	?
Effektoreinbindung (Zellenausgleich)	+	?
Kosten	-	+

Tabelle 1.1.: Vergleich von verdrahteter und drahtloser Zellenüberwachung [2]

Der Aufbau der im Rahmen des BATSEN-Projektes entstehenden Zellsensoren lässt sich anhand ihrer Kommunikationskanäle in drei Funktionstypen untergliedern. So haben Sensoren der Klasse 1 nur einen Uplink-Kanal, können also nur Messwerte an eine Steuereinheit senden, ohne von dieser gesteuert zu werden. Klasse 2 und Klasse 3 Sensoren haben Up- und Downlink-Kanäle, unterscheiden sich aber in ihrem Kommunikationsumfang voneinander.

Zusätzlich wird im Zuge des Projektes an optischen Sensoren gearbeitet, mit welchen Informationen über sich ändernde Parameter im Inneren einer Batterie gewonnen werden.

1.2. Motivation

Die Bestimmung von Zuständen einer Batterie basiert auf der statistischen Auswertung von gemessenen Größen. SOC und SOH werden dann anhand dieser Messungen mithilfe eines Modells der Batterie ermittelt.
Zur genaueren Bestimmung der Zustände einer Batterie wäre es erstrebenswert, weitere Stützstellen mit weiteren Informationen über die Zellen heranzuziehen.

Eine Möglichkeit wäre die optische Vermessung von Elektrolyt oder Elektroden. Harris *et al.* haben bereits optische Veränderungen an der Graphitseite einer Lithium-Ionen-Zelle nachweisen können [3]. In Abbildung 1.1 sind die Auswirkungen der Einlagerung von Lithium-Ionen in die Graphitelektrode gezeigt.

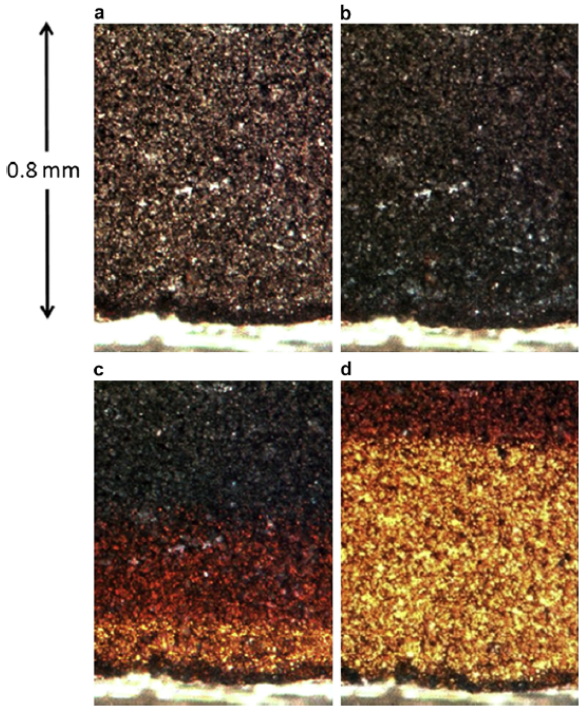

Abbildung 1.1.: Lithiumeinlagerung an einer Graphit-Anode bei vier Ladezuständen zu unterschiedlichen Zeitpunkten. Die Bilder zeigen die optischen Auswirkungen auf eine geladene Graphit-Anode. Teilbild a stellt den Ausgangszustand dar. Mit einem konstantem Strom von 150 μA geladen, wurden weitere Bilder nach sechs (b), neun (c) und 13 (d) Stunden aufgenommen [3].

Das Teilbild a zeigt einen Ausschnitt einer aus Graphit bestehenden Elektrode einer Lithium-Ionen-Zelle ohne Einlagerung von Lithium-Atomen. Die Bilder b, c und d zeigen denselben Ausschnitt der Elektrode der konstant aufgeladenen Zelle nach ca. sechs, neun und 13 Stunden. Das Laden der Zelle verursacht die Einlagerung (Interkalation) von Lithium-Atomen zwischen den Graphitschichten.
Die zunehmende Einlagerung führt zu einer Verfärbung des Materials, welche in den Bildern deutlich erkennbar ist.
Könnte man nun eine solche Verfärbung messen, würden die erhobenen Daten sich gut eignen, die zuvor gewünschten zusätzlichen Informationen zur Bestimmung des Lade- und Alterungszustandes einer Zelle zu liefern.

1.3. Aufgabe

Aufgabe dieser Bachelorarbeit ist die experimentelle Voruntersuchung für ein optisches Messverfahren für Kathoden von Lithium-Eisenphosphat-Akkumulatoren.
Dafür ist ein kommerzielles Spektrometer in Betrieb zu nehmen. Mit diesem sollen grundlegende Messungen durchgeführt werden, welche insbesondere die Absorptionseigenschaften von Lithium-Eisenphosphat bestimmen sollen.
Für die optische Sensorik sollen Lichtwellenleiter zum Einsatz kommen. Diese sind hinsichtlich ihres vorgesehenen Einsatzes innerhalb eines Lithium-Ionen-Akkus zu testen und entsprechende optische Messmethoden müssen überprüft werden.
Unter Berücksichtigung der erlangten Informationen ist anschließend ein Laboraufbau eines vereinfachten LED[1]-Spektrometers als Grundlage für Messungen an verschieden lithinierten Eisenphosphatproben zu realisieren.
Vor dem Entwurf einer entsprechenden Platine sind, neben Lichtleitern und einem geeigneten Steckersystem, insbesondere verschiedene LEDs und Lichtsensoren zu erproben und auszuwählen.
Die entstandene Platine und das verwendete Messkonzept sollen dann anhand von unterschiedlichen Probepräparaten und Lithium-Eisenphosphatproben geprüft und bewertet werden.

[1] Light-emitting diode (Leuchtdiode)

2. Grundlagen

2.1. Optik

Sichtbares Licht besteht aus elektromagnetischen Wellen. Darüber hinaus existieren aber auch für das menschliche Auge nicht sichtbare elektromagnetische Wellen. Ob diese sichtbar sind oder nicht, hängt von der Wellenlänge beziehungsweise von der Frequenz ab, mit welcher sich diese Strahlung ausbreitet. Das Spektrum von elektromagnetischen Wellen geht dabei von wenigen Pikometern ($10^{-12}\,m$) bis zu einigen Kilometern ($10^{3}\,m$).

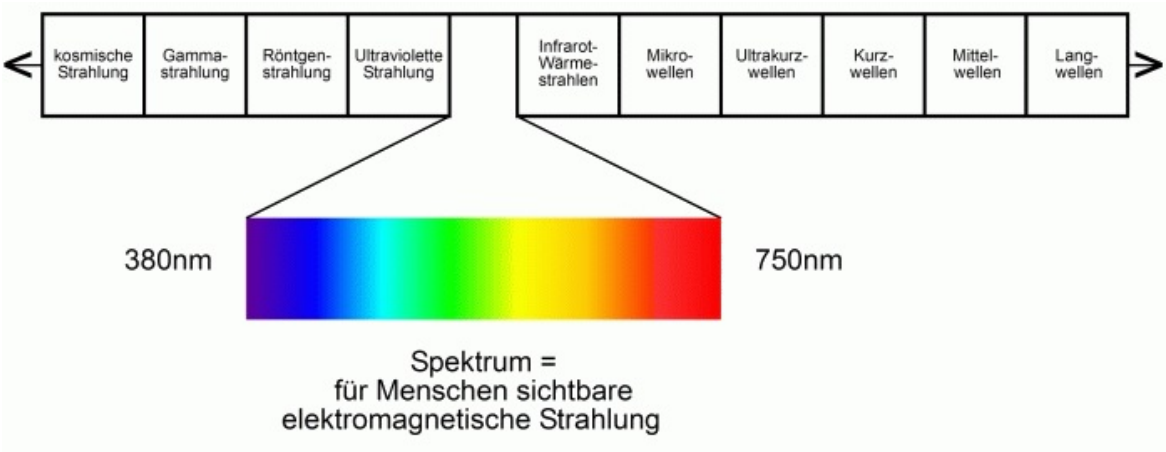

Abbildung 2.1.: Spektrum elektromagnetischer Wellen [4]

Wie in Abbildung 2.1 dargestellt, liegt die für einen Menschen sichtbare Strahlung bei Wellenlängen von 380 nm, welche als Blau wahrgenommen wird, bis 750 nm, welche vom Auge als Rot interpretiert wird. Auch zur optischen Strahlung gehören die unterhalb der 380 nm auftretende ultraviolette Strahlung sowie die oberhalb der 750 nm auftretende infrarote Strahlung. Diese sind für das menschliche Auge zwar nicht sichtbar, unterliegen aber denselben physikalischen Gesetzen wie die sichtbare Strahlung [5].

2.1.1. Elektromagnetische Wellen

Wie in Abbildung 2.2 dargestellt, besteht eine elektromagnetische Welle aus der Kopplung zweier Transversalwellen[1]. Es handelt sich bei den harmonischen Schwingungen dabei um ein magnetisches und ein elektrisches Feld, welche gleichphasig, aber senkrecht zueinander auftreten.

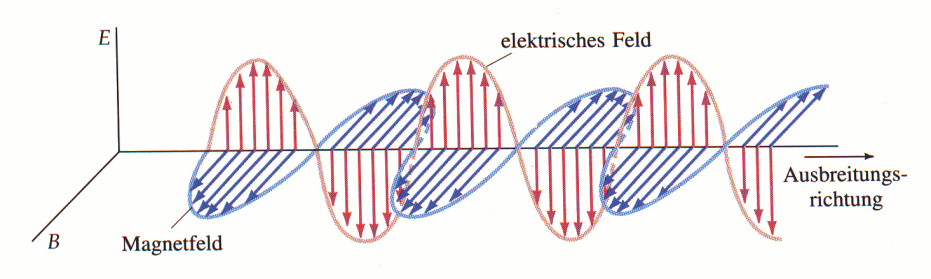

Abbildung 2.2.: Elektromagnetische Welle [6]

Die theoretischen Grundlagen dafür ergeben sich aus den Maxwell-Gleichungen.

Die Ausbreitungsgeschwindigkeit einer elektromagnetischen Welle im Vakuum entspricht der Lichtgeschwindigkeit im Vakuum und beträgt:

$$c_0 = \frac{1}{\sqrt{\epsilon_0 \mu_0}} = 299792458 \frac{m}{s} \tag{2.1}$$

Bei ϵ_0[2] und μ_0[3] handelt es sich um Naturkonstanten.

Elektromagnetische Wellen unterscheiden sich in ihrer Frequenz, welche sich in verschiedenen Übertragungsmedien nicht ändert. Je nach Medium kann sich aber die Ausbreitungsgeschwindigkeit und dementsprechend auch die Wellenlänge ändern. Die Wellenlänge (λ) ist gleich dem Quotienten aus der Ausbreitungsgeschwindigkeit (c) und der Frequenz (f):

$$\lambda = \frac{c}{f} \tag{2.2}$$

Im Allgemeinen wird jedoch die Wellenlänge im Vakuum angegeben [8].

[1] Die Schwingung einer Transversalwelle findet senkrecht zu ihrer Ausbreitungsrichtung statt.
[2] Permittivität - Durchlässigkeit von Materialien für elektrische Felder [7]
[3] Permeabilität - magnetische Leitfähigkeit von Materialien [7]

2.1.2. Reflexion und Brechung

Eine Eigenschaft von Licht ist dessen unterschiedliche Ausbreitungsgeschwindigkeit in verschiedenen Medien. Für die Ausbreitungsgeschwindigkeit gilt:

$$c = \frac{c_0}{n} \tag{2.3}$$

Dabei entspricht c_0 der zuvor erwähnten Lichtgeschwindigkeit im Vakuum und n dem sogenannten Brechungsindex oder Brechzahl, welche die optische Materialeigenschaft des Mediums beschreibt. Der Brechungsindex wird mit der folgenden Gleichung bestimmt:

$$n = \sqrt{\epsilon_r \cdot \mu_r} \tag{2.4}$$

ϵ_r gibt die relative Permittivität, μ_r die relative Permeabilität der Materie an.
Ein Medium mit einem zu seinem Vergleichsmedium relativ höheren Brechungsindex, wird dabei als optisch dichter bezeichnet. Umgekehrt nennt man es optisch dünner, wenn der Brechungsindex geringer ist.

Trifft ein Lichtstrahl auf die Grenzfläche zweier Medien mit unterschiedlichen Brechungsindizes wird das Licht teilweise gebrochen und teilweise reflektiert. In Abbildung 2.3 ist ein solcher einfallender Strahl dargestellt. Der Winkel zwischen dem Lichtstrahl und dem Lot senkrecht zur Grenzfläche wird als Einfallswinkel bezeichnet. Bei einer Reflexion gilt, dass der Einfallswinkel α des Lichtstrahls gleich dem Ausfallswinkel α' ist.

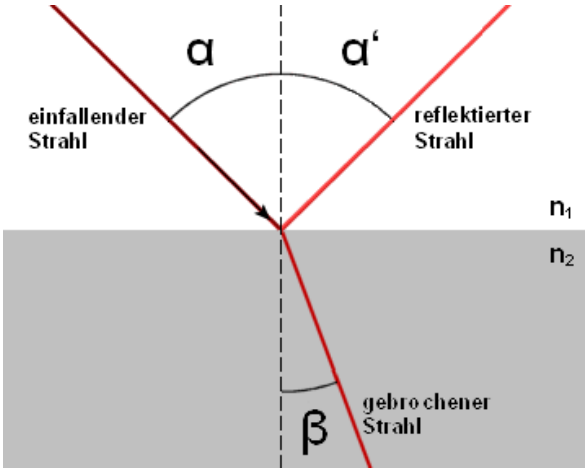

Abbildung 2.3.: Brechung und Reflexion eines Lichtstrahls [9]

Die Brechung des Lichtstrahls ist abhängig von den Brechungsindizes beider Medien. Die Brechzahl von dem ersten Medium wird durch n_1 und vom zweiten durch n_2 beschrieben. Der

Grund für die Brechung ist die zuvor erwähnte Änderung der Ausbreitungsgeschwindigkeit der Lichtwellen innerhalb der Medien.

$$\frac{sin\alpha}{sin\beta} = \frac{c_1}{c_2} = \frac{n_2}{n_1} \tag{2.5}$$

Trifft ein Lichtstrahl von einem optisch dichteren Medium auf ein optisch dünneres Medium, kann es zu einer Totalreflexion kommen. Dann wird das Licht vollständig reflektiert und nicht mehr in das optisch dünnere Medium hinein gebrochen. Die Bedingung dafür ist, dass der Lichtstrahl den folgenden Grenzwinkel α_G beim Einfall auf die Grenzschicht beider Medien überschreitet:

$$sin\alpha_G = \frac{c_1}{c_2} = \frac{n_2}{n_1} \tag{2.6}$$

Aus Gleichung 2.6 folgt als weitere Bedingung, dass die Brechzahl n_1 größer als n_2 sein muss, damit der Grenzwinkel kleiner als $90°$ wird. [10].

2.1.3. Lichtwellenleiter

Zur Leitung von Licht werden Lichtwellenleiter verwendet. Entsprechend der Abbildung 2.4 bezeichnet man das Innere eines Lichtwellenleiters als Kern (engl. core). Den Kern umgibt ein Mantel (engl. cladding). Zum Schutz der Faser sind Kern und Mantel zusätzlich noch von einer äußeren Umschichtung (engl. coating) umhüllt.

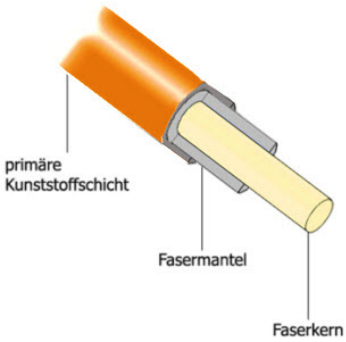

Abbildung 2.4.: Querschnitt eines Lichtwellenleiters [11]

Das Prinzip der Leitung des Lichtes beruht auf der in Abschnitt 2.1.2 beschriebenen Totalreflexion. Dafür muss der Brechungsindex vom Kern höher sein, als der des Mantels.

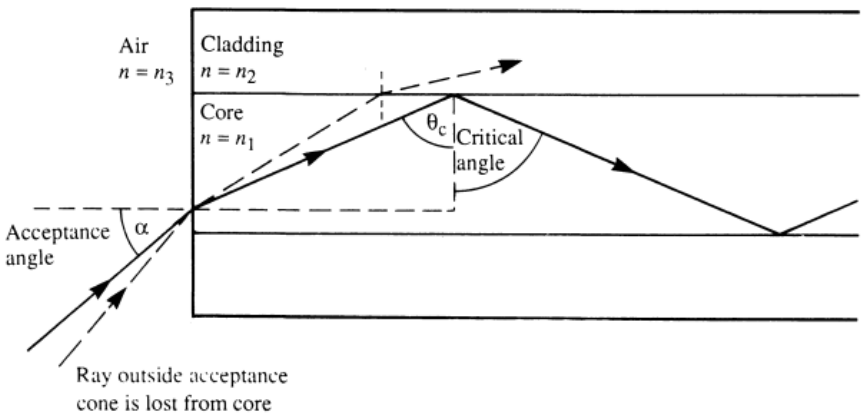

Abbildung 2.5.: Lichteintritt in einen Lichtwellenleiter. Eingezeichnet ist ein Lichtstrahl, welcher den maximal möglichen Eintrittswinkel zeigt. Der weitere Lichtstrahl außerhalb des Akzeptanzkegels wird von der Faser nicht geleitet. Bild nach Quelle [12].

Aus Abbildung 2.5 wird ersichtlich, dass bei Einkopplung von Licht in die Faser nicht alle einfallenden Lichtstrahlen aus der Luft (Brechzahl n_3) die Bedingung der Totalreflexion nach Gleichung 2.6 erfüllen können. Oberhalb des entsprechenden Akzeptanzwinkels gelangen die Lichtstrahlen nach Eintritt in den Faserkern (Brechzahl n_1) direkt in den Fasermantel (Brechzahl n_2) und gehen dort für die Lichtleitung verloren.

Die verwendeten Materialien bestehen im Allgemeinen aus elektrisch nicht oder schwach leitenden Werkstoffen. Das ist zum Beispiel von wesentlichem Vorteil für die Verwendung im Inneren einer Batterie.

Bei dem in Abbildung 2.5 dargestellten Verlauf des Lichtstrahls handelt es sich um einen Stufenindex-Lichtwellenleiter. Das bedeutet, dass sich Brechindizes zwischen Kern und Mantel sprunghaft ändern. Es gibt aber auch andere Brechzahlverläufe, wie beispielsweise bei einem Gradientenindex-Lichtwellenleiter. Bei einem solchen Lichtleiter verringert sich der Brechungsindex des Kerns von der Mitte hin zum Rand kontinuierlich. Die Brechzahl des Mantels ist, wie auch bei dem Stufenindex-Lichtwellenleiter, konstant. Den Gradienten-Lichtwellenleiter zeichnen bessere Übertragungseigenschaften des Lichtes aus, was unter anderem damit zusammenhängt, dass er besser für die später in Kapitel 2.1.5 erwähnte vielmodige Übertragung des Lichtes geeignet ist [8].

Zur Herstellung von Lichtleitern bieten sich verschiedene Materialien an. Übliche Kombinationen und ihre Bezeichnungen können der Tabelle 2.1 entnommen werden.

Bei einer Glasfaser beispielsweise bestehen Kern und Mantel aus Quarzglas. Um unterschiedliche Brechindizes zu erhalten, kann unter anderem das Quarzglas dotiert werden,

Namenskürzel	Kernmaterial	Mantelmaterial
Glasfaser	Quarzglas	Quarzglas
PCS-Faser (plastic cladded silica)	Quarzglas	Silikonharz
HCS-Faser (hard cladded silica)	Quarzglas	Hartpolymer
POF-Faser (polymer optical fiber)	Plexiglas(PMMA)	Fluoriniertes Polymer

Tabelle 2.1.: Lichtwellenleiter Materialzusammensetzung [8]

damit das Licht durch die Faser hindurch geleitet werden kann. Die äußere Beschichtung (coating) der Glasfaser ist sehr wichtig für dessen Stabilität, da sie ohne sehr brüchig ist und eine geringe Biegsamkeit aufweist.

Im Gegensatz zu den Glasfasern besteht eine POF-Faser im Kern und Mantel aus Kunststoff, zumeist PMMA (Plexiglas). Auch ohne die äußere Beschichtung ist die Faser flexibel und stabil. Dies führt zu Vorteilen gegenüber der Glasfaser in Einsatzgebieten, bei welcher diese äußere Schutzschicht entfernt wird, wie beispielsweise die später die in Abschnitt 2.3 ausgeführte Verwendung als Sensor. Des Weiteren sind POF-Fasern kostengünstig und kommen daher häufig in Heim-Audioanwendungen zum Einsatz.

Abhängig von verwendeten Kern- und Mantelmaterialien treten Dämpfungen bei der Lichtleitung in den Fasern auf. Bei einer Glasfaser beispielsweise werden charakteristische Absorptionen durch OH^--Verunreinigungen verursacht, welche auf den Herstellungsprozess der Faser zurückzuführen sind. Die OH-Ionen absorbieren Licht unter anderem bei einer passenden Resonanzenergie von 0,45 eV und deren Vielfachen. Die Energie des Lichtes und somit die entsprechende Wellenlänge, lässt sich über die folgende Gleichung berechnen:

$$E = hf = \frac{hc}{\lambda} \tag{2.7}$$

Der Teilchencharakter des Lichtes, welcher für diese Energie verantwortlich ist, wird im noch folgenden Kapitel 2.1.7 erläutert.
Demnach wird Licht in einer Glasfaser ungefähr bei folgenden Wellenlängen absorbiert: 0,45 eV $\widehat{=}$ 2750 nm, 0,90 eV $\widehat{=}$ 1380 nm, sowie 1,35 eV $\widehat{=}$ 920 nm.
Außerdem sind die OH-Ionen mitverantwortlich für eine Lichtabsorption bei 1,00 eV $\widehat{=}$ 1240 nm [8].

Absorptionen im Infrarotbereich treten aufgrund von Resonanzabsorptionen des Quarzglases in Kern und Mantel auf.

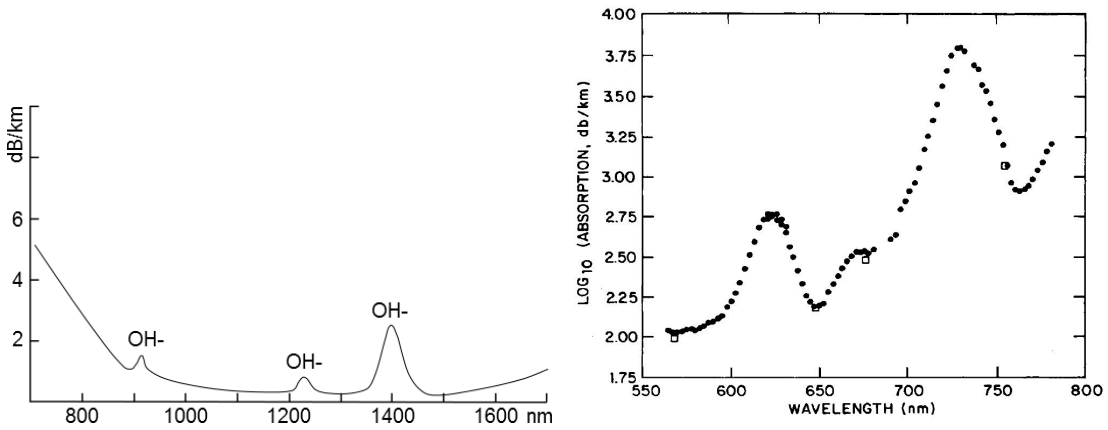

Abbildung 2.6.: Links: Dämpfungsverlauf einer Glasfaser [13]. Rechts: Dämpfungsverlauf einer POF-Faser [14].

Bei niedrigeren Wellenlängen ist die Rayleigh-Streuung[4] hauptverantwortlich für Dämpfungen. Abbildung 2.6 zeigt beispielhaft die Dämpfung einer Glasfaser bei unterschiedlichen Wellenlängen.

Bei POF-Fasern wirkt, wie auch bei den Glasfasern, die Rayleigh-Streuung. Weitere Dämpfungen entstehen durch Resonanzabsorptionen von Schwingungen der Kohlenwasserstoffverbindungen, welche in der beispielhaften Abbildung 2.6 an den Erhebungen deutlich zu erkennen sind.

Im Vergleich zu Glasfasern treten in POF-Fasern, wie auch in den jeweiligen Abbildungen deutlich wird, weitaus höhere Dämpfungen auf. Dadurch beschränken sich die Einsatzgebiete von POF-Fasern auf kürzere Übertragungsstrecken.

[4] Lichtstreuung an Teilchen, deren geometrische Abmessungen gegen die Lichtwellenlänge sehr klein sind [8].

2.1.4. Beugung von Lichtwellenleitern

Wird ein Lichtwellenleiter gebogen, so kann sich dadurch das Reflexionsverhalten von Lichtstrahlen ändern.

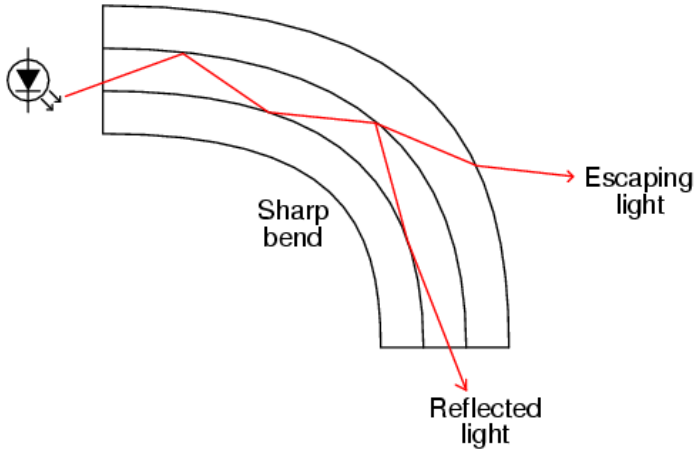

Abbildung 2.7.: Lichtstrahl im gebogenen Lichtleiter [15]

Wie in Abbildung 2.7 dargestellt, kann es bei stark gebogenen Lichtwellenleitern dazu kommen, dass sich der Winkel ändert, in welchem ein Lichtstrahl auf die Grenzfläche zwischen Kern und Mantel trifft. Bei Änderung des Einfallswinkels kann es geschehen, dass die Bedingung für eine Totalreflexion nicht mehr erfüllt wird. Im Zuge dessen wird das Licht nur noch teilweise reflektiert und teilweise in den Mantel hinein gebrochen und von dort in das umgebende Medium gestreut. Das führt zu einer reduzierten Lichtintensität des im Kern verbleibenden Lichtstrahls.

2.1.5. Wellenausbreitung im Lichtleiter, Moden

Bisher wurden in sämtlichen Überlegungen einfache Strahlen verwendet, um das Verhalten von Licht zu erklären. Diese Vereinfachung reicht aber nicht aus, um bestimmte Effekte zu erklären.

Dafür ist es notwendig, den Wellencharakter des Lichtes zu beachten. Bei Licht handelt es sich nicht um einen unendlich schmalen Strahl, sondern um eine breitere Wellenfront.

Bei einer Reflexion im Lichtwellenleiter kommt es nun zu destruktiven und konstruktiven Interferenzen. Dies führt zu einer cosinusförmigen Intensitätsverteilung längs der Faser, welche als stehende Welle bezeichnet wird. Nun wird aber die Wellenfront im Lichtleiter nicht nur einmal, sondern zweimal reflektiert. Bei zweifacher Reflexion kommt es nur dann zu maximalen konstruktiven Interferenzen, wenn die stehenden Wellen genau aufeinander liegen, also phasengleich sind. Eine Wellenlänge gekoppelt mit dessen Eintrittswinkel, welche genau diese Bedingungen erfüllt, wird als Modus bezeichnet.
Je nach Kerndurchmesser kann es bei gegebener Wellenlänge einen oder mehrere Winkel geben, bei welchem sich die Reflexionen nicht auslöschen. Es sind bei gegebenen Wellenlängen also nur bestimmte Einfallswinkel möglich.
Je nach Einfallswinkel, Wellenlänge und Kerndurchmesser können durch die Interferenzen unterschiedliche Anzahlen an Intensitätsmaxima quer des Lichtwellenleiters entstehen. Dabei hat ein Modus von 0, der sogenannte Grundmodus, ein Intensitätsmaximum, der Modus 1 hat zwei Intensitätsmaxima usw.

2.1.6. Evaneszenzfeld

Die Intensitätsmaxima der in Kapitel 2.1.5 erläuterten Moden in Lichtwellenleiten enden nicht direkt an der Grenzfläche zwischen Kern und Mantel. Das Licht hält sich mit exponentiell abklingender (evaneszenter) Intensität auch im Mantel des Lichtwellenleiters auf. Es schließt sich an die cosinusförmige Intensitätsverteilung im Kern an. Zu sehen ist eine solche Verteilung in dem folgenden Abschnitt 2.3.1 in Abbildung 2.11. Dieser exponentiell abklingende Bereich wird als Evaneszenzfeld bezeichnet.

Die Eindringtiefe des evaneszenten Feldes in den Mantel ist umso höher, je geringer der Einfallswinkel des Lichtes bei der Totalreflexion im Lichtleiter ist. Der Einfallswinkel bezieht sich auf den Winkel zwischen dem senkrecht zur Grenzschicht zwischen Mantel und Kern stehenden Lot und dem einfallenden Lichtstrahl. Die Eindringtiefe beträgt dabei Bruchteile einer Wellenlänge bei geringen Winkeln und bis zu einigen Wellenlängen bei großen [8].
Das Licht dringt also je nach Einkopplungswinkel verschieden stark in den Mantel des Lichtleiters ein. Daher muss dieser auch aus stark transparenten Materialien bestehen, damit das Licht nicht vom Mantel absorbiert wird.

2.1.7. Photonen und photoelektrischer Effekt

Können Effekte von Lichtstrahlen wie Interferenzen und Beugung noch mit dessen Wellencharakter erklärt werden, so existieren andere Verhaltensweisen, bei welchen dieser Ansatz nicht zielführend ist.
Darunter fällt beispielsweise der photoelektrische Effekt. Dieser beschreibt, dass durch Licht

bestrahltes Metall Elektronen emittieren kann. Die durch das Licht freigesetzten Elektronen verursachen einen Elektronenstrom, der sich proportional zur Lichtintensität verhält.
Die Emission von Elektronen findet spontan statt. Das bedeutet, dass Elektronen unmittelbar mit der Lichteinstrahlung freigesetzt werden.
Würden die elektromagnetischen Wellen dafür verantwortlich sein, müssten sie die Elektronen solange in Schwingung versetzen, bis sie genügend Energie besitzen, um aus dem Metallgitter austreten zu können. Je nach Lichtintensität müsste es dadurch eine Verzögerung zwischen dem Eintreffen von Licht und dem Herauslösen von Elektronen geben. Ebenfalls würde die Anzahl der herausgelösten Elektronen unabhängig von der Frequenz der elektromagnetischen Welle sein. In Experimenten zeigt sich jedoch, dass der photoelektrische Effekt zu niedrigeren Frequenzen hin eine Grenze besitzt. Ab einer vom Material abhängigen Grenzfrequenz werden keine Elektronen mehr emittiert.
Schon diese beiden Beobachtungen zeigen, dass elektromagnetische Schwingungen als Erklärung für die Freisetzung der Elektronen nicht ausreichen [16] [17].

Zur Erklärung dieses Effektes wird Licht anstelle der elektromagnetischen Welle als Teilchen beschrieben. Die Teilchen werden als Photonen bezeichnet und treten entsprechend der folgenden Gleichung in Energiequanten auf:

$$E = hf = \frac{hc}{\lambda} \tag{2.8}$$

Bei h[5] handelt es sich um das Planck'sche Wirkungsquantum, eine Naturkonstante, bei f um die Frequenz des Lichtes.

Ein Elektron kann durch Licht aus Metall herausgelöst werden, wenn die Energie eines Photons die benötigte Austrittsarbeit (W_A), welche vom jeweiligen Material abhängt, des Elektrons übersteigt:

$$E_{kin} = hf - W_A \tag{2.9}$$

Die Abbildung 2.8 zeigt beispielhaft, wie Elektronen durch Photonen freigesetzt werden. Die Elektronen absorbieren dabei die Energie von jeweils einem Photon. Je höher die Energie des Photons, desto höher auch die kinetische Energie des herausgelösten Elektrons.

[5] $h = 6,62606957 \cdot 10^{-34} Js$

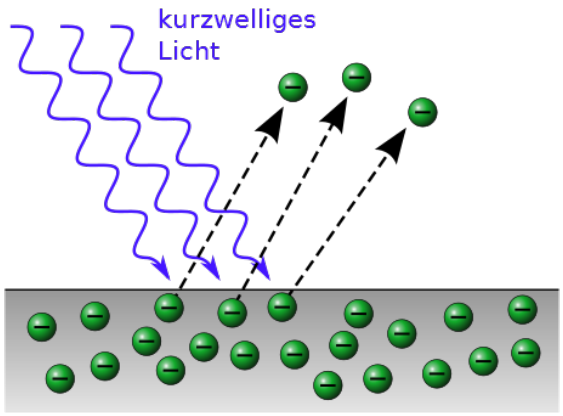

Abbildung 2.8.: Photoelektrischer Effekt [18]

2.2. Lithium-Eisenphosphat-Akkumulator

2.2.1. Galvanische Zelle

Batterien und Akkumulatoren sind galvanische Zellen.
Eine galvanische Zelle besteht im Allgemeinen aus zwei Elektroden und einem Elektrolyt. Die Elektroden werden als Anode und Kathode bezeichnet.
Bei der Anode handelt es sich im Falle von elektrischen Erzeugern um die Elektrode negativer Polarität, bei Verbrauchern um die positive. Dementsprechend ist die Kathode beim Ladevorgang die negative, beim Entladevorgang die positive Elektrode.
Bei Batterien und Akkumulatoren werden Anode und Kathode entsprechend des Falles der Entladung definiert.
Für einen kompakten Aufbau werden Separatoren in die Zellen eingebaut. Diese dienen dazu, Kathode und Anode elektrisch zu trennen und somit Kurzschlüsse zu verhindern. Separatoren müssen jedoch durchlässig für die bei Ladung und Entladung chemisch aktiven Atome sein [1].

Galvanische Zellen werden in Primär- und Sekundärzellen unterteilt. Sie unterscheiden sich in ihrer Fähigkeit, wiederaufladbar (sekundär) oder nicht wiederaufladbar (primär) zu sein. Der Begriff Batterie beschreibt eigentlich eine Primärzelle, Akkumulator eine Sekundärzelle. Im allgemeinen Sprachgebrauch werden jedoch auch Akkumulatoren zunehmend als Batterien bezeichnet.

Beim Laden einer galvanischen Zelle wird elektrische Energie in chemische Energie umgewandelt und gespeichert. Beim Entladen wird die chemisch gespeicherte Energie wieder in

elektrische Energie umgewandelt.
Dieses Funktionsschema wird in Abbildung 2.9 gezeigt.

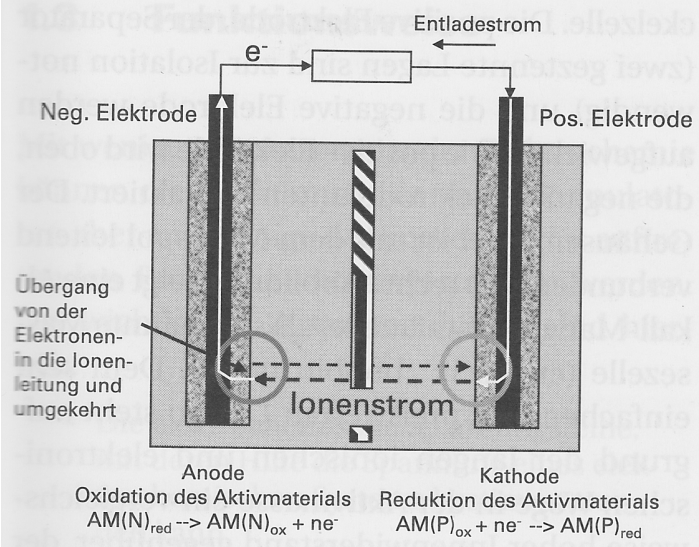

Abbildung 2.9.: Schema einer elektrochemischen Zelle für den Fall der Entladung [1]

2.2.2. Lithium-Ionen-Akkus im Allgemeinen

Lithium besitzt das höchste elektrochemische Standardpotential[6] und ist, unter Standardbedingungen, das leichteste in fester Form vorkommende Element des Periodensystems. Es eignet sich somit hervorragend für die Speicherung von elektrischer Energie.
Lithium gehört zu den Alkalimetallen[7] und ist daher vergleichsweise reaktiv.
Ein großes Problem bei der Verwendung von Lithium ist dessen Reaktionsfreudigkeit in Verbindung mit Wasser. Treffen Lithium-Atome auf Wassermoleküle, so entsteht bei der Reaktion elementarer Wasserstoff. Dieser Vorgang wird durch die folgende Reaktionsgleichung beschrieben [1]:

$$2\mathrm{Li} + 2\mathrm{H_2O} \rightarrow 2\mathrm{LiOH} + \mathrm{H_2} + \text{Wärme} \tag{2.10}$$

Der verwendete Elektrolyt ist in der Regel auch ein entzündliches organisches Lösungsmittel und stellt daher beim Eindringen von Wasser durch die sich entwickelnde Wärme eine potentielle Gefahrenquelle dar.

[6] Potential gegen eine Wasserstoffelektrode bei Normalbedingungen
[7] Alle Elemente außer Wasserstoff aus der ersten Hauptgruppe des Periodensystems werden als Alkalimetalle bezeichnet.

Bei der Reaktion zwischen Wasser und dem im Elektrolyt gelösten Salz entsteht Flusssäure, welche aufgrund seiner Toxizität und starken Ätzwirkung sehr gefährlich ist.

Aufgrund dieser Eigenschaften sind Lithiumbatterien besonders vor dem Eindringen von Feuchtigkeit zu schützen.

In Lithium-Ionen-Akkumulatoren wird die elektrische Energie an der negativen Elektrode in Lithium-Atomen und an der positiven Elektrode meist in Übergangsmetall-Ionen[8] gespeichert. Lithium-Ionen können durch den Elektrolyten hindurch von einer Elektrode zur anderen wandern.

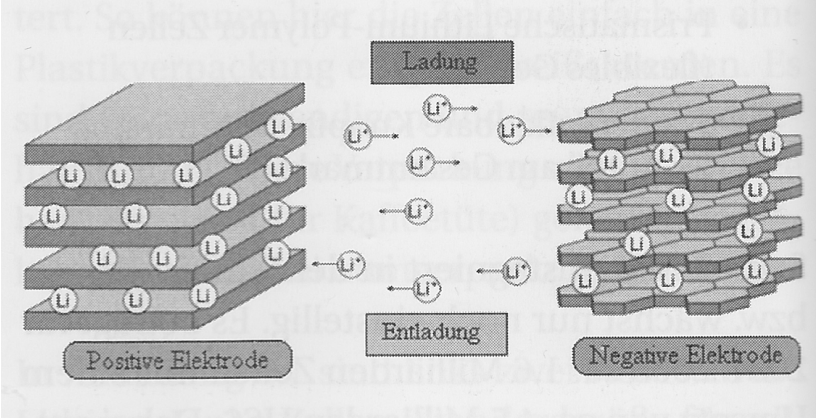

Abbildung 2.10.: Schaubild einer wiederaufladbaren Lithiumzelle [1]

Beim Entladen werden Elektronen von den Lithium-Atomen an der negativen Elektrode abgegeben. Die abgegebenen Elektronen fließen über einen äußeren Stromkreis von der negativen zur positiven Elektrode. Die nun ionisierten Lithium-Atome wandern, wie in Abbildung 2.10 gezeigt, durch den Elektrolyten ebenfalls zur positiven Elektrode, wo sie zwischen den Gittern als Gastatom gespeichert werden. Die Elektronen werden auf der Seite der positiven Elektrode in das beim Ladevorgang zuvor stark ionisierte Kathodenmaterial aufgenommen. Beim Laden des Akkumulators wandern die Lithium-Ionen von der positiven Elektrode durch den Elektrolyten hindurch zur negativen Elektrode. Dort reagieren die Lithium-Ionen mit Elektronen, welche durch einen äußeren Strompfad zugeführt werden, zu ladungsfreien Lithium-Atomen. Sie werden dort zwischen den Schichten der Elektrode, meist Graphit, eingelagert. Dieser Prozess der Einlagerung wird als Interkalation bezeichnet [1].

An den Elektroden werden üblicherweise elektrisch gut leitende Materialien als Stromableiter verwendet. An der negativen Elektrode wird zumeist Kupfer eingesetzt, welches jedoch an der positiven Elektrode stark korrodieren würde. Stattdessen benutzt man dort häufig Aluminium. Aluminium wiederum wird nicht an der negativen Elektrode eingesetzt, da es dort mit

[8] Elemente zwischen der zweiten und dritten Hauptgruppe des Periodensystems.

Lithium reagieren würde.
In den positiven Elektroden werden außerdem häufig noch geringe Mengen an gut leitendem Material hinzugegeben, um dessen Leitfähigkeit und damit auch die elektrische Kontaktierbarkeit zu erhöhen. An der negativen Elektrode ist dies nicht notwendig, da es sich bei dem meistens verwendeten Graphit bereits um einen guten elektrischen Leiter handelt.

Bei Lithium-Ionen-Akkumulatoren stellt Lithium selbst nicht das Aktivmaterial der Elektroden dar, sondern andere Materialien, welche Lithium-Atome einlagern oder mit denen Lithium-Atome reagieren können. Die verschiedenen Aktivmaterialien zeichnen sich dabei durch ihre unterschiedlichen elektrischen Potentiale aus.

Als Materialien für die negative Elektrode eignen sich Stoffe, welche vom Potential nahe dem des Lithiums sind. Dieses stellt die 0V-Referenz dar. Ein Auszug der verwendeten Materialien ist in Tabelle 2.2 gemeinsam mit Bewertungen der jeweiligen Sicherheits- und Stabilitätseigenschaften dargestellt.

	Lithium Metall	**Armorpher Kohlenstoff**	**Graphit**	**Lithium-Legierungen**	**Titanat, Li4Ti5O12**
Potential-bereich vs Li/Li+ in mV	0	100-700	50-300	50-600	1400-1600
Kapazität mAh/g	Achtung: Referenz, andere Basis! 3860	ca. 200	372	3990 für Silizium bzw. 1000 für Zinn	150
Sicherheit	-	+	+	o	++
Stabilität	-	+	+	-	++
Preis	+	o	+	++	o

Tabelle 2.2.: Zusammenstellung verschiedener Aktivmaterialien für die negative Elektrode einer Lithium-Ionen-Zelle [1]. Der Bezugspunkt für die Kapazität kann zwischen Lithium-Metall und den restlichen Materialien nicht verglichen werden, da es sich bei ihnen um Wirtsmaterialien für Lithium handelt.

Materialien für die positive Elektrode sollten ein möglichst hohes Potential gegenüber der $0V$-Referenz besitzen. Eine nicht vollständige Zusammenstellung ist in Tabelle 2.3 dargestellt.

Aus den verschiedenen Kombinationen ergeben sich unterschiedliche Spannungen einer Zelle.

	$LiCoO_2$	$LiNiO_2$	$LiMn_2O_4$	$Li(Ni_xCo_yMn_z)O_2$	$LiFePO_4$
Mittlere Spannung vs Li/Li+ in V	3,9	3,80	4,0	3,8-4,0	3,4
Kapazität mAh/g	150	170	120	130-160	160
Sicherheit	-	-	+	o	++
Stabilität	-	-	o	o	++
Preis	- -	-	+	o	+

Tabelle 2.3.: Zusammenstellung verschiedener Aktivmaterialien für die positive Elektrode einer Lithium-Ionen-Zelle [1]

2.2.3. Lithium-Eisenphosphat-Akkumulatoren

Eine Abwandlung der Lithium-Ionen-Akkumulatoren stellt der Lithium-Eisenphosphat-Akkumulator dar. Dabei handelt es sich um eine verhältnismäßig neue Entwicklung. Die Eignung von Lithium-Eisen-Phosphat ($LiFePO_4$) als Aktivmaterial wurde 1996/1997 entdeckt [19].
Diese Verbindung zeichnet eine hohe thermische Stabilität aus, welches es weniger anfällig für Brände und Explosionen macht. Das Material eignet sich daher besonders für sicherheitsrelevante Anwendungen, wie auch aus Tabelle 2.3 hervorgeht.
Ein weiterer Vorteil sind die günstigen Beschaffungskosten, da die benötigten Materialien gut verfügbar sind. Dazu gehören im Wesentlichen Lithium, Eisen, Aluminium und Kupfer. Dadurch sind Lithium-Eisenphosphat-Akkus besonders interessant für Anwendungen mit großen Kapazitäten und somit großen Mengen an Batteriematerial, wie zum Beispiel in Elektrofahrzeugen.
Negativ schlägt die schlechte elektrische Leitfähigkeit zu Buche. Um diese zu verbessern wird daher das Lithium-Eisenphosphat üblicherweise in kleinen Kristallen verarbeitet (Nano-Eisenphosphat) und mit Kohlenstoff beschichtet [20].

In einem Akku wird Lithium-Eisenphosphat als Kathodenmaterial eingesetzt. Beim Laden eines solchen Akkus löst sich Lithium vom Eisenphosphat entsprechend der Reaktionsgleichung:

$$LiFePO_4 \rightarrow Li^+ + FePO_4 + e^- \tag{2.11}$$

Die gelösten Lithium-Ionen wandern durch den Elektrolyten zur Anode. Sie werden dort in der üblicherweise aus Graphit bestehenden Anode eingelagert:

$$C_6 + Li^+ + e^- \rightarrow LiC_6 \tag{2.12}$$

Die in beiden Reaktionsgleichungen teilnehmenden Elektronen wandern beim Ladevorgang von der Lithium-Eisenphosphat-Kathode zur Graphit-Anode.

Beim Entladevorgang finden genau entgegengesetzte Reaktionen statt.

2.3. Optische Messverfahren

Lichtwellenleiter können als optische Sensoren in einer Vielzahl von Fällen verwendet werden. Voraussetzung, um Lichtleiter als Sensoren für eine Messgröße verwenden zu können, ist, dass diese Messgrößen dazu in der Lage sind, die im Lichtleiter geführten Strahlen zu beeinflussen. Eine Möglichkeit ist die Beeinflussung des Transmissionsverhaltens im Leiter durch die Absorptionseigenschaften oder dem Brechungsindex des zu untersuchenden Materials.
Abgesehen von solchen "Intensitätsmessungen" können auch Messungen der "Phasenlage, Polarisation oder unter besonderen Voraussetzungen der Frequenz der Lichtwelle" durchgeführt werden. Für Sensoren, bei welchem die Phasenlage des Lichtes gemessen werden soll, sind nur Einmoden-Lichtwellenleiter verwendbar. Bei Vielmoden-Lichtwellenleitern gehen die Informationen über die Phasenlage aufgrund von Laufzeitdifferenzen der einzelnen Moden sowie durch Austausch von Lichtleistung zwischen den Moden verloren. Auch für Polarisationsmessungen können nur Einmoden-Lichtwellenleiter verwendet werden [8].

2.3.1. Evaneszenzfeld-Sensor

Wie in Kapitel 2.1.6 erläutert, dringt das durch den Lichtwellenleiter geführte Licht auch bei Totalreflexionen in den Mantel ein.
Ersetzt man den Mantel eines Lichtwellenleiters mit einem zu messenden Material oder einer Substanz, so können Änderungen in diesem Material zu unterschiedlichen Lichtleitungseigenschaften führen.

Um einen messbaren Effekt herbeizuführen, muss das Material seine Absorptionseigenschaften für bestimmte Wellenlängen des Lichtes abhängig von seinem Zustand ändern.
Ist dies der Fall, so variiert bei konstanter Einspeiseleistung des Lichtes, je nach Zustand des Materials, die Lichtintensität am Ende des Lichtwellenleiters. Die gemessene Größe kann in Relation zu einer weiteren Referenzgröße gesetzt werden, welche durch das zu messende Material möglichst unbeeinflusst sein sollte.

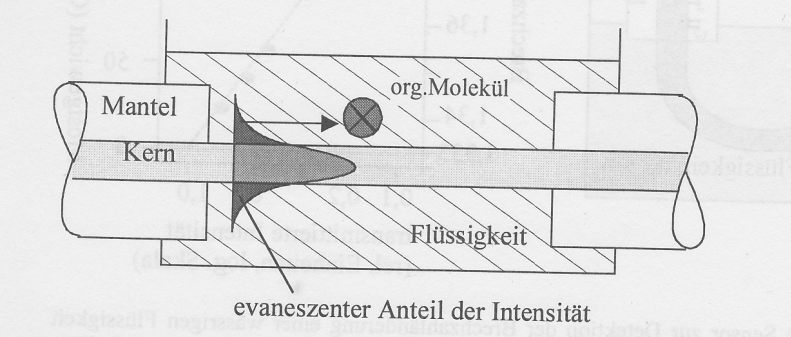

Abbildung 2.11.: Beispiel eines Evaneszenzfeld-Sensors. Die gestreift eingezeichnete Flüssigkeit ersetzt den Mantel des Lichtwellenleiters. Der Kreis mit dem Kreuz in der Mitte stellt ein Molekül dar, welches das geleitete Licht entsprechend seiner Absorptionseigenschaften beeinflusst [8].

2.3.2. Brechungsindex-Sensor

Ändert eine zu messende Substanz je nach Zustand seine Dichte und somit seinen Brechungsindex, so kann dies mithilfe von Lichtwellenleitern gemessen werden. Es gibt zwei Möglichkeiten, einen solchen Sensor zu realisieren. Zum einen kann der Mantel des Lichtwellenleiters, wie auch schon in Abschnitt 2.3.1, mit einem zu vermessenden Material ersetzt werden. So wird, abhängig vom Brechungsindex des Materials, das Licht verschieden stark aus dem Lichtleiter ausgekoppelt. Die sich dadurch ändernde Lichtleistung im Leiter kann wiederum am Ende der Faser gemessen werden.

Es muss aber nicht zwingend der Mantel des Lichtwellenleiters ersetzt werden. Man kann den Leiter auch so stark biegen, dass Licht entsprechend der Abbildung 2.7 ausgekoppelt wird. Umgibt das zu messende Material nun den Mantel des Lichtwellenleiters, so wirkt dieser als zweiter Mantel. Dieser zweite Mantel kann abhängig von seinem Brechungsindex dazu beitragen, Licht im Leiter zu halten.

Sinnvoll kann eine Kombination beider Varianten sein. Bei einem gebeugten Leiter mit ersetztem Mantel hat die zu messende Substanz einen verhältnismäßig starken Einfluss auf das ausgekoppelte oder reflektierte Licht.

Ein solcher Sensor kommt in der, parallel zu dieser laufenden, Bachelorarbeit von Wahid Nasimzada zum Einsatz [21]. Dort wird er innerhalb einer Blei-Säure-Batterie zur Bestimmung der Ladung eingesetzt.

2.3.3. Messmethodik der Pulsoxymetrie

Bei der Pulsoxymetrie wird durch optische Messungen unter anderem die Sauerstoffsättigung des Blutes und die Herzfrequenz ermittelt. Das Messprinzip soll daher als Beispiel für optische Messungen im Allgemeinen dienen.
Hämoglobin ist ein wichtiger Bestandteil von Blut. Es dient zur Bindung und zum Transport von Sauerstoff, welcher in der Lunge durch Atmung zugeführt wird. Dieser Vorgang wird als Oxygenierung bezeichnet. In Abbildung 2.12 sind die Absorptionsspektren von oxygeniertem und desoxygeniertem Hämoglobin zu sehen, welche sich deutlich unterscheiden.

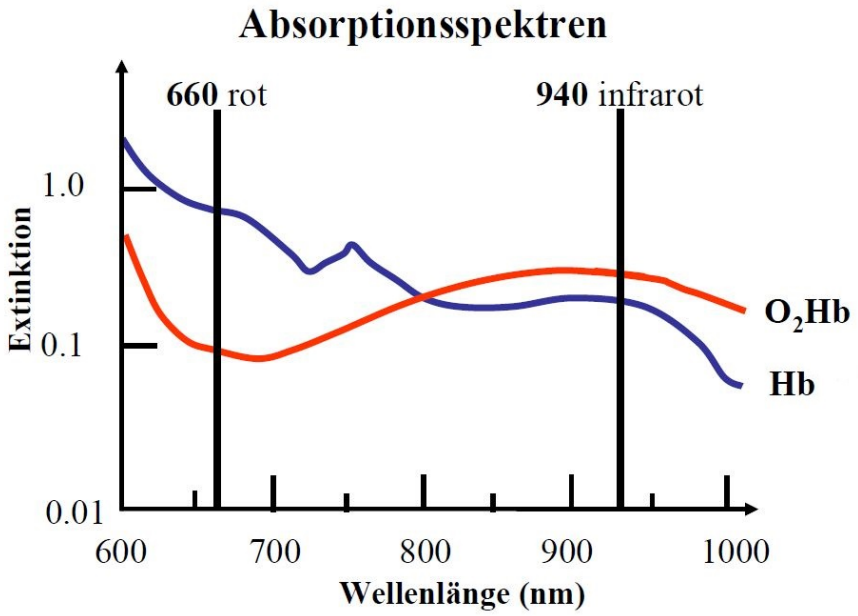

Abbildung 2.12.: Absorptionsspektren von oxygeniertem (O_2Hb) und desoxygeniertem Hämoglobin (Hb) [22]

Dies wird sich zu Nutze gemacht, indem mit LEDs Transmissionsmessungen bei rotem (660nm) und infrarotem Licht (940nm) durchgeführt werden. Das Licht einer LED durchleuchtet, wie in Abbildung 2.13 gezeigt, einen Finger. Auf der gegenüberliegenden Seite wird die Intensität des Lichtes mit einem Lichtsensor, in der Regel einer Photodiode, gemessen. Bei der Transmission durch den Finger hindurch wird das Licht je nach Sauerstoffgehalt unterschiedlich stark absorbiert. Bei 660 nm wird gemessen, da hier die die Differenz zwischen oxygeniertem und desoxygeniertem Hämoglobin am größten ist. Bei 940 nm erreicht oxygeniertes Hämoglobin ein Maximum. Durch den Vergleich der gemessenen Intensitäten des roten und infraroten Lichtes lässt sich auf den Sauerstoffgehalt im Blut schließen, indem diese beiden Werte ins Verhältnis zueinander gesetzt werden.

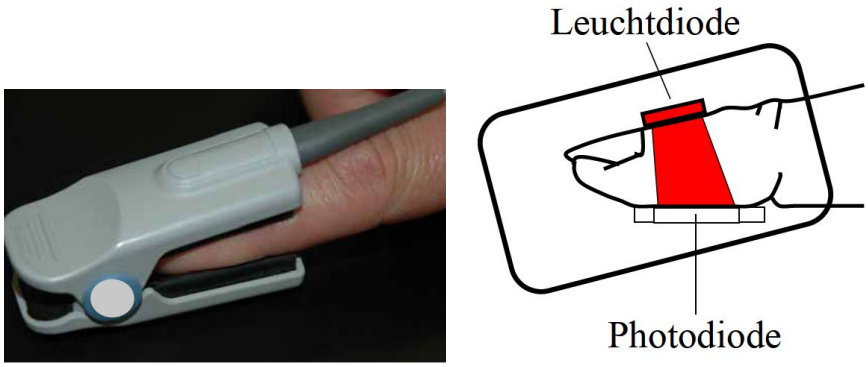

Abbildung 2.13.: Mechanischer Aufbau eines Pulsoxymeters [23] und das schematische Funktionsprinzip [22]

Neben der oben beschriebenen Messmethode gibt es noch eine weitere, seltener eingesetzte Variante. Anstelle der Anordnung des Lichtsensors direkt gegenüber der Lichtquelle wird er direkt neben ihm angebracht. Gemessen wird dann nicht das transmittierte, sondern das reflektierte Licht. Zur Bestimmung der Sauerstoffsättigung wird in diesem Fall anstelle des hier nicht gültigen Absorptionsspektrums ein Reflexionsspektrum (Abbildung 2.14) herangezogen. Auch hier haben Hb und O_2Hb bei der Wellenlänge 660 nm die größte Differenz. Außerdem ist die Reflexion des oxygeniertem Hämoglobins bei dieser Wellenlänge bereits maximal. Daher wird bei der Wellenlänge 890 nm lediglich eine Referenzmessung durchgeführt. Bei dieser Wellenlänge schneiden sich die Graphen von Hb und O_2Hb und somit hat der Sauerstoff an diesem Punkt keinen Einfluss auf das reflektierte Licht. Ein einfacher Vergleich der Intensitäten bei beiden Wellenlängen liefert einen Hinweis auf die Sauerstoffsättigung des Blutes.

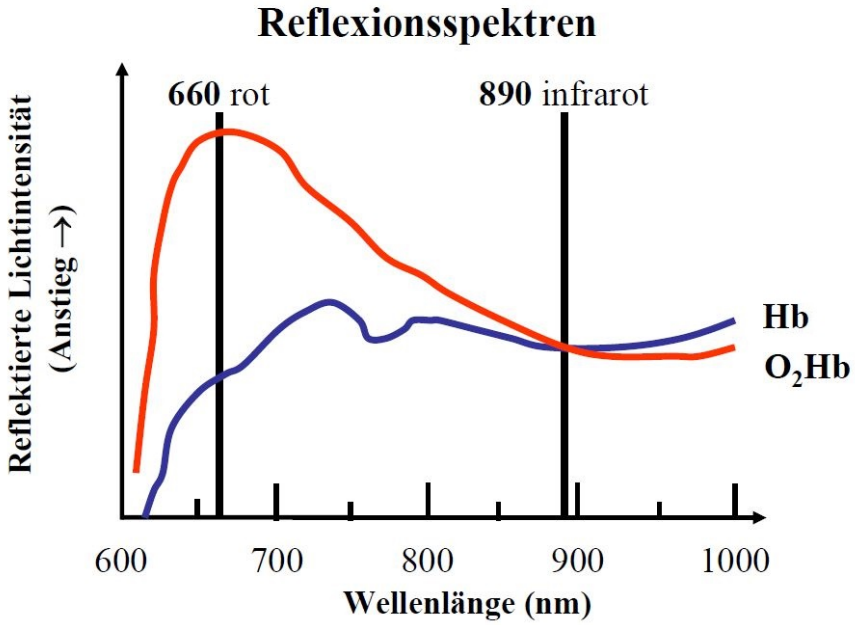

Abbildung 2.14.: Reflexionsspektren von oxygeniertem (O_2Hb) und desoxygeniertem (Hb) Hämoglobin [22]

3. Lichtleiter-Sensorik für Lithium-Batterien

Die vorangegangen Ausführungen behandeln die benötigten Grundlagen zur Umsetzung von optischer Sensorik.
In diesem Kapitel sollen diese Grundlagen für die Realisierung eines optischen Sensors für Lithium-Eisenphosphat-Kathoden angewandt werden.

3.1. Konzept

Durch die Einlagerung von Lithium-Ionen in Lithium-Eisenphosphat-Kathoden finden optische Veränderungen statt. Die Ursache dafür sind die Absorptionseigenschaften des Materials, das von $FePO_4$ zu $LiFePO_4$ umgewandelt wird. Aber auch durch die Volumenänderung, welche durch die Einlagerung von Lithium hervorgerufen wird, werden die Absorptionseigenschaften des Materials beeinflusst.
Bei der Lithinierung von Eisenphosphat sind ähnliche Effekte zu erwarten, wie an der in Abschnitt 1.2 gezeigten Graphit-Anode bei einem Ladevorgang. Dabei finden die optischen Veränderungen nicht gleichmäßig im Material statt. Bei der Entladung eines Lithium-Eisenphosphat-Akkumulators suchen sich die durch den Elektrolyten wandernden Lithium-Ionen (Abschnitt 2.2.2) den nächstgelegenen Ort im Metallgitter des Eisenphosphats, in welchem sie eingelagert werden können. Aus demselben Grund kommt es zu dem in Abbildung 1.1 zu sehenden Farbverlauf der Graphit-Anode.

In Vorarbeiten wurden Proben hergestellt, für die Eisenphosphat mit verschiedenen Mengen eines Lithium-Wassergemisches (Voruntersuchung 1) und mit einer Butyllithium-Lösung (Voruntersuchung 2) versetzt wurden. Diese Proben wurden anhand von Reflexionsmessungen analysiert. Das Ziel dieser Untersuchungen war, unterschiedliche Lithinierungen einzustellen und ihre Absorptionseigenschaften zu ermitteln.

In Abbildung 3.1 sind Reflexionsmessungen an Voruntersuchung 1 gezeigt. Bei dieser Untersuchung ist kein Lithium-Eisenphosphat entstanden, sondern höchstwahrscheinlich roter

Rost (Fe_2O_3). Die Messungen stellen daher keinen Indikator für die Variation der Absorptionseigenschaften von Lithium-Eisenphosphat dar, können aber stellvertretend für Reflexionsmessungen an Feststoffen herangezogen werden.
In dem Bild sind deutliche Intensitätsänderungen bei den unterschiedlichen Proben bei circa 545 nm und 610 nm zu erkennen: je mehr Lithium-Wassergemisch zu dem Eisenphosphat hinzugegeben wurde, desto intensiver sind die Reflexionen bei diesen beiden Wellenlängen.

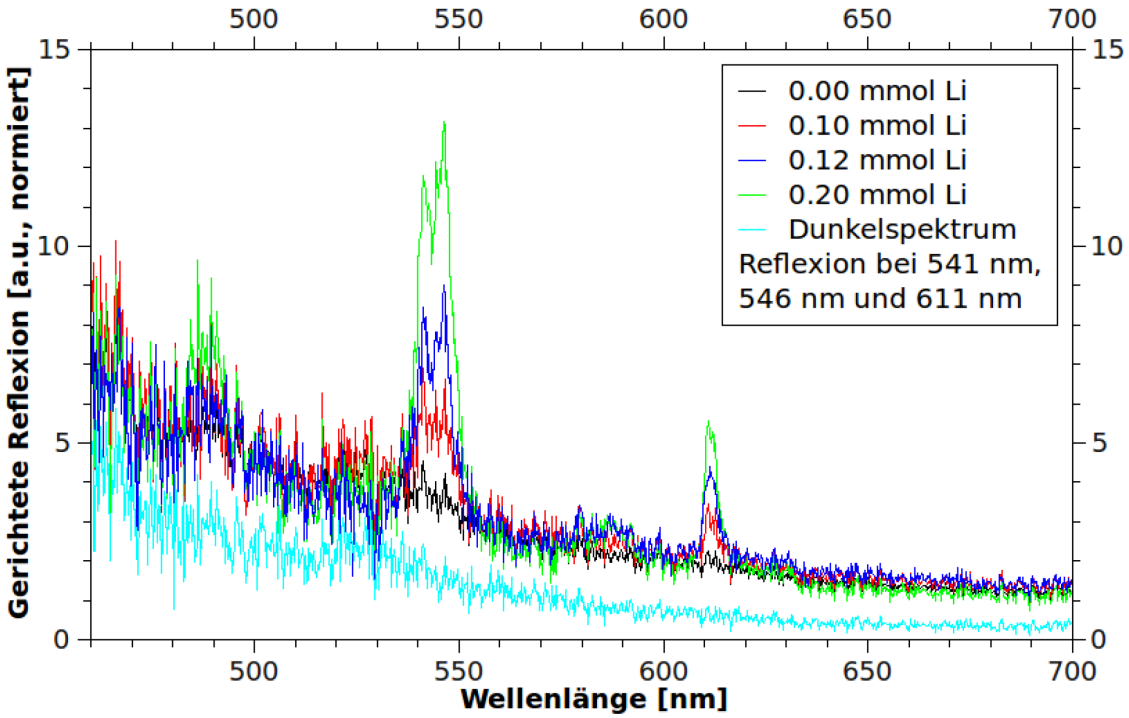

Abbildung 3.1.: Optische Voruntersuchung an oxidierendem Eisenphosphat. Bei fortschreitender Oxidation steigt bei einer Reflexionsmessung die gemessene Intensität auf zwei spezifischen Wellenlängen (ca. 545 nm und 612 nm) an. Eine Messung bei diesen Wellenlängen könnte spezifisch beispielsweise durch Laserdioden oder LED erfolgen, so dass das Verhalten bei anderen Wellenlängen die Messung nicht beeinflusst. Messungen auf Wellenlängen, bei denen keine Änderung stattfindet, können als Referenzwerte verwendet werden [24].

In Abbildung 3.2 ist der Effekt an mit Butyllithium versetztem Eisenphosphat zu sehen. Für die Herstellung dieser Probe wurden kleine Mengen von Eisenphosphat unter Schutzatmosphäre (in einer Glovebox unter Argon) versetzt. Zu sehen ist eine Bruchfläche eines un-

gefähr 1x2x2 mm^3 messenden Probestücks. Proben aus dieser Durchführung wurden in Kapitel 4.2 optisch vermessen.

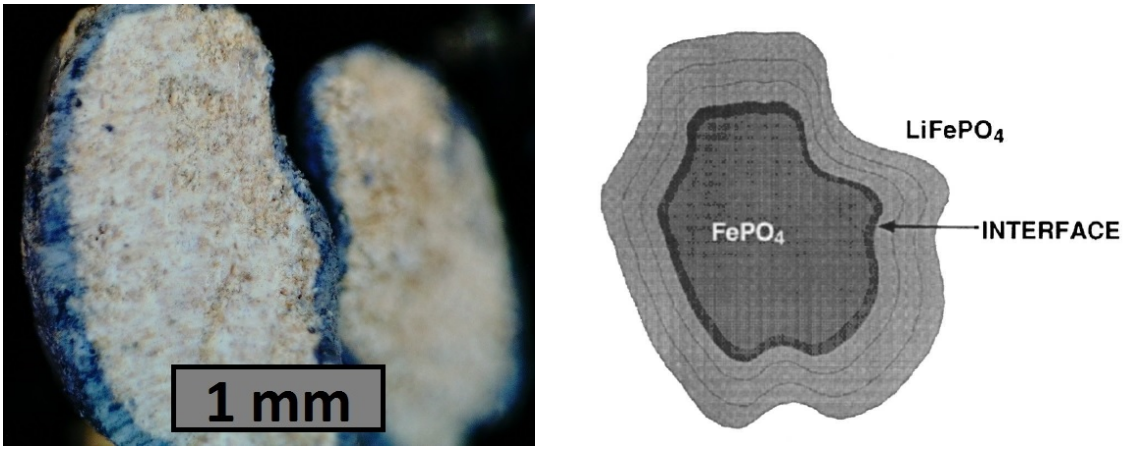

Abbildung 3.2.: Links: Mit Butyllithium versetztes Eisenphosphat, gezeigt ist eine Bruchstelle. Die Einlagerung findet von außen nach innen und nicht gleichmäßig über die gesamte Fläche statt [24].
Rechts: Schematische Darstellung des Lithiumtransports in Eisenphosphat nach Goodenough *et al.* [19].

3.2. Neuer Lösungsansatz

Im Rahmen des BATSEN-Projektes ist die neue Idee entstanden, wie die in Abschnitt 3.1 erläuterten optischen Veränderungen von Lithium-Eisenphosphat-Kathoden bei verschiedenen Ladezuständen ausgenutzt werden können. Der Ladezustand eines Lithium-Ionen-Akkus steht im direkten Zusammenhang mit den in den Elektrodenmaterialien eingelagerten Lithium-Ionen. Der Grad der Lithinierung einer Eisenphosphat-Elektrode wiederum lässt die Absorptionseigenschaften des Materials variieren. Die im Projekt entwickelte Idee ist es, diese optische Eigenschaften innerhalb einer Zelle, also in situ, zu messen. Als Sensor sollen Lichtwellenleiter dienen.

Ähnlich der in Abschnitt 2.3.3 vorgestellten Pulsoxymetrie, sollen dafür Messungen mit LEDs verschiedener Wellenlänge durchgeführt werden. Welche Wellenlängen diese LEDs haben sollen, richtet sich nach den Absorptionseigenschaften von Lithium-Eisenphosphat. Beim Pulsoxymeter sind die Wellenlängen der LEDs anhand der in Abbildung 2.12 gezeigten Absorptionsspektren für oxygeniertes und desoxygeniertes Hämoglobin bestimmt worden. Um entsprechende Informationen über die Absorptionseigenschaften von Lithium-Eisenphosphat zu erlangen, wurden die in Abbildung 3.1 gezeigten Messungen durchge-

führt. Dementsprechend würde eine sinnvolle Wellenlänge bei ungefähr 545 nm liegen, da dort die Intensitätsänderungen der Reflexionsmessungen am größten sind. Da Messungen nur anhand von Absolutwerten in diesem Fall sehr anfällig für Störungen sind, soll eine weitere Messung mit einer weiteren LED mit einer anderen Wellenlänge durchgeführt werden. Dafür bietet sich zum Beispiel eine Wellenlänge an, bei welcher die Absorptionseigenschaften des Materials möglichst konstant sind. Dafür würden sich nach Abbildung 3.1 besonders Wellenlängen größer als 620 nm, zum Beispiel im Infrarotbereich, anbieten. Da die Proben bei den Reflexionsmessungen aber wahrscheinlich eher oxidiert als lithiniert sind, müssten noch aussagekräftigere Messungen durchgeführt werden, damit sinnvolle Wellenlängen der LEDs ermittelt werden können. Dafür wurden mit Butyllithium versetzte Proben im Rahmen dieser Arbeit untersucht.

3.3. Modifikation des Lichtleiters

Um optische Messungen an Lithium-Eisenphosphat-Kathoden durchführen zu können, muss ein Lichtleiter in das Kathodenmaterial eingeführt werden und von eben diesem umgeben sein. Vorgesehen ist, den Mantel der Faser entsprechend dem Abschnitt 2.3.1 innerhalb des Kathodenmaterials teilweise zu entfernen. Wie in Abbildung 3.3 gezeigt wird, ersetzt das Kathodenmaterial an diesen Stellen den Mantel, wodurch es das durch den Lichtleiter geführte Licht beeinflussen kann.

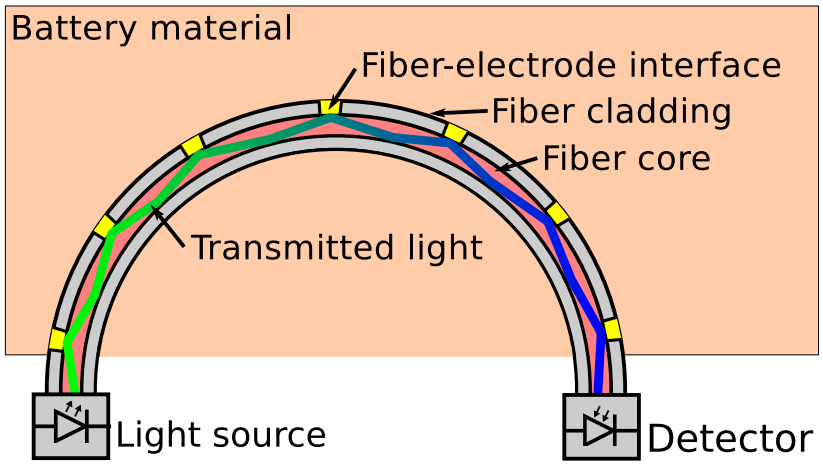

Abbildung 3.3.: Gebeugter Lichtleiter mit stellenweise modifiziertem Mantel im Batteriematerial. Das von der Quelle emittierte Licht wird an den gelb eingezeichneten Stellen durch das Elektrodenmaterial und dessen Absorptionseigenschaften beeinflusst [24].

Entsprechend der vorhergegangenen Erläuterungen in 3.1 und 3.2 wird erwartet, dass die optischen Veränderungen nicht gleichmäßig über die gesamte Fläche des Kathodenmaterials, sondern ähnlich zum bereits gezeigtem Verlauf der Graphit-Anoden in Abbildung 1.1 auftreten. Um diese ungleichmäßigen Veränderungen messen zu können, ist es notwendig, den Lichtleiter über eine möglichst große Kathodenfläche zu führen und möglichst weitgehend den Mantel der Faser mit Lithium-Eisenphosphat zu ersetzen.
Abbildung 3.3 zeigt außerdem einen gebeugten Verlauf der Faser. Wie in Abschnitt 2.1.3 und 2.1.4 erläutert, kann das Licht nur geleitet werden, wenn es einen bestimmten Einfallswinkel beim Auftreffen auf die Grenzschicht zwischen Kern und Mantel des Lichtleiters nicht unterschreitet. Ansonsten tritt das Licht teilweise aus dem Lichtleiter aus. Der Winkel bestimmt sich aus den Brechindizes des Kern- und Mantelmaterials.
Entsprechend Abschnitt 2.1.6 bestimmt der Einfallswinkel an der Grenzfläche zwischen Kern und Mantel die Eindringtiefe des Lichtes in das Mantelmaterial bei der Reflexion. Durch diese Beugung werden die Einfallswinkel des Lichtes auf die Grenzschicht zwischen Kern und Mantel verringert, wodurch die Eindringtiefe des Lichtes in das Material des Mantels erhöht wird. Die höhere Eindringtiefe hat eine stärkere Beeinflussung des durch den Lichtleiter geführten Lichtes durch die Absorptionseigenschaften des Kathodenmaterials zur Folge.

4. Vorarbeiten für optische Messverfahren

4.1. Inbetriebnahme des Spektrometers und Auswertung durch Matlab-Skripte

Zur Auswertung der spektralen Zusammensetzung von Licht wird ein Spektrometer benötigt. Bei dem verwendeten Gerät ASEQ-Instruments LR1-T handelt es sich um ein kommerzielles Spektrometer, welches per USB an einen PC angeschlossen und mithilfe der mitgelieferten Software Spektralanalysen durchgeführt werden. Es besteht aber auch die Möglichkeit, die Messergebnisse des Sensors selbst auszuwerten oder in LabVIEW einzubinden. Entsprechende Beispiele werden vom Hersteller bereitgestellt.

Abbildung 4.1.: ASEQ-Instruments LR1-T Spektrometer [25]

Der Messbereich des Spektrometers reicht von 200 nm bis 1100 nm und es hat eine Auflösung von bis zu 0.2 nm. In dem Spektrometer wird ein CCD-Sensor von Toshiba mit 1x3648 Pixeln verwendet [26]. Das in das Spektrometer einfallende Licht wird mit mit einem Beugungsgitter in seine spektralen Anteile aufgespalten. Jeder der Pixel des CCD-Sensors misst daher die Lichtintensität bei einer eigenen Wellenlänge. Die Auflösung des Spektrometers ergibt sich aus den Parametern für Beugungsgitter, Lichtspalt und Lichtsensor.

In der Software können Einstellungen vorgenommen werden, welche die Messungen beeinflussen. Die Oberfläche der Software ist in Abbildung 4.2 gezeigt.

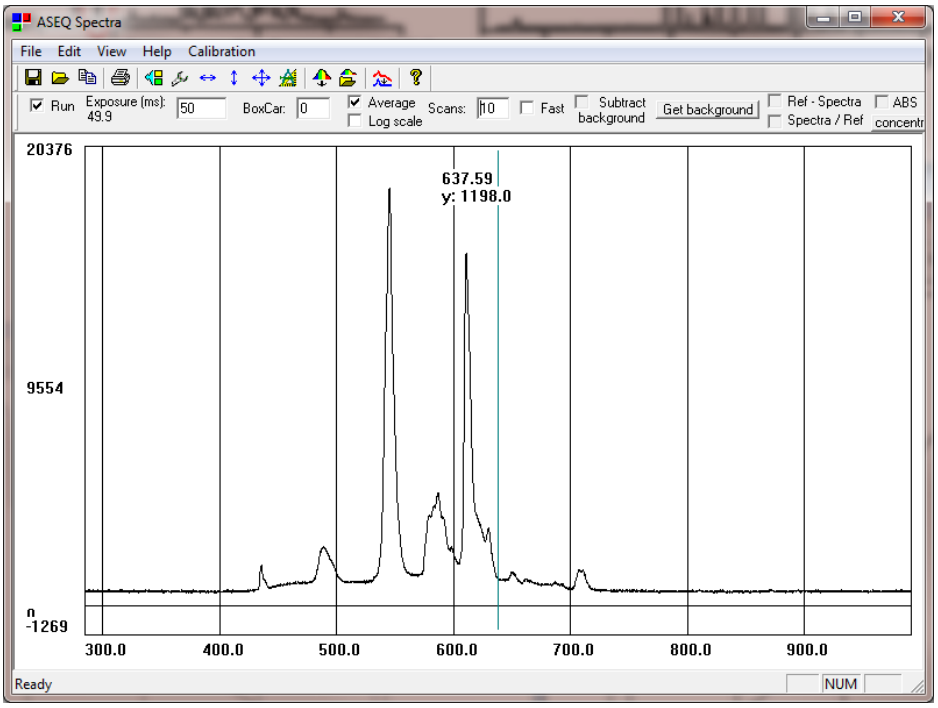

Abbildung 4.2.: ASEQ Spectra Mess-Software

Mit "Exposure" stellt man die Belichtungszeit des Sensors in Millisekunden ein. Wird diese beispielsweise erhöht, so wird die Lichtintensität länger gemessen und entsprechend wird auch der Ausschlag des Sensors höher.
Bei "BoxCar" handelt es sich um einen Glättungsfilter. Durch die Konfiguration des BoxCar-Wertes wird die Anzahl der benachbarten Messwerte bestimmt, welche zum Mitteln herangezogen werden sollen.
Bei dem in "Scans" eingetragene Wert handelt es sich um die Anzahl der Messungen, über welche gemittelt werden soll, wenn das Feld "Average" ausgewählt wird.
Des Weiteren gibt es die Möglichkeit, mit "Get Background" ein Dunkelspektrum aufzunehmen, welches dann von den folgenden Messungen abgezogen wird, wenn das Feld "Subtract background" ausgewählt wird.

Im Gegensatz zu dem im Datenblatt erwähnten messbaren Spektralbereich, lässt sich in der Software nur ein Bereich von 284.29 nm bis 990.76 nm darstellen.

Das aktuell gemessene Spektrum kann in einer Textdatei gespeichert werden. In einer solchen Datei werden lediglich die Wellenlängen und die entsprechenden Messdaten abgelegt. Diese abgespeicherten Spektren können dann leicht mit Matlab ausgewertet werden.
Im Laufe der Bachelorarbeit sind eine Vielzahl an solchen Skripten entstanden. Grundsätzlich wurde immer die Möglichkeit implementiert, die Spektren auf eine Wellenlänge zu normieren, bevor ein Graph erzeugt wird.
Das Normieren der Wellenlängen spiegelt dabei schon die Idee wider, wie im später aufzubauenden LED-Reflektometer, Referenzmessungen einzubinden.
Zum Einlesen einer Textdatei gibt es eine große Anzahl an verschiedenen Matlabfunktionen. Abgesehen von der Möglichkeit, eine Datei selbst mittels fopen, fread und fscanf auszulesen, gibt es auch die Möglichkeit, Funktionen wie csvread, tdfread, tblread, dlmread einzusetzen. Als effektiv für die Daten der Spektren hat sich die Matlab-Funktion dlmread herausgestellt, welches dafür vorgesehen ist, durch ein ASCII-Zeichen getrennte numerische Werte in einer Matrix abzuspeichern.

Die erstellten Skripte zur Auswertung können der beigelegten CD entnommen werden.

4.1.1. Breitbandige Lichtquelle

Für Reflexionsmessungen, welche sich über ein breites Lichtspektrum erstrecken sollen, wurde eine breitbandige Halogenlampe verwendet. Die Lichtquelle "SL1 Tungsten Halogen" von StellarNet erstreckt sich laut Datenblatt über einen Spektralbereich von 350 nm bis 2200 nm.

Abbildung 4.3.: StellarNet Halogenlampe [27]

Es wurde zwei Messungen mit der Halogenlampe durchgeführt. Bei der ersten stand die Lampe gegenüber dem Spektrometer, sodass der Eingang der SMA-Buchse des Spektrometers direkt angestrahlt wurde. Bei der zweiten Messung wurde die Halogenlampe verkippt gegenüber dem Spektrometer aufgestellt. Bei den beiden Messungen ergaben sich folgende Spektren :

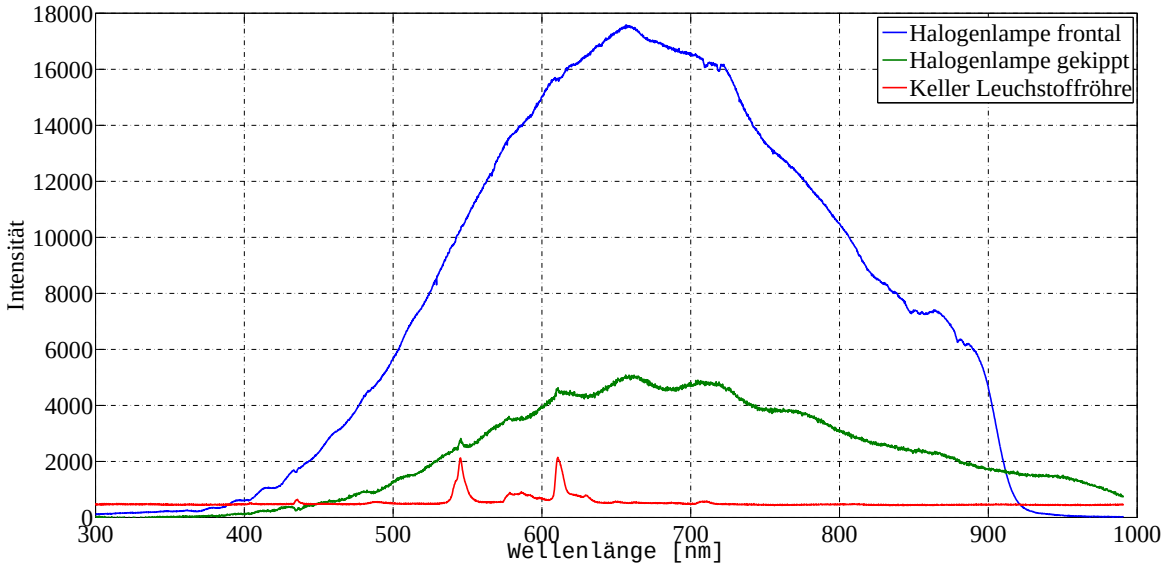

Abbildung 4.4.: Spektren einer Halogenlampe frontal und verkippt gegenüber dem Spektrometer. Die gemessene Hintergrundbeleuchtung wurde von den beiden Graphen abgezogen.

Das auch in Abbildung 4.4 dargestellte Hintergrundleuchten des Raumes wurde von den Messungen mit der Halogenlampe abgezogen. Trotzdem ist der Einfluss der von den Leuchtstoffröhren der Raumbeleuchtung erzeugten Spektren noch in beiden Messungen zu erkennen.
Vergleicht man nun die beiden Spektren, welche von derselben Halogenlampe erzeugt werden, fällt auf, dass die verkippt gegenüber des Spektrometers aufgestellte Halogenlampe ab ungefähr 920 nm eine höhere Intensität besitzt, als wenn sie frontal gegenüber aufgestellt wird. Es konnte keine Ursache für diesen ungewöhnlichen Effekt festgestellt werden.

In der folgenden Abbildung 4.5 soll der Einfluss einer POF-Faser auf die Lichtleitung demonstriert werden. Dafür wurde eine mit Toslink-Steckern ausgestattete POF-Faser zwischen der Halogenlampe und dem Spektrometer angeschlossen. Bei den optischen Anschlüssen des Spektrometers und der Halogenlampe handelt es sich jedoch um SMA-Buchsen. Da es keine kommerziellen Adapter für diese beiden Stecksysteme gibt, mussten Eigenlösungen hergestellt werden. Daher wurde bei der Messung des Einflusses der POF-Faser auf die Lichtlei-

tung zusätzlich eine Glasfaser als Leiter angeschlossen. Das Spektrum, welches nur von der Glasfaser übertragen wird, ist ebenfalls in Abbildung 4.5 dargestellt.

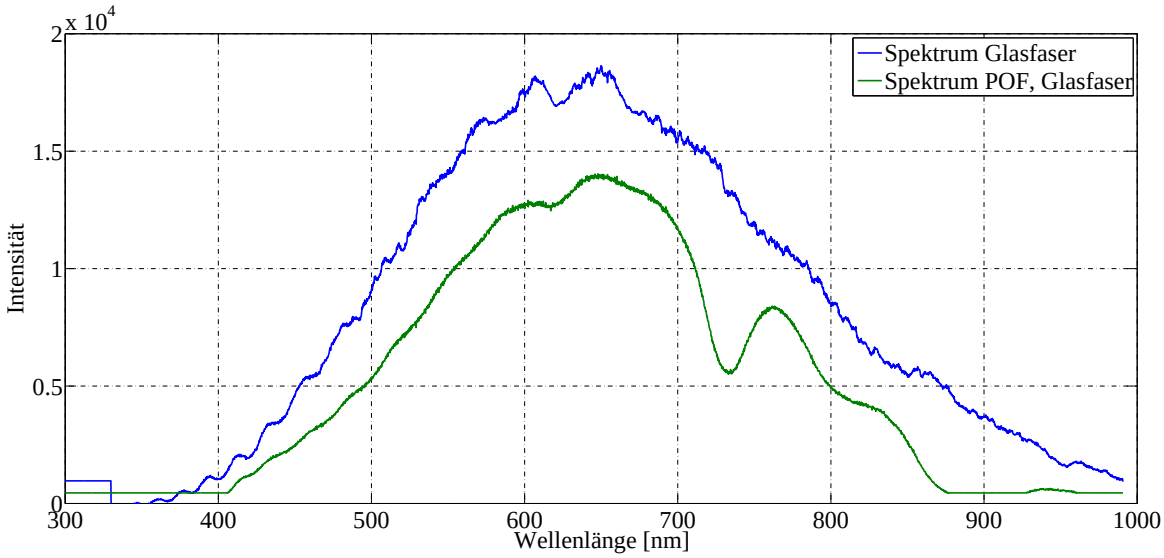

Abbildung 4.5.: Lichtleitung mit POF/Glasfaser. Absorptionen, verursacht durch die Materialeigenschaften der POF-Faser, sind bei den Wellenlängen 620 nm und 730 nm erkennbar.

Vergleicht man den Einfluss des POF-Leiters auf das übertragene Licht mit den charakteristischen Dämpfungen aus Abbildung 2.6, so kann man den Abfall bei ca. 730 nm deutlich den Eigenschaften der POF-Faser zuschreiben. Auch bei circa 620 nm ist ein Abfall erkennbar, welcher sich aber auch in dem Spektrum der Glasfaser zeigt. Diese ist jedoch leicht übersteuert, was durchaus für die dort zu sehende Welle im Intensitätsmaximum verantwortlich sein kann. Ähnliches Verhalten des Spektrometers wurden bei mehreren Messungen an hellen Lichtquellen beobachtet.

Während der Verwendung der Halogenlampe fiel auf, dass sich diese nach dem Einschalten nicht konstant verhält. Daher wurden verschiedene Messungen durchgeführt, welche das Verhalten der Halogenlampe über die Zeit zeigen sollen.

Die in Abbildung 4.6 zu sehenden Spektren bestätigen die Beobachtung, dass sich die Intensität der Lichtquelle nach dem Einschalten verändert. Aufgrund der vielen aufgenommenen Spektren ist es jedoch nicht leicht, weitere Informationen aus dem Graphen abzuleiten.

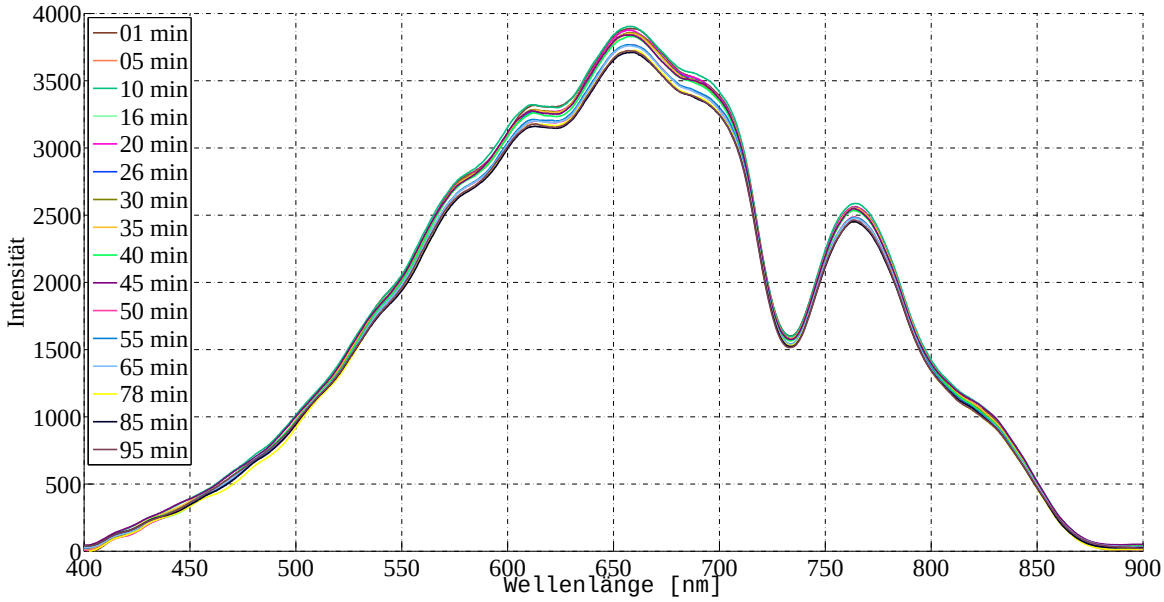

Abbildung 4.6.: Spektren der Halogenlampe zu verschiedenen Zeitpunkten nach dem Einschalten

Um einen besseren Eindruck über das Verhalten bei Erwärmung zu erlangen, wurde mit Abbildung 4.7 ein weiteres Diagramm erzeugt. Dieses zeigt eine geringere Anzahl an Messungen und ist auf den Bereich um die Intensitätsmaxima herum vergrößert. Die Maxima schwanken leicht bei zeitlich aufeinanderfolgenden Messungen. So steigt das Maximum von erster zu zehnter Minute. Von zehnter Minute zu zwanzigster Minute fällt das Intensitätsmaximum aber. Über die gesamte Zeitdauer zeigen die Spektren in der Tendenz einen Abfall der Intensitäten.

Zur Verdeutlichung sind in Abbildung 4.8 die maximalen Intensitäten bei ungefähr 660 nm und die über alle Wellenlängen integrierte Intensität der mit dem Spektrometer aufgenommenen Messdaten dargestellt. Dafür wurden wieder alle aufgezeichneten Messungen verwendet. Auch hier sieht man in der Tendenz in beiden Graphen einen Abfall der gemessenen Lichtintensitäten.

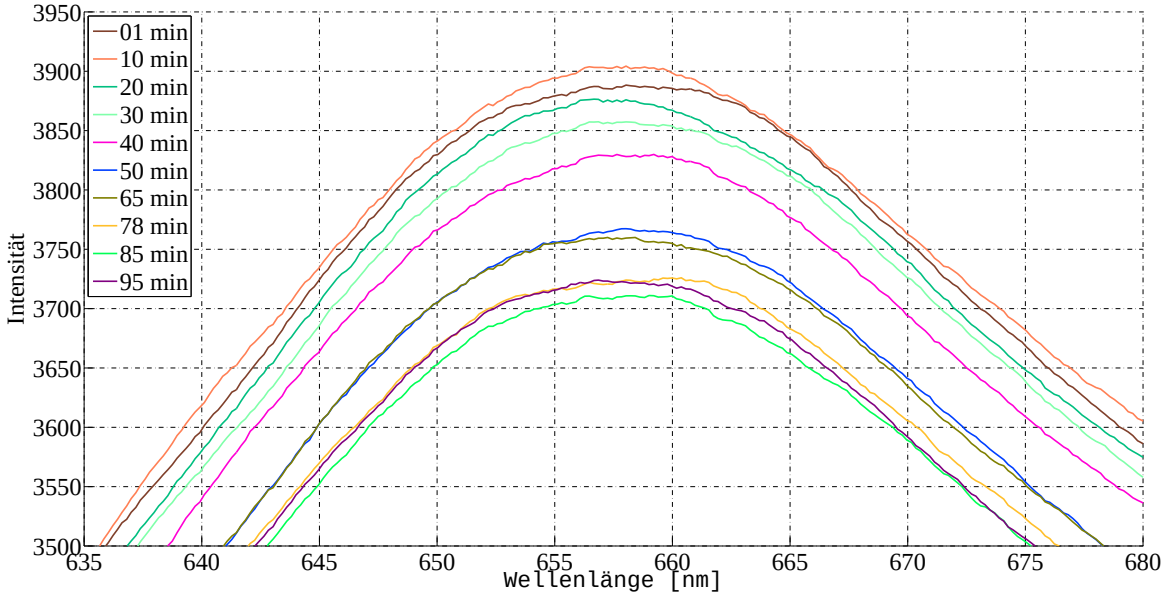

Abbildung 4.7.: Spektren der Halogenlampe zu verschiedenen Zeitpunkten nach dem Einschalten in vergrößerter Darstellung

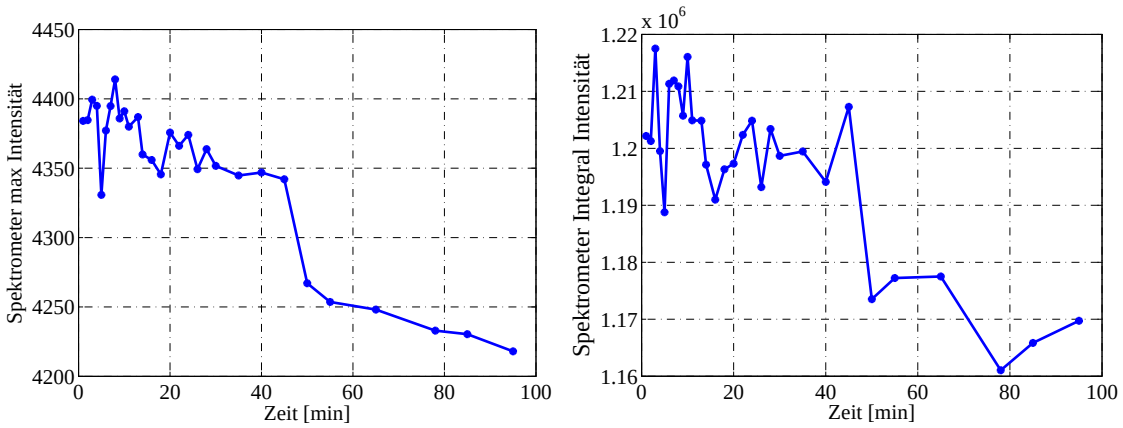

Abbildung 4.8.: Maximale Intensität bei ca. 660 nm und über alle Wellenlängen integrierte Intensität der Halogenlampe zu den Messzeitpunkten aus Abbildung 4.6.

Deutlicher wird der Effekt bei Messungen des Anlaufverhaltens der Halogenlampe mit einem Lichtsensor, welcher die Gesamtintensität des Lichtes misst. Abbildung 4.9 zeigt Messungen eines Einschaltvorganges der Halogenlampe mit dem später in Abschnitt 5.2.3 erläuterten Licht-/Frequenz-Wandler. Die Frequenzen wurden mit einem Multimeter 45 der Firma Fluke gemessen und über die serielle Schnittstelle des Messgerätes an einen PC übertragen und abgespeichert.

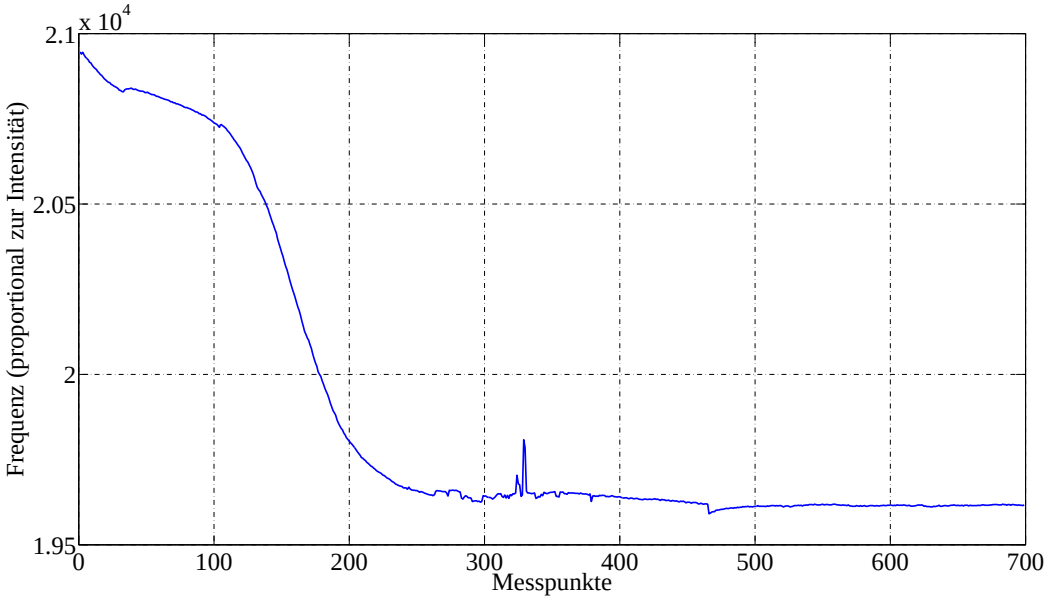

Abbildung 4.9.: Messungen der Frequenz vom Lichtsensor TAOS TCS3200 über die Zeit mit dem Multimeter Fluke 45 beim Anlauf der Halogenlampe. Die Werte wurden von dem Multimeter mit der konfigurierten Rate N = 5 übermittelt. Dies soll einer Messrate von circa einem Messwert pro Sekunde entsprechen. Das würde zu einer Gesamtdauer von 11 Minuten führen. Die gezeigte Messung erstreckte sich aber über mehr als eine halbe Stunde.

Auch diese Messung zeigt über einen gewissen Zeitraum einen Abfall der Lichtintensitäten. Die Messwerte schwanken aber im Gegensatz zu den Messungen mit dem Spektrometer kaum. Der Sprung bei Messwert 320 kommt durch ein kurzzeitiges Abkühlen des Gehäuses der Halogenlampe (durch Berührung) zustande.

Die Messungen mit dem Multimeter wurden mit der konfigurierten Übertragungsrate N = 5 an einen PC gesendet. Laut Datenblatt soll dies einer Senderate von circa einem Messwert pro Sekunde entsprechen [28]. Bei 700 Messwerten führt dies zu einer gesamten Messdauer von 11,67 Minuten. Die genauen Messzeiten wurden nicht aufgenommen, die Dauer betrug allerdings weit über 30 Minuten. Der Grund für die abweichende Senderate ist nicht bekannt. Bei Frequenzmessungen mit dem Multimeter Fluke 45 sind jedoch Verzögerungen bei der

Ermittlung der Messwerte festgestellt worden, welche in dieser Form bei Strommessungen nicht auftreten.

Als Konsequenz aus den Messungen ist zu ziehen, dass die Halogenlampe vor ihrem Einsatz vorgeheizt werden muss, damit die Messwerte nicht durch Intensitätsänderungen beim Anlauf beeinflusst werden.

4.2. Reflektometrie an Eisenphosphat und Lithium-Eisenphosphat als Feststoff

Um den Einfluss von Lithiumeinlagerungen in Eisenphosphat an dessen Reflexionseigenschaften zu untersuchen, wurden mit dem Spektrometer Reflexionsmessungen durchgeführt. Als Messobjekte dienten die in Abbildung 4.10 nicht lithinierten (1-3) und lithinierten (6 und 7) Eisenphosphatproben. Die lithinierten Eisenphosphatproben wurden entsprechend Abschnitt 3.1 durch die Versetzung mit Butyllithium hergestellt.

Bei Reflexionsmessungen werden die Proben durch eine Glasfaser angestrahlt. Das reflektierte Licht wird wiederum in eine Glasfaser eingekoppelt, welches anschließend im Spektrometer ausgewertet wird (Abb. 4.11 und 4.12).

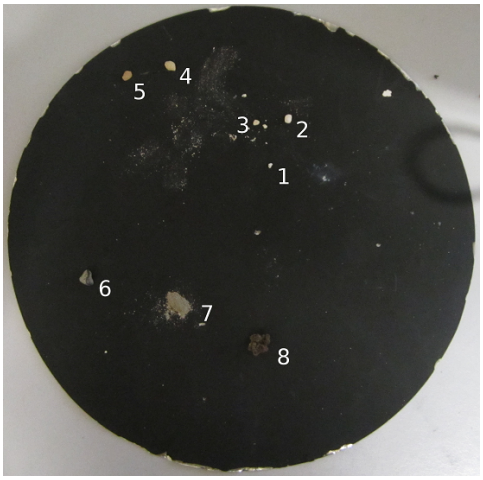

Abbildung 4.10.: Eisenphosphatproben, 1-3 unbehandelt (nicht lithiniert), 6 und 7 mit Butyllithium behandelt (lithiniert), 4, 5 und 8 unbehandelt (vermutlich leicht oxidiert)

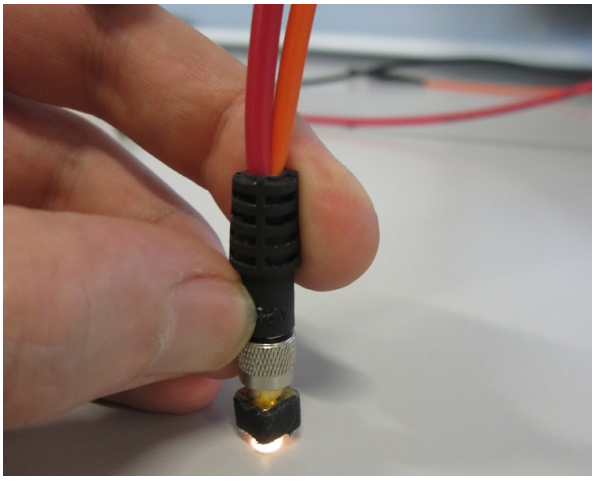

Abbildung 4.11.: Messvorrichtung für Reflexionsmessungen. Durch eine Faser wird ein Messobjekt angestrahlt. Das reflektierte Licht tritt in die zweite Faser ein, welches an einem Spektrometer gemessen wird.

Da durch die angewandte Messmethode leichte Intensitätsänderungen durch unterschiedliche Messaufbauten auftreten, müssen die Spektren normiert werden. Dabei diente eine Wellenlänge bei ungefähr 650 nm als Bezugsgröße. Hier wurde der Messwert durch sich selbst geteilt, sodass der Wert 1 erreicht wurde. Mit demselben Faktor sind die Linien desselben

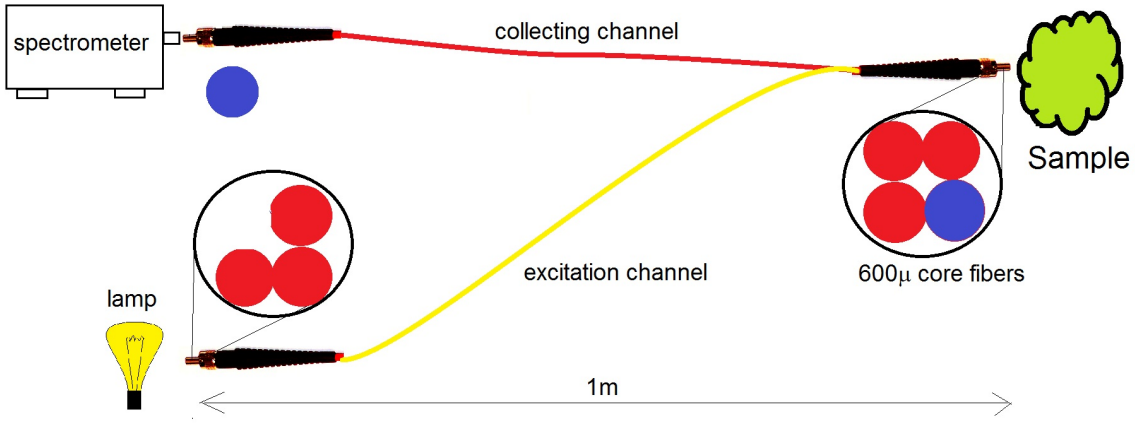

Abbildung 4.12.: Funktionsschema des Messkabels für die Reflexionsmessungen [25]. Licht wird durch drei Faserstränge auf eine Probe geleitet, dort reflektiert und über einen Faserstrang zum Spektrometer weitergeführt.

Spektrums skaliert worden. Damit wird die unterschiedliche Gesamtreflexion der Messung ausgeglichen.

Die vermutlich oxidierten Proben 4, 5 und 8 zeigen größere Abweichungen zu den restlichen Messungen. Aufgrund der vielen Messungen innerhalb des einen Graphen wurde ein weiteres Bild (4.14) erstellt, welches nur die nicht lithinierten Proben 2 und 3 sowie die lithinierten Proben 6 und 7 zeigt. Man erkennt, dass alle Proben einen ähnlichen Verlauf haben. So steigt die Intensität des reflektierten Lichtes beider lithinierten Proben bei 660-670 nm etwas länger an, als die der nicht lithinierten. Die dabei entstehende Differenz der Intensitäten bleibt weit bis in den Infrarotbereich bestehen.
Die in dem Diagramm dargestellten Graphen zeigen, dass sich Änderungen der Reflexionseigenschaften durch die Einlagerung von Lithium in Eisenphosphat einstellen. Diese Änderungen können mittels optischer Sensorik gemessen werden. Anhand dessen können Informationen über den Grad der Lithinierung und somit dem Ladezustand eines Lithium-Eisenphosphat-Akkumulators gewonnen werden.

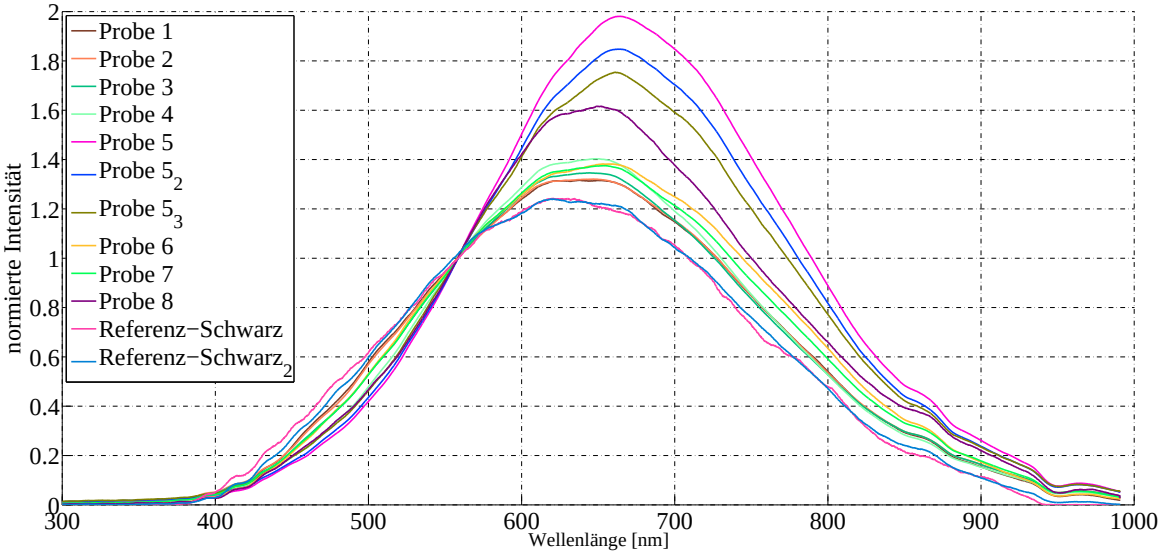

Abbildung 4.13.: Reflektionsmessungen von unterschiedlichen Eisenphosphatproben

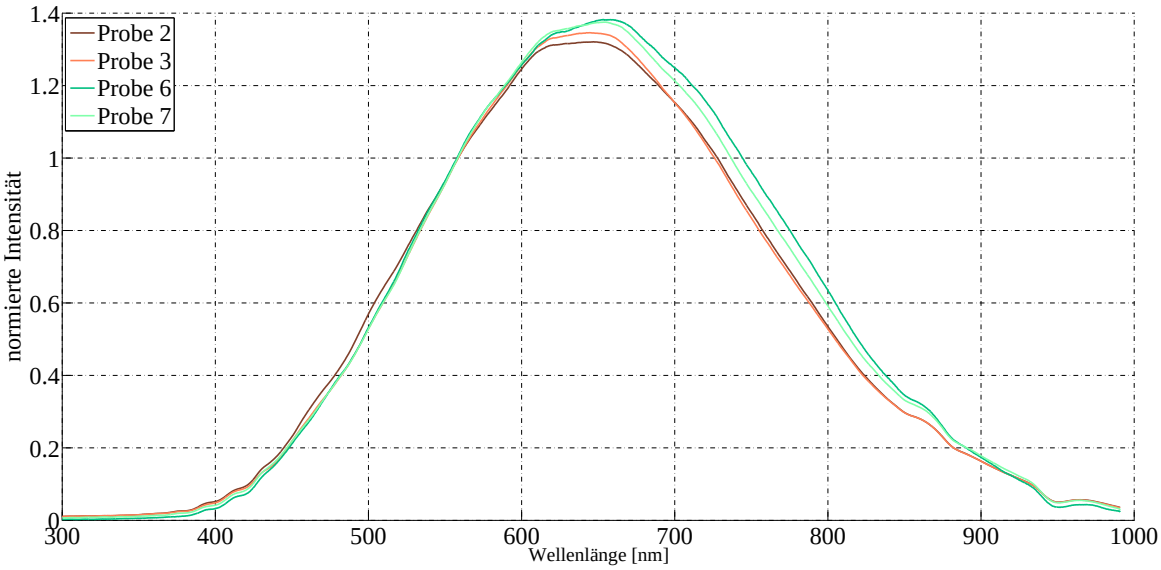

Abbildung 4.14.: Messungen an lithiniertem (Probe 6 und 7) und nicht lithiniertem (Probe 2 und 3) Material

4.3. Evaneszenzfeldspektroskopie zum Vergleich an Flüssigkeiten

Zur Überprüfung und Veranschaulichung des angedachten Messprinzips für die Eisenphosphat-Elektroden wurden Messungen an Methylenblau-Wassergemischen (Tinte) durchgeführt. Die Messungen wurden an drei unterschiedlich bearbeiteten Fasern durchgeführt. Die Fasern wurden in einem kleinen Reagenzglas aus Plastik (Abb. 6.9) hinein- und wieder herausgeführt, wodurch eine Biegung des Lichtleiters mit sehr geringem Radius (ungefähr 0,5 cm) entsteht. Bei solch einem Radius wird das geführte Licht teilweise aus der Faser ausgekoppelt. Die Faser 1 war unbehandelt. Die Faser 3 wurde einmal mit grobem Schmirgelpapier (200er Körnung) und anschließend mit feinerem Schmirgelpapier (1000er Körnung) an der Biegung abgeschliffen. Faser 6 wurde ausschließlich mit dem grobem Schmirgelpapier bearbeitet.

Bei den Messungen wurde zu Beginn ein Röhrchen mit blauer Tinte aufgefüllt. Diese Tinte wurde dann vollkommen entfernt und das Röhrchen wurde anschließend mit klarem Leitungswasser aufgefüllt. Zurückgebliebene Partikel des Blaus waren so ergiebig, dass dieser Prozess mehrere Male wiederholt werden musste, bis die Flüssigkeit im Röhrchen wieder vollkommen klar wurde. Nach jeder Auffüllung wurde eine Messung mit dem Spektrometer durchgeführt.

Abbildung 4.15 zeigt die Messungen mit der unbehandelten Faser. Es ist zu erkennen, dass bei einer Normierung auf circa 550 nm die Tinte das Licht im Infrarotbereich im Verhältnis weniger absorbiert, als die Wassergemische. Die Unterschiede sind aber gering.

Abbildung 4.16 zeigt die Spektren des ausschließlich grob abgeschmirgelten Lichtleiters. Auch hier zeigt sich die verhältnismäßig geringere Absorption und dadurch höhere Lichtintensität im Rot- und Infrarot-Bereich. Die Differenzen zwischen Tinte und den Tinte-Wasser-Gemischen sind aber stärker, als bei der unbehandelten Faser.

Die Messergebnisse der erst grob und anschließend fein abgeschmirgelten Faser in Abbildung 4.17 zeigen die größten Differenzen zwischen den einzelnen Messungen. Das Licht kann durch die feinere Bearbeitung besser aus der Faser aus- und auch wieder eintreten, wodurch der Einfluss der Flüssigkeit in dem Röhrchen auf das Licht stärker ist, als bei den beiden anderen Fasern. Diese Bearbeitungsmethode ist daher unter den gegebenen Rahmenbedingungen den anderen beiden vorzuziehen.

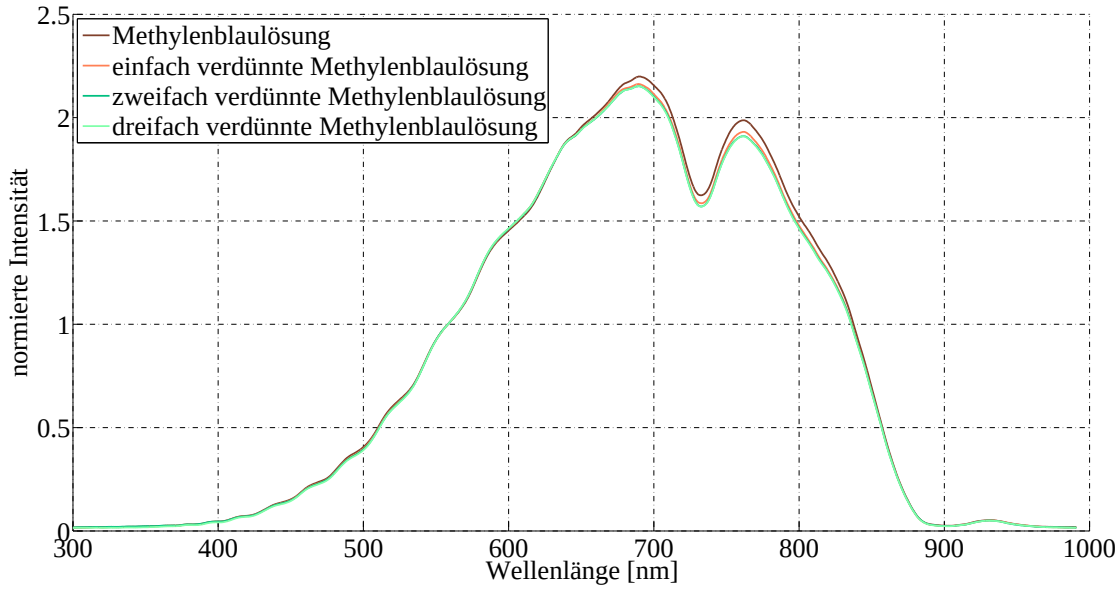

Abbildung 4.15.: Spektren der Vorversuche mit Faser 1 an einer Methylenblaulösung. Die Faser 1 ist unbehandelt. Die auf circa 550 nm normierten Spektren zeigen verhältnismäßig geringe Veränderungen bei Verdünnung der Methylenblaulösung.

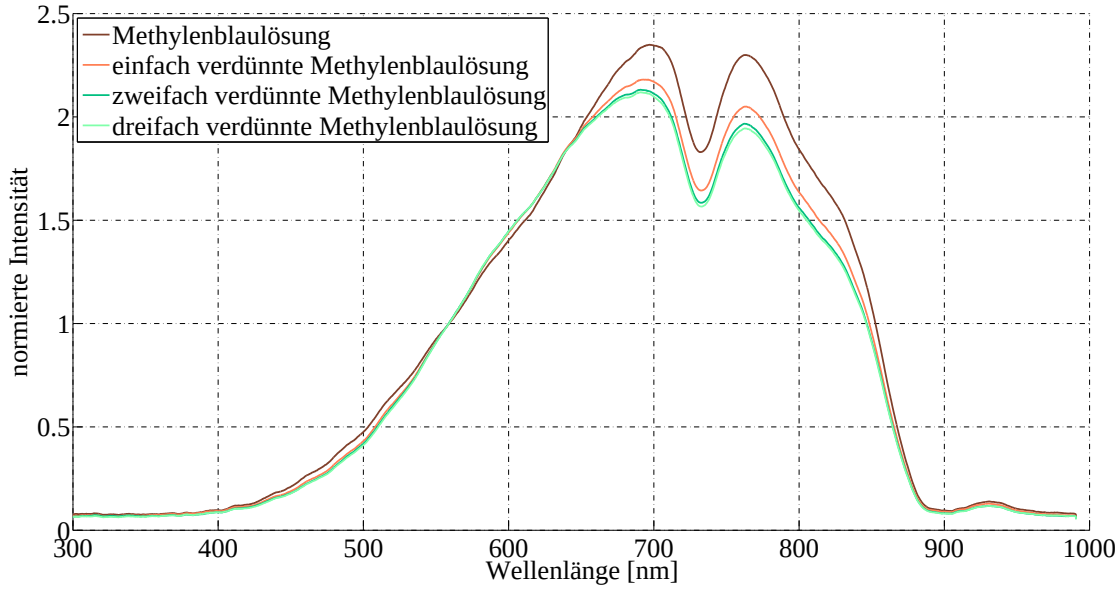

Abbildung 4.16.: Spektren der Vorversuche mit Faser 6 an einer Methylenblaulösung. Die Faser 6 wurde mit groben Schmirgelpapier an der Biegung abgeschmirgelt. Die auf circa 550 nm normierten Spektren zeigen im Vergleich zu Faser 1 deutlichere Veränderungen bei Verdünnung der Lösung.

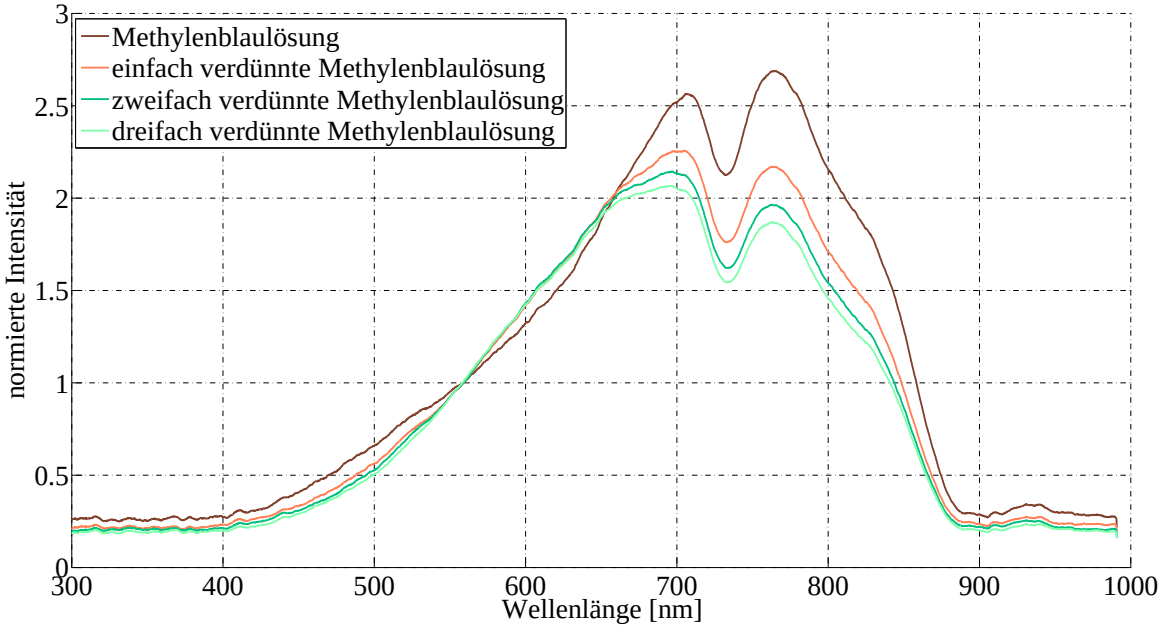

Abbildung 4.17.: Spektren der Vorversuche mit Faser 3 an einer Methylenblaulösung. Die Faser 3 wurde mit feinem Schmirgelpapier an der Biegung abgeschmirgelt. Die auf circa 550 nm normierten Spektren zeigen die deutlichsten Veränderungen im Spektrum bei Verdünnung der Lösung.

4.4. Fixierung und Präparation des Lichtleiters für den Lichtaustritt

Für gleichbleibende Bedingungen bei Messungen mit den Lichtleitern müssen diese befestigt und stabilisiert werden. Um die in Kapitel 3 gezeigte Führung des Lichtleiters zu gewährleisten, muss der Lichtleiter fixiert werden. Dies wurde durch bearbeitete Plastikschrauben realisiert.
Mit einer Schraube direkt an der Biegung wird sichergestellt, dass alle Lichtleiter im selben Radius gebogen werden. Sie dient auch als erste Fixierung. Um die durch die Biegung auf Spannung stehenden Fasern weiter zu stabilisieren, wird eine weitere Schraube eingesetzt, welche die hin- und zurückführenden Fasern festklemmt.

Abbildung 4.18 zeigt einen solchen Aufbau. Zu sehen sind dort drei nebeneinander aufgereihte Lichtwellenleiter. Die Lichtleiter wurden zur besseren Fixierung noch zusätzlich mit Klebeband auf einer Glasplatte angebracht.

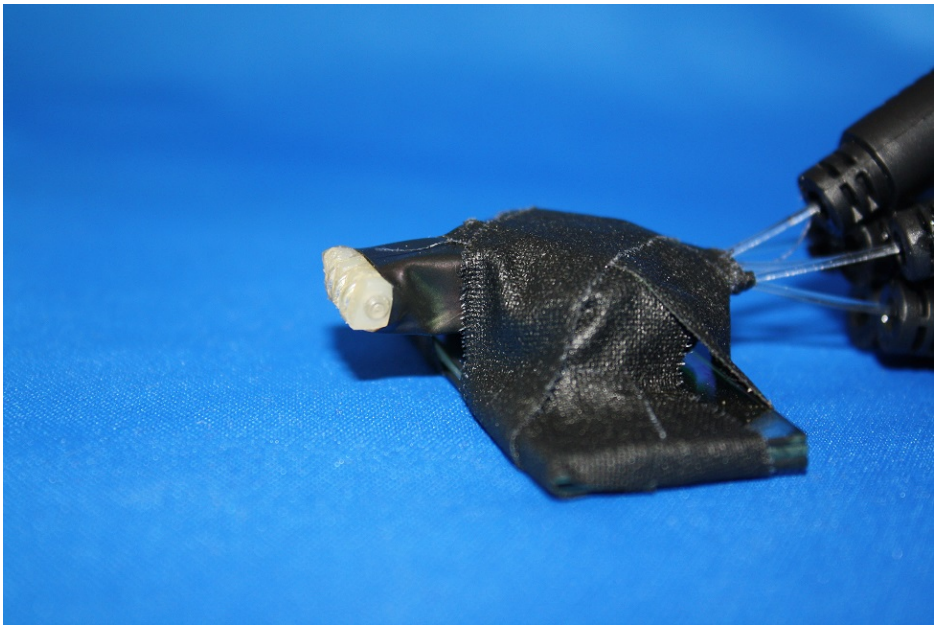

Abbildung 4.18.: Präparation und Fixierung von Lichtwellenleitern für den Lichtaustritt

Diese Fasern können an der Biegung weiter bearbeitet werden, wie es bereits in dem vorangegangenem Kapitel 4.3 geschehen ist.

4.5. Eisenphosphat-Sputterbeschichtung eines Lichtleiters

Beim Sputtern handelt es sich um eine Beschichtungsmethode, welche verwendet wurde, um dünne Lithium-Eisenphosphatschichten auf Lichtleiter aufzubringen.
Es existieren unterschiedliche Varianten der Sputtertechnik. Im Rahmen dieser Bachelorarbeit wurde eine Hochfrequenzsputteranlage eingesetzt. Das Funktionsprinzip ist in Abbildung 4.19 dargestellt.
Unter einem einstellbaren Druck werden die sich in einer Kammer befindlichen Gasatome (z.B. Argon) zum Schwingen angeregt. Dadurch trennen sich die Elektronen vom Atom, wodurch sich Plasma bildet.
Durch das Wechselfeld innerhalb der Kammer treffen die Gasionen auf das sogenannte Target. Dort werden durch die Impulse der auftreffenden Gasionen Atome des Targets herausgeschlagen. Diese bewegen sich durch die Kammer hindurch und schlagen wiederum auf einem Substrat ein. Dadurch wird das Substrat oberflächlich mit Targetmaterial beschichtet.

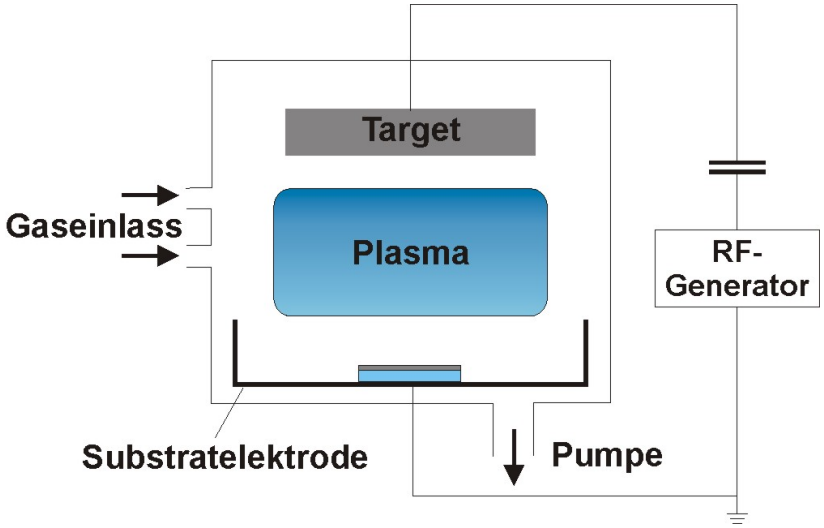

Abbildung 4.19.: Schematische Darstellung einer Hochfrequenzsputteranlage [29]

Zur Sputterbeschichtung der Lichtleiter wurde ein Target aus Lithium-Eisenphosphat eingesetzt. Die Beschichtungsmethode dient dazu, eine möglichst glatte Lithium-Eisenphosphatschicht auf dem Lichtwellenleiter aufzutragen. Wie in Abschnitt 3 erläutert, soll das Eisenphosphat den Mantel ersetzten und dessen Teilaufgabe der Lichtleitung übernehmen. Dafür muss die Schicht möglichst plan sein, da das Licht sonst teilweise an der

unebenen Oberfläche unregelmäßig reflektiert und gestreut wird und somit für Messungen am Ende des Lichtleiters verloren geht.

Abbildung 4.20 zeigt einen solchen beschichteten Lichtleiter. Bei dem Vorversuch sollte geprüft werden, ob das Beschichten einer POF-Faser möglich ist. Die Eisenphosphat-Beschichtung ist an der gelblichen Oberfläche (links) zu erkennen. Weiter rechts im Bild wurde die Faser während des Beschichtungsverfahrens abgeklebt und zeigt somit die unbeschichtete Faser.

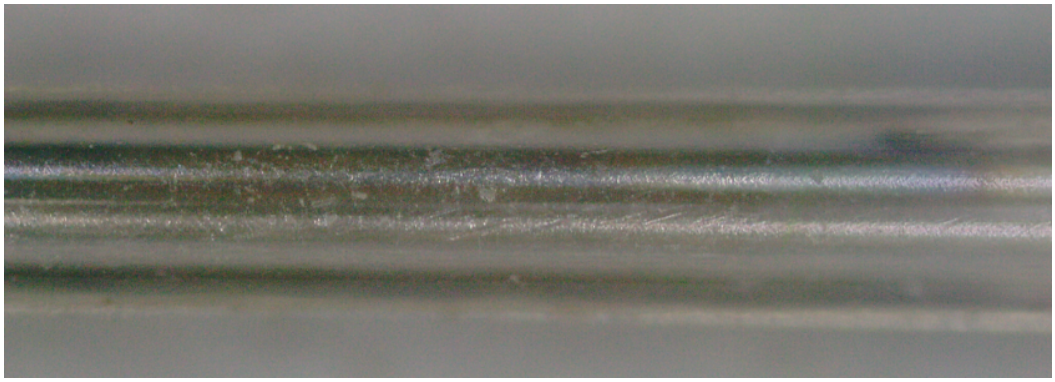

Abbildung 4.20.: Mit Sputterverfahren beschichteter (links) und unbeschichteter (rechts) Lichtwellenleiter

Abbildung 4.21 zeigt zum Vergleich eine beschichtete (rechts) und unbeschichtete Separatorfolie.

Abbildung 4.21.: Separatorfolie einer Batterie unbeschichtet (links) und mittels Sputterverfahren beschichtet (rechts)

An dieser Separatorfolie wurden Tranmissionsmessungen durchgeführt. Die Folie wurde also an der einen Seite mit der breitbandigen Halogenlampe durchleuchtet. Das transmittierte Licht wurde mit dem Spektrometer aufgenommen.

In Abbildung 4.22 zeigt die optische Untersuchung der beschichteten Folie eine Verschiebung des Spektrums zu höheren Wellenlängen im Vergleich zur unbeschichteten Folie.

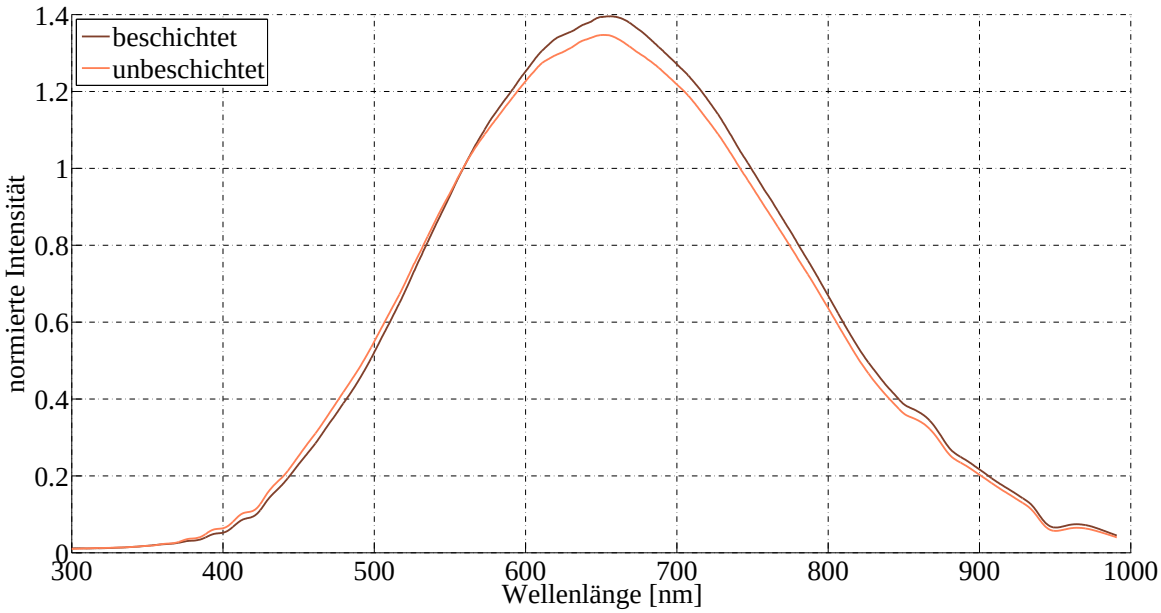

Abbildung 4.22.: Transmissionsmessungen an unbeschichteter und beschichteter Separatorfolie

5. Laboraufbau eines vereinfachten LED-Reflektometers

Das Ziel des in dieser Bachelorarbeit entworfenen Laboraufbaus eines LED-Reflektometers ist es, die in Abschnitt 3 erläuterten Messungen an Lithium-Eisenphosphat-Kathoden durchzuführen. Das von LEDs emittierte Licht soll durch einen Lichtleiter geleitet und für Farbmessungen an den Kathoden verwendet werden. Das durch die Elektrode beeinflusste Licht soll in einem Sensor ausgewertet werden. Im folgenden Kapitel wird auf die Auswahl von Lichtquellen, Sensoren und Stecker eingegangen, welche im noch folgenden Platinenlayout verwendet werden sollen. Anschließend wird das realisierte Platinenlayout sowie die Mikrocontroller- und PC-Software vorgestellt.

5.1. Erprobung und Auswahl von Stecker, Lichtquellen und Sensoren

Die folgenden Tests sind für die Realisierung eines LED-Reflektometers notwendig, da im Handel solche Geräte mit speziellen Anforderungen nicht erhältlich sind.
Zu kaufende Lichtquellen, welche zur Einkopplung in Lichtwellenleitern mit entsprechenden Buchsen versehen sind, emittieren meist nur Licht mit einem Intensitätsmaximum bei nur einer Wellenlänge. Eine solche Lichtquelle ist zum Beispiel das Bauteil TORX173 von Toshiba [30].
Eine weitere mögliche Lichtquelle wäre die für Reflexionsmessungen verwendete breitbandige Halogenlampe (siehe Abschnitt 4.1.1). Bei der Messplatine sollen jedoch günstige Bauteile zum Einsatz kommen. Außerdem ist es vorgesehen, bei mehreren Wellenlängen Messungen durchführen zu können.
Da es also keine entsprechenden Bauelemente gibt, mussten Lichtquellen und auch Lichtsensoren in Buchsen untergebracht werden, an denen Lichtwellenleiter mit kompatible Stecker angebracht werden können.
Diese Bauteile werden in den nachfolgenden Kapiteln erläutert.

5.1.1. Stecker

Es gibt verschiedene Stecksysteme für Lichtwellenleiter aus unterschiedlichen Anwendungsgebieten. Sie unterscheiden sich prinzipiell in ihren Anschlusstechniken und ihren Ausmaßen. Der Toslink-Stecker, welcher meist zur optischen Audioübertragung verwendet wird, ist ein einfaches Stecksystem und stellt daher die mechanisch instabilste der in Abbildung 5.1 gezeigten Steckverbindungen dar. FSMA-Stecker werden häufig bei Laborgeräten, wie dem in Kapitel 4.1 beschriebenen Spektrometer, verwendet. Es handelt sich dabei um ein stabiles Schraubsystem. Der ST- und LC-Stecker werden häufiger in der Industrie eingesetzt. Sie werden mit einem Bajonett-, beziehungsweise mit einem Klick-Verschluss verbunden.

Abbildung 5.1.: Toslink-Stecker und Buchse (1), FSMA-Stecker und Buchse (2), ST-Stecker und Buchse(3), LC-Stecker und Buchse(4) [31] [32].

Bei der Realisierung des LED-Reflektometers wurden die Toslink-Stecker verwendet. Die Verbindungen sind zwar nicht so stabil, wie die der anderen aufgeführten Systeme, aber sie reichen für den Einsatz als stationäres Messgerät vollkommen aus. Der entscheidende Vorteil, insbesondere gegenüber den metallischen Buchsen und Steckern, ist die leichte Bearbeitbarkeit. Diese ist sehr wichtig, da Sensoren und LEDs in den Buchsen unterzubringen sind. Dafür sind die Toslink-Buchsen bestens geeignet. Sie lassen sich auseinandersägen, ausbohren und die Fassungen für die Stecker lassen sich einseitig entfernen. Damit kann leicht Platz für LEDs und Sensoren geschaffen werden. Des Weiteren lassen sich die Buchsen mit Klebstoff stabil auf Platinen befestigen. Für den Laboraufbau sind sie daher eine sehr gute Wahl.
Sollten stabilere Steckverbindungen notwendig sein, zum Beispiel für den Sensorbetrieb in Fahrzeugen, bieten sich unter anderem LC-Stecker und Buchsen an. Diese bestehen auch aus Kunststoff und lassen sich daher ebenfalls gut bearbeiten.

5.1.2. Lichtquellen

Da es keine fertigen Sendemodule für Lichtwellenleiter mit mehreren verschiedenen Lichtquellen gibt, müssen diese Lichtquellen zur Einkopplung in einen Lichtleiter selbst hergestellt werden.
Es gibt zwar LEDs zu kaufen, welche für den ähnlichen Einsatz der bereits erläuterten Pulsoxymetrie gedacht sind [33]. Diese sind aber aufgrund von Kopplungsverlusten und der angedachten Messmethodik zu schwach. Daher wurde bei der Auswahl der LEDs darauf geachtet, dass diese möglichst intensiv sind und einen möglichst geringen Abstrahlwinkel besitzen. In das später vorgestellte Platinenlayout wurden verschiedene Farbkombinationen eingesetzt, welche einen flexiblen Einsatz des Messplatine erlauben.

Die eingesetzten LEDs wurden mit dem Spektrometer einzeln ausgemessen und für jeden Toslink-Block in einem Graph dargestellt. Mit einem Toslink-Block sind die LEDs gemeint, welche sich hinter einer Toslink-Buchse verbergen. Die LEDs wurden mit den jeweiligen Nennströmen betrieben. Die Lichtleiterverbindungen von den Leuchtdioden wurden für jeden Block so gelegt und gesteckt, dass das Spektrometer nicht übersteuert wird. Zur Minderung der Lichtintensität wurden zum Beispiel mehrere POF-Leiter nacheinander mit Toslink-Buchsen aus Abbildung 5.1 miteinander verbunden. Durch Kopplungsverluste an den Buchsen erfährt das durch die Fasern geführte Licht eine vergleichsweise starke Dämpfung seiner Intensität.

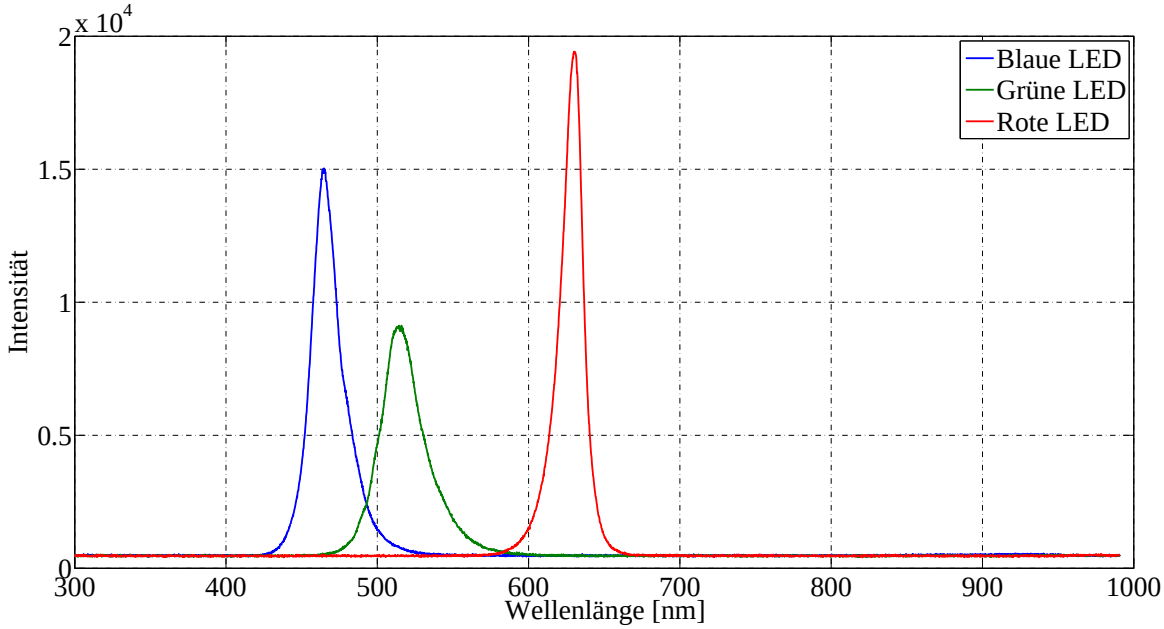

Abbildung 5.2.: 5mm-RGB-LED LED-50RGB-CA von kt-eletronic

Die RGB-LED, dargestellt in Abbildung 5.2, zeigt Intensitätsmaxima bei 465 nm, 514 nm und 630 nm. Laut Datenblatt liegen die Maxima bei 470 nm, 525 nm und 630 nm [34]. Es zeigen sich also bei der blauen und der grünen LED leichte Abweichungen, zumindest in den Maxima der Wellenlängen.

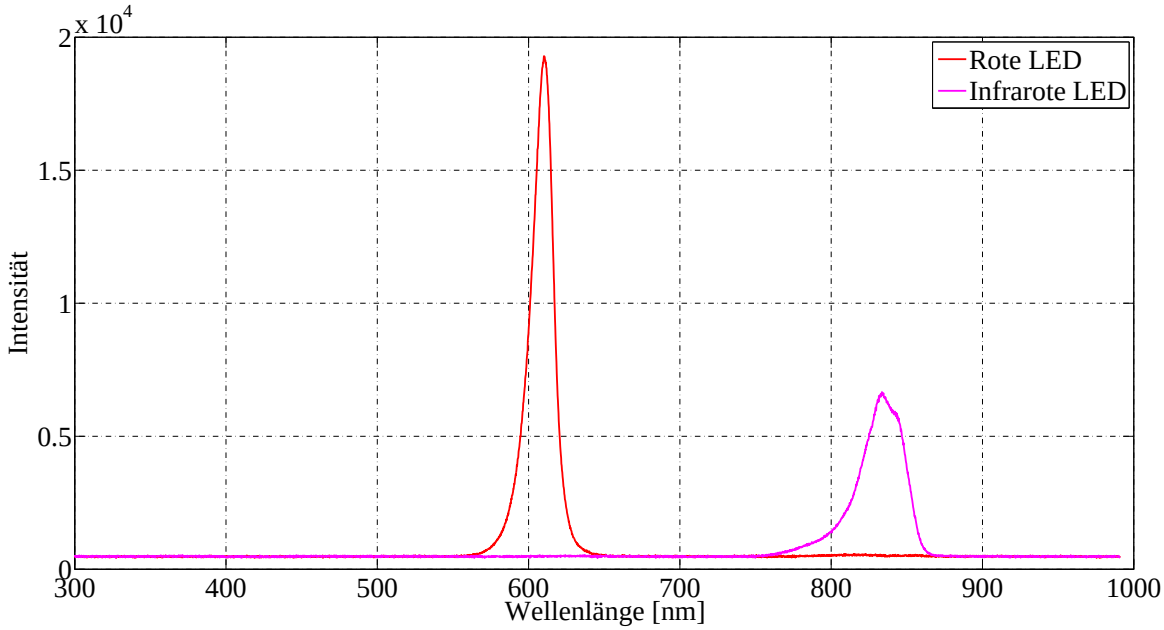

Abbildung 5.3.: Orange 3 mm LED WP710A10SEC/J4 von Kingbright und infrarote 3 mm LED SFH4350 von Osram

Die in Abbildung 5.3 dargestellten Spektren der beiden einzelnen 3 mm LEDs haben Intensitätsmaxima bei 610 nm und 833 nm. Laut Datenblatt sollen sie bei 605 nm [35] und 850 nm [36] liegen.

In Abbildung 5.4 konnten die Messungen für das rote und infrarote Licht nicht mit demselben Aufbau der Lichtleiterverbindung durchgeführt werden. Bei gleicher Verlegung und Verbindung mit denselben Fasern und Toslink-Buchsen wurde das infrarote Licht mit dem Spektrometer selbst dann nicht sichtbar, wenn bei Verwendung der roten LED das Spektrometer kurz vor der Übersteuerung war. Daher wurden für die Messung der infraroten LED weniger POF-Fasern miteinander verbunden, als bei der Messung mit der roten LED. Da die LEDs dadurch unterschiedlich stark gedämpft wurden, sind die Intensitäten in diesem Bild nicht vergleichbar. Das Intensitätsmaximum des gemessenen Spektrums der roten LED liegt bei ungefähr 623 nm, sehr nah an dem im Datenblatt angegebenen Maximum von 617 nm [37]. Die infrarote LED emittiert Licht überraschender Weise bei zwei verschiedenen Wellenlängen, mit Maxima bei 848 nm und 949 nm. Laut Datenblatt soll das abgestrahlte Licht jedoch nur ein Maximum bei der Wellenlänge 950 nm besitzen [38]. Das um die Wellenlän-

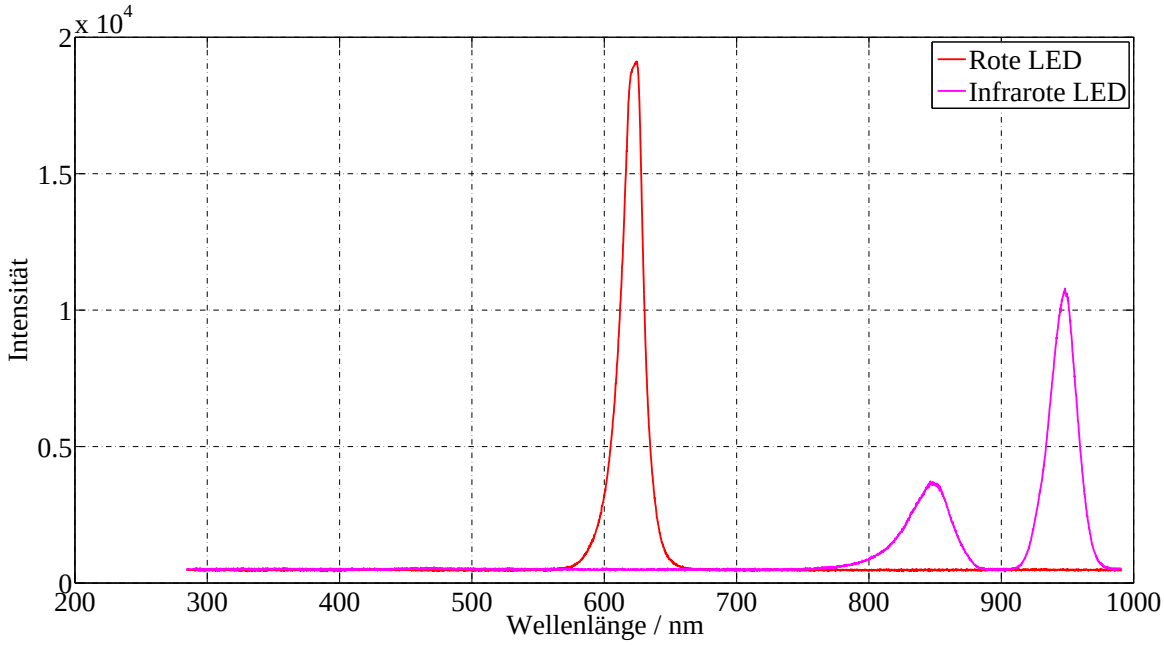

Abbildung 5.4.: Rote SMD LED LA E63F-EBGA-24-3A4B-Z und infrarote SMD LED SFH4248-Z von Osram

ge 848 nm herum emittierte Licht wird im Datenblatt der LED nicht erwähnt und auch nicht dargestellt.

Wichtig für Messungen ist, dass die LEDs möglichst konstant in ihrer Wellenlänge und Intensität sind. Faktoren, welche dies beeinflussen sind unter anderem die Temperatur und der Strom, welcher durch die LEDs fließt. Eine Möglichkeit, die Eigenerwärmung der LEDs zu reduzieren ist, sie in Pulsen zu betreiben. Dadurch erhält die LED die Möglichkeit sich im ausgeschaltetem Zustand abzukühlen. Des Weiteren muss der Strom möglichst konstant gehalten werden, da die Lichtintensität direkt mit dem fließenden Strom zusammenhängt.

5.1.3. Sensoren

Sensoren zur Messung von Lichtintensität beruhen auf dem in Kapitel 2.1.7 erläuterten photoelektrischen Effekt.
Es existieren unterschiedliche Bauelemente, mit welchen sich die Intensität des einstrahlenden Lichtes messen lassen. Getestet wurden Photowiderstände, Photodioden und Phototransistoren.

Photowiderstände verändern aufgrund von Lichteinstrahlungen ihre Leitfähigkeit. In einer einfachen Schaltung wird ein Photowiderstand mit einem festen Widerstand in Reihe als

Spannungsteiler betreiben. Durch den sich ändernden Widerstand des Photowiderstandes kann zum Beispiel am festen Widerstand eine zur Lichtintensität proportionale Spannung gemessen werden.
Photodioden erzeugen einen Photostrom bei Lichteinfall. Eine einfach zu realisierende Schaltungen ist in Abbildung 5.5 gezeigt. In der links dargestellten Schaltung wird die Photodiode als Photoelement eingesetzt. Der proportional zur Lichteinstrahlung erzeugte Photostrom I_P sorgt über den Widerstand R_L für die Spannung V_{OUT}:

$$V_{OUT} = I_P \cdot R_L \tag{5.1}$$

Ein entscheidender Nachteil dieser Schaltung ist der von Photodioden in der Regel sehr geringe gelieferte Photostrom, welcher sich im μA-Bereich bewegt.

In der in Abbildung 5.5 rechts gezeigten Schaltung ist die Photodiode in Sperrrichtung zu VCC geschaltet. Durch auf die Photodiode einstrahlendes Licht wird proportional dazu ein Strom I_P erzeugt. Auch hier gilt die Gleichung 5.1. Im Vergleich zum Betrieb als Photoelement ist aber der Strom I_P größer, woraus auch größere Spannungen über dem Widerstand R_L resultieren.

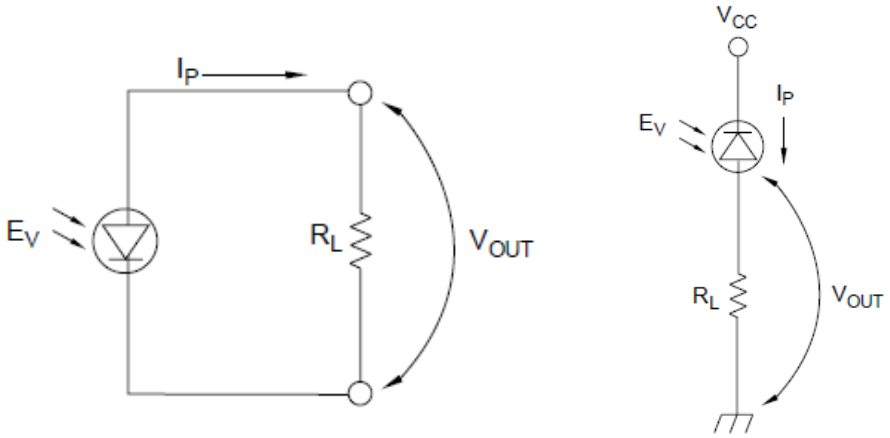

Abbildung 5.5.: Photodiode in Photoelement- und Photodiodenbetrieb [39]

Ein in kommerziellen Schaltungen weiter verbreiteter Aufbau ist der in Abbildung 5.6 gezeigte Betrieb als Transimpedanzverstärker. Dieser wandelt den Strom in eine Spannung am Ausgang des Operationsverstärkers. Auch bei dieser Schaltung gilt für die Ausgangsspannung die Gleichung 5.1.

Große Vorteile dieser Schaltung gegenüber den beiden zuvor gezeigten sind der geringere Einfluss vom Dunkelstrom und dessen große Linearität [39]. Dafür ist jedoch beim Entwurf

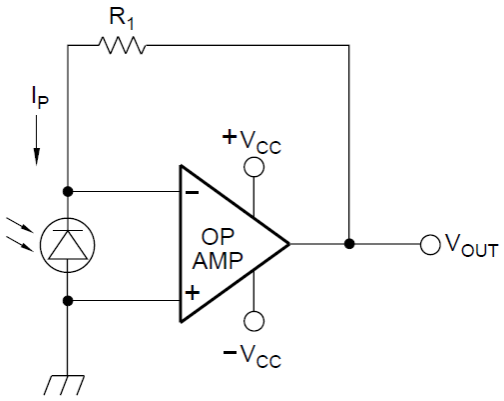

Abbildung 5.6.: Transimpedanzverstärker - Strom-Spannungs-Wandler [39]

darauf zu achten, dass die Schaltung stabil und besonders rauscharm ist, was durchaus eine besondere Herausforderung darstellen kann [40].
Phototransistoren erzeugen, wie Photodioden, einen Strom. Dieser ist bei einer gleichgroßen lichtempfindlichen Fläche jedoch weitaus höher [41]. Sie können prinzipiell mit denselben Schaltungen wie Photodioden verwendet werden.

Neben diesen Bauelementen gibt es auch integrierte Schaltungen, meist Photodioden mit dem erwähnten Transimpedanzverstärker. Es gibt Varianten, welche den Strom in eine Spannung wandeln und anschließend mithilfe eines ADCs und Mikrocontrollers ausgewertet werden können. Neben dieser Umsetzung gibt es aber auch Licht-/Frequenz-Wandler oder seriell ansprechbare Lichtsensoren.

Im Zuge der Auswahl eines geeigneten Sensors wurden die folgenden Varianten getestet:

- Photowiderstand A 1060 als Spannungsteiler [42]
- Photodiode OSRAM SFH203 geschaltet entsprechend Abbildung 5.5 (2) mit $VCC = 5V$, $RL = 10MOHM$ [43]
- Phototransistor OSRAM SFH309 wie in Abbildung 5.5 (2) mit $VCC = 5V$, $RL = 20MOHM$ [44]
- RGB Photodiode Kingbright KPS-5130PD7C geschaltet entsprechend Abbildung 5.5 (2) mit $VCC = 5V$, $RL = 10MOHM$ [45]
- Licht-/Spannungs-Wandler TAOS TSL250R (Photodiode mit TIA) mit $VCC = 5,5V$ [46]
- Licht-/Frequenz-Wandler TAOS TCS3200D mit $VCC = 5V$ [47]

Bei den Tests wurden eine LED und mit einem 50 Ω Widerstand in Reihe geschaltet. Die Spannung (Spannungsquelle: Manson HCS 3202), gesteuert mit einem PC über eine RS232-Schnittstelle, wurde mit den kleinstmöglichen Sprüngen (ca. 0,1 V) erhöht. Über zwei Multimeter (Fluke 45) wurden die Spannungen über Widerstand und LED, sowie der jeweilige Sensorwert am PC aufgezeichnet.

Die erzeugten Lichtspektren und die über alle Wellenlängen integrierten Intensitäten werden in Abbildung 5.7 gezeigt. Sich linear verhaltende Sensoren sollten sich ähnlich verhalten, wie die Werte der integrierten Spektren bei steigender Spannung. Zu beachten ist jedoch, dass die optische Empfindlichkeit der Bauelemente für unterschiedliche Wellenlängen des Lichtes nicht konstant ist. In Abbildung 5.7 ist zu sehen, dass sich das Intensitätsmaximum des emittierten Lichtes bei steigender Intensität ein wenig zu höheren Wellenlängen hin verschiebt und dass bei höheren Intensitäten sich das Band der abgestrahlten Wellenlängen verbreitert.

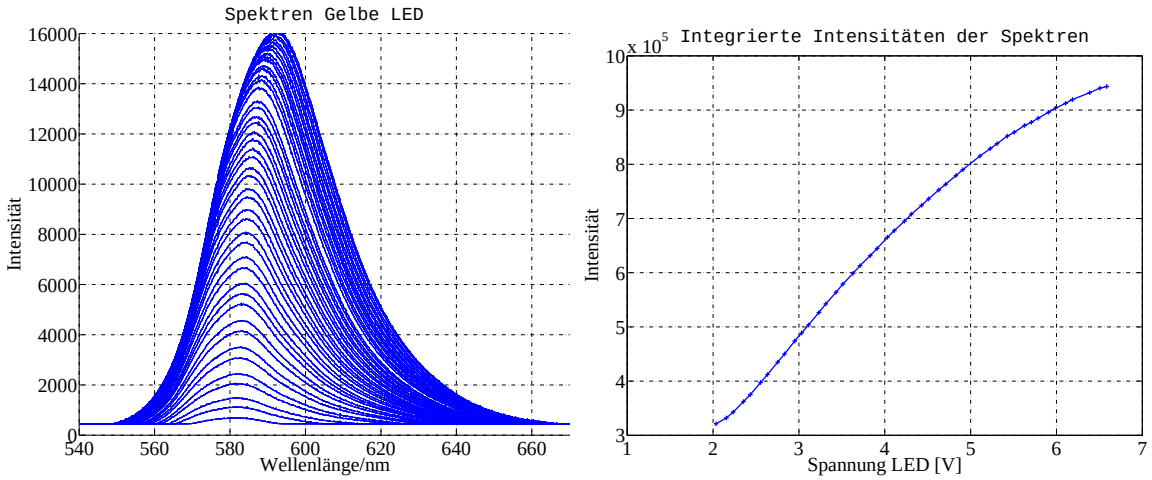

Abbildung 5.7.: Spektren und integrierte Intensitäten der Spektren der für die Sensortests verwendeten LED. Der Diodenstrom wurde durch einen Vorwiderstand begrenzt. Rechts ist die integrierte Gesamtintensität über die Spannung an LED und Vorwiderstand dargestellt.

Der Photowiderstand in Abbildung 5.8 zeigt als einziger Sensor einen logarithmischen verlauf. Die Spannung wurde dabei über einen festen 10 MOhm Widerstand abgegriffen. Wird anstelle dessen einfach nur der Widerstand des A1060 gemessen, so ergibt sich eine invertierte Kennlinie.
Die in der selben Abbildung gezeigte Photodiode zeigt dagegen einen ähnlichen Verlauf, wie integrierten Messwerte des Spektrometers in Abbildung 5.7.

Auch der Phototransistor zeigt einen Verlauf entsprechend der berechneten Werte aus Abbildung 5.7. Der Licht-/Spannungswandler wird offensichtlich übersteuert und geht in seine

Begrenzung. Daher ist dieser Sensor nur für vergleichsweise geringe Lichtintensitäten verwendbar.

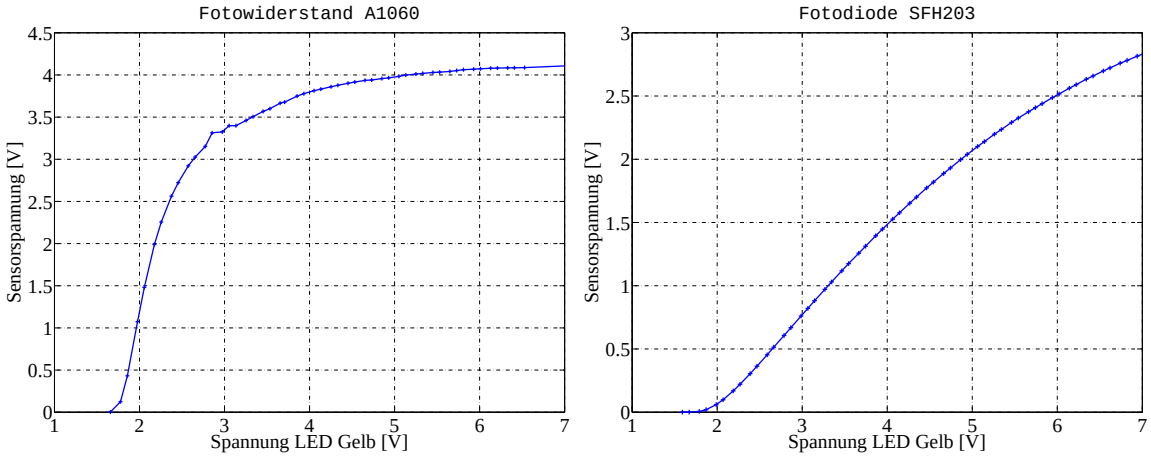

Abbildung 5.8.: Photowiderstand A1060 betrieben als Spannungsteiler und Photodiode betrieben entsprechend der zweiten Schaltung in Abbildung 5.6

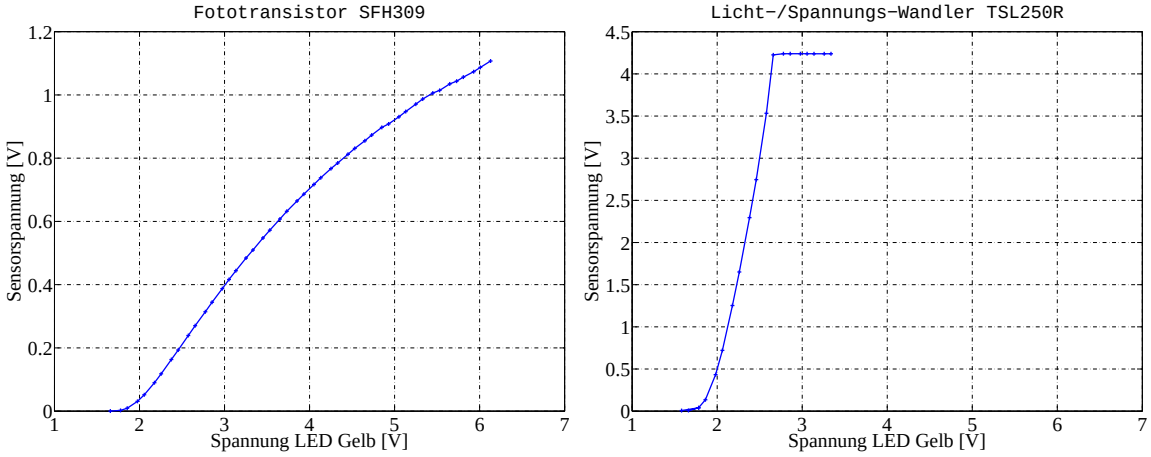

Abbildung 5.9.: Phototransistor entsprechend Schaltung 2 aus Abbildung 5.5 und Licht-/Spannungs-Wandler

Die RGB-Photodiode, dessen Messdaten in Abbildung 5.9 zu sehen sind, stellt eine Besonderheit unter den hier getesteten Sensoren dar. Vor den Photodioden sind Farbfilter angebracht. An einer Diode wird eher rotes Licht, bei einer weiteren mehr grünes und bei der letzten größtenteils blaues Licht durchgelassen. Die Amplitude der roten Photodiode ist am größten. Dies kommt einerseits durch die Eigenschaften der LED zustande, welche ein Maximum bei einer Wellenlänge von circa 590 nm besitzt. Andererseits hat die Photodiode aber auch eine höhere Empfindlichkeit bei eben dieser Wellenlänge.

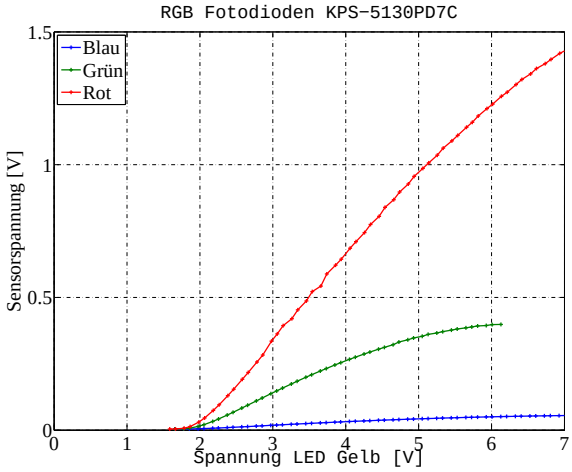

Abbildung 5.10.: RGB-Farbsensor KPS-5130PD7C

Von dem Sensor TAOS TCS3200D ist hier leider kein Plot zu sehen, obwohl dieselben Messungen mit diesem Licht-/Frequenz-Wandler durchgeführt wurden. Bei der Aufnahme der Messwerte mit dem Messgerät Fluke 45 kam es jedoch zu vielen Fehlern, wodurch diese letztendlich unbrauchbar wurden. Das Multimeter scheint Probleme bei Frequenzmessungen und gleichzeitiger Übertragung dieser Messwerte über die RS232-Schnittstelle zu haben. So wurden sehr häufig Kommas in den Messwerten weggelassen, wodurch die Messwerte zwischen 3,012 und 3000 schwankten. Letztendlich funktioniert dieser Sensor aber wie der hier demonstrierte RGB-Sensor, bietet jedoch zusätzlich die Möglichkeit, ungefilterte Photodioden zur Messung heranzuziehen.
Aufgrund dieser Flexibilität wurde bis auf Weiteres der Farbsensor TCS3200D eingesetzt, welcher in Abschnitt 5.2.3 näher erläutert wird.

5.2. Schaltungs- und Platinenlayout als Basis für den Anschluss der Lichtleiter

Die entworfene Platine soll zur Durchführung von Messungen an LiFePO4 Kathoden dienen. In Abbildung 5.11 ist eine bestückte Platine gezeigt. Da noch keine Messungen an Lithium-Eisenphosphat erfolgten, welche aussagekräftig genug sind, um daraus die sinnvollsten LED-Kombinationen zu ermitteln, wurde der Laboraufbau möglichst flexibel gestaltet. Das bedeutet, dass der Aufbau möglichst variabel hinsichtlich der verwendeten LEDs und Sensoren sein soll. Daher sind Montagepunkte für verschiedene LED-Varianten vorgesehen. Es können RGB-LEDs, DUO-LEDs, zwei einzelne 3 mm-LEDs und zwei SMD-LEDs (PLCC4) eingesetzt werden.

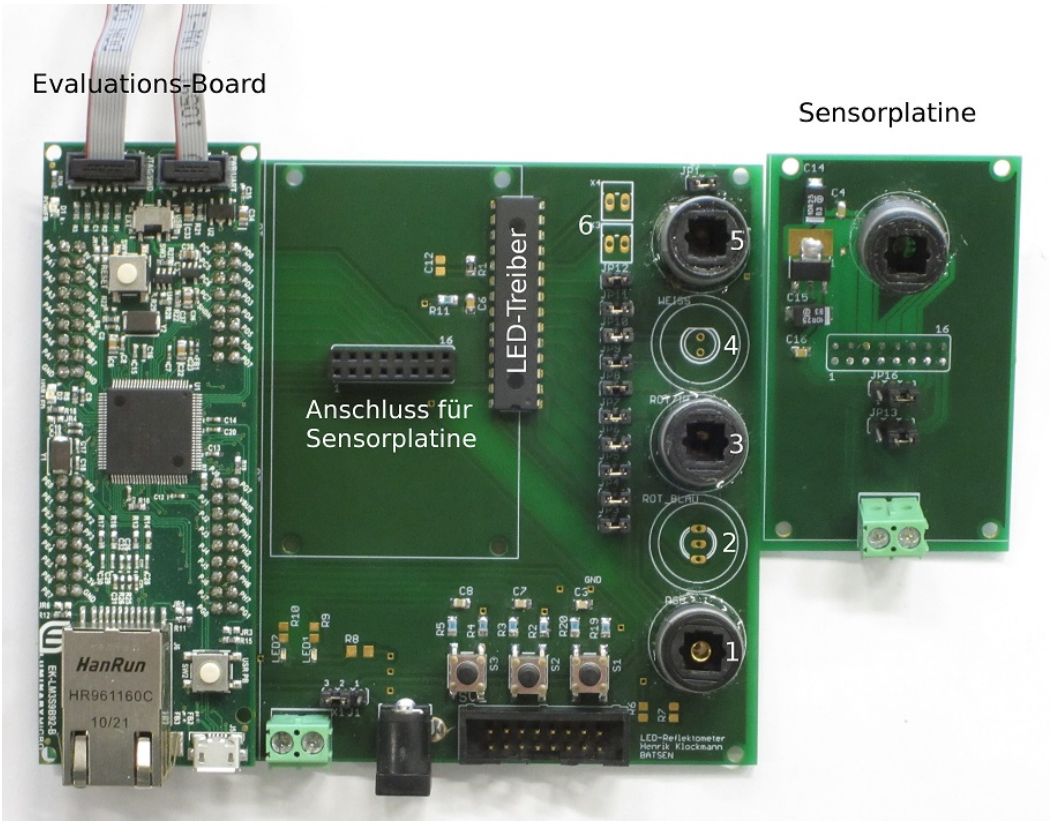

Abbildung 5.11.: Realisierter Aufbau des LED-Reflektometers. Links ist das Evaluations-Board des Mikrocontrollers LM3S9B92 von TI aufgesetzt. Daneben ist der Steckplatz für die Sensorplatine. Weiter rechts sind der LED-Treiber und unterhalb der Toslink-Stecker unterschiedliche LED-Kombinationen verbaut. Neben der Hauptplatine ist die Sensorplatine zu sehen

Zum Betreiben der LEDs wird ein in Abschnitt 5.2.2 näher erläuterter LED-Treiber eingesetzt. Dieser ist über Brücken mit den LEDs verbunden und kann somit durch eine externe Stromversorgung ersetzt werden.

Als Sensor wird ein Licht-/Frequenz-Wandler eingesetzt, beschrieben in Abschnitt 5.2.3. Für den Lichtsensor ist eine zusätzliche Platine realisiert worden. Diese kann über eine Stiftleiste mit der Hauptplatine verbunden werden. Durch diesen Aufbau soll es möglich sein, den Lichtsensor bei Bedarf zu erweitern oder auszutauschen. Außerdem besteht die Option, die Sensorplatine örtlich getrennt zur Hauptplatine einzusetzen.

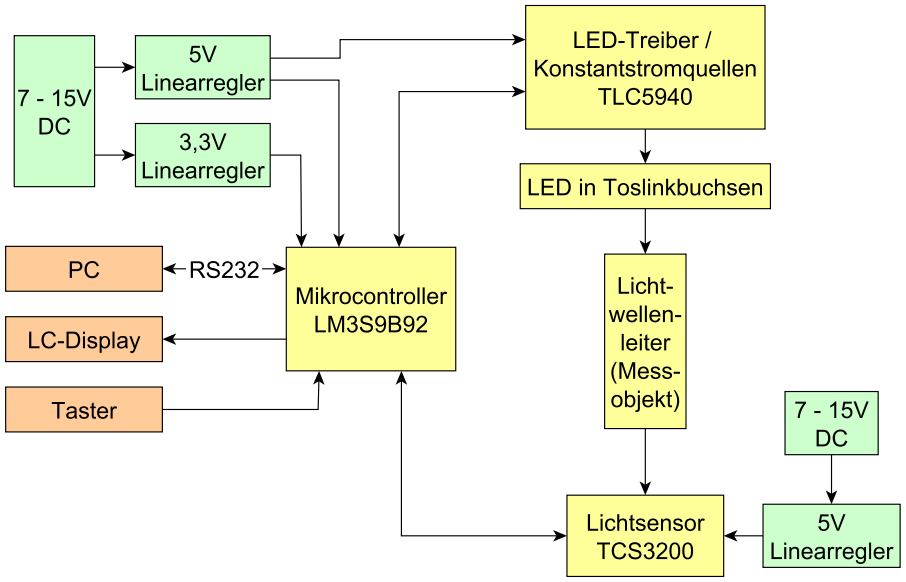

Abbildung 5.12.: Blockschaltbild des Gesamtkonzeptes

5.2.1. Mikrocontroller

Da das LED-Reflektometer als Laboraufbau gedacht ist, wird ein Evaluations-Board mit dem Mikrocontroller LM3S9B92 von Texas Instruments verwendet.

Das EK-LM3S9B92 wird auch in Laboren an der HAW Hamburg und vereinzelt für Messaufbauten im BATSEN-Projekt eingesetzt.

Der entscheidende Vorteil bei Verwendung eines solchen Evaluations-Boards liegt darin begründet, dass viele Peripherie-Einheiten schon einsatzbereit zur Verfügung gestellt werden. Mit dem "In Circuit Debug Interface" aus Abbildung 5.13, welches mit dem Evaluations-Board

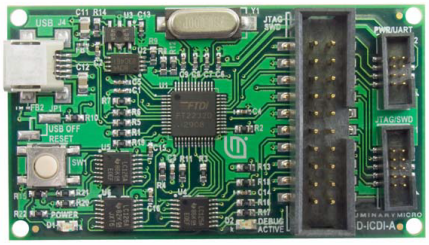

Abbildung 5.13.: In Circuit Debug Interface Board für den Mikrocontroller LM3S9B92 von Texas Instruments aus [48]

in Abbildung 5.14 verbunden werden kann, wird der Mikrocontroller über einen Mini-USB-Anschluss mit einem PC verbunden. Diese Schnittstelle dient als virtueller COM-Port und kann für die serielle Kommunikation zwischen einem PC und dem Mikrocontroller eingesetzt werden.

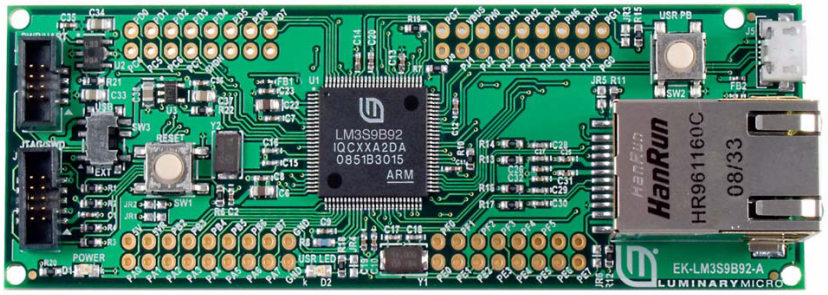

Abbildung 5.14.: LM3S9B92 Evaluation Board von Texas Instruments aus [48]

Derselbe Anschluss am Evaluations-Board dient zum Programmieren des Controllers über die JTAG-Schnittstelle und kann ebenso zum Debuggen eingesetzt werden.
Neben den in diesem Aufbau hauptsächlich verwendeten Einheiten stellt das Evaluations-Board auch Anschlüsse für Ethernet und einen USB-OTG-Port zur Verfügung.

Als Haupttakt für den Mikrocontroller steht ein 16 MHz Quarz bereit.

Der Mikrocontroller LM3S9B92 basiert auf der ARM® Cortex-M3-Architektur. Er stellt eine Reihe von Peripherie bereit, unter anderem [49]:

- Nested Vectored Interrupt Controller (NVIC)
- GPIOs, General Purpose Timer, System Timer
- Kommunikation: UART, SSI, I2C, I2S, CAN, Ethernet, USB

- Analog Komparatoren und 10-Bit ADC
- PWM und QEI

Der Controller kann mittels PLL mit bis zu 80 MHz betrieben werden und bietet 256 KB Flash-Speicher und 96 KB SRAM.

In dem realisierten Schaltungslayout wird der Mikrocontroller zur Steuerung des LED-Treibers, Licht-/Frequenz-Wandlers und des LC-Displays eingesetzt. Weitere Taster und LEDs sind zur Steuerung des LED-Reflektometers unabhängig von einem PC vorgesehen. Dafür wurden die in Tabelle D.1 aufgelisteten Pins des Mikrocontrollers verwendet.

Auf die Sensorplatine wurden, neben den benötigten Pins zur Steuerung des Sensors und zum Abgreifen der Messdaten, noch weitere Pins des Microcontrollers geführt, welche derzeit keine Verwendung finden. Damit aber auch andere Sensoren als der in dieser Arbeit verwendete Licht-/Frequenz-Wandler eingesetzt werden können, wurden unter anderem Pins von internen Analog-Digital-Umsetzern sowie der Pin für die externe Referenzspannung herausgeführt. Auch diese Belegungen sind in der Tabelle D.1 aufgeführt.

5.2.2. LED-Treiber

Mit dem TLC5940 von Texas Instruments sollen die verschiedenen LEDs, welche über die Toslink-Buchsen in Lichtwellenleiter eingekoppelt werden können, betrieben werden.
Bei dem TLC5940 handelt es sich um einen LED-Treiber mit 16 Kanälen. Für jeden der 16 Kanäle kann einzeln die Helligkeit der angeschlossenen LED über eine 4096-stufige PWM oder eine 64-stufige Konstantstromsenke begrenzt werden.
Bei einer Versorgungsspannung von über 3,6 V kann der LED-Treiber bis zu 120 mA pro Kanal liefern. Der maximale Strom kann über den Widerstand R_{IREF} eingestellt werden. Mit der Gleichung

$$I_{max} = \frac{1,24V}{R_{IREF}} 31,5 \tag{5.2}$$

aus dem Datenblatt lässt sich der maximale Strom mit dem eingesetzten Widerstand berechnen [50].
Im derzeitigen Aufbau wurde ein Widerstand von circa 374 Ω verwendet, woraus ein maximaler Strom von etwa 104 mA resultiert.

Das Einstellen der PWM und der "dot correction" genannten Konstantstromquelle geschieht mittels einer seriellen Schnittstelle über den Mikrocontroller.
Bei der Schaltungsentwicklung wurde darauf geachtet, dass die entsprechenden Pins des LED-Treibers an eine SSI-Einheit des Mikrocontrollers geführt werden.

Die Strombegrenzung durch "dot correction" lässt sich durch die Formel berechnen [50]:

$$I_{OUT} = I_{max} \frac{DC}{63} \tag{5.3}$$

Bei dem DC-Wert handelt es sich um eine Zahl zwischen null und 63, welche mit der entsprechenden sechsstelligen Bit-Zahl über die serielle Schnittstelle konfiguriert werden kann. Die Konfiguration des LED-Treibers wird in dem noch folgenden Software-Abschnitt in Kapitel 5.3.1 näher erläutert.

Zwölf der 16 Ausgänge des LED-Treibers wurden mit Anschlüssen auf der Platine versehen. Jeder realisierte Anschluss ist für eine eigene LED-Art vorgesehen. Bei einem Blick auf Abbildung 5.11 erkennt man diese Anschlüsse an der Nummerierung von eins bis sechs:

1. 5 mm RGB-LED (Bedrahtet 4 Pins)
2. 5 mm DUO-LED (Bedrahtet 3 Pins)
3. zwei 3 mm LEDs (Bedrahtet je 2 Pins)
4. 3 mm LED (Bedrahtet 2 Pins)
5. zwei SMD-LEDs (PLCC4)
6. Anschluss einer Leuchtquelle über Molex-Stecker

In der Tabelle 5.1 sind die montierten LEDs aufgeführt. Die angegebenen LED-Nummern setzten sich durch die in Abbildung 5.11 eingezeichneten Zahlen und der Anzahl der LEDs unter einer Toslink-Buchse zusammen. Die Spektren der eingesetzten LEDs wurden bereits in Abschnitt 5.1.2 ausgemessen.

LED Nr.	LED Bezeichnung	Wellenlänge	Farbe	Vorwärtsstrom
LED11	KT-electronic 50RGB-CA	630 nm	Rot	20 mA
LED12	KT-electronic 50RGB-CA	525 nm	Grün	20 mA
LED13	KT-electronic 50RGB-CA	470 nm	Blau	20 mA
LED31	Kingbright WP710A10SECJ4	605 nm	Orange	20 mA
LED32	OSRAM SFH 4350	850 nm	Infrarot	100 mA
LED51	OSRAM LAE63FEBGA243A4B	617 nm	Bernstein	50 mA
LED52	OSRAM SFH4248-Z	940 nm	Infrarot	100 mA

Tabelle 5.1.: Verwendete LEDs im LED-Reflektometer

5.2.3. Lichtsensor

Der verwendete Lichtsensor TCS3200 von TAOS[1] wandelt die einfallende Lichtintensität in eine proportionale Frequenz mit 50-prozentigem Tastgrad [47].

Mit den vier Pins S0, S1, S2, S3 lassen sich die Ausgabefrequenz skalieren und die verwendeten Photodioden konfigurieren.
Der Sensor besteht aus einem $8x8$-Feld aus Photodioden, wovon jeweils sechzehn einen Rot-, Grün-, Blau- oder keinen Filter haben.

Bei dem Schaltungslayout wurde darauf geachtet, dass die Ausgabefrequenz des Sensors an einen Capture Compare PWM (CCP)-Pin des Mikrocontrollers geführt wird, damit entsprechende Auswertungen mithilfe eines Zählers des Mikrocontrollers möglich sind.

Die spektralen Empfindlichkeiten der Photodioden mit den unterschiedlichen Filtern sind in Abbildung 5.15 dargestellt. Wichtig ist die Beobachtung, dass die eingesetzten Filter im Infrarotbereich an Wirkung verlieren, weshalb die spektrale Empfindlichkeit der Blau- und Grünfilter ab ungefähr 650 nm wieder beginnen zu steigen.

Dieser Sensor kann auf der entworfenen Platine durch den Lichtsensor TSL230 von TAOS ersetzt werden, welcher pinkompatibel zum TCS3200 ist. Dieser bietet jedoch nicht die Möglichkeit der Auswahl gefilterter Photodioden an. Dadurch besitzt der Sensor eine höhere Sensitivität, da das gesamte Photodiodenarray verwendet werden kann. Statt der zu messenden Farbe kann die Sensitivität durch die entsprechenden Pins konfiguriert werden.

In dem realisierten Entwurf wurde jedoch aus Gründen der Flexibilität der Farbsensor eingesetzt. So ist mit diesem möglich, Messungen mit einer breitbandigen Lichtquelle durchzuführen. Anstelle der Methode, verschiedene schmalbandige Lichtquellen zeitversetzt leuchten zu lassen, könnte der Sensor die breitbandige Lichtquelle mit den verschiedenen Filtern auswerten.

Ein Vorteil der Licht-/Frequenz-Wandler gegenüber von Sensoren, welche mit einem Analog-Digital-Umsetzer ausgewertet werden müssen, ist die bessere Auflösung. Diese ist beim Analog-Digital-Umsetzer begrenzt durch die Anzahl der Bits und dem zu verarbeitenden Spannungsbereich, welche eine gewisse Quantisierung der Messwerte vorgeben.
Eine Frequenz kann jedoch, abhängig von der Messmethode, genau ausgewertet werden, wodurch ein solcher Quantisierungsfehler nicht auftritt.

[1] Seit kurzem in die Firma AMS eingegliedert

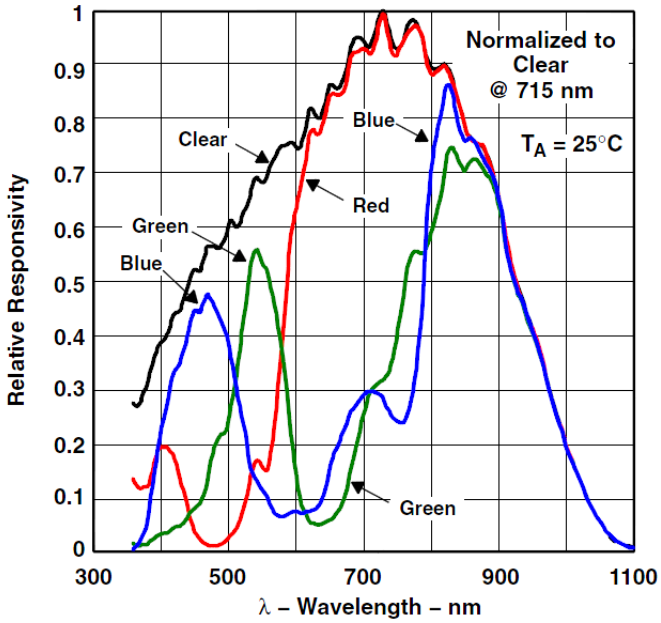

Abbildung 5.15.: Spektrale Empfindlichkeit des TAOS TCS3200D [47]

5.2.4. Weitere Hardware

Neben der bereits erwähnten elementaren Hardware des LED-Reflektometers sind auch Taster und ein Displayanschluss im Design vorgesehen. Die Taster sind zur simplen Auswahl der LEDs und zur Steuerung von Messungen vorgesehen. Sie wurden entsprechend dem Schaltbild in Abbildung 5.16 entprellt.

Da die genauen Schaltschwellen der GPIO-Pins des Mikrocontrollers bei der Auslegung nicht bekannt waren, wurde die Entprellung nach Quelle [52] realisiert. Die implementierte Schaltung wurde so ausgelegt, dass nach Betätigen des Tasters die Spannung am Eingang des Schmitt-Triggers nach 20 ms von 5 V auf 2 V fällt. Die Schaltung wurde für $R1$ = 10 kΩ, $R2$ = 22 kΩ, C = 1 μF mit der Simulations-Software LTSpice geprüft. Die Simulation ist in Abbildung 5.17 dargestellt.
Die Schaltschwellen des Mikrocontrollers liegen für ein "High"-Signal bei 2,1 V, für ein "Low"-Signal bei 1,2 V [49]. Diese Schaltschwellen wurden in der Abbildung 5.17 nachträglich eingezeichnet. Die Simulation zeigt, dass nach der Betätigung des Tasters nach ungefähr 32 ms ein "LOW"-Pegel am Eingang des Mikrocontrollers anliegt. Da für einen menschlichen Anwender kein Unterschied zwischen 20 ms und 32 ms auszumachen ist, braucht die Dimensionierung der Entprellschaltung nicht angepasst werden.

Neben den Tastern wurde auf der Platine der Anschluss für ein LC-Display vorgesehen. Als Displays sind die Verwendung des 20x4NHD-0420DZ-FL-YBW-36757 der Firma Newhaven

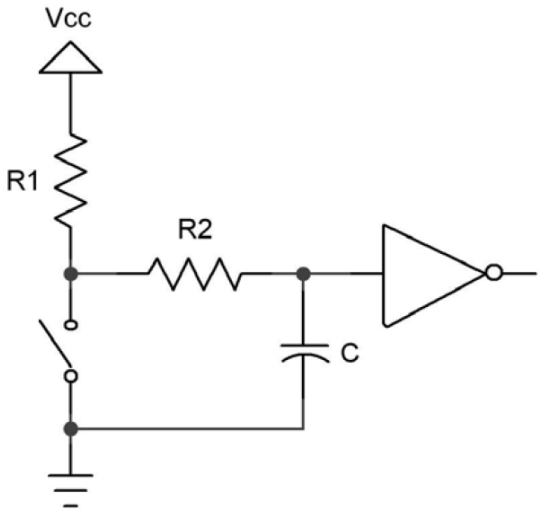

Abbildung 5.16.: Entprellung von Tastern mittels RC-Schaltung [51]. Nach Betätigen des Tasters fällt die Spannung am Eingang des Schmitt-Triggers nach 20 ms von 5 V auf 2 V. $R1 = 10\ \text{k}\Omega$, $R2 = 22\ \text{k}\Omega$, $C = 1\ \mu\text{F}$

Display sowie W204B-NLW von Electronic Assembly eingeplant. Für die Versorgung der Hintergrundbeleuchtung ist ein Vorwiderstand auf der Platine vorgesehen. Für das Display von Newhaven können die Pads des entsprechenden Widerstandes überbrückt werden. Beim Anschluss ist die Polarität des Anschlusses auf der Platine zu beachten, damit die Kontakte nicht gespiegelt werden. Das Display von Newhaven ist für den Betrieb an 5 V-Logik vorgesehen. Damit es auch mit 3,3 V-Logik funktioniert muss die Versorgungsspannung auf circa 4,6 V herabgesetzt werden, da das Display Spannungen zwischen 0,7 VCC bis VCC als "High"-Pegel interpretiert, der Mikrocontroller aber nur 3,3 V erreicht.
Auf einen Pegelumsetzer wurde verzichtet, da das Display von Electronic Assembly für den Betrieb mit einer 3,3 V-Logik vorgesehen ist.

Das Display wurde in Betrieb genommen, aber aufgrund der für die Datenaufnahme benötigten PC-Anbindung später nicht weiter verwendet.

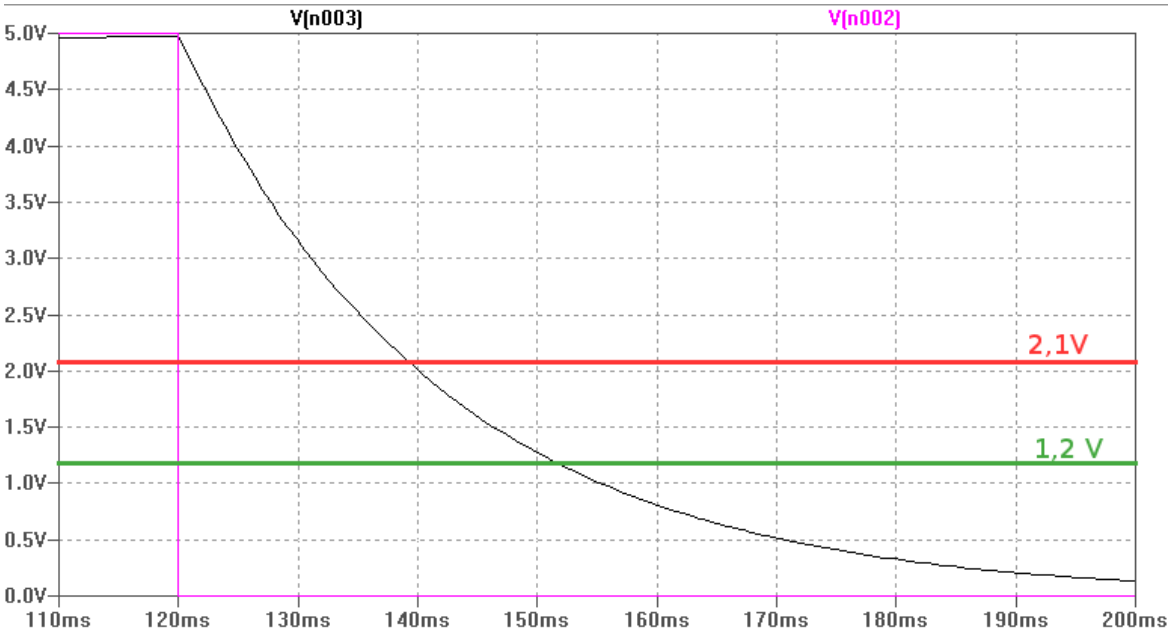

Abbildung 5.17.: Simulation der Entprellung eines Tasters mittels RC-Schaltung. Das Signal V[n003] entspricht der Spannung am Eingang des Schmitt-Triggers, V[n002] die über dem Widerstand R_2 dem Kondensator anliegende Spannung.

5.3. Software zur Steuerung der optischen Messung und Datenerfassung

Die Software für die Messungen wird auf dem Mikrocontroller und dem PC implementiert. Die Software auf dem Mikrocontroller stellt die Voraussetzungen für die Messungen bereit. Es kann konfiguriert werden, welche Leuchtdioden bei den Messungen beteiligt sein sollen. Diese werden wiederum in einem eingestellten Zyklus an und ausgeschaltet, bevor zu der nächsten LED gewechselt wird. Die Möglichkeit, die LED an und auszuschalten soll Erwärmungen der LED reduzieren. Dadurch kann verhindert werden, dass sich die durch den Betrieb verursachte Erwärmung auf die Intensität des ausgestrahlten Lichtes auswirkt. Dies ist wichtig, um eine möglichst konstante Lichtintensität und somit gleichbleibende Rahmenbedingungen für Messungen zu gewährleisten.
Neben diesen Zyklen lassen sich auch die gesamte Messdauer, der LED-Strom sowie Farbfilter und Skalierung des Farbsensors konfigurieren. Um dies komfortabel bewerkstelligen zu können, wurde eine Matlab-Oberfläche erstellt, welche über eine UART-Schnittstelle mit dem Mikrocontroller kommunizieren kann. Neben der Möglichkeit, die erwähnten Parameter zu konfigurieren, zeigt die Software die aktuellen Messungen an und speichert diese ab.

5.3.1. Controller-Software

Die Software des Mikrocontrollers steuert in erste Linie die Leuchtdioden über einen LED-Treiber und wertet die zur Lichtintensität proportionale Rechteckfrequenz des Lichtsensors aus. Über eine UART-Schnittstelle lassen sich Leuchtzyklen, LED-Strom, Farbfilter und Skalierung des Lichtsensors von einem PC aus konfigurieren. Gleichzeitig sendet der Mikrocontroller Messungen, derzeit alle 200 Millisekunden, über UART, welche an einem PC ausgewertet werden können.

Nachfolgend sind die wichtigsten Funktionen der Controller-Software erläutert. Bei aufwändigeren Funktionen wurden zur Veranschaulichung UML-Aktivitätsdiagramme erstellt.
Die Funktionen sind in verschiedenen C-Dateien untergebracht. Es wurde auf eine möglichst sinnvolle Gruppierung geachtet. Funktionen, welche zum Beispiel den LED-Treiber betreffen, sind so in der Datei *ledDriver.c* zu finden.

main()

Die Main-Funktion des C-Programmes des Mikrocontrollers besteht lediglich, wie in Abbildung 5.18 dargestellt, aus den Initialisierungen des LED-Treibers, des Lichtsensors, aller Timer und der seriellen Schnittstelle UART[2].

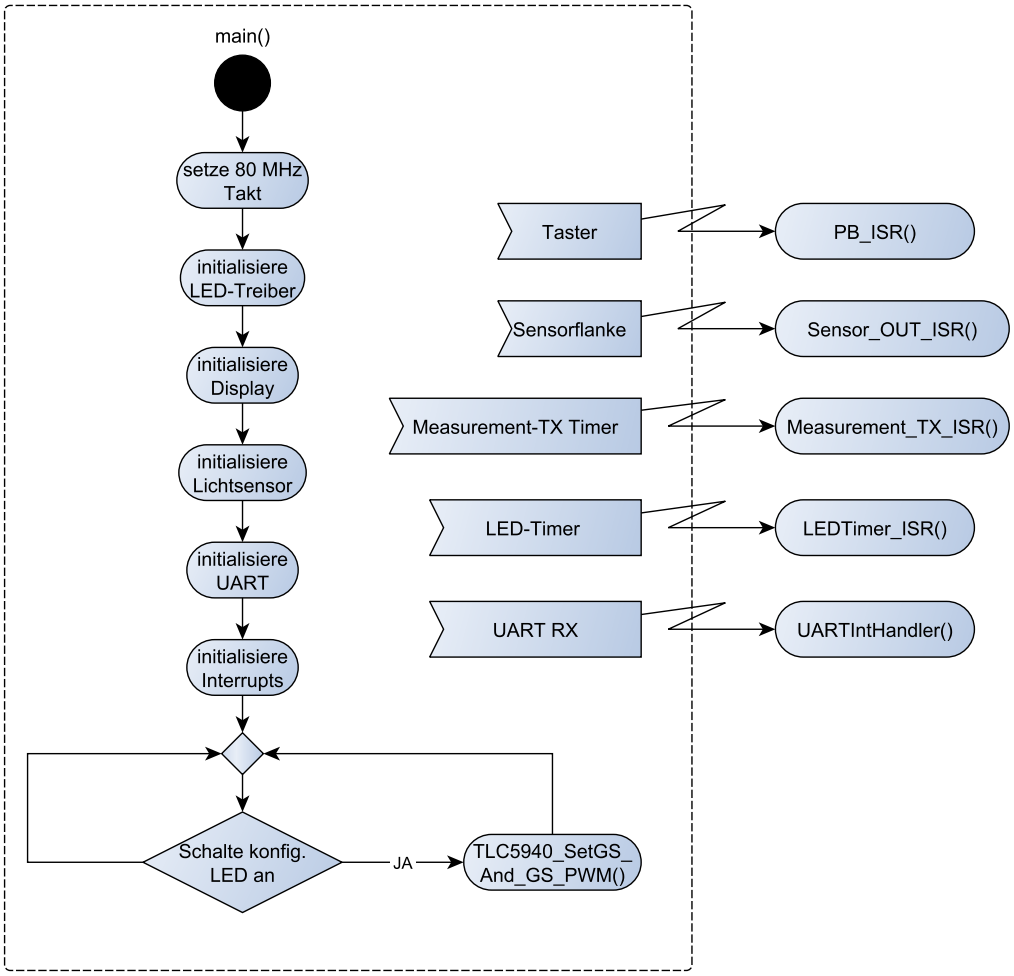

Abbildung 5.18.: Main-Funktion der Mikrocontroller-Software

Nach den Initialisierungen wird in einer Dauerschleife überprüft, ob der LED-Treiber eingeschaltet werden soll. Das entsprechende Flag wird außerhalb der Main-Funktion gesetzt. Das Aufrufen der entsprechenden Funktion wurde nicht interruptgesteuert realisiert, damit bei eingeschalteter LED eine Interrupt Service Routine (ISR) nicht entsprechend lange aktiv ist.

[2]Universal Asynchronous Receiver Transmitter

Die Dauerschleife innerhalb der Main-Funktion kann durch Interrupts unterbrochen werden. Diese können ausgelöst werden von Tastern, Timer der LED-Zyklen und zur Übertragung der Messungen sowie durch steigende Flanken der Sensorfrequenz und UART-Eingaben.

uartIntHandler()

Für Benutzereingaben über die serielle Schnittstelle UART ist es notwendig, die entsprechenden Interrupts zu behandeln und gesendete Zeichen zu analysieren.
Die realisierte Funktion UARTIntHandler basiert auf der Funktion UARTStdioIntHandler aus utils\uartstdio.c, bereitgestellt durch StellarisWare von Texas Instruments. Der Ablauf ist in Abbildung 5.19 dargestellt.

Bei der Eingabe einer Zeichenkette wird diese in einem Feld abgelegt, bis die Eingabetaste, repräsentiert von der Zeichenfolge "\r" und "\n", gedrückt wird. Dann wird die Zeichenkette an die Funktion CmdLineProcess() weitergegeben. Diese von StellarisWare zur Verfügung gestellte Funktion dient zur Auswertung der Zeichenkette. Sie interpretiert die Zeichenkette in folgender Form: *Befehl* $Argument_1$ $Argument_2 \ldots Argument_n$. In einer zuvor angelegten Tabelle überprüft die Funktion dann, ob der eingegebene Befehl vorhanden ist und ruft die entsprechend konfigurierte Funktion auf. Die Argumente werden, entsprechend dem Prinzip des Aufrufes der main-Funktion in C, in einem Argumentenfeld gemeinsam mit der Anzahl der Argumente übergeben. Tabelle 5.2 zeigt die implementierten Eingabebefehle.

Befehl	Beschreibung
start	Messung starten
stop	Messung stoppen
setLED	DC-Wert einer LED setzen (setLED LED-Nr(0-N) DC-Wert(0-63)
setCycle	An-/Aus-Zyklus setzen (setCycle LEDON(ms) LEDOFF(ms) TOTAL(ms))
setScaling	Setze Skalierung des Sensors (setScaling s0 s1)
setFilter	Setze Filter des Sensors (setScaling s2 s3)
status	Sendet Takt des Controllers
help	Sendet die Befehlstabelle

Tabelle 5.2.: Befehlstabelle der seriellen Schnittstelle des Mikrocontrollers

LEDTimer_ISR()

Die Controller-Software soll, wie bereits erläutert, in der Lage sein, in bestimmten Zyklen eine LED an- und wieder auszuschalten, bevor zur nächsten LED gewechselt wird. Dafür

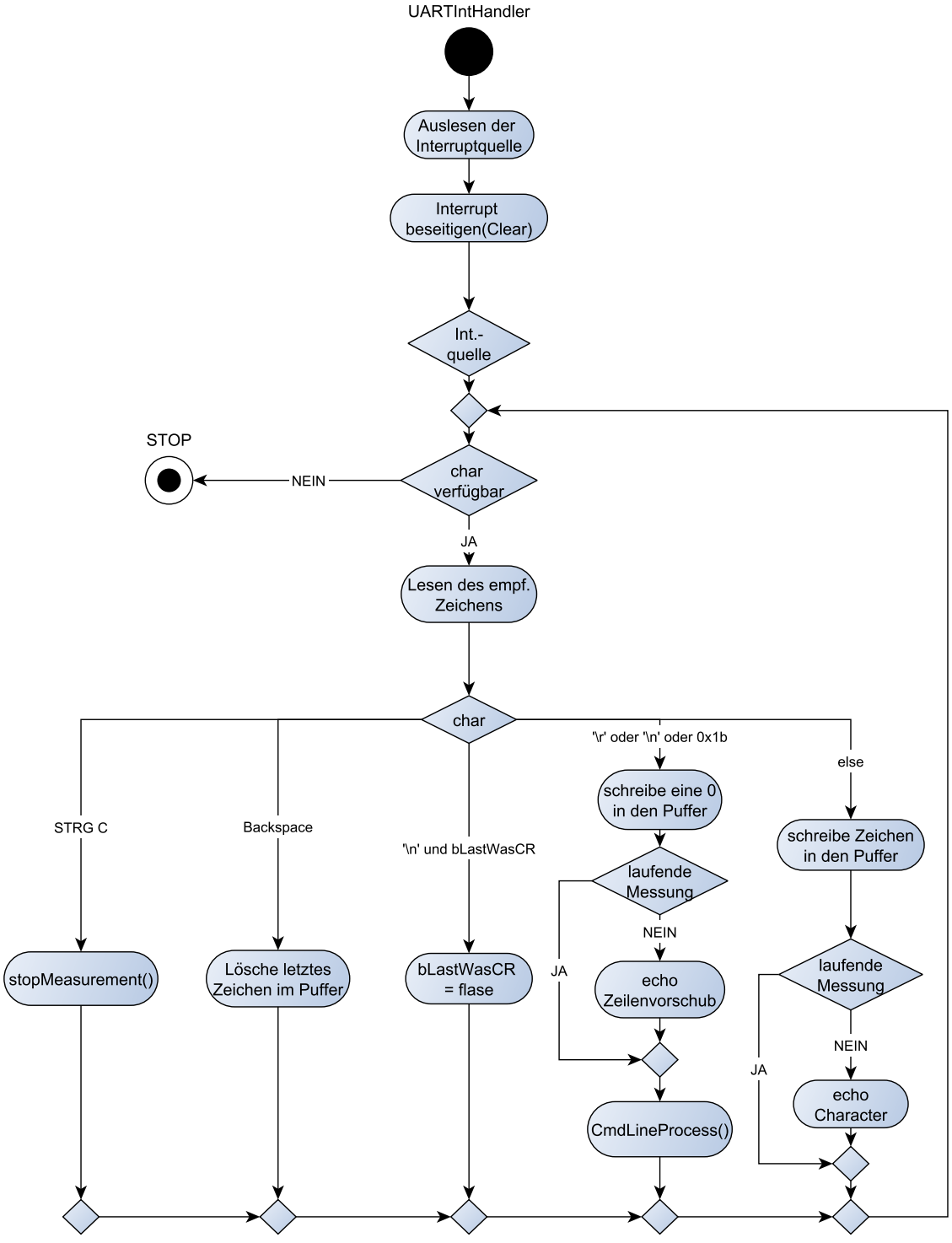

Abbildung 5.19.: Interrupthandler für Befehle über UART

wurde als Interrupt Service Routine eine einfache Zustandsmaschine realisiert, welche zwischen den beiden Zuständen an und aus wechselt. Dabei werden die jeweils konfigurierten Zeiten eingestellt und ein Flag gesetzt, welches in der main-Funktion ausgewertet wird.
Die Funktion zum Einschalten der LED sucht dabei erst die nächste LED, welche eingeschaltet werden soll, setzt den entsprechenden Wert in dem Feld *dcValue* und überträgt diesen an den LED-Treiber. Die einzuschaltenden LEDs wurden zuvor in dem Feld *storedDcValue* durch Eingaben über die serielle Schnittstelle oder durch Betätigen des Tasters S_3 gesetzt.

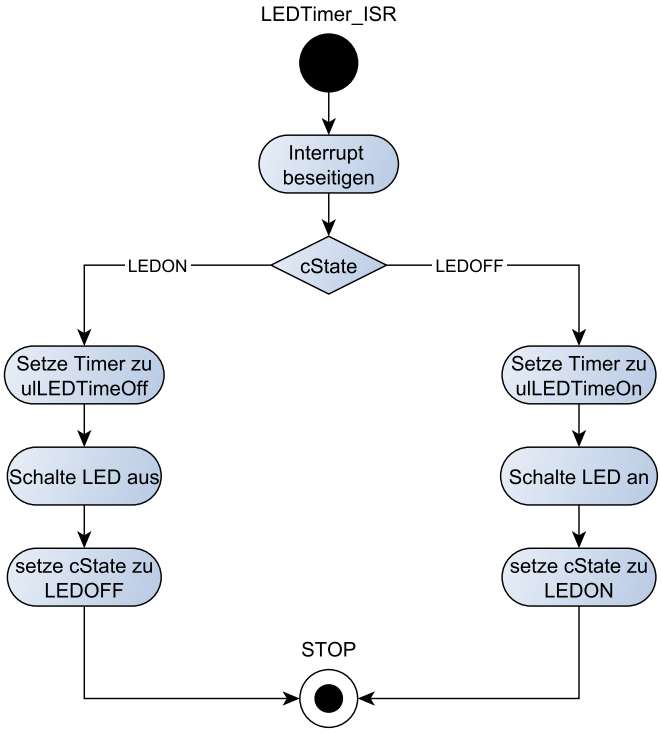

Abbildung 5.20.: Timer für das Umschalten zwischen den LEDs

PB_ISR()

Auf der Platine sind drei Taster vorgesehen, welche zur manuellen Steuerung des LED-Reflektometers vorgesehen sind. Alle sind am selben Port des Mikrocontrollers angeschlossen und werden somit in einer ISR ausgewertet. Abbildung 5.21 zeigt die Behandlung eines Interrupts, ausgelöst durch die Betätigung eines Tasters.

Taster S_3 (PJ7) dient zum Weiterschalten der LEDs in Blöcken. Jede Toslink-Buchse auf der Platine entspricht dabei einem Block. Nach dem aus drei LEDs bestehenden RGB-Block

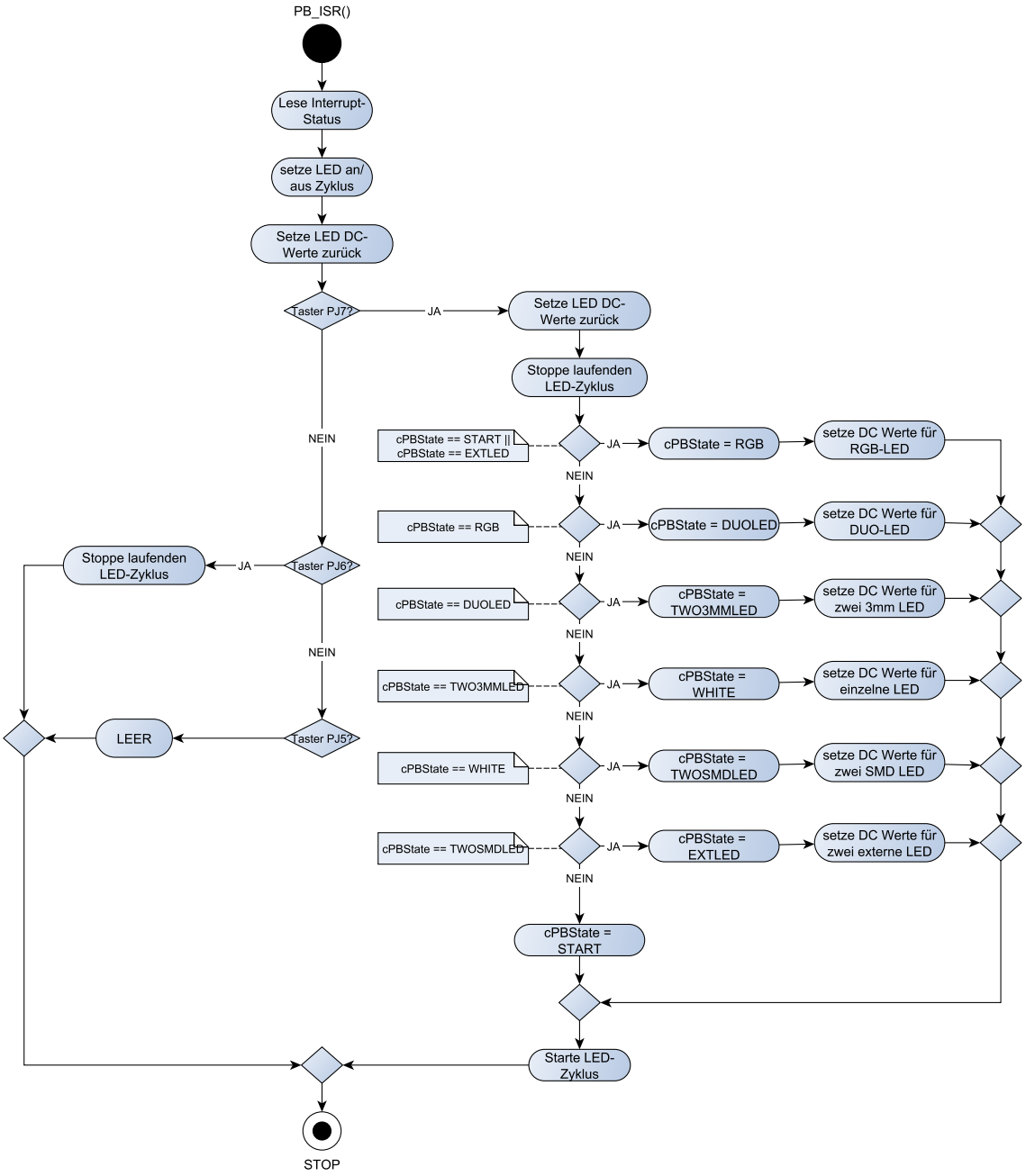

Abbildung 5.21.: Interrupt Service Routine für Benutzertaster

wird so beispielsweise zum DUO-LED-Block weiter geschaltet. In der Software werden die entsprechenden DC-Werte der zum Einsatz kommenden LEDs in dem vorgesehenen Feld gesetzt. Der Ablauf wurde in einem Zustandsautomaten realisiert, in welchem die eingeschalteten LEDs von Block zu Block geschaltet werden. Nach dem Setzen eines neuen Blockes wird der konfigurierte LED-Zyklus gestartet. Taster S_2 (PJ6) stoppt den laufenden LED-Zyklus sowie alle Messungen. Taster S_1 (PJ5) besitzt in der derzeit realisierten Umsetzung keine Funktion.

setNextLED()

Die Funktion *setNextLED()* sucht die nächste vom Benutzer oder durch den Taster S_3 konfigurierte LED und setzt den konfigurierten DC-Wert in einem Feld. Alle anderen DC-Werte werden auf 0 gesetzt. Dieses Feld, in welchem dadurch nur eine einzige LED konfiguriert wird, ist zur Übertragung an den LED-Treiber vorgesehen.

TLC5940_SendDC()

TLC5940_SendDC() sendet die aktuell ausschließlich durch *setNextLED()* konfigurierten DC-Werte an den LED-Treiber.

Zu Beginn wurde eine Funktion zur seriellen Übertragung realisiert, welche sich am Funktionsablaufplan für den LED-Treiber von Texas-Instruments orientiert [53].
Später wurde die serielle Übertragung mittels SSI-Einheit des Mikrocontrollers realisiert. Diese ist dahingehend konfiguriert, 6-Bit-Werte zu übertragen. Die DC-Werte sind intern zwar als 8-Bit-Werte abgespeichert, in Tests zeigte sich jedoch, dass die SSI-Einheit nur die ersten sechs Bit überträgt und die restlichen zwei verwirft.

TLC5940_SetGS_And_GS_PWM()

Der TLC5940 von Texas Instruments ist vorgesehen, im PWM-Modus betrieben zu werden. Bei dem Aktivitätsdiagramm in Abbildung 5.22 handelt es sich um ein modifiziertes Flussdiagramm nach Quelle [53], welches in dieser Funktion umgesetzt wurde. Bei den verschiedenen Tests zeigte es sich als notwendig, die PWM-Funktion des LED-Treibers zu verwenden, um LEDs einzuschalten. Das bedeutet, dass dafür mindestens ein Puls am GSCLK-Pin, also dem Takt der PWM-Funktion, notwendig ist. Zur vollständigen Funktionstüchtigkeit des LED-Treibers schien es nach durchgeführten Tests unerlässlich, die vorgesehenen 4095 GSCLK-Pulse durchzuführen, bevor dem LED-Treiber neue DC-Werte übermittelt werden.

Nach dem Flussdiagramm von Texas Instruments ist es bei jedem Durchlauf dieser Funktion vorgesehen, den BLANK-Pin auf "HIGH" zu setzen [53]. Durch das Setzen dieses Pins werden alle Ausgänge des LED-Treibers ausgeschaltet. Um dies zu verhindern, wurde nach dem ersten GSCLK-Puls eine Schleife eingefügt. Diese wird solange nicht verlassen, bis das entsprechende Flag durch den LED-Timer wieder gelöscht wird. Durch den ersten GSCLK-Puls werden alle konfigurierten Ausgänge des LED-Treibers eingeschaltet. Nach Setzen des Flags wird dann das PWM-Programm einmalig bis zum nächsten Aufruf der Funktion durchlaufen. Durch diese Methode soll ein minimales Flackern der LED unterbunden werden. Sollte die PWM-Funktion eingesetzt werden sollen, müsste die blockierende Schleife entfernt werden.

Die dargestellte Funktion übermittelt dem LED-Treiber schon neue Einstellungen für seine PWM-Funktion (GS-Data), während die PWM mit vorherigen Daten noch ausgeführt wird. Diese eingetakteten Daten stehen somit erst im nächsten Aufruf der Funktion, nach dem XLAT-Puls, zur Verfügung.
Im Datenblatt des LED-Treibers ist im Timing-Diagramm eine abweichende Variante dargestellt, in welcher die Daten für die PWM zeitlich getrennt zur Durchführung der PWM durchgeführt werden kann [50]. Das würde dazu führen, dass diese Daten, genauso wie die DC-Daten, über die SSI-Einheit übertragen werden könnten.
Es stellt sich daher die Frage, ob die wiederholte Konfiguration der PWM-Daten nicht überflüssig ist und dadurch auch der XLAT-Puls sowie das Ausschalten der LED's überflüssig werden würde. Dadurch könnte die Funktion stark vereinfacht werden.

SENSOR_OUT_ISR()

Zur Auswertung der Ausgabefrequenz des Lichtsensors gibt es unterschiedliche Möglichkeiten.
Die derzeit implementierte Variante stellt die simpelste dar. Bei einer steigenden Flanke der vom Lichtsensor erzeugten Frequenz wird durch die ISR eine Zählervariable inkrementiert. Diese Variable wird in dem Zeitintervall, in welchem Messwerte über UART übertragen werden, ausgewertet. Dieser Zeitintervall ist derzeit auf 200 ms konfiguriert.
Die Frequenz (f_{OUT}) wird berechnet, indem die gezählten steigenden Flanken ($ticks$) durch die Messzeit (t_M) in Sekunden geteilt wird.

$$f_{OUT} = \frac{ticks}{t_M} \tag{5.4}$$

Im Programmcode wird die Zeit in Millisekunden angegeben. Die gezählten steigenden Flanken werden daher entsprechend mit 1000 Multipliziert.
Die Frequenz bei einer Messzeit von einer Sekunde entspricht genau der Frequenz, mit einer möglichen Abweichung von einem Herz. Die Abweichung von einem Herz kommt dadurch

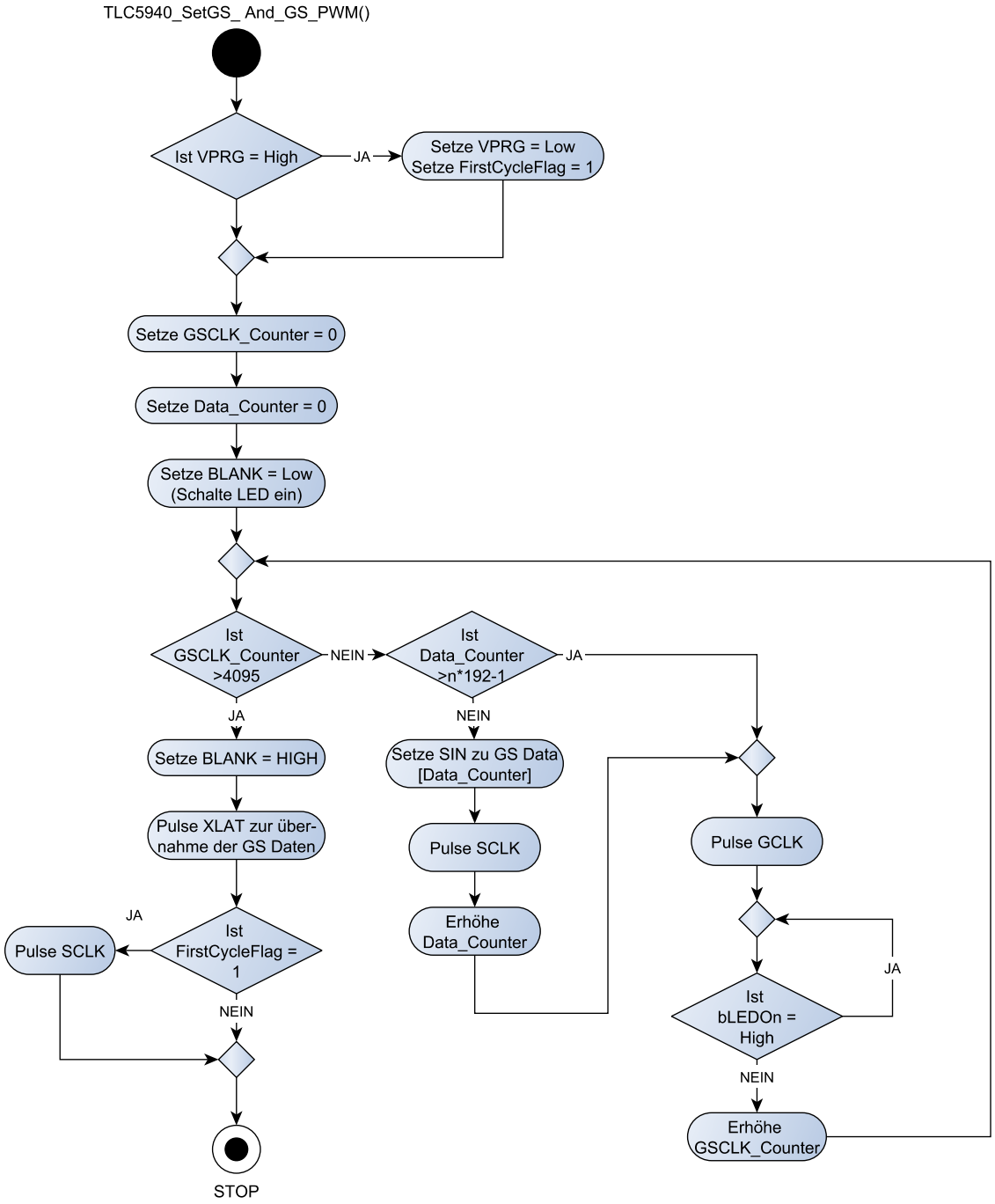

Abbildung 5.22.: Funktion zur Steuerung der PWM-Funktion des LED-Treibers modifiziert nach [53]

zustande, dass die Auswertung der Messung direkt vor oder nach einer Sensorflanke stattfinden kann. Dadurch wird auch klar, dass bei einer Messzeit von 200 ms die Abweichung 5 Hz betragen kann, da die gezählten Sensorflanken mit fünf multipliziert werden müssen, um die Frequenz zu berechnen.

Eine weitere getestete Variante zur Frequenzmessung ist die Zeitmessung zwischen zwei steigenden Flanken durch Verwendung eines Timers des Mikrocontrollers im "Capture"-Modus.
Bei einer Sensorflanke wird der aktuelle Zählwert des Timers ausgelesen. Der Zählwert des Timers bei der zuvor aufgetretenen Sensorflanke wurde in einer Variable gesichert. Durch einen Vergleich der beiden Zählwerte kann die Ausgangsfrequenz des Lichtsensors berechnet werden. Neben von den Sensorflanken erzeugten Interrupts, müssen aber auch Interrupts des Timers ausgewertet werden, welche beim Überlauf des Zählers auftreten. Bei Vergleichsmessungen mit einem Oszilloskop stellte sich heraus, dass die durch diese Methode nur Frequenzen von bis zu 350 KHz ausgewertet werden können. Der Lichtsensor kann aber eine Frequenz von mehr als 700 KHz erzeugen. Die Messmethode ist für solche Frequenzen nicht geeignet. Da die Sensorfrequenz aber herunterskaliert werden kann, wurde die Funktion mit der Bezeichnung Time_Measurement_Timer_ISR() im Quellcode belassen.

In einer dritten Methode zur Messung der Frequenz könnte die Sensorfrequenz als Takt für einen Timer verwendet werden. Diese Variante wurde jedoch nicht näher untersucht und kann daher nicht bewertet werden.

5.3.2. Steuerungs-Software

Zur Steuerung der Messungen des Mikrocontrollers wurde eine Oberfläche in Matlab mithilfe der Matlab-Funktion GUIDE erstellt. Diese Software ist in der Lage, sämtliche möglichen Konfigurationen, welche die Messungen betreffen, auf dem Mikrocontroller einzustellen. Die Kommunikation findet über die serielle Schnittstelle UART statt, welche mit einem PC über einen COM-Port angesprochen werden kann. Zu den Einstellungsmöglichkeiten gehören die DC-Werte des LED-Treibers (siehe Abschn. 5.2.2), welche den Strom der LEDs begrenzen, sowie Filter und Skalierung des Lichtsensors. Des Weiteren können die Zyklen der LEDs und die Gesamtdauer der Messung konfiguriert werden. Durch das Einsetzen einer Null als Gesamtzeit wird dem Controller eine Dauermessung signalisiert.
Messungen werden graphisch in einem Diagramm dargestellt. Um die Stabilität des Programmes nicht zu beeinträchtigen, ist die maximale Anzahl der angezeigten Werte begrenzt. Wird der derzeitige Puffer von 10000 Werten überschritten, so wird dieser abgespeichert und wieder neu befüllt. In Abbildung 5.23 ist die Oberfläche der Steuerungssoftware gezeigt. Die Funktionalität der Matlab-Software ist größtenteils in den Callbacks implementiert. Diese Callback-Funktionen werden nach Verwendung eines Buttons, einer Checkbox oder anderer

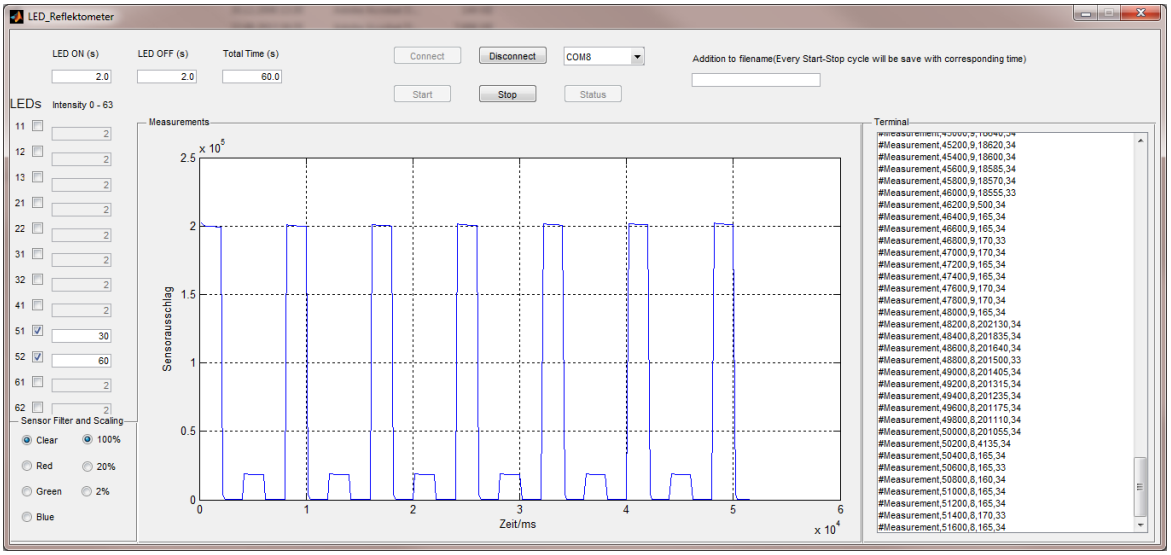

Abbildung 5.23.: Steuerungssoftware des LED-Reflektometers. Im Anhang F befindet sich eine vergrößerte Darstellung der Abbildung

Elemente der Benutzeroberfläche aufgerufen. Die wichtigsten Funktionen sind nachfolgend erläutert.

connectButton_Callback

Die Hauptfunktionalität der Steuerungssoftware steckt hinter dem "Connect"-Button. Nach Betätigung wird versucht, eine Verbindung zum Mikrocontroller herzustellen. Wie in dem stark vereinfachtem Aktivitätsdiagramm in Abbildung 5.24 dargestellt, wird, im Falle einer erfolgreichen Verbindung, die serielle Schnittstelle auf Nachrichten überwacht. Andernfalls wird die Verbindung wieder geschlossen. Tabelle 5.3 listet die zu erwartenden Nachrichten vom Mikrocontroller und ihre Verarbeitung im Matlab-Skript auf.
Empfangene Messdaten werden in einem Puffer abgespeichert und auf der graphischen Oberfläche angezeigt. Wird die Puffergröße überschritten, werden die Messdaten in einer Datei abgespeichert. Als Dateiname dient dafür das Datum und die Zeit, bei der eine Messung gestartet wurde. Zusätzliche Angaben für den Dateinamen können in einem dafür vorgesehenem Eingabefeld in der Oberfläche vorgenommen werden.

startButton_Callback

Durch Betätigung des Startknopfes werden die in der Oberfläche vorgenommenen Einstellungen an den Mikrocontroller übermittelt sowie der Befehl zum Starten der Messung ge-

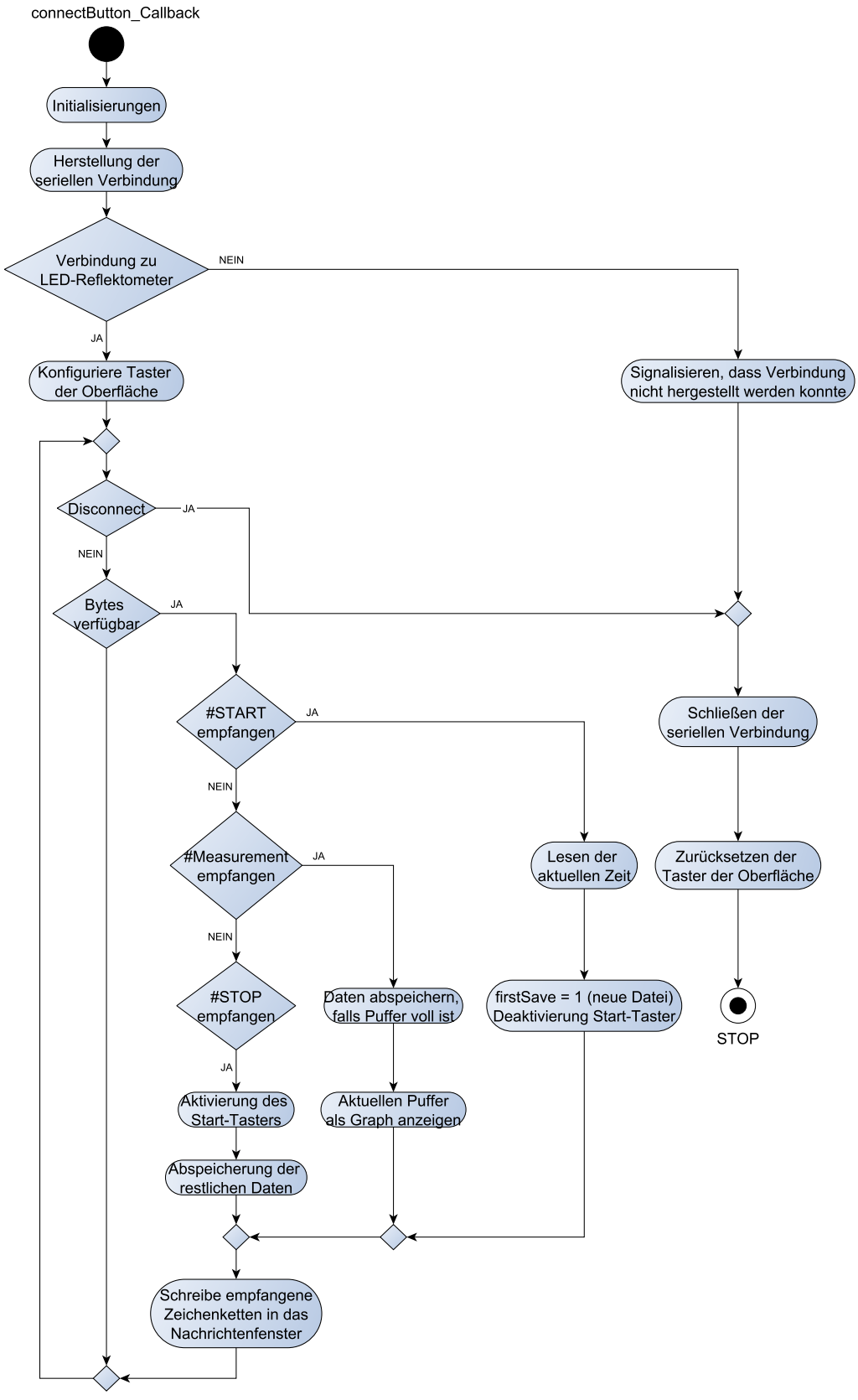

Abbildung 5.24.: UML-Aktivitätsdiagramm der connect-Funktion

Nachricht des Controllers	Inhalt	Beschreibung	Reaktion der Matlab-Software
#START	-	Der Controller startet eine neue Messung	Speichern der Zeit, Zurücksetzen des Puffers
#STOP	-	stoppt Messungen	Abspeichern der Werte aus dem Puffer
#Measurement	Zeit, Led Nr., Sensorwert, μC Temp.	Der Controller sendet Messwerte	Messwerte im Puffer ablegen
#SUCCESS	-	Ein Kommando wurde erfolgreich ausgeführt	-
#WARNING	Beschreibung	Ein Befehl konnte nicht ausgeführt werden	Ausgabe im Nachrichtenfenster

Tabelle 5.3.: Nachrichten des Mikrocontrollers über die serielle Schnittstelle

sendet. Dafür werden die bereits in Tabelle 5.2 aufgelisteten Kommandos eingesetzt. Als Reaktion auf das Kommando "start" antwortet der Controller seinerseits mit der Nachricht "#START", welche wiederum durch die Überwachung der seriellen Verbindung detektiert wird.

stopButton_Callback

Wird der Knopf "Stop" betätigt, so wird dem Mikrocontroller lediglich das Kommando "stop" übermittelt.

Der Controller antwortet mit der Nachricht "#STOP". Diese wird dann durch die, mit dem Knopf "Connect" gestarteten, Überwachung der seriellen Schnittstelle detektiert, woraufhin die aufgenommenen Daten abgespeichert werden.

statusButton_Callback

Das Matlab-Programm sendet dem Controller den Befehl "status". Der Controller antwortet mit "#STATUS, LED-Reflektometer, Controller-Frequenz".
Dasselbe Kommando wird bei der Überprüfung, ob das Matlab-Programm mit einem LED-Relektometer verbunden ist, verwendet und ausgewertet.

disconnectButton_Callback

Durch Betätigen des Knopfes "Disconnect" wird ein Flag gesetzt, welches der durch den vorherigen Aufruf von connectButton_Callback gestarteten Schleife signalisiert, dass diese verlassen werden kann. Dies ist auch in der Abbildung 5.24 angedeutet.
Die Überwachung der seriellen Schnittstelle wird dadurch beendet.

6. Prüfung des Messkonzeptes

6.1. Untersuchung des Aufbaus auf Reproduzierbarkeit

Im folgenden Abschnitt wurden die vom LED-Treiber gelieferten Ströme überprüft. Gleichzeitig wurden die Lichtintensitäten der am entsprechenden Ausgang angeschlossenen LED aufgezeichnet. Der Strom, welcher LEDs durchfließt, steht im direkten Zusammenhang mit der Intensität des von ihnen abgestrahlten Lichtes. Daher soll überprüft werden, wie konstant sich der durch den LED-Treiber gelieferte Strom verhält. Zur Messung der Ströme kam das Multimeter 45 der Firma Fluke zum Einsatz und für die Lichtintensität der Licht-/Frequenz-Wandler auf dem LED-Reflektometer. Die Messungen des Fluke-Messgerätes wurden über eine seriellen Schnittstelle an einem PC aufgenommen.

Die einzelnen LEDs wurden jeweils einer Dauerstrommessung und einer Messung bei gepulstem Strom unterzogen. Bei der gepulsten Messung wurde der Strom abwechselnd für zwei Sekunden ein- und ausgeschaltet. Die Daten des Lichtsensors wurden bei der Auswertung über die zwei Sekunden, in welchen die LED eingeschaltet war, gemittelt. Bei den mit dem Multimeter 45 der Firma Fluke gemessenen Stromwerten wurde aufgrund des Messverhaltens bei Sprüngen zwischen dem konfigurierten Strom und 0 mA jeweils der Maximalstrom eines Blockes ermittelt.

Messungen bei Dauerstrom

In den ersten Messungen wurden der LED-Treiber und der Controller so konfiguriert, dass der LED-Strom dauerhaft eingeschaltet war. Die Messungen wurden an zwei verschiedenen Ausgängen durchgeführt. Der geschaltete Strom orientierte sich an den im entsprechenden Datenblatt der jeweiligen LED genannten Nennströmen. Sie wurden mittels "dot correction" im LED-Treiber konfiguriert.

Bild 6.1 zeigt eine Messung an der LED11 (siehe Tabelle 5.1). Der DC-Wert des LED-Treibers wurde auf zwölf eingestellt. Dieser Wert ergibt nach den Gleichungen 5.2 und 5.3, mit dem verbauten 374 Ω Referenzwiderstand, einen Strom von 19,89 mA.

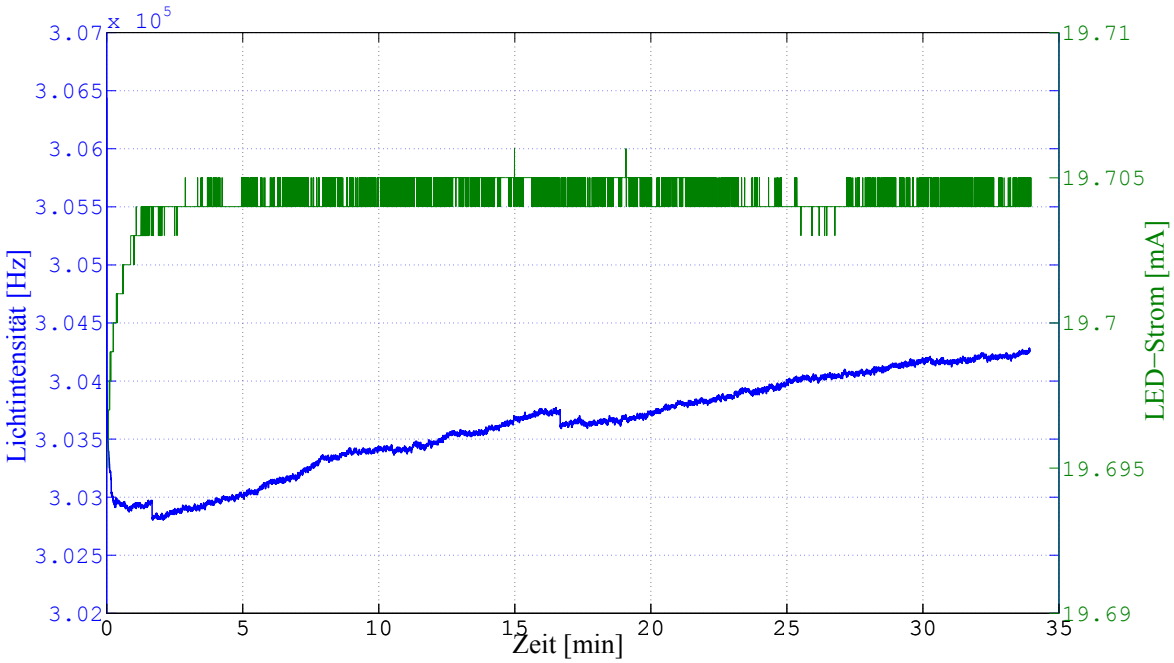

Abbildung 6.1.: Strom und Intensitätsmessung über die Zeit an LED11 (siehe Tabelle 5.1) bei Dauerstrom

Die Messung zeigt, dass der vom LED-Treiber gelieferte Strom nach ungefähr 2,5 Minuten eingeschwungen ist. Bei den kleinen Spitzen, welche zu sehen sind, handelt es sich um die Quantisierung des Multimeters. Diese liegt bei 1 μA. Obwohl sich der LED-Strom nicht ändert, steigt die gemessene Lichtintensität über die Zeit an. Eine mögliche Ursache dafür könnten Erwärmungen der LED, verursacht durch den Betriebsstrom, sein. Die gemessene Intensität bewegt sich über dem gesamten Zeitraum innerhalb von 1,2 %, der Strom innerhalb von 0,1 %.

Abbildung 6.2 zeigt die gleiche Messung mit LED51 (siehe Tabelle 5.1). Der DC-Wert des LED-Treibers wurde mit 30 konfiguriert. Der mit diesem Wert berechnete Strom beträgt 49,73 mA.

Im Unterschied zur ersten Dauerstrommessung schwingt der Strom innerhalb der ersten 30 Minuten nicht auf einen festen Wert ein. Es ist jedoch zu erkennen, dass der Anstieg des Stromes über die Zeit langsam geringer wird. Der Anstieg des Stromes hat einen Abfall der gemessenen Lichtintensität zur Folge. Über den gesamten aufgezeichneten Zeitraum ändert sich der Strom um 0,7 %, die gemessene Lichtintensität um 1,1 %.

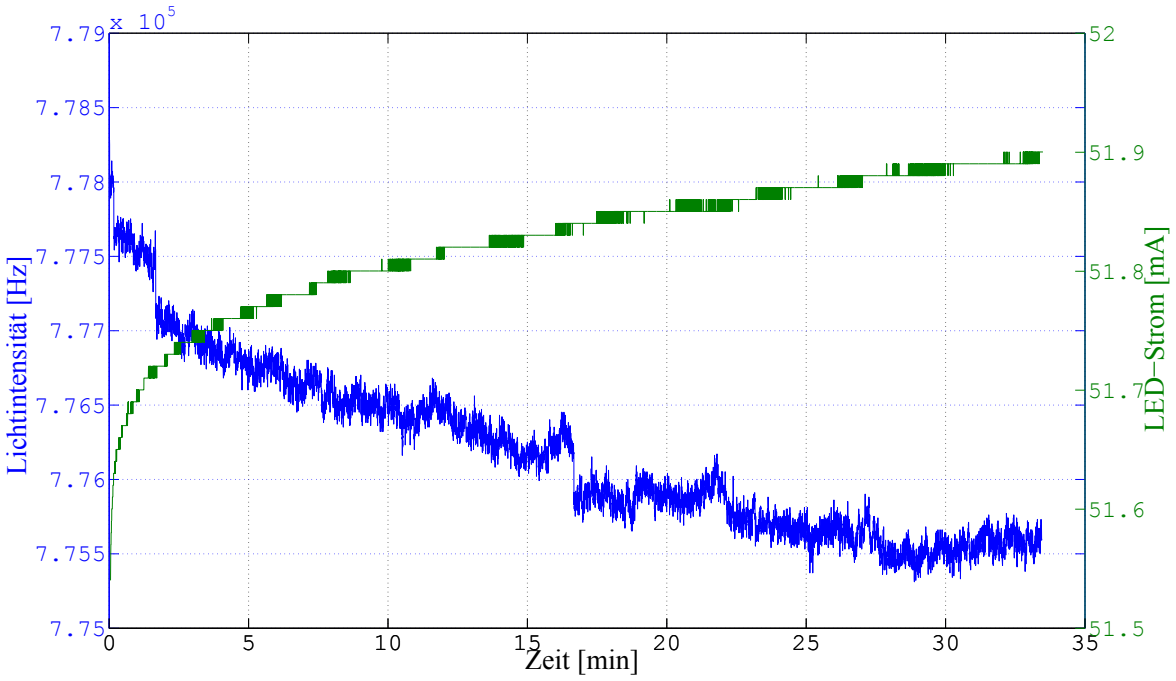

Abbildung 6.2.: Strom und Intensitätsmessung über die Zeit an LED51 (siehe Tabelle 5.1) bei Dauerstrom

Messungen bei gepulstem Strom

Um den Einfluss von Erwärmungen zu verringern, ist in der Controller-Software die Möglichkeit vorgesehen, LEDs zyklisch an- und auszuschalten. In den beiden folgenden Messungen wurden dieselben LEDs mit denselben DC-Werten des LED-Treibers betrieben. Im Unterschied zu den vorherigen Messungen wurden sie jedoch im zwei Sekunden-Takt an- und ausgeschaltet. Die Messwerte für jeden Puls wurden mit einem Matlab-Skript ausgewertet und gemittelt.

In Abbildung 6.3 sind der gemessene Strom und die Lichtintensität der LED11 (siehe Tabelle 5.1) zu sehen.

Die Stromstärke im gepulsten Betrieb entspricht der zu Beginn der Dauerstrommessung gemessenen Stromstärke. Der Unterschied zum Dauerstrom ist jedoch, dass der Strom im gepulsten Betrieb sofort, abgesehen von leichtem Quantisierungsrauschen der Messwerte, stabil ist. Die gemessene Lichtintensität schwankt zwar leicht, sie steigt aber nicht kontinuierlich an, wie bei der Messung bei Dauerstrom. Es ist also durch das gepulste Messverfahren gelungen, Strom und Spannung in diesem Fall zu stabilisieren.

Abbildung 6.4 zeigt Messungen bei gepulstem Strom an der LED51 (siehe Tabelle 5.1).

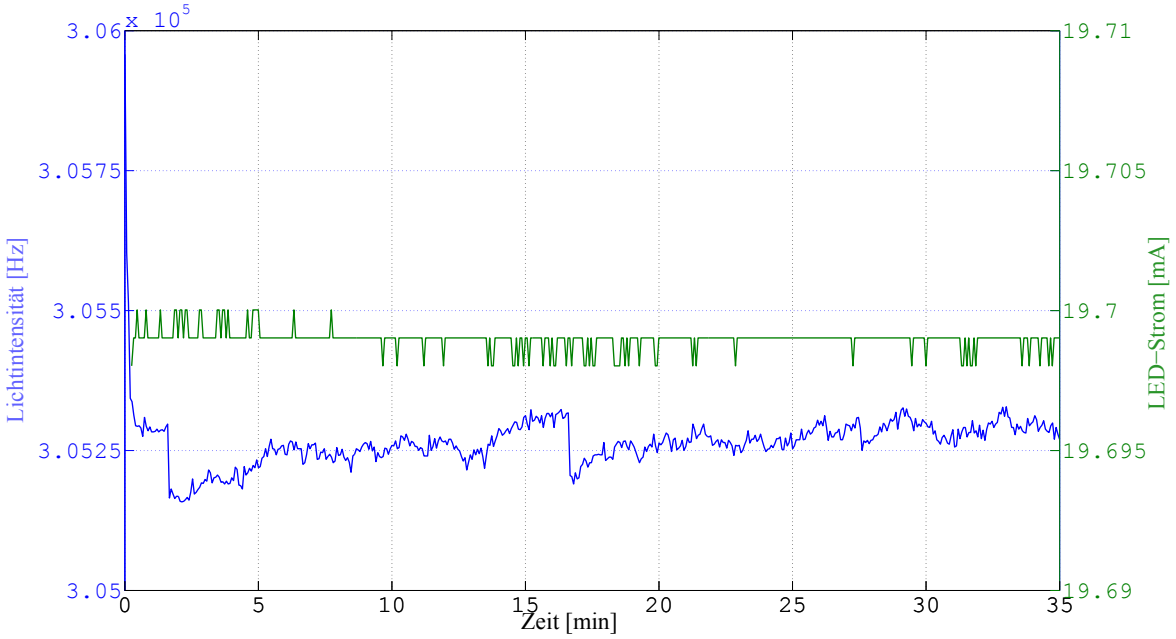

Abbildung 6.3.: Strom und Intensitätsmessung über die Zeit an LED11 (siehe Tabelle 5.1), gepulst mit 2 Sekunden an- und 2 Sekunden ausgeschaltetem Strom

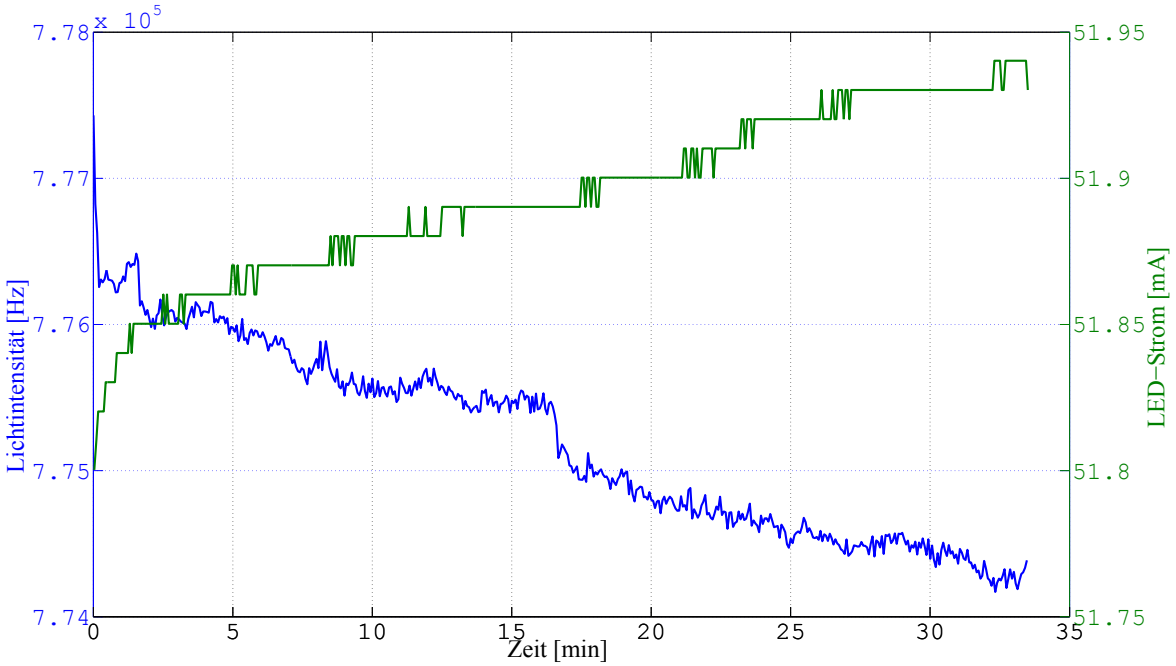

Abbildung 6.4.: Strom und Intensitätsmessung über die Zeit an LED51 (siehe Tabelle 5.1), gepulst mit 2 Sekunden an- und 2 Sekunden ausgeschaltetem Strom

Im Gegensatz zur gepulsten Messung an LED11 werden Strom und Lichtintensität offensichtlich nicht stabilisiert, sondern zeigen nahezu den gleichen Verlauf wie im Betrieb mit Dauerstrom. Der Prozentbereich, in welchem sich der Strom bewegt, ist mit 0,27 % zwar geringer, der Startwert des Stromes bei der gepulsten Messung ist allerdings auch leicht größer.
Ein weiterer sichtbarer Effekt sind die leichten Einbrüche der gemessenen Lichtintensität nach ungefähr zwei und 15 Minuten, welche in allen vier durchgeführten Messungen auftreten. Die Ursache für diese Auffälligkeit wurde nicht näher untersucht und müsste gegebenenfalls überprüft werden. Zur stärkeren Stabilisation des Stromes von LED51 stellt sich die Frage, ob dies durch längere Phasen, in denen die LED ausgeschaltet wird, möglich ist. Dies wäre in zukünftige Arbeiten zu testen.

6.2. Vergleich von Messungen mit dem Spektrometer und mit dem eigenen LED-Reflektometer

Probefarben

Zum Vergleich vom Spektrometer und dem eigenen LED-Reflektometer wurden Messungen an den in Abbildung 6.5 zu sehenden Probefarben durchgeführt. Bei den Probefarben han-

Abbildung 6.5.: Probefarben für Reflexionsmessungen. Die eingekreisten Stellen stellen die Messpunkte dar.

delt es sich um gefärbte Plastikbehälter. Diese wurden mit der Halogenlampe aus Abschnitt 4.1.1 mit der in Abbildung 4.11 gezeigten Vorrichtung für Reflexionsmessungen beleuchtet. Der Reflexionen wurden schon in den Voruntersuchungen mit dem Spektrometer gemessen. In Abbildung 6.6 sind die aufgezeichneten Spektren dargestellt.

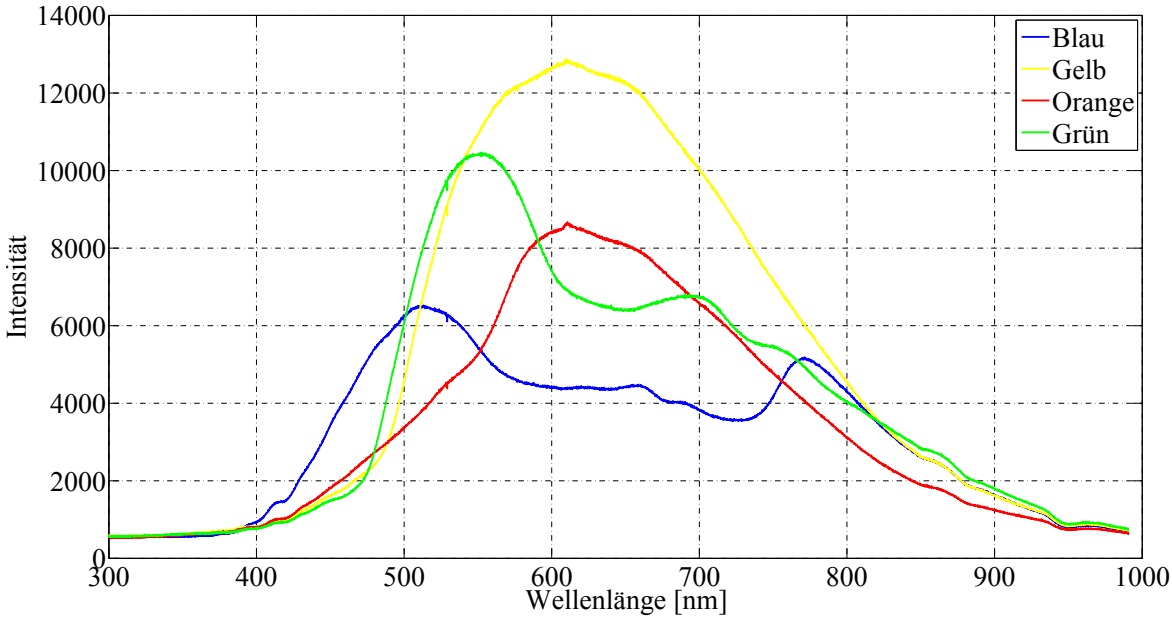

Abbildung 6.6.: Reflexionsmessungen an Probefarben mit dem Spektrometer

Die Reflexionsspektren der Probefarben zeigen deutliche Unterschiede zueinander. Das bläuliche Probestück hat seine maximale Intensität bei circa 510 nm. Vom menschlichen Auge wird Licht bei dieser Wellenlänge als Übergang zwischen Blau und Grün wahrgenommen (siehe Abb. 2.1). Die Reflexion des gelben Probestücks besitzt die höchste Intensität und entspricht mit circa 600 nm seinem optischen Eindruck. Das orange und gelbe Probestück reflektieren das meiste Licht bei einer Wellenlänge von ungefähr 610 nm. Bei der orangefarbenen Probefarbe ist jedoch ein geringerer Anstieg der reflektierten Intensitäten bis ungefähr 550 nm zu erkennen als beim gelben. Dadurch hat es im Verhältnis geringere Gelb- und größere Rotanteile, wodurch der orangefarbene Eindruck entsteht.

Auch mit dem LED-Reflektometer wurden Messungen an denselben Probefarben durchgeführt. Als Lichtquelle diente beim ersten Versuch ebenfalls die im Abschnitt 4.1.1 beschriebene Halogenlampe. Gemessen wurde mit den vier unterschiedlichen Filtereinstellungen klar, rot, grün und blau des in der Platine eingesetzten Licht-/Frequenz-Wandlers. Es wurden jeweils für zehn Sekunden Messungen vorgenommen. Die gemittelten Werte sind in Abbildung 6.7 in einem Balkendiagramm dargestellt.

Die Messergebnisse mit rotem Farbfilter sind immer höher, als die Messungen mit den anderen Filtereinstellungen, selbst ohne Farbfilter. Der Grund dafür sind die Sensitivitäten des

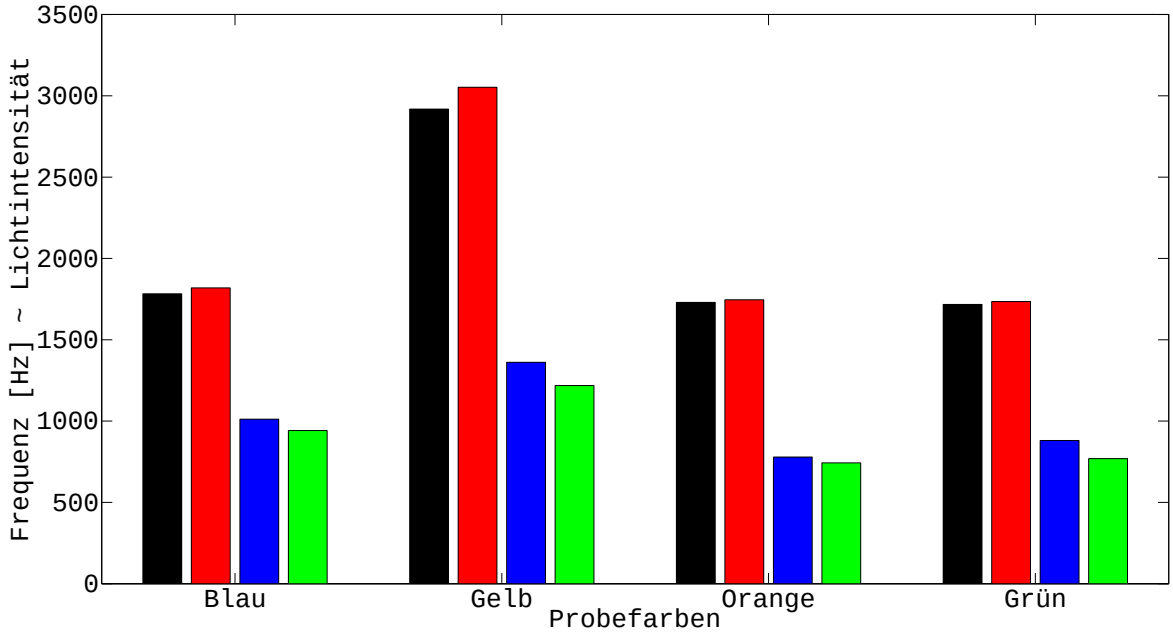

Abbildung 6.7.: Reflexionsmessungen an Probefarben mit dem LED-Reflektometer. Der schwarze Balken zeigt den klaren Farbfilter. Die roten, grünen und blauen Balken entsprechen der Farbfiltereinstellung am Sensor.

Farbsensors, welche in Abbildung 5.15 gezeigt wurden. Laut Sensitivitäten der einzelne Filteroptionen des Lichtsensors sollte dies nicht möglich sein, da die Pixel mit rotem Farbfilter nie sensitiver sind, als die klaren Pixel. Mögliche Ursachen für diesen Effekt könnten Verunreinigungen auf dem Sensor, aber auch ein schräg angeschlossener Toslink-Stecker sein, welcher intensiver auf die roten, als auf die klaren Pixel strahlt.
Der Sensor mit rotem Farbfilter ist sensibel für Wellenlängen von ungefähr 550 nm bis 1000 nm, mit blauem Farbfilter zwischen 380 nm und 550 nm und grünem zwischen 500 nm und 600 nm. In Abbildung 6.8 sind die Überschneidungen der Reflexionsmessung am blauen Messobjekt und der Sensitivitäten der Farbfilter des TAOS TCS3200D zu sehen. Das Diagramm macht deutlich, dass der größte spektrale Anteil des reflektierten Lichtes sich innerhalb der Sensorkennlinie mit rotem Farbfilter befindet.

Auch die starke Ähnlichkeit der Messwerte mit blauem und grünem Farbfilter sind einleuchtend, da alle reflektierten Spektren der Probefarben bis weit in den Infrarotbereich hineinreichen. Dort wirken die Filter nicht mehr, wodurch dort auftretende Wellenlängen in die Messungen mit hineinspielen.

Es wurde versucht, anhand der reflektierten Spektren und den Sensorkennlinien die zu erwartenden Messwerte zu ermitteln. Dafür wurden die aufgenommenen Reflexionsspektren mit den Sensorkennlinien multipliziert. Bei den Messungen musste zusätzlich zur Reflexions-

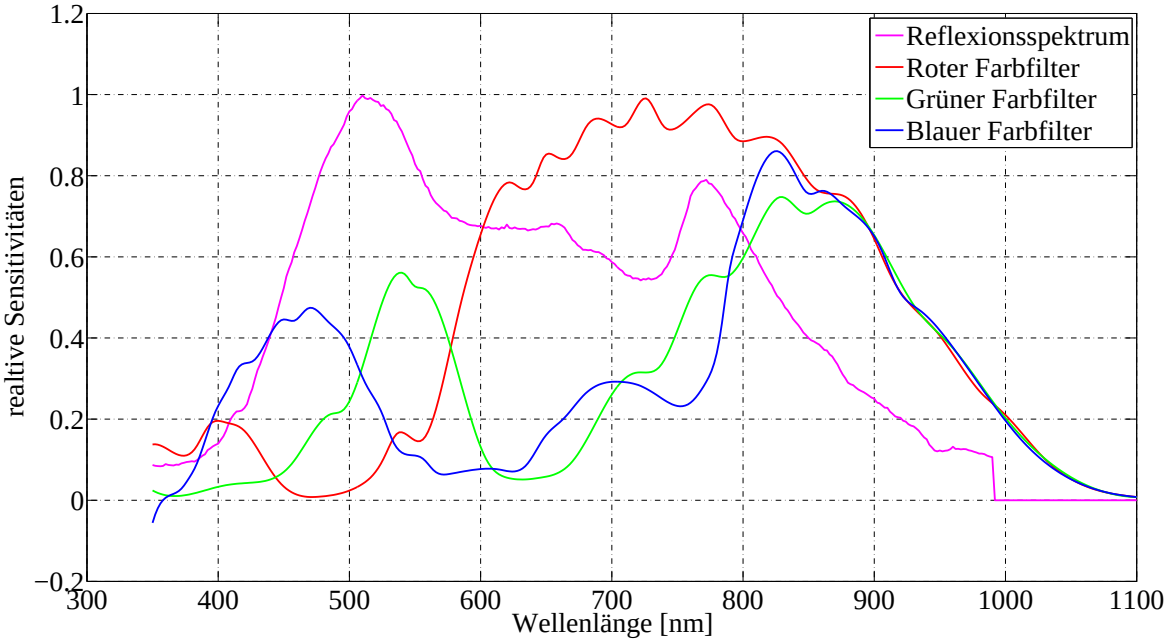

Abbildung 6.8.: Überschneidungen der Sensorsensitivitäten des Farbsensors TAOS TCS3200D und der Reflexionsmessung an der blauen Referenzfläche

Messvorrichtung, welche aus Glasfasern besteht, ein POF-Leiter angeschlossen werden, damit das reflektierte Licht am Farbsensor gemessen werden konnte.
Dies wurde versucht mit in die Berechnungen mit einzubeziehen, indem die Kennlinien mit der Absorptionsmessung einer POF-Faser aus Abbildung 4.5 multipliziert wurden. Dies ist so aber nicht richtig, da dadurch die Eigenschaften der Halogenlampe zweimal einbezogen werden. Stattdessen wäre es besser gewesen, die Messungen mit einer POF-Faser am Spektrometer durchzuführen. Durch unpassende Anschlüsse wurden die reflektierten Lichtintensitäten bei entsprechenden Versuchen aber zu stark gedämpft.
Die berechneten und die gemessenen Werte sind, normiert auf den Messwert mit dem blauen Farbfilter, in Tabelle 6.2 dargestellt.

Die gemessenen und berechneten Werte stimmen nicht vollkommen überein. Aber die Tendenzen der jeweils zusammengehörenden Verhältnisse sind fast immer gleich. So sind der normierte rote Messwert und der berechnete Wert der gelben Farbprobe in beiden Fällen der größte, der Messwert mit Grünfilter der zweitgrößte. Der kleinste gemessene und berechnete Wert am roten Farbfilter ist jeweils die blaue Farbprobe. Nur der kleinste berechnete und gemessene normierte Wert am Grünfilter stimmen nicht miteinander überein.

Bessere Ergebnisse lassen sich mit Sicherheit erzielen, indem der Einfluss der POF-Faser auf bessere Weise, zum Beispiel mit zur Faser passenden Absorptionsspektren wie in Abbildung 2.6, berücksichtigt wird.

Farbprobe	Mess. Rot/Blau	Mess. Grün/Blau	Berechn. Rot/Blau	Berechn. Grün/Blau
Gelb	2,51	1,1	2,6	1,25
Orange	2,35	1,05	2,46	1,15
Grün	2,26	1,15	2,13	1,27
Blau	1,93	1,07	1,73	1,09

Tabelle 6.1.: Normierte Messungen des LED-Reflektometers und Berechnungn der erwarteten Messwerte. Die Messungen wurden auf den Messwert des blauen Farbfilters normiert. Die Berechnungen wurden anhand der Reflexionsspektren von den Farbproben und eines POF-Leiters und den Sensorkennlinien des Farbsensors durchgeführt.

Trotzdem zeigen die Messungen, dass der verwendete Farbsensor prinzipiell so arbeitet, wie es von ihm erwartet wird.

Messungen an einer Methylenblaulösung

Besser auszuwerten sind Messungen bei der Verwendung von Lichtquellen mit schmalbandigen Spektren.

Wie in Abschnitt 4.3 wurden mit dem LED-Reflektometer Messungen an einer Methylenblaulösung durchgeführt. Anstelle der Halogenlampe wurde jedoch die RGB-LED (LED10, LED11 und LED12) der Messplatine zur Durchführung der Messungen verwendet. Abbildung 6.9 zeigt den entsprechenden Messaufbau mit dem LED-Reflektometer und im Hintergrund den Behälter für die zu messende Flüssigkeit.

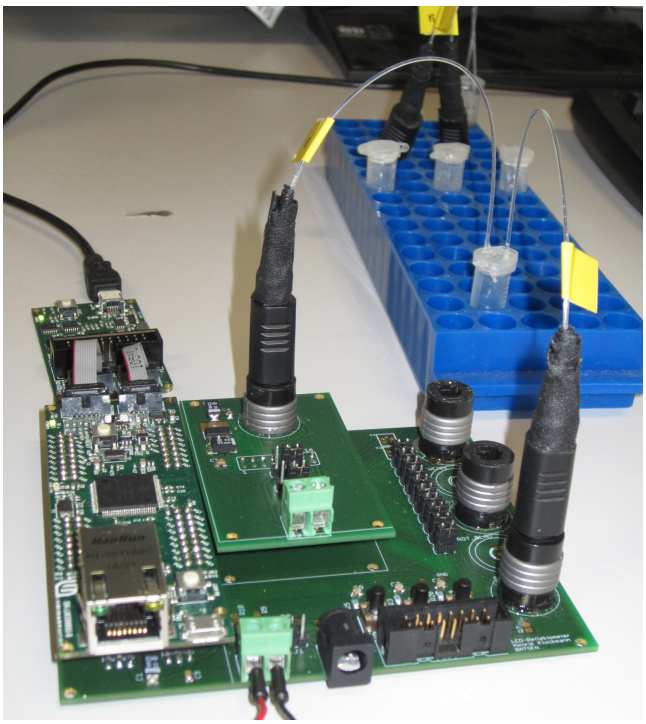

Abbildung 6.9.: Messaufbau für Messungen an der Methylenblaulösung. Im Vordergrund ist das LED-Reflektometer zu sehen, im Hintergrund ein Reagenzglas aus Plastik, in welchem der Lichtleiter zur Messung eingeführt ist.

Die Durchführung der Messungen erfolgte mit derselben Faser (Beschriftung "3"), welche bei den entsprechenden Messungen mit dem Spektrometer verwendet wurde. Erst wurde das Reagenzglas vollständig mit einer Methylenblaulösung gefüllt, anschließend Messungen abwechselnd mit den drei Farben der RGB-LED durchgeführt. Die An-/Aus-Zyklen der LED waren auf je zwei Sekunden eingestellt, die gesamte Messdauer betrug 60 Sekunden. Anschließend wurde die Flüssigkeit wiederholt entfernt und das Reagenzglas wieder vollständig mit Wasser gefüllt. Nach jedem Auffüllen des Reagenzglases wurden erneut Messungen

durchgeführt. Die Messdaten wurden mit Matlab ausgewertet und gemittelt. In Abbildung 6.10 sind die Verläufe der gemessenen Lichtintensitäten über die Anzahl der Verdünnungen zu sehen.

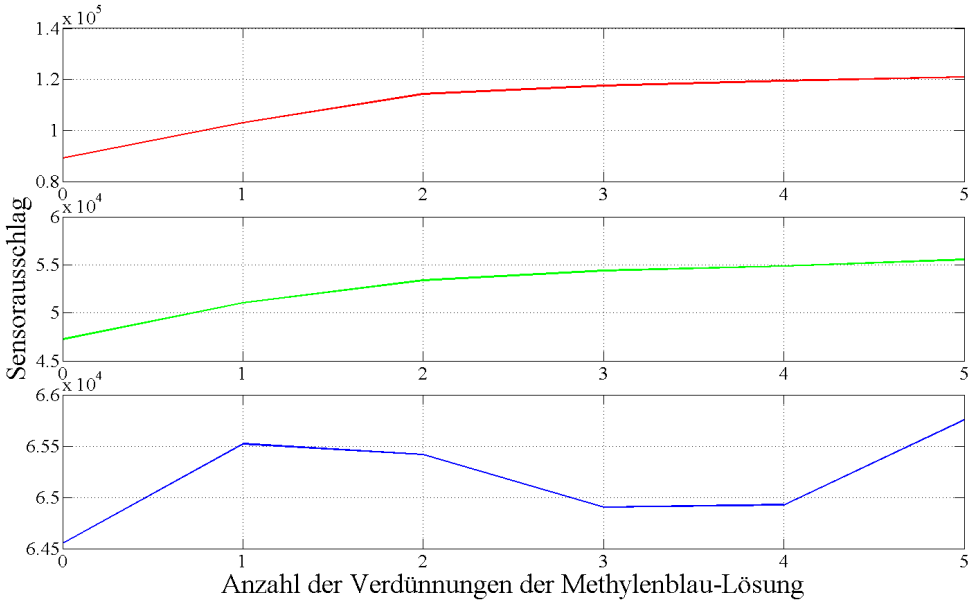

Abbildung 6.10.: Messungen an einer Methylenblaulösung mit dem LED-Reflektometer. Die Farben der Graphen entsprechen den Farben der RGB-LED (LED1) aus Tabelle 5.1

Die mit der roten und grünen LED durchgeführten Messungen zeigen einen Anstieg der absoluten Lichtintensität am Sensor bei Verdünnung der Methylenblaulösung. Im Gegensatz dazu hat die Verdünnung, also eine Verringerung des Blauanteils der Lösung, einen weitaus geringeren Einfluss auf die Absorption blauen Lichtes.

Angenommen, die Verdünnungen haben keinen Einfluss auf die Absorptionseigenschaften bei den Wellenlängen der blauen LED. Die Schwankungen der gemessenen Lichtintensität würden nur aufgrund unterschiedlicher Lichtkopplung an den Toslink-Steckern und leichter Änderung des Biegeradius' des Lichtleiters zu Stande kommen. Für diesen Fall würden sich die Messungen mit der blauen LED hervorragend als Referenz zur Normierung der Messwerte der roten und grünen LED anbieten. Die daraus resultierenden Graphen sind in Abbildung 6.11 gezeigt.

Im Vergleich zu den nicht normierten Messungen bleiben die Verläufe der Intensitäten prinzipiell erhalten. Nur zwischen der vierten und fünften Verdünnung legen die normierten Messwerte die Vermutung nahe, dass sich durch die letzte Verdünnung der Methylenblaulösung

keine Veränderungen der Absorptionseigenschaften mehr einstellen. Dies gilt jedoch nur für den Fall, dass die gemessenen Intensitäten des Lichtes der blauen LED nur Veränderungen aufgrund von Störungen erfährt. Um dies genauer bewerten zu können wären weitere Messungen nötig, welchen den Einfluss von Störungen durch Bewegung von Messapparatur und Toslink-Steckern näher untersuchen.

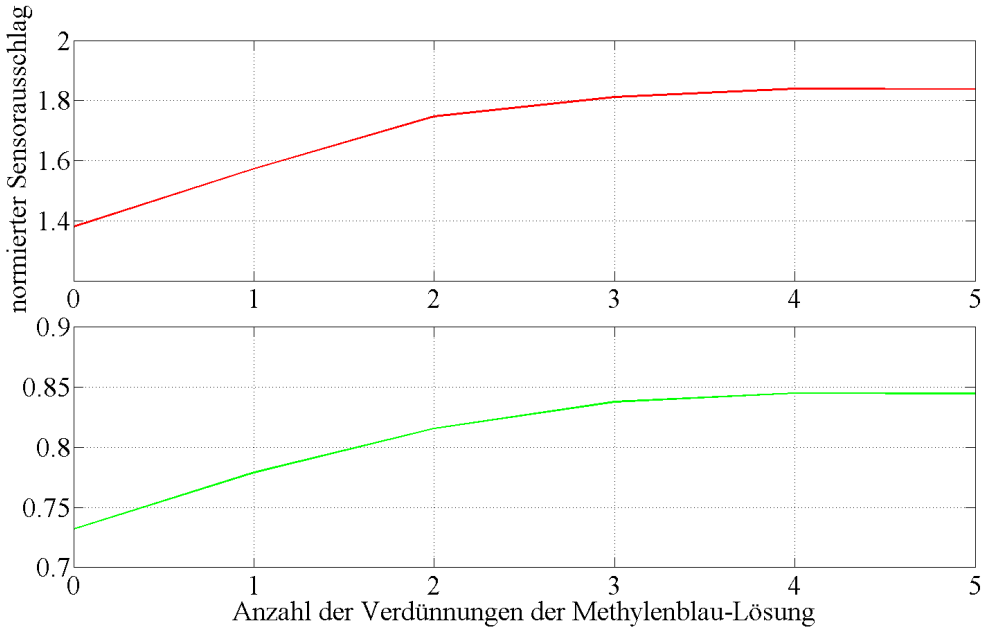

Abbildung 6.11.: Auf blaue LED normierte Messungen an einer Methylenblaulösung

7. Einordnung, Bewertung und Ausblick

7.1. Zusammenfassung der Ergebnisse

Zu Beginn dieser Bachelorarbeit wurde das kommerzielle Spektrometer LR1-T von ASEQ-Instruments in Betrieb genommen (Abschn. 4.1). Es diente zur Beobachtung von unterschiedlichen Absorptionseigenschaften verschiedener Materialien. Im Vordergrund sollten die Bestimmung der optischen Eigenschaften von Lithium-Eisenphosphat und Tests bezüglich des optischen Messverfahrens stehen.
Beispielhaft überprüft wurde die Messmethode an einer Methylenblaulösung (Abschn. 4.3). Bei Verdünnung verändern sich die Absorptionseigenschaften der Methylenblaulösung. Erprobt wurde das Messverfahren an verschieden bearbeiteten Lichtleitern. Alle getesteten Lichtleiter waren innerhalb der zu messenden Lösung gebeugt. Einer von ihnen war ansonsten nicht weiter bearbeitet. Bei zwei Lichtleitern wurde an der Beugung innerhalb der Lösung der Mantel entfernt, bei einem mit groben Schmirgelpapier, bei dem anderen mit feinem. Die besten gemessenen Ergebnisse zeigten sich bei dem fein abgeschmirgelten Lichtleiter, da sich hier die größten Differenzen zwischen den Messungen bei unterschiedlichen Verdünnungsgraden einstellen. Mit dem Spektrometer wurden auch Reflexionsmessungen an Eisenphosphat und lithiniertem Eisenphosphat als Feststoff durchgeführt (Abschn. 4.2). Sie zeigten messbare Veränderungen der unterschiedlichen Proben hinsichtlich ihrer Reflexionseigenschaften. Die gewonnenen Erkenntnisse reichten jedoch nicht aus um festzustellen, bei welchen Wellenlängen Messungen die besten Ergebnisse liefern würden.

Anhand der Voruntersuchungen wurde beschlossen, eine möglichst flexible Messplatine zu erstellen, welche unterschiedliche LED-Kombinationen auf eine einzelne Faser einkoppeln kann (Abschn. 5.2). Zum Betreiben der LEDs wurde ein LED-Treiber mit 16 Kanälen verwendet. Dieser ermöglicht es, den fließenden Strom pro Kanal flexibel per Software zu konfigurieren.
Der Lichtsensor wurde auf einer separaten Platine untergebracht, welche über eine Stiftleiste angeschlossen werden kann. Auf diese Weise ist es möglich, Sensor und LEDs örtlich zu trennen, falls dies durch den Aufbau einer zu messende Probe erforderlich ist. In diesem Fall lässt sich die Sensorplatine mit einem Flachbandkabel mit der restlichen Platine verbinden. Außerdem kann die Messplatine leicht ersetzt werden. Dadurch besteht die Möglichkeit,

einen alternativen Lichtsensor oder zusätzliche Sensoren, zum Beispiel zur Messung von Temperaturen, einzusetzen.

Die realisierte Umsetzung der Controller- und PC-Software ermöglicht es, jede verbaute LED einzeln einzuschalten. Bei mehreren ausgewählten LEDs werden diese nacheinander eingeschaltet. Um die gelieferte Stromstärke des Treibers und die Lichtintensität der LEDs zu stabilisieren, wurde die Möglichkeit geschaffen, An- und Aus-Zyklen zu konfigurieren. Eine LED wird also für die konfigurierte Zeit ein- und anschließend ausgeschaltet, bevor die nächste ausgewählte LED aktiviert wird.

Gegen Ende dieser Bachelorarbeit wurden Tests durchgeführt, welche die Funktionstüchtigkeit der Messplatine überprüfen sollten (Abschn. 6). So wurde gezeigt, dass es möglich ist, durch das zyklische An- und Ausschalten der LEDs die abgestrahlte Lichtleistung zu stabilisieren. Es wurden auch, wie schon mit dem Spektrometer, Messungen mit Probefarben und einer Methylenblaulösung durchgeführt, welche die grundsätzliche Funktionstüchtigkeit des LED-Reflektometers unter Beweis stellen. Messungen an lithiniertem Eisenphosphat konnten aufgrund von Zeitmangel und fehlenden Proben nicht mehr durchgeführt werden.

7.2. Bewertung der gewählten Konzepte und Lösungsvarianten

In diesem Kapitel sollen die in dieser Bachelorarbeit gefällten Entscheidungen, erstellten Konzepte und gewählte Lösungsvarianten beurteilt werden.
Zum Anschluss der Lichtwellenleiter an Lichtquellen und Sensoren war es notwendig, ein geeignetes Steckersystem auszuwählen. Die Wahl fiel auf Toslink-Stecker. Diese sind sehr gut geeignet zur einfachen Unterbringung von LEDs. Einkopplungsverluste aufgrund fehlender fokussierender Optik wurden durch möglichst intensive LEDs ausgeglichen. Trotzdem kann noch nicht genau beurteilt werden, welche LEDs und welche Intensitäten letztendlich in $LiFePO_4$-Batterien als Lichtquellen zur Messung am besten geeignet sind. Dafür ist es notwendig, das Übertragungsverhalten der in dem Material eingebetteten Lichtleiter näher zu untersuchen. Da die Rahmenbedingungen noch nicht ausreichend untersucht sind, war es in diesem Fall die richtige Entscheidung, das Platinenlayout flexibel zu gestalten. Es können verschiedene LEDs mit unterschiedlichen Montagearten und Vorwärtsströmen flexibel eingesetzt werden.
Auch die Sensorik durch den gebogenen und am Mantel bearbeiteten Lichtleiter ist in den Tests positiv aufgefallen. Durch das Entfernen des Mantels ist jedoch nur der POF-Leiter verwendbar gewesen. Die Glasfaser, welche sich durch ihre bessere Lichtleitfähigkeit auszeichnet (vgl. 2.1.3), wurde durch das Entfernen der äußeren Schutzschicht und des Mantels sehr brüchig und konnte daher nicht für die Messungen eingesetzt werden.

Die durch Controller-Software geschaffene Möglichkeit zum gepulsten An- und Ausschalten der LEDs ermöglicht, wie in Abschnitt 6.1 gezeigt, die Stabilisation des durch den LED-Treiber gelieferten Stromes und der Lichtintensität. Der gepulste Strom führt aber nicht in jedem Fall zu einer stabileren Stromstärke, wie in Abbildung 6.4 gezeigt wurde. Es besteht die Möglichkeit, dass durch längeres Ausschalten der LED auch diese stabilisiert werden kann. Es wäre jedoch für die Zukunft eventuell besser, nach weiteren Methoden zur Stabilisation zu suchen.

Insgesamt bietet die entworfene Platine eine gute Basis für weitere Tests bezüglich der Umsetzung eines Sensors zur Messung der Absorptionseigenschaften des Elektrodenmaterials eines $LiFePO_4$-Akkus. Nach weiteren entsprechenden Untersuchungen kann das umgesetzte Platinenlayout weiterentwickelt, vereinfacht und gegebenenfalls in die Sensoren des BATSEN-Projektes integriert werden.

Dafür sind insbesondere intensivere Untersuchungen der spektralen Eigenschaften von LiFePO4 bei unterschiedlichen Lithinierungen notwendig, welche bereits in dieser Bachelorarbeit vorgesehen waren.

7.3. Offene Punkte und Alternativen als Basis für weitere Arbeiten

Die Umsetzung eines Sensors zur Messung der optischen Spektraleigenschaften des Kathodenmaterials einer $LiFePO_4$-Zelle befindet sich erst am Anfang. Es sind noch viele weitere Tests und Untersuchungen notwendig. Einige waren schon im Zuge dieser Bachelorarbeit (Anhang A) vorgesehen, wurden aber aus Zeitmangel nicht durchgeführt.

So war angedacht, einen Lichtleiter mit Eisenphosphat mittels Bindemittel zu ummanteln. Dies sollte geschehen, nachdem eine Faser mit Lithium-Eisenphosphat mittels Sputterverfahren beschichtet wurde. Die Herausforderung beim Ummanteln der beschichteten Faser ist zum einen, dass erreicht werden muss, dass die auf den Lichtleiter aufgetragene Lihtium-Eisenphosphat-Schicht an chemischen Reaktionen im ummantelten Eisenphosphat teilnehmen kann. Nur so können die Veränderungen der spektralen Eigenschaften bei unterschiedlicher Lithinierung mittels Lichtleitersensorik gemessen werden.

Nach der erfolgreichen Ummantelung eines bearbeiteten Lichtwellenleiters ist es notwendig, Proben herzustellen, welche unterschiedlich stark lithiniert sind. Mit diesen Proben sollten dann Messungen mit einem kommerziellen und präzisem Spektrometer durchgeführt werden. Anhand der Messungen sollte sich das spektrale Verhalten bei unterschiedlichen Lithinierungen zeigen. Sind sinnvolle Lichtwellenlängen für die Messungen gefunden, so können sie mit dem aktuellen Messaufbau überprüft werden.

Anschließend kann die Platine angepasst und vereinfacht werden, da die derzeit gebotene Flexibilität dann nicht mehr benötigt wird.
In dem Fall müsste der aktuell eingesetzte LED-Treiber ersetzt werden, da er mit seinen 16 Kanälen stark überdimensioniert wäre. Der LED-Treiber könnte durch einen kleineren ersetzt werden, welcher gegebenenfalls in der Lage ist, einen präziseren Konstantstrom zu liefern. Neben der Verwendung eines bereitgestellten IC's, gibt es auch die Möglichkeit selbst eine Konstantstromquelle zu entwickeln, zum Beispiel mit dem Einsatz einer hochpräzisen Referenzspannungsquelle. Integrierte Lösungen sind jedoch in der Regel die bessere Wahl, zum Beispiel der LED-Treiber AS3691 von austrianmicrosystems, welcher im Datenblatt mit einer absoluten Genauigkeit des Stromes von $\pm$ 0.5 % angegeben wird [54].

Für eine industrielle Anwendung muss außerdem sichergestellt werden, dass die Faser bei der Einbringung in die Zelle nicht beschädigt wird. In der Industrie wird bei der Elektrodenfertigung ein Verdichtungsverfahren eingesetzt, bei welchem die mit Material beschichteten Elektroden durch Walzen bei mehreren Tonnen Druck verdichtet werden. Dadurch wird sichergestellt, dass eine gleichbleibende Elektrodendicke und Stabilität erzielt wird. Bei solch einer Methode wären Beschädigungen an der Faser kaum zu vermeiden. Es muss daher eine passende Vorgehensweisen gefunden werden, den Lichtleiter mit dem Lithium-Eisenphosphat der Elektrode zu verbinden.

7.4. Beitrag und Einordnung in das Forschungsvorhaben

Diese Bachelorarbeit stellt für das Projekt BATSEN den Einstieg in die optische Sensorik hinsichtlich der spektralen Analyse von Kathodenmaterialien von $LiFePO_4$-Akkus dar. Sie konzentrierte sich daher zu einem großen Teil auf grundlegende Fragen der Sensorik mittels Lichtwellenleitern.

Durch diese Arbeit hat das Projekt an Erfahrungen mit dem Umgang mit Lichtleitern gewonnen. Es wurde festgestellt, dass zum Beispiel Glasfasern keine gute Alternative darstellen, solange der Mantel der Faser entfernt wird. POF-Fasern dagegen können für unsere Zwecke, zumindest nach dem aktuellen Kenntnisstand, für die vorgesehene Messmethode eingesetzt werden.

Es wurden Erfahrungen mit speziellen Bauteilen gesammelt, welche bereits in anderen Arbeiten des BATSEN-Projektes angewandt werden und in der späteren Sensorintegration eine Rolle spielen können [21].

Mit dem erstellten LED-Reflektometer ist insgesamt eine Basis entstanden, auf welcher weitere Tests zur Untersuchung von Lithium-Eisenphosphat durchgeführt werden können. Außerdem stellt das Platinenlayout einen Ausgangspunkt für Weiterentwicklungen und weitere Messvorhaben dar.

Literaturverzeichnis

[1] W. Weydanz and A. Jossen, “Moderne Akkumulatoren richtig einsetzen,” 2006.

[2] K.-R. Riemschneider and M. Schneider, “Drahtlose Sensoren in den Zellen von Fahrzeug-Batterien.”

[3] S. J. Harris, A. Timmons, D. R. Baker, and C. Monroe, “Direct *in situ* measurements of Li transport in Li-ion battery negative electrodes,” *Chemical Physics Letters*, vol. 485, no. 4, pp. 265–274, 2010. [Online]. Available: http://lithiumbatteryresearch.com/pdf/Colors.pdf

[4] Landesakademie für Fortbildung und Personalentwicklung an Schulen , “Licht als Welle.” [Online]. Available: http://lehrerfortbildung-bw.de/kompetenzen/gestaltung/farbe/physik/welle/spektrum.html

[5] O. Strobel, *Lichtwellenleiter-Übertragungs-und Sensortechnik*. Vde-Verlag, 2002.

[6] Rwth Aachen, “Elektromagnetische Wellen.” [Online]. Available: http://web.physik.rwth-aachen.de/~hebbeker/lectures/ph2_02/p202_l05.htm

[7] ITWissen, “Das große Online-Lexikon für Informationstechnologie.” [Online]. Available: http://www.itwissen.info/

[8] W. Bludau, *Lichtwellenleiter in Sensorik und optischer Nachrichtentechnik*. Springer, 1998.

[9] Wikibooks, “Optik.” [Online]. Available: http://de.wikibooks.org/wiki/Optik

[10] H. Kuchling, *Taschenbuch der Physik*. Fachbuchverlag Leipzig im C. Hanser Verlag, 2004.

[11] PCP, “Querschnitt eines Lichtwellenleiters.” [Online]. Available: http://www.pcp.ch/Triotronik-Lightwin-LWL-Patchkabel-LC-LC-50-125u-5m-1a15916117.htm

[12] Fiber Optics For Sale Co., “Eintritt von Licht in ein Lichtwelllenleiter.” [Online]. Available: www.fiberoptics4sale.com

[13] Wikipedia, “Lichtwellenleiter.” [Online]. Available: http://de.wikipedia.org/wiki/Lichtwellenleiter

[14] P. Avakian, W. Hsu, P. Meakin, and H. Snyder, “Optical absorption spectrum of PMMA via laser calorimetry,” *Journal of Polymer Science: Polymer Physics Edition*, vol. 21, no. 4, pp. 647–655, 1983.

[15] All About Circuits, “Optical data communication.” [Online]. Available: http://www.allaboutcircuits.com/vol_4/chpt_14/5.html

[16] Uni Bonn, “Der photoelektrische Effekt.” [Online]. Available: http://www.iap.uni-bonn.de/P2K/quantumzone/photoelectric.html

[17] Uni Graz, “Der photoelektrische Effekt.” [Online]. Available: http://physik.uni-graz.at/~cbl/QM/contents/Projekte_2004/p1/G7_Photoeffekt.pdf

[18] Wikipedia, “Photoelektrischer Effekt.” [Online]. Available: http://de.wikipedia.org/wiki/Photoelektrischer_Effekt

[19] A. . K. Padhi, K. Nanjundaswamy, and J. B. d. Goodenough, “Phospho-olivines as Positive-Electrode Materials for Rechargeable Lithium Batteries,” *Journal of the Electrochemical Society*, vol. 144, no. 4, pp. 1188–1194, 1997. [Online]. Available: http://dx.doi.org/10.1149/1.1837571

[20] T. B. Reddy, *Linden’s Handbook of Batteries*. McGraw-Hill, 2011, vol. 4.

[21] W. Nasimzada, “Hard- und Softwareentwicklung eines Lichtleiter-Sensors für die optische Analyse des Elektrolyten von Bleibatterien ,” Bachelor’s thesis, Hochschule für Angewandte Wissenschaften Hamburg, Germany, 2013.

[22] M. Albert, “In vivo Validierung eines neuen Verfahrens zur Pulsoxymetrie im niedrigen Sauerstoffsättigungsbereich,” Ph.D. dissertation, lmu, 2008.

[23] WHO, “Using the Pulse Oximeter.” [Online]. Available: www.who.int/entity/patientsafety/safesurgery/pulse_oximetry/who_ps_pulse_oxymetry_tutorial2_advanced_en.pdf

[24] V. Roscher, “Advanced Built-In Battery Sensors.”

[25] Aseq-Instruments, “LR1 - Spektrometer.” [Online]. Available: http://www.aseq-instruments.com/

[26] Toshiba, “TCD1304DG - CCD-Sensor.” [Online]. Available: http://www.eureca.de/datasheets/01.xx.xxxx/01.04.xxxx/01.04.0080/TCD1304DG.pdf

[27] StellarNet Inc, “SL1 Tungsten Halogen.” [Online]. Available: http://www.stellarnet-inc.com/

[28] Fluke, *45 - Multimeter Manual*, 1999. [Online]. Available: http://www.fluke.com/Fluke/usen/Support/Manuals/default.htm?ProductId=56082

[29] D. A. P. Prof. Dr. Bruno K. Meyer, “Sputtern.” [Online]. Available: http://meyweb.physik.uni-giessen.de/1_Forschung/Sputtern/Sputtern.html

[30] Toshiba, *TOTX173 - Fiber optic transmitting module*, 2001. [Online]. Available: http://www.jitter.de/pdfextern/TOTX173.pdf

[31] Amazon, “Toslink Buchse.” [Online]. Available: http://ecx.images-amazon.com/images/I/41GTUymU0vL._SY300_.jpg

[32] all4fiber, “Toslink Stecker.” [Online]. Available: http://www.all4fiber.de

[33] LUMEX, *DUO-LED infrared/red*, 2001. [Online]. Available: http://www.mouser.com/ds/2/244/SSL-LX5099IEW-121444.pdf

[34] KT-Ecectronic, “5mm RGB LED.” [Online]. Available: http://www.ebay.de/itm/25-Stuck-LED-5mm-RGB-4-Pin-ultrahell-gem-Plus-10000mcd-/261097436440?pt=Bauteile&hash=item3cca9eb518

[35] OSRAM, “SFH4350 - 3mm LED Infrarot.” [Online]. Available: http://catalog.osram-os.com/catalogue/catalogue.do;jsessionid=7BD2FAFDDFDF198C38F025D6B6EFE759?act=downloadFile&favOid=020000030000cb2f000100b6

[36] Kingbright, “WP710A10SECJ4 - 3mm LED Rot.” [Online]. Available: http://www.mouser.com/ds/2/216/WP710A10SEC-J4-83645.pdf

[37] OSRAM, “LA E63F-EBGA-24-3A4B-Z - SMD LED Rot.” [Online]. Available: http://catalog.osram-os.com/catalogue/catalogue.do;jsessionid=1F8BEC6E00FEC1B259D9061816CA04C5?act=downloadFile&favOid=020000030001d405000100b6

[38] OSRAM, “SFH4248-Z - SMD LED Infrarot.” [Online]. Available: http://www.mouser.com/ds/2/311/FH4248_9_Pb_free-63782.pdf

[39] Sharp, “Application Note - Photodiode/Phototransistor Application Circuit.” [Online]. Available: http://physlab.lums.edu.pk/images/1/10/Photodiode_circuit.pdf

[40] Texas Instruments, “TIA Application Note TI.” [Online]. Available: http://www.ti.com/lit/an/sboa055a/sboa055a.pdf

[41] FH Emden, “Bauelemente der Optoelektronik .” [Online]. Available: http://www.et-inf.fho-emden.de/~elmalab/bauelement/download/BdE_9.pdf

[42] *Fotowiderstand - A1060*. [Online]. Available: http://www.produktinfo.conrad.com/datenblaetter/175000-199999/183563-da-01-de-A1060.pdf

[43] OSRAM, *SFH203 - Photodiode*, 2011. [Online]. Available: http://catalog.osram-os.com/catalogue/catalogue.do?act=downloadFile&favOid=02000003000150a5000100b6

[44] OSRAM, *SFH309 - Phototransistor*, 2007. [Online]. Available: http://catalog.osram-os.com/catalogue/catalogue.do;jsessionid=C41F6122F57893783137B84654B9DC1F?act=downloadFile&favOid=0200000200006da3000200b6

[45] Kingbright, *KPS-5130PD7C - RGB Photodioden*, 2013. [Online]. Available: http://www.kingbright.com/manager/upload/pdf/(1369107801)KPS-5130PD7C(Ver.11).pdf

[46] TAOS, *TSL250R - Licht-/Spannungs-Wandler* , 2007. [Online]. Available: http://ams.com/eng/content/download/250406/976421/142378

[47] TAOS, *TCS3200, TCS3210 PROGRAMMABLE COLOR LIGHT-TO-FREQUENCY CONVERTER*, 2011. [Online]. Available: http://www.ams.com/eng/content/download/250259/976005/142755

[48] Texas Instruments, *Stellaris® LM3S9B92 Evaluation Kit Users Manual*, 2011. [Online]. Available: http://www.ti.com/lit/ug/spmu035c/spmu035c.pdf

[49] Texas Instruments, *Stellaris® LM3S9B92 Microcontroller DATA SHEET*, 2012. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm3s9b92.pdf

[50] Texas Instruments, *TLC5940 - 16 channel LED driver with dot correction and grayscale pwm control*, 2007. [Online]. Available: http://www.ti.com/lit/ds/slvs515c/slvs515c.pdf

[51] J. G. Ganssle, “A guide to debouncing,” 2004.

[52] Mikrocontroller.net, “Mikrocontroller.net - Entprellung.” [Online]. Available: http://www.mikrocontroller.net/articles/Entprellung

[53] Texas Instruments, *TLC5940 Programing Flow Chart v0.1*, 2005. [Online]. Available: http://www.ti.com/lit/sw/slvc106/slvc106.pdf

[54] ams, *AS3691 - Backlighting IC*, 2007. [Online]. Available: http://ams.com/eng/content/download/250406/976421/142378

Abkürzungsverzeichnis

ADC	Analog-Digital-Converter
BATSEN	Drahtlose Zellensensoren für Fahrzeugbatterien
$FePO_4$	Eisenphosphat
HCS	Hard-clad Silica
IC	Integrated Circuit
ISR	Interrupt Service Routine
LED	Light-emitting Diode
$LiFePO_4$	Lithium-Eisenphosphat
PLL	Phase-locked Loop
PMMA	Polymethylmethacrylat, Plexiglas
POF	Plastic Optical Fibre
PWM	Pulse-width Modulation
QEI	Quadrature Encoder Interface
RGB	Rot Grün Blau
SMD	Surface-mounted Device
SOC	State of Charge
SOH	State of Health
TIA	Transimpedance Amplifier
UART	Universal Asynchronous Receiver Transmitter

A. Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

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

18. Juni 2013

Bachelorthesis Henrik Klockmann

Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern

Motivation

Im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten Forschungsvorhabens BATSEN (drahtlose Zellensensoren für Fahrzeugbatterien) und der Graduiertenschule 'Key Technologies for Sustainable Energy Systems in Smart Grids' soll der Zustand von Batterien durch Sensorik erfasst werden.
Für moderne Lithiumbatterien besteht eine noch zu schließende Lücke zwischen dem Bedarf an genauer Erfassung des Batteriezustandes aus Gründen der Sicherheit und wirtschaftlichen Ausnutzung der Batterie einerseits und der relativ schlechten Beobachtbarkeit des Zustandes durch elektrische Messwerte andererseits. Die einfach erfassbaren Größen wie Klemmspannung, Batteriestrom und Temperatur sollen um eine physikalische/chemische Beobachtung des Elektrodenmaterials ergänzt werden. In der Arbeitsgruppe entstand der neuartige Vorschlag, Lichtleiter in den Batterie-Innenraum einzubringen und für optisch-spektrale Messungen zu nutzen.

Aufgabe

Herr Klockmann erhält die Aufgabe, für Lithium-Eisenphosphat-Kathoden die Anwendbarkeit eines optischen Messverfahrens experimentell zu prüfen. Das Verfahren soll prinzipiell 'in-situ' in den laufenden Batteriebetrieb einzubringen sein. Es soll ebenso vom technischen Aufwand her so minimiert werden, dass einer wirtschaftlichen Nutzung keine absehbaren Hindernisse entgegen stehen. Daher sollen LED und kostengünstige Sensoren genutzt werden, ebenso wie handelsübliche Lichtleiter auf Kunststoff- oder Glasfaserbasis und deren Verbindungstechnik. Das in der Messtechnik übliche Verfahren von Referenzmessungen soll sinngemäß genutzt werden. Das Messsystem soll mit Mikrocontroller und Software gesteuert werden.

Für die Abschlussarbeit sind die folgenden Arbeitspakete geplant:

1. Vorarbeiten für optische Messverfahren
    - Inbetriebnahme des Spektrometers und Matlab-Scripte zur Auswertung
    - Reflektometrie an Eisenphosphat und Lithium-Eisenphosphat als Feststoff
    - Evaneszenzfeldspektroskopie zum Vergleich an Flüssigkeiten
2. Entwicklung des Messaufbaus
    - Mechanischer Aufbau von LED, Lichtsensoren und Steckern für Lichtleiter
    - Spektrometer-Charakterisierung von Lichtquellen und -sensoren
    - Fixierung und Präparation des Lichtleiters für den Lichtaustritt
    - Eisenphosphat-Sputterbeschichtung eines Lichtleiters
    - Ummantelung des Lichtwellenleiters mit Eisenphosphat mit Bindemittel

3. Laboraufbau eines vereinfachten LED-Reflektometers

    - Auswahl und Inbetriebnahme eines Mikrocontroller-Evaluationsboards
    - Erprobung und Auswahl von Stecker, Lichtquellen und Sensoren
    - Schaltungs- und Platinenlayout als Basis für den Anschluss der Lichtleiter
    - Software zur Steuerung der optischen Messung und Datenerfassung
    - Einbindung von Referenzmessungen in anderem Spektralbereich in Steuersoftware und Auswertung

4. Prüfung des Messkonzeptes

    - Herstellung von unterschiedlich lithinierten Proben als Modell für unterschiedliche Ladezustände
    - Erprobung des LED-Reflektometers mit den Probepräparaten
    - Untersuchung des Aufbaus auf Reproduzierbarkeit (Stabilisierung der Lichtleiter im Reflektionsbereich, mechanische Fixierung der Biegeradien, definierter Bereich mit entferntem Cladding o.ä.)
    - Erfasssung einer möglichst aussagefähigen Messreihe mit verschiedenen Proben
    - Vergleich von Messungen mit dem Spektrometer und mit dem eigenen LED-Reflektometer

5. Einordnung, Bewertung und Ausblick

    - Zusammenfassung der Ergebnisse
    - Bewertung der gewählten Konzepte und Lösungsvarianten
    - Offene Punkte und Alternativen als Basis für weiterführende Arbeiten
    - Beitrag und Einordnung in das Forschungsvorhaben

Dokumentation

Die Fachliteratur und die kommerziellen Unterlagen bzw. Datenblätter sind zielgerichtet zu recherchieren. Dabei sind insbesondere wichtige Grundlagen der Lichtleiteroptik näher zu betrachten. Die gesetzten Rahmenbedingungen, gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die Messergebnisse sind in exemplarischem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch einordnend zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

B. Schaltpläne

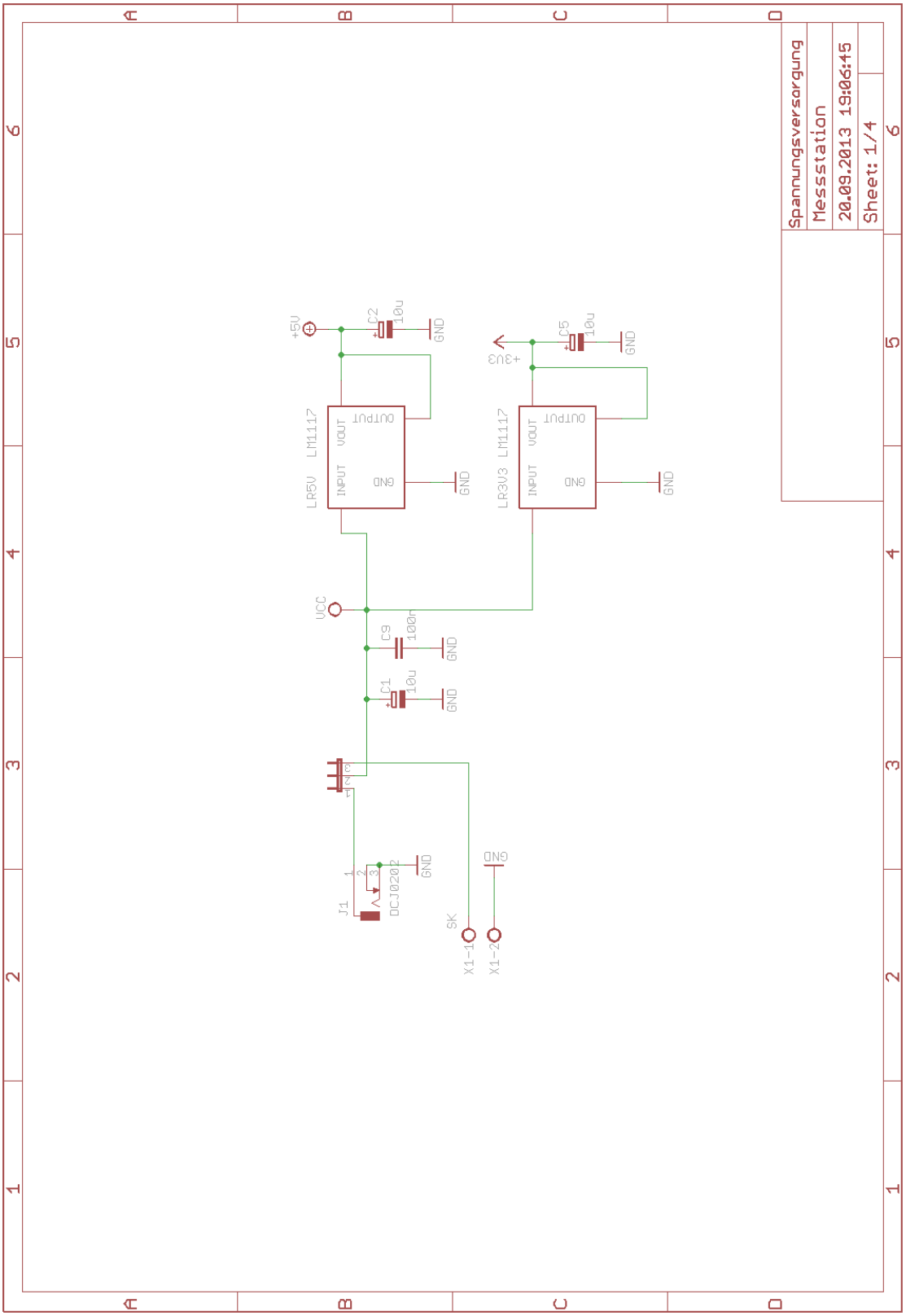

Abbildung B.1.: Schaltplan - Spannungsversorgung

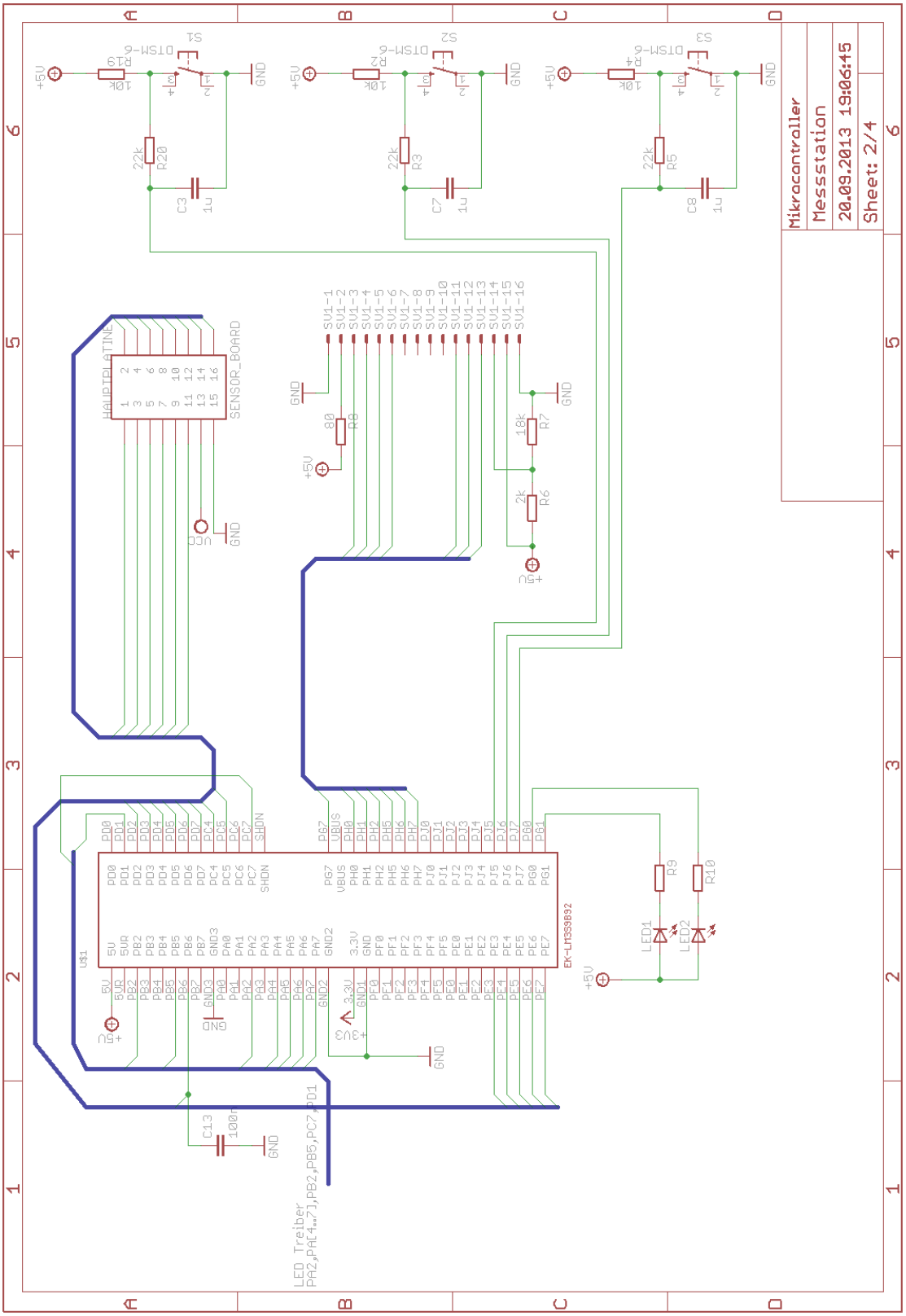

Abbildung B.2.: Schaltplan - Controller und Display

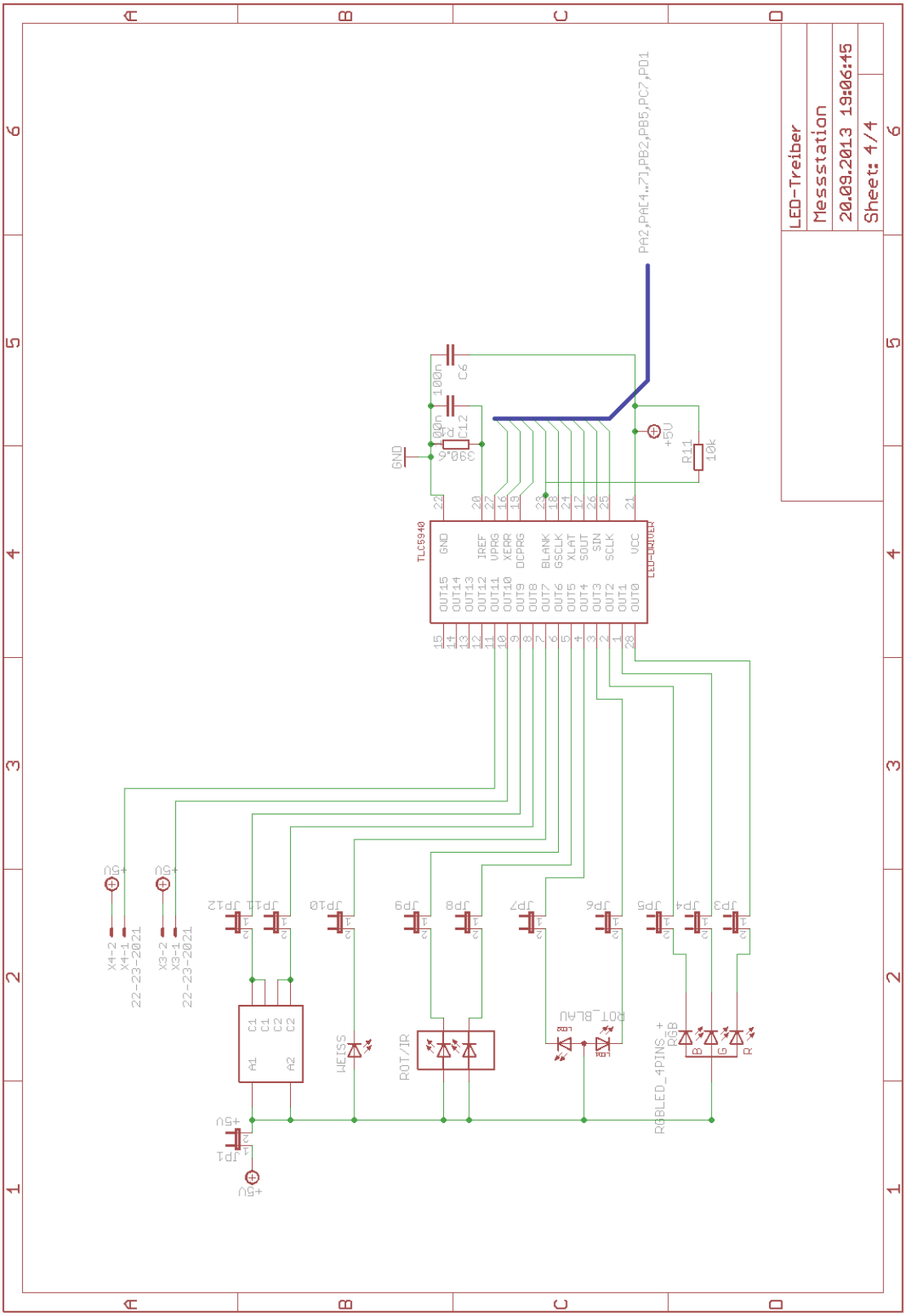

Abbildung B.3.: Schaltplan - LED-Treiber und LEDs

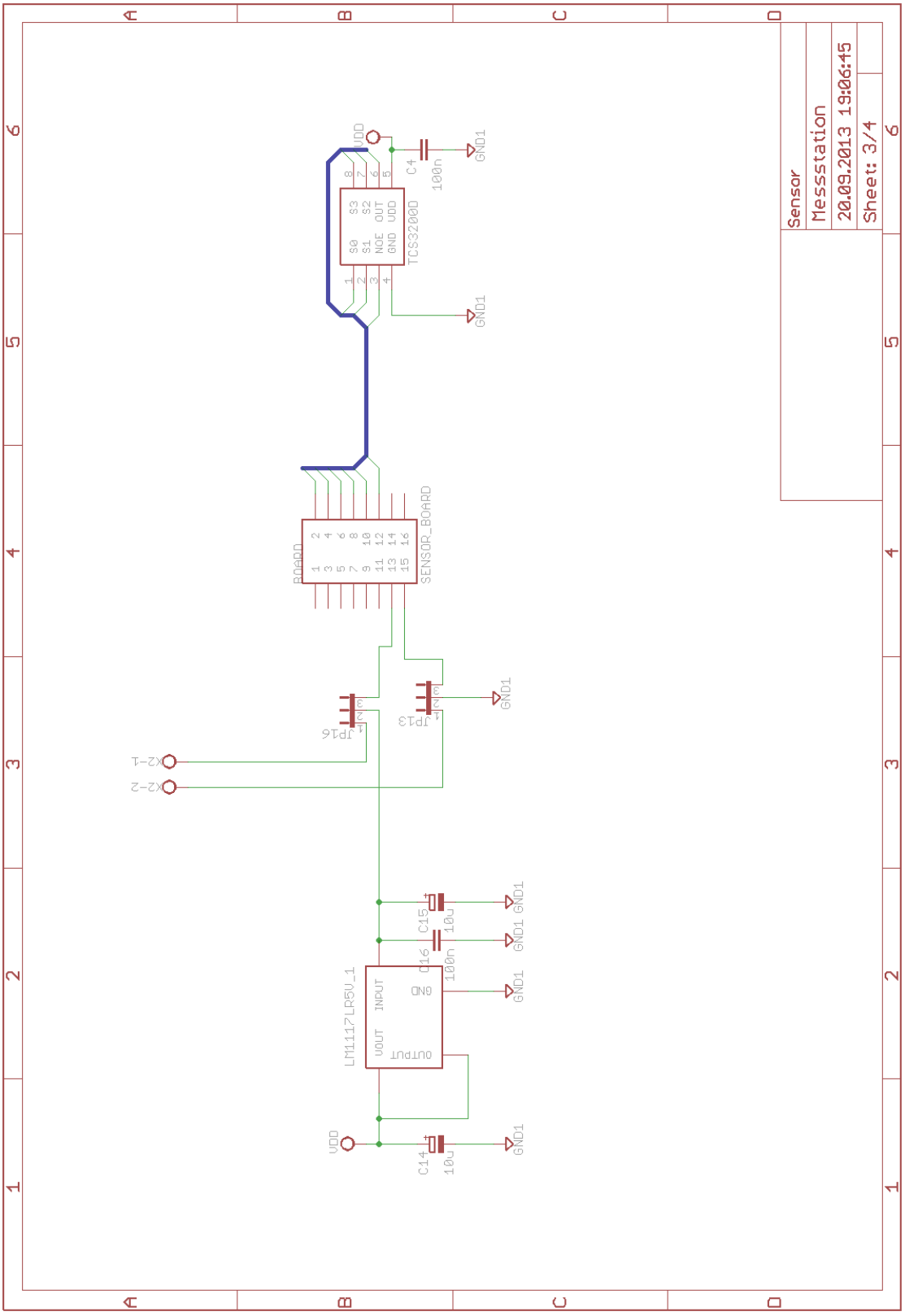

Abbildung B.4.: Schaltplan - Lichtsensor

C. Platinenlayouts

Hinweise:

- Die Anschlüsse für die PLCC4-SMD-LEDs sind für Bauelemente mit einer gemeinsamen Kathode ausgelegt. In der Regel besitzen PLCC4-LEDs jedoch eine gemeinsame Anode [37]. Bei zukünftigen Umsetzungen würde es sich anbieten, dies im Layout zu ändern.
- Der Abstand der Pads für die drei Taster sind minimal zu gering
- Damit das Flachbandkabel vom LC-Display zum Anschluss am Platinenlayout nicht gedreht werden muss, wäre das Spiegeln dieses Anschlusses sinnvoll.

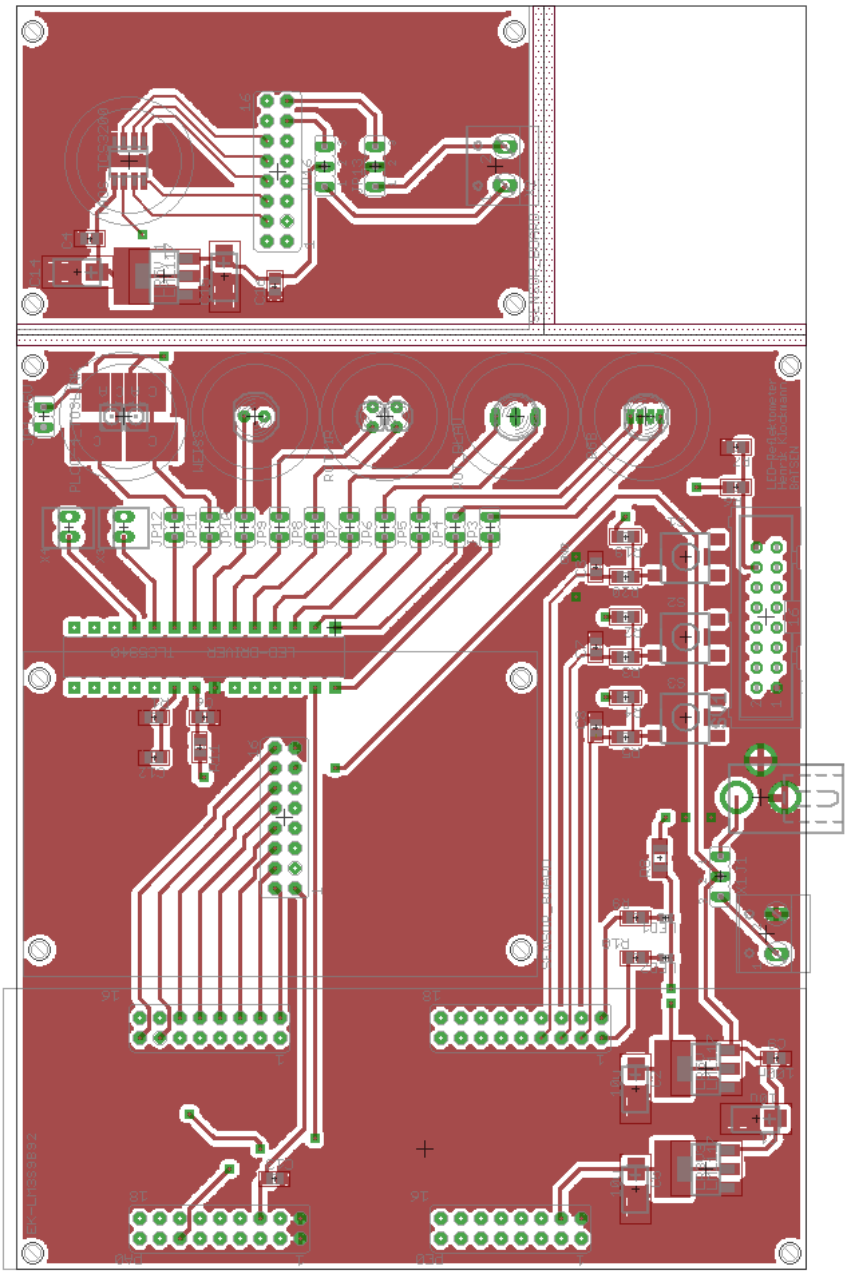

Abbildung C.1.: Platinenlayout Oberseite

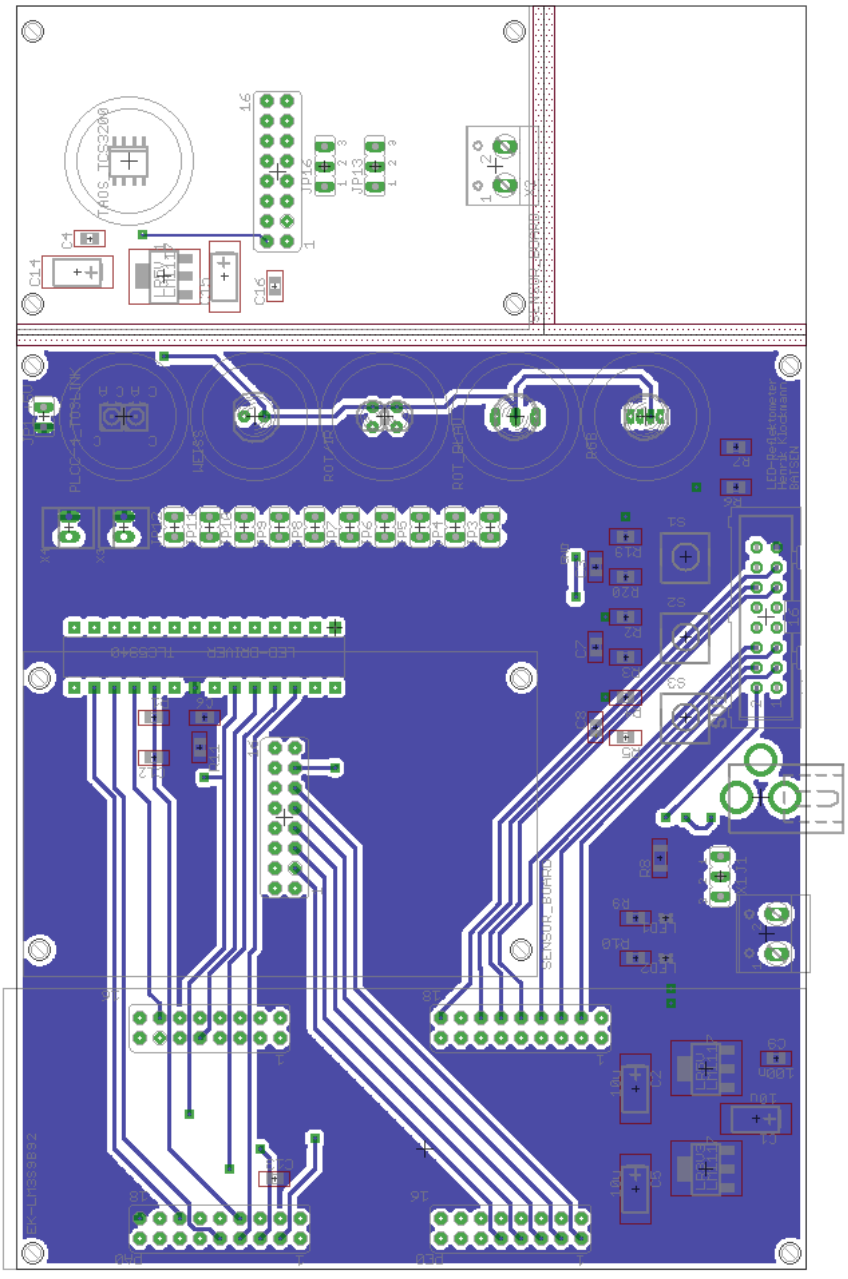

Abbildung C.2.: Platinenlayout Unterseite

D. Pinbelegungen

Pin μC	Peripherie	Pin Peripherie	Erläuterung
PA2	LED-Treiber	SCLK	Takt für serielle Daten
PA4	LED-Treiber	SOUT	Ausgang für serielle Daten
PA5	LED-Treiber	SIN	Eingang für serielle Daten
PA6	LED-Treiber	BLANK	Deaktivierung aller Ausgänge
PA7	LED-Treiber	VPRG	0 = GS mode, 1 = DC mode
PB2	LED-Treiber	XERR	-
PB5	LED-Treiber	DCPRG	0 = EEPROM, 1 = Register für DC-Werte
PB6	Sensor-Platine	-	Externe Referenzspannung
PD1	LED-Treiber	GSCLK	Takt für die PWM
PD2	Sensor-Platine	NOE	Enable-Pin (active low)
PD3	Sensor-Platine	S1	Skalierung der Ausgangsfrequenz
PD4	Sensor-Platine	S0	Skalierung der Ausgangsfrequenz
PD5	Sensor-Platine	S3	Wahl der Photodiode
PD6	Sensor-Platine	S2	Wahl der Photodiode
PD7	Sensor-Platine	OUT	Ausgangsfrequenz
PC4	Sensor-Platine	-	-
PC5	Sensor-Platine	-	-
PC7	LED-Treiber	XLAT	Übernahme der eingetakteten Daten
PE3	Sensor-Platine	-	ADC AIN8
PE4	Sensor-Platine	-	ADC AIN3
PE5	Sensor-Platine	-	ADC AIN2
PE6	Sensor-Platine	-	ADC AIN1
PE7	Sensor-Platine	-	ADC AIN0
PG0	Benutzer-LED	-	-
PG1	Benutzer-LED	-	-
PG7	LC-Display	RS	-
PH0	LC-Display	RW	-
PH1	LC-Display	E	-
PH2	LC-Display	DB4	-
PH5	LC-Display	DB5	-
PH6	LC-Display	DB6	-
PH7	LC-Display	DB7	-
PJ5	Taster	-	-
PJ6	Taster	-	Stoppt laufende Messungen
PJ7	Taster	-	Startet Messungen

Tabelle D.1.: Pinbelegungen des Mikrocontrollers mit Beschreibung der beschalteten Bauelemente

E. Quellcodes

E.1. Mikrocontroller

main.c

```
1  /*
2   * main.c
3   *
4   *  Created on: 10.07.2013
5   *  Author: Henrik
6   *
7   *  main function for the LED reflectometer
8   */
9
10 #include "inc/lm3s9b92.h"
11 #include "inc/hw_memmap.h"
12 #include "inc/hw_types.h"
13 #include "inc/hw_ints.h"
14 #include "driverlib/gpio.h"
15 #include "driverlib/sysctl.h"
16 #include "inc/hw_ssi.h"
17 #include "driverlib/ssi.h"
18
19 #include "ledDriver.h"
20 #include "measurements.h"
21 #include "display.h"
22 #include "communication.h"
23 #include "pushButtons.h"
24
25
26 extern void Timer_ISR();
27 extern void Sensor_OUT_ISR();
28 extern tBoolean bLEDon;
29
30
31
32 int main(void)
33 {
34
35
36     // Set clock to 80MHz, use external 16 MHz oscillator, use PLL.
37     SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
38     //SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
39
40
41
42     ledDriverInit();
43     initDisplay();
44     initTCS3200();
45     initInternalTempSensor();
46     initUart();
47     initInterrupts();
48     pBInit();
49
50
51     while(1)
52     {
53         if(bLEDon) // bLEDon is declared with keyword extern in ledDriver.h
54         {
55             TLC5940_SetGS_And_GS_PWM(); // let the led driver do its thing
56                                         // even if pwm isn't actually used this is still needed
57
58         }
59     }
```

```
60
61  }
```

communication.h

```
1   /*
2    * communication.h
3    *
4    *  Created on: 24.07.2013
5    *      Author: Henrik
6    */
7
8   #ifndef COMMUNICATION_H_
9   #define COMMUNICATION_H_
10
11  void initUart();
12  void UARTIntHandler(void);
13  void UARTwriteMod(const char *pcBuf, unsigned long ulLen);
14  int ProcessStop(int argc, char *argv[]);
15  int ProcessStop(int argc, char *argv[]);
16
17
18  #endif /* COMMUNICATION_H_ */
```

communication.c

```
1   /*
2    * coummunication.c
3    *
4    *  Created on: 24.07.2013
5    *      Author: Henrik
6    *
7    *  This file contains functions for a serial communication between the microcontroller and a PC.
8    *
9    */
10
11  #include "inc/lm3s9b92.h"
12  #include "inc/hw_memmap.h"
13  #include "inc/hw_types.h"
14  #include "inc/hw_ints.h"
15  #include "driverlib/gpio.h"
16  #include "driverlib/sysctl.h"
17  #include "driverlib/uart.h"
18  #include "utils/uartstdio.h"
19  #include "driverlib/interrupt.h"
20  #include "string.h"
21  #include "stdio.h"
22
23
24  #include "communication.h"
25  #include "measurements.h"
26  #include "ledDriver.h"
27  #include "display.h"
28
29  #include "cmdline.h"
30
31
32  static tBoolean ongoingMeasurement = false;
33  static char cCMDBuffer[128];
34  static volatile unsigned long ulCMDIndex = 0;
35
36
37  // Initialisation for UART
38  //*****************************************************************/
39  void initUart(void)
40  {   // Enable Peripheral for UART 0
41      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
42      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
43
44      // Configure Pins
45      GPIOPinConfigure(GPIO_PA0_U0RX);
46      GPIOPinConfigure(GPIO_PA1_U0TX);
47      GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
48
49      // Configure protocol
50      UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
51                          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
52                           UART_CONFIG_PAR_NONE));
53
54      // Register interrupt handler for UART
```

```
55      UARTIntRegister(UART0_BASE, UARTIntHandler);
56      IntPrioritySet(INT_UART0, 0x00);
57      IntEnable(INT_UART0);
58      UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
59      UARTprintf("UART init completed \n"); // Send something after initialisation
60  }
61
62  // Modified UARTwrite from Stellarisware
63  /*****************************************************************/
64  void UARTwriteMod(const char *pcBuf, unsigned long ulLen)
65  {
66      unsigned int uldx;
67      for(uldx = 0; uldx < ulLen; uldx++)
68      {
69          //
70          // If the character to the UART is \n, then add a \r before it so that
71          // \n is translated to \n\r in the output.
72          //
73          if(pcBuf[uldx] == '\n')
74          {
75              UARTCharPut(UART0_BASE, '\r');
76          }
77
78          //
79          // Send the character to the UART output.
80          //
81          UARTCharPut(UART0_BASE, pcBuf[uldx]);
82      }
83
84  }
85
86
87  // Interrupt handler for UART, modified Stellaris example
88  /*****************************************************************/
89  void UARTIntHandler(void)
90  {
91      unsigned long ulInts;
92      char cChar;
93      long lChar;
94      static tBoolean bLastWasCR = false;
95      //
96      // Get and clear the current interrupt source(s)
97      //
98      ulInts = UARTIntStatus(UART0_BASE, true);
99      UARTIntClear(UART0_BASE, ulInts);
100
101     if(ulInts & UART_INT_TX)
102     {
103
104     }
105     if(ulInts & (UART_INT_RX | UART_INT_RT))
106     {
107         //
108         // Get all the available characters from the UART.
109         //
110         while( UARTCharsAvail(UART0_BASE) )
111         {
112             //
113             // Read a character
114
115             lChar = UARTCharGetNonBlocking(UART0_BASE);
116             cChar = (unsigned char)(lChar & 0xFF);
117
118             if(cChar == 0x03) // End of Text (STRG + C)
119                 ProcessStop(0,0); // stop measurement and configure cmdline
120
121             else if(cChar == 0x7F) // if received character is a backspace
122             {
123                 //
124                 // If there are any characters already in the buffer, then
125                 // delete the last.
126                 //
127                 if(ulCMDIndex)
128                 {
129                     //
130                     // Rub out the previous character on the users terminal.
131                     //
132                     if (!ongoingMeasurement)// don't send during measurement
133                         UARTwriteMod(&cChar, 1);// echo
134
135                     cCMDBuffer[--ulCMDIndex] = 0;
136                 }
137
138             }
139
```

```
140             //
141             // If this character is LF and last was CR, then just gobble up
142             // the character since we already echoed the previous CR and we
143             // don't want to store 2 characters in the buffer if we don't
144             // need to.
145             //
146             else if((cChar == '\n') && bLastWasCR)
147             {
148                 bLastWasCR = false;
149             }
150 
151             //
152             // See if a newline or escape character was received.
153             //
154             else if((cChar == '\r') || (cChar == '\n') || (cChar == 0x1b))
155             {
156                 //
157                 // If the character is a CR, then it may be followed by an
158                 // LF which should be paired with the CR.  So remember that
159                 // a CR was received.
160                 //
161                 if(cChar == '\r')
162                 {
163                     bLastWasCR = 1;
164                 }
165 
166                 //
167                 // Regardless of the line termination character received,
168                 // put a 0 in the receive buffer as a marker telling
169                 // CmdLineProcess() where the line ends.  We also send an
170                 // additional LF to ensure that the local terminal echo
171                 // receives both CR and LF.
172                 //
173                 if (!ongoingMeasurement) // don't send during measurement
174                     UARTwriteMod("\n", 1); // echo
175                 cCMDBuffer[ulCMDIndex] = 0; // instead of writing new line into buffer, write a 0
176                 ulCMDIndex = 0;
177                 CmdLineProcess(cCMDBuffer);
178             }
179             else
180             {
181                 if (!ongoingMeasurement)
182                     UARTwriteMod(&cChar, 1);// echo
183                 cCMDBuffer[ulCMDIndex++] = cChar;
184             }
185 
186 
187         }
188     }
189 }
190 
191 
192 // Function for the "setLED" command. This function sets a DC Value of the LED-Driver
193 /*****************************************************************/
194 int ProcessLED(int argc, char *argv[])
195 {
196     if(!ongoingMeasurement){
197         if (argc == 3) // Expected number of modifier
198         {
199             setDCValue(atoi(argv[1]), atoi(argv[2])); // set DC-Value
200             UARTprintf("#SUCCESS\n"); //
201         }
202         else // wrong number of modifier
203             UARTprintf("#ERROR_Wrong_number_of_modifier\n");
204     }
205     return 1;
206 }
207 
208 // Function for the "setCycle" command. This function sets the total measurement time and the LED on/off cycle
209 /*****************************************************************/
210 int ProcessCycle(int argc, char *argv[])
211 {
212     if(!ongoingMeasurement){
213         if(argc == 4) // Expected number of modifier
214         {
215             setCycle(atoi(argv[1]), atoi(argv[2]), atoi(argv[3])); // set total time and on/off cycle
216             UARTprintf("#SUCCESS\n"); //
217         }
218         else // wrong number of modifier
219             UARTprintf("#ERROR_Wrong_number_of_modifier\n");
220     }
221     return 1;
222 }
223 
224 
```

```
225 // Function for the "setScaling" command. This function configures the scaling of the light sensor
226 /*********************************************************/
227 int ProcessScaling(int argc, char *argv[])
228 {
229     if(!ongoingMeasurement){
230         if(argc == 3) // Expected number of modifier
231         {
232             setTCS3200Scaling(atoi(argv[1]), atoi(argv[2]));
233             UARTprintf("#SUCCESS\n"); //
234         }
235         else // wrong number of modifier
236             UARTprintf("#ERROR Wrong number of modifier\n");
237     }
238     return 1;
239 }
240
241
242 // Function for the "setFilter" command. This function configures the filter of the light sensor
243 /*********************************************************/
244 int ProcessFilter(int argc, char *argv[])
245 {
246     if(!ongoingMeasurement){
247         if(argc == 3)
248         {
249             setTCS3200Filter(atoi(argv[1]), atoi(argv[2]));
250             UARTprintf("#SUCCESS\n"); //
251         }
252         else
253             UARTprintf("#ERROR Wrong number of modifier\n");
254     }
255     return 1;
256 }
257
258
259 // Function for the "help" command. This function returns a command list (modified stellaris example)
260 /*********************************************************/
261 int ProcessHelp(int argc, char *argv[])
262 {
263     if(!ongoingMeasurement){
264         tCmdLineEntry *pEntry;
265         //UARTCharPutNonBlocking(UART0_BASE, 0x14);
266         UARTprintf("\033[0;0H"); // set cursor to 0,0
267         UARTprintf("\f"); // new page
268         UARTprintf("***** LED-Reflektometer Hilfe *****\n");
269         UARTprintf("-----------------------------------------\n\n");
270         UARTprintf("Funktion\tBeschreibung\n");
271         UARTprintf("-----------------------------------------\n");
272         //
273         // Point at the beginning of the command table.
274         //
275         pEntry = g_sCmdTable;
276
277         //
278         // Enter a loop to read each entry from the command table.  The
279         // end of the table has been reached when the command name is NULL.
280         //
281         while(pEntry->pcCmd)
282         {
283             //
284             // Print the command name and the brief description.
285             //
286
287             UARTprintf("%s%s\n", pEntry->pcCmd, pEntry->pcHelp);
288
289             //
290             // Advance to the next entry in the table.
291             //
292             pEntry++;
293         }
294     }
295     return 1;
296 }
297
298
299 // Function for the "start" command. This function starts a measurement with all configured values(LEDs, time, current etc.)
300 /*********************************************************/
301 int ProcessStart(int argc, char *argv[])
302 {
303     if(!ongoingMeasurement){
304         startMeasurement();
305         ongoingMeasurement = true; // set flag. this disables the echo functionality of the UART-ISR
306         UARTprintf("#START\n");
307     }
308     return 1;
309 }
```

```
310
311 // Function for the "stop" command. This function stops a measurement
312 /*****************************************************************/
313 int ProcessStop(int argc, char *argv[])
314 {
315     if(ongoingMeasurement){
316         stopMeasurement();
317         ongoingMeasurement = false;// unset flag. this enables the echo functionality of the UART-ISR
318         UARTprintf("#STOP\n");
319     }
320     else
321         UARTprintf("#WARNING: already stopped\n");
322     return 1;
323 }
324
325 int ProcessStatus(int argc, char *argv[])
326 {
327     if(!ongoingMeasurement){
328         UARTprintf("#STATUS,LED-Reflektometer,%d\n", SysCtlClockGet());
329     }
330 }
331
332 // Command-table (modified stellaris example)
333 // This table is used by CmdLineProcess() in cmdline.c to check an entered command
334 // and to call a corresponding function.
335 /*****************************************************************/
336 tCmdLineEntry g_sCmdTable[] =
337 {
338         { "start",   ProcessStart,    "\t\tStarte Messungen" },
339         { "stop",    ProcessStop,     "\t\tStoppe Messungen" },
340         { "setLED",      ProcessLED,      "\t\tsetLED LEDnr(0 - N) Power(0-100)" },
341         { "setCycle",    ProcessCycle,    "\tsetCycle LEDON(ms) LEDOFF(ms) TOTAL(ms)" },
342         { "setScaling", ProcessScaling,      "\tsetScaling s0 s1 (see TCS3200 documentation)" },
343         { "setFilter",   ProcessFilter,   "\tsetFilter s2 s3 (see TCS3200 documentation)" },
344         { "status",      ProcessStatus,   "\t\tLED-Reflektometer Status." },
345         { "help",    ProcessHelp,     "\t\tApplication help." },
346         { 0,         0,               "-- terminating entry --" }
347 };
```

ledDriver.h

```
1  /*
2   * ledDriver.h
3   *
4   *  Created on: 16.07.2013
5   *      Author: Henrik
6   */
7
8  #ifndef LEDDRIVER_H_
9  #define LEDDRIVER_H_
10
11 #define SCLK_PIN GPIO_PIN_2
12 #define SCLK_PORT GPIO_PORTA_BASE
13
14 #define SOUT_PIN GPIO_PIN_4
15 #define SOUT_PORT GPIO_PORTA_BASE
16
17 #define SIN_PIN GPIO_PIN_5
18 #define SIN_PORT GPIO_PORTA_BASE
19
20 #define BLANK_PIN GPIO_PIN_6
21 #define BLANK_PORT GPIO_PORTA_BASE
22
23 #define VPRG_PIN GPIO_PIN_7
24 #define VPRG_PORT GPIO_PORTA_BASE
25
26 #define XERR_PIN GPIO_PIN_2
27 #define XERR_PORT GPIO_PORTB_BASE
28
29 #define GSCLK_PIN GPIO_PIN_1
30 #define GSCLK_PORT GPIO_PORTD_BASE
31
32 #define DCPRG_PIN GPIO_PIN_5
33 #define DCPRG_PORT GPIO_PORTB_BASE
34
35 #define XLAT_PIN GPIO_PIN_7
36 #define XLAT_PORT GPIO_PORTC_BASE
37
38 #define NUM_LED_CH 16
39 #define NUM_SSI_DATA 3
40
41 void ledDriverInit(void);
42 void TLC5940_ClockInDC(void);
```

```
43  void TLC5940_SetGS_And_GS_PWM(void);
44  void initSSI(void);
45  void TLC5940_SendDC(void);
46  void setDCValue(char ledNR, char value);
47  void turnLEDOn(void);
48  void turnLEDOff(void);
49  void setNextLED(void);
50  char anyLedSet(void);
51
52  #endif /* LEDDRIVER_H_ */
```

ledDriver.c

```
1   /*
2    * ledDriver.c
3    *
4    *  Created on: 16.07.2013
5    *      Author: Henrik
6    *
7    *  This file contains functions regarding the led driver TLC5940 from TI
8    */
9
10  #include "ledDriver.h"
11  #include "inc/lm3s9b92.h"
12  #include "inc/hw_memmap.h"
13  #include "inc/hw_types.h"
14  #include "inc/hw_ints.h"
15  #include "driverlib/gpio.h"
16  #include "driverlib/sysctl.h"
17  #include "inc/hw_ssi.h"
18  #include "driverlib/ssi.h"
19
20
21  int gsData[192] = { // GS Values for PWM
22          // MSB LSB
23          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,// Channel 15
24          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 14
25          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 13
26          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 12
27          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 11
28          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 11
29          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 9
30          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 8
31          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 7
32          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 6
33          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 5
34          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 4
35          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 3
36          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 2
37          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 1
38          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, // Channel 0
39  };
40
41  char dcValue[16] = { // DC Values for SPI
42          0, // Channel 0
43          0, // Channel 1
44          0, // Channel 2
45          0, // Channel 3
46          0, // Channel 4
47          0, // Channel 5
48          0, // Channel 6
49          0, // Channel 7
50          0, // Channel 8
51          0, // Channel 9
52          0, // Channel 10
53          0, // Channel 11
54          0, // Channel 12
55          0, // Channel 13
56          0, // Channel 14
57          0, // Channel 15
58  };
59  char storedDcValue[16] = { // stored values - 0 - 63
60          15, // Channel 0
61          0, // Channel 1
62          0, // Channel 2
63          0, // Channel 3
64          0, // Channel 4
65          0, // Channel 5
66          0, // Channel 6
67          0, // Channel 7
68          0, // Channel 8
69          0, // Channel 9
70          0, // Channel 10
```

```
71          0, // Channel 11
72          0, // Channel 12
73          0, // Channel 13
74          0, // Channel 14
75          0, // Channel 15
76  };
77
78  enum leds {LED11, LED12, LED13, // LED numbers
79             LED21, LED22,
80             LED31, LED32,
81             LED41,
82             LED51, LED52,
83             LED61, LED62};
84
85  extern char cNextLED = LED11 ;
86
87  extern tBoolean bLEDon = false; // bLEDon is declared with keyword extern in ledDriver.h
88
89  // Initialisation for communictaion with the LED-Driver
90  /*************************************************************/
91  void ledDriverInit(void){
92
93      // Enable Peripheral
94      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
95      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
96      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
97      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
98
99      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,BLANK_PIN | VPRG_PIN); // Output
100     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,0xFF); // Output
101
102     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,XLAT_PIN); // Set whole Port as Output
103     GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,GSCLK_PIN | GPIO_PIN_0);// Set whole Port as Output
104
105     initSSI(); // Initialise SSI
106
107     GPIOPinWrite(GSCLK_PORT, GSCLK_PIN , 0x00);
108     GPIOPinWrite(SCLK_PORT, SCLK_PIN , 0x00);
109     GPIOPinWrite(DCPRG_PORT, DCPRG_PIN , 0x00); // DC Value -> EEPROM
110     GPIOPinWrite(VPRG_PORT, VPRG_PIN , VPRG_PIN); // Dot Correction Data Input Mode
111     GPIOPinWrite(XLAT_PORT, XLAT_PIN , 0x00);
112     GPIOPinWrite(BLANK_PORT, BLANK_PIN, BLANK_PIN);
113 }
114
115 // Function for configuring DC values
116 /*************************************************************/
117 void TLC5940_SendDC(void) {
118     int i = 0;
119     GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2);
120     GPIOPinWrite(DCPRG_PORT, DCPRG_PIN , DCPRG_PIN); // Dot Correction Data Input Mode
121     GPIOPinWrite(VPRG_PORT, VPRG_PIN , VPRG_PIN); // Dot Correction Data Input Mode
122     //dcValue[8] = 63;
123     // Send whole DC data. Blocking
124         for(i = NUM_LED_CH-1; i >= 0; i--) // for(i = 0; i < NUM_LED_CH; i++)
125             SSIDataPut(SSI0_BASE, dcValue[i]);
126         // Wait for SSI for being ready
127         while(SSIBusy(SSI0_BASE))
128         {
129         }
130
131         GPIOPinWrite(XLAT_PORT, XLAT_PIN , XLAT_PIN); // Dot Correction Data Input Mode
132         GPIOPinWrite(XLAT_PORT, XLAT_PIN , 0x00); // Dot Correction Data Input Mode
133
134 }
135
136 // PWM-Function - (inspired by TLC5940 Programing Flow Chart v0.1 from TI)
137 /*************************************************************/
138 void TLC5940_SetGS_And_GS_PWM(void) {
139
140     // Set SCLK and SIN pins as GPIO output, since they could have been used by SSI
141     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,SCLK_PIN | SIN_PIN);
142
143     int firstCycleFlag = 0;
144     int GSCLK_Counter = 0;
145     int Data_Counter = 0;
146
147     if (GPIOPinRead(VPRG_PORT, VPRG_PIN)) {
148         GPIOPinWrite(VPRG_PORT, VPRG_PIN , 0x00); // Set to Greyscale-Input-Mode
149         firstCycleFlag = 1;
150     }
151     //GPIOPinWrite(VPRG_PORT, VPRG_PIN , 0x00);
152
153
154
155     GPIOPinWrite(BLANK_PORT, BLANK_PIN, 0x00); // enable output
```

```
156
157         while(GSCLK_Counter < 4096) {
158             if (!(Data_Counter > 192 - 1)) {
159                 if (gsData[Data_Counter])
160                     GPIOPinWrite(SIN_PORT, SIN_PIN , SIN_PIN); // set bit on pin
161                 else
162                     GPIOPinWrite(SIN_PORT, SIN_PIN , 0x00);
163
164                 //pulse SCLK
165                 GPIOPinWrite(SCLK_PORT, SCLK_PIN , SCLK_PIN);
166                 GPIOPinWrite(SCLK_PORT, SCLK_PIN , 0x00);
167                 Data_Counter++;
168             }
169
170             //pulse GSCLK
171             GPIOPinWrite(GSCLK_PORT, GSCLK_PIN , GSCLK_PIN);
172
173             // easy solution for stopping the flickering due to the PWM mode, which turns off the led's
174             // for a short time. Function will wait here, till led is to be turned off again by led-timer.
175             while(bLEDon);
176
177             GPIOPinWrite(GSCLK_PORT, GSCLK_PIN , 0x00);
178             GSCLK_Counter++;
179         }
180         GPIOPinWrite(BLANK_PORT, BLANK_PIN, BLANK_PIN); // disable output
181
182         //pulse XLAT
183         GPIOPinWrite(XLAT_PORT, XLAT_PIN , XLAT_PIN);
184         GPIOPinWrite(XLAT_PORT, XLAT_PIN , 0x00);
185
186         if (firstCycleFlag) {
187
188             //pulse SCLK
189             GPIOPinWrite(SCLK_PORT, SCLK_PIN , SCLK_PIN);
190             GPIOPinWrite(SCLK_PORT, SCLK_PIN , 0x00);
191             firstCycleFlag = 0;
192         }
193
194     }
195
196     // Function for configuration of a DC value in an array
197     /*************************************************************/
198     void setDCValue(char ledNR, char value)
199     {
200         if ( value > 63 )
201             value = 63;
202
203         if(ledNR < NUM_LED_CH)
204             storedDcValue[ledNR] = value;
205     }
206
207     // Function for turning on the LED
208     // this function sets a flag which is evaluated in the main loop
209     /*************************************************************/
210     void turnLEDOn(void)
211     {
212
213         setNextLED(); // before turning the driver on, next led will be configured
214         bLEDon = true;// flag -> evaluated in the main loop
215     }
216
217     // Function for turning off the LED
218     /*************************************************************/
219     void turnLEDOff(void)
220     {
221         bLEDon = false;
222     }
223
224     void initSSI(void)
225     {
226         unsigned long ulDataTx[NUM_SSI_DATA];
227         unsigned long ulDataRx[NUM_SSI_DATA];
228         unsigned long ulindex;
229
230         // Enable SSI0 peripheral
231         SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
232         // Enable GPIOA peripheral
233         SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
234
235         // Configure pins for SSI0, probably not necessary, because SSI doesn't support muxing on my Hardware
236         GPIOPinConfigure(GPIO_PA2_SSI0CLK);
237         GPIOPinConfigure(GPIO_PA3_SSI0FSS);
238         GPIOPinConfigure(GPIO_PA4_SSI0RX);
239         GPIOPinConfigure(GPIO_PA5_SSI0TX);
240
```

```
241         // Give Control of Pins to the SSI Hardware
242         GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2);
243
244         // Configure SSI for being Master, SPI Mode, 1MHz, 6Bit
245         SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 6);
246
247         // Enable SSI0 module
248         SSIEnable(SSI0_BASE);
249
250         // Clear Fifo
251         while(SSIDataGetNonBlocking(SSI0_BASE, &ulDataRx[0]))
252         {
253         }
254     }
255
256     // This function looks for the next led , which should be turned on,
257     // then writes the configured value in an array and calls the TLC5940_SendDC function
258     // which sends the dc value to the led driver
259     /*************************************************************/
260     void setNextLED(void)
261     {
262         int i = 0;
263         tBoolean nextLEDfound = false;
264         // first set all values to zero
265         for(i = 0;i < NUM_LED_CH; i++)
266             dcValue[i] = 0;
267
268         //now find the next activated led starting from the last one
269         for(i = cNextLED+1;i < NUM_LED_CH; i++)
270         {
271             if(storedDcValue[i] > 0 && !nextLEDfound)
272             {
273                 cNextLED = i;
274                 nextLEDfound = true;
275                 dcValue[i] = storedDcValue[i];
276             }
277         }
278
279         // wrap around if no next led was found yet
280         if(!nextLEDfound)
281         {
282             for(i = 0;i < cNextLED+1; i++)
283             {
284                 if(storedDcValue[i] > 0 && !nextLEDfound)
285                 {
286                     cNextLED = i;
287                     nextLEDfound = true;
288                     dcValue[i] = storedDcValue[i];
289                 }
290                 else
291                     dcValue[i] = 0;
292
293             }
294         }
295
296         TLC5940_SendDC(); // Configure LED's on LED-Driver
297     }
298
299     // checks if a LED was set by the user
300     /*************************************************************/
301     char anyLedSet(void)
302     {
303         char i = 0;
304         char ledSet = 0;
305         tBoolean nextLEDfound = false;
306         for(i =0;i < NUM_LED_CH; i++)
307         {
308             if(storedDcValue[i] > 0)
309             {
310                 ledSet = 1;
311             }
312
313         }
314         return ledSet;
315     }
```

measurements.h

```
1   /*
2    * measurements.h
3    *
4    *  Created on: 17.07.2013
5    *      Author: Henrik
```

```
6    */
7
8   #ifndef TAOS_TCS3200_H_
9   #define TAOS_TCS3200_H_
10
11  #define Sensor_S0 GPIO_PIN_4
12  #define Sensor_S1 GPIO_PIN_3
13  #define Sensor_S2 GPIO_PIN_6
14  #define Sensor_S3 GPIO_PIN_5
15  #define Sensor_NOE GPIO_PIN_2
16  #define Sensor_OUT GPIO_PIN_7
17
18  void initTCS3200(void);
19  void initInternalTempSensor(void);
20  void readInternalTemperature(void);
21  void LEDTimer_ISR();
22  void Measurement_Timer_ISR();
23  void Sensor_OUT_ISR();
24  void Measurement_TX_ISR();
25  void initInterrupts();
26  void startMeasurement(void);
27  void stopMeasurement(void);
28  void setCycle(unsigned long ulCycleOn, unsigned long ulCycleOff, unsigned long ulCycleTotal);
29
30  #endif /* TAOS_TCS3200_H_ */
```

measurements.c

```
1   /*
2    * measurements.h
3    *
4    *  Created on: 17.07.2013
5    *      Author: Henrik
6    */
7   #include "measurements.h"
8   #include "inc/lm3s9b92.h"
9   #include "inc/hw_memmap.h"
10  #include "inc/hw_types.h"
11  #include "inc/hw_ints.h"
12  #include "inc/hw_timer.h"
13  #include "driverlib/gpio.h"
14  #include "driverlib/sysctl.h"
15  #include "driverlib/timer.h"
16  #include "driverlib/interrupt.h"
17  #include "driverlib/systick.h"
18  #include "driverlib/adc.h"
19  #include <inttypes.h>
20
21  //uart
22  #include "driverlib/uart.h"
23  #include "string.h"
24  #include "utils/uartstdio.h"
25  #include "stdio.h"
26
27  #include "display.h"
28  #include "communication.h"
29  #include "ledDriver.h"
30
31
32  extern char cNextLED;
33  tBoolean newMeasurement = true;
34
35  unsigned long ulTicks = 0;
36  unsigned long ulLEDTimeOn = 1000; //default value LED on ms
37  unsigned long ulLEDTimeOff = 1000; // default value LED off ms
38  unsigned long ulLEDTimeTotal = 60000; // default value measurement duration
39  unsigned long ulSendValueTime = 200; // default time between two sent measurements in ms (max 1s at 16Mhz)
40  unsigned long ulSpentTime = 0;
41  unsigned long ulSensorRisingEdgeTime = 0; // time between the last 2 rising edges
42  unsigned long long ulMeanTimeBetweenRisingEdges = 0; // for calculating mean time, reset every time the value is sent to pc
43  unsigned long ulTimer0LastValue = 0xFFFF;
44  unsigned long ulTimer0ActValue = 0;
45  unsigned long ulTimer0Overflows = 0; // Overflows since last measured rising edge of sensor
46  unsigned long ulTimerIntStatus;
47  unsigned long ulADC0_Value[1]; // FIFO for sequence 3 -> depth 1
48  unsigned long ulTemp_ValueC;
49  unsigned long txFrequency = 0;
50  unsigned int additions = 1;
51  int tmp;
52
53  enum states {
54      START,
55      LEDON,
```

```
56      LEDOFF,
57      STOP
58  };
59
60  char cState = START;
61
62  // Interrupt initialisations
63  /*********************************************************/
64  void initInterrupts(void)
65  {
66
67      IntMasterDisable();
68      // Timer for measuring Sensor-Frequency Input Compare time (16 Bit, not available in 32 bit)
69
70  /* Disabled, because controller could not handle too fast interrupts from the sensor(over 400.000 kHz)
71   * Its ISR is too heavy due to interrupt organisation.
72   * Could possibly be optimised for better performance, but the easy counting method is working also good
73
74      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // Timer
75      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Sensor, (TIMER 0B, because PD7 is on CCP1)
76
77      GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, Sensor_OUT);
78      GPIOIntTypeSet(GPIO_PORTD_BASE, Sensor_OUT, GPIO_RISING_EDGE);
79      GPIOPinConfigure(GPIO_PD7_CCP1);
80      GPIOPinTypeTimer(GPIO_PORTD_BASE, Sensor_OUT); // Sensor pin for time capture
81      TimerConfigure(TIMER0_BASE, TIMER_CFG_SPLIT_PAIR | TIMER_CFG_B_CAP_TIME);
82      IntPrioritySet(INT_TIMER0B, 0);
83      IntPrioritySet(INT_GPIOD, 0);
84      TimerLoadSet(TIMER0_BASE, TIMER_B, 0xFFFF);  // Start Countdown from 0xFFFF
85      TimerIntRegister(TIMER0_BASE, TIMER_B, Time_Measurement_Timer_ISR);
86  */
87
88      // Sensor(TCS3200) GPIO Interrupt
89      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
90
91      GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, Sensor_OUT); // This init is already done before...
92      GPIOIntTypeSet(GPIO_PORTD_BASE, Sensor_OUT, GPIO_RISING_EDGE);//Interrupt to be triggered on rising edge
93      IntRegister(INT_GPIOD, Sensor_OUT_ISR);
94      IntPrioritySet(INT_GPIOD, 0);
95      GPIOPinIntEnable(GPIO_PORTD_BASE,Sensor_OUT);
96
97      // Timer for LEDON, LEDOFF cycles
98      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
99
100     TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);// 32 Bit Periodic Timer
101     TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()); // Sysclock = 1 sec
102
103     TimerIntRegister(TIMER1_BASE, TIMER_BOTH, LEDTimer_ISR);
104     IntPrioritySet(INT_TIMER1A, 0xE0);
105
106     // Output timer
107     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
108
109     TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);// 32 Bit Periodic Timer
110     TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()/1000 * ulSendValueTime); // Sysclock = 1 sec/10 = 100ms
111
112     TimerIntRegister(TIMER2_BASE, TIMER_BOTH, Measurement_TX_ISR);
113     IntPrioritySet(INT_TIMER2A, 0xE0);
114
115     // Systick timer for sending measurements continuously over uart
116     //SysTickPeriodSet( SysCtlClockGet()/1000 *ulSendValueTime ); // Size of systick is 2^24. At 16 MHz
117     //SysTickIntRegister(Systick_ISR);                          //the max time before a overflow accours
118                                                                 //would be 1s. If more time or a faster clock speed is
119                                                                 //needed, consider using a common 32bit full width timer
120
121     //
122     IntPriorityGroupingSet(7); // priority which can be interrupted -> preemptable
123     //Start Timer and enable interrupts
124     IntMasterEnable(); // is enabled on reset, but for safety reasons
125     stopMeasurement();
126
127     //Internal Temperature sensor
128
129 }
130
131 // Start measurements - configures alls timer and start measurements
132 /*********************************************************/
133 void startMeasurement(void)
134 {
135     ulTicks = 0;
136     IntEnable(INT_GPIOD); // Enable Sensor Interrupt
137     //LED Timer
138     TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/1000 * ulLEDTimeOn); // Sysclock/1000 = 1 msec
139     cState = LEDON;
140     turnLEDOn();// in first state the led is turned on
```

```
141 /* Disabled due to performance issues
142     //Measurement Timer
143     newMeasurement = true;
144     TimerLoadSet(TIMER0_BASE, TIMER_B, 0xFFFF);
145     TimerIntEnable(TIMER0_BASE, TIMER_TIMB_TIMEOUT | TIMER_CAPB_EVENT);
146     TimerEnable(TIMER0_BASE, TIMER_B); // Start Capture timer
147     IntEnable(INT_GPIOD); // Enable Capture Interrupt
148     IntEnable(INT_TIMER0B); // Enable Interrupt for counter
149 */
150
151     // Enable Timer + Timer interrupts
152     IntEnable(INT_TIMER1A);
153     TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);//periodic timeout interrupt
154     TimerEnable(TIMER1_BASE, TIMER_BOTH);
155
156     //Output Timer
157     TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()/1000 * ulSendValueTime); // Sysclock/1000 = 1 msec
158
159     IntEnable(INT_TIMER2A);
160     TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);//periodic timeout interrupt
161     TimerEnable(TIMER2_BASE, TIMER_BOTH);
162
163     // Enable SysTick Interrupt and SysTick
164     //SysTickIntEnable();
165     //SysTickEnable();
166
167 }
168
169 // Start measurements - configures alls timer and start measurements
170 /*************************************************************/
171 void stopMeasurement(void)
172 {
173 /*Disabled due to performance issues
174     // Disable Measurement Timer and Sensor Capture Interrupt
175     IntDisable(INT_GPIOD); // Disable Capture Interrupt
176     IntDisable(INT_TIMER0B); // Disable Interrupt for counter
177     TimerIntDisable(TIMER0_BASE, TIMER_TIMB_TIMEOUT | TIMER_CAPB_EVENT);
178     TimerDisable(TIMER0_BASE, TIMER_B); // Stop Capture timer
179 */
180     // Disable Sensor Edgecounter
181     IntDisable(INT_GPIOD); // Disable Sensor Interrupt
182
183     // Disable SysTick Interrupt and SysTick
184     SysTickIntDisable();
185     SysTickDisable();
186
187     // Disable Timer + Timer interrupts
188     IntDisable(INT_TIMER1A);
189     TimerIntDisable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);//periodic timeout interrupt
190     TimerDisable(TIMER1_BASE, TIMER_BOTH);
191
192     // Disable Output Timer
193     IntDisable(INT_TIMER2A);
194     TimerIntDisable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);//periodic timeout interrupt
195     TimerDisable(TIMER2_BASE, TIMER_BOTH);
196
197
198     // Turn LEDs off and reset states
199     turnLEDOff();
200     ulSpentTime = 0;
201     cState == STOP;
202 }
203
204 void setCycle(unsigned long ulCycleOn, unsigned long ulCycleOff, unsigned long ulCycleTotal)
205 {
206     ulLEDTimeOn = ulCycleOn;
207     ulLEDTimeOff = ulCycleOff;
208     ulLEDTimeTotal = ulCycleTotal;
209 }
210
211 // Simple Interrupt Service Routine for counting interrupts by the sensor output
212 /*************************************************************/
213 void Sensor_OUT_ISR()
214 {
215     GPIOPinIntClear(GPIO_PORTD_BASE, Sensor_OUT);//clear interrupt flag
216     ulTicks++;
217 }
218
219 // Function for tranmission of measured values
220 /*************************************************************/
221 void Measurement_TX_ISR()
222 {
223     TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);//clear interrupt flag
224     readInternalTemperature();
225     ulSpentTime += ulSendValueTime;
```

```
226
227     /*Disabled due to performance issues
228     txFrequency = ((unsigned long long)SysCtlClockGet()*10e6)/((ulMeanTimeBetweenRisingEdges*10e6)/additions);
229     // send measured data with elapsed time since start, led number, sys clock devided by measured period time und uc temp SysCtlClockGet()/additions
230     UARTprintf("#Measurement, %d, %d, %d, %d\n",ulSpentTime, cNextLED,txFrequency,ulTemp_ValueC);//
231     */
232
233     // send measured data with elapsed time since start, led number, ticks*1000/send intervalls in ms
234     UARTprintf("#Measurement,%d,%d,%d,%d\n",ulSpentTime, cNextLED,(ulTicks*1000)/ulSendValueTime,ulTemp_ValueC);
235     ulTicks = 0;
236
237     // Check if total measurement time is exceeded
238     if( ulSpentTime >= ulLEDTimeTotal && ulLEDTimeTotal != 0 ) // if configured time = 0, do not stop
239     {
240         ProcessStop(0,0); // stop measurement(this function is used, for using a central place for stopping measurements
241     }
242
243     newMeasurement = true;
244 }
245
246 // ISR for the LED on/off cycle
247 /************************************************************/
248 void LEDTimer_ISR()
249 {
250     TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);//clear interrupt flag
251     if( cState == LEDON)
252     {
253         TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/1000 * ulLEDTimeOff); // Sysclock/1000 = 1 msec
254         turnLEDOff();
255         cState = LEDOFF;
256         //newMeasurement = true;
257     }
258     else if( cState == LEDOFF )
259     {
260         TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/1000 * ulLEDTimeOn); // Sysclock/1000 = 1 msec
261         turnLEDOn();
262         cState = LEDON;
263         //newMeasurement = true;
264
265     }
266
267 }
268
269 // This ISR is supposed to measure the time between two rising edges of the TCS3200 output signal
270 // Due to its administration workload it is not fast enough to manage higher frequencies. Tests showed
271 // a maximum of about 350 KHz
272 // Therefore this ISR isn't used at the Moment, but it could be if the sensor output generates lower frequencies
273 /************************************************************/
274 void Time_Measurement_Timer_ISR(void)
275 {
276     ulTimerIntStatus = HWREG(TIMER0_BASE + TIMER_O_MIS);// get masked interrupt status of TIMER0_BASE
277     ulTimer0ActValue = HWREG(TIMER0_BASE + TIMER_O_TBR);// get actual capture value
278
279     // Check if Interrupts are thrown by both possible sources at once
280     if(ulTimerIntStatus & TIMER_TIMB_TIMEOUT && ulTimerIntStatus & TIMER_CAPB_EVENT)
281     {
282         // Check if the timer wrapped around before the capture interrupt occurred
283         if(ulTimer0ActValue > ulTimer0LastValue)
284         {
285             ulTimer0Overflows++;
286             TimerIntClear(TIMER0_BASE, TIMER_TIMB_TIMEOUT);// clear interrupt flag for timer timeout
287         }
288     }
289
290     if(ulTimerIntStatus & TIMER_CAPB_EVENT)
291     {
292         // calculate time between two rising edges
293         ulSensorRisingEdgeTime = ulTimer0LastValue - ulTimer0ActValue + 0xFFFF*ulTimer0Overflows;
294         if (newMeasurement == false)
295         {
296             ulMeanTimeBetweenRisingEdges = ulMeanTimeBetweenRisingEdges + ulSensorRisingEdgeTime; // sum up
297             additions++;
298         }
299         else
300         {
301             ulMeanTimeBetweenRisingEdges = ulSensorRisingEdgeTime;
302             newMeasurement = false;
303             additions = 1;
304         }
305         ulTimer0LastValue= ulTimer0ActValue;
306         ulTimer0Overflows = 0;
307         TimerIntClear(TIMER0_BASE, TIMER_CAPB_EVENT);//clear interrupt flag
308
309     }
310     else if(ulTimerIntStatus & TIMER_TIMB_TIMEOUT)  // if timeout and cap interrupt occur at the same time, this part will also be entered
```

```
311                                                 // if the wrap around occurred after the input capture
312     {                                           // overflows are needed for time calculation
313         ulTimer0Overflows++;
314         TimerIntClear(TIMER0_BASE, TIMER_TIMB_TIMEOUT);//clear interrupt flag
315     }
316
317 }
318
319 // TAOS TCS3200 initialisation
320 /*************************************************************/
321 void initTCS3200(void)
322 {
323     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
324     // Set S0,S1,S2,S3,NOE(Not Output Enable) as output
325     GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, Sensor_S0 | Sensor_S1 | Sensor_S2 | Sensor_S3 | Sensor_NOE);
326     GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, Sensor_OUT);// Out of Sensor as input
327
328     //Set Output Frequency Scaling to 100%
329     GPIOPinWrite(GPIO_PORTD_BASE, Sensor_S0 | Sensor_S1, Sensor_S0 | Sensor_S1); //S0 and S1 high
330
331     // Set clear photodiode(no filter)
332     GPIOPinWrite(GPIO_PORTD_BASE, Sensor_S2 | Sensor_S3, Sensor_S2 ); //S2 high and S3 low
333 }
334
335 // TAOS TCS3200 set color filter
336 /*************************************************************/
337 void setTCS3200Filter(char s2, char s3)
338 {
339     char value = 0;
340     if (s2)
341         value = Sensor_S2;
342     if (s3)
343         value = value|Sensor_S3;
344     GPIOPinWrite(GPIO_PORTD_BASE, Sensor_S2 | Sensor_S3, value); // set s2 and s3 as configured
345 }
346
347 // TAOS TCS3200 set scaling
348 /*************************************************************/
349 void setTCS3200Scaling(char s0, char s1)
350 {
351     char value = 0;
352     if (s0)
353         value = Sensor_S0;
354     if (s1)
355         value = value|Sensor_S1;
356     GPIOPinWrite(GPIO_PORTD_BASE, Sensor_S0 | Sensor_S1, value); //S0 and S1 high
357 }
358
359 // Initialisation of temp. sensor ( internal microcortoller temperature)
360 /*************************************************************/
361 void initInternalTempSensor(void)
362 {
363     SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
364     ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
365     ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE |
366                              ADC_CTL_END);
367     ADCSequenceEnable(ADC0_BASE, 3);
368     ADCIntClear(ADC0_BASE, 3);
369 }
370
371 // Read of temp. sensor ( internal microcortoller temperature) (stellaris example)
372 /*************************************************************/
373 void readInternalTemperature(void)
374 {
375     //
376             // Trigger the ADC conversion.
377             //
378             ADCProcessorTrigger(ADC0_BASE, 3);
379
380             //
381             // Wait for conversion to be completed.
382             //
383             while(!ADCIntStatus(ADC0_BASE, 3, false))
384             {
385             }
386
387             //
388             // Clear the ADC interrupt flag.
389             //
390             ADCIntClear(ADC0_BASE, 3);
391
392             //
393             // Read ADC Value.
394             //
395             ADCSequenceDataGet(ADC0_BASE, 3, ulADC0_Value);
```

```
396
397             //
398             // Use non-calibrated conversion provided in the data sheet.  Make
399             // sure you divide last to avoid dropout.
400             //
401             ulTemp_ValueC = ((1475 * 1023) - (2250 * ulADC0_Value[0])) / 10230;
402
403 }
```

measurements.c

```
1  /*
2   * pushButtons.h
3   *
4   *  Created on: 30.07.2013
5   *      Author: Henrik
6   */
7
8  #ifndef PUSHBUTTONS_H_
9  #define PUSHBUTTONS_H_
10
11 void PB_ISR(void);
12 void TEST_ISR(void);
13 void pBInit(void);
14
15 #endif /* PUSHBUTTONS_H_ */
```

pushButtons.h

```
1  /*
2   * pushButtons.c
3   *
4   *  Created on: 30.07.2013
5   *      Author: Henrik
6   */
7  #include "inc/lm3s9b92.h"
8  #include "inc/hw_memmap.h"
9  #include "inc/hw_types.h"
10 #include "inc/hw_ints.h"
11 #include "driverlib/gpio.h"
12 #include "driverlib/sysctl.h"
13 #include "driverlib/interrupt.h"
14
15 #include "pushButtons.h"
16 #include "ledDriver.h"
17 #include "communication.h"
18 #include "measurements.h"
19
20 char dcDefaultValue[16] = { // DC Values which will be set by pushbuttons
21         35, // Channel 0
22         35, // Channel 1
23         35, // Channel 2
24         35, // Channel 3
25         35, // Channel 4
26         35, // Channel 5
27         35, // Channel 6
28         35, // Channel 7
29         35, // Channel 8
30         35, // Channel 9
31         35, // Channel 10
32         35, // Channel 11
33         35, // Channel 12
34         35, // Channel 13
35         35, // Channel 14
36         35, // Channel 15
37 };
38
39 unsigned long ulDefaultLEDTimeOn = 1000; //default value LED on ms
40 unsigned long ulDefaultTimeOff = 1000; // default value LED off ms
41 unsigned long ulDefaultTimeTotal = 0; // default value measurement duration
42
43 enum states
44 { // led groups
45     START,
46     RGB,
47     DUOLED,
48     TWO3MMLED,
49     WHITE,
50     TWO5MDLED,
```

```
51          EXTLED
52      };
53      char cPBState = START;
54      int ledon = 0;
55      unsigned long ulPBIntStatus;
56
57      // Init. push buttons
58      /*********************************************************/
59      void pBInit(void)
60      {
61          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
62          GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
63          // PB3 init
64
65          GPIOIntTypeSet(GPIO_PORTJ_BASE, GPIO_PIN_6|GPIO_PIN_7, GPIO_FALLING_EDGE);//Interrupt to be triggered on falling edge
66          GPIOPinIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_6 | GPIO_PIN_7);
67          IntRegister(INT_GPIOJ, PB_ISR);
68          IntEnable(INT_GPIOJ);
69          IntPrioritySet(INT_GPIOJ, 0);
70
71      }
72
73      // Push button ISR
74      /*********************************************************/
75      void PB_ISR(void)
76      {
77          char i = 0;
78          ulPBIntStatus = GPIOPinIntStatus(GPIO_PORTJ_BASE, true);
79          // First stop measurements, If LED Button is pushed
80
81          setCycle(ulDefaultLEDTimeOn, ulDefaultTimeOff, ulDefaultTimeTotal);
82          // put dc values of all LED's to 0
83          for(i = 0;i < NUM_LED_CH; i++ ){
84              setDCValue(i, 0);
85          }
86
87          if(ulPBIntStatus == GPIO_PIN_7)
88          {
89              ProcessStop(0,0); // function from communication, used everywhere to stop
90              switch (cPBState)
91              {
92              case START:
93              case EXTLED:
94                  cPBState = RGB;
95                  setDCValue(0, dcDefaultValue[0]);
96                  setDCValue(1, dcDefaultValue[1]);
97                  setDCValue(2, dcDefaultValue[2]);
98                  break;
99              case RGB:
100                 cPBState = DUOLED;
101                 setDCValue(3, dcDefaultValue[3]);
102                 setDCValue(4, dcDefaultValue[4]);
103                 break;
104             case DUOLED:
105                 cPBState = TWO3MMLED;
106                 setDCValue(5, dcDefaultValue[5]);
107                 setDCValue(6, dcDefaultValue[6]);
108                 break;
109             case TWO3MMLED:
110                 cPBState = WHITE;
111                 setDCValue(7, dcDefaultValue[7]);
112                 break;
113             case WHITE:
114                 cPBState = TWOSMDLED;
115                 setDCValue(8, dcDefaultValue[8]);
116                 setDCValue(9, dcDefaultValue[9]);
117                 break;
118             case TWOSMDLED:
119                 cPBState = EXTLED;
120                 setDCValue(10, dcDefaultValue[10]);
121                 setDCValue(11, dcDefaultValue[11]);
122                 break;
123             default:
124                 cPBState = START;
125                 break;
126             }
127             ProcessStart(0,0);
128             GPIOPinIntClear(GPIO_PORTJ_BASE, GPIO_PIN_7);//clear interrupt flag
129         }
130         else if(ulPBIntStatus == GPIO_PIN_6)
131         {
132             ProcessStop(0,0);
133             cPBState = START;
134             GPIOPinIntClear(GPIO_PORTJ_BASE, GPIO_PIN_6);//clear interrupt flag
135
```

```
136         }
137         else if(ulPBIntStatus == GPIO_PIN_5)
138         {
139
140         }
141
142         // start process with changed value
143         GPIOPinIntClear(GPIO_PORTB_BASE, GPIO_PIN_4);//clear interrupt flagg
144         if (ledon == 0){
145             GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, GPIO_PIN_0);
146             ledon = 1;
147         }
148         else{
149             GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, 0);
150             ledon = 0;
151         }
152     }
```

cmdline.h

```
1   //*****************************************************************************
2   //
3   // cmdline.h - Prototypes for command line processing functions.
4   //
5   // Copyright (c) 2007-2012 Texas Instruments Incorporated.  All rights reserved.
6   // Software License Agreement
7   //
8   // Texas Instruments (TI) is supplying this software for use solely and
9   // exclusively on TI's microcontroller products. The software is owned by
10  // TI and/or its suppliers, and is protected under applicable copyright
11  // laws. You may not combine this software with "viral" open-source
12  // software in order to form a larger program.
13  //
14  // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
15  // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
16  // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
17  // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
18  // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
19  // DAMAGES, FOR ANY REASON WHATSOEVER.
20  //
21  // This is part of revision 9107 of the Stellaris Firmware Development Package.
22  //
23  //*****************************************************************************
24
25  #ifndef __CMDLINE_H__
26  #define __CMDLINE_H__
27
28  //*****************************************************************************
29  //
30  // If building with a C++ compiler, make all of the definitions in this header
31  // have a C binding.
32  //
33  //*****************************************************************************
34  #ifdef __cplusplus
35  extern "C"
36  {
37  #endif
38
39  //*****************************************************************************
40  //
41  //! \addtogroup cmdline_api
42  //! @{
43  //
44  //*****************************************************************************
45
46  //*****************************************************************************
47  //
48  //! Defines the value that is returned if the command is not found.
49  //
50  //*****************************************************************************
51  #define CMDLINE_BAD_CMD            (-1)
52
53  //*****************************************************************************
54  //
55  //! Defines the value that is returned if there are too many arguments.
56  //
57  //*****************************************************************************
58  #define CMDLINE_TOO_MANY_ARGS      (-2)
59
60  //*****************************************************************************
61  //
62  // Command line function callback type.
63  //
```

```
64  //*****************************************************************************
65  typedef int (*pfnCmdLine)(int argc, char *argv[]);
66
67  //*****************************************************************************
68  //
69  //! Structure for an entry in the command list table.
70  //
71  //*****************************************************************************
72  typedef struct
73  {
74      //
75      //! A pointer to a string containing the name of the command.
76      //
77      const char *pcCmd;
78
79      //
80      //! A function pointer to the implementation of the command.
81      //
82      pfnCmdLine pfnCmd;
83
84      //
85      //! A pointer to a string of brief help text for the command.
86      //
87      const char *pcHelp;
88  }
89  tCmdLineEntry;
90
91  //*****************************************************************************
92  //
93  //! This is the command table that must be provided by the application.
94  //
95  //*****************************************************************************
96  extern tCmdLineEntry g_sCmdTable[];
97
98  //*****************************************************************************
99  //
100 // Close the Doxygen group.
101 //! @}
102 //
103 //*****************************************************************************
104
105 //*****************************************************************************
106 //
107 // Prototypes for the APIs.
108 //
109 //*****************************************************************************
110 extern int CmdLineProcess(char *pcCmdLine);
111
112 //*****************************************************************************
113 //
114 // Mark the end of the C bindings section for C++ compilers.
115 //
116 //*****************************************************************************
117 #ifdef __cplusplus
118 }
119 #endif
120
121 #endif // __CMDLINE_H__
```

cmdline.c

```
1   //*****************************************************************************
2   //
3   // cmdline.c - Functions to help with processing command lines.
4   //
5   // Copyright (c) 2007-2012 Texas Instruments Incorporated.  All rights reserved.
6   // Software License Agreement
7   //
8   // Texas Instruments (TI) is supplying this software for use solely and
9   // exclusively on TI's microcontroller products. The software is owned by
10  // TI and/or its suppliers, and is protected under applicable copyright
11  // laws. You may not combine this software with "viral" open-source
12  // software in order to form a larger program.
13  //
14  // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
15  // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
16  // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
17  // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
18  // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
19  // DAMAGES, FOR ANY REASON WHATSOEVER.
20  //
21  // This is part of revision 9107 of the Stellaris Firmware Development Package.
22  //
```

```
23  //*****************************************************************************
24
25  //*****************************************************************************
26  //
27  //! \addtogroup cmdline_api
28  //! @{
29  //
30  //*****************************************************************************
31
32  #include <string.h>
33  #include "utils/cmdline.h"
34
35  //*****************************************************************************
36  //
37  // Defines the maximum number of arguments that can be parsed.
38  //
39  //*****************************************************************************
40  #ifndef CMDLINE_MAX_ARGS
41  #define CMDLINE_MAX_ARGS        8
42  #endif
43
44  //*****************************************************************************
45  //
46  //! Process a command line string into arguments and execute the command.
47  //!
48  //! \param pcCmdLine points to a string that contains a command line that was
49  //! obtained by an application by some means.
50  //!
51  //! This function will take the supplied command line string and break it up
52  //! into individual arguments.  The first argument is treated as a command and
53  //! is searched for in the command table.  If the command is found, then the
54  //! command function is called and all of the command line arguments are passed
55  //! in the normal argc, argv form.
56  //!
57  //! The command table is contained in an array named <tt>g_sCmdTable</tt> which
58  //! must be provided by the application.
59  //!
60  //! \return Returns \b CMDLINE_BAD_CMD if the command is not found,
61  //! \b CMDLINE_TOO_MANY_ARGS if there are more arguments than can be parsed.
62  //! Otherwise it returns the code that was returned by the command function.
63  //
64  //*****************************************************************************
65  int
66  CmdLineProcess(char *pcCmdLine)
67  {
68      static char *argv[CMDLINE_MAX_ARGS + 1];
69      char *pcChar;
70      int argc;
71      int bFindArg = 1;
72      tCmdLineEntry *pCmdEntry;
73
74      //
75      // Initialize the argument counter, and point to the beginning of the
76      // command line string.
77      //
78      argc = 0;
79      pcChar = pcCmdLine;
80
81      //
82      // Advance through the command line until a zero character is found.
83      //
84      while(*pcChar)
85      {
86          //
87          // If there is a space, then replace it with a zero, and set the flag
88          // to search for the next argument.
89          //
90          if(*pcChar == ' ')
91          {
92              *pcChar = 0;
93              bFindArg = 1;
94          }
95
96          //
97          // Otherwise it is not a space, so it must be a character that is part
98          // of an argument.
99          //
100         else
101         {
102             //
103             // If bFindArg is set, then that means we are looking for the start
104             // of the next argument.
105             //
106             if(bFindArg)
107             {
```

```
108                 //
109                 // As long as the maximum number of arguments has not been
110                 // reached, then save the pointer to the start of this new arg
111                 // in the argv array, and increment the count of args, argc.
112                 //
113                 if(argc < CMDLINE_MAX_ARGS)
114                 {
115                     argv[argc] = pcChar;
116                     argc++;
117                     bFindArg = 0;
118                 }
119 
120                 //
121                 // The maximum number of arguments has been reached so return
122                 // the error.
123                 //
124                 else
125                 {
126                     return(CMDLINE_TOO_MANY_ARGS);
127                 }
128             }
129         }
130 
131         //
132         // Advance to the next character in the command line.
133         //
134         pcChar++;
135     }
136 
137     //
138     // If one or more arguments was found, then process the command.
139     //
140     if(argc)
141     {
142         //
143         // Start at the beginning of the command table, to look for a matching
144         // command.
145         //
146         pCmdEntry = &g_sCmdTable[0];
147 
148         //
149         // Search through the command table until a null command string is
150         // found, which marks the end of the table.
151         //
152         while(pCmdEntry->pcCmd)
153         {
154             //
155             // If this command entry command string matches argv[0], then call
156             // the function for this command, passing the command line
157             // arguments.
158             //
159             if(!strcmp(argv[0], pCmdEntry->pcCmd))
160             {
161                 return(pCmdEntry->pfnCmd(argc, argv));
162             }
163 
164             //
165             // Not found, so advance to the next entry.
166             //
167             pCmdEntry++;
168         }
169     }
170 
171     //
172     // Fall through to here means that no matching command was found, so return
173     // an error.
174     //
175     return(CMDLINE_BAD_CMD);
176 }
177 
178 //*****************************************************************************
179 //
180 // Close the Doxygen group.
181 //! @}
182 //
183 //*****************************************************************************
```

E.2. PC

```
1  function varargout = LED_Reflektometer(varargin)
2  % LED_REFLEKTOMETER MATLAB code for LED_Reflektometer.fig
```

```
3  %      LED_REFLEKTOMETER, by itself, creates a new LED_REFLEKTOMETER or raises the existing
4  %      singleton*.
5  %
6  %      H = LED_REFLEKTOMETER returns the handle to a new LED_REFLEKTOMETER or the handle to
7  %      the existing singleton*.
8  %
9  %      LED_REFLEKTOMETER('CALLBACK',hObject,eventData,handles,...) calls the local
10 %      function named CALLBACK in LED_REFLEKTOMETER.M with the given input arguments.
11 %
12 %      LED_REFLEKTOMETER('Property','Value',...) creates a new LED_REFLEKTOMETER or raises the
13 %      existing singleton*.  Starting from the left, property value pairs are
14 %      applied to the GUI before LED_Reflektometer_OpeningFcn gets called.  An
15 %      unrecognized property name or invalid value makes property application
16 %      stop.  All inputs are passed to LED_Reflektometer_OpeningFcn via varargin.
17 %
18 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help LED_Reflektometer
24
25 % Last Modified by GUIDE v2.5 09-Aug-2013 19:51:40
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @LED_Reflektometer_OpeningFcn, ...
32                    'gui_OutputFcn',  @LED_Reflektometer_OutputFcn, ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 %--- Executes just before LED_Reflektometer is made visible.
48 function LED_Reflektometer_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to LED_Reflektometer (see VARARGIN)
54
55 % Choose default command line output for LED_Reflektometer
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60 delete(instrfindall);
61 serialInfo = instrhwinfo('serial'); % get information of serial ports
62 ports = serialInfo.AvailableSerialPorts; %get all available serial ports
63 set(handles.popupmenu1, 'String', ports) % populate menu with available com ports
64 % deactivate all edit boxes
65 set(handles.valLED0,'Enable','off'); % turn off edit box
66 set(handles.valLED1,'Enable','off'); % turn off edit box
67 set(handles.valLED2,'Enable','off'); % turn off edit box
68 set(handles.valLED3,'Enable','off'); % turn off edit box
69 set(handles.valLED4,'Enable','off'); % turn off edit box
70 set(handles.valLED5,'Enable','off'); % turn off edit box
71 set(handles.valLED6,'Enable','off'); % turn off edit box
72 set(handles.valLED7,'Enable','off'); % turn off edit box
73 set(handles.valLED8,'Enable','off'); % turn off edit box
74 set(handles.valLED9,'Enable','off'); % turn off edit box
75 set(handles.valLED10,'Enable','off'); % turn off edit box
76 set(handles.valLED11,'Enable','off'); % turn off edit box
77 set(handles.startButton,'Enable','off'); % turn off start button
78 set(handles.stopButton,'Enable','off'); % turn off stop button
79 set(handles.statusButton,'Enable','off'); % turn off stop button
80 set(handles.statusButton,'Enable','off'); % turn off stop button
81 set(handles.disconnectButton,'Enable','off'); % turn on disc. button
82 set(handles.rbClear,'Value',1); % turn on disc. button
83 set(handles.rb100,'Value',1); % turn on disc. button
84 userData = get(handles.figure1, 'UserData');
85 userData.connected = 0; % put serieal connection into userdata, e.g. for the stop function
86 userData.disconnect = 0;
87 set(handles.figure1,'UserData',userData);
```

```
88
89  ylabel('Sensorausschlag');
90  xlabel('Zeit/ms');
91  grid on;
92  %set(gcf,'CloseRequestFcn',@my_closefcn) % own close request function
93  % UIWAIT makes LED_Reflektometer wait for user response (see UIRESUME)
94  % uiwait(handles.figure1);
95
96  %--- Outputs from this function are returned to the command line.
97  function varargout = LED_Reflektometer_OutputFcn(hObject, eventdata, handles)
98  % varargout   cell array for returning output args (see VARARGOUT);
99  % hObject    handle to figure
100 % eventdata  reserved - to be defined in a future version of MATLAB
101 % handles    structure with handles and user data (see GUIDATA)
102
103 % Get default command line output from handles structure
104 varargout{1} = handles.output;
105
106 %--- Executes on button press in connectButton.
107 function connectButton_Callback(hObject, eventdata, handles)
108
109 % this function will connect to the led-reflectometer and if connected it
110 % will be monitored
111 bufValues = 10000;
112 statusValues = 200;
113 timeout = 100;
114 attempts = 100;
115 data = zeros(bufValues,4);
116 status = zeros(statusValues,1);
117 index = 1;
118 firstSave = 1;
119
120 % connect to chosen serial port
121 value = get(handles.popupmenu1,'value');
122 strings = get(handles.popupmenu1,'string');
123 string = strings(value);
124 s = serial(string, 'BaudRate', 115200);
125 fopen(s);
126
127 % set global disconnected flag to 0
128 userData = get(handles.figure1,'UserData'); % get userdata
129 userData.disconnect = 0; % will be set to 1 by the disconnect button
130
131 userData.statusValues = statusValues;
132 set(handles.figure1,'UserData',userData); % save userdata
133
134 % check if LED-Reflektometer is on chosen port, check a couple of messages
135 % since controller could be sending measurements at the same time
136 fprintf(s,'status\n'); % try to get led-reflectometer information
137 i = 0;
138 k = 0;
139 while userData.connected == 0 && i < timeout && k < attempts
140     if s.BytesAvailable > 0; % check to avoid blocking fscanf
141         status = fscanf(s, '%s')
142         statusCh = regexp(status,',', 'split');
143         if length(statusCh) == 3 % avoid reading empty fields
144             if strcmp(statusCh(2),'LED-Reflektometer') == 1
145                 userData.connected = 1; % expected word was phrase, programm
146             end                         % is connected to LED-Reflektometer
147         else
148             k = k+1
149         end
150     else
151          pause(0.001); % avoid blocking
152          i = i+1;
153     end
154 end
155
156 % If LED-Reflektometer was detected on chosen port, port will be monitored
157 % for messages from the controller
158 if userData.connected == 1
159
160     % save connected port to userdata
161     userData = get(handles.figure1,'UserData'); % save userdata
162     userData.s = s % copy serial port to userdata for the start and stop button
163     set(handles.figure1,'UserData',userData); % save userdata
164
165     % configure availability of buttons
166     set(handles.startButton,'Enable','on'); % turn on staruserData = get(handles.figure1,'UserData'); % save userdata
167     userData.s = s % copy serial port to userdata for the start and stop buttont button
168     set(handles.stopButton,'Enable','on'); % turn on stop button
169     set(handles.statusButton,'Enable','on'); % turn on stop button
170     set(handles.disconnectButton,'Enable','on'); % turn on disc. button
171     set(handles.connectButton,'Enable','off'); % turn off con. button
172
```

```
173     % start monitoring serial port
174     while userData.disconnect == 0
175         if s.BytesAvailable > 0; % check to avoid blocking fscanf
176             status = fscanf(s, '%s');
177             statusCh = regexp(status,',', 'split');
178             if strcmp(statusCh(1),'#START')
179                 data = zeros(bufValues,4); % reset data array
180                 index = 1; % reset index
181                 filename = datestr(now,'yyyy-mmm-dd_HH-MM-SS');
182                 filename =sprintf('%s%s.csv',filename,get(handles.fileName, 'String'));
183                 firstSave = 1;
184                 set(handles.startButton,'Enable','off'); % turn on start button
185                 set(handles.statusButton,'Enable','off'); % turn on status button
186                 %clear plot
187             elseif strcmp(statusCh(1),'#Measurement')
188                 if index > bufValues % is bufferarray full?
189                     % save array content into file
190                     if firstSave == 1 %create file
191                         dlmwrite(filename,data,'delimiter',',');
192                         firstSave = 0;
193                     else %append to existing file
194                         dlmwrite(filename,data,'delimiter',',','-append');
195                     end
196                     data = zeros(bufValues,4); % reset data array
197                     index = 1;
198                 end
199                 % store received measurement in array
200                 data(index,1) = str2double(statusCh(2));
201                 data(index,2) = str2double(statusCh(3));
202                 data(index,3) = str2double(statusCh(4));
203                 data(index,4) = str2double(statusCh(5));
204                 plot(data(1:index,1),data(1:index,3));
205                 ylabel('Sensorausschlag');
206                 xlabel('Zeit/ms');
207                 grid on;
208                 index = index +1;
209             elseif strcmp(statusCh(1),'#STOP') == 1 % #STOP is sent by controller
210                 set(handles.startButton,'Enable','on'); % turn on start button
211                 set(handles.statusButton,'Enable','on'); % turn on status button
212                 % Save rest of data
213                 if firstSave == 1
214                     dlmwrite(filename,data(1:(index-1), :),'delimiter',',');
215                     firstSave = 0;
216                 else
217                     dlmwrite(filename,data(1:(index-1), :),'delimiter',',','-append');
218                 end
219             end
220             statusOld = get(handles.listbox1,'String');
221
222             % check for numer of shown status messages
223             if length(statusOld) >= statusValues %maximum reached
224                 status = [statusOld(2:statusValues); status];
225             else
226                 status = [statusOld; status];
227             end
228
229             %update status listbox
230             set(handles.listbox1,'String', status);
231             entries = numel(get(handles.listbox1,'string')) ;%get number of entries
232             set(handles.listbox1,'ListboxTop',entries); % display last entry
233             drawnow();
234
235         else
236             pause(0.00001); % nonblocking pause(fscanf blocks entire matlab
237         end             % without pause, no other functions can be called
238         % priority?
239         userData = get(handles.figure1,'UserData'); % get userdata, for disconnect
240     end
241 else % no LED-Reflektometer on selected com port
242     h = msgbox('No cennection possible. Please try different port or check connection of device ','Error','error') ;
243 end
244
245 fclose(s);
246 set(handles.startButton,'Enable','off'); % turn on start button
247 set(handles.stopButton,'Enable','off'); % turn on stop button
248 set(handles.statusButton,'Enable','off'); % turn on stop button
249 set(handles.disconnectButton,'Enable','off'); % turn on disc. button
250 set(handles.connectButton,'Enable','on'); % turn off con. button
251 set(handles.figure1,'UserData',userData); % save userdata
252
253
254 %--- Executes on button press in startButton.
255 function startButton_Callback(hObject, eventdata, handles)
256 % hObject    handle to startButton (see GCBO)
257 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
258 % handles      structure with handles and user data (see GUIDATA)
259
260 set(handles.startButton,'Enable','off'); % turn off start button
261 set(handles.stopButton,'Enable','on'); % turn off Status button
262 set(handles.statusButton,'Enable','off'); % turn off Status button
263
264
265 measurementRate = 0.1; %100ms
266 values = str2num( get(handles.LEDCycle,'String') ) /measurementRate; %a value is sent every 100 ms
267
268 userData = get(handles.figure1, 'UserData');
269 s=userData.s; % get serial connection
270
271 fprintf(s,'\n');%clear buffer
272
273 set(handles.figure1,'UserData',userData);
274 %readasync(s);
275 status = '';
276
277 ledon = str2num( get(handles.LEDON,'String') );
278 ledoff = str2num( get(handles.LEDOFF,'String') );
279 ledcycle = str2num( get(handles.LEDCycle,'String') );
280
281 for i = 0:11
282     test =  get(handles.(sprintf('cbLED%d',i)),'Value');
283     if(test == 1)
284         ledVal = str2num( get(handles.(sprintf('valLED%d',i)),'String') );
285         fprintf(s,'setLED %s\n',sprintf('%d %d',i,ledVal)); %sprintf because
286     else                                                    % fprintf didn't
287         fprintf(s,'setLED %s\n',sprintf('%d %d',i,0));      % like 2 values
288     end
289     status = fscanf(s, '%s');
290     statusOld = get(handles.listbox1,'String');
291     status = [statusOld; status];
292     set(handles.listbox1,'String', status);
293     entries = numel(get(handles.listbox1,'string')) ;%get number of entries
294     set(handles.listbox1,'ListboxTop',entries); % display last entry
295
296     status = fscanf(s, '%s');
297     statusOld = get(handles.listbox1,'String');
298     status = [statusOld; status];
299     set(handles.listbox1,'String', status);
300     entries = numel(get(handles.listbox1,'string')) ;%get number of entries
301     set(handles.listbox1,'ListboxTop',entries); % display last entry
302     drawnow();
303 end
304
305 %configure LED cycles
306 ledOn = str2num( get(handles.LEDON,'String') );
307 ledOff = str2num( get(handles.LEDOFF,'String') );
308 ledCycle = str2num( get(handles.LEDCycle,'String') );
309 LEDstring = sprintf('setCycle %d %d %d\n',ledOn*1000, ledOff*1000, ledCycle*1000)%values converted to ms
310 fprintf(s,'%s\n',LEDstring);
311
312 %configure Sensor filter
313 if get(handles.rbClear,'Value') == 1
314     fprintf(s,'setFilter 1 0\n');
315 elseif get(handles.rbRed,'Value') == 1
316     fprintf(s,'setFilter 0 0\n');
317 elseif get(handles.rbGreen,'Value') == 1
318     fprintf(s,'setFilter 1 1\n');
319 elseif get(handles.rbBlue,'Value') == 1
320     fprintf(s,'setFilter 0 1\n');
321 end
322
323 %configure Sensor scaling
324 if get(handles.rb100,'Value') == 1
325     fprintf(s,'setScaling 1 1\n');
326 elseif get(handles.rb20,'Value') == 1
327     fprintf(s,'setScaling 1 0\n');
328 elseif get(handles.rb2,'Value') == 1
329     fprintf(s,'setScaling 0 1\n');
330 end
331
332 fprintf(s,'start\n');
333
334
335 %--- Executes on button press in statusButton.
336 function statusButton_Callback(hObject, eventdata, handles)
337 % hObject    handle to statusButton (see GCBO)
338 % eventdata  reserved - to be defined in a future version of MATLAB
339 % handles    structure with handles and user data (see GUIDATA)
340
341 userData = get(handles.figure1,'UserData'); % get userdata
342 s = userData.s;
```

```
343  statusValues = userData.statusValues;
344  fprintf(s,'status\n');
345  status = fscanf(s); % repeated text
346  status = fscanf(s);
347
348  statusOld = get(handles.listbox1,'String');
349
350  % check for numer of shown status messages
351  if length(statusOld) >= statusValues %maximum reached
352      status = [statusOld(2:statusValues); status];
353  else
354      status = [statusOld; status];
355  end
356
357   %update status listbox
358   set(handles.listbox1,'String', status);
359   entries = numel(get(handles.listbox1,'string')) ;%get number of entries
360   set(handles.listbox1,'ListboxTop',entries); % display last entry
361   drawnow();
362
363
364  %--- Executes on button press in stopButton.
365  function stopButton_Callback(hObject, eventdata, handles)
366  % hObject    handle to stopButton (see GCBO)
367  % eventdata  reserved - to be defined in a future version of MATLAB
368  % handles    structure with handles and user data (see GUIDATA)
369  userData = get(handles.figure1, 'UserData');
370  fprintf(userData.s,'stop\n');
371
372
373  function LEDON_Callback(hObject, eventdata, handles)
374  % hObject    handle to LEDON (see GCBO)
375  % eventdata  reserved - to be defined in a future version of MATLAB
376  % handles    structure with handles and user data (see GUIDATA)
377
378  % Hints: get(hObject,'String') returns contents of LEDON as text
379  %        str2double(get(hObject,'String')) returns contents of LEDON as a double
380
381
382  %--- Executes during object creation, after setting all properties.
383  function LEDON_CreateFcn(hObject, eventdata, handles)
384  % hObject    handle to LEDON (see GCBO)
385  % eventdata  reserved - to be defined in a future version of MATLAB
386  % handles    empty - handles not created until after all CreateFcns called
387
388  % Hint: edit controls usually have a white background on Windows.
389  %       See ISPC and COMPUTER.
390  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
391      set(hObject,'BackgroundColor','white');
392  end
393
394
395
396  function LEDOFF_Callback(hObject, eventdata, handles)
397  % hObject    handle to LEDOFF (see GCBO)
398  % eventdata  reserved - to be defined in a future version of MATLAB
399  % handles    structure with handles and user data (see GUIDATA)
400
401  % Hints: get(hObject,'String') returns contents of LEDOFF as text
402  %        str2double(get(hObject,'String')) returns contents of LEDOFF as a double
403
404
405  %--- Executes during object creation, after setting all properties.
406  function LEDOFF_CreateFcn(hObject, eventdata, handles)
407  % hObject    handle to LEDOFF (see GCBO)
408  % eventdata  reserved - to be defined in a future version of MATLAB
409  % handles    empty - handles not created until after all CreateFcns called
410
411  % Hint: edit controls usually have a white background on Windows.
412  %       See ISPC and COMPUTER.
413  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
414      set(hObject,'BackgroundColor','white');
415  end
416
417
418
419  function LEDCycle_Callback(hObject, eventdata, handles)
420  % hObject    handle to LEDCycle (see GCBO)
421  % eventdata  reserved - to be defined in a future version of MATLAB
422  % handles    structure with handles and user data (see GUIDATA)
423
424  % Hints: get(hObject,'String') returns contents of LEDCycle as text
425  %        str2double(get(hObject,'String')) returns contents of LEDCycle as a double
426
427
```

```
428 %--- Executes during object creation, after setting all properties.
429 function LEDCycle_CreateFcn(hObject, eventdata, handles)
430 % hObject    handle to LEDCycle (see GCBO)
431 % eventdata  reserved - to be defined in a future version of MATLAB
432 % handles    empty - handles not created until after all CreateFcns called
433
434 % Hint: edit controls usually have a white background on Windows.
435 %       See ISPC and COMPUTER.
436 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
437     set(hObject,'BackgroundColor','white');
438 end
439
440
441
442 function LEDs_Callback(hObject, eventdata, handles)
443 % hObject    handle to LEDs (see GCBO)
444 % eventdata  reserved - to be defined in a future version of MATLAB
445 % handles    structure with handles and user data (see GUIDATA)
446
447 % Hints: get(hObject,'String') returns contents of LEDs as text
448 %        str2double(get(hObject,'String')) returns contents of LEDs as a double
449
450
451 %--- Executes during object creation, after setting all properties.
452 function LEDs_CreateFcn(hObject, eventdata, handles)
453 % hObject    handle to LEDs (see GCBO)
454 % eventdata  reserved - to be defined in a future version of MATLAB
455 % handles    empty - handles not created until after all CreateFcns called
456
457 % Hint: edit controls usually have a white background on Windows.
458 %       See ISPC and COMPUTER.
459 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
460     set(hObject,'BackgroundColor','white');
461 end
462
463
464 %--- Executes on selection change in popupmenu1.
465 function popupmenu1_Callback(hObject, eventdata, handles)
466 % hObject    handle to popupmenu1 (see GCBO)
467 % eventdata  reserved - to be defined in a future version of MATLAB
468 % handles    structure with handles and user data (see GUIDATA)
469
470 % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
471 %        contents{get(hObject,'Value')} returns selected item from popupmenu1
472
473
474
475
476 %--- Executes during object creation, after setting all properties.
477 function popupmenu1_CreateFcn(hObject, eventdata, handles)
478 % hObject    handle to popupmenu1 (see GCBO)
479 % eventdata  reserved - to be defined in a future version of MATLAB
480 % handles    empty - handles not created until after all CreateFcns called
481
482 % Hint: popupmenu controls usually have a white background on Windows.
483 %       See ISPC and COMPUTER.
484 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
485     set(hObject,'BackgroundColor','white');
486 end
487
488
489 %--- Executes on selection change in listbox1.
490 function listbox1_Callback(hObject, eventdata, handles)
491 % hObject    handle to listbox1 (see GCBO)
492 % eventdata  reserved - to be defined in a future version of MATLAB
493 % handles    structure with handles and user data (see GUIDATA)
494
495 % Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as cell array
496 %        contents{get(hObject,'Value')} returns selected item from listbox1
497
498
499 %--- Executes during object creation, after setting all properties.
500 function listbox1_CreateFcn(hObject, eventdata, handles)
501 % hObject    handle to listbox1 (see GCBO)
502 % eventdata  reserved - to be defined in a future version of MATLAB
503 % handles    empty - handles not created until after all CreateFcns called
504
505 % Hint: listbox controls usually have a white background on Windows.
506 %       See ISPC and COMPUTER.
507 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
508     set(hObject,'BackgroundColor','white');
509 end
510
511
512 %--- Executes on button press in cbLED0.
```

```
513 function cbLED0_Callback(hObject, eventdata, handles)
514 % hObject    handle to cbLED0 (see GCBO)
515 % eventdata  reserved - to be defined in a future version of MATLAB
516 % handles    structure with handles and user data (see GUIDATA)
517
518 % Hint: get(hObject,'Value') returns toggle state of cbLED0
519 test = get(hObject,'Value');
520 if test == 1
521     set(handles.valLED0,'Enable','on'); % turn off Status button
522 else
523      set(handles.valLED0,'Enable','off'); % turn off Status button
524 end
525
526
527 %--- Executes on button press in cbLED1.
528 function cbLED1_Callback(hObject, eventdata, handles)
529 % hObject    handle to cbLED1 (see GCBO)
530 % eventdata  reserved - to be defined in a future version of MATLAB
531 % handles    structure with handles and user data (see GUIDATA)
532
533 % Hint: get(hObject,'Value') returns toggle state of cbLED1
534 test = get(hObject,'Value');
535 if test == 1
536     set(handles.valLED1,'Enable','on'); % turn off Status button
537 else
538      set(handles.valLED1,'Enable','off'); % turn off Status button
539 end
540
541
542 %--- Executes on button press in cbLED2.
543 function cbLED2_Callback(hObject, eventdata, handles)
544 % hObject    handle to cbLED2 (see GCBO)
545 % eventdata  reserved - to be defined in a future version of MATLAB
546 % handles    structure with handles and user data (see GUIDATA)
547
548 % Hint: get(hObject,'Value') returns toggle state of cbLED2
549 test = get(hObject,'Value');
550 if test == 1
551     set(handles.valLED2,'Enable','on'); % turn off Status button
552 else
553      set(handles.valLED2,'Enable','off'); % turn off Status button
554 end
555
556 %--- Executes on button press in cbLED3.
557 function cbLED3_Callback(hObject, eventdata, handles)
558 % hObject    handle to cbLED3 (see GCBO)
559 % eventdata  reserved - to be defined in a future version of MATLAB
560 % handles    structure with handles and user data (see GUIDATA)
561
562 % Hint: get(hObject,'Value') returns toggle state of cbLED3
563 test = get(hObject,'Value');
564 if test == 1
565     set(handles.valLED3,'Enable','on'); % turn off Status button
566 else
567      set(handles.valLED3,'Enable','off'); % turn off Status button
568 end
569
570 %--- Executes on button press in cbLED4.
571 function cbLED4_Callback(hObject, eventdata, handles)
572 % hObject    handle to cbLED4 (see GCBO)
573 % eventdata  reserved - to be defined in a future version of MATLAB
574 % handles    structure with handles and user data (see GUIDATA)
575
576 % Hint: get(hObject,'Value') returns toggle state of cbLED4
577 test = get(hObject,'Value');
578 if test == 1
579     set(handles.valLED4,'Enable','on'); % turn off Status button
580 else
581      set(handles.valLED4,'Enable','off'); % turn off Status button
582 end
583
584 %--- Executes on button press in cbLED5.
585 function cbLED5_Callback(hObject, eventdata, handles)
586 % hObject    handle to cbLED5 (see GCBO)
587 % eventdata  reserved - to be defined in a future version of MATLAB
588 % handles    structure with handles and user data (see GUIDATA)
589
590 % Hint: get(hObject,'Value') returns toggle state of cbLED5
591 test = get(hObject,'Value');
592 if test == 1
593     set(handles.valLED5,'Enable','on'); % turn off Status button
594 else
595      set(handles.valLED5,'Enable','off'); % turn off Status button
596 end
597
```

```
598 %--- Executes on button press in cbLED6.
599 function cbLED6_Callback(hObject, eventdata, handles)
600 % hObject    handle to cbLED6 (see GCBO)
601 % eventdata  reserved - to be defined in a future version of MATLAB
602 % handles    structure with handles and user data (see GUIDATA)
603
604 % Hint: get(hObject,'Value') returns toggle state of cbLED6
605 test = get(hObject,'Value');
606 if test == 1
607     set(handles.valLED6,'Enable','on'); % turn off Status button
608 else
609      set(handles.valLED6,'Enable','off'); % turn off Status button
610 end
611
612 %--- Executes on button press in cbLED7.
613 function cbLED7_Callback(hObject, eventdata, handles)
614 % hObject    handle to cbLED7 (see GCBO)
615 % eventdata  reserved - to be defined in a future version of MATLAB
616 % handles    structure with handles and user data (see GUIDATA)
617
618 % Hint: get(hObject,'Value') returns toggle state of cbLED7
619 test = get(hObject,'Value');
620 if test == 1
621     set(handles.valLED7,'Enable','on'); % turn off Status button
622 else
623      set(handles.valLED7,'Enable','off'); % turn off Status button
624 end
625
626 %--- Executes on button press in cbLED8.
627 function cbLED8_Callback(hObject, eventdata, handles)
628 % hObject    handle to cbLED8 (see GCBO)
629 % eventdata  reserved - to be defined in a future version of MATLAB
630 % handles    structure with handles and user data (see GUIDATA)
631
632 % Hint: get(hObject,'Value') returns toggle state of cbLED8
633 test = get(hObject,'Value');
634 if test == 1
635     set(handles.valLED8,'Enable','on'); % turn off Status button
636 else
637      set(handles.valLED8,'Enable','off'); % turn off Status button
638 end
639
640 %--- Executes on button press in cbLED9.
641 function cbLED9_Callback(hObject, eventdata, handles)
642 % hObject    handle to cbLED9 (see GCBO)
643 % eventdata  reserved - to be defined in a future version of MATLAB
644 % handles    structure with handles and user data (see GUIDATA)
645
646 % Hint: get(hObject,'Value') returns toggle state of cbLED9
647 test = get(hObject,'Value');
648 if test == 1
649     set(handles.valLED9,'Enable','on'); % turn off Status button
650 else
651      set(handles.valLED9,'Enable','off'); % turn off Status button
652 end
653
654 %--- Executes on button press in cbLED10.
655 function cbLED10_Callback(hObject, eventdata, handles)
656 % hObject    handle to cbLED10 (see GCBO)
657 % eventdata  reserved - to be defined in a future version of MATLAB
658 % handles    structure with handles and user data (see GUIDATA)
659
660 % Hint: get(hObject,'Value') returns toggle state of cbLED10
661 test = get(hObject,'Value');
662 if test == 1
663     set(handles.valLED10,'Enable','on'); % turn off Status button
664 else
665      set(handles.valLED10,'Enable','off'); % turn off Status button
666 end
667
668 %--- Executes on button press in cbLED9.
669 function cbLED11_Callback(hObject, eventdata, handles)
670 % hObject    handle to cbLED9 (see GCBO)
671 % eventdata  reserved - to be defined in a future version of MATLAB
672 % handles    structure with handles and user data (see GUIDATA)
673
674 % Hint: get(hObject,'Value') returns toggle state of cbLED9
675 test = get(hObject,'Value');
676 if test == 1
677     set(handles.valLED11,'Enable','on'); % turn off Status button
678 else
679      set(handles.valLED11,'Enable','off'); % turn off Status button
680 end
681
682
```

```
683 function valLED0_Callback(hObject, eventdata, handles)
684 % hObject    handle to valLED0 (see GCBO)
685 % eventdata  reserved - to be defined in a future version of MATLAB
686 % handles    structure with handles and user data (see GUIDATA)
687
688 % Hints: get(hObject,'String') returns contents of valLED0 as text
689 %        str2double(get(hObject,'String')) returns contents of valLED0 as a double
690
691
692 %--- Executes during object creation, after setting all properties.
693 function valLED0_CreateFcn(hObject, eventdata, handles)
694 % hObject    handle to valLED0 (see GCBO)
695 % eventdata  reserved - to be defined in a future version of MATLAB
696 % handles    empty - handles not created until after all CreateFcns called
697
698 % Hint: edit controls usually have a white background on Windows.
699 %       See ISPC and COMPUTER.
700 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
701     set(hObject,'BackgroundColor','white');
702 end
703
704
705
706 function valLED1_Callback(hObject, eventdata, handles)
707 % hObject    handle to valLED1 (see GCBO)
708 % eventdata  reserved - to be defined in a future version of MATLAB
709 % handles    structure with handles and user data (see GUIDATA)
710
711 % Hints: get(hObject,'String') returns contents of valLED1 as text
712 %        str2double(get(hObject,'String')) returns contents of valLED1 as a double
713
714
715 %--- Executes during object creation, after setting all properties.
716 function valLED1_CreateFcn(hObject, eventdata, handles)
717 % hObject    handle to valLED1 (see GCBO)
718 % eventdata  reserved - to be defined in a future version of MATLAB
719 % handles    empty - handles not created until after all CreateFcns called
720
721 % Hint: edit controls usually have a white background on Windows.
722 %       See ISPC and COMPUTER.
723 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
724     set(hObject,'BackgroundColor','white');
725 end
726
727
728
729 function valLED2_Callback(hObject, eventdata, handles)
730 % hObject    handle to valLED2 (see GCBO)
731 % eventdata  reserved - to be defined in a future version of MATLAB
732 % handles    structure with handles and user data (see GUIDATA)
733
734 % Hints: get(hObject,'String') returns contents of valLED2 as text
735 %        str2double(get(hObject,'String')) returns contents of valLED2 as a double
736
737
738 %--- Executes during object creation, after setting all properties.
739 function valLED2_CreateFcn(hObject, eventdata, handles)
740 % hObject    handle to valLED2 (see GCBO)
741 % eventdata  reserved - to be defined in a future version of MATLAB
742 % handles    empty - handles not created until after all CreateFcns called
743
744 % Hint: edit controls usually have a white background on Windows.
745 %       See ISPC and COMPUTER.
746 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
747     set(hObject,'BackgroundColor','white');
748 end
749
750
751
752 function valLED3_Callback(hObject, eventdata, handles)
753 % hObject    handle to valLED3 (see GCBO)
754 % eventdata  reserved - to be defined in a future version of MATLAB
755 % handles    structure with handles and user data (see GUIDATA)
756
757 % Hints: get(hObject,'String') returns contents of valLED3 as text
758 %        str2double(get(hObject,'String')) returns contents of valLED3 as a double
759
760
761 %--- Executes during object creation, after setting all properties.
762 function valLED3_CreateFcn(hObject, eventdata, handles)
763 % hObject    handle to valLED3 (see GCBO)
764 % eventdata  reserved - to be defined in a future version of MATLAB
765 % handles    empty - handles not created until after all CreateFcns called
766
767 % Hint: edit controls usually have a white background on Windows.
```

```
768 %       See ISPC and COMPUTER.
769 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
770     set(hObject,'BackgroundColor','white');
771 end
772
773
774
775 function valLED4_Callback(hObject, eventdata, handles)
776
777
778 %--- Executes during object creation, after setting all properties.
779 function valLED4_CreateFcn(hObject, eventdata, handles)
780
781 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
782     set(hObject,'BackgroundColor','white');
783 end
784
785
786
787 function valLED5_Callback(hObject, eventdata, handles)
788
789
790 %--- Executes during object creation, after setting all properties.
791 function valLED5_CreateFcn(hObject, eventdata, handles)
792
793 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
794     set(hObject,'BackgroundColor','white');
795 end
796
797
798
799 function valLED6_Callback(hObject, eventdata, handles)
800
801 function valLED6_CreateFcn(hObject, eventdata, handles)
802
803 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
804     set(hObject,'BackgroundColor','white');
805 end
806
807
808
809 function valLED7_Callback(hObject, eventdata, handles)
810
811
812 %--- Executes during object creation, after setting all properties.
813 function valLED7_CreateFcn(hObject, eventdata, handles)
814
815 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
816     set(hObject,'BackgroundColor','white');
817 end
818
819
820
821 function valLED8_Callback(hObject, eventdata, handles)
822
823
824 %--- Executes during object creation, after setting all properties.
825 function valLED8_CreateFcn(hObject, eventdata, handles)
826
827 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
828     set(hObject,'BackgroundColor','white');
829 end
830
831
832
833 function valLED9_Callback(hObject, eventdata, handles)
834
835
836 %--- Executes during object creation, after setting all properties.
837 function valLED9_CreateFcn(hObject, eventdata, handles)
838
839 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
840     set(hObject,'BackgroundColor','white');
841 end
842
843
844
845 function valLED10_Callback(hObject, eventdata, handles)
846
847
848 %--- Executes during object creation, after setting all properties.
849 function valLED10_CreateFcn(hObject, eventdata, handles)
850
851 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
852     set(hObject,'BackgroundColor','white');
```

```
853 end
854
855
856
857 function valLED11_Callback(hObject, eventdata, handles)
858
859
860 %--- Executes during object creation, after setting all properties.
861 function valLED11_CreateFcn(hObject, eventdata, handles)
862
863 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
864     set(hObject,'BackgroundColor','white');
865 end
866
867
868 %--- Executes on button press in disconnectButton.
869 function disconnectButton_Callback(hObject, eventdata, handles)
870
871 userData = get(handles.figure1, 'UserData');
872 userData.disconnect = 1;
873 set(handles.figure1,'UserData',userData); % save userdata
874
875
876
877 function fileName_Callback(hObject, eventdata, handles)
878
879
880 %--- Executes during object creation, after setting all properties.
881 function fileName_CreateFcn(hObject, eventdata, handles)
882
883 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
884     set(hObject,'BackgroundColor','white');
885 end
886
887
888 %--- Executes on button press in rbClear.
889 function rbClear_Callback(hObject, eventdata, handles)
890
891 if get(hObject,'Value') == 1
892     set(handles.rbRed,'Value',0);
893     set(handles.rbGreen,'Value',0);
894     set(handles.rbBlue,'Value',0);
895 end
896
897
898 %--- Executes on button press in rbRed.
899 function rbRed_Callback(hObject, eventdata, handles)
900
901 if get(hObject,'Value') == 1
902     set(handles.rbClear,'Value',0);
903     set(handles.rbGreen,'Value',0);
904     set(handles.rbBlue,'Value',0);
905 end
906
907
908 %--- Executes on button press in rbGreen.
909 function rbGreen_Callback(hObject, eventdata, handles)
910
911 if get(hObject,'Value') == 1
912     set(handles.rbRed,'Value',0);
913     set(handles.rbClear,'Value',0);
914     set(handles.rbBlue,'Value',0);
915 end
916
917
918 %--- Executes on button press in rbBlue.
919 function rbBlue_Callback(hObject, eventdata, handles)
920
921 if get(hObject,'Value') == 1
922     set(handles.rbRed,'Value',0);
923     set(handles.rbGreen,'Value',0);
924     set(handles.rbClear,'Value',0);
925 end
926
927
928 %--- Executes on button press in rb100.
929 function rb100_Callback(hObject, eventdata, handles)
930 if get(hObject,'Value') == 1
931     set(handles.rb20,'Value',0);
932     set(handles.rb2,'Value',0);
933 end
934
935
936 %--- Executes on button press in rb20.
937 function rb20_Callback(hObject, eventdata, handles)
```

```
938
939 if get(hObject,'Value') == 1
940     set(handles.rb100,'Value',0);
941     set(handles.rb2,'Value',0);
942 end
943
944
945 %--- Executes on button press in rb2.
946 function rb2_Callback(hObject, eventdata, handles)
947
948 if get(hObject,'Value') == 1
949     set(handles.rb100,'Value',0);
950     set(handles.rb20,'Value',0);
951 end
```

F. Matlab-Obefläche

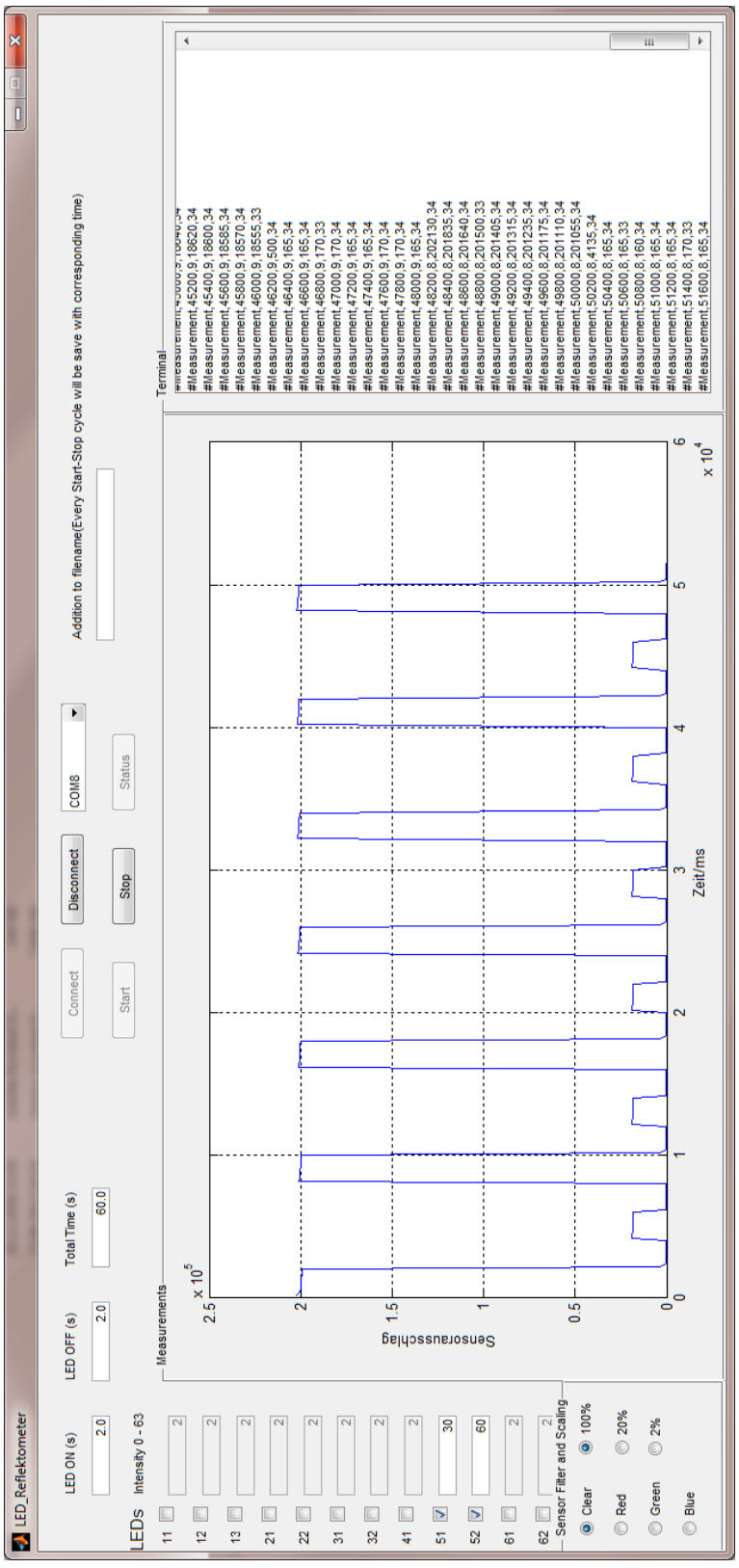

Abbildung F.1.: Steuerungssoftware des LED-Reflektometers

G. Sonstiges

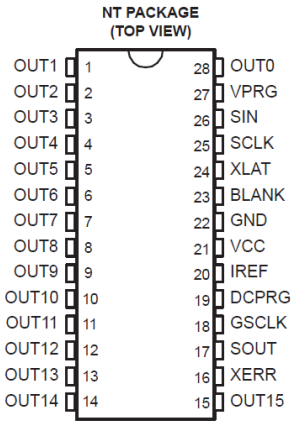

Abbildung G.1.: Gehäuse des TLC5940 LED-Treiber von Texas Instruments aus [50]

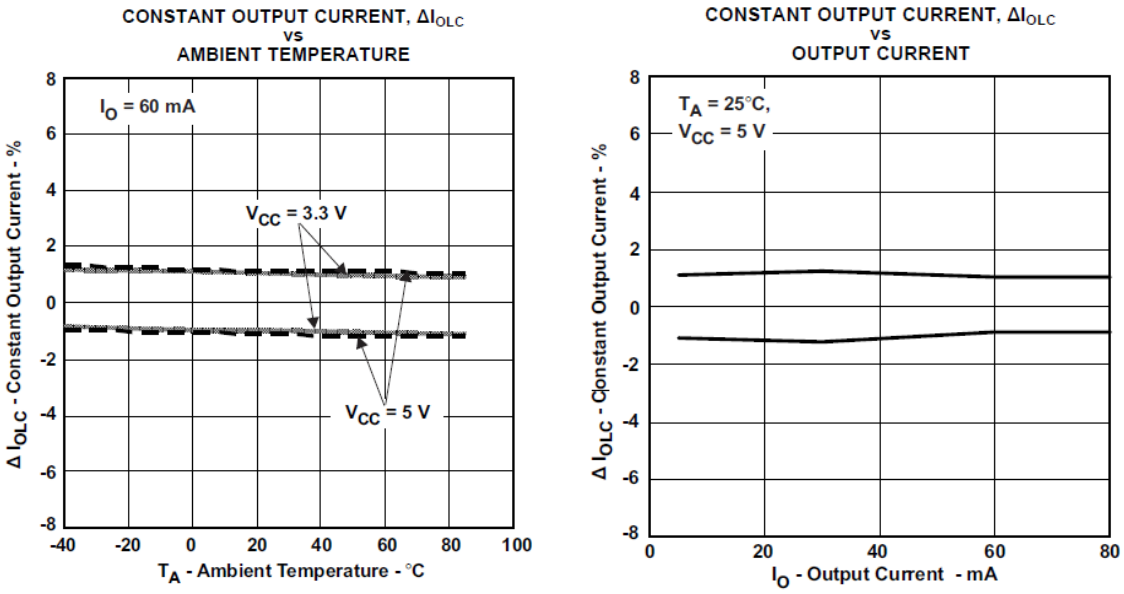

Abbildung G.2.: Links: Abweichung des Ausgangsstromes in Prozent gegen die Temperatur bei 60 mA. Rechts: Prozentuale Abweichung des Ausgangsstromes [50]

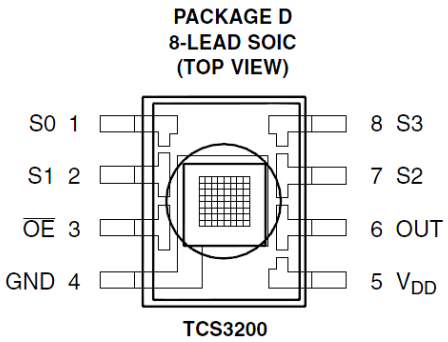

Abbildung G.3.: Licht-/Frequenz-Wandler TAOS TCS3200 aus [47]

Versicherung über die Selbstständigkeit

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

Hamburg, 27. September 2013	
Ort, Datum	Unterschrift