HAW HAMBURG

Bachelorarbeit

Simon von Riegen

Controllersteuerung eines faseroptischen Sensors für Lithium-Ionen-Batterien

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Simon von Riegen

Controllersteuerung eines faseroptischen Sensors für Lithium-Ionen-Batterien

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Lutz Leutelt

Eingereicht am: 16.08.2021

Simon von Riegen

Thema der Arbeit

Controllersteuerung eines faseroptischen Sensors für Lithium-Ionen-Batterien

Stichworte

Optik, faseroptischer Sensor, Lithium-Ionen-Batterien, Mikrocontroller, Ringoszillator

Kurzzusammenfassung

Diese Arbeit ist im Bereich der faseroptischen Sensoren zur Zustandsbestimmung in Lithium-Ionen-Batterien angesiedelt. Mit Hinblick auf eine anwendungsnahe Integration der Messmethode erfolgt die Erfassung des optischen Sensorsignals mit möglichst einfacher Elektronik. Passend dazu wird entsprechende Software entwickelt. Die Steuerung der Elektronik erfolgt durch einen Mikrocontroller. Die Daten werden anschließend offline am Computer ausgewertet. Das Lichtsignal des faseroptischen Sensors wird mit zwei Methoden erfasst. Zum einen wird ein Farbsensor benutzt. Zum anderen erfolgt mit Hilfe eines Ringoszillators eine Umwandlung des Lichtes in eine Frequenz. Beide Methoden werden umgesetzt und miteinander verglichen.

Simon von Riegen

Title of Thesis

fiber optic sensor for lithium ion batteries controlled by a microcontroller

Keywords

optic, fiber optic sensor, ring oscillator, lithium ion batteries, microcontroller

Abstract

This work is located in the field of fiber optic sensors for determining the condition of lithium-ion batteries. With regard to an application-oriented integration of the measurement method, the acquisition of the optical sensor signal takes place with the simplest possible electronics. Appropriate software is being developed for this. The electronics are

controlled by a microcontroller. The data is then evaluated offline on the computer. The light signal from the fiber optic sensor is recorded using two methods. On the one hand, a color sensor is used. On the other hand, the light is converted into a frequency with the help of a ring oscillator. Both methods are implemented and compared with one another.

Danksagung

An dieser Stelle möchte ich mich zunächst bei meinen Erst- und Zweitprüfer bedanken, Herrn Prof. Dr. Karl-Ragmar Riemschneider und Herrn Prof. Dr. Lutz Leutelt. Auch gegenüber den Herren Günter Müller, Florian Rittweger und Philipp Schiepel möchte ich aufgrund ihrer großartigen Hilfe bei fachlichen und formalen Problemen meinen Dank aussprechen.
Gleichermaßen bedanke ich mich bei meinen Eltern Angelika und Stephan von Riegen, welche mich in all den Jahren meines Studiums moralisch und auch finanziell erheblich unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **x**

1 Einleitung **1**

2 Einarbeitung und Recherche **3**

2.1 Konzepte/Erläuterung für Batteriemanagement und dessen Forschungsbedarf . . . 3

2.2 Einarbeitung in die Grundlagen faseroptischer Sensoren . . . 4

2.3 Recherche von vergleichbaren Vorarbeiten an der HAW von Herrn Klockmann und Herrn Nasimzada . . . 11

2.4 Inbetriebnahme der Entwicklungsumgebung für ARM-Mikrocontroller, bspw. DAvE (Infineon) . . . 12

2.5 Grundlagenrecherche zu Bauelementen und Schnittstellen . . . 14

3 Konzeption **18**

3.1 Spezifikation der Anforderungen . . . 18

3.2 Konstruktion des Lichtwellenleiter-Adapters . . . 19

3.2.1 CAD-Konstruktion der Adapter zur Ankopplung der Lichtleitfasern 19

3.2.2 Fertigung des Adapters mit 3D-Druck . . . 20

3.2.3 Montage, Erprobung und Korrektur des Adapters . . . 21

3.3 Systemarchitektur als Übersicht . . . 23

3.4 Grundsätzlicher Aufbau des Systems mit Schnittstellen . . . 26

3.5 Entwurf der Steuerungs- und Softwarestruktur . . . 26

3.6 Festlegung von Dateiformaten, Bedienoberfläche und Funktionsumfang . . . 27

4 Entwicklung der Hardware **28**

4.1 Auswahl geeigneter Bauelemente für die Lichtquelle und den Sensor . . . 28

4.2 Entwurf der Analogbaugruppen und Stromversorgung . . . 29

4.3 Definition der Schnittstellen zu den Controllerboards 29
4.4 Platinenentwurf und Bestückung 30
4.5 Inbetriebnahme und Test 32

5 Entwicklung der Controllersoftware 37
5.1 Ansteuerung der Lichtquellen 37
5.1.1 LED-Steuerung der Farbsensorplatine 37
5.1.2 LED-Steuerung der Ringoszillatorplatine 38
5.2 Digitale Schnittstelle für den integrierten Lichtsensor/Farbsensor 39
5.3 Funktion des Ringoszillator 40
5.4 Schnittstelle zum PC 41
5.5 Steuerungskonzept 41
5.6 Modul- und Gesamtfunktionstest gemäß Testplan 42

6 Entwicklung der PC-Software 45
6.1 Umsetzung als Matlab-Skript 45
6.2 Schnittstelle zum Controller 45
6.3 Visualisierung von Messdaten 46
6.4 Aufzeichnung der Messdaten mit Zeitstempel 47
6.5 Modul- und Gesamtfunktionstest gemäß Testplan 49

7 Erprobung des gesamten Messsystems 51
7.1 Planung der Messreihen 51
7.2 Durchführung und Auswertung 52
7.3 Auswertung von Einflussgrößen und Streuungen, Kalibrierung bzw. Referenzmessungen 65
7.4 Untersuchung des Aufbaus auf Reproduzierbarkeit 70

8 Einordnung, Bewertung und Ausblick 71
8.1 Zusammenfassung der Ergebnisse, Beurteilung des Messkonzeptes 71
8.2 Bewertung der gewählten Konzepte und Lösungsvarianten 72
8.3 Offene Punkte und einschränkende Erfahrungen und Ausblick 73

Literaturverzeichnis 76

A Anhang 79
A.1 Schaltplan 80
A.2 Platinenlayout 82

A.3 Quellcodes 84
A.3.1 Mikrocontroller C-Codes 84
A.3.2 Matlab Code 117
A.4 Aufgabenstellung 130
A.5 Wichtige Datenblätter 134

Selbstständigkeitserklärung **171**

Abbildungsverzeichnis

2.1 Aufbau eines Lichtwellenleiters 4
2.2 Lichtverlauf in einer Multimodefaser mit Stufenindexprofil 5
2.3 Lichtverlauf in einer Multimodefaser mit Gradientenindexprofil 6
2.4 Lichtverlauf in einer Monomodefaser 6
2.5 Visualisierung des Snelliusschen Brechungsgesetzes 7
2.6 Visualisierung des Snelliusschen Brechungsgesetzes: Spezialfall Totalreflexion 8
2.7 Aufbau einer Lithium-Ionen Batteriezelle 9
2.8 Optischer Effekt bei der Anoden einer Lithium-Ionen-Batteriezelle 10
2.9 Optischer Effekt bei Kathodenmaterialen einer Lithium-Ionen-Batteriezelle 11
2.10 Reihenfolge des Datenrahmens bei der Übertragung und Interpretation der Bits 17

3.1 LWL-Adapter für LED 20
3.2 LWL-Adapter für Farbsensor 20
3.3 Blockdiagramm für die Platine mit Farbsensor 23
3.4 Blockdiagramm für die Platine mit Ringoszillator 25

4.1 Platine als 3D-Ansicht in der Draufsicht, generiert mit der Entwicklungssoftware KiCad 31
4.2 Mögliche Einstellungen des Jumpers und deren Teilerfaktor 34
4.3 Ausgelesene Device-ID des Farbsensors auf dem I^2C-Bus als Bitfolge. Die gelbe Linie ist die SDA Leitung, die blaue Linie ist die SCL Leitung und unten die übertragene Bitfolge 35
4.4 Mit dem Ringoszillator gemessene Frequenz in Hz auf Jumperposition Q3 36
4.5 Mit dem Oszilloskop gemessene Frequenz in kHz 36

5.1 Empfindlichkeit des Farbsensors 43

6.1 Beispielplot des Farbsensors für eine Messung mit der RGB-LED 48

7.1 Lichtwellenleiter mit schwacher Biegung 52
7.2 Lichtwellenleiter mit starker Biegung 52
7.3 Infrarot-Messung mit Farbsensor mit Plastikfaser 53
7.4 RGB-Messung mit Farbsensor mit Plastikfaser 54
7.5 Infrarot-Messung mit Farbsensor mit Glasfaser 55
7.6 RGB-Messung mit Farbsensor mit Glasfaser 56
7.7 Infrarot-Messung mit Farbsensor mit Biegung 57
7.8 RGB-Messung mit Farbsensor mit Biegung 58
7.9 Infrarot-Messung mit Ringoszillator mit Plastikfaser 59
7.10 RGB-Messung mit Ringoszillator mit Plastikfaser 60
7.11 Infrarot-Messung mit Ringoszillator mit Glasfaser 61
7.12 RGB-Messung mit Ringoszillator mit Glasfaser 62
7.13 Infrarot-Messung mit Ringoszillator mit Glasfaser 63
7.14 RGB-Messung mit Ringoszillator mit Glasfaser 64
7.15 Infrarot-Messung mit Farbsensor, Glasfaser und Umgebungslicht 65
7.16 RGB-Messung mit Farbsensor, Glasfaser und Umgebungslicht 66
7.17 Infrarot-Messung mit Ringoszillator, Glasfaser und Umgebungslicht 67
7.18 RGB-Messung mit Ringoszillator, Glasfaser und Umgebungslicht 68

A.1 Schaltplan der Ringoszillatorplatine 80
A.2 Schaltplan der Farbsensorplatine 81
A.3 Platinenlayout der Farbsensorplatine von oben 82
A.4 Platinenlayout der Farbsensorplatine von unten 83

Tabellenverzeichnis

4.1 Testplan für die Hardware 32

5.1 Wichtige Register des Farbsensors 40
5.2 Testplan für die Software der Farbsensorplatine 42
5.3 Gemessene Empfindlichkeit des Farbsensors 43
5.4 Testplan für die Software der Ringoszillatorplatine 44

6.1 Buchstaben zur Auswahl der LED 46
6.2 Buchstaben zur Auswahl der LED für die switch-case-Bedingung 47
6.3 Testplan für die Software der Matlab-Oberfläche 49

7.1 Anzahl an Messungen, je nach Lichtwellenleiter und Sensor 51
7.2 Übersicht der ermittelten Bitauflösungen für die Plastikfaser 69
7.3 Übersicht der ermittelten Bitauflösungen für die Glasfaser 69

1 Einleitung

Die Menschheit muss ihre Energieversorgung von fossilen Rohstoffen auf regenerative Energie umstellen, um den Klimawandel zu verlangsamen oder bestenfalls zu stoppen. Im Bereich der Mobilität wird derzeit vermehrt auf einen Einsatz elektrischer Energie gesetzt, da sich elektrische Energie verlustarm in andere Energieformen wandeln lässt. Die temporäre Speicherung der Energie erfolgt dabei mit Lithium-Ionen-Batterien, welche aktuell die leistungsfähigsten Batterien sind, die kommerziell zur Verfügung stehen. Für die Nutzung der Lithium-Ionen Technologie in der Elektromobilität müssen die Akkumulatoren überwacht werden. Die Zustandsüberwachung übernimmt dabei verschiedene Aufgaben, z.B. muss der Ladezustand bestimmt werden, die Alterung überwacht werden und die verbleibende Reichweite geschätzt werden. Mit steigenden Anforderungen werden diese Aufgaben schwieriger, wobei vor allem der Sicherheitsaspekt im Vordergrund steht. Damit das Batteriemanagementsystem eine sichere Betriebsführung gewährleisten kann, werden Parameter des Akkumulators vermessen. Für den Ladezustand werden u.a. der Strom und die Spannung benötigt. Für die Sicherheit wird zusätzlich noch die Temperatur bestimmt, um bei Überhitzung den Akkumulator abzuschalten. Da die Überwachung der Akkumulatoren immer wichtiger wird, werden auch neuartige oder auch bereits bekannte Messmethoden für die Nutzung im Automobil untersucht. Diese sind zum Beispiel die Verwendung einer Referenzelektrode, die elektrochemische Impedanzspektroskopie oder die faseroptische Sensorik.
Die faseroptische Sensorik nutzt dabei die beim Laden und Entladen auftretenden optischen Effekte in den Batterieelektroden innerhalb der Lithium-Ionen-Batterien aus. Die Anode (typischerweise Graphit) erfährt eine Farbänderung. Bei der Kathode (z.B. Lithiumeisenphosphat) konnte eine Helligkeitsänderung in Laborversuchen festgestellt werden. Diese Veränderungen können mit einem Lichtwellenleiter durch die Messung der Lichttransmission mit einem Sensor erfasst werden.
Dieses Messverfahren wurde bisher nur mit kostenintensiven Labormessgeräten realisiert. Als Lichtquelle wird dabei eine Halogenlampe mit breitem Lichtspektrum verwendet. Als Sensor dient ein Spektrometer, das die Intensität von einzelnen Wellenlängen des Lichtes

erfassen kann.
Ziel dieser Arbeit ist es einen Prototypen mit preiswerter Elektronik zu entwickeln, um zu zeigen, dass faseroptische Sensoren zukünftig als Messmethode im Automobil Einsatz finden könnten. Die messtechnische Erfassung des optischen Signals sowie die Steuerung mittels Mikrocontroller stehen dabei im Vordergrund.

2 Einarbeitung und Recherche

2.1 Konzepte/Erläuterung für Batteriemanagement und dessen Forschungsbedarf

Die Elektromobilität wird in der nahen Zukunft, die größte Akkumulatorindustrie die es geben wird. Hierfür müssen neue Messverfahren entwickelt werden, um den Ladestand von Akkumulatoren genauer zu bestimmen. Die genaue Bestimmung des Ladestandes ist wichtig, da sich hieraus die noch zu erreichende Entfernung berechnen lässt. Die momentan zur Verfügung stehenden Messmethoden benötigen Leistung aus den zu messenden Akkumulatorzellen. Dies soll durch eine neue Messmethode, die nicht auf elektrische Messgrößen zurückgreift, realisiert werden. Vorteilhaft wäre es wenn beide Messmethoden verwendet werden können, um einen höheren Datensatz zur Bestimmung des Ladestandes zu erhalten.
Es gibt mehrere Probleme bei Akkumulatoren, besonders bei den leistungsfähigen Lithium-Ionen-Akkumulatoren sind diese zu vermeiden. Zum einen gibt es Probleme mit Über- bzw. Unterspannung. Bei der Überspannung kann ein Lithium-Ionen-Akkumulatoren einen Benutzer gefährden. Er könnte sich ausdehnen und anfangen zu brennen. Bei einer Unterspannung wird die maximale Ladungsdichte herabgesetzt. Der Akkumulator kann nicht mehr die angegebene Leistung speichern. Ein weiteres Problem ist das Relaxationsverhalten von Akkumulatoren. Beim Laden bzw. Entladen von Akkumulatoren werden nicht alle Zellen gleich belastet. Wird dann der Akkumulator in Ruhe gelassen, lädt die Zelle mit einer hohen Spannung die Zelle mit einer niedrigen Spannung, bis alle Zellen die gleiche Spannung haben. Aus diesem Grund kann die Ruhespannung eines Akkumulators erst nach einigen Minuten oder mehreren Stunden gemessen werden.
Beim Schnellladen von Akkumulatoren dürfen die Zellen nicht zu stark belastet werden, da eines der oben erwähnten Probleme auftreten kann. Hier würde eine neue Messmethode eine weitere Sicherheitsebene dem System hinzufügen. Es wäre so möglich die Notabschaltung des Systems zu verbessern. Bei den bisherigen Messmethoden könnten

hier möglicherweise inkorrekte Werte liefern, was eine längere Ladezeit als nötig ergeben kann.
Die bislang genutzte Messmethode des Coulomb Counting integriert den Strom über die Zeit und errechnet daraus den Ladezustand. Es kann hier zu Integrationsfehlern kommen, die sich nur durch eine Rekalibrierung vermeiden lassen. Bei der Rekalibrierung wird der Akkumulator vom geladenen Zustand komplett entladen. Diese muss in zeitlichen Abständen wiederholt werden. Alternativ können Verhältnisse aus Spannung und Strom Regeln bestimmt werden, wie der Ladezustand ist.
Durch eine genauere Messmethode können die Akkumulatoren besser ausgenutzt werden, da hier kein Integrationsfehler vorliegen kann. Die bessere Ausnutzung von Akkumulatoren würde zu einer Reichweitenvelängerung bei der Elektromobilität führen.
In der Zukunft können neue Messmethoden das Lebensalter einer Zelle bestimmen. Das wäre möglich, da die Zelle sich physisch mit der Zeit ändert. Nach vielen Lade-Entlade-Zyklen können die Zellen an der Kathode irgendwann Dornen bilden, die zum Kurzschluss der Zelle führen können.

2.2 Einarbeitung in die Grundlagen faseroptischer Sensoren

Eigenschaften von Lichtwellenleitern

Aufbau eines Lichtwellenleiters

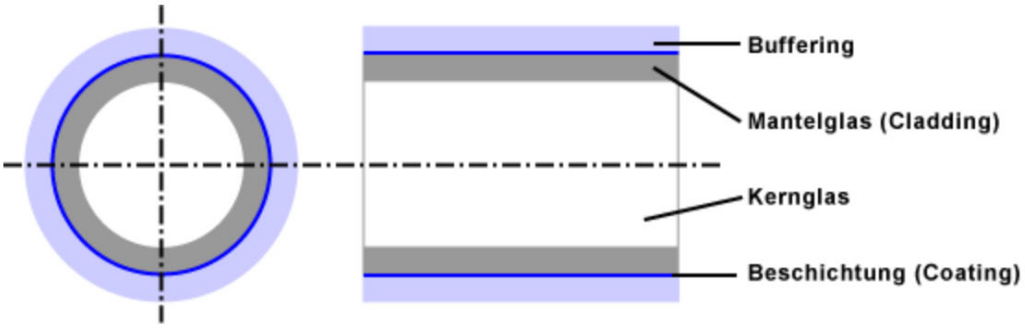


Abbildung 2.1: Aufbau eines Lichtwellenleiters[1]

Ein Lichtwellenleiter ist hat zwei Schichten. Diese Schichten sind der Kern und der Mantel. Durch den Kern wird das Licht geleitet. Damit das Licht nicht aus dem Kern austritt hat dieser einen höheren Brechungsindex als der Mantel. Das Licht wird an der Übergangsstelle von Kern zu Mantel reflektiert. Diese Reflexion wird Totalreflexion genannt. Es kann hierdurch das Licht nahezu verlustfrei um Ecken geleitet werden. Als äußerste Schichten werden zwei Kunststoffbeschichtungen auf den Lichtwellenleiter aufgetragen. Diese Schichten sollen den Lichtwellenleiter vor Beschädigungen und Umwelteinflüssen schützen. Dieser Aufbau gilt für Glas-, Quarz- und Plastikfasern.[1]

Arten von Lichtwellenleitern

Es gibt zwei Arten von Lichtwellenleitern Multimode- und Monomodefasern. Durch Multimodefasern können mehrere Lichtwellen gleichzeitig geschickt werden. Diese teilen sich in zwei Arten auf, die Multimodefasern mit Stufenindexprofil und die Multimodefaseren mit Gradientenindexprofil.[1]

Abbildung 2.2: Lichtverlauf in einer Multimodefaser mit Stufenindexprofil[1]

Bei den Multimodefasern mit Stufenindexprofil wird das Signal hart von dem Mantel in den Kern reflektiert. Dies ist auf hohen Unterschied in der Brechzahl von Kern und Mantel zurück zu führen. Es wird hierbei das Signal leicht verzerrt.[1]

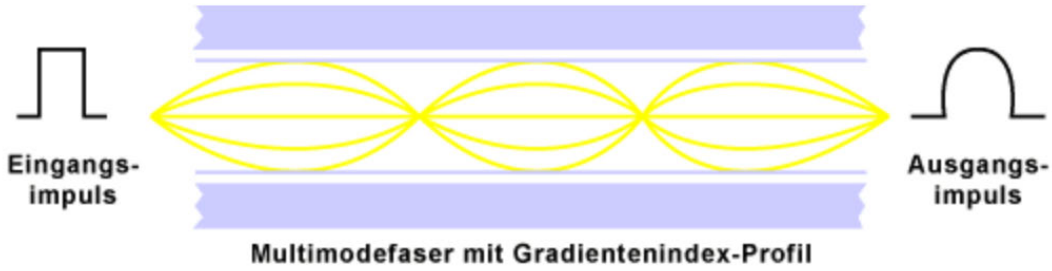


Abbildung 2.3: Lichtverlauf in einer Multimodefaser mit Gradientenindexprofil[1]

Die Multimodefasern mit Gradientenindexprofil reflektieren das Signal weich. Dies liegt an der parabelförmig abnehmenden Brechzahl vom Kern zum Mantel. Hierdurch wird eine Verzerrung des Signals minimiert. [1]

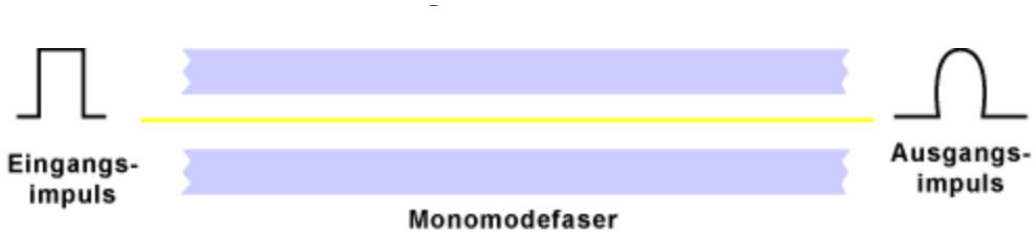


Abbildung 2.4: Lichtverlauf in einer Monomodefaser[1]

Durch die Monomodefasern werden die Lichtwellen gerade hindurchgeleitet. Der Kerndurchmesser eine Monomodefaser ist so klein, dass sich nur ein Modus in der Faser ausbreiten kann. Für diese Art von Faser werden teure Laser benötigt, was die Kosten für diese Technik in die Höhe treibt.[1]

Brechungsindex eines Lichtwellenleiters

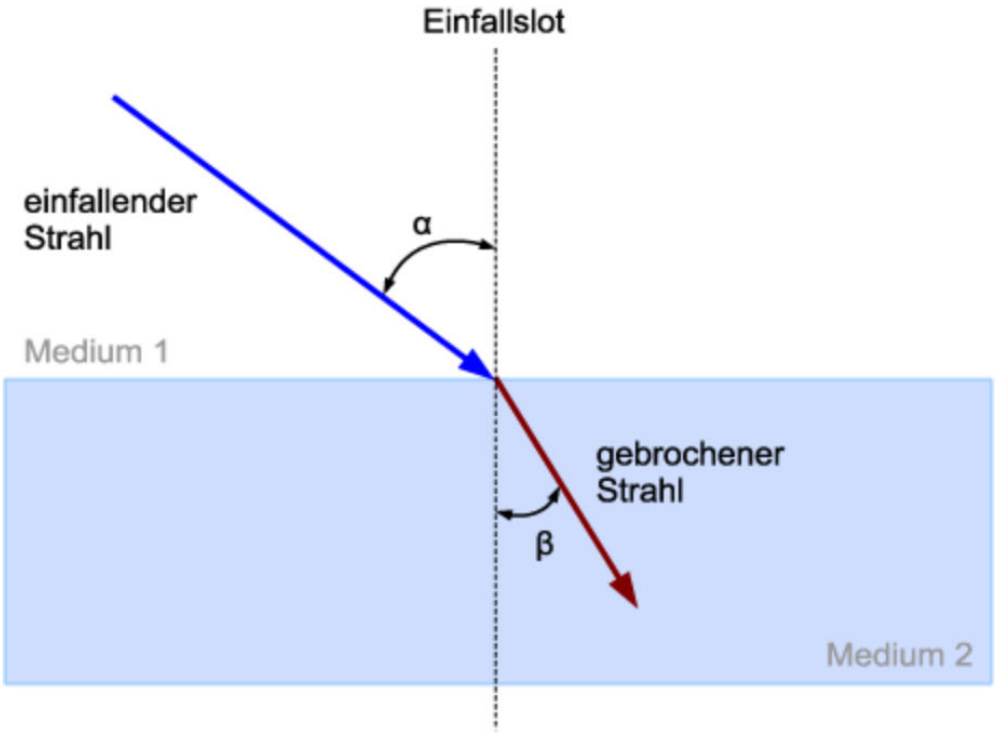


Abbildung 2.5: Visualisierung des Snelliusschen Brechungsgesetzes[2]

Nach dem Snelliusschen Brechungsgesetz wird der Brechungsindex beschrieben als das Verhältnis aus der Lichtgeschwindigkeit im Medium zu der Vakuumlichtgeschwindigkeit nach. Die wird in der Formel $\frac{sin(\alpha)}{sin(\beta)} = n = \frac{c1}{c2}$ [3] ausgedrückt. $sin(\alpha)$ beschreibt den Winkel des einfallenden Lichtstrahls und $sin(\beta)$ beschreibt den Winkel des ausfallenden Lichtstrahles jeweils zum Lot. Der Brechungsindex wird mit dem Buchstaben n in der Formel angegeben. $c1$ gibt die Lichtgeschwindigkeit im Vakuum an und $c2$ gibt die Lichtgeschwindigkeit im Medium an.[2]

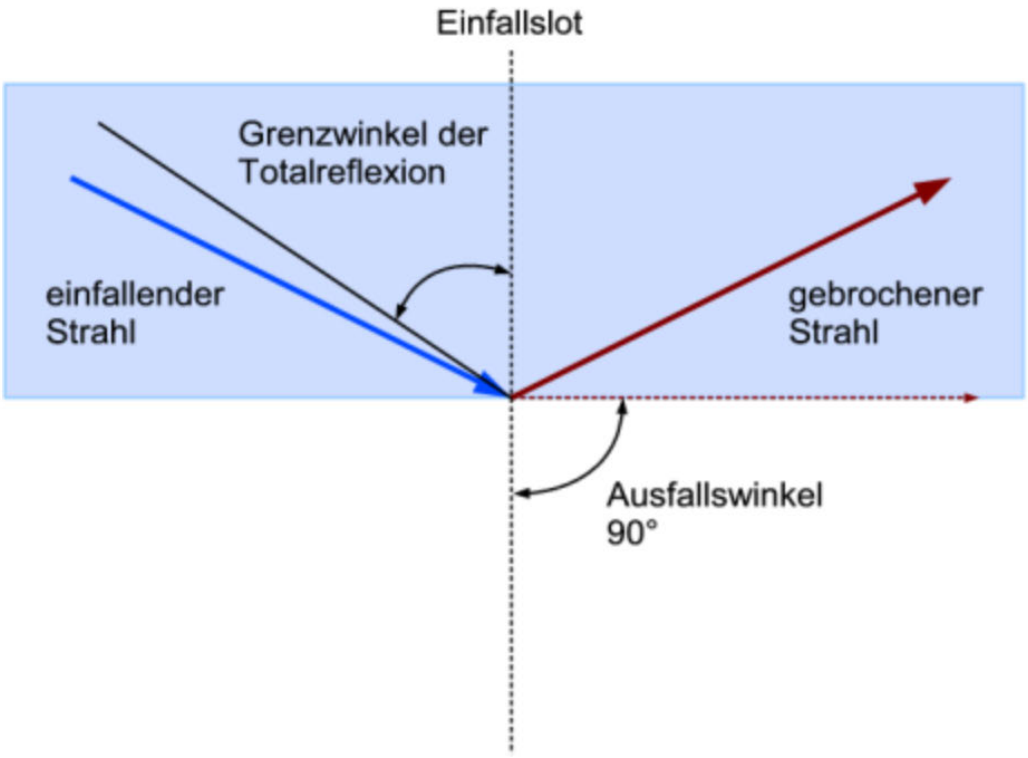


Abbildung 2.6: Visualisierung des Snelliusschen Brechungsgesetzes: Spezialfall Totalreflexion[4]

Bei dem Spezialfall der Totalreflexion ist der Einfallswinkel des einfallenden Strahles größer als der Grenzwinkel der Totalreflexion zum Lot. Der gebrochene Strahl wird an dem anderen Medium reflektiert und der Ausfallswinkel ist größer als 90° zum Lot.[4]

Verluste eines Lichtwellenleiters

Es treten unterschiedlich starke Verluste bei einem Lichtwellenleiter auf. Die relevanten Verluste für diese Arbeit sind Veränderung des Biegeradius und Reduzierung des Mantels. Wird ein bestimmter Biegeradius bei einem Lichtwellenleiter unterschritten findet keine Totalreflexion statt und es tritt Licht aus dem Mantel aus. Wird der Mantel reduziert wird der Brechindex verändert. Es wird so auch der Ausfallswinkel verändert. Dieses Verhalten ist für einen faseroptischen Sensor unter Umständen gewünscht. Durch die Lichtauskopplung aus dem Mantel kann eine Wechselwirkung mit dem Elektroden-

material stattfinden. Das austretende Licht wird je nach Farbänderung unterschiedlich stark in den Lichtwellenleiter zurück reflektiert.[5]

Wichtige Eigenschaften einer Batteriezelle

Aufbau einer Batteriezelle

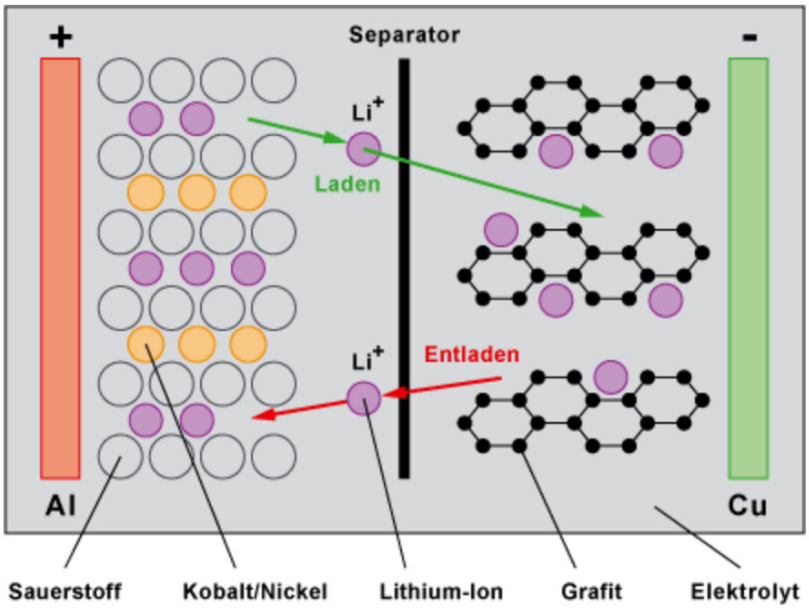


Abbildung 2.7: Aufbau einer Lithium-Ionen Batteriezelle[6]

Es gibt viele unterschiedliche Varianten von Batteriezellen, diese bestehen jedoch alle aus vier Bestandteilen, diese sind eine Anode, eine Kathode, das Elektrolyt und ein Separator. Alles diese Bestandteile können aus den verschiedensten Materialien gefertigt werden. Es wird sich in dieser Arbeit auf einen Teil der Lithium-Ionen-Batterien bezogen. Die Anode besteht meistens aus einer Kupferfolie, auf die ein Aktivmaterial aufgetragen ist. Dieses Aktivmaterial besteht meistens aus Graphit. Für die Kathoden sind gängige Materialien Lithium-Eisen-Phosphat und Lithium-Nickel-Cobalt-Mangan. Der Elektrolyt darf nicht auf einer Wasserbasis bestehen, da Lithium stark mit Wasser reagiert. Es werden meistens Lithiumsalze auf Basis organischer Lösemittel verwendet. Der Separator einer Zelle hat die Aufgabe die Anode und die Kathode räumlich zu trennen, um so Kurzschlüsse zu verhindern. Damit Ionen durch den Separator wandern können muss dieser porös sein.

Speziell für Lithium-Ionen-Batterien können die sogenannten Shutdown-Separatoren verwendet werden. Diese Separatoren schließen ihre Poren, wenn die Zelltemperatur zu stark steigt. Es wird so der Ionenaustausch vermindert und die Zelltemperatur sinkt.[7, 8]

Funktionsweise einer Lithium-Ionen-Batteriezelle

Die Aktivmaterialien an den Elektroden können mit Lithium-Ionen sogenannte Interkalationsverbindungen eingehen. Dies bedeutet es werden Lithium-Ionen als positiv geladene Ionen in die Aktivmaterialien eingelagert.
Beim Entladen wandern die Ionen von der Anode in den Elektrolyten. Von diesem wandern sie durch den Separator zur Kathode. Hier lagern sich die Ionen in dem Aktivmaterial an. Durch die positive Ladung des Lithium-Ions muss die Kathode für einen Ladungsausgleich ein Elektron aufnehmen. Die Elektronen kommen von der Anode und gelangen durch den angeschlossenen Stromkreis zur Kathode. Durch die freigesetzten Elektronen oxidiert die Anode. Reicht die Anzahl an Elektronen nicht aus um einen Ladungsausgleich zu erzeugen werden Lithium-Ionen im Elektrolyt freigesetzt. Beim Laden der Zelle dreht sich der Prozess und es Lagern sich Lithium-Ionen wieder in der Anode an.[7]

Wichtige optische Effekte einer Lithium-Ionen-Batteriezelle

Beim Laden und entladen einer Lithium-Ionen-Batteriezelle treten optische Effekte auf. Diese Effekte sind an der Anode und Kathode sichtbar.

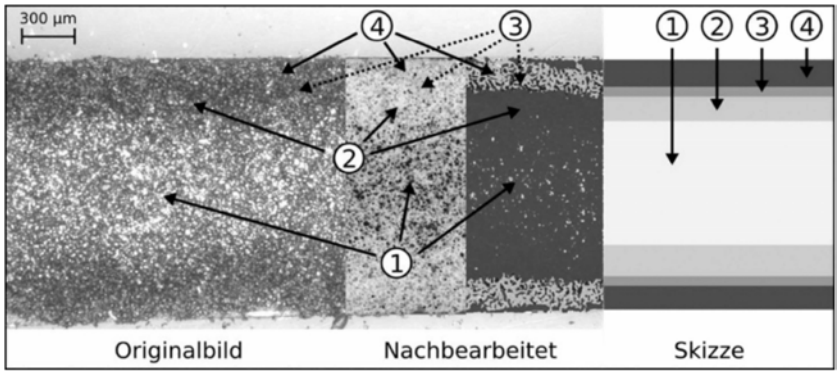


Abbildung 2.8: Optischer Effekt bei der Anoden einer Lithium-Ionen-Batteriezelle[9]

Durch die Anlagerung von Lithium-Ionen in der Anode ändert die Graphitschicht seine Färbung. Die Stärke dieses Effekts korreliert mit der Anzahl an Lithium-Ionen in der Graphitschicht.[9]

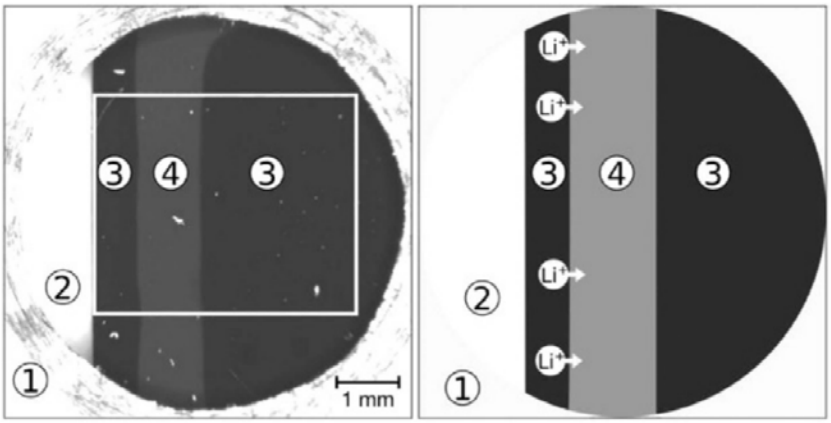


Abbildung 2.9: Optischer Effekt bei Kathodenmaterialen einer Lithium-Ionen-Batteriezelle[9]

Bei den Kathodenmaterialen Lithium-Eisen-Phosphat und Lithium-Nickel-Cobalt-Mangan sind ebenfalls optische Effekte sichtbar. Diese erfahren eine Helligkeitsänderung. Um aus diesen Änderungen der Zellen verwendbare Daten zu erhalten müssen Referenzwerte gebildet werden. Dies geschieht am besten bei der Produktion der Zellen, da hier einige Ladezyklen durchgeführt werden müssen. Diese Ladezyklen schließen einige elektrochemische Vorgänge ab, bevor die Zellen verwendet werden können. Es sollten zu Referenzwertbildung mehrere Wellenlängen an Licht verwendet werden, um einen höheren Datensatz zur Auswertung zu erhalten. Genauere Informationen sind in dem Buch Automobil-Sensorik 3 unter Kapitel 5 von Herrn Rittweger, Herrn Modrzynski und Herrn Roscher zu erhalten.[9]

2.3 Recherche von vergleichbaren Vorarbeiten an der HAW von Herrn Klockmann und Herrn Nasimzada

Erkenntnisse aus der Bachelorarbeit von Herrn Nasimzada

In der Arbeit von Herrn Nasimzada wurde mit einem Spektrometer überprüft , ob die Transmissionsänderung an die Brechungsindexänderung zu einem Elektrolyt gekoppelt

ist. Es konnte ein Veränderung in der Transmissionsleistung, bei Änderung der Elektrolytkonzentration festgestellt werden. Aus dieser Untersuchung wurde mit einer roten Leuchtdiode und einem Sensor ein Prototyp entwickelt. Auf dem Prototypen sind zwei Leuchtdioden verbaut. Eine rote und eine infrarote Leuchtdiode. Es wurden keine weiteren Untersuchungen mit der infraroten Leuchtdiode durchgeführt. Mit einem mitentwickelten Matlab-Programm konnten die aufgezeichneten Daten von dem Prototypen auf einen Computer übertragen und gespeichert werden.[10]

Erkenntnisse aus der Bachelorarbeit von Herrn Klockmann

Die Arbeit von Herrn Klockmann wurden unter Nutzung eines Spektrometers die Absorptionseigenschaften von verschiedenen Materialien untersucht. Es wurde eine Methylenblaulösung verwendet, um die Absorptionseigenschaften festzustellen. Eine Verdünnung der Lösung hat eine Veränderung der Absorptionseigenschaften gezeigt. In weiteren Versuchen wurden die Reflexionseigenschaften von Lithium-Ionen-Batteriezellen geprüft. Da die gewonnen Ergebnisse nicht aussagekräftig genug waren, welche Wellenlänge des Lichtes zum Messen genutzt werde musste, wurde ein flexibler Prototyp entwickelt. Dieser Prototyp hat die Möglichkeit durch eine große Auswahl an Leuchtdioden, die am besten geeigneten zu identifizieren. Die empfangene Lichtleistung wurde mit einem Farbsensor ermittelt. Dieser wandelt das empfangene Licht in eine Frequenz, die gemessen werden kann. Mit diesem Prototypen wurden die Messungen mit der Methylenblaulösung wiederholt und die Funktionstüchtigkeit festgestellt. Die erhaltenen Daten wurden mit einem Matlab-Programm aufgenommen und abgespeichert.[11]

2.4 Inbetriebnahme der Entwicklungsumgebung für ARM-Mikrocontroller, bspw. DAvE (Infineon)

Wegen der Kooperation mit Infineon ist der zu verwendende Mikrocontroller vorgegeben. Dieses war der XMC1100, welcher aufgrund seiner Größe, Verbrauch und Kosten für das Projekt ausreicht. Dieser Mikrocontroller ist als ein Evalutationsboard verfügbar, dass den Vorteil einer Debug-Schnittstelle mit zusätzlicher Stromversorgung hat. Dieses Evalutationsboard hat den Namen *XMC2Go* und verwendet als XMC1100 den ARM Cortex M0 in der 32 Bit Ausführung.
Die nötige Entwicklungsumgeben kommt ebenfalls von Infineon und heißt DAvE. Diese

Entwicklungsumgebung basiert auf einer Eclips-Oberfläche, welche ursprünglich für die Programmiersprache Java entwickelt wurde. Durch die Erweiterbarkeit von Eclips, wird es auch für andere Entwicklungsaufgaben genutzt. Als hardwarenahe Programmiersprache wird C verwendet. Es wird den Programmierern aber etwas einfacher gemacht, da es möglich ist *Apps* hinzuzufügen. Diese *Apps* beinhalten Bibliotheken, die anwendungsspezifische Funktionen enthalten. Als Beispiel kann die *App* $DGIGTAL\_IO$ hinzugefügt werden. Diese App enthält die Funktionen zum Setzen oder Lesen eines GPIO-Pins. Es müssen so nur die benötigten Apps dem Projekt hinzugefügt und die benötigten Funktion in die *main.c* kopiert werden.
Um den in C geschriebenen Programmcode für einen Mikrocontroller verständlich zu machen wird von ARM die $CommonMicrocontrollerSoftwareInterfaceStandard(CMSIS)$ bereitgestellt. CMSIS ist eine Hardwareabstraktionsschicht für ARM Cortex Mikrocontroller. Sie stellt allen eine Basisauswahl an Funktionen für die Mikrocontroller und einen breiten Gerätesupport zur Verfügung. Jeder Hersteller von ARM Cortex Mikrocontrollern passt die CMSIS an und fügt an die eigenen Bedürfnisse zusätzliche Funktionen hinzu.[12]
Zuerst wird in der Entwicklungsumgebung ein neues Projekt angelegt und diesem ein Name gegeben. Als Projekttyp wird das $DAVECEProject$ ausgewählt. Es muss nun der Mikrocontroller ausgewählt, der programmiert werden soll. Hier wird der Listeneintrag $XMC2Go$ ausgewählt. Zum Abschluss wird in dem Fenster mit einem Klick auf den Finish Kopf das Projekt erstellt. Um das Evaluationsboard komplett nutzten zu können muss in der App $CPU\_CTRL\_XMC1\_0$ das Debug Interface von $SWD0$ auf $SWD1$ umgestellt werden.
Zum Testen ob alles richtig eingerichtet ist wird die App $DGIGTAL\_IO$ dem Projekt hinzugefügt. Auf dem Evaluationsboard sind Leuchtdioden angebracht. Diese sind mit Pins von dem Mikrocontroller verbunden und können über die hinzugefügte App gesteuert werden. Sie werden mit der Funktion $DIGITAL_IO_SetOutputHigh$ angeschaltet. Nach einer Pause werden die Leuchtdioden wieder mit der Funktion $DIGITAL_IO_SetOutputLow$ ausgeschaltet. Die Entwicklungsumgebung ist jetzt einsatzbereit.

neon)

2.5 Grundlagenrecherche zu Bauelementen und Schnittstellen

Leuchtdioden

Leuchtdioden sind Halbleiterbauteile, die durch Stromanregung Licht emittieren. Der Halbleiter ist mit der Anode verlötet und über einen dünnen Draht mit der Kathode verbunden. Über die verwendeten Materialien der Halbleiter können die Farben bestimmt werden. Für die Arbeit werden zwei unterschiedliche Leuchtdioden verwendet. Es sind eine Infrarot und RGB-LED. Es gibt verschiedenen Bauformen von Leuchtdioden. Sie sind als bedrahtete Leuchtdioden, SMD-Leuchtdioden, Superflux Leuchtdioden und COB-Leuchtdioden erhältlich.
Bedrahtete Leuchtdioden sind in den Baugrößen 3 mm, 5 mm und 10 mm vorhanden und besitzen zwei Drahtbeinchen als Anschlüsse. Die Farbauswahl ist hier sehr viel reich und sind in einfachen Signalanzeigen und Lichterketten wieder zu finden.
SMD-Leuchtdioden sind nicht durch ihre Drahtbeinchen mit der Platine verbunden, sondern sind direkt auf der Platine verlötet. Sie besitzen eine starke Leuchtkraft und sind Platzsparend. Auch diese Leuchtdioden sind in allen möglichen Farben erhältlich.
Die Superflux Leuchtdioden sind eine Weiterentwicklung der bedrahteten Leuchtdioden. Sie besitze vier Drahtbeinchen als Anschlüsse. Sie haben eine bessere Wärmeableitung und einen größeren Abstrahlwinkel.
COB-Leuchtdioden (Chip on Board) sind ohne eine Verkapselung direkt auf eine wärme leitende Platte geklebt. Über Drähte sind die Gegenpole angeschlossen. Mit vielen Modulen lässt sich eine hohe Lichtintensität erreichen. Die COB-Leuchtdioden sind auch in vielen Farben erhältlich.[13, 14]

Sensoren

Fototransistor

Ein Fototransistor ist in einem ähnlichem Gehäuse verbaut wie die bedrahtete Leuchtdiode. Die Drahtbeinchen sind hier der Kollektor und der Emitter. Die Basis wird in den meisten Fällen nicht angeschlossen. Das empfangene Licht erzeugt einen Basisstrom, der die Kollektor-Emitter-Strecke schaltet. Durch das Halbleitermaterial Silizium liegt die

Empfindlichkeit eines Fototransistors bei einer Wellenlänge von ca. 850 nm. Diese Wellenlänge liegt im nahen Infrarotbereich. Zu den kürzeren Wellenlängen nimmt die Empfindlichkeit ab und endet bei höheren Wellenlängen bei ca. 1100 nm. Durch die Verstärker Eigenschaften ist ein Fototransistor empfindlicher gegenüber von Lichtveränderungen als eine Fotodiode.[15, 16]

Farbsensor

Es gibt unterschiedliche Farbsensoren, einige detektieren Licht über eine Frequenzänderung und andere integrieren die aufgenommene Strahlungsleistung. Sie bestehen aus zusammengeschalteten Fotodioden, dies ermöglicht eine richtungsunabhängige Erfassung des Lichtes. Hierfür sind die Sensoren in den meisten Fällen als SMD-Bauteile ausgelegt, so kann das Licht großflächig auf die Fotodioden treffen. Die Farbsensoren sind in allen möglichen Varianten erhältlich, was die Empfindlichkeit auf bestimmte Wellenlängen betrifft. So ist es möglich Sensoren zu bekommen, die farbiges Licht einzeln bestimmen können, als auch weißes und Infrarotes Licht.
Farbsensoren sind programmierbar, so ist zum Beispiel die Integrationszeit und die Verstärkung auf die Bedürfnisse einstellbar. Deshalb sind Farbsensoren in vielen Einsatzbereichen vertreten.

Schnittstellen

Es gibt verschiedene Schnittstellen die eingesetzt werden können. Sie bestehen mindestens aus einem Master und einem Slave. Der Master ist zum Beispiel ein Mikrocontroller der Befehle oder Anfragen an einen Slave zum Beispiel einen Sensor sendet. Der Slave setzt den Befehl um oder sendet die angefragten Daten an den Master zurück. Die Schnittstellen unterscheiden sich hauptsächlich in der Art der Übertragung der Daten, es gibt die synchrone und asynchrone Übertragung der Daten. Bei der synchronen Übertragung wartet der Master solange bis der Slave die Daten oder eine Bestätigung zurück gesendet hat. Des Weiteren hat eine synchrone Übertragung eine Taktleitung, über die der Bustakt durch den Master für den Slave bereitgestellt wird. Bei einer asynchronen Übertragung wartet der Master nicht auf die Antwort des Slaves. Es ist auch keine Taktleitung vorhanden, die einen Bustakt zur Verfügung stellt. Damit die Bits aus den erhaltenen Daten interpretiert werden können, müssen bei Sender und Empfänger die Übertragungsgeschwindigkeit gleich eingestellt sein. Aus dem vorgeschriebenen Datenrahmen und der

eingestellten Übertragungsgeschwindigkeit können dann das Start- und Stoppbit und die Datenbits bestimmt werden.

UART

Die UART-Schnittstelle ist eine serielle und asynchrone Schnittstelle. Sie hat einen fest eingestellten Datenrahmen, der für beide Kommunikationspartner gleich eingestellt werden muss. der Datenrahmen besteht aus:

- einem Startbit
- fünf bis neun Datenbits
- einem optionalem Paritätsbit
- einem oder zwei Stoppbits

Die Schnittstelle benötigt zwei Anschlüsse. Der Anschluss Tx (Transmit Data) sendet die Daten. Der zweite Anschluss ist der Rx(Receive Data) und empfängt die Daten. Eine gemeinsame Masseverbindung muss auch hergestellt werden, da es im schlimmsten Fall zu Zerstörung des Mikrocontrollers kommen kann. Die Datenübertragungsrate kann von 50 Bits/s bis 3.000.000 Bits/s eingestellt werden. Dies Schnittstelle wird oft zur Kommunikation zwischen einem Mikrocontroller und Computer verwendet. Dies geschieht über ein USB-Kabel. Hierfür werden die UART-Signale in RS232-Signale übersetzt. Die RS232-Signale können dann in den USB-Standard übertragen und vom Computer verarbeitet werden.[17]

I²C

Die I²C-Schnittstelle ist eine serielle und synchrone Schnittstelle. An eine I²C-Schnittstelle können bis zu 127 Slaves von einem Master gesteuert werden. Die zulässige Gesamtkapazität des I²C-Busses beträgt 400 pF und beschränkt die Anzahl an möglichen Slaves auf 20. Alle Bussteilnehmer sind an zwei Leitungen angeschlossen. Die Datenleitung wird SDA genannt und überträgt den Datenrahmen an den Adresszugehörigem Teilnehmer. Damit der Teilnehmer die Daten verstehen kann wird vom Master ein Bus-Clock über die Leitung SCL bereitgestellt. Die beiden Leitungen müssen über einen Pull-Up Widerstand auf High gesetzt werden.

Die der Übertragung werden insgesamt 20 Bits übertragen. In der Abbildung 2.10 ist zu sehen welche Bits übertragen werden und sie zu interpretieren sind.[18]

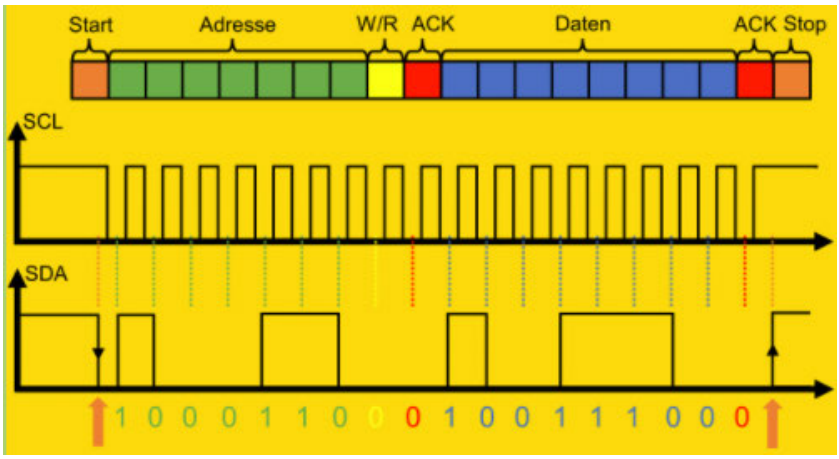


Abbildung 2.10: Reihenfolge des Datenrahmens bei der Übertragung und Interpretation der Bits[18]

Die einstellbare Übertragungsgeschwindigkeit liegt bei dem I²C-Bus liegen zwischen 100 kHz und 400 kHz. Diese Übertragungsgeschwindigkeiten sind ausreichend für eine Übertragung zwischen Bauteilen, die auf derselben Platine verlötet sind.[18]

3 Konzeption

3.1 Spezifikation der Anforderungen

Die folgenden Anforderungen müssen erfüllt werden, um einen funktionsfähigen Prototypen zu erstellen. Zum einen muss die Intensität eines Lichtsignales gemessen werden. Diese Lichtsignale sollen in ihrer Intensität schaltbar sein sowie eine Farbauswahl ermöglichen. Als Farben sollen Rot, Grün, Blau und Infrarot zur Verfügung stehen. Die Leuchtdioden sollen mit einer festen Spannung und einem hohen Strom versorgt werden, um viel Lichtleistung zu erzeugen. Deswegen wird ein LED-Treiber als LED-Versorgung zum Einsatz kommen. Als feste Verbindung zu dem Lichtwellenleiter muss ein Adapter konstruiert werden. Das Messsystem soll an einen Lade-Entladezyklus einer Batteriezelle anpassbar sein. Hierfür sind verschiedene Messzeiten nötig, dafür müssen eine LED- und eine Zeitsteuerung realisiert werden. Des Weiteren muss eine Steuerung des Integrationswertes erfolgen, um die Veränderung der gemessenen Lichtleistung erfassen zu können. All diese Einstellungen sollen über Matlab gesteuert werden. Die Datenausgabe erfolgt über eine UART-Verbindung zu einem PC mit dem Verarbeitungsprogramm Matlab.
Für die Software gelten die folgenden Anforderungen. Die Daten werden live grafisch angezeigt. Bei einer voreingestellten Anzahl an aufgezeichneten Daten werden diese in einer Datei gespeichert. Die gespeicherten Daten sollen auch aus der Matlab-Umgebung heraus wieder als Plot sichtbar gemacht werden können. Da diese Bachelorarbeit im Rahmen des Projektes LImeS [19] angesiedelt ist, sind vorgegebene Rahmenbedingungen für die Baugröße der Platine, sowie die Lage der Anschlüsse und Befestigungslöcher einzuhalten.

3.2 Konstruktion des Lichtwellenleiter-Adapters

3.2.1 CAD-Konstruktion der Adapter zur Ankopplung der Lichtleitfasern

Das CAD-Design wurde nach der technischen Zeichnung des Anschlusses 905-117-5000 von Amphenol [20] entworfen und ist in Zusammenarbeit mit Tim Overath entstanden. Er hat zwei Varianten entworfen. Eine Variante ist konzipiert für die verwendeten Leuchtdioden bzw. den Phototransistor. Die zweite Variante wurde für den Farbsensor erstellt. In dieser Variante wurde das Höhenmaß verringert. Die Befestigungslöcher wurden für eine Schraube angepasst. Der Anschluss der SMA-Verbindung wurde etwas verkleinert. Hiermit kann die Verbindung als Übergangspassung angesehen werden.

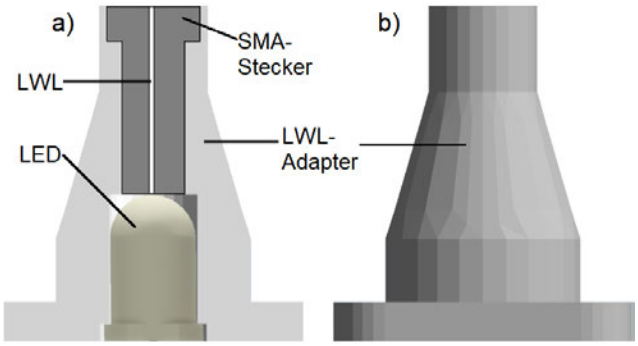


Abbildung 3.1: a): Querschnitt des LWL-Adapters nach Amphenol 905-117-5000 für Leuchtdioden
b): Außenansicht des LWL-Adapters nach Amphenol 905-117-5000 für Leuchtdioden

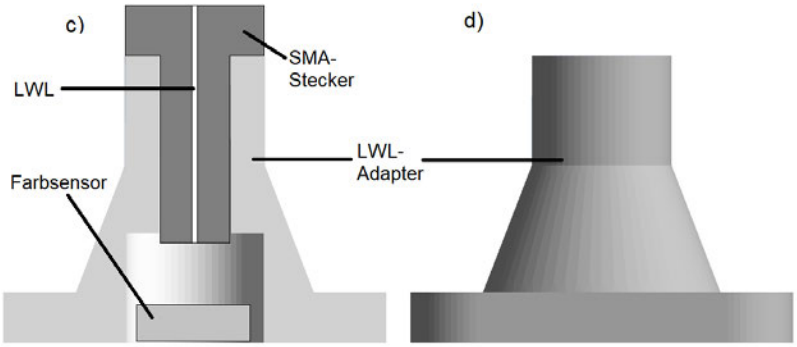


Abbildung 3.2: c): Querschnitt des LWL-Adapters nach Amphenol 905-117-5000 angepasst für den Farbsensor
d): Außenansicht des LWL-Adapters nach Amphenol 905-117-5000 angepasst für den Farbsensor

3.2.2 Fertigung des Adapters mit 3D-Druck

Um einen 3D-Druck herzustellen, müssen in einem Slicer-Programm die Einstellungen für den 3D-Drucker festgelegt werden. Die Einstellungen umfassen u.a.:

- Schichtdicke
- Support-Struktur
- Temperatur des Druckbetts
- Temperatur der Düse
- Düsenöffnung

- Laufgeschwindigkeit
- Infill

Die Schichtdicke gibt die Höhe einer einzelnen Schicht an. Bei Überhängen im Design können die Schichten nicht maßhaltig sein. Hierfür werden Support-Strukturen verwendet, die das Bauteil beim Druck stabilisieren. Mit einer Einstellung können die Menge und das Muster der Support-Strukturen, die eingefügt werden soll, verändert werden. Die aufzubauenden Schichten sind durch Kontakthaftung mit dem Druckbett verbunden. Ohne diese Kontakthaftung kann sich das Bauteil beim Drucken verschieben und wird nicht maßhaltig. Mit einem beheizten Druckbett kann die Kontakthaftung vergrößert werden. Je nach Bauteilgrundfläche muss eine andere Temperatur für das Druckbett eingestellt werden. Um das Material schmelzen zu können, muss die Temperatur der Düse eingestellt werden. Diese Temperatur richtet sich nach dem verwendeten Kunststoff, aus dem das Filament besteht. Die Größe der Düsenöffnung wird benötigt, um die Schichtbreite bestimmen zu können. Die Laufgeschwindigkeit gibt an, wie schnell der 3D-Drucker die einzelnen Schichten druckt. Um Material zu sparen, kann der Anteil des Infills verändert werden. Stützstruckturen innerhalb eines Bauteils werden als Infill bezeichnet und füllen Leerräume aus. Diese Einstellung ist jedoch erst bei großen Bauteilen von Bedeutung. Alle genannten Einstellungen können unterschiedlich sein. Dies hängt zum einen von dem verwendeten 3D-Drucker, zum anderen vom verwendeten Material ab.

3.2.3 Montage, Erprobung und Korrektur des Adapters

Nach dem erfolgreichen Druck werden zuerst die Support-Strukturen entfernt. Des Weiteren werden die Kanten und Löcher entgratet. Bei den Befestigungslöchern hat das Entgraten einen weiteren Vorteil, da es als Führung für die M3-Schrauben dient. Für den SMA-Anschluss wird zusätzlich zu der Übergangspassung ein Außengewinde der Größe 1/4-36 UNS [20] geschnitten. Es kann so eine zweite feste Verbindung hergestellt werden. Das Bauteil wurde aus Polylactide gefertigt, welches auf einer Rolle als Faser geliefert wird. Es wurde mit einem Gewindeschneider der Größe M3 in die Befestigungslöcher geschnitten.
Das Bauteil ließ sich einfach auf der Platine mit vier M3-Schrauben befestigen. Das Loch für die SMA-Verbindung war zu klein und entsprach eher einer Presspassung. Das Außengewinde ließ sich ohne Einschränkungen nutzen. Bei der Erprobung mit

eingeschalteten Leuchtdioden konnte mit bloßem Auge austretendes Licht festgestellt werden. Aus dieser Beobachtung wurde geschlossen, dass Umgebungslicht den Messaufbau stören kann.
Aus den Beobachtungen bei der Erprobung wurden Korrekturen bzw. Änderungen vorgenommen. Das Loch für die SMA-Verbindung wurde durch Schleifen vergrößert. Der SMA-Stecker lässt sich jetzt ohne größeren Kraftaufwand aufstecken. Um einfallendes Umgebungslicht ausschließen zu können, wurde das Bauteil nachträglich mit mehreren Schichten schwarzen Klebebands abgedeckt.

3.3 Systemarchitektur als Übersicht

Es werden zwei unterschiedliche Messkonzepte aufgebaut und getestet. Die Leuchtdiodensteuerung und -versorgung ist bei beiden Messkonzepten ähnlich. Die Sensoreinheiten unterscheiden sich bei den Messkonzepten. Für das eine Messkonzept wird ein Farbsensor verwendet und für das andere ein Fototransistor mit hinter geschalteten Ringoszillator. Beide Messkonzepte werden auf unterschiedlichen Platinen realisiert, sodass in der Arbeit zwischen diesen mit den Bezeichnungen Farbsensorplatine und Ringoszillatorplatine unterschieden wird. In dem Blockdiagramm (Abbildung 3.3) ist vereinfacht

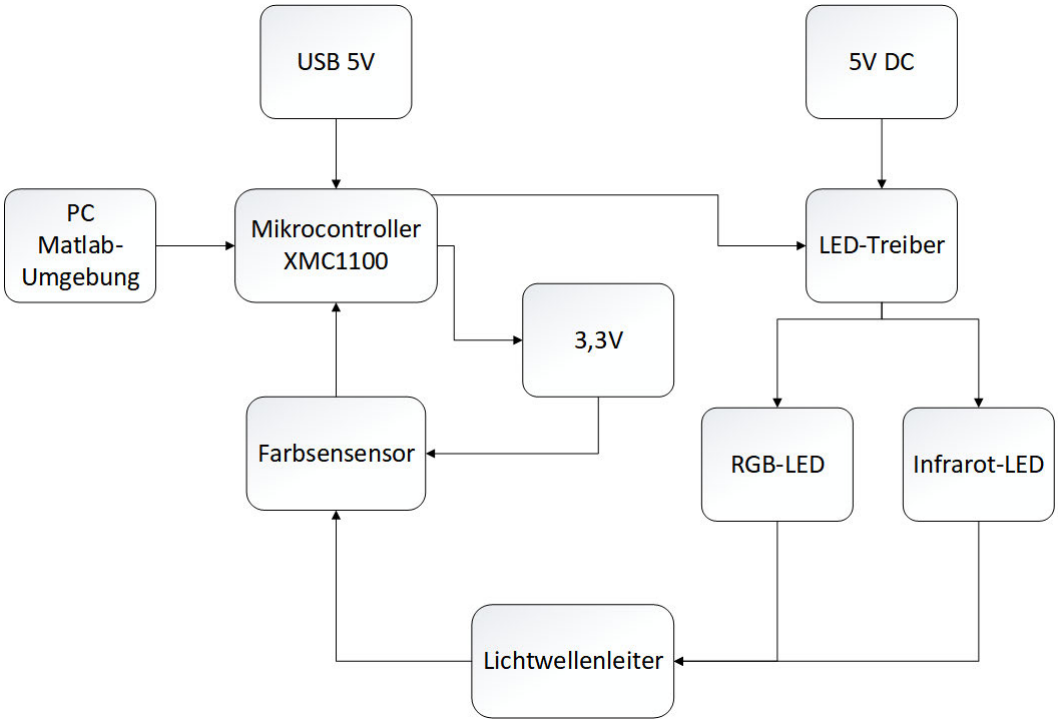


Abbildung 3.3: Blockdiagramm für die Platine mit Farbsensor

die Funktionsweise der Farbsensorplatine aufgezeigt. Der Mikrocontroller wird über eine USB-Verbindung mit 5 V Betriebsspannung versorgt und dient als Steuereinheit der Schaltung. Um den Mikrocontroller nicht zu belasten, werden die Leuchtdioden über einen LED-Treiber durch einem externen Anschluss mit einer 5 V Spannung versorgt. Ein auf dem Mikrocontroller verbauter Linearregler versorgt den Farbsensor und den I^2C-Bus mit 3,3 V. Als Schnittstellen kommen der vorher erwähnte I^2C-Bus für die Verbindung zwischen Farbsensor und Mikrocontroller zum Einsatz. Die Verbindung zum Computer

wird über einen UART-Bus realisiert. Als Lichtquelle dienen zwei Leuchtdioden. Es sind eine RGB-LED und eine Infrarot-LED verbaut, um ein breites Farbspektrum abzudecken. Die Veränderung der Lichtintensität wird mit einem Farbsensor festgestellt. Die Daten des Farbsensors werden von dem Mikrocontroller abgefragt und über die UART-Schnittstelle zum Computer geleitet.

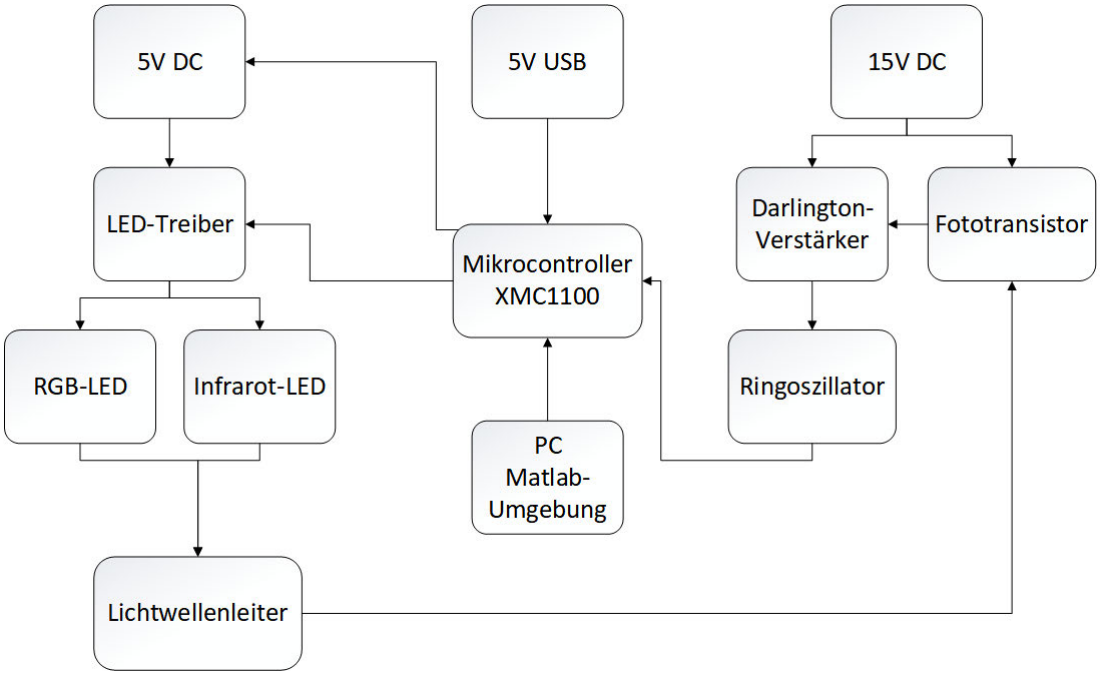


Abbildung 3.4: Blockdiagramm für die Platine mit Ringoszillator

Als zweiter Sensor wurde ein Fototransistor ausgewählt und eine Verstärkerschaltung dafür entwickelt. Die Funktionsweise dieser Platine wird in einem weiteren Blockdiagramm (Abbildung 3.4) aufgezeigt. Die Betriebsspannung des Fototransistors liegt bei 15 V. Da das Signal des Fototransistors nicht ausreicht, wird dieses mit einem Darlington-Verstärker verstärkt. Der Verstärker benötigt auch eine Spannungsversorgung von 15 V. Das Signal vom Verstärker verändert die Messfrequenz des Ringoszillators, welcher auch mit dieser Gleichspannung von 15 V versorgt wird. Die Versorgung des Mikrocontrollers wird über eine USB-Verbindung mit 5 V bereitgestellt. Von dieser werden auch die Leuchtdioden über einen LED-Treiber versorgt. Die Leuchtdioden sind die gleichen wie für die Farbsensorplatine. Hieraus entsteht keine Abweichung durch die Leuchtdioden, zur Steuerung der Leuchtdioden und zur Datenübertragung, die durch eine UART-Verbindung vom Computer zu dem Mikrocontroller hergestellt wird.

Weitere Vorarbeiten zum Fototransistor

Es wurde anfangs eine andere Schaltung für den Fototransistor entwickelt. Diese sollte mit zwei Verstärkern arbeiten. Über einen Rheostat sollte die Verstärkung dynamisch eingestellt werden. Und mit einem ADC des Evaluationsboards die verstärkte Spannung gemessen werden. Die maximale Eingangsspannung des ADC beträgt 3,3 V. Bei einem ADC, der eine Auflösung von zwölf Bit besitzt, entspricht dem LSB die Spannung 800 μV. Um nicht in die Grenzbereiche des ADC zukommen wird der Aussteuerbereich jeweils um 10% an der oberen und unteren Grenze verkleinert. So sind noch 3277 Werte möglich. Dadurch ist der ADC nicht verwendbar, da der Messbereich sehr klein wird.

3.4 Grundsätzlicher Aufbau des Systems mit Schnittstellen

Die I²C-Schnittstelle wurde ausgewählt, da der ausgewählte Farbsensor diese benötigt. Sie verfügt über eine ausreichend hohe Übertragungsgeschwindigkeit und kann bis auf 400 kHz erhöht werden.
Die UART-Schnittstelle war eine Vorgabe aus dem LImeS-Projekt [19]. Über diese Schnittstelle kann mit dem Computer kommuniziert werden. Es ist auch eine Kommunikation zu anderen Mikrocontrollern mit dieser Schnittstelle möglich.
Der Mikrocontroller wird über eine USB-Verbindung mit 5 V versorgt. Über die USB-Verbindung ist gleichzeitig die Programmierung des Mikrocontrollers möglich. Um eine Überlast des Mikrocontrollers zu vermeiden, ist als Spannungsversorgung für leistungstreibene Bauteile eine externe Versorgung angedacht. Der LED-Treiber wird mit 5 V über einen Buchsen Anschluss versorgt. Der Ringoszillator wird ebenfalls über einen Buchsen Anschluss versorgt. Hier wird eine Versorgungsspannung mit 15 V angeschlossen.

3.5 Entwurf der Steuerungs- und Softwarestruktur

Die Steuerung der Sensoren erfolgt über ein Matlab-Skript am PC. Die Steuerungsbefehle sollen über die Matlab-Umgebung an den Mikrocontroller gesendet werden. Das Senden der Befehle erfolgt asynchron. Der Mikrocontroller nimmt die Befehle als Interrupt wahr. Der Einstellungsbefehl führt einen Statuswechsel herbei, weshalb eine Switch-Case-Struktur verwendet wird. In dieser Struktur können die Einstellungen vorgenommen

werden. Die Befehlstruktur sind für beide Platinen bis auf wenige Unterschiede gleich. Diese werden in dem Kapitel 5 genau beschrieben. Außerhalb des Einstellungsstatus läuft das Programm zyklisch zwischen den Zuständen Daten messen und senden in einer Endlosschleife.

3.6 Festlegung von Dateiformaten, Bedienoberfläche und Funktionsumfang

Wie oben erwähnt, werden die Daten nach einer eingestellten Anzahl an Datenpunkten gespeichert. Als Dateiformat wurde csv gewählt. Dieses Format wird von den meisten Programmen erkannt. So wird eine hohe Kompatibilität für die Zukunft erreicht.
Die Bedienoberfläche ist die Matlab-Oberfläche. Hier können die einzelnen Funktion über Sektionen ausgewählt und ausgeführt werden. Mit den Funktionen wird eine Verbindung zum Computer hergestellt, sowie Grafiken aus Live-Daten und gespeicherten Daten erstellt. Auf die genannten Funktionen wird im Kapitel 6 näher eingegangen.

4 Entwicklung der Hardware

4.1 Auswahl geeigneter Bauelemente für die Lichtquelle und den Sensor

Die Auswahl der mit Licht interagierenden Bauteile begann mit der Suche des passenden Farbsensors. Dieser sollte ursprünglich aus der Bachelorthesis „Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern“ von Henrik Klockmann übernommen werden [11]. In dieser Arbeit wurde der Typ TCS3200D von der Firma AMS genutzt [11]. Da nach Recherchen auf der Firmeninternetseite dieser nicht für Neuentwicklungen verwendet werden sollte, wurde eine Alternative gewählt [21]. Diese Alternative ist der Farbsensor vom Typ TCS34001FNM, ebenfalls von der Firma AMS. Beide Sensoren können die Farben Rot, Grün und Blau und Infrarot messen. Bei dem TCS34001FNM kann selektierbar Infrarot bis zu der Wellenlänge 950 nm erfasst werden. Dieser Sensor lässt sich über einen großen Befehlssatz an besondere Bedürfnisse anpassen. Über den I^2C-Bus können die Befehle gesendet und die Daten abgefragt werden. Die Leuchtdioden wurden nach den angegebenen Wellenlängen des Sensors gewählt, die bei 465 nm, 525 nm, 615 nm und 850 nm liegen [22]. Ein weiterer Auswahlfaktor für die Leuchtdioden ist der Ausstrahlwinkel. Dieser sollte möglichst klein sein, um soviel Lichtleistung in den Lichtwellenleiter zu bringen wie möglich. Als RGB-LED wurde die LED des Typs LL 5-8000RGB von Lucky Light ausgewählt. Sie hat Wellenlängen von 470 nm, 525 nm und 626 nm. Der Winkel liegt bei 20° [23]. Als Infrarot-LED wurde die LED des Typs WP7113SF7C von Kingbright ausgewählt. Die Wellenlänge beträgt 850 nm bei einem Ausstrahlwinkel von 20° [24]. Der Fototransistor wurde nach der Wellenlänge der Infrarot-LED ausgesucht. So wurde der Typ SFH 300-3/4 von der Firma Osram gewählt. Dieser liefert einen Fotostrom von mindestens 1000 μA bei einer Versorgungsspannung von 5 V[25].

4.2 Entwurf der Analogbaugruppen und Stromversorgung

Um das XMC2Go-Entwicklungskit von Infineon nicht zu belasten, wird für die Leuchtdioden eine externe Spannungsversorgung genutzt. Diese setzt sich aus einem Lienarregler und einem LED-Treiber zusammen. Als Linearregler kommt der Typ L7805ACV von dem Hersteller STMicroelectronics zur Verwendung. Dieser liefert eine stabile Ausgangsspannung von 5 V. Der maximale Ausgangsstrom liegt bei 1,5 A [26]. Dies reicht um den LED-Treiber mit ausreichend Leistung zu versorgen. Als LED-Treiber sollte ein Leistungstreiber zum Einsatz kommen, der einen hohen statischen Ausgangsstrom bei hoher Ausgangsspannung liefern kann. Leistungstreiber mit Puls-Weiten-Modulation kommen nicht infrage, da die Leuchtdioden statisch betrieben werden sollen. Bei der Puls-Weiten-Modulation wird die LED über ein Rechtecksignal betrieben, um die Helligkeit der LED zu steuern wird die Länge des Pulses gesteuert. So gibt es Zeiten, in denen die LED nicht leuchtet. Es ergibt sich hieraus ein Störverhalten, welches die Messwerte verfälschen könnte. Das Problem ergibt sich aus dem Detailschaltbild (ref anhang ringoszillatorbild), das kommt über einen LWL an den Fototransistor, dessen Ausgangsspannung einen Ringoszillator moduliert. Um keine weiteren Störspektren zu erzeugen, müssen die Leuchtdioden aus einer störarmen Quelle versorgt werden. Aus diesen Voraussetzungen fiel die Wahl auf den Leistungstreiber TC74ACT244FTEL von Toshiba. Dies ist eine allgemeine digitale Schaltung, die als Treiberbaustein Verwendung findet. Dieser verfügt über acht Kanäle, an denen eine LED angeschlossen werden kann. Jeder dieser Kanäle kann einen maximalen Ausgangsstrom von 24 mA bei einer Ausgangsspannung von 4,5 V liefern [27]. Es werden aber nur vier Leuchtdioden verwendet, jeweils zwei Kanäle sind zusammengeschaltet worden. Daraus kann pro LED ein maximaler statischer Strom von 35 mA sichergestellt werden.
Einige Komponenten benötigen eine Spannung von 3,3 V. Dieses sind zum einen der Farbsensor, zum anderen der I^2C-Bus. Die Spannung wird von dem XMC2Go-Entwicklungskit über den Pin 14 den Bauteilen bereitgestellt.
Für die Versorgung des Ringoszillators wird eine externe Spannung von 15 V genutzt.

4.3 Definition der Schnittstellen zu den Controllerboards

Vom Mikrocontroller werden die folgenden Schnittstellen von der Hardware benötigt:

- USB/UART als Schnittstelle für die Entwicklungsumgebung

- USB/UART als Schnittstelle für die Datenübertragung zum Computer
- I^2C für Steuerung und Daten abfrage des Farbsensors
- GPIO-Ports zur LED-Ansteuerung
- GPIO-Port als Eingang des Ringoszillators

Die Funktionen der Schnittstellen befinden sich in den Abschnitten 5 und 6. In der Hardware ergeben sich folgende Rahmenbedingungen:

- USB/UART-Verbindung wird mit einem Mikrocontroller als Übersetzer auf dem Evaluationboard realisiert. Als Schnittstelle wird die Mikro-USB-Buchse genutzt. In der Hardware sind keine weiteren Maßnahmen notwendig.
- Der I^2C-Bus benötigt zwei Signal-Leitungen. Die Signal-Leitung SDA wird am Pin 2.10 angeschlossen. Die Takt-Leitung SCL wird am Pin 2.11 angeschlossen. Beide Leitungen benötigen einen 10 kΩ Widerstand als Pull-Up. Es wird eine gemeinsame Masseleitung GND als Bezugspotential genutzt. In der Controller-Software werden die Pins der Schnittstelle USIC als serieller I^2C-Bus zugewiesen. Die Konfiguration befindet sich in Abschnitt 5.
- Die GPIO-Ports für die LED-Steuerung nutzen die Pins 0.8, 0.9, 0.14 und 0.15 bei der Farbsensorplatine. Für die Ringoszillatorplatine werden die Pins 0.5, 0.6, 0.9, 0.12, 2.10 und 2.11 genutzt. Sie werden als Output konfiguriert. Die Konfiguration befindet sich in Abschnitt 5.
- Ein Pin des GPIO-Ports wurde ausgewählt, um in der Schaltung als Eingang eines flankengesteuerten Interrupt zu dienen. Mit diesem Interrupt werden die Flanken des Signals gezählt, die aus dem nachgeschaltetem Ringoszillator entstehen. Hierzu wurde Pin 2.7 ausgewählt. Die Konfiguration befindet sich in Abschnitt 5.

4.4 Platinenentwurf und Bestückung

Zum Platinenentwurf wurde das Opensource-Programm KiCad genutzt. Mit diesem Programm können Schaltpläne mit Standardbauteilen sehr einfach erstellt werden. Für Bauteile, die nicht in der Bibliothek enthalten sind, können vorhandene Bauteile im Bauteileditor angepasst werden. Zum Beispiel musste der Farbsensor als Bauteil angelegt werden, da dieser nicht vorhanden war. Aus diesem Grund muss auch der Footprint neu

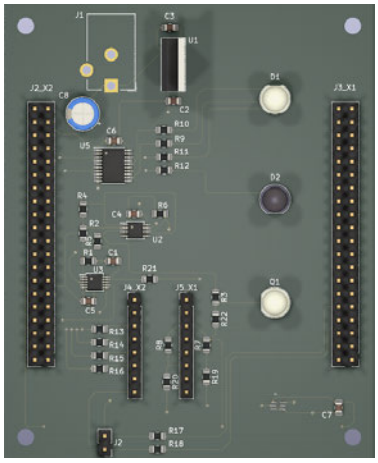


Abbildung 4.1: Platine als 3D-Ansicht in der Draufsicht, generiert mit der Entwicklungssoftware KiCad

erstellt werden. Dies geschieht mit dem Footprinteditor. Es werden auch hier bei vorhandenen Bauteilen Anpassungen durchgeführt. Die Lötpads müssen den Pins der Bauteile zugeordnet werden. Das einzige Bauteil, welches nur als Footprint erstellt wurde, ist der Lichtwellenleiter-Adapter. Dieser Footprint besteht nur aus vier Löchern, die mit den Befestigungslöchern des LWL-Adapters übereinstimmen. Ist der Schaltplan erstellt, öffnet man den PCB Layout Editor. Durch den Befehl „Aktualisiere PCB aus dem Schaltplan“ wird der Schaltplan mit den Verknüpfungen in den PCB-Layout-Editor übertragen. Die Bauteile werden nun platzsparend angeordnet. Es wird hierbei darauf geachtet, dass die Platinenmaße nicht überschritten werden. In die Ecken der Platine werden Befestigungslöcher eingearbeitet. Wenn alles an der richtigen Position ist, werden die Leiterbahnen gezogen. Die Platine wurde mit einer Zweilagenverdrahtung entworfen. So ist es möglich, die Leiterbahnen auf Ober- und Unterseite der Platine zu platzieren. So können sich die Leiterbahnen kreuzen, ohne einen Kurzschluss zu erzeugen. Es sollte hierbei beachtet werden, die Leiterbahnen nicht mit hochfrequenten Signalen übereinander zu platzieren, da es zu Kopplungen kommen könnte. Dies würde Signalverfälschungen erzeugen und die Kommunikation auf den Leiterbahnen stören. Eine Maßnahme ist eine Masseebene, die dieser Platine ebenfalls hinzugefügt wurde.

Die Bestückung der Platine wurde per Hand durchgeführt. Die SMD-Bauteile werden mit Heißluft und Lötkolben gelötet. Bei dem Farbsensor und dem LED-Treiber werden die Lötpads vor dem Verlöten mit Lötzinn vorbereitet. Die Bauteile werden dann mittels Pinzette auf die präparierten Lötpads gelegt. Es wird dann Flussmittel über die Bauteile verteilt. Mit der Heißluft wird das Bauteil nun erwärmt. Bei ausreichend Tem-

peratur schmilzt das Lot. Das Bauteil zieht sich dann von allein auf die Lötpads. Die SMD-Widerstände und -Kondensatoren werden nur mit dem Lötkolben auf der Platine verlötet. Es wäre bei dem Farbsensor und dem LED-Treiber wahrscheinlich besser, diese in einem Reflow-Lötofen zu verarbeiten, da bei dieser Technik stabilere Ergebnisse erzielt werden. Nachdem alles verlötet wurde, erfolgt eine Sichtprüfung der Lötstellen unter dem Mikroskop. Wenn schlechte Lötstellen erkennbar sind, werden diese repariert.
Das einzige mechanische Bauteil ist der LWL-Adapter. Dieser wird mit vier Schrauben der Größe M3 verschraubt. Es müssen keine Muttern verwendet werden, da in das Material des LWL-Adapters Gewindegänge geschnitten werden.

4.5 Inbetriebnahme und Test

Was ist zu Testen?	Wie zu Testen?
auf Kurzschlüsse prüfen	mit dem Multimeter Versorgungs- und Datenleiterbahnen zur Masse prüfen
LEDs ansteuerbar	mit kleinem Testprogramm LEDs ansteuern
LED-Ströme ermitteln	mit Multimeter Spannungsabfall über Vorwiderstand messen und Ströme berechnen
Farbsensor ansprechen	mit kleinem Testprogramm Abfrage der Sensor-ID
Veränderung der Frequenz durch Jumper	mit Oszilloskop Frequenzen bestimmen, Änderung beim Jumper umstecken von Position Q0 zu Q6 (siehe Abbildung 4.2)
Veränderung der Frequenz durch Versorgungsspannung	mit Oszilloskop Frequenz bestimmen, bei Variation der gemessenen Versorgungsspannung des Ringoszillators
Veränderung der Frequenz durch Lichteinstrahlung	mit Oszilloskop Frequenz bestimmen mit ein- bzw. ausgeschalteter Taschenlampe auf Fototransistor leuchten
Daten erhalten vom Farbsensor	mit kleinem Testprogramm Abfrage der Sensorregister
Daten erhalten vom Ringoszillator	mit kleinem Testprogramm die gezählten Flanken im Debugger mit Oszilloskop Frequenzzähler vergleichen
Daten erhalten über UART-Bus	mit kleinem Testprogramm Daten senden und mit H-Term erfassen

Tabelle 4.1: Testplan für die Hardware

Der Testplan 4.1 gibt kurz wieder, welche Punkte wichtig sind, um festzustellen, ob die Hardware funktionsfähig ist. Als erstes müssen die Platine und die Lötstellen auf Kurzschlüsse geprüft werden. Bei den Lötstellen kann die Sichtprobe nur oberflächliche Defekte aufdecken. Als Messgerät wird ein Multimeter verwendet. Dieses wird auf die Einstellung zur Druchgangsprüfung gestellt. Ist ein Kurzschluss vorhanden, ertönt ein Piepton. So werden die Fehler detektiert.
Um zu überprüfen, ob die Leuchtdioden ein- und ausgeschaltet werden können, muss ein kleines Testprogramm geschrieben werden. Dieses schaltet den laut Plan zugehörigen Pin des Mi¬k¬ro¬con¬t¬rol¬lers auf High und der LED-Treiber lässt die entsprechende LED leuchten. Fehler, die hier auftreten können, sind eine Verpolung der LED oder es schaltet sich eine falsche LED an. Als Lösungen sollten die LEDs auf ihre Einbauposition überprüft werden. Bei Verpolung muss die LED gedreht werden. Leuchtet eine falsche LED auf, kann dieser Fehler über Software gelöst werden. Es wird hierfür der laut Plan dazugehörige Pin mit dem passenden Pin getauscht. Die Pläne müssen dann angepasst werden, um wieder eine Übereinstimmung mit der Hard- und Software zu erhalten.
Leuchten die LEDs wie geplant, können nun die Ströme erfasst werden. Die einzige Methode ist, diese zu berechnen. Hierfür wird mit dem Multimeter die Spannung über den Vorwiderständen gemessen. Da die Werte der Widerstände bekannt sind können die Ströme mit $I_{LED} = \frac{U_{Vor}}{R_{Vor}}$ berechnet werden. Gleichzeitig ist die Ausgangsspannung des Treibers zu überprüfen, da sich die Ausgangsspannung des Treibers mit dem Strom ändert. Die LED besitzt eine typische Diodenkennlinie, die zueinander eine unterschiedliche quadratische Form besitzen [24] [23]. Die Ausgangsspannung des LED-Treibers vom Typ TC74ACT244 beträgt etwa 4,5 V bei einem Ausgangsstrom von maximal 24 mA. Um den gewünschten Strom von 35 mA zu erhalten. müssen zwei Treiber parallel geschaltet werden. Der gewünschte Strom ergibt eine typische Spannung, welche sich nur iterativ ermitteln lässt. Ein weiterer Fehler könnte ein falscher Vorwiderstand sein. Hier muss dieser nur durch den richtigen ausgetauscht werden.
Der Ringoszillator lässt sich testen, indem sich die Ausgangsfrequenzen mit Eingriff in die Ringoszillatorglieder, über drei Hardware-Einstellungen verändern lassen. Diese Faktoren sind ein Jumper, der die Ringoszillatorfrequenz über einen zwölfstufigen asynchronen Binärteiler mit Faktor zwei herabsetzt. Die Lichteinstrahlung auf den Fototransistor beeinflusst den Versorgungsstrom für den Ringoszillator.

Als Messgerät kommt ein Oszilloskop zum Einsatz. Mit diesem Messgerät können die Frequenzen an dem Pin 2.7 des Evaluationsboards XMC2Go abgegriffen werden. Da es sich um drei Funktionsprüfungen handelt, müssen diese einzeln durchgeführt werden.

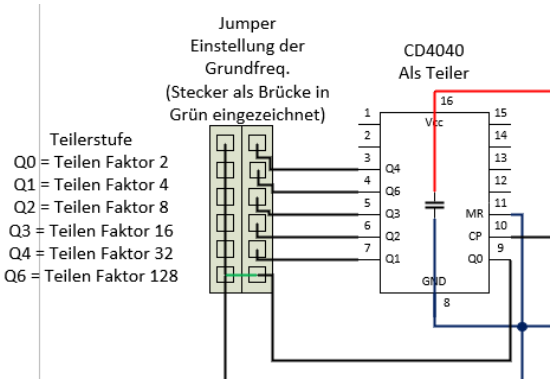


Abbildung 4.2: Mögliche Einstellungen des Jumpers und deren Teilerfaktor

Eine gleichzeitige Durchführung der Test verfälscht die Ergebnisse. Als erstes wird der Jumper in die Position Q0 gesteckt, siehe Abbildung 4.2. Die auf dem Oszilloskop gemessene Frequenz wird notiert. Nun wird der Jumper auf die Position Q6 gesteckt, siehe Abbildung 4.2. Ist die gemessene Frequenz um ca. den Faktor 128 unterschiedlich zu der notierten von der Position Q0, ist dies als gutes Zeichen zu werten. Es werden nun in aufsteigender Reihenfolge die restlichen Positionen des Jumpers überprüft. Alle so erhaltenen Frequenzen werden aufgeschrieben und miteinander Verglichen. Sind alle Frequenzen zueinander unterschiedlich mit einem Faktor von zwei, dann funktioniert der Teiler vollständig. Die Ausnahme ist der Sprung von Q4 auf Q6, hier wird ein Faktor von vier erwartet.

Der Jumper wird nun in die Position Q6 gesteckt und dient als Ausgangspunkt für den nächsten Test. Die Versorgungsspannung des Ringoszillators wird über ein Netzgerät verändert. Eine Änderung der Versorgungsspannung führt zu einer Veränderung der Schaltzeiten der Gatter des Ringoszillators. Daraus ergibt sich eine Veränderung der Frequenz. Ist diese Veränderung auf dem Oszilloskop zu beobachten, ist dieser Test bestanden.
Als letzter Test des Ringoszillators wird mit einer starken Taschenlampe der Fototransistor beleuchtet. Hierfür sollte die Position des Jumpers Q6 und die Versorgungsspannung 15 V betragen. Es wird auch hier die Frequenz von dem Oszilloskop notiert. Die notierte Frequenz ist der Nullpunkt mit Rauschen. Mit der Taschenlampe wird der Fototransistor beleuchtet. Durch die Lichteinstrahlung erzeugt der Fototransistor einen Strom. Dieser durch Darlington-Verstärker verstärkt und beeinflusst die Versorgungsspannung des Ringoszillators. Dies hat eine Veränderung der Frequenz zur Folge. Ist diese Frequenz höher als die notierte Frequenz, funktioniert die Schaltung.

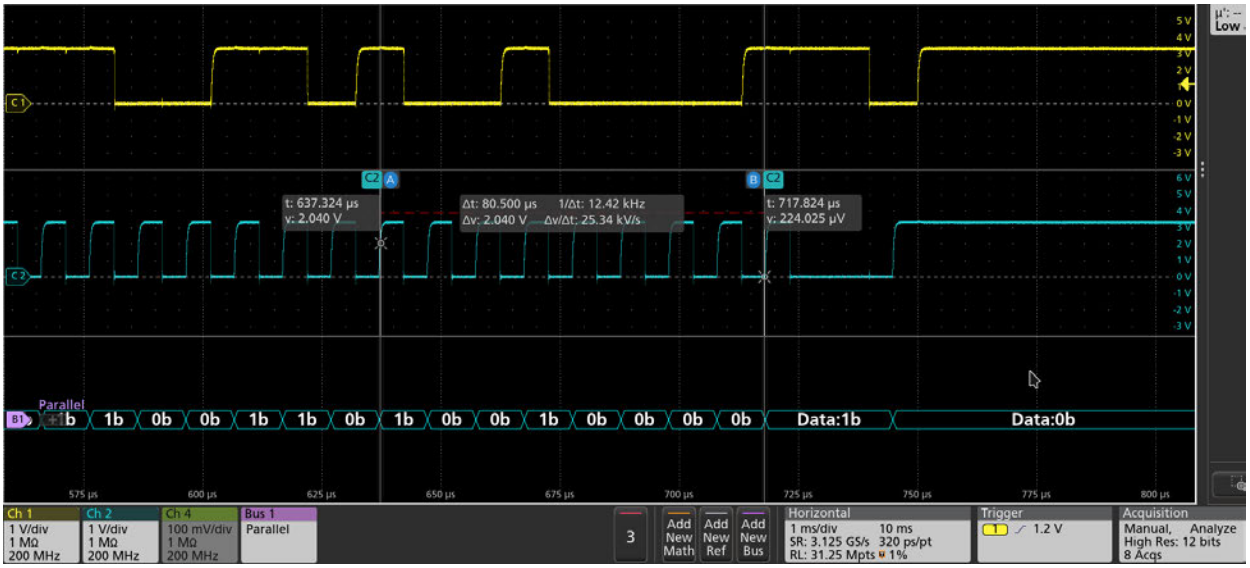


Abbildung 4.3: Ausgelesene Device-ID des Farbsensors auf dem I²C-Bus als Bitfolge. Die gelbe Linie ist die SDA Leitung, die blaue Linie ist die SCL Leitung und unten die übertragene Bitfolge

Es kann nun überprüft werden, ob Daten von dem Mikrocontroller empfangen werden können. Dafür müssen zwei Test durchgeführt werden. Für den Test des Farbsensor wird über den I²C-Bus eine Anfrage an das Register 0x92 des Farbsensors gestellt. Dieses Register enthält die Device-ID des Farbsensors. Erhält man eine Antwort des Farbsensors, kann diese Antwort mit der Device-ID [22] verglichen werden. Stimmen die Antwort (siehe Abbildung 4.3 zwischen den Linien) und die angegebene Device-ID überein ist der I²C-Bus und der Farbsensor funktionsfähig.

```
out
235549
ir
235531
```

Abbildung 4.4: Mit dem Ringoszillator gemessene Frequenz in Hz auf Jumperposition Q3

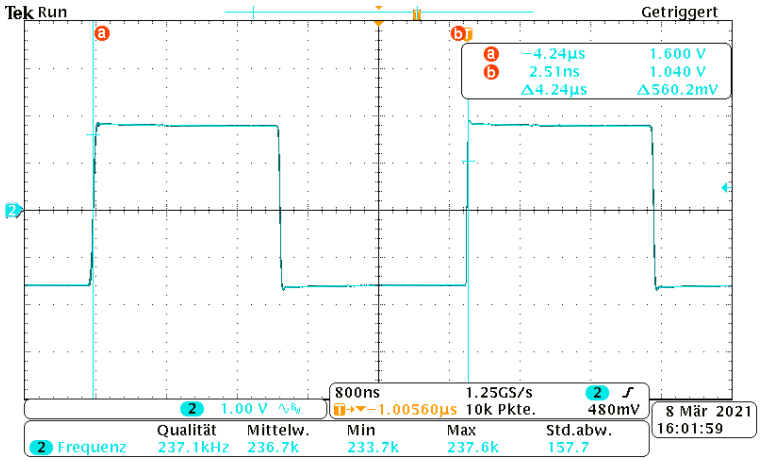


Abbildung 4.5: Mit dem Oszilloskop gemessene Frequenz in kHz

Für den Ringoszillator wird ein kleines Testprogramm geschrieben, das über einen Pin-Interrupt eine Variable hochzählt. Über einen Timer wird die Variable nach einer Sekunde in eine andere Variable kopiert. Das Oszilloskop wird an den Pin für den Pin-Interrupt angeschlossen. Eine Mittelung erleichtert, das Ablesen des Frequenzbereiches am Oszilloskop. Es werden nun zeitgleich die Messungen durchgeführt. Liegt die kopierte Variable (siehe Abbildung 4.4) zwischen dem minimalen und maximalen Wert der gemessenen Frequenz am Oszilloskop (siehe Abbildung 4.5), ist auch hier die Funktion überprüft.

Zuletzt wird der UART-Bus getestet. Hierfür wird ein Testprogramm geschrieben. Dieses Programm sendet einen String, z.B. "Hallo World!", von dem Mikrocontroller über ein Adapterkabel zum Computer. Auf dem Computer kann mit dem Programm HTerm dieser String empfangen werden. Es müssen dafür der COM-Port und die Baudrate eingestellt werden. Der COM-Port gibt an, welcher USB-Port für das Adapterkabel genutzt wird. Mit der Baudrate wird die Übertragungsgeschwindigkeit eingestellt. Ist diese auf beiden Seiten nicht gleich, können keine Daten empfangen werden. Stimmen alle Einstellungen überein, sollte in dem Programm HTerm der String in dem Empfangsfenster auftauchen. Ist dies der Fall, funktioniert die Übertagung von Daten vom Mikrocontroller zum Computer.

5 Entwicklung der Controllersoftware

Der Mikrocontroller wird mit der Entwicklungsumgebung DAVE von dem Hersteller Infineon programmiert. In DAVE wird die Programmierung über sogenannte Apps gehandhabt. Diese Apps enthalten Funktionen mit allen wichtigen Befehlen. Die Apps müssen dem Projekt über die Eingabe „Add new APP“ hinzugefügt werden. Damit diese Apps funktionieren, müssen diesen in DAVE die Pins des Mikrocontrollers zugewiesen werden über den Button „Manual Pin Allocator“ geschieht. Um die Funktionen zu nutzen, muss die Eingabe „Generate Code“ aktiviert werden. Es können nun aus den generierten Dateien die passenden Funktionen der main.c hinzugefügt werden.

5.1 Ansteuerung der Lichtquellen

5.1.1 LED-Steuerung der Farbsensorplatine

Es muss die App *DIGITAL_IO* viermal dem Projekt hinzugefügt werden. Jede hinzugefügte App steht für eine LED. Die Bezugsnamen von den Apps werden geändert, um diese verständlich zu machen. Sie wurden LED_RED, LED_GREEN, LED_BLUE und LED_IR genannt. Es wurden für jede LED Funktionen für Ein- und Ausschalten geschrieben. Die Funktionen werden beispielhaft an denen für die rote LED erklärt. Aus dem generierten Code von Dave wird die Funktion *DIGITAL_IO_SetOutputHigh*() in die main.c kopiert. Der Ausdruck in den Klammern wird mit dem Bezugsnamen LED_RED ersetzt. LED_RED ist die Referenz für den anzusteuernden GIPO-Port. Es ist jetzt möglich, dem GPIO-Pin mit dieser Funktion ein High-Signal zu geben. Die Funktion zum Ausschalten wird ebenfalls aus dem generierten Code von Dave entnommen, *DIGITAL_IO_SetOutputLow*(). Es wird auch hier der Bezugsname mit LED_RED ersetzt. Der GPIO-Pin bekommt mit dieser Funktion ein Low-Signal. Man ist nun in der Lage, die rote LED ein- bzw. auszuschalten. Das Ein- und Ausschalten ist aber noch nicht

zeitlich gesteuert. Hierfür wird ein Timer benötigt. Es wird hierfür die App $TIMER$ hinzugefügt. In den Einstellungen der App muss bei den Event Settings das Kästchen „Time interval event“ markiert werden. Hierdurch wird es ermöglicht, einen Handler aufzurufen, wenn die Timerzeit erreicht ist. Aus dem generierten Code werden fünf Funktionen in die main.c übernommen. Mit der Funktion $TIMER\_Stop(\&TIMER\_0)$ wird der Timer gestoppt. Um nun die Laufzeit des Timers zu verändern, wird die Funktion $timer\_status = TIMER\_SetTimeInterval(\&TIMER\_0, TIME1)$ genutzt. Die gewollte Laufzeit des Timers wird an die zweite Stelle in der Klammer geschrieben. Dies kann als Define oder Integer geschehen. Zum Starten des Timers wird die Funktion $timer\_status = TIMER\_Start(\&TIMER\_0)$ genutzt. Mit der Funktion $while(TIMER\_GetTimerStatus(\&TIMER\_0))$ kann abgefragt werden, ob der Timer läuft. Läuft der Timer, ist der Timer_Status true. Man verbleibt in der While-Schleife. Läuft der Timer nicht ist der Timer_Status false und man verlässt die While-Schleife. Ist die Laufzeit des Timers abgelaufen, wird ein Event ausgelöst. Der Handler wird ausgeführt. Um ein neues Event zu erfassen, muss das alte gelöscht werden. Dies geschieht mit der Funktion $TIMER\_ClearEvent(\&TIMER\_0)$.
Mit all diesen Funktionen können dann die Leuchtdioden gesteuert werden. Zuerst wird die gewünschte Laufzeit für den Timer eingestellt. Beim Einschalten der Leuchtdioden startet der Timer. Wird nach Ablauf der Laufzeit das Timer-Event ausgelöst, springt das Programm in den Handler. Hier wird der Timer gestoppt und das Timer-Event wieder gelöscht. Dadurch ändert sich der Timer_Status von true zu false und man geht aus der While-Schleife. Dann werden die Leuchtdioden ausgeschaltet.

5.1.2 LED-Steuerung der Ringoszillatorplatine

Die Leuchtdioden werden über den LED-Treiber und zwei weitere GPIO-Pins angesteuert. Diese dienen der Stromerhöhung, indem parallel zwei Treiber auf eine Diode geschaltet werden. Die Zeitsteuerung wird auch über einen Timer gesteuert. Die Zeit ist diesmal auf eine Sekunde festgelegt und kann inkrementell erhöht werden.

5.2 Digitale Schnittstelle für den integrierten Lichtsensor/Farbsensor

Die Kommunikation zu dem Farbsensor geschieht über den I²C-Bus. Ein I²C-Bus benötigt einen Master, der mit den verschiedenen Slaves Daten austauscht. Der Master ist in diesem Fall der Mikrocontroller. Es wird deswegen die App I2C_MASTER hinzugefügt. In den Einstellungen wird der „Desired bus speed“ auf 400 gesetzt. Die Busfrequenz beträgt somit 400 kHz. In der erweiterten Einstellung werden Interrupts für den „Transmit mode und Receive mode“ ausgewählt. Als Pins werden für SCL P2.11 und für SDA P2.10 ausgewählt. Aus dem generierten Code werden zwei Funktionen und zwei While-Schleifen verwendet. Es werden jeweils zwei Funktionen zusammen verwendet. Um vom Master Informationen zum Slave zu senden, werden die Funktionen $I2C\_MASTER\_Transmit()$ und $while()$ genutzt. Die Klammer aus der ersten Funktion ist wie folgt aufgebaut. Zuerst wird der Referenzwert des Masters eingetragen. Die zweiter Stelle ist die Angabe, ob das Startbit gesendet werden soll. Als drittes wird die I²C-Adresse des Slaves eingetragen. An vierter Stelle sind die Daten für die Übertragung. An fünfter Stelle wird die Anzahl der zu sendenden Bytes eingetragen. Als letztes wird entschieden, ob das Stoppbit gesetzt werden soll. Die While-Schleife wartet, bis die Übertragung abgeschlossen ist. Um Daten zu empfangen, werden ebenfalls zwei Funktionen benötigt. Die Funktion ist $I2C\_MASTER\_Receive()$ und $while()$. Im Vergleich zu der Sendefunktion sind in der Empfangsfunktion zwei Parameter in der Klammer verändert. Anstatt der Daten, die gesendet werden sollen, wird der Referenzwert der Speicherstelle der zu empfangenen Daten eingetragen. Es kommt nun nach der Stelle mit dem Stoppbit eine Einstellung, ob ein NOACK gesendet werden soll.
Die wichtigen Register des Farbsensors können in Befehls- und Datenregister aufgeteilt werden. Die Befehlsregister sind “ENABLE”, „ATIME“, „CONTROL“ und „IR“. Die verwendeten Datenregister sind „ID“, „CDATAL“, „CDATAH“, „RDATAL“, „RDATAH“, „GDATAL“, „GDATAH“, „BDATAL“ und „BDATAH“. Die Adressen können in dem Datenblatt eingesehen werden [22].

Mittlung der Erhaltenen Messwerte des Farbsensors

Um eine höhere Genauigkeit zu erreichen werden die Daten von fünf Messzyklen aufaddiert und mit diesen eine Mittelung durchgeführt. Dieser gemittelte Wert wird dann über die UART-Schnittstelle an den Computer gesendet.

Register	Beschreibung
ENABLE	Starten des Farbsensors und des ADC
ATIME	Steuerung der Integrationszeit
CONTROL	Steuerung der Verstärkung
IR	Schaltet Infrarotkanal frei
ID	Enthält die Geräte-Identifikation
CDATAL	Enthält niedrigere Daten der IR-LED
CDATAH	Enthält höhere Daten der IR-LED
RDATAL	Enthält niedrigere Daten der R-LED
RDATAH	Enthält höhere Daten der R-LED
GDATAL	Enthält niedrigere Daten der G-LED
GDATAH	Enthält höhere Daten der G-LED
BDATAL	Enthält niedrigere Daten der B-LED
BDATAH	Enthält höhere Daten der B-LED

Tabelle 5.1: Wichtige Register des Farbsensors

5.3 Funktion des Ringoszillator

Die Frequenz des Ringoszillators kann durch Lichteinfluss variiert werden. Hierfür wurde die App *PIN_INTERRUPT* ausgewählt. Bei den Einstellungen der App ist die Flankenerkennung auf „Rising Edge“ gestellt. Die Interrupt-Priorität wurde erhöht. Dies ist wichtig, um eine genaue Zählung zu garantieren, weil sonst Interrupts nicht erfasst werden könnten. Bei einem Interrupt wird in einen Handler gesprungen. Dieser Handler zählt bei jedem Aufruf eine Variable hoch. Da die Zeit fest auf eine Sekunde gesetzt wurde, entspricht es damit der ankommenden Frequenz. Um einen stabileren Wert der Frequenz zu erhalten, wird ein Mittelwert erzeugt. Über einen Jumper auf der Platine kann die Grundfrequenz des Ringoszillators über die Anzahl der Kettenglieder variiert werden. Es ist ein Bereich von einigen kHz bis einigen MHz möglich. Hierfür muss eine Abschätzung des Frequenzbereichs durchgeführt werden. Aus dem Datenblatt [28] ist zu entnehmen, dass ein Interrupt 21 Zyklen des Systemtakts benötigt. Dieser ist auf 32 MHz gesetzt, so können ohne Berücksichtigung des Interrupt-Handlers bis zu 1,5 Millionen Interrupts pro Sekunde ausgeführt werden. Diese Zahl reduziert sich durch die Ausführungszeit des Interrupt-Handlers. In unser Anwendung ist dieser auf das inkrementieren einer Globalen-Variable beschränkt. Es ergibt sich abgeschätzt die Möglichkeit, über 100000 Interrupts durchzuführen. Durch den Frequenzteiler ist sichergestellt, dass die höchste Anzahl an Interrupt-Anforderungen diesen Maximalwert nicht überschreitet.

5.4 Schnittstelle zum PC

Als Interface zu anderen Mikrocontrollern bzw. zum Computern wurde der UART-Bus ausgewählt. Dies gilt sowohl für die Farbsensor- und Ringoszillatorplatine. Es wird deswegen nur einmal auf die Programmierung eingegangen. Die App *UART* wird dem Projekt hinzugefügt. In den Einstellungen werden unter „General Settings“ die „Desired Speed [baud]“ auf 115200 gesetzt. In „Advanced Settings“ wird der „Transmit mode“ auf Direct und der „Receive mode“ auf Interrupt gestellt. Dies ist wichtig, da die ausgehenden Daten ohne Verzögerung versendet werden können. Die eingehenden Daten können zu jeder Zeit empfangen werden und müssen deswegen mit einem Interrupt behandelt werden. Für den Sende-Pin des UART-Busses wurde der Pin P2.0 für beide Platinen ausgewählt. Der Empfangs-Pin für die Farbsensorplatine ist der Pin P2.6. Bei der Ringoszillatorplatine wurde der Pin P2.9 verwendet. Um eine Kommunikation mit einem Computer herzustellen, muss ein Adapter verwendet werden. Dieser Adapter wandelt die UART-Verbindung in eine USB-Verbindung. Um eine Verbindung zu anderen Mikrocontrollern herzustellen, muss der Sende-Pin des einen Mikrocontrollers mit dem Empfangs-Pin des anderen Mikrocontrollers verbunden werden. Es werden zwei Funktionen genutzt. Diese Funktion $UART\_Transmit()$ wird zum Senden von Daten genutzt. Die erste Stelle der Klammer ist der Referenzwert zu der UART_t Struktur. Die zweite Stelle ist ein Pointer oder Array zu den Daten, die gesendet werden sollen. Die letzte Stelle gibt an, wie viele Bytes gesendet werden sollen. Um Daten zu empfangen, wird die folgende Funktion genutzt: $UART\_Receive() == UART\_STATUS\_SUCCESS$. Die erste Stelle dieser Funktion verweist auf die gleiche UART_t Struktur. Die zweite Stelle gibt an, wohin die erhaltenen Daten gespeichert werden sollen. Dies kann über einen Pointer oder ein Array geschehen. Die letzte Stelle gibt die Anzahl der Bytes an, die empfangen werden sollen.

5.5 Steuerungskonzept

Die Steuerung beinhaltet drei einstellbare Werte. Diese Werte sind die Zeit, die Verstärkung und die Auswahl zwischen der Infrarot-LED und der RGB-LED. Über die Eingabe des Befehls „e“ wird in einen Einstellungsmodus gegangen. Zuerst kann die Integrationszeit eingestellt werden. Hierfür wird eine Zahl von Null bis 255 vom Computer an den Mikrocontroller gesendet. Die Integrationszeit sind für die Farbsensor- und Ringoszillatorplatine unterschiedlich. Bei der Farbsensorplatine wird mit diesem Wert die Inte-

grationszeit des Sensors verändert. Der auswählbare Bereich liegt zwischen 0 = 2,78 ms und 255 = 712 ms [22]. Bei der Ringoszillatorplatine wird die Messzeit für jedes Inkrement des Wertes um eine Sekunde verlängert. Die Standard-Messzeit beträgt 1 s. Dann wird automatisch in die nächste Einstellung gewechselt. Diese Einstellung steuert die Verstärkung bzw. den Diodenstrom der Sensoren. Bei der Farbsensorplatine können vier Verstärkungstufen des Sensors ausgewählt werden, diese sind „1x“ „„4x“, „16x“ und „64x“ [22]. Bei der Ringoszillatorplatine sind vier unterschiedliche Stromstärken auswählbar, diese sind 0 mA, für alle Leuchtdiodenfarben. Die drei Stufen für die Infrarot-LED sind ca. 18,7 mA, ca. 18,7 mA und ca. 37,3 mA. Die drei Stufen für die RGB-LED mit der Farbe Rot sind ca. 25 mA, ca. 46,6 mA und ca. 71,6 mA. Die drei Stufen für die RGB-LED mit der Farbe Grün sind ca. 81,8 mA, ca. 163,6 mA und ca. 245,5 mA. Die drei Stufen für die RGB-LED mit der Farbe Blau sind ca. 81,8 mA, ca. 163,6 mA und ca. 245,5 mA. Als letzte Einstellung kann ausgewählt werden ob die RGB-LED oder die Infrarot-LED verwendet werden soll. Die Auswahl geschieht über die Bezeichner „r“ = rote LED, „g“= grüne LED, „b“= blaue LED und „i“= Infrarot-LED.

5.6 Modul- und Gesamtfunktionstest gemäß Testplan

Was ist zu testen?	Wie zu testen?
Überprüfung, ob LED mit LWL auf Sensor leuchtet	Test LED schalten und Messwert im Debugger bestimmen
Zeitsteuerung der LED überprüfen	ein- und ausschalten mit Stoppuhr messen
Optische Charakteristik der Sensoren überprüfen	Werte aufnehmen und mit Datenblatt vergleichen

Tabelle 5.2: Testplan für die Software der Farbsensorplatine

Um die wichtigsten Funktionen der Software der Farbsensorplatine zu testen, werden drei Tests durchgeführt. Bei dem ersten Test wird überprüft, ob die LED genügend Licht zum Sensor überträgt. Ist dies nicht der Fall, muss der Strom der für die LED erhöht werden oder ein dickerer Lichtwellenleiter verwendet werden. Ein dickerer Lichtwellenleiter fängt mehr von der ausgestrahlten Lichtleistung ein. Um die Zeitsteuerung der Leuchtdioden zu überprüfen, wird mit einer Stoppuhr oder elektronisch die eingestellte Zeit gemessen. Bei dem letzten Test werden die Sensordaten mit dem Datenblatt verglichen.

Test Conditions	Red / Clear Channel		Green / Clear Channel		Blue / Clear Channel	
	Min	Max	Min	Max	Min	Max
$\lambda_D = 465$ nm (1)	0%	13%	10%	38%	70%	91%
$\lambda_D = 525$ nm (2)	3%	22%	59%	86%	10%	40%
$\lambda_D = 615$ nm (3)	80%	110%	0%	15%	3%	26%

Abbildung 5.1: Empfindlichkeit der Sensoren des Farbsensors nach Wellenlängen des Lichtes[22]

LED Farbe	Roter Sensor	Grüner Sensor	Blauer Sensor
Blau	1%	31%	100%
Grün	13%	100%	54%
Rot	100%	1%	14%

Tabelle 5.3: Gemessene Empfindlichkeit des Farbsensors in Prozent nach Wellenlänge des Lichtes

Die 100% bei den jeweiligen Farben ist als Ausgangswert für die Berechnung der anderen Prozentwerte genommen worden. Der einzige Wert der nicht in den vorgegebenen Bereich passt sind die 54% bei der Kombination grüne LED und blauer Sensor. Dies kann an der RGB-LED liegen, da der blaue Lichtpunkt in der Mitte liegt und der grüne Lichtpunkt an der Seite. Dadurch ist das eingekoppelte Licht nicht identisch, welches die Messwerte beeinflussen kann.

Was ist zu testen?	Wie zu testen?
Überprüfung, ob LED mit LWL auf Sensor leuchtet	Test LED anschalten und Messwert im Debugger bestimmen
Zeitsteuerung der LED überprüfen	ein- und ausschalten mit Stoppuhr messen
Frequenzen überprüfen	Messwert im Debugger mit Oszilloskopmessung vergleichen

Tabelle 5.4: Testplan für die Software der Ringoszillatorplatine

Um die wichtigsten Funktionen der Software der Ringoszillatorplatine zu testen, werden ebenfalls drei Test durchgeführt. Bei den ersten zwei Tests, ist die Durchführung gleich zu den ersten zwei Tests der Farbsensorplatine. Der dritte Test überprüft, ob die gemessene Frequenz stimmt. Hierfür wird ein Oszilloskop an den Interrupt-Pin angeschlossen. Mit dem Oszilloskop wird die reelle Frequenz gemessen. Es gibt einen Frequenzbereich an, in dem die gemessene Frequenz der Ringoszillatorplatine liegen muss. Liegt die Frequenz außerhalb dieses Bereiches, muss die Programmierung überprüft werden. Es könnte sein, dass die Abschätzung aus 5.3 nicht stimmt. Als Beispiel können die Abbildungen Abbildung 4.4 und Abbildung 4.5 dienen.

6 Entwicklung der PC-Software

Die auf dem Computer genutzte Software wird mit dem Programm Matlab geschrieben. Mit dieser Software können Grafiken erstellt und Datenverbindungen abgefragt werden. Darüber hinaus erfolgt in der Matlab-Umgebung die Steuerung des Mikrocontrollers und somit der Messung.

6.1 Umsetzung als Matlab-Skript

Das Programm wird als Matlab-Skript konzipiert. Hier können einzelne Programmteile getrennt voneinander ausgeführt werden. Das Programm wird in vier Teile aufgegliedert. Ein Teil ist die Erzeugung einer Schnittstelle und Einstellung von Standardwerten. Die Teile 2-4, Synchronisation live-Darstellung sowie Datenspeicherung, sind in einem Programmteil zusammengefasst.

6.2 Schnittstelle zum Controller

Um eine Schnittstelle einzurichten, werden die folgenden zwei Funktionen genutzt. Die Funktion $device = serialport()$ stellt die Verbindung mit einem COM-Port her. In der Klammer stehen zwei Parameter. Der erste Parameter gibt den COM-Port an, an dem der UART-Adapter angeschlossen ist. Der zweite Parameter stellt die Übertragungsgeschwindigkeit der Verbindung ein. Der Wert wird in Bit pro Sekunde angegeben. Um den Timeout der Verbindung zu erhöhen, wird die Funktion $device.Timeout = 30$ genutzt. Die Zahl gibt den Timeout in Sekunden an.
Über die eingestellte Schnittstelle können drei verschiedene Einstellungen getätigt werden. Es werden die Integrationszeit , die Verstärkung und die LED eingestellt.
Die Integrationszeit hat 255 verschiedene Stufen. Diese Stufen sind individuell für jede Platine zu interpretieren. Der Farbsensor hat 255 fest installierte Integrationszeiten.

Der Ringoszillator hat einen fest eingestellten Timer von einer Sekunde. In diesem Fall wird die eingestellte Stufe als Multiplikator interpretiert. Um dieses Kommando über den UART-Bus zu senden, wird die Funktion *write*() genutzt. In der Klammer wird als erstes der COM-Port angegeben. Die zweite Stelle gibt die Stufe der Integrationszeit an und die letzte gibt an, welcher Datentyp zum Senden verwendet wird.
Bei der zweiten Einstellungen kann die Verstärkung eingestellt werden. Hier kann zwischen drei verschiedenen Stufen gewählt werden. Stufe eins ist die niedrigste Verstärkung und Stufe drei die höchste. Die Einstellung wird automatisch als String mit der folgenden Funktion gesendet *writeline*(). Es werden in der Funktion der COM-Port und die Stufe der Verstärkung als Zahl angegeben.
Die gleiche Funktion wird genutzt, um die dritte Einstellung (Auswahl der LED) vorzunehmen. Statt der Zahl wird an der zweiten Stelle ein Buchstabe eingetragen. Die Buchstaben repräsentieren die Farben der Leuchtdioden.

LED-Farbe	Buchstabe
Rot	r
Grün	g
Blau	b
Infrarot	i

Tabelle 6.1: Buchstaben zur Auswahl der LED

6.3 Visualisierung von Messdaten

Um die Daten darstellen zu können, wurde die Funktion *plot*() verwendet. Es gibt drei Parameter in der Funktion. Die ersten beiden Parameter geben die X- und Y-Achse des Plots an. Die Anzahl an Datenpunkten müssen übereinstimmen, um eine Grafik erstellen zu können. Der letzte Parameter gibt die Farbe der Kurve in der Grafik an. Es werden alle Farbdaten und das Rauschen dargestellt. Die Farbdaten ergeben sich aus einer eingeschalteten LED und der Lichtauskopplung aus dem LWL. Das Rauschen ergibt sich aus dem Umgebungslicht bei ausgeschalteter LED, welches durch den Farbsensor erfasst wird. Das Rauschen, bei dem Ringoszillator ergibt sich aus dem Umgebungslicht und der Grundfrequenz des Ringoszillators. Hier gibt es Unterschiede zwischen dem Farbsensor und dem Fototransistor. Da der Farbsensor für jede Farbe eine Sensorfläche besitzt, werden hier bei jeder LED-Farbe alle Sensorflächen ausgewertet. Dies bedeutet, es sind bis zu zwölf Datenpunkte in einer Messung in der Grafik sichtbar Abbildung 6.1. Die Farben

in der Grafik gibt an welcher ADC des Farbsensors die Daten aufnimmt. Die x- und y-Skale wird automatisch an die aufgezeichnete Kurve angepasst. Der Lichtwellenleiter wurde während der Messung an den richtigen Abstand zum Farbsensor angepasst. Die x-Achse ist von 0 fortlaufend bis zum 1000 Datenpunkt. Es wird dann wieder bei der 0 begonnen. Die y-Achse hat einen Messbereich von 65536 Werten bei dem Farbsensor. Der Messbereich bei dem Ringoszillator ist begrenzt durch die Maximale Frequenz welche aus der Abschätzung 5.3 ermittelt wurde. Dieser Wert sollte aber nicht erreicht werden, da bei diesem Wert keine anderen Berechnungen des Mikrocontrollers einbezogen wurden. Das Rauschsignal entsteht wenn Umgebungslicht durch den Lichtwellenleiter-Adapter dringt. Ist die LED eingeschaltet wird die ankommende Lichtleistung von dem passenden ADC des Farbsensors aufgenommen und als dimensionslos Größe an den Mikrocontroller gesendet.

6.4 Aufzeichnung der Messdaten mit Zeitstempel

In Vorbereitung einer Synchronisation mit einem Zykliersystem werden die Daten mit einem Zeitstempel versehen. Dies wird im folgenden Abschnitt näher beschrieben.
Um die Daten zu synchronisieren, werden Buchstaben mit den Daten vom Mikrocontroller verschickt. Die Buchstaben geben die Farben der Leuchtdioden an, welche aktuell eingeschaltet ist. Diese werden mit der Funktion $choose\_color = read()$ empfangen. In der Klammer werden die Verbindung, die Datenmenge und Datentyp angegeben. Mit einer Switch-Case-Anweisung werden die Daten den gewählten Plots zugewiesen. Die Auswahl der entsprechenden Case-Fälle geschieht über Buchstaben. Der folgenden Tabelle ist zu entnehmen, welcher Buchstabe zu den entsprechenden Daten gehört.

Buchstabe	Daten
r	rote Daten
g	grüne Daten
b	blaue Daten
i	infrarote Daten
o	Rauschen

Tabelle 6.2: Buchstaben zur Auswahl der LED für die switch-case-Bedingung

Um die Daten zu speichern, werden sie in einer Matrix zwischengespeichert. In dieser Matrix sind die Daten und der Zeitpunkt, wann die Daten gesendet wurden, eingetragen.

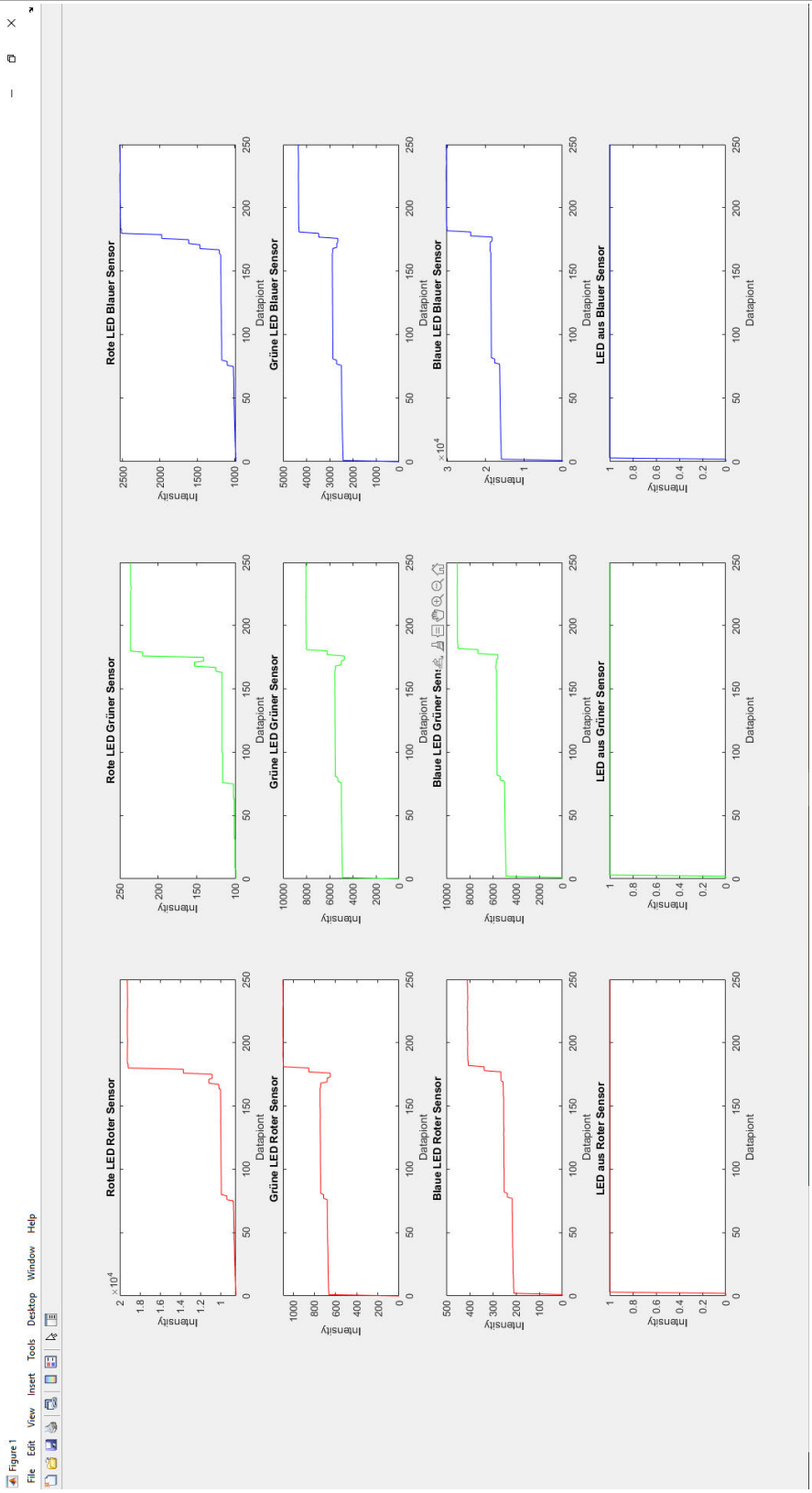


Abbildung 6.1: Beispielplot des Farbsensors für eine Messung mit der RGB-LED

Diese Matrizen werden mit der Funktion *writematrix*() in einer CSV-Datei gespeichert. Die Parameter der Funktion sind die Anzahl der Daten und der Name der Datei, die immer in einem eigenen Datenrahmen gespeichert werden. Der Dateiname wird bei Erstellen jeder neuen Datei mit einer anderen Zahl gespeichert. Die Dateien werden durch den angelegten Dateipfad in einzelne Ordner gelegt. So ist ein Überschreiben der alten Daten ausgeschlossen.

6.5 Modul- und Gesamtfunktionstest gemäß Testplan

Was ist zu Testen?	Wie zu Testen?
Daten erhalten	Testprogramm schreiben zur Datenerhaltung
Daten synchron in Grafik einordnen	Testprogramm schreiben um Daten von einer LED zu erhalten
Erhaltene Daten darstellen	Testprogramm schreiben zum darstellen von Daten
Controller ansprechen	Testprogramm, um Daten zu senden und Änderungen mit Debugger vergleichen
Datenspeicherung	Gespeicherte Files überprüfen, ob Anzahl an Daten und Nummerierung stimmt
Daten aus gespeicherten Files darstellen	Grafik aus gespeicherten Daten mit Screenshot von Datenaufnahme vergleichen

Tabelle 6.3: Testplan für die Software der Matlab-Oberfläche

Es werden einige Tests durchgeführt, um die Funktionen des Matlab-Codes zu prüfen. Der erste Test prüft, ob Daten von dem Mikrocontroller zu der Matlab-Oberfläche gesendet werden. Es wird hierfür ein Programm geschrieben, das die Daten von dem Mikrocontroller dauerhaft erhält. Hierbei ist erstmal nicht wichtig, in welcher Reihenfolge die Daten erhalten werden. Die Daten werden auf der Matlab-Konsole ausgegeben. Kommen keine Daten an, wird zuerst das Kabel überprüft. Die Verbindungen bei dem Kabel können lose sein. Diese müssen dann richtig gesteckt werden. Danach sollte der eingestellte COM-Port überprüft werden. Ist der falsche COM-Port ausgewählt, können keine Daten erhalten werden. Die restlichen möglichen Fälle können durch die Tests aus Kapitel 5 ausgeschlossen werden.
Um synchrone Daten zu erhalten, wird ein Testprogramm geschrieben. Dieses Programm besteht aus einer Switch-Case-Anweisung, die alle Leuchtdioden abfragt. Es wird über den Mikrocontroller eine LED dauerhaft angesteuert. Es wird so erkannt, ob die Daten in die passende Spalte eingeordnet werden. Werden die Daten falsch eingeordnet, müs-

sen die Case-Fälle überprüft werden. Hier kann es sein, dass die Fälle falsch eingetragen wurden. Ein weiterer Fehler kann sein, dass der Timeout zu kurz gewählt wurde. Dieser muss dann schrittweise erhöht werden.
Sind die Daten richtig eingeordnet, muss geprüft werden, ob diese richtig dargestellt wurden. Hierfür wird ein Programm geschrieben, das die Daten darstellt. Es werden zwei Grafiken erzeugt. Bei der ersten Grafik wird der Sensor ganz normal betrieben. Hier sollte die Grafik kontinuierlich neue Werte anzeigen. Bei der zweiten Grafik wird der Lichtwellenleiter entfernt. Auf der Grafik sollte jetzt nur das Rauschen angezeigt werden. Liefert die zweite Grafik Daten, so müssen die äußeren Einflüsse überprüft werden. Es leuchtet dann Umgebungslicht auf den Sensor.
Um zu überprüfen, ob sich der Sensor ansprechen lässt, werden die Befehle in dem Matlab-Skript ausgeführt. Als Ausgangswert sind die Defaultwerte des Mikrocontrollers zu empfehlen. Es sollten nun deutliche Abweichungen zum Prüfen verwendet werden. Über den Debugger des Mikrocontrollers kann beobachtet werden, wenn sich ein Wert nach den Vorgaben in dem Register ändert.
Um zu testen, ob die Daten mit Zeitstempeln gespeichert, werden wird ein kleines Testprogramm geschrieben. Dieses Programm erzeugt jeweils bei Datenerhalt einen Zeitstempel. Die Daten und der Zeitstempel werden in einer Matrix zwischengespeichert. Nach den ersten 50 Datenpunkten sollte eine neue CSV-Datei im Matlab-Verzeichnis auftauchen. Diese Datei öffnet man mit einem Tabellenprogramm. Sind hier 50x4 Felder mit glaubwürdigen Werten befüllt, werden die Daten und der Zeitstempel richtig gespeichert. Das Programm lässt man dann einige Zeit im Hintergrund laufen. Ist die laufende Nummer der erstellten CSV-Dateien aufsteigend, werden alle 50 Datenpunkte neue Dateien erstellt. Verändert sich die laufende Nummer nicht, werden immer die alten Datenpunkte überschrieben. Es gehen so Daten verloren.
Um zu testen, ob die gespeicherten Daten mit den aufgenommen Daten übereinstimmen, wird ein Screenshot von der aktuellen Grafik erzeugt. Dieser wird mit der Grafik aus den gespeicherten Daten verglichen. Stimmen die Grafiken überein, sind die gespeicherten Daten richtig wiedergegeben worden.

7 Erprobung des gesamten Messsystems

7.1 Planung der Messreihen

Für den Funktionsvergleich zwischen dem Messaufbau mit Farbsensorplatine und Ringoszillatorplatine werden insgesamt 16 Messungen geplant. Dabei werden jeweils zwei verschiedene Lichtwellenleiter verwendet. Die Lichtwellenleiter unterscheiden sich im Durchmesser und im Material. Es wird deswegen unterschiedlich viel Licht von Leuchtdioden in die Lichtwellenleiter eingekoppelt, weshalb ein Unterschied in den Messergebnissen erwartet wird. Diese Messung wird jeweils für die RGB-LED und die Infrarot-LED durchgeführt. Es werden hier für jede Kombination aus Sensor, LED und Lichtwellenleiter insgesamt acht Messungen durchgeführt (siehe Tabelle 7.1).
Um zu überprüfen, ob das Licht auskoppelt, wenn der Lichtwellenleiter gebogen wird, werden vier weitere Messungen durchgeführt. Es werden für die Messungen zwei unterschiedliche Biegeradien verwendet, eine schwache und starke Biegung. Die schwache Biegung sollte eine schwache Auskopplung des Lichtes erzeugen, die starke Biegung dagegen eine starke Auskopplung.
Alle Messungen werden unter einer Abdeckung durchgeführt. Diese verhindert, dass Umgebungslicht die Ergebnisse beeinflusst. Um den Einfluss des Umgebungslichtes festzustellen, werden vier Messungen gemacht. Die Leuchtdioden werden hierfür nicht angeschaltet.

	Plastikfaser	Glasfaser
Farbsensorplatine	2	6
Ringoszillatorplatine	2	6

Tabelle 7.1: Anzahl an Messungen, je nach Lichtwellenleiter und Sensor

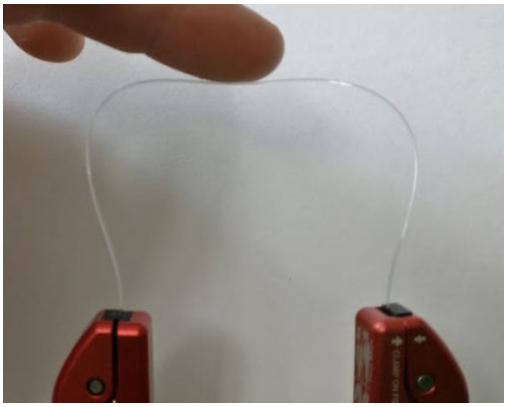


Abbildung 7.1: Lichtwellenleiter mit schwacher Biegung

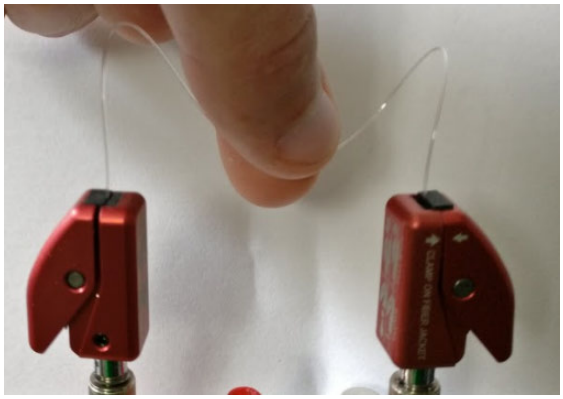


Abbildung 7.2: Lichtwellenleiter mit starker Biegung

7.2 Durchführung und Auswertung

Zuerst wird der Farbsensor mit der Plastikfaser getestet. Hierfür wird der Parameterwert für die Integrationszeit auf 192 gesetzt. Dieser Wert entspricht 178 ms. Für die Messung wird eine Verstärkung von 16 gewählt. Der zugehörige Parameter wird daher auf zwei gesetzt. Es werden jeweils für die RGB- und Infrarot-LED mindestens 100 Datenpunkte aufgenommen, um ein eingeschwungenes System zu erhalten. Für die Messungen mit der Glasfaser werden andere Parameterwerte bei der RGB-LED genutzt. Hier beträgt der Parameterwert 143, dies entspricht einer Integrationszeit von 314 ms. Als Verstärkung wird ein Parameterwert von drei eingestellt. Dies entspricht einer Verstärkung von 64. Die restlichen Messbedingungen bleiben unverändert. Für die Messung der Glasfaser mit der Infrarot-LED werden die Einstellungen aus der ersten Messung verwendet. Für die letzten Messungen wird die Glasfaser leicht mit dem Finger in Biegeradius verändert. Dies wird für beide Leuchtdioden mit den jeweiligen Einstellungen gemacht. Hiermit wird eine Auskopplung des Lichtes erzwungen. Es wird eine Veränderung in der Messkurve erwartet.

Messungen mit der Farbsensorplatine und der Plastikfaser

In der Abbildung 7.3 in der linken Grafik ist die natürliche Messungenauigkeit des Farbsensors für den Infrarotkanal erkennbar. Diese resultiert aus der unterschiedlichen Anzahl an Photonen, die die Sensorfläche treffen. Prozentual liegt die Messungenauigkeit bei dieser Messung bei ca. $\pm 0{,}3\,\%$ und ist somit vernachlässigbar. Es ist auch erkennbar, dass

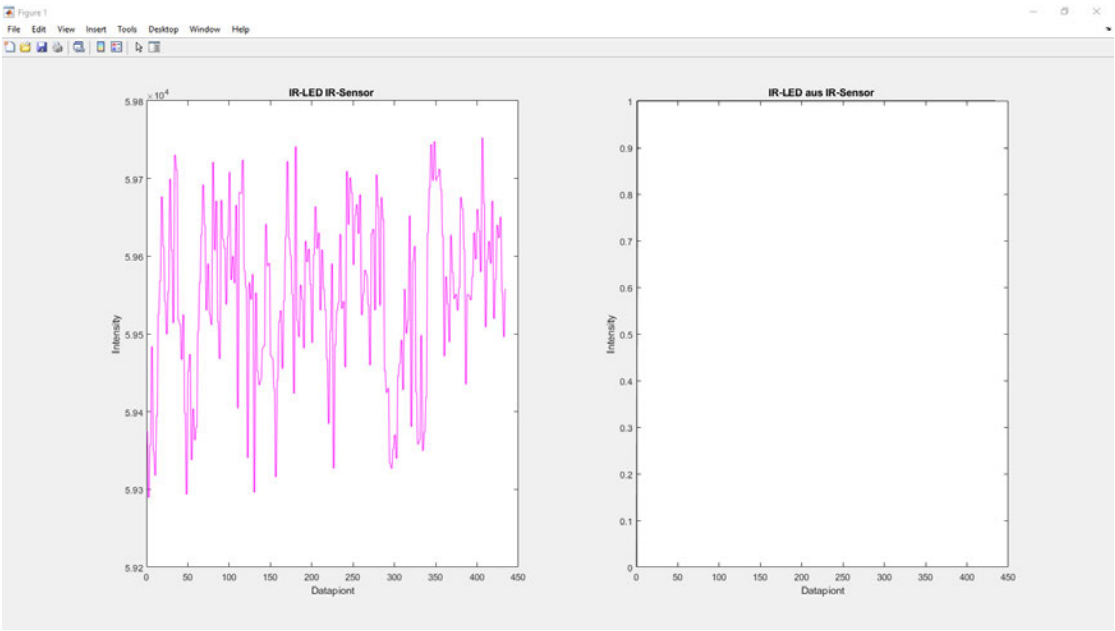


Abbildung 7.3: Infrarot-Messung mit Farbsensor mit Plastikfaser
linke Grafik: Messung des ankommenden Lichtes bei eingeschalteter Infrarot-LED
rechte Grafik: Messung des ankommenden Lichtes bei ausgeschalteter Infrarot-LED

die Einstellungen des Sensors ausreichen. Würde die Verstärkung erhöht, würde man den Sättigungsbereich kommen. Der Messbereich liegt zwischen 0 und 65535 und bei mehr Licht wird nur der Wert 65535 angezeigt. Hier ist es dann ratsam, die Integrationszeit oder die Verstärkung zu verringern. Die rechte Grafik zeigt die Rausch-Messung, die mit dem Wert von eins, Restlicht erkannt wird. Die Ursache für das Signal in der Rauschmessung kann sein, dass Umgebungslicht trotz Abdeckung zu dem Farbsensor gelangt ist. Dieser Wert ist aber so gering, dass er keine Auswirkungen auf die Messung hat und ist somit vernachlässigbar.

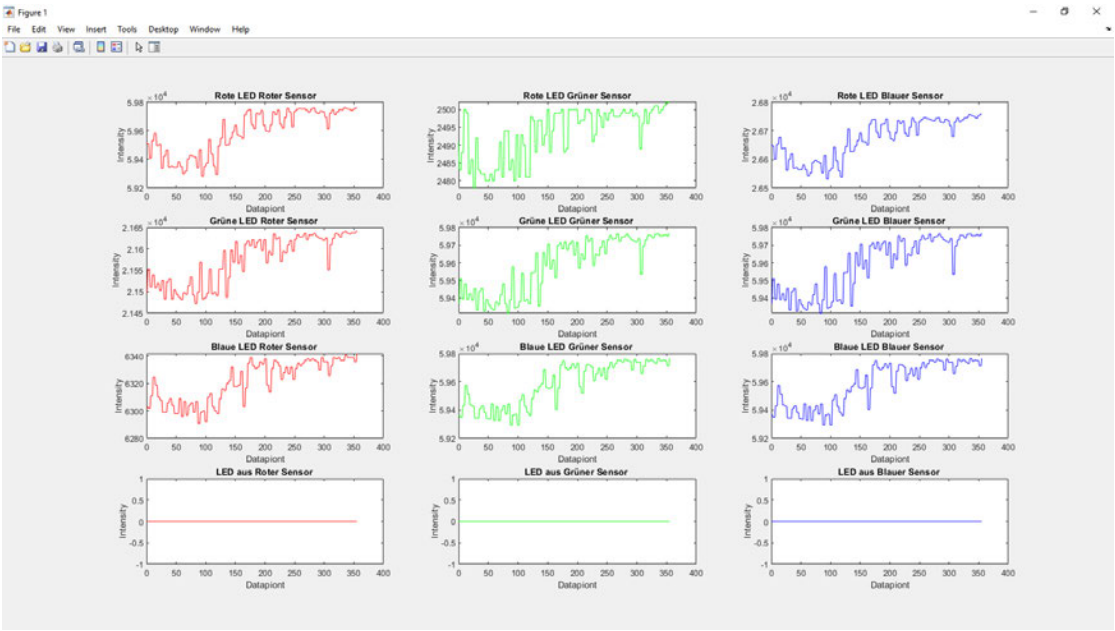


Abbildung 7.4: RGB-Messung mit Farbsensor und Plastikfaser
Die ersten drei Zeilen sind Messungen mit eingeschalteter RGB-LED die letzte Zeile ist die Rauschmessung bei ausgeschalteter RGB-LED.

In der Abbildung 7.4 sind die Messungen für die RGB-LED abgebildet. Die Farben geben den Farbsensorkanal an, der die Werte aufgezeichnet hat. Die erste Zeile wurde mit der roten Leuchtdiode aufgezeichnet, die zweite Zeile ist mit der grünen Leuchtdiode. Die dritte Zeile ist mit der blauen Leuchtdiode aufgezeichnet worden. Die letzte Zeile ist die Rauschmessung und wurde mit ausgeschalteten Leuchtdioden aufgezeichnet. Die natürliche Messungenauigkeit des Farbsensors für die RGB-Kanäle ist aus den erhaltenen Daten erkennbar. Prozentual liegt die Messungenauigkeit bei dieser Messung bei ca. $\pm 0{,}4\,\%$ und ist somit vernachlässigbar. Es ist auch erkennbar, dass die Einstellungen des Farbsensors ausreichen. Bei einer Erhöhung der Verstärkung erhält man das gleiche Problem wie bei der Messung aus Abbildung 7.3. Die Rausch-Messung ist mit dem Wert von Null bei allen Kanälen als kein ankommendes Licht zu interpretieren.

Messungen mit der Farbsensorplatine und der Glasfaser

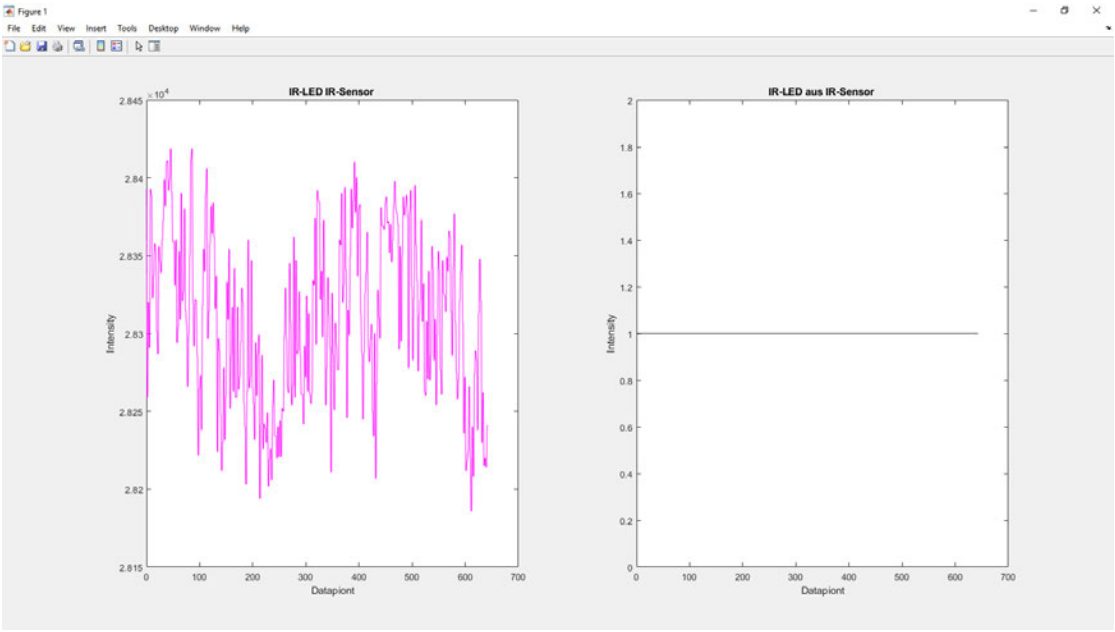


Abbildung 7.5: Infrarot-Messung mit Farbsensor und Glasfaser
linke Grafik: Messung des ankommenden Lichtes bei eingeschalteter Infrarot-LED
rechte Grafik: Messung des ankommenden Lichtes bei ausgeschalteter Infrarot-LED

In der Abbildung 7.5 werden die gleichen Einstellungen wie aus der Abbildung 7.3 verwendet. Die linke Grafik zeigt die Datenaufnahme des Infrarot-Kanals des Farbsensors. Prozentual liegt die Messungenauigkeit bei dieser Messung bei ca. $\pm 0{,}5\,\%$ und ist somit vernachlässigbar. Im Vergleich zur Plastikfaser ist das Messsignal ungefähr halbiert. Dies ergibt sich aus dem kleineren Durchmesser der Glasfaser, weshalb weniger Licht von der Lichtquelle zum Lichtsensor geleitet wird. Durch eine Erhöhung der Verstärkung würde man auch hier in den Sättigungsbereich kommen. Die Grafik auf der rechten Seite zeigt die Rausch-Messung. Durch den Wert von eins ist ankommendes Restlicht erkennbar, dieses beeinflusst aber aufgrund seiner Amplitude nicht die Messung, es ist somit vernachlässigbar.

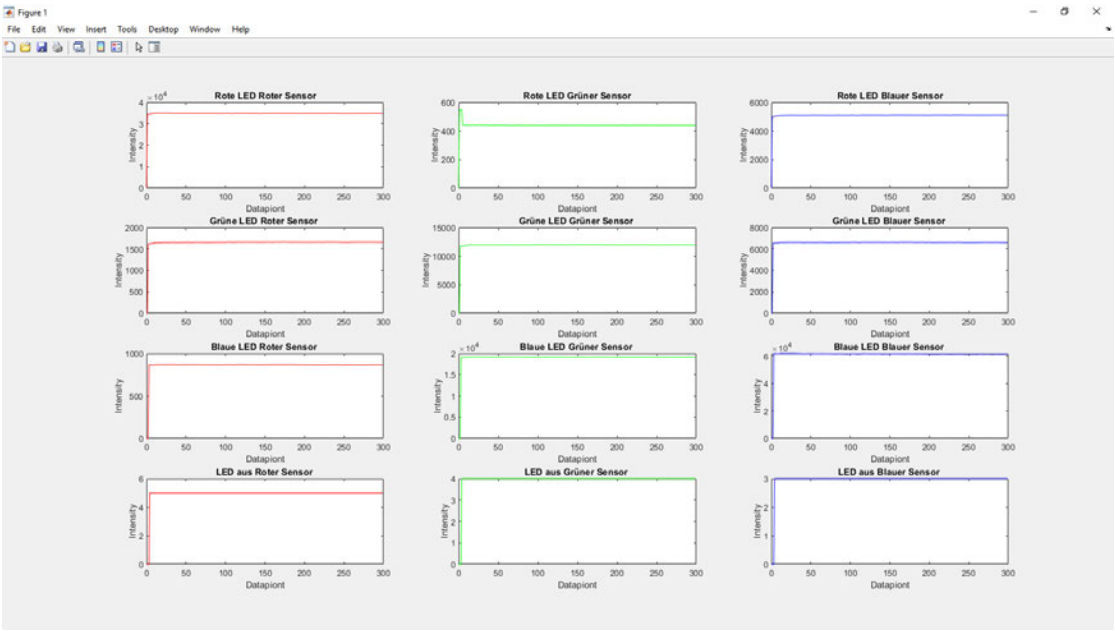


Abbildung 7.6: RGB-Messung mit Farbsensor mit Glasfaser
Die ersten drei Zeilen sind Messungen mit eingeschalteter RGB-LED die letzte Zeile ist die Rauschmessung bei ausgeschalteter RGB-LED.

In der Abbildung 7.6 werden andere Einstellungen des Farbsensors benutzt. Dies liegt an der schlechteren Lichteinkopplung in die Glasfaser. Die Nullen am Anfang der Messungen sind Defaultwerte und treten nur bei Beginn einer neuen Messung auf, da solange gewartet wird, bis die ersten Werte erhalten werden. Die Nullen gehören nicht zur Messungenauigkeit und werden nicht in den Prozentwert eingerechnet. Bei dieser Messung liegt die Messungenauigkeit dann bei ca. $\pm 0{,}5\,\%$ und ist somit vernachlässigbar. Eine Erhöhung der Integrationszeit würde hier auch knapp die Werte in den Sättigungsbereich führen. Es wurden bei den Rauschmessungen Werte von fünf, vier und drei gemessen. Dieses Restlicht ist so klein, dass es die Messungen nicht beeinflusst und vernachlässigt werden kann.

Es sind deutliche Unterschiede zwischen den verwendeten Lichtwellenleitern festgestellt worden. Diese Unterschiede resultieren vor allem aus den unterschiedlichen Durchmessern, die verwendet wurden. Ein größerer Durchmesser des Lichtwellenleiters kann Licht der LED an den Sensor weiterleiten. Will man also den Lichtwellenleiter im Durchmesser verkleinern, muss mehr Leistung von der LED in den Lichtwellenleiter gekoppelt werden. Um die eingekoppelte Leistung zu erhöhen, sind zwei Parameter bei der Auswahl

der Leuchtdioden wichtig. Es kann eine LED mit einer höheren Nennleistung verwendet werden oder der Lichtaustrittswinkel muss verkleinert werden. Eine Kombination aus höherer Nennleistung und kleinerem Lichtaustrittswinkel dürfte eine deutliche Verbesserung bringen. Es muss hier nur darauf geachtet werden, dass Hochleistungs-Leuchtdioden unter Umständen gekühlt werden müssen und ein anderer LED-Treiber verwendet werden muss. Die Position des Lichtwellenleiters über der RGB-LED verändert auch die eingekoppelte Leistung in den Lichtwellenleiter. Dies ist der Bauform bedingt, da die Lichtpunkte nebeneinander liegen. So koppelt die Lichtquelle in der Mitte der RGB-LED mehr Licht in den Lichtwellenleiter als die anderen Lichtquellen. Die RGB-LED sollte gegen einen leistungsfähigeren Typ ausgetauscht werden, da hier dann die Verstärkung des Farbsensors herabgesetzt werden kann. Außerdem sollte darauf geachtet werden, dass alle Lichtquellen möglichst gleichviel Licht in den Lichtwellenleiter einkoppeln.

Messungen mit dem Farbsensorplatine und Biegung der Glasfaser

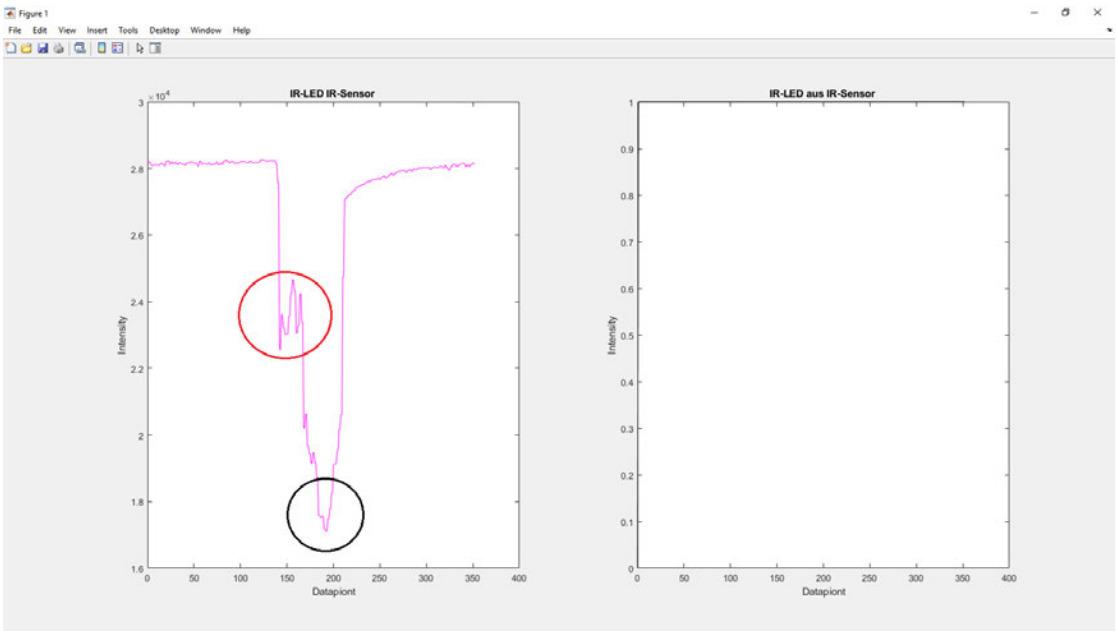


Abbildung 7.7: Infrarot-Messung mit Farbsensor mit schwacher (roter Kreis) und starker Biegung (schwarzer Kreis) der Glasfaser

In der Abbildung 7.7 ist eine Auskopplung des Lichtes in der linken Grafik erkennbar. Der rote Kreis gibt den Bereich an, in dem der Lichtwellenleiter schwach gebogen wurde.

Der schwarze Kreis gibt den Bereich an, in dem der Lichtwellenleiter stark gebogen wurde. Aus der Abbildung 7.7 ist auch zu erkennen, dass bei gleichbleibender Biegung, die Werte in dem gleichen Bereich liegen. Die sprunghaften Ausschläge sind an dem nicht gleichbleibenden Druck beim manuellen Biegen des Lichtwellenleiters zu erklären. Wird die Biegung des Lichtwellenleiters aufgehoben, näheren sich die Werte dem Ausgangszustand erst schnell und dann langsam wieder an. Dies könnte an der Mittelung über fünf Messwerte liegen. Die Welligkeit des Ausgangszustandes ist zu vernachlässigen, da während der Integrationszeit des Farbsensors pro Messzyklus unterschiedlich viele Photonen die Sensorfläche treffen können. Das Signal in der Rauschmessung ist wie in der Abbildung 7.3 so gering im Vergleich zum Nutzsignal, dass es vernachlässigt werden kann.

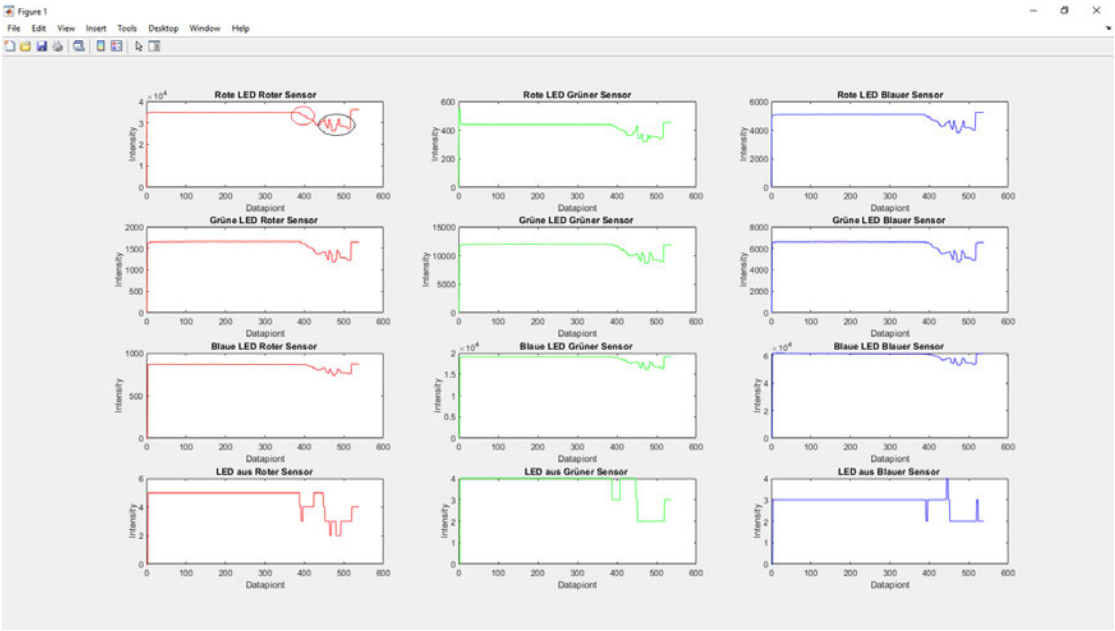


Abbildung 7.8: RGB-Messung mit Farbsensor mit schwacher und starker Biegung der Glasfaser

In der Abbildung 7.8 ist eine Untersuchung zur Lichtauskopplung während einer RGB-LED Messung am Farbsensor gezeigt. Es wurde auch hier eine Auskopplung des Lichtes beim Biegen des Lichtwellenleiters festgestellt. Es wird beispielhaft auf die Grafik oben links eingegangen. Die erste Abschwächung des Signal im roten Kreis zeigt die schwache Biegung der Glasfaser. In dem schwarzen Kreis ist die starke Biegung erkennbar. Die Spitzen bei der starken Biegung sind durch das manuelle Biegen zu erklären. Aus

den aufgezeichneten Kurven sind die rote und grüne Leuchtdiode zu empfehlen, da hier die größten Veränderungen in den Kurven sichtbar sind. In der Rauschmessung ist die Biegung des Lichtwellenleiters zu erkennen. Dies deutet darauf, dass die LED-Steuerung nicht komplett synchron mit der Messung des Farbsensors ist. Durch den sehr geringen Ausschlag der Rauschwerte sind diese nicht gravierend. Es muss trotzdem die Programmierung der LED-Steuerung in einer zukünftigen Arbeit angepasst werden. Die höheren Absolutwerte der Messung sind durch die anderen Einstellung des Farbsensors zu begründen. Die Einstellungen des Farbsensors für die RGB-Messung können nicht für die Infrarot-Messung verwendet werden. Hier würde man sonst in den Sättigungsbereich kommen.

Messungen mit dem Ringoszillatorplatine und der Plastikfaser

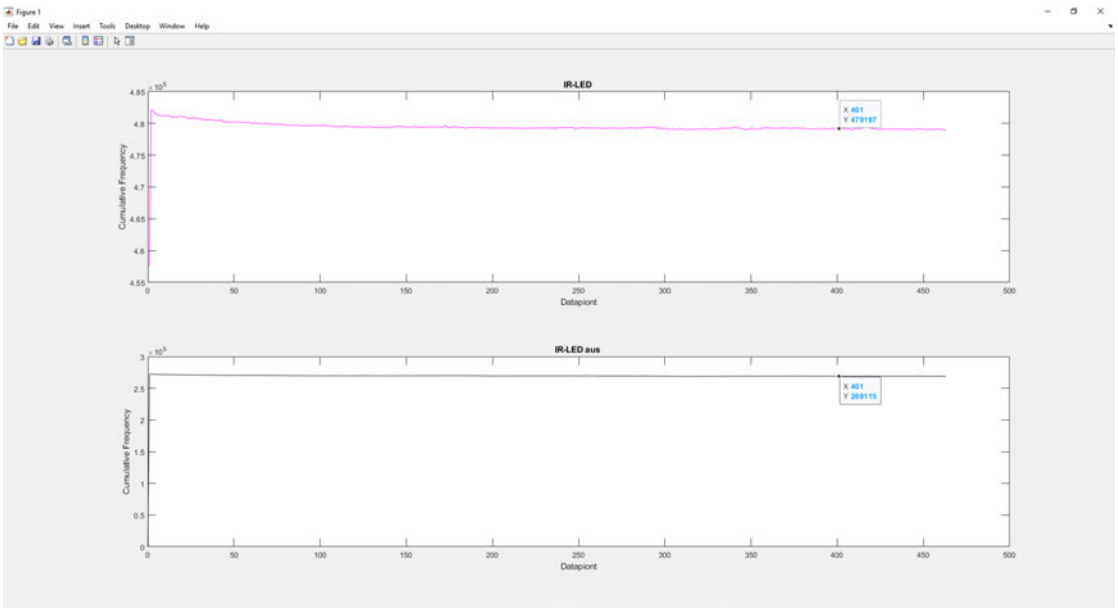


Abbildung 7.9: Infrarot-Messung mit Ringoszillator mit Plastikfaser
obere Grafik: Messung des ankommenden Lichtes bei eingeschalteter Infrarot-LED
untere Grafik: Messung des ankommenden Lichtes bei ausgeschalteter Infrarot-LED

Die folgenden Messungen wurden mit der Ringoszillatorplatine aufgezeichnet. Diese folgen der gleichen Struktur wie die Messungen mit der Farbsensorplatine. In der Abbil-

dung 7.9 ist die Infrarot-Messung mit der Plastikfaser durchgeführt worden. Die verwendeten Einstellungen in dieser Messung sind eine Integrationszeit von 10 s und die Verstärkung auf der Stufe vier. Der Jumper wurde auf die Position Q3 (siehe Abbildung 4.2) gesetzt und für alle nachfolgenden Messungen nicht verändert. Die obere Grafik zeigt die Messung mit eingeschalteter Infrarot-LED. Dieses System benötigt eine gewisse Zeit bis es eingeschwungen ist, dies ist der Fall nach ca. 150 Messpunkten. Der Mittelwert für die Infrarot-Messung liegt um ca. 210 kHz höher als das Rauschen.

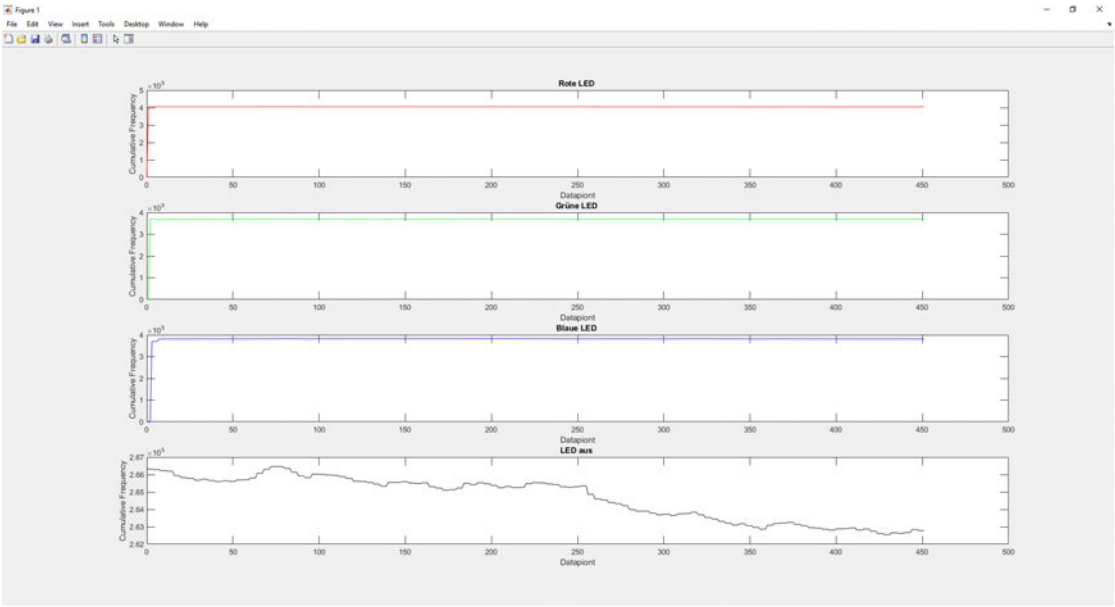


Abbildung 7.10: RGB-Messung mit Ringoszillator mit Plastikfaser
ersten drei Grafiken: Messung des ankommenden Lichtes bei eingeschalteter RGB-LED
rechte Grafik: Messung des ankommenden Lichtes bei ausgeschalteter RGB-LED

In der Abbildung 7.10 ist zu erkennen, dass die sich die Frequenz erhöht bei eingeschalteter LED. Der Wert für die rote LED liegt um der Mittelwert ca. 140 kHz höher als das Rauschen. Für die grüne LED liegt der Mittelwert bei ca. 120 MHz und für die blaue LED ca. 130 khz höher als bei der Rauschmessung. Die unterschiedlichen Werte kommen durch die Empfindlichkeit des Fototransistors und der Position der Lichtquelle innerhalb der RGB-LED zustande. Der Fototransistor hat die höchste Empfindlichkeit bei einer Wellenlänge von 850 nm, die Empfindlichkeit nimmt zu beiden Seiten ab. In der Rauschmessung ist eine Abnahme der Frequenz erkennbar. Diese Abnahme kann durch

Umgebungslicht kommen, dass auf den Fototransistor kommt. Die Abdeckung kann nicht alles Licht ausschließen, da diese unbefestigt über die Messaufbauten gelegt wurde.

Messungen mit dem Ringoszillatorplatine und der Glasfaser

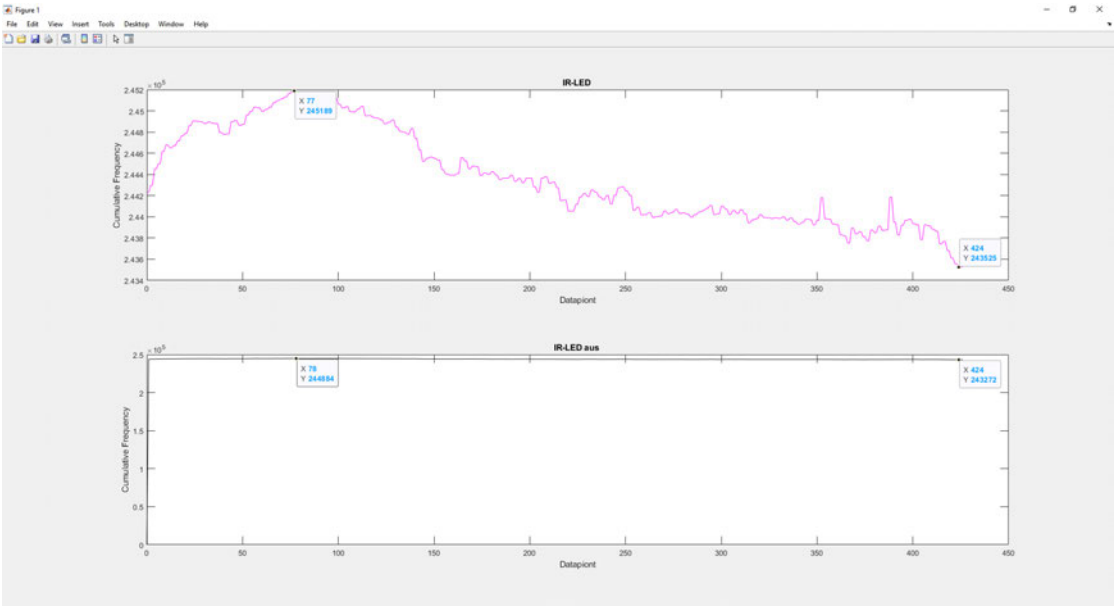


Abbildung 7.11: Infrarot-Messung mit Ringoszillator mit Glasfaser
obere Grafik: Messung des ankommenden Lichtes bei eingeschalteter Infrarot-LED
untere Grafik: Messung des ankommenden Lichtes bei ausgeschalteter Infrarot-LED

In der Abbildung 7.11 ist zu erkennen, dass für die Ringoszillatorplatine die Lichtleistung über die Glasfaser zu gering für den Fototransistor ist. In der oberen Grafik ist ein Abfallen der Kurve erkennbar. Dieser Abfall kann wie bei der Abbildung 7.10 durch Umgebungslicht kommen. Um die Unterschiede in den Werten von LED und Rauschen zu ermitteln, wurden jeweils zwei Messpunkte ausgewählt und die Werte verglichen. Aus diesem Vergleich konnte ein Werteunterschied von ca. 300 Hz festgestellt werden. Dieser Messunterschied könnte eine Messungenauigkeit sein. Wenn es keine Messungenauigkeit ist, beruht dieser Messwertunterschied auf der zugeführten Lichtleistung. Mit dem Unterschied von 300 Hz sind quantitative Aussagen über die ankommende Lichtleistung schwer zu treffen.

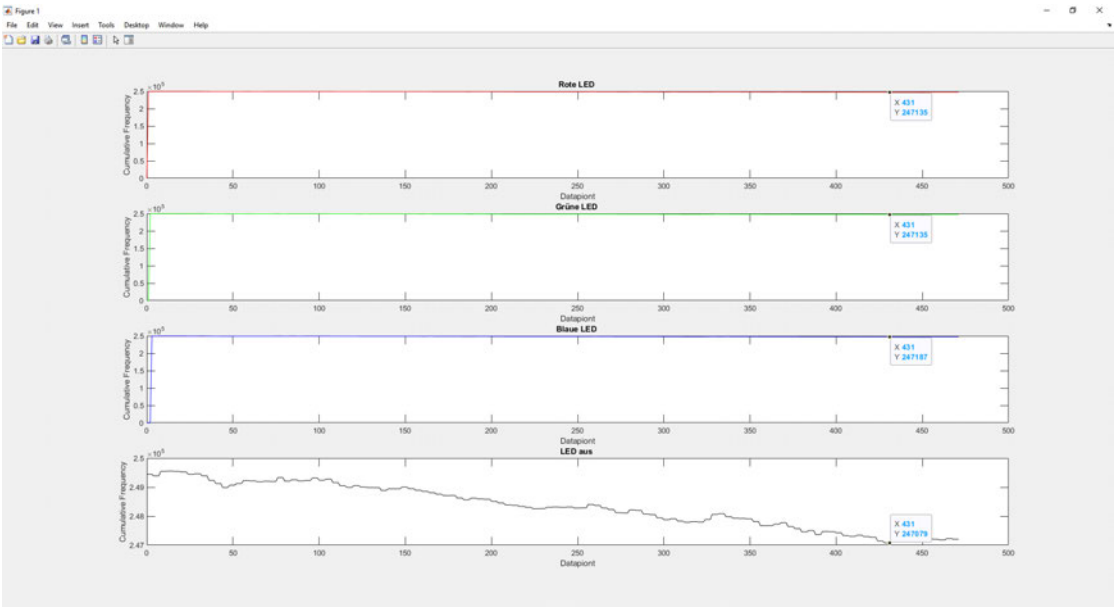


Abbildung 7.12: RGB-Messung mit Ringoszillator mit Glasfaser
ersten drei Grafiken: Messung des ankommenden Lichtes bei eingeschalteter RGB-LED
rechte Grafik: Messung des ankommenden Lichtes bei ausgeschalteter RGB-LED

In der Abbildung 7.11 ist zu erkennen, dass für den Ringoszillator die Lichtleistung über die Glasfaser zu gering für den Fototransistor ist. Die Unterschiede in den Werten von mit LED und Rauschen von ca. 100 Punkten kann eine Messungenauigkeit sein. Ist dies keine Messungenauigkeit ist dieser Messwerte unterschied zu gering um gute Aussagen über die ankommende Lichtleistung treffen zu können. Die Verringerung von 200 Punkten zu der Infrarot-Messung kann auch hier in der Empfindlichkeit des Fototransistors oder in der Lage der Glasfaser begründet werden. Eine weitere Begründung könnte die geringere Leistung der RGB-LED sein.

Messungen mit dem Ringoszillatorplatine und Biegung der Glasfaser

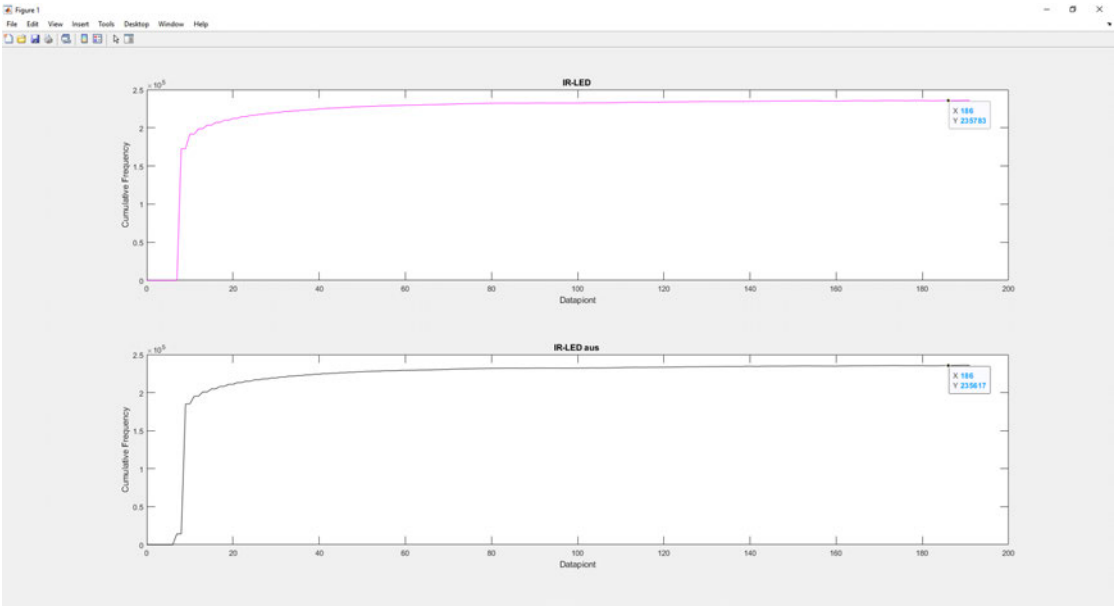


Abbildung 7.13: RGB-Messung mit Ringoszillator mit starker Biegung der Glasfaser

In der Abbildung 7.13 ist keine Veränderung in der Kurve zu erkennen, die auf eine Auskopplung durch Biegen des Lichtwellenleiters zurückzuführen ist. Dies liegt, wie oben beschrieben, an der zu geringen Lichtleistung.

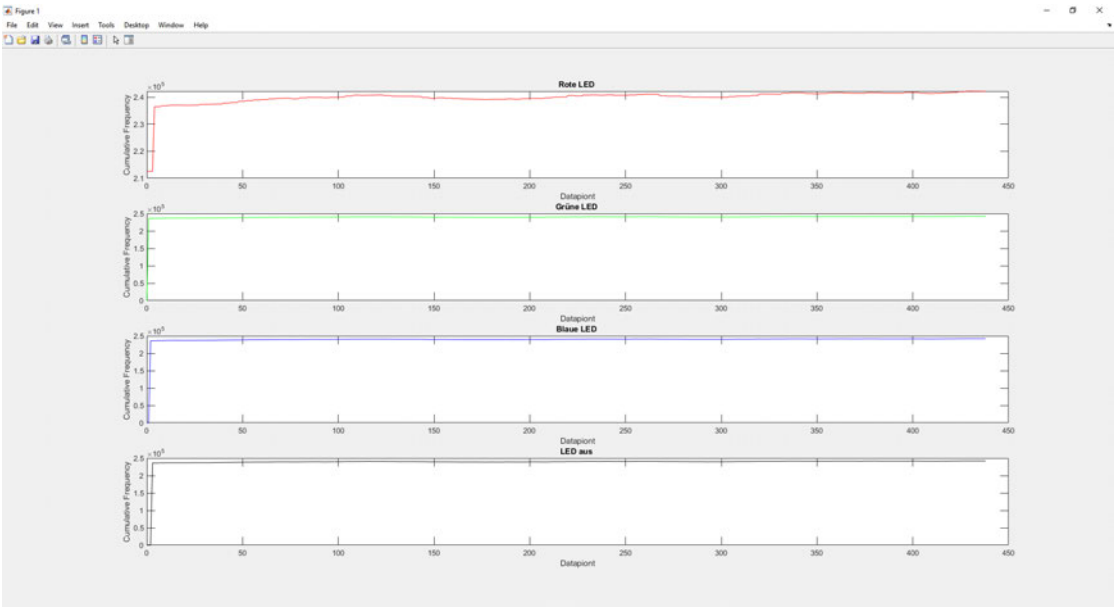


Abbildung 7.14: RGB-Messung mit Ringoszillator mit starker Biegung der Glasfaser

In der Abbildung 7.14 ist keine Veränderung in der Kurve zu erkennen, die auf eine Auskopplung durch Biegen des Lichtwellenleiters zurückzuführen ist. Dies liegt, wie oben beschrieben, an der zu geringen Lichtleistung. Die Dellen in der roten Kurve können vernachlässigt werden, da die Skalierung der Grafik hier bei $2{,}5\text{x}10^5$ anfängt. Die anderen Grafiken beginnen bei 0 als Defaultwert. In einer zukünftigen Arbeit müssen hier die Grafiken so angepasst werden, dass nur relevante Kurvenausschnitte sichtbar sind.

Aus allen Messungen kann geschlossen werden, dass das Messprinzip der Ringoszillatorplatine generell funktioniert. Dies ist an den statischen Messungen mit der Plastikfaser zu erkennen. Nach der Verringerung des Durchmessers des Lichtwellenleiters konnten keine verwertbaren Messergebnisse erzeugt werden. Die Verstärkung war bei allen Messungen auf der maximalen Stufe. Die Integrationszeit von 10 s ist fast um den Faktor 32 größer als bei dem Farbsensor. Eine weitere Erhöhung der Integrationszeit ist möglich, aber nicht zielführend, da dann nur der Mittelwert aus einem größeren Datensatz gebildet wird.

7.3 Auswertung von Einflussgrößen und Streuungen, Kalibrierung bzw. Referenzmessungen

Für die Berechnung des Bitabschätzung werden gemittelte Werte aus den Grafiken visuell entnommen. Dies reicht aus, um einen Überblick auf die Signalleistung der Messsysteme zu erhalten. Es wird die Formel $Bitauflösung = log_2(Licht - Rauschen)$ für die Berechnung genutzt.

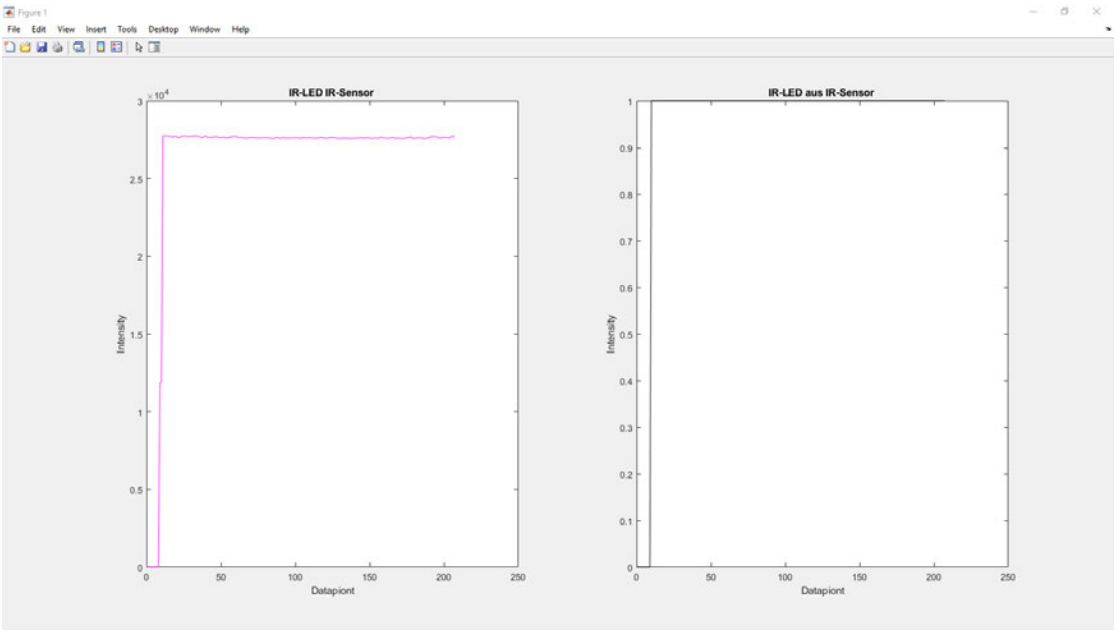


Abbildung 7.15: Infrarot-Messung mit Farbsensor ohne Abdeckung mit Glasfaser

Als erstes wurde ist die Farbsensorplatine mit der Glasfaser, Infrarot-LED und ohne Abdeckung vermessen. Die ausgewählten Werte in dieser Messung sind 27500 und 1, daraus ergibt sich eine Bitauflösung von 14.7 Bit. Mit dieser Bitauflösung beeinflusst das Rauschen nicht die Messung. Die Infrarot-Messung der Farbsensorplatine ist somit gegen Umgebungslicht nicht beeinflussbar.

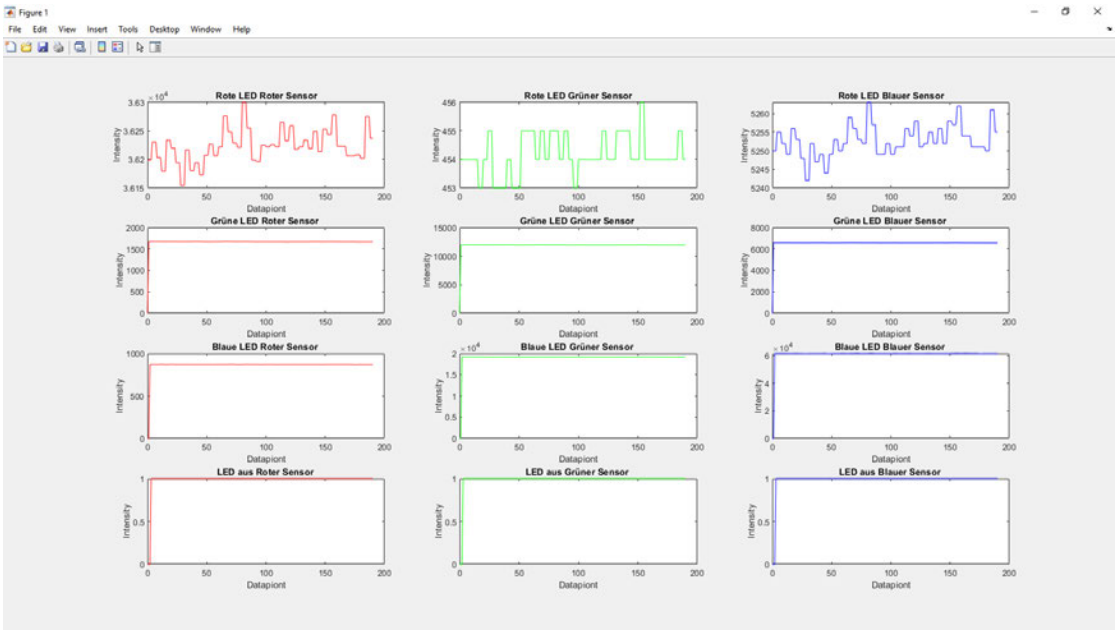


Abbildung 7.16: RGB-Messung mit Farbsensor ohne Abdeckung mit Glasfaser

Zur Berechnung der Bitauflösung werden bei der RGB-Messung der Farbsensorplatine, jeweils die Werte aus der Übereinstimmung von LED-Farbe und Sensor-Empfindlichkeit verwendet. Hier können die höchsten Bitauflösungen erreicht werden. Für die rote LED sind es die Werte 36225 und 1, daraus ergibt sich eine Auflösung von 15.1 Bit. Für die grüne LED sind es die Werte 12500 und 1, daraus ergibt sich eine Auflösung von 13.6 Bit. Für die blaue LED sind es die Werte 61000 und 1, daraus ergibt sich eine Bitauflösung von 15.9 Bit. Mit diesen Auflösungen beeinflusst das Rauschen nicht die Messung, somit ist aus die RGB-Messung der Farbsensorplatine nicht vom Umgebungslicht beeinflussbar.

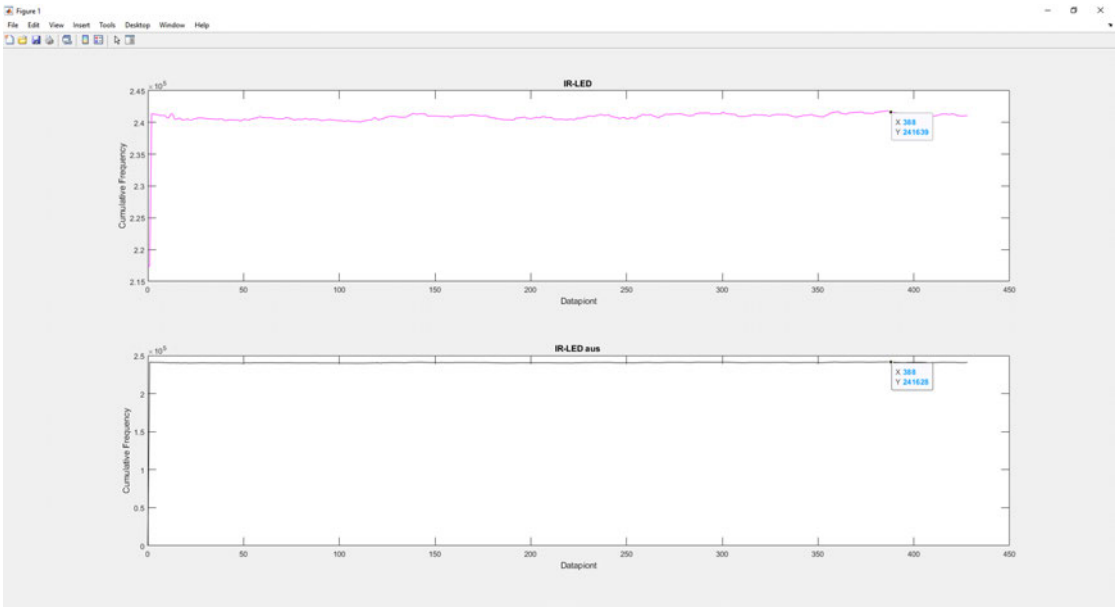


Abbildung 7.17: Infrarot-Messung mit Ringoszillator ohne Abdeckung mit Glasfaser

Für die Messung der Ringoszillatorplatine mit Glasfaser, Infrarot-LED und ohne Abdeckung, wurde ein Wert aus beiden Grafiken entnommen. Die ausgewählten Werte in dieser Messung sind 241639 und 241628, daraus ergibt sich eine Bitauflösung von 3.4594 Bit. Diese Bitauflösung ist sehr klein und es lässt sich daraus schließen, dass das Umgebungslicht die Messung beeinflussen kann.

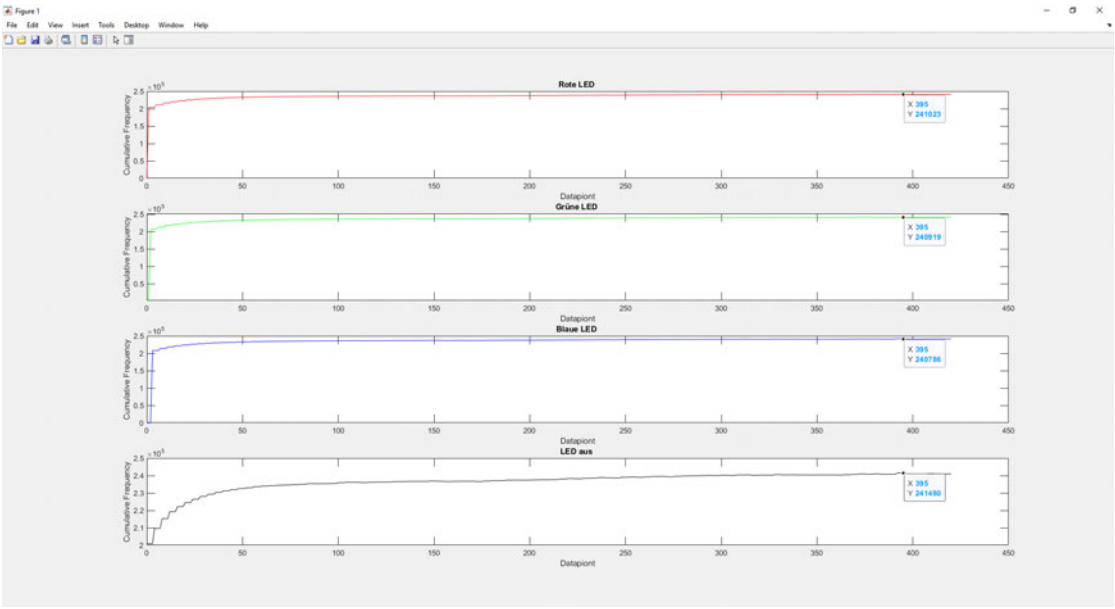


Abbildung 7.18: RGB-Messung mit Ringoszillator ohne Abdeckung mit Glasfaser

Zur Berechnung der Bitauflösung werden bei der RGB-Messung der Ringoszillatorplatine, vier Werte aus den Grafiken entnommen. Für die rote LED sind es die Werte 241023 und 241480, daraus ergibt sich eine Auflösung von 8.8361 + 4.5324i Bit. Für die grüne LED sind es die Werte 240919 und 241480, daraus ergibt sich eine Bitauflösung von 9.1319 + 4.5324i Bit. Für die blaue LED sind es die Werte 240786 und 241480, daraus ergibt sich eine Bitauflösung von 9.4388 + 4.5324i Bit. Die Bitauflösungen sind hier nicht zu gebrauchen, da das Rausch-Signal größer ist als das ankommende Licht-Signal und sich komplexe Werte ergeben.

Die Werte für die Berechnungen der Bitauflösung sind aus den Abbildungen 7.3, 7.4, 7.9 und 7.10 entnommen. Die Werte für die Abbildung 7.3 sind 59500 und 1. Dies ergibt eine Bitauflösung von 15.9 Bit. Für die Abbildung 7.4 sind es die Werte 59745 und 0 für Rot, 59760 und 0 für Grün und 59760 und 0 für Blau. Aus diesen Werten folgen die Bitauflösungen von 15.9 Bit, 15.9 Bit und 15.9 Bit. Für die Ringoszillatorplatine sind die Werte aus der Abbildung 7.9 479197 und 269115. Hieraus ist die berechnete Bitauflösung 17.7 Bit. Für die RGB-LED sind die folgenden Werte entnommen worden für Rot 410000, für Grün 366667, für Blau 366667 und als Rauschmessung 262750. Die errechneten Bitauflösungen sind für Rot 17.1 Bit, für Grün 16.7 Bit und für Blau 16.7 Bit.

Aus den Berechnungen der Bitauflösung zeigt sich, dass die Farbsensorplatine sowohl mit der Plastikfaser als auch der Glasfaser eine hohe Auflösung erreicht. Die Ringoszillatorplatine erreicht eine höhere Auflösung bei der Plastikfaser als die Farbsensorplatine. Bei der Glasfaser sind die Bitauflösungen nicht anwendbar, da sie sehr kleine oder komplexe Werte ergeben. Als Erkenntnis können beide Platinen hohe Bitauflösungen mit der Plastikfaser erreichen. Dies zeigt die Messkonzepte funktionieren generell. Für die Glasfaser hat die Farbsensorplatine ebenfalls eine ausreichend hohe Bitauflösung. Die Ringoszillatorplatine ist für die Glasfaser momentan noch nicht brauchbar. Die Gründe könnten die Einkopplung des Lichtes oder die Erfassung des Lichtes durch den Sensor sein. Es müssen hier weitere Untersuchungen durchgeführt werden.

Platine	LED	theoretische Bitauflösung	reelle Bitauflösung
Farbsensorplatine	Infrarot	15.9 Bit	15 Bit
Ringoszillatorplatine	Infrarot	17.7 Bit	17 Bit
Farbsensorplatine	Rot	15.9 Bit	15 Bit
Ringoszillatorplatine	Rot	17.2 Bit	17 Bit
Farbsensorplatine	Grün	15.9 Bit	15 Bit
Ringoszillatorplatine	Grün	16.7 Bit	16 Bit
Farbsensorplatine	Blau	15.9 Bit	15 Bit
Ringoszillatorplatine	Blau	16.7 Bit	16 Bit

Tabelle 7.2: Übersicht der ermittelten Bitauflösungen für die Plastikfaser

Platine	LED	theoretische Bitauflösung	reelle Bitauflösung
Farbsensorplatine	Infrarot	14.7 Bit	14 Bit
Ringoszillatorplatine	Infrarot	3.5 Bit	3 Bit
Farbsensorplatine	Rot	15.1 Bit	15 Bit
Ringoszillatorplatine	Rot	n. a.	n. a.
Farbsensorplatine	Grün	13.6 Bit	13 Bit
Ringoszillatorplatine	Grün	n. a.	n. a.
Farbsensorplatine	Blau	15.9 Bit	15 Bit
Ringoszillatorplatine	Blau	n. a.	n. a.

Tabelle 7.3: Übersicht der ermittelten Bitauflösungen für die Glasfaser

tabelle glasfaser ring screiben nicht möglich tabelle plastik faser

7.4 Untersuchung des Aufbaus auf Reproduzierbarkeit

Die hier gemessenen Ergebnisse müssen auf einer zweiten Platine bestätigt werden. Diese Platine wurde angefertigt, konnte aber leider nicht mehr vermessen werden. Es sind aber keine großen Abweichungen zum ersten Prototypen zu erwarten. Die möglichen Abweichungen könnten sich aus vier bauteilspezifischen Eigenschaften ergeben. Zum einen kann die ausgestrahlte Wellenlänge der LED verschieden sein. Als weiterer Faktor kann die zur Verfügung stehende Stromstärke des LED-Treibers variieren. Dies hat einen Einfluss auf die Lichtleistung der LED. Dann gibt es noch zwei Eigenschaften des Farbsensors, die variieren können. Dies sind die Empfindlichkeit auf die gemessene Wellenlänge sowie der ADC, welcher leicht unterschiedlich die analogen Werte interpretieren kann. All diese Bauteileigenschaften können jedoch durch eine Anpassung der Verstärkung und Integrationszeit des Farbsensors kompensiert werden.
Für die Ringoszillatorplatine steht zur Zeit kein zweiter Prototyp zur Verfügung. Auch hier müsste eine zweite Platine aufgebaut werden, um die gemessenen Ergebnisse zu validieren. Hier sind die möglichen Unterschiede zur ersten Platine wie oben erwähnt auch die Eigenschaften der Leuchtdioden und des LED-Treibers. Weitere Unterschiede sind die Laufzeiten der Gatter des Ringoszillators sowie die Empfindlichkeit des Fototransistors. Eine Anpassung wäre hier durch das Umstecken des Jumpers und eine Verlängerung der Integrationszeit möglich. Ein weiterer Einfluss bei der Ringoszillatorplatine ist die Temperatur. Die Ringoszillatorplatine benötigt mindestens 15 min um auf Betriebstemperatur zu sein, erst dann kann die erste Messung erfolgen.

8 Einordnung, Bewertung und Ausblick

8.1 Zusammenfassung der Ergebnisse, Beurteilung des Messkonzeptes

Um die Transmission einer Batteriezelle zu vermessen, müssen Lichtwellenleiter in der Batteriezelle verbaut werden. Die Schaltung soll möglichst platzsparend sein, da sie für die Elektromobilität gedacht ist. Hierfür wurden zwei Aufbauten entwickelt. Der erste Aufbau hat als Sensor einen Farbsensor, welcher das ankommende Licht in einen digitalen Wert wandelt. Der zweite Aufbau hat als Sensor einen Fototransistor, welcher über eine Spannungsänderung die Frequenz eines Ringoszillators ändert.
Für die Messungen werden zwei unterschiedliche Lichtwellenleiter genutzt. Eine Plastikfaser mit einem großen Kerndurchmesser, um das Funktionsprinzip zu überprüfen. Als zweites wird eine Glasfaser mit kleinerem Kerndurchmesser verwendet, um näher am künftigen Aufbau zu sein.
Die Messungen mit dem Plastikfaser haben ergeben, dass beide Sensoren für diese Art von Messkonzept erstmal geeignet ist. Wie oft schon erwähnt, wird durch einen geringeren Durchmesser des Lichtwellenleiters die eingekoppelte Lichtleistung verringert. Hierfür müssen die gleichen Messungen mit der Glasfaser durchgeführt werden. Es kamen außerdem noch weitere Messungen hinzu. Diese dienten der Überprüfung, ob das ausgekoppelte Licht gemessen werden kann bzw. ob einfallendes Umgebungslicht die Sensoren beeinflusst.
Um einen Vergleich zwischen den verschiedenen Sensorkonzepten zu erhalten ist die Bitauflösung ein gutes Mittel, um festzustellen wie fein das empfangene Licht vermessen werden kann. Aus den Messungen der Bitauflösung wurde fest gestellt, dass die Ringoszillatorplatine nicht mit der Glasfaser verwendbare Ergebnisse liefert. Es kann hier zwischen Signal und Rauschen nicht unterschieden werden. Für die Messungen mit der Plastikfaser konnte ein hohe Auflösung erreicht werden. Die Farbsensorplatine hat beiden Messung eine Auflösung erreicht, die gut ist um diesen Prototypen weiter zu verfolgen.

Die wiederholten Messungen mit der Glasfaser haben gezeigt, dass bei der Ringoszillatorplatine keine sichtbaren Veränderungen in den Messwerten erkennbar sind. Die Messungen für die Auskopplung des Lichtes zeigen, dass bei der Farbsensorplatine gemessen werden kann, wie viel Licht aus dem Lichtwellenleiter ausgekoppelt wird. Es sind aber auch hier unterschiedliche Einstellung nötig, um die Empfindlichkeit des Sensor anzupassen. Das gesamte Messkonzept kann kann für zukünftige Prototypen auf den dünnen Lichtwellenleiter beschränkt werden, da hier alle Fälle abgedeckt werden.

8.2 Bewertung der gewählten Konzepte und Lösungsvarianten

Es sind mit beiden Platinen Messergebnisse zu erhalten. Diese sind aber von unterschiedlicher Güte. Zuerst wird die Farbsensorplatine betrachtet. Die Farbsensorplatine liefert in allen Fällen verwertbare Messergebnisse, um Aussagen aus einer Veränderung des Lichtes zu erhalten. Dies liegt auch daran, dass die Bitauflösung ausreichend groß ist. Um eine Leistungsverbesserung bei der Farbsensorplatine zu bekommen könnten leistungsfähigere Leuchtdioden verwendet werden. Durch die leistungsfähigeren Leuchtdioden könnten die Integrationszeit und die Verstärkung bei der Farbsensorplatine verringert werden.
Die Kommunikation ließ sich durch den I^2C-Bus zwischen dem Mikrocontroller und dem Farbsensor einfach implementieren. Einstellungen an Integrationszeit oder Verstärkung sind durch vorgegeben Befehlssatz schnell und einfach umsetzbar. Die Datenabfrage des Farbsensors ist ebenfalls unkompliziert, da hier ein digitaler Wert aus den Registern des Farbsensors gelesen wird.
Für die Darstellung der digitalen Werte in Matlab werden über die UART-Schnittstelle die digitalen Werte von der Platine an den Computer gesendet. Die Erzeugung der Grafiken bzw. die Steuerung der Farbsensorplatine in Matlab konnte mit einfachen Befehlen umgesetzt werden. Die maximale Anzahl an gleichzeitigen Grafiken, die für die Farbsensorplatine möglich sind, beträgt zwölf.
Die Ringoszillatorplatine liefert nur in den Messungen mit der Plastikfaser verwertbare Ergebnisse. Diese Ergebnisse zeigen, dass die Ringoszillatorplatine generell funktioniert. In allen Messungen mit der Glasfaser eignet sich die Ringoszillatorplatine nicht. In diesen Messungen ist selbst bei einer Integrationszeit von 10 s ein kaum erkennbarer Unterschied zum Rauschen vorhanden ist. Dies kann mehrere Gründe haben. Der Fototransistor, der gewählt wurde, ist nicht geeignet für diese Art von Messkonzept, da er bei der Glas-

faser zu wenig Strom liefert. Ein weiterer Grund kann die zu geringe Lichtleistung der Leuchtdioden für die Glasfaser sein. Als Lösung sind hier andere Bauteile möglich. Der Fototransistor könnte durch einen ersetzt werden, der bei Lichteintrag einen höheren Strom liefert, als der verwendete Fototransistor vom Typ SFH 300-3/4. Bei den Leuchtdioden könnten leistungsfähigere ausgewählt werden.
Die durch den Ringoszillator erzeugte Frequenz wird durch einen Pin-Interrupt ermittelt. Dieser Wert wird über die eingestellte Integrationszeit aufaddiert und ein Mittelwert erstellt. Dieser Mittelwert wird dann über die UART-Schnittstelle an den Computer gesendet. Dieses Konzept ist einfach, da die Frequenz über über eine Sekunde ermittelt wird und so keine weiteren Berechnungen zur Ermittlung der Frequenz benötigt werden. Die Befehlsstruktur musste für die Ringoszillatorplatine selbst entwickelt werden, es konnte keine vorhandene Befehlsstruktur verwendet werden.
Es sind fast die gleichen Befehle für die Matlab-Oberfläche verwendet wie für die Farbsensorplatine. Einzig in den Grafiken unterscheiden sich die Platinen, was aber kein Vor- oder Nachteil ist. Die maximale Anzahl an gleichzeitigen Grafiken, die für die Ringoszillatorplatine möglich sind, beträgt vier.

Beide Konzepte funktionieren mit der Plastikfaser. Die Farbsensorplatine ist momentan zu bevorzugen, da diese auch mit der Glasfaser verwendbare Ergebnisse liefert. Mit den genannten Änderungen könnte auch die Ringoszillatorplatine mit der Glasfaser verwendbare Ergebnisse liefern.

8.3 Offene Punkte und einschränkende Erfahrungen und Ausblick

Offene Punkte

Es war eine Untersuchung mit unterschiedlichen Ethanolkonzentrationen angedacht. Es sollte der dünne Lichtwellenleiter um eine Sonde gewickelt werden. Der Radius dieser Wicklung wurde aus Altarbeiten übernommen. Dieser Radius erzeugt eine gleichmäßige Auskopplung des Lichtes über die komplette Länge des Lichtwellenleiter. Es wird so sichergestellt, dass nur die Änderung der Ethanolkonzentrationen eine Veränderung der Messkurve erzeugt. Aus zeitlichen Gründen wurde dieser Versuch nicht durchgeführt und sollte in zukünftigen Arbeiten nachgeholt werden. Es müssen die Messungen jeweils

mit mindestens einem weiteren Prototypen überprüft werden. Dies ist wichtig, um die Variation zwischen den Platinen festzustellen zu können.

Einschränkende Erfahrungen

Es musste viel an der Lichteinkopplung in die Lichtwellenleiter probiert werden. Die Lichtleistung viel in den ersten Versuchen fiel zu gering aus. Es wurden z.B. Leuchtdioden und Fototransistoren angebohrt um einen Kanal für die Glasfaser zu schaffen. Es sollte so durch einen geringeren Abstand zur Lichtquelle die Leistung erhöht werden. Eine weitere Einschränkung war es, dass für zwei Platinen Programmcode geschrieben und zwei Platinen jeweils eingestellt werden mussten. Die größte Einschränkung war der Lockdown wegen des Corona-Virus. Dies hinderte mich an der HAW die Arbeit fertigzustellen.

Ausblick

Für zukünftige Arbeiten können leistungsstärkere Leuchtdioden ausgesucht werden. Diese sollte im Optimalfall einen kleineren Leuchtwinkel und die gleichen Wellenlängen haben wie die Leuchtdioden aus dieser Arbeit. Durch die Erhöhung der Leistung und Verkleinerung des Leuchtwinkels wird mehr Leistung auf einen Punkt konzentriert.
Es kann unter Umständen eine Linse zum Zentrieren des Lichtkegels der Leuchtdiode verwendet werden, wodurch das so ausgestrahlte Licht auf einem Punkt konzentriert wird. In diesem Brennpunkt sitzt dann der Lichtwellenleiter und es wird mehr Lichtleistung eingekoppelt.
Die Lichtwellenleiter-Adapter sollten aus einem lichtundurchlässigem Material gefertigt werden, da hier der Eintrag von Umgebungslicht ausgeschlossen werden kann. Werden diese aus Metall gefertigt, können die Toleranzen gering gehalten werden, was eine Positionierung des Lichtwellenleiters erleichtert.
Es können auch Leuchtdioden verwendet werden, die schon auf eine Einkopplung des Lichtes in einen Lichtwellenleiter optimiert sind, so werden viele der oben genannten Punkte ausgeschlossen.
Es kann bei der Ringoszillatorplatine ein anderer Fototransistor verwendet werden, der einen höheren Strom für den Ringoszillator liefert. Für die Farbsensorplatine könnte man weitere Farbsensoren ausprobieren, um zu gucken, ob gravierende Leistungsunterschiede zwischen diesen existieren. Da die Platine in ferner Zukunft integriert werden soll wäre

es gut, die Abmessungen dieser zu verkleinern, um so wenig Platz wie möglich zu nutzen. Softwareseitig kann das Matlabprogramm verbessert werden. Es kann hier eine Oberfläche geschrieben werden, die über Buttons und Regler die Verbindung und die Einstellungen der Sensoren vornimmt. Dies wäre effektiver als das vorhandene Matlabprogramm. Als letztes sollte man testen, ob der Sensor wirklich die unterschiedlichen Ladezustände einer Batteriezelle erkennt. Eine Auswertung dieser Werte könnte dann auch in dem Matlabprogramm implementiert werden.

Literaturverzeichnis

[1] Elektronik-Kompendium.de. Glasfaser / Lichtwellenleiter (LWL), 11/08/2021. https://www.elektronik-kompendium.de/sites/kom/0301282.htm.

[2] Universität Wien. Brechung zum Lot, 11/08/2021. https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/strahlenoptik/2b_-brechung_zLot.htm.

[3] Universität Wien. Snelliussches Brechungsgesetz, 11/08/2021. https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/strahlenoptik/2a_-brechungsgesetz.htm.

[4] Universität Wien. Sonderfall: Totalrefelxion, 11/08/2021. https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/strahlenoptik/2d_totalrefelxion.htm.

[5] Hans-Peter Willig. Lichtwellenleiter, 11/08/2021. https://physik.cosmos-indirekt.de/Physik-Schule/Lichtwellenleiter.

[6] Elektronik-Kompendium.de. Lithium-Ionen-Akkus, 11/08/2021. https://www.elektronik-kompendium.de/sites/bau/0810281.htm.

[7] Kompetenznetzwerk Lithium-Ionen-Batterien e. V. (KLiB). Lithium-Ionen-Batterien, 11/08/2021. https://www.batterieforum-deutschland.de/infoportal/batterie-kompendium/sekundaere-batterie/metall-ionen-batterien/lithium-ionen-batterien/.

[8] Kompetenznetzwerk Lithium-Ionen-Batterien e. V. (KLiB). Separator, 11/08/2021. https://www.batterieforum-deutschland.de/infoportal/lexikon/separator/.

[9] Thomas Tille (Hrsg.). *Automobil-Sensorik 3.* Springer-Verlag GmbH Deutschland, 07/04/2020. Kapitel 5 Ansätze der optischen Zustandsbestimmung in Lithium-Ionen-Batterien für die Nutzung in Elektro-Fahrzeugen, Autoren: Florian Rittweger, Christian Modrzynski1, Valentin Roscher,Karl-Ragmar Riemschneider.

[10] Wahid Nasimzada. Hard- und Softwareentwicklung eines Lichtleiter-Sensors für die optische Analyse des Elektrolyten von Bleibatterien. Bachelor thesis, Hochschule für angewandte Wissenschaften Hamburg, 11/2013.

[11] Henrik Klockmann. Optische Spektralanalyse der Elektroden von Lithiumbatterien mit Lichtleitern. Bachelor thesis, Hochschule für angewandte Wissenschaften Hamburg, 09/2013. https://reposit.haw-hamburg.de/handle/20.500.12738/6313.

[12] Arm Limited. Common Microcontroller Software Interface Standard (CMSIS), 11/08/2021. https://developer.arm.com/tools-and-software/embedded/cmsis.

[13] SmartLight GmbH. Das sind die 4 gängigsten LED-Bauformen, 11/08/2021. https://www.ledlager.de/blog/led-bauformen-welche-es-gibt-und-wofur-sie-gemacht-sind/.

[14] Wikipedia. Leuchtdiode, 11/08/2021. https://de.wikipedia.org/w/index.php?titleLeuchtdiode&oldid214409029.

[15] Elektronik-Kompendium.de. Fototransistor, 11/08/2021. https://www.elektronik-kompendium.de/sites/bau/0207012.htm.

[16] Wikipedia. Fototransistor, 11/08/2021. https://de.wikipedia.org/w/index.php?titleFototransistor&oldid194876116.

[17] Jirka Weissgärber. Die UART Schnittstelle, 11/08/2021. http://www.mathe-mit-methode.com/schlaufuchs_web/elektrotechnik/mikrocontroller_lernmaterial/microcontroller_allgemein/mikrocontroller_ext_hardware/mikrocontroller_-uart.html.

[18] Enrique Fernandez. I2C-Bus, 11/08/2021. http://fmh-studios.de/theorie/informationstechnik/i2c-bus/.

[19] Verbundprojekt LImeS Vorhabensbeschreibung - BMWi Projekt, 2020-2022. FKZ 03ETE019B.

[20] Amphenol Fiber Optic Products. *Receptacle Flange Mount Technische Zeichnung*, 03/2011.

[21] ams AG. Color Sensors Product List, 06/2021. https://ams.com/color-sensors.

[22] ams AG. *TCS3400 Color Light-to-Digital Converter Datenblatt*, v1-06 edition, 10/2017.

[23] Kingbright. *WP7113SF7C T-1 3/4 (5mm) Infrared Emitting Diode Datenblatt*, v.5b edition, 10/2019.

[24] Ltd Luckylight Electronics Co. *LL-509RGBC2E-006 Datenblatt*, n.a.

[25] OSRAM Opto Semiconductors GmbH. *Silicon NPN Phototransistor SFH 300 Datenblatt*, 1.3 edition, 01/2016.

[26] STMicroelectronics International N.V. *Positive voltage regulator ICs Datenblatt*, 36 edition, 09/2018.

[27] Toshiba Europe Limited. *TC74ACT240P/TC74ACT240F/TC74ACT240FT, TC74ACT244P/TC74ACT244F/TC74ACT244FT Datenblatt*, 03/2014.

[28] Infineon Technologies AG. *XMC1100 AB-Step Datenblatt*, v1.8 edition, 09/2016. Kapitle 5 Seite 6.

A Anhang

A.1 Schaltplan

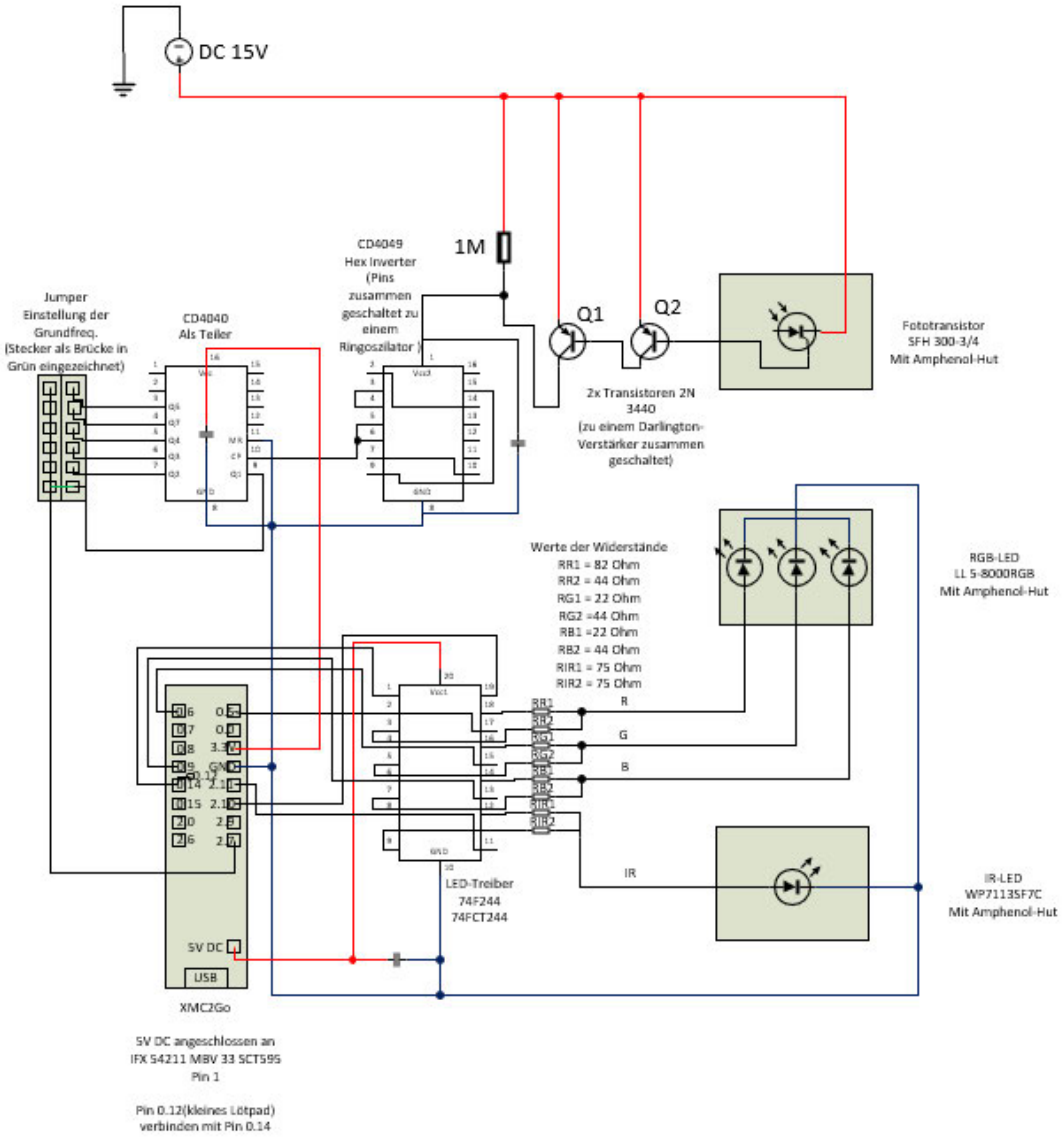


Abbildung A.1: Schaltplan der Ringoszillatorplatine mit zusätzlichen Informationen für den Jumper

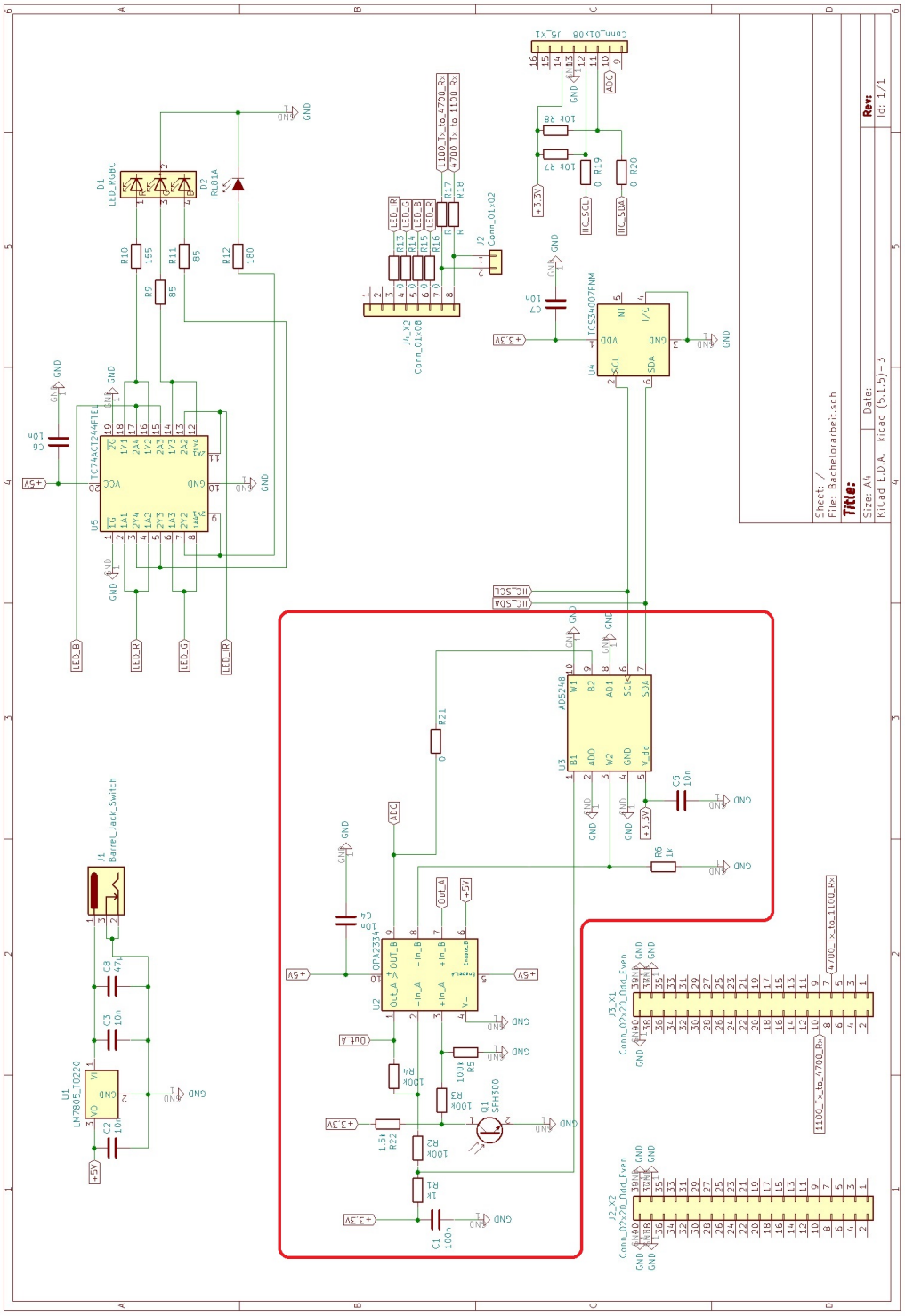


Abbildung A.2: Schaltplan der Farbsensorplatine
Der rote Kreis beinhaltet eine optionale Bestückung für eine Rheostat gesteuerte Verstärkerschaltung

A.2 Platinenlayout

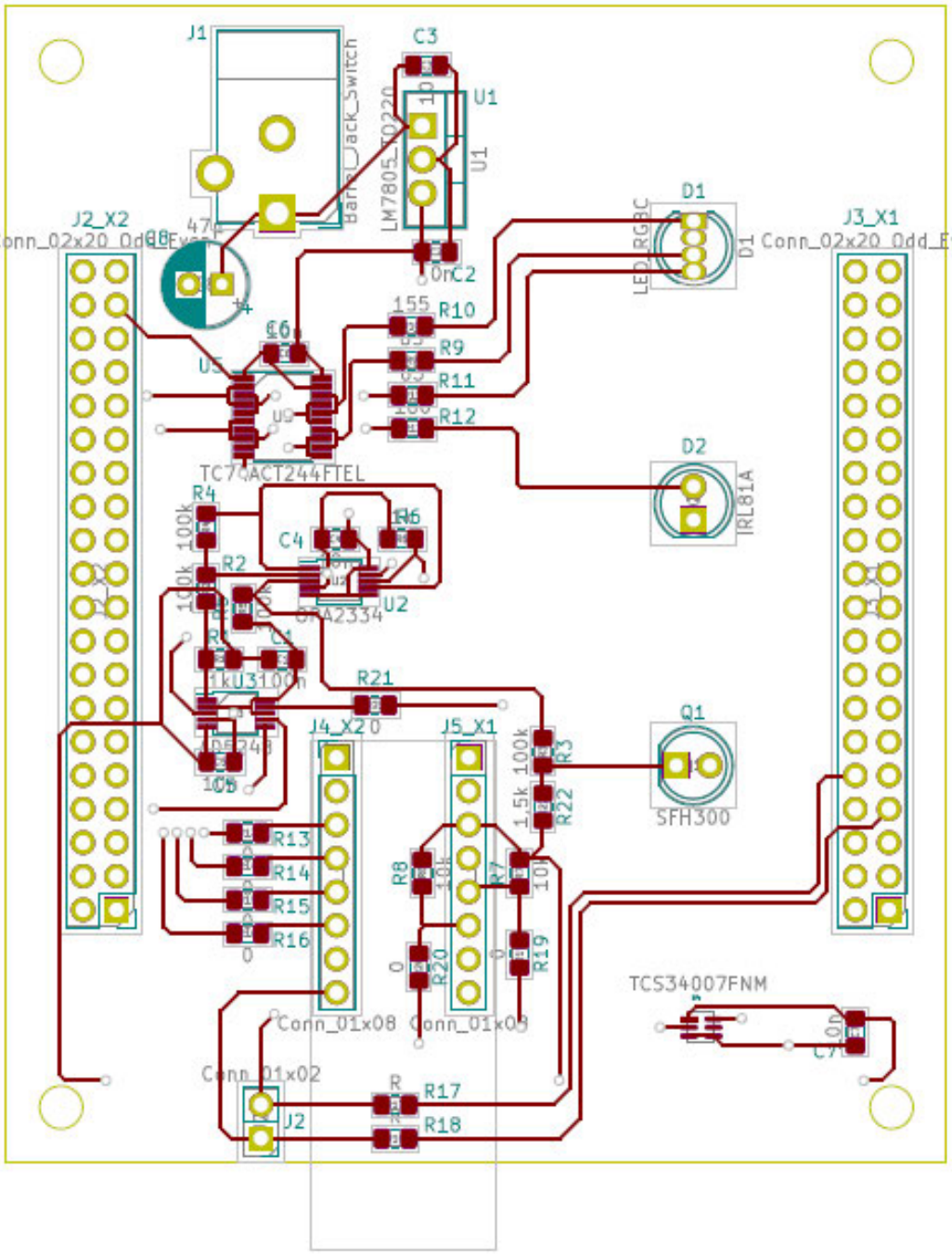


Abbildung A.3: Platinenlayout der Farbsensorplatine von oben

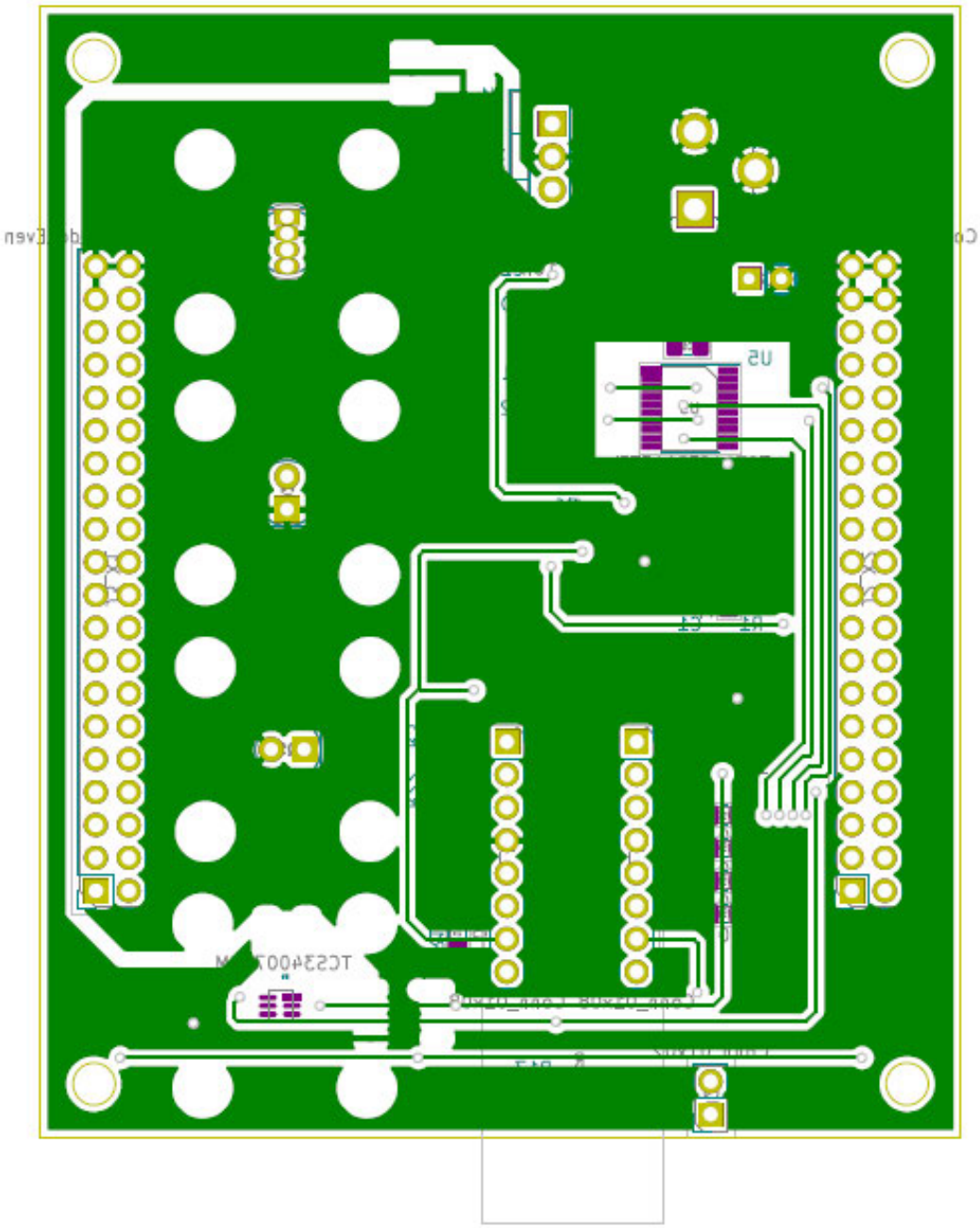


Abbildung A.4: Platinenlayout der Farbsensorplatine von unten

A.3 Quellcodes

A.3.1 Mikrocontroller C-Codes

Farbsensorplatine main.c

```
/*
 * main.c
 *
 *  Created on: 2020 Sep 23 11:53:12
 *  Author: simon
 */

#include <DAVE.h>                 //Declarations from DAVE Code Generation (includes SFR declaration)
#include "functions.h"
#include <time.h>
#include "XMC1100.h"
#include "string.h"
#include "math.h"

/**
 * @brief main() - Application entry point
 *
 * <b>Details of function</b><br>
 * This routine is the application entry point. It is invoked by the device startup code. It is responsible for
 * invoking the APP initialization dispatcher routine - DAVE_Init() and hosting the place-holder for user application
 * code.
 */

#define sens_addr 0x72      /*I2C Addresse des Farbsensors*/

#define STATE_LED_IR    0x01  /*Status fuer LED-Angabe hier IR-LED*/
#define STATE_LED_RED   0x02  /*Status fuer LED-Angabe hier rote LED*/
#define STATE_LED_GREEN   0x03  /*Status fuer LED-Angabe hier gruene LED*/
#define STATE_LED_BLUE    0x04  /*Status fuer LED-Angabe hier blaue LED*/
#define STATE_LED_OUT   0x05  /*Status fuer LED-Angabe hier blaue LED*/
#define STATE_COMANND   0x06  /*Status um neue Einstellungen vorzunehmen*/
uint8_t LED = STATE_LED_IR;   /*Status Variable*/

```

```
34 //#define TRANSISTOR      false /*Angabe ob Transistor also Sensor fuer
       Messung ausgewaehlt ist*/
35
36 #define UART         false /*Angabe ob Daten aus den Messungen ueber UART-
       Bus geschickt werden*/
37
38 #define TIME1        314500*100U   /*Zeit, die der Timer laeuft*/
39
40 #define ROUNDS         5   /*Anzahl an Messdurchlaeufen die durchgefuehrt
       werden*/
41
42 TIMER_STATUS_t timer_status;
43
44 uint8_t UART_IR[] = "0";  /*Angabe welche Daten ueber den UART-Bus
       geschickt werden hier Daten aus Messung mit IR-LED*/
45 uint8_t UART_RED[] = "1"; /*Angabe welche Daten ueber den UART-Bus
       geschickt werden hier Daten aus Messung mit roter LED*/
46 uint8_t UART_GREEN[] = "2"; /*Angabe welche Daten ueber den UART-Bus
       geschickt werden hier Daten aus Messung mit gruener LED*/
47 uint8_t UART_BLUE[] = "3";  /*Angabe welche Daten ueber den UART-Bus
       geschickt werden hier Daten aus Messung mit blauer LED*/
48
49
50 //Sensor IIC Befehle
51 uint8_t tx_poweron[2] = {0x80,0x03};
52 uint8_t tx_enb[2] = {0x80,0x01};     /*0x80 = Enable Register, 0x03 = Power
       ON, ADC Enable (siehe Datenblatt S.14)*/
53 uint8_t tx_int_time[2] = {0x81,0xC0}; /*0x81 = RGBC Integration Time
       Register,DickeFaser 0xC0 = (Cycles=64 Time=178ms Max Count=65535)
       DuenneFaser IR:0xC0 (Cycles=64 Time=178ms Max Count=65535)RGB:0x8F(
       Cycles=113 Time=314ms Max Count=65535)(siehe Datenblatt S.15)*/
54 uint8_t tx_gain[2] = {0x8F,0x02};    /*0x8F = Control Register, DickeFaser 0
       x02 (= 16X Gain); IR:DuenneFaser 0x02 (= 16X Gain), RGB:DuenneFaser 0
       x03 (= 64X Gain) (siehe Datenblatt S.18)*/
55 uint8_t tx_ir[2] = {0xC0,0x80};      /*0xC0 = IR Register, 0x80 = IR Sensor
       access (siehe Datenblatt S.22)*/
56
57 //Sensor Register zum Abfragen
58 uint8_t reg_device_id[2] = {0x92};     /*ID Register, erwarteter Wert
       10010000*/
59 uint8_t reg_ClearIR_low[2] = {0x94};  /*Clear / IR data low byte,
       Resetvalue = 0x00*/
60 uint8_t reg_ClearIR_high[2] = {0x95}; /*Clear / IR data high byte,
       Resetvalue = 0x00*/
```

```
61 uint8_t reg_red_low[2] = {0x96};     /*Red data low byte, Resetvalue = 0x00
       */
62 uint8_t reg_red_high[2] = {0x97};    /*Red data high byte, Resetvalue = 0x00
       */
63 uint8_t reg_green_low[2] = {0x98};     /*Green data low byte, Resetvalue = 0
       x00*/
64 uint8_t reg_green_high[2] = {0x99};    /*Green data high byte, Resetvalue =
       0x00*/
65 uint8_t reg_blue_low[2] = {0x9A};    /*Blue data low byte, Resetvalue = 0x00
       */
66 uint8_t reg_blue_high[2] = {0x9B};     /*Blue data high byte, Resetvalue = 0
       x00*/
67
68 uint16_t data_RGB[5][ROUNDS] = {0}; //test fuer rgb_ir messung
69
70 void Timetick_Handler(void)
71 {
72   timer_status = TIMER_Stop(&TIMER_0);   /*Timer beenden*/
73   TIMER_ClearEvent(&TIMER_0);       /*Timer Flag loeschen*/
74 }
75
76 void UartRecevie_Handler(void)
77 {
78   int i=0;
79
80   i++;
81   //UART_ClearFlag(&UART_0,(XMC_UART_CH_STATUS_FLAG_RECEIVE_INDICATION |
       XMC_UART_CH_STATUS_FLAG_ALTERNATIVE_RECEIVE_INDICATION));
82 }
83
84 void Sensor_Init(void)
85 {
86   /*Erleuterung der I2C_MASTER_Transmit Funktionsparameter
87    *1. Geraete Handle auf den Master
88    *2. Aussage ob Start Bedingung gesendet werden soll
89    *3. Adresse des Slave
90    *4. Buffer mit zu schreibenen Daten
91    *5. Anzahl an Byte die gesendet werden sollen
92    *6. Aussage ob Stop Bedingung gesendet werden soll
93       Erklaerung der I2C_MASTER_IsTxBusy Funktion
94    *Diese Funktion wartet solange bis der Master die uebertragung beendet
       hat
95    */
96
```

```
97    //Sensor und ADC einschalten (siehe Datenblatt S.14)
98    I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_poweron, 2, true);
99    while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
100
101   //Integration Time auf 27,8ms stellen (siehe Datenblatt S.15)
102   I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_int_time, 2, true);
103   while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
104
105   //Gain auf 64x stellen (siehe Datenblatt S.18)
106   I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_gain, 2, true);
107   while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
108
109   //IR Mode einschalten (siehe Datenblatt S.22)
110   I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_ir, 2, true);
111   while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
112
113 }
114
115 uint16_t Calc_Data_Sensor(rounds)
116 {
117   uint8_t rx_data_low = 0;
118   uint8_t rx_data_high = 0;
119   uint16_t rx_data_high_shift = 0;
120   uint16_t rx_data_ClearIR = 0;
121   uint16_t rx_data_red = 0;
122   uint16_t rx_data_green = 0;
123   uint16_t rx_data_blue = 0;
124   uint8_t tx_gain_1x[2] = {0x8F,0x00};     //0x8F = Control Register; Gain =
        1x
125   uint8_t tx_gain_4x[2] = {0x8F,0x01};     //0x8F = Control Register; Gain =
        4x
126   uint8_t tx_gain_16x[2] = {0x8F,0x02};    //0x8F = Control Register; Gain =
        16x
127   uint8_t tx_gain_64x[2] = {0x8F,0x03};    //0x8F = Control Register; Gain =
        64x
128   uint16_t decimal_decide = 0;
129
130   bool receive_int_time = false;
131   bool receive_gain = false;
132   bool receive_led = false;
133
134   uint8_t ReadData[2];// = {0};
135
136   uint8_t Data_Gain;
```

```
137   uint8_t Data_IntTime;
138   uint8_t Data_LED;
139   double calc_decimal;
140   long long int int_time;
141
142   static  uint8_t colour = STATE_LED_IR;
143   switch(LED){
144   case 1:{
145
146     if(colour==STATE_LED_IR){
147
148       I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_poweron, 2, true);
      //ADC an
149       while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
150
151       wait();
152
153       LED_IR_High();
154       LED_IR_Low();
155
156       colour=STATE_LED_OUT;
157     }
158
159     else{
160       I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_poweron, 2, true);
161       while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
162
163       wait();
164
165       LED_Out_High();
166       LED_Out_Low();
167       colour=STATE_LED_IR;
168     }
169
170     I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, reg_ClearIR_low, 1,
      false);//Ansprechen des Registers fuer IR-Daten mit den niedrigen
      Werten
171     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
172
173     I2C_MASTER_Receive(&I2C_MASTER_0, true, sens_addr,&rx_data_low, 1, true,
      true);//lesen des Registers fuer IR-Daten mit den niedrigen Werten
174     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
175
```

```
176     I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, reg_ClearIR_high, 1,
        false);//Ansprechen des Registers fuer IR-Daten mit den hohen Werten
177     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
178
179     I2C_MASTER_Receive(&I2C_MASTER_0, true, sens_addr,&rx_data_high, 1, true,
        true);//lesen des Registers fuer IR-Daten mit den hohen Werten
180     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
181
182     rx_data_high_shift = (rx_data_high_shift|rx_data_high)<<8;  //Daten aus
         Register fuer hohe Werte zu MSB shiften
183     rx_data_ClearIR = rx_data_high_shift | rx_data_low;    //Daten aus
        Register fuer niedrige Werte zu kompletten Daten verodern
184
185     I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_enb, 2, true);
186     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
187
188     LED = STATE_LED_IR;
189
190     data_RGB[0][rounds]=rx_data_ClearIR;
191
192     if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
193     {
194       if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
195       {
196         LED=STATE_LED_RED;
197       }
198       if((ReadData[0]=='e')||(ReadData[1]=='e'))
199       {
200         LED=STATE_COMANND;
201       }
202     }
203
204     break;
205   }
206   case 2:{
207
208     if(colour==STATE_LED_RED){
209
210       I2C_MASTER_Transmit(&I2C_MASTER_0, true, sens_addr, tx_poweron, 2, true);
      //ADC an
211       while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
212
213       wait();
214
```

```
215         LED_Red_High();
216         LED_Red_Low();
217         colour=STATE_LED_GREEN;
218       }
219       else if(colour==STATE_LED_GREEN){
220         I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_poweron,2,true);
221         while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
222
223         wait();
224
225         LED_Green_High();
226         LED_Green_Low();
227         colour=STATE_LED_BLUE;
228       }
229       else if(colour==STATE_LED_BLUE){
230
231         I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_poweron,2,true);
232         while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
233
234         wait();
235
236         LED_Blue_High();
237         LED_Blue_Low();
238         colour=STATE_LED_OUT;
239       }
240       else{
241         I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_poweron,2,true);
242         while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
243
244         wait();
245
246         LED_Out_High();
247         LED_Out_Low();
248         colour=STATE_LED_RED;
249       }
250
251       I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_red_low,1,false);
      //Ansprechen des Registers fuer rote Daten mit den niedrigen Werten
252       while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
253
254       I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_low,1,true,
      true);//lesen des Registers fuer rote Daten mit den niedrigen Werten
255       while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
256
```

```
257     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_red_high,1,false);
        //Ansprechen des Registers fuer rote Daten mit den hohen Werten
258     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
259
260     I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_high,1,true,
        true);//lesen des Registers fuer rote Daten mit den hohen Werten
261     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
262
263     rx_data_high_shift = (rx_data_high_shift|rx_data_high)<<8;  //Daten aus
         Register fuer hohe Werte zu MSB shiften
264     rx_data_red = rx_data_high_shift | rx_data_low;      //Daten aus
        Register fuer niedrige Werte zu kompletten Daten verodern
265
266
267     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_green_low,1,false)
        ;//Ansprechen des Registers fuer gruene Daten mit den niedrigen Werten
268     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
269
270     I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_low,1,true,
        true);//lesen des Registers fuer gruene Daten mit den niedrigen Werten
271     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
272
273     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_green_high,1,false
        );//Ansprechen des Registers fuer gruene Daten mit den hohen Werten
274     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
275
276     I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_high,1,true,
        true);//lesen des Registers fuer gruene Daten mit den hohen Werten
277     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
278
279     rx_data_high_shift = (rx_data_high_shift|rx_data_high)<<8;  //Daten aus
         Register fuer hohe Werte zu MSB shiften
280     rx_data_green = rx_data_high_shift | rx_data_low;      //Daten aus
        Register fuer niedrige Werte zu kompletten Daten verodern
281
282
283
284     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_blue_low,1,false);
        //Ansprechen des Registers fuer blaue Daten mit den niedrigen Werten
285     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
286
287     I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_low,1,true,
        true);//lesen des Registers fuer blaue Daten mit den niedrigen Werten
288     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
```

```
289
290     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,reg_blue_high,1,false)
        ;//Ansprechen des Registers fuer blaue Daten mit den hohen Werten
291     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
292
293     I2C_MASTER_Receive(&I2C_MASTER_0,true,sens_addr,&rx_data_high,1,true,
        true);//lesen des Registers fuer blaue Daten mit den hohen Werten
294     while(I2C_MASTER_IsRxBusy(&I2C_MASTER_0));
295
296     rx_data_high_shift = (rx_data_high_shift|rx_data_high)<<8;  //Daten aus
         Register fuer hohe Werte zu MSB shiften
297     rx_data_blue = rx_data_high_shift | rx_data_low;       //Daten aus
        Register fuer niedrige Werte zu kompletten Daten verodern
298
299     I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_enb,2,true);
300     while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
301
302
303     data_RGB[1][rounds]=rx_data_red;
304     data_RGB[2][rounds]=rx_data_green;
305     data_RGB[3][rounds]=rx_data_blue;
306
307
308     if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
309     {
310       if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
311       {
312         LED=STATE_LED_RED;
313       }
314       if((ReadData[0]=='e')||(ReadData[1]=='e'))
315       {
316         LED=STATE_COMANND;
317       }
318     }
319     break;
320   }
321   case 6:{
322     while(receive_int_time == false)
323     {
324       if(UART_Receive(&UART_0, ReadData, 1) == UART_STATUS_SUCCESS) //mit
      hex werten versuchen
325       {
326         Data_IntTime=ReadData[0];
327
```

```
328         decimal_decide = 0x100-Data_IntTime;
329 
330         if(decimal_decide<4)
331         {
332           tx_int_time[1]=ReadData[0];
333           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_int_time,2,
    true);
334           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
335           calc_decimal = 2.78*decimal_decide*100;
336           int_time=round(calc_decimal)*10;//314500*100U
337           int_time=int_time*100U;
338 
339           timer_status = TIMER_SetTimeInterval(&TIMER_0, int_time);
340           receive_int_time=true;
341         }
342         else if((3<decimal_decide)&&(decimal_decide<35))
343         {
344           tx_int_time[1]=ReadData[0];
345           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_int_time,2,
    true);
346           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
347           calc_decimal = 2.78*decimal_decide*10;
348           int_time=round(calc_decimal)*100;//31450000U
349           int_time=int_time*100U;
350 
351           timer_status = TIMER_SetTimeInterval(&TIMER_0, int_time);
352           UART_Transmit(&UART_0, ReadData, 1);
353           receive_int_time=true;
354         }
355         else
356         {
357           tx_int_time[1]=ReadData[0];
358           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_int_time,2,
    true);
359           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
360           calc_decimal = 2.78*decimal_decide;
361           int_time=ceil(calc_decimal)*1000;//314500*100U
362           int_time=int_time*100U;
363 
364           timer_status = TIMER_SetTimeInterval(&TIMER_0, int_time);
365 
366           receive_int_time=true;
367         }
368 
```

```
369       }
370
371     }
372
373     while(receive_gain == false)
374     {
375       if(UART_Receive(&UART_0, ReadData, 2)==UART_STATUS_SUCCESS)
376       {
377         if(ReadData[0]=='\n')
378         {
379           Data_Gain=ReadData[1];
380         }
381         else
382         {
383           Data_Gain=ReadData[0];
384         }
385         switch(Data_Gain)
386         {
387         case '0':
388           //Gain auf 1x stellen (siehe Datenblatt S.18)
389           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_gain_1x,2,
    true);
390           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
391           receive_gain = true;
392           break;
393         case '1':
394           //Gain auf 4x stellen (siehe Datenblatt S.18)
395           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_gain_4x,2,
    true);
396           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
397           receive_gain = true;
398           break;
399         case '2':
400           //Gain auf 16x stellen (siehe Datenblatt S.18)
401           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_gain_16x,2,
    true);
402           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
403           receive_gain = true;
404           break;
405         case '3':
406           //Gain auf 64x stellen (siehe Datenblatt S.18)
407           I2C_MASTER_Transmit(&I2C_MASTER_0,true,sens_addr,tx_gain_64x,2,
    true);
408           while(I2C_MASTER_IsTxBusy(&I2C_MASTER_0));
```

```
409             receive_gain = true;
410             break;
411           }
412
413       }
414
415     }
416
417     while(receive_led == false)
418     {
419       if(UART_Receive(&UART_0, ReadData, 2)==UART_STATUS_SUCCESS)
420       {
421         if(ReadData[0]=='\n')
422         {
423           Data_LED=ReadData[1];
424         }
425         else
426         {
427           Data_LED=ReadData[0];
428         }
429         switch(Data_LED)
430         {
431         case 'i':
432           LED=STATE_LED_IR;
433           colour=STATE_LED_IR;
434           receive_led = true;
435           break;
436         case 'r':
437           LED=STATE_LED_RED;
438           colour=STATE_LED_RED;
439           receive_led = true;
440           break;
441         case 'g':
442           LED=STATE_LED_RED;
443           colour=STATE_LED_RED;
444           receive_led = true;
445           break;
446         case 'b':
447           LED=STATE_LED_RED;
448           colour=STATE_LED_RED;
449           receive_led = true;
450           break;
451         }
452       }
```

```
453     }
454     break;
455   }
456   }
457 }
458
459 void wait(void)
460 {
461   for(int i=0;i<25000;i++);//5000 fuer 178ms LED //25000 fuer 503ms LED
462 }
463
464 void Uart_Init()
465 {
466   UART_STATUS_t init_status;
467   init_status = (UART_STATUS_t)UART_Init(&UART_0);
468   wait();
469   if(init_status != UART_STATUS_SUCCESS)
470   {
471     XMC_DEBUG("main: Application initialization failed");
472     while(1U)
473     {
474     }
475   }
476 }
477
478 void test1(void)
479 {
480   wait();
481 }
482
483 void test2(void)
484 {
485   wait();
486 }
487 void test3(void)
488 {
489   wait();
490 }
491 void test4(void)
492 {
493   wait();
494 }
495 int main(void)
496 {
```

```
497 DAVE_STATUS_t init_status;
498
499 uint8_t rx_data_sens;          /*Variable fuer Daten aus ID_Register*/
500 uint16_t data_ClearIR[ROUNDS] = {0};  /*Variable fuer Daten aus den
      Messungen mit IR_LED*/
501 uint16_t data_red[3][ROUNDS] = {0};   /*Variable fuer Daten aus den
      Messungen mit roter LED*/
502 uint16_t data_green[3][ROUNDS] = {0};   /*Variable fuer Daten aus den
      Messungen mit gruener LED*/
503 uint16_t data_blue[3][ROUNDS] = {0};    /*Variable fuer Daten aus den
      Messungen mit blauer LED*/
504 uint16_t data_out[3][ROUNDS] = {0};   /*Variable fuer Daten aus den
      Messungen mit blauer LED*/
505 uint8_t uart_test;           /*Variable fuer UART uebertragung*/
506
507 static int i; /*Zaehlvariable*/
508 int r1=0;
509 int r2=0;
510 int r3=0;
511
512 int g1=0;
513 int g2=0;
514 int g3=0;
515
516 int b1=0;
517 int b2=0;
518 int b3=0;
519
520 int o1=0;
521 int o2=0;
522 int o3=0;
523
524 int ir=0;
525 uint16_t red_mittel1=0;
526 uint16_t red_mittel2=0;
527 uint16_t red_mittel3=0;
528 int red_calc1=0;
529 int red_calc2=0;
530 int red_calc3=0;
531 uint16_t green_mittel1=0;
532 uint16_t green_mittel2=0;
533 uint16_t green_mittel3=0;
534 int green_calc1=0;
535 int green_calc2=0;
```

```
536  int green_calc3=0;
537  uint16_t blue_mittel1=0;
538  uint16_t blue_mittel2=0;
539  uint16_t blue_mittel3=0;
540  int blue_calc1=0;
541  int blue_calc2=0;
542  int blue_calc3=0;
543  uint16_t out_mittel1=0;
544  uint16_t out_mittel2=0;
545  uint16_t out_mittel3=0;
546  int out_calc1=0;
547  int out_calc2=0;
548  int out_calc3=0;
549  uint16_t ir_mittel=0;
550  int ir_calc=0;
551  //uint16_t test=0;
552  uint8_t Send_Data_red[] = "r";
553  uint8_t Send_Data_green[] = "g";
554  uint8_t Send_Data_blue[] = "b";
555  uint8_t Send_Data_out[] = "o";
556  uint8_t Send_Data_ir[] = "i";
557  char array[8];
558
559  init_status = DAVE_Init();           /* Initialization of DAVE APPs  */
560  wait();
561  if(init_status != DAVE_STATUS_SUCCESS)
562  {
563    /* Placeholder for error handler code. The while loop below can be
       replaced with an user error handler. */
564    XMC_DEBUG("DAVE APPs initialization failed\n");
565
566    while(1U)
567    {
568
569    }
570
571  }
572  //SYSTIMER_Stop();
573
574
575  /*
576   * UART_Receive(&UART_0, LED, 1);      //Befehl erhalten welche LED
       ausgewaehlt ist
```

```
577   * UART_Receive(&UART_0, TRANSISTOR, 1);  //Befehl erhalten welcher
       Sensor ausgewaehlt ist
578   * UART_Receive(&UART_0, UART, 1);    //Befehl erhalten ob Daten per UART
        an XMC4700 gesendet werden sollen
579   * UART_Receive(&UART_0, TIME, 1);    //Befehl erhalten wie lange die
       Messung dauern soll
580   * UART_Receive(&UART_0, ROUNDS, 1)   //Befehl erhalten wie viele
       Messdurchlaeufe durchgefuehrt werden
581   * UART_Receive(&UART_0, RHEO1, 1)    //Befehl erhalten welchen Wert
       Rheostat 1 erhalten soll
582   * UART_Receive(&UART_0, RHEO2, 1)    //Befehl erhalten welchen Wert
       Rheostat 2 erhalten soll
583   */
584  TIMER_Stop(&TIMER_0);   //Timer anhalten
585  timer_status = TIMER_SetTimeInterval(&TIMER_0, TIME1);    //Zeit, die der
       Timer laeuft einstellen Wert TIME = mikro sec * 100
586
587
588
589  /* Placeholder for user application code. The while loop below can be
       replaced with user application code. */
590  Uart_Init();       /*Initalisierung des UART Busses*/
591  wait();
592  //Rheo_Init();       /*Initalisierung des Rheostaten*/
593  //wait();
594  Sensor_Init();     /*Initalisierung des Farbsensors*/
595  wait();
596
597  //UART_Transmit(&UART_0, Send_Data, sizeof(Send_Data)-1);
598
599  //berechnung mittelwert ohne 0 werte
600
601  while(1U)
602  {
603    for(i=0;i<ROUNDS;i++)
604    {
605      if(LED==STATE_LED_IR)
606      {
607        Calc_Data_Sensor(i);
608        data_ClearIR[i] = data_RGB[0][i];//IR
609
610        Calc_Data_Sensor(i);
611        data_out[1][i] = data_RGB[0][i]; //IR
612      }
```

```
613       else
614       {
615         Calc_Data_Sensor(i);
616         data_red[1][i] = data_RGB[1][i]; //ROT
617         data_red[2][i] = data_RGB[2][i]; //GRueN
618         data_red[3][i] = data_RGB[3][i]; //BLAU
619
620         Calc_Data_Sensor(i);
621         data_green[1][i] = data_RGB[1][i]; //ROT
622         data_green[2][i] = data_RGB[2][i]; //GRueN
623         data_green[3][i] = data_RGB[3][i]; //BLAU
624
625         Calc_Data_Sensor(i);
626         data_blue[1][i] = data_RGB[1][i]; //ROT
627         data_blue[2][i] = data_RGB[2][i]; //GRueN
628         data_blue[3][i] = data_RGB[3][i]; //BLAU
629
630         Calc_Data_Sensor(i);
631         data_out[1][i] = data_RGB[1][i]; //ROT
632         data_out[2][i] = data_RGB[2][i]; //GRueN
633         data_out[3][i] = data_RGB[3][i]; //BLAU
634
635       }
636     }
637     for(i=0;i<ROUNDS;i++)
638     {
639       if((data_red[1][i]==0)&&(i<5))
640       {
641         red_calc1=red_calc1;
642       }
643       else
644       {
645         red_calc1=red_calc1+data_red[1][i];
646         r1++;
647         if(i==ROUNDS-1 && r1>0)
648         {
649           red_mittel1=red_calc1/r1;
650           red_calc1=0;
651           r1=0;
652         }
653       }
654       if((data_red[2][i]==0)&&(i<5))
655       {
656         red_calc2=red_calc2;
```

```
657       }
658       else
659       {
660         red_calc2=red_calc1+data_red[2][i];
661         r2++;
662         if(i==ROUNDS-1 && r2>0)
663         {
664           red_mittel2=red_calc2/r2;
665           red_calc2=0;
666           r2=0;
667         }
668       }
669       if((data_red[3][i]==0)&&(i<5))
670       {
671         red_calc3=red_calc3;
672       }
673       else
674       {
675         red_calc3=red_calc3+data_red[3][i];
676         r3++;
677         if(i==ROUNDS-1 && r3>0)
678         {
679           red_mittel3=red_calc3/r3;
680           red_calc3=0;
681           r3=0;
682         }
683       }
684       ////////////////////////////////////////////////////////
685       if((data_green[1][i]==0)&&(i<5))
686       {green_calc1=green_calc1;}
687       else
688       {
689         green_calc1=green_calc1+data_green[1][i];
690         g1++;
691         if(i==ROUNDS-1 && g1>0)
692         {
693           green_mittel1=green_calc1/g1;
694           green_calc1=0;
695           g1=0;
696         }
697       }
698       if((data_green[2][i]==0)&&(i<5))
699       {green_calc2=green_calc2;}
700       else
```

```
701       {
702         green_calc2=green_calc2+data_green[2][i];
703         g2++;
704         if(i==ROUNDS-1 && g2>0)
705         {
706           green_mittel2=green_calc2/g2;
707           green_calc2=0;
708           g2=0;
709         }
710       }
711       if((data_green[3][i]==0)&&(i<5))
712       {green_calc3=green_calc3;}
713       else
714       {
715         green_calc3=green_calc3+data_green[3][i];
716         g3++;
717         if(i==ROUNDS-1 && g3>0)
718         {
719           green_mittel3=green_calc3/g3;
720           green_calc3=0;
721           g3=0;
722         }
723       }
724       ////////////////////////////////////////////////////////
725       if((data_blue[1][i]==0)&&(i<5))
726       {blue_calc1=blue_calc1;}
727       else
728       {
729         blue_calc1=blue_calc1+data_blue[1][i];
730         b1++;
731         if(i==ROUNDS-1 && b1>0)
732         {
733           blue_mittel1=blue_calc1/b1;
734           blue_calc1=0;
735           b1=0;
736         }
737
738       }
739       if((data_blue[2][i]==0)&&(i<5))
740       {blue_calc2=blue_calc2;}
741       else
742       {
743         blue_calc2=blue_calc2+data_blue[2][i];
744         b2++;
```

```
745         if(i==ROUNDS-1 && b2>0)
746         {
747           blue_mittel2=blue_calc2/b2;
748           blue_calc2=0;
749           b2=0;
750         }
751 
752       }
753       if((data_blue[3][i]==0)&&(i<5))
754       {blue_calc3=blue_calc3;}
755       else
756       {
757         blue_calc3=blue_calc3+data_blue[3][i];
758         b3++;
759         if(i==ROUNDS-1 && b3>0)
760         {
761           blue_mittel3=blue_calc3/b3;
762           blue_calc3=0;
763           b3=0;
764         }
765 
766       }
767       ////////////////////////////////////////////////////////
768       if((data_out[1][i]==0)&&(i<5))
769       {out_calc1=out_calc1;}
770       else
771       {
772         out_calc1=out_calc1+data_out[1][i];
773         o1++;
774         if(i==ROUNDS-1 && o1>0)
775         {
776           out_mittel1=out_calc1/o1;
777           out_calc1=0;
778           o1=0;
779         }
780 
781       }
782       if((data_out[2][i]==0)&&(i<5))
783       {out_calc2=out_calc2;}
784       else
785       {
786         out_calc2=out_calc2+data_out[2][i];
787         o2++;
788         if(i==ROUNDS-1 && o2>0)
```

```
789         {
790           out_mittel2=out_calc2/o2;
791           out_calc2=0;
792           o2=0;
793         }
794 
795       }
796       if((data_out[3][i]==0)&&(i<5))
797       {out_calc3=out_calc3;}
798       else
799       {
800         out_calc3=out_calc3+data_out[3][i];
801         o3++;
802         if(i==ROUNDS-1 && o3>0)
803         {
804           out_mittel3=out_calc3/o3;
805           out_calc3=0;
806           o3=0;
807         }
808 
809       }
810       ////////////////////////////////////////////////////////
811       if((data_ClearIR[i]==0)&&(i<5))
812       {ir_calc=ir_calc;}
813       else
814       {
815         ir_calc=ir_calc+data_ClearIR[i];
816         ir++;
817         if(i==ROUNDS-1 && ir>0)
818         {
819           ir_mittel=ir_calc/ir;
820           ir_calc=0;
821           ir=0;
822         }
823 
824       }
825     }
826     //switch LED
827     if(LED == STATE_LED_RED)
828     {
829       UART_Transmit(&UART_0, Send_Data_red, sizeof(Send_Data_red)-1);
830       sprintf (array, "%d\n", red_mittel1);
831       UART_Transmit(&UART_0, array, strlen(array));
832       sprintf (array, "%d\n", red_mittel2);
```

```
833       UART_Transmit(&UART_0, array, strlen(array));
834       sprintf (array, "%d\n", red_mittel3);
835       UART_Transmit(&UART_0, array, strlen(array));
836
837       UART_Transmit(&UART_0, Send_Data_green, sizeof(Send_Data_green)-1);
838       sprintf (array, "%d\n", green_mittel1);
839       UART_Transmit(&UART_0, array, strlen(array));
840       sprintf (array, "%d\n", green_mittel2);
841       UART_Transmit(&UART_0, array, strlen(array));
842       sprintf (array, "%d\n", green_mittel3);
843       UART_Transmit(&UART_0, array, strlen(array));
844
845       UART_Transmit(&UART_0, Send_Data_blue, sizeof(Send_Data_blue)-1);
846       sprintf (array, "%d\n", blue_mittel1);
847       UART_Transmit(&UART_0, array, strlen(array));
848       sprintf (array, "%d\n", blue_mittel2);
849       UART_Transmit(&UART_0, array, strlen(array));
850       sprintf (array, "%d\n", blue_mittel3);
851       UART_Transmit(&UART_0, array, strlen(array));
852
853       UART_Transmit(&UART_0, Send_Data_out, sizeof(Send_Data_out)-1);
854       sprintf (array, "%d\n", out_mittel1);
855       UART_Transmit(&UART_0, array, strlen(array));
856       sprintf (array, "%d\n", out_mittel2);
857       UART_Transmit(&UART_0, array, strlen(array));
858       sprintf (array, "%d\n", out_mittel3);
859       UART_Transmit(&UART_0, array, strlen(array));
860     }
861     else
862     {
863       UART_Transmit(&UART_0, Send_Data_ir, sizeof(Send_Data_ir)-1);
864       sprintf (array, "%d\n", ir_mittel);
865       UART_Transmit(&UART_0, array, strlen(array));
866
867       UART_Transmit(&UART_0, Send_Data_out, sizeof(Send_Data_out)-1);
868       sprintf (array, "%d\n", out_mittel1);
869       UART_Transmit(&UART_0, array, strlen(array));
870     }
871
872   }
873 }
```

Farbsensorplatine functions.h

```
/*
 * functions.h
 *
 *  Created on: 7 Oct 2020
 *      Author: simon
 */

#ifndef FUNCTIONS_H_
#define FUNCTIONS_H_

#include <Dave.h>

void LED_Red_High(void);
void LED_Green_High(void);
void LED_Blue_High(void);
void LED_IR_High(void);
void LED_Red_Low(void);
void LED_Green_Low(void);
void LED_Blue_Low(void);
void LED_IR_Low(void);


#endif /* FUNCTIONS_H_ */
```

Farbsensorplatine led_functions.c

```
#include "functions.h"
#include <Dave.h>


TIMER_STATUS_t timer_status;

void LED_Red_High(void)
{
  DIGITAL_IO_SetOutputHigh(&LED_RED);
  timer_status = TIMER_Start(&TIMER_0);
}

void LED_Red_Low(void)
{
  while(TIMER_GetTimerStatus(&TIMER_0));
```

```
16   DIGITAL_IO_SetOutputLow(&LED_RED);
17 }
18
19 void LED_Green_High(void)
20 {
21   DIGITAL_IO_SetOutputHigh(&LED_GREEN);
22   timer_status = TIMER_Start(&TIMER_0);
23 }
24
25 void LED_Green_Low(void)
26 {
27   while(TIMER_GetTimerStatus(&TIMER_0));
28   DIGITAL_IO_SetOutputLow(&LED_GREEN);
29 }
30
31 void LED_Blue_High(void)
32 {
33   DIGITAL_IO_SetOutputHigh(&LED_BLUE);
34   timer_status = TIMER_Start(&TIMER_0);
35 }
36
37 void LED_Blue_Low(void)
38 {
39   while(TIMER_GetTimerStatus(&TIMER_0));
40   DIGITAL_IO_SetOutputLow(&LED_BLUE);
41 }
42
43 void LED_IR_High(void)
44 {
45   DIGITAL_IO_SetOutputHigh(&LED_IR);
46   timer_status = TIMER_Start(&TIMER_0);
47 }
48
49 void LED_IR_Low(void)
50 {
51   while(TIMER_GetTimerStatus(&TIMER_0));
52   DIGITAL_IO_SetOutputLow(&LED_IR);
53 }
54
55 void LED_Out_High(void)
56 {
57   timer_status = TIMER_Start(&TIMER_0);
58 }
59
```

```
60 void LED_Out_Low(void)
61 {
62   while(TIMER_GetTimerStatus(&TIMER_0));
63 }
```

Ringoszillatorplatine main.c

```
1  /*
2   * main.c
3   *
4   *  Created on: 2020 Dec 16 15:10:34
5   *  Author: Simon
6   */
7
8  #include <DAVE.h>                 //Declarations from DAVE Code Generation
       (includes SFR declaration)
9  #include "string.h"
10 #include "inttypes.h"
11
12
13 #define ONESEC 1000000U        //250000U Wert fuer 250ms
14 #define STATE_LED_IR    0x01   //Status fuer LED-Angabe hier IR-LED
15 #define STATE_LED_RED   0x02   //Status fuer LED-Angabe hier rote LED
16 #define STATE_LED_GREEN   0x03   //Status fuer LED-Angabe hier gruene LED
17 #define STATE_LED_BLUE    0x04   //Status fuer LED-Angabe hier blaue LED
18 #define STATE_LED_NOISE   0x05   //Status zum ermitteln des Rauschen
19 #define STATE_COMANND   0x06   //Status um neue Einstellungen vorzunehmen
20 uint8_t LED = STATE_LED_IR;    //Status Variable
21
22
23 static uint32_t cycle = 0;
24 static uint32_t cyclemax = 10;
25 static TIMER_STATUS_t timer_status;
26 static uint32_t edge_count = 0;
27 static uint32_t freq = 0;
28 static uint32_t uart_out_ir;
29 static uint32_t uart_out_rgb[4];
30 static uint8_t semaphore = 0;
31 static uint8_t colour = 0;
32 /**
33
34  * @brief main() - Application entry point
35  *
```

```
36  * <b>Details of function</b><br>
37  * This routine is the application entry point. It is invoked by the device
        startup code. It is responsible for
38  * invoking the APP initialization dispatcher routine - DAVE_Init() and
       hosting the place-holder for user application
39  * code.
40  */
41 void DataHandler(void)
42 {
43 cycle++; // runden hochzaehlen
44 
45   if(cycle<cyclemax){
46     freq = freq+edge_count; // aufaddieren von edge_count
47     edge_count = 0;//zuruecksetzen von edge_count
48   }
49 
50   if(cycle==cyclemax-1)
51   {
52     if(LED==STATE_LED_IR)
53     {
54       uart_out_ir = freq/cyclemax;//mittelwert bliden
55     }
56     else
57     {
58       uart_out_rgb[colour] =  freq/cyclemax;//mittelwert bliden
59     }
60 
61   }
62   if(cycle==cyclemax)
63   {
64     cycle = 0; //zuruecksetzen von cycle
65     freq = 0; //zuruecksetzen von freq
66 
67     semaphore = 1;
68   }
69 
70   TIMER_ClearEvent(&TIMER_0); // Timerevent zuruecksetzen
71 }
72 
73 
74 void EdgeCountHandler(void)
75 {
76   edge_count++; // detect a rising edge
77 }
```

```
78
79 int main(void)
80 {
81   DAVE_STATUS_t status;
82   uint32_t pin_status;
83   //int i;
84   char Send_Data[8];
85   uint32_t cycle_old = 0;
86
87   uint8_t LED_COLOUR = STATE_LED_IR;
88   uint8_t Send_Data_red[] = "r";
89   uint8_t Send_Data_green[] = "g";
90   uint8_t Send_Data_blue[] = "b";
91   uint8_t Send_Data_out[] = "o";
92   uint8_t Send_Data_ir[] = "i";
93   uint8_t answer[]="Eingabe erfolgt\n";
94
95   uint8_t ReadData[2];
96
97   bool receive_int_time = false;
98   bool receive_gain = false;
99   bool receive_led = false;
100
101  uint8_t Data_Gain;
102  uint8_t Data_IntTime;
103  uint8_t Data_LED;
104
105  status = DAVE_Init();           /* Initialization of DAVE APPs  */
106
107  if(status != DAVE_STATUS_SUCCESS)
108  {
109    /* Placeholder for error handler code. The while loop below can be
     replaced with an user error handler. */
110    XMC_DEBUG("DAVE APPs initialization failed\n");
111
112    while(1U)
113    {
114
115    }
116  }
117  else
118  {
119
120  }
```

```
121   DIGITAL_IO_SetOutputLow(&not_EN1_out);
122   DIGITAL_IO_SetOutputLow(&not_EN2_out);
123
124   timer_status = TIMER_Start(&TIMER_0); // timer starten
125
126   while(1U)
127   {
128     if(cycle==0)
129     {
130       switch(LED){
131       case 1:{
132         DIGITAL_IO_SetOutputHigh(&IR_LED_out);
133         break;
134       }
135       case 2:{
136         colour = 0;
137         DIGITAL_IO_SetOutputHigh(&RGB_R_out);
138         break;
139       }
140       case 3:{
141         colour = 1;
142         DIGITAL_IO_SetOutputHigh(&RGB_G_out);
143         break;
144       }
145
146       case 4:{
147         colour = 2;
148         DIGITAL_IO_SetOutputHigh(&RGB_B_out);
149         break;
150       }
151       case 5:{
152         colour = 3;
153         break;
154       }
155       }
156
157       if(semaphore==1)
158       {
159
160         switch(LED){
161         case 1:{
162           DIGITAL_IO_SetOutputLow(&IR_LED_out);
163           sprintf(Send_Data, "%" PRIu32 "\n", uart_out_ir); // uint32_t in
    string wandeln
```

```
164           UART_Transmit(&UART_0, Send_Data_ir, sizeof(Send_Data_ir)-1);
165           UART_Transmit(&UART_0, Send_Data, strlen(Send_Data)); // daten
    ueber den UART Bus senden
166           semaphore=0;
167           LED = STATE_LED_NOISE;//LED = STATE_LED_N;
168
169           if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
170           {
171             if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
172             {
173               LED=STATE_LED_NOISE;
174             }
175             if((ReadData[0]=='e')||(ReadData[1]=='e'))
176             {
177               UART_Transmit(&UART_0, answer, sizeof(answer)-1);
178               LED=STATE_COMANND;
179             }
180           }
181           break;
182         }
183         case 2:{
184           DIGITAL_IO_SetOutputLow(&RGB_R_out);
185           sprintf(Send_Data, "%" PRIu32 "\n", uart_out_rgb[0]); // uint32_t
     in string wandeln
186           UART_Transmit(&UART_0, Send_Data_red, sizeof(Send_Data_red)-1);
187           UART_Transmit(&UART_0, Send_Data, strlen(Send_Data)); // daten
    ueber den UART Bus senden
188           semaphore=0;
189           LED = STATE_LED_GREEN;
190
191           if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
192           {
193             if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
194             {
195               LED=STATE_LED_GREEN;
196             }
197             if((ReadData[0]=='e')||(ReadData[1]=='e'))
198             {
199               UART_Transmit(&UART_0, answer, sizeof(answer)-1);
200               LED=STATE_COMANND;
201             }
202           }
203           break;
204         }
```

```
205         case 3:{
206           DIGITAL_IO_SetOutputLow(&RGB_G_out);
207           sprintf(Send_Data, "%" PRIu32 "\n", uart_out_rgb[1]); // uint32_t
     in string wandeln
208           UART_Transmit(&UART_0, Send_Data_green, sizeof(Send_Data_green)
    -1);
209           UART_Transmit(&UART_0, Send_Data, strlen(Send_Data)); // daten
    ueber den UART Bus senden
210           semaphore=0;
211           LED = STATE_LED_BLUE;
212
213           if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
214           {
215             if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
216             {
217               LED=STATE_LED_BLUE;
218             }
219             if((ReadData[0]=='e')||(ReadData[1]=='e'))
220             {
221               UART_Transmit(&UART_0, answer, sizeof(answer)-1);
222               LED=STATE_COMANND;
223             }
224           }
225           break;
226         }
227         case 4:{
228           DIGITAL_IO_SetOutputLow(&RGB_B_out);
229           sprintf(Send_Data, "%" PRIu32 "\n", uart_out_rgb[2]); // uint32_t
     in string wandeln
230           UART_Transmit(&UART_0, Send_Data_blue, sizeof(Send_Data_blue)-1);
231           UART_Transmit(&UART_0, Send_Data, strlen(Send_Data)); // daten
    ueber den UART Bus senden
232           semaphore=0;
233           LED = STATE_LED_NOISE;
234
235           if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
236           {
237             if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
238             {
239               LED=STATE_LED_NOISE;
240             }
241             if((ReadData[0]=='e')||(ReadData[1]=='e'))
242             {
243               UART_Transmit(&UART_0, answer, sizeof(answer)-1);
```

```
244                 LED=STATE_COMANND;
245               }
246             }
247             break;
248           }
249           case 5:{
250             sprintf(Send_Data, "%" PRIu32 "\n", uart_out_rgb[3]); // uint32_t
      in string wandeln
251             UART_Transmit(&UART_0, Send_Data_out, sizeof(Send_Data_out)-1);
252             UART_Transmit(&UART_0, Send_Data, strlen(Send_Data)); // daten
      ueber den UART Bus senden
253             semaphore=0;
254             if(LED_COLOUR==STATE_LED_IR)
255             {
256               LED = STATE_LED_IR;
257
258               if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
259               {
260                 if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
261                 {
262                   LED=STATE_LED_IR;
263                 }
264                 if((ReadData[0]=='e')||(ReadData[1]=='e'))
265                 {
266                   UART_Transmit(&UART_0, answer, sizeof(answer)-1);
267                   LED=STATE_COMANND;
268                 }
269               }
270               break;
271             }
272             else
273             {
274               LED = STATE_LED_RED;
275
276               if(UART_Receive(&UART_0, ReadData, 2) == UART_STATUS_SUCCESS)
277               {
278                 if((ReadData[0]=='\n')&&(ReadData[1]=='\n'))
279                 {
280                   LED=STATE_LED_RED;
281                 }
282                 if((ReadData[0]=='e')||(ReadData[1]=='e'))
283                 {
284                   UART_Transmit(&UART_0, answer, sizeof(answer)-1);
285                   LED=STATE_COMANND;
```

```
286                 }
287               }
288               break;
289             }
290           }
291           case 6:{
292             receive_int_time = false;
293             receive_gain = false;
294             receive_led = false;
295             while(receive_int_time == false)
296             {
297               receive_int_time = false;
298               if(UART_Receive(&UART_0, ReadData, 1) == UART_STATUS_SUCCESS)
299               {
300                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
301                 cyclemax=ReadData[0];
302                 receive_int_time=true;
303               }
304             }
305             while(receive_gain == false)
306             {
307               receive_gain = false;
308               if(UART_Receive(&UART_0, ReadData, 2)==UART_STATUS_SUCCESS)
309               {
310                 if(ReadData[0]=='\n')
311                 {
312                   Data_Gain=ReadData[1];
313                 }
314                 else
315                 {
316                   Data_Gain=ReadData[0];
317                 }
318                 switch(Data_Gain)
319                 {
320                 case '0':
321                   //Gain auf  Stufe 1 stellen
322                   UART_Transmit(&UART_0, answer, sizeof(answer)-1);
323                   DIGITAL_IO_SetOutputLow(&not_EN1_out);
324                   DIGITAL_IO_SetOutputHigh(&not_EN2_out);
325                   receive_gain = true;
326                   break;
327
328                 case '1':
329                   //Gain auf Stufe 2 stellen
```

```
330                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
331                 DIGITAL_IO_SetOutputHigh(&not_EN1_out);
332                 DIGITAL_IO_SetOutputLow(&not_EN2_out);
333                 receive_gain = true;
334                 break;
335
336               case '2':
337                 //Gain auf Stufe 3 stellen
338                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
339                 DIGITAL_IO_SetOutputLow(&not_EN1_out);
340                 DIGITAL_IO_SetOutputLow(&not_EN2_out);
341                 receive_gain = true;
342                 break;
343               }
344             }
345           }
346
347           while(receive_led == false)
348           {
349             receive_led = false;
350             if(UART_Receive(&UART_0, ReadData, 2)==UART_STATUS_SUCCESS)
351             {
352               if(ReadData[0]=='\n')
353               {
354                 Data_LED=ReadData[1];
355               }
356               else
357               {
358                 Data_LED=ReadData[0];
359               }
360               switch(Data_LED)
361               {
362               case 'i':
363                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
364                 LED=STATE_LED_IR;
365                 LED_COLOUR=STATE_LED_IR;
366                 cycle=0;
367                 receive_led = true;
368                 break;
369               case 'r':
370                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
371                 LED=STATE_LED_RED;
372                 LED_COLOUR=STATE_LED_RED;
373                 cycle=0;
```

```
374                 receive_led = true;
375                 break;
376               case 'g':
377                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
378                 LED=STATE_LED_GREEN;
379                 LED_COLOUR=STATE_LED_RED;
380                 cycle=0;
381                 receive_led = true;
382                 break;
383               case 'b':
384                 UART_Transmit(&UART_0, answer, sizeof(answer)-1);
385                 LED=STATE_LED_BLUE;
386                 LED_COLOUR=STATE_LED_RED;
387                 cycle=0;
388                 receive_led = true;
389                 break;
390               }
391             }
392           }
393           break;
394         }
395         }
396       }
397     }
398   }
399 }
```

A.3.2 Matlab Code

```
1  %% Auswahl des Sensors und der LED
2  Transistor = 0;
3  Farbsensor = 1;
4  sensor = Transistor;
5
6  IR = 0;
7  RGB = 1;
8  LED = IR;
9  % Speicher Matrizen und Variablen anlegen
10 x = 0:1000;
11 if sensor == Transistor
12     data_red1 = "0";
13     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
14     time_stamp_red = datenum(t);
```

```
15      data_green1 = "0";
16      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
17      time_stamp_green = datenum(t);
18      data_blue1 = "0";
19      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
20      time_stamp_blue = datenum(t);
21      data_ir = "0";
22      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
23      time_stamp_ir = datenum(t);
24      data_out1 = "0";
25      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
26      time_stamp_out = datenum(t);
27
28
29      data_led_red = zeros(2,1001);
30      data_led_green = zeros(2,1001);
31      data_led_blue = zeros(2,1001);
32      data_led_out = zeros(2,1001);
33      data_led_ir = zeros(2,1001);
34  else
35      data_red1 = "0";
36      data_red2 = "0";
37      data_red3 = "0";
38      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
39      time_stamp_red = datenum(t);
40      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
41      data_green1 = "0";
42      data_green2 = "0";
43      data_green3 = "0";
44      time_stamp_green = datenum(t);
45      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
46      data_blue1 = "0";
47      data_blue2 = "0";
48      data_blue3 = "0";
49      time_stamp_blue = datenum(t);
50      data_ir = "0";
51      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
52      time_stamp_ir = datenum(t);
53      data_out1 = "0";
54      data_out2 = "0";
55      data_out3 = "0";
56      t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
57      time_stamp_out = datenum(t);
58
```

```
59      data_led_red = zeros(4,1001);
60      data_led_green = zeros(4,1001);
61      data_led_blue = zeros(4,1001);
62      data_led_out = zeros(4,1001);
63      data_led_ir = zeros(2,1001);
64  end
65
66  data_number = 0;
67  ending = 0;
68  matrix_save = 0;
69  matrix_datapoints = 100;
70
71  delaySec = 10;
72
73
74  % Uart-Verbindung herstellen
75  device = serialport("COM5",115200);
76  device.Timeout = 30;
77
78  %% Komando um Einstellungen vorzunehmen
79  writeline(device,"e");
80  writeline(device,"e");
81  %% Komando um Integrationszeit festzulegen
82  IntTime = 192;
83  write(device,IntTime,"uint8");
84  write(device,IntTime,"uint8");
85
86  %% Komando um Gain festzulegen
87  writeline(device,"2");
88  %% Komando um LED auszuwaehlen
89  writeline(device,"r");
90  %%
91  while true
92      recevied = true;
93      while recevied
94          device.BytesAvailableFcnCount = 1;
95          device.BytesAvailableFcnMode = 'byte';
96          device.BytesAvailableFcn = @instrcallback;
97          device.BytesAvailable
98          choose_color1 = read(device,1,"string")
99
100         choose_color = readline(device)
101          choose_color = readline(device)
102           choose_color = readline(device)
```

```
103
104         if 0==strcmp(choose_color,'')
105             recevied = false;
106         end
107     end
108 end
109 %% Plot erzeugen
110
111 f1 = figure;
112 figure(f1)
113
114 while true
115
116     if ending==1001
117         ending = 0;
118         matrix_save = 0;
119         old_ending = 1;
120         wraparound = 1;%nach 1000 datenpunkten von vorne beginnen
121     end
122     ending = ending+1;
123     matrix_save = matrix_save+1;
124
125     if sensor == Transistor
126         if LED==RGB
127             recevied = true;
128             while recevied
129                 choose_color = read(device,1,"string");
130                 if 0==strcmp(choose_color,'')
131                     recevied = false;
132                 end
133             end
134             switch choose_color
135                 case "r"
136                     data_red1 = readline(device);
137                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
138                     time_stamp_red = datenum(t);
139                     pause(delaySec);
140
141                 case "g"
142                     data_green1 = readline(device);
143                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
144                     time_stamp_green = datenum(t);
145                     pause(delaySec);
146
```

```
147                 case "b"
148                     data_blue1 = readline(device);
149                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
150                     time_stamp_blue = datenum(t);
151                     pause(delaySec);
152 
153                 case "o"
154                     data_out1 = readline(device);
155                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
156                     time_stamp_out = datenum(t);
157                     pause(delaySec);
158             end
159         else
160             recevied = true;
161             while recevied
162                 choose_color = read(device,1,"string");
163                 if 0==strcmp(choose_color,'')
164                     recevied = false;
165                 end
166             end
167             switch choose_color
168                 case "i"
169                     data_ir = readline(device);
170                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
171                     time_stamp_ir = datenum(t);
172                     pause(delaySec);
173                 case "o"
174                     data_out1 = readline(device);
175                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
176                     time_stamp_out = datenum(t);
177                     pause(delaySec);
178             end
179         end
180         figure(f1)
181 
182         if LED==RGB
183             subplot(4,1,1)
184             plot(x(1:ending-1),data_led_red(1,1:ending-1),'r')
185             title('Rote LED ')
186             xlabel('Datapiont')
187             ylabel('Cumulative Frequency')
188 
189             subplot(4,1,2)
190             plot(x(1:ending-1),data_led_green(1,1:ending-1),'g')
```

```
191             title('Gruene LED ')
192             xlabel('Datapiont')
193             ylabel('Cumulative Frequency')
194 
195             subplot(4,1,3)
196             plot(x(1:ending-1),data_led_blue(1,1:ending-1),'b')
197             title('Blaue LED ')
198             xlabel('Datapiont')
199             ylabel('Cumulative Frequency')
200 
201             subplot(4,1,4)
202             plot(x(1:ending-1),data_led_out(1,1:ending-1),'k')
203             title('LED aus ')
204             xlabel('Datapiont')
205             ylabel('Cumulative Frequency')
206 
207         else
208             subplot(2,1,1)
209             plot(x(1:ending-1),data_led_ir(1,1:ending-1),'m')
210             title('IR-LED ')
211             xlabel('Datapiont')
212             ylabel('Cumulative Frequency')
213 
214             subplot(2,1,2)
215             plot(x(1:ending-1),data_led_out(1,1:ending-1),'k')
216             title('IR-LED aus ')
217             xlabel('Datapiont')
218             ylabel('Cumulative Frequency')
219         end
220         drawnow;
221 
222         if LED==RGB
223             data_led_red(1,ending) = str2double(data_red1);
224             data_led_red(2,ending) = time_stamp_red;
225 
226             data_led_green(1,ending) = str2double(data_green1);
227             data_led_green(2,ending) = time_stamp_green;
228 
229             data_led_blue(1,ending) = str2double(data_blue1);
230             data_led_blue(2,ending) = time_stamp_blue;
231 
232             data_led_out(1,ending) = str2double(data_out1);
233             data_led_out(2,ending) = time_stamp_out;
234         else
```

```
235             data_led_ir(1,ending) = str2double(data_ir);
236             data_led_ir(2,ending) = time_stamp_ir;
237 
238             data_led_out(1,ending) = str2double(data_out1);
239             data_led_out(2,ending) = time_stamp_out;
240         end
241 
242         if matrix_save==matrix_datapoints
243             data_number = data_number+1;
244             if data_number == 1
245                 old_ending=1;
246             else
247                 old_ending=ending-matrix_datapoints;
248             end
249             if LED==RGB
250                 writematrix(data_led_red(:,old_ending:ending),sprintf('
    Transistor_Red\\TransistorDataRed%d.csv', data_number));
251 
252                 writematrix(data_led_green(:,old_ending:ending),sprintf('
    Transistor_Green\\TransistorDataGreen%d.csv', data_number));
253 
254                 writematrix(data_led_blue(:,old_ending:ending),sprintf('
    Transistor_Blue\\TransistorDataBlue%d.csv', data_number));
255 
256                 writematrix(data_led_out(:,old_ending:ending),sprintf('
    Transistor_Out\\TransistorDataOutRGB%d.csv', data_number));
257             else
258                 writematrix(data_led_ir(:,old_ending:ending),sprintf('
    Transistor_IR\\TransistorDataIR%d.csv', data_number));
259 
260                 writematrix(data_led_out(:,old_ending:ending),sprintf('
    Transistor_IROut\\TransistorDataOutIR%d.csv', data_number));
261             end
262             matrix_save=0;
263         end
264 
265 
266     else
267         if LED==RGB
268             recevied = true;
269             while recevied
270                 choose_color = read(device,1,"string");
271                 if 0==strcmp(choose_color,"")
272                     recevied = false;
```

```
273                 end
274             end
275             switch choose_color
276                 case "r"
277                     data_red1 = readline(device);
278                     data_red2 = readline(device);
279                     data_red3 = readline(device);
280                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
281                     time_stamp_red = datenum(t);
282 
283                 case "g"
284                     data_green1 = readline(device);
285                     data_green2 = readline(device);
286                     data_green3 = readline(device);
287                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
288                     time_stamp_green = datenum(t);
289 
290                 case "b"
291                     data_blue1 = readline(device);
292                     data_blue2 = readline(device);
293                     data_blue3 = readline(device);
294                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
295                     time_stamp_blue = datenum(t);
296 
297                 case "o"
298                     data_out1 = readline(device);
299                     data_out2 = readline(device);
300                     data_out3 = readline(device);
301                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
302                     time_stamp_out = datenum(t);
303 
304             end
305         else
306             recevied = true;
307             while recevied
308                 choose_color = read(device,1,"string");
309                 if 0==strcmp(choose_color,'')
310                     recevied = false;
311                 end
312             end
313             switch choose_color
314                 case "i"
315                     data_ir = readline(device);
316                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
```

```
317                     time_stamp_ir = datenum(t);
318                 case "o"
319                     data_out1 = readline(device);
320                     t = datestr(now,'dd mmmm yyyy HH:MM:SS.FFF ');
321                     time_stamp_out = datenum(t);
322             end
323
324         end
325
326
327
328         figure(f1)
329
330         if LED==RGB
331             subplot(4,3,1)
332             plot(x(1:ending-1),data_led_red(1,1:ending-1),'r')
333             title('Rote LED Roter Sensor')
334             xlabel('Datapiont')
335             ylabel('Intensity')
336             subplot(4,3,2)
337             plot(x(1:ending-1),data_led_red(2,1:ending-1),'g')
338             title('Rote LED Gruener Sensor')
339             xlabel('Datapiont')
340             ylabel('Intensity')
341             subplot(4,3,3)
342             plot(x(1:ending-1),data_led_red(3,1:ending-1),'b')
343             title('Rote LED Blauer Sensor')
344             xlabel('Datapiont')
345             ylabel('Intensity')
346             subplot(4,3,4)
347             plot(x(1:ending-1),data_led_green(1,1:ending-1),'r')
348             title('Gruene LED Roter Sensor ')
349             xlabel('Datapiont')
350             ylabel('Intensity')
351             subplot(4,3,5)
352             plot(x(1:ending-1),data_led_green(2,1:ending-1),'g')
353             title('Gruene LED Gruener Sensor ')
354             xlabel('Datapiont')
355             ylabel('Intensity')
356             subplot(4,3,6)
357             plot(x(1:ending-1),data_led_green(3,1:ending-1),'b')
358             title('Gruene LED Blauer Sensor ')
359             xlabel('Datapiont')
360             ylabel('Intensity')
```

```
361             subplot(4,3,7)
362             plot(x(1:ending-1),data_led_blue(1,1:ending-1),'r')
363             title('Blaue LED Roter Sensor ')
364             xlabel('Datapiont')
365             ylabel('Intensity')
366             subplot(4,3,8)
367             plot(x(1:ending-1),data_led_blue(2,1:ending-1),'g')
368             title('Blaue LED Gruener Sensor ')
369             xlabel('Datapiont')
370             ylabel('Intensity')
371             subplot(4,3,9)
372             plot(x(1:ending-1),data_led_blue(3,1:ending-1),'b')
373             title('Blaue LED Blauer Sensor ')
374             xlabel('Datapiont')
375             ylabel('Intensity')
376             subplot(4,3,10)
377             plot(x(1:ending-1),data_led_out(1,1:ending-1),'r')
378             title('LED aus Roter Sensor ')
379             xlabel('Datapiont')
380             ylabel('Intensity')
381             subplot(4,3,11)
382             plot(x(1:ending-1),data_led_out(2,1:ending-1),'g')
383             title('LED aus Gruener Sensor ')
384             xlabel('Datapiont')
385             ylabel('Intensity')
386             subplot(4,3,12)
387             plot(x(1:ending-1),data_led_out(3,1:ending-1),'b')
388             title('LED aus Blauer Sensor ')
389             xlabel('Datapiont')
390             ylabel('Intensity')
391         else
392             subplot(1,2,1)
393             plot(x(1:ending-1),data_led_ir(1,1:ending-1),'m')
394             title('IR-LED IR-Sensor ')
395             xlabel('Datapiont')
396             ylabel('Intensity')
397             subplot(1,2,2)
398             plot(x(1:ending-1),data_led_out(1,1:ending-1),'k')
399             title('IR-LED aus IR-Sensor ')
400             xlabel('Datapiont')
401             ylabel('Intensity')
402         end
403         drawnow;
404
```

```
405         if LED==RGB
406             data_led_red(1,ending) = str2double(data_red1);
407             data_led_red(2,ending) = str2double(data_red2);
408             data_led_red(3,ending) = str2double(data_red3);
409             data_led_red(4,ending) = time_stamp_red;
410 
411             data_led_green(1,ending) = str2double(data_green1);
412             data_led_green(2,ending) = str2double(data_green2);
413             data_led_green(3,ending) = str2double(data_green3);
414             data_led_green(4,ending) = time_stamp_green;
415 
416             data_led_blue(1,ending) = str2double(data_blue1);
417             data_led_blue(2,ending) = str2double(data_blue2);
418             data_led_blue(3,ending) = str2double(data_blue3);
419             data_led_blue(4,ending) = time_stamp_blue;
420 
421             data_led_out(1,ending) = str2double(data_out1);
422             data_led_out(2,ending) = str2double(data_out2);
423             data_led_out(3,ending) = str2double(data_out3);
424             data_led_out(4,ending) = time_stamp_out;
425         else
426             data_led_ir(1,ending) = str2double(data_ir);
427             data_led_ir(2,ending) = time_stamp_ir;
428 
429             data_led_out(1,ending) = str2double(data_out1);
430             data_led_out(4,ending) = time_stamp_out;
431         end
432 
433 
434 
435         if matrix_save==matrix_datapoints
436             data_number = data_number+1;
437             if data_number == 1||wraparound==1
438                 old_ending=1;
439                 wraparound=0;
440             else
441                 old_ending=ending-matrix_datapoints;
442             end
443             test=old_ending:ending;
444             if LED==RGB
445                 writematrix(data_led_red(:,old_ending:ending),sprintf('
    Sensor_Red\\SensorDataRed%d.csv', data_number));
446 
```

```
447                 writematrix(data_led_green(:,old_ending:ending),sprintf('
    Sensor_Green\\SensorDataGreen%d.csv', data_number));
448
449                 writematrix(data_led_blue(:,old_ending:ending),sprintf('
    Sensor_Blue\\SensorDataBlue%d.csv', data_number));
450
451                 writematrix(data_led_out(:,old_ending:ending),sprintf('
    Sensor_Out\\SensorDataOutRGB%d.csv', data_number));
452             else
453                 matrix_ir=data_led_ir(:,old_ending:ending);
454                 writematrix(matrix_ir,sprintf('Sensor_IR\\SensorDataIR%d.
    csv', data_number));
455                 matrix_out_ir=data_led_out(:,old_ending:ending);
456                 writematrix(matrix_out_ir,sprintf('Sensor_IROut\\
    SensorDataOutIR%d.csv', data_number));
457             end
458             matrix_save=0;
459         end
460     end
461 end
462 %%
463
464 Transistor = 0;
465 Farbsensor = 1;
466 sensor = Farbsensor;
467
468 IR = 0;
469 RGB = 1;
470 LED = IR;
471
472 x = 0:50;
473 data_red = zeros(4,1001);
474 data_green = zeros(4,1001);
475 data_blue = zeros(4,1001);
476 data_out = zeros(4,1001);
477 data_ir = zeros(2,1001);
478
479 %data_ir = readmatrix('SensorDataIR1.csv');
480 %data_out_ir = readmatrix('SensorDataOutIR1.csv');
481
482 f1 = figure;
483 figure(f1)
484
485 if sensor == Transistor
```

```
486
487     if LED == RGB
488         data_red = readmatrix('TransistorDataRed1.csv');
489         data_green = readmatrix('TransistorDataGreen1.csv');
490         data_blue = readmatrix('TransistorDataBlue1.csv');
491         data_out = readmatrix('TransistorDataOutRGB1.csv');
492
493         plot(x,data_red(1,:),'r',x,data_green(1,:),'g',x,data_blue(1,:),'b'
    ,x,data_out(1,:),'k')
494         title('Rote LED Roter Sensor')
495         xlabel('Datapiont')
496         ylabel('Cumulative Frequency')
497     else
498         data_ir = readmatrix('TransistorDataIR10.csv');
499         data_out = readmatrix('TransistorDataOutIR10.csv');
500
501         plot(x,data_ir(1,:),'m',x,data_out(1,:),'k')
502         title('Rote LED Roter Sensor')
503         xlabel('Datapiont')
504         ylabel('Cumulative Frequency')
505     end
506
507 else
508
509     if LED == RGB
510         data_red = readmatrix('SensorDataRed1.csv');
511         data_green = readmatrix('SensorDataGreen1.csv');
512         data_blue = readmatrix('SensorDataBlue1.csv');
513         data_out_rgb = readmatrix('SensorDataOutRGB1.csv');
514
515         subplot(3,1,1)
516         plot(x,data_red(1,:),'r',x,data_red(2,:),'g',x,data_red(3,:),'b',x,
    data_out_rgb(1,:),'k')
517         title('Rote LED Roter Sensor')
518         xlabel('Datapiont')
519         ylabel('Intensity')
520         subplot(3,1,2)
521         plot(x,data_green(1,:),'r',x,data_green(2,:),'g',x,data_green(3,:),
    'b',x,data_out_rgb(2,:),'k')
522         title('Rote LED Roter Sensor')
523         xlabel('Datapiont')
524         ylabel('Intensity')
525         subplot(3,1,3)
```

```
526         plot(x,data_blue(1,:),'r',x,data_blue(2,:),'g',x,data_blue(3,:),'b'
    ,x,data_out_rgb(3,:),'k')
527         title('Rote LED Roter Sensor')
528         xlabel('Datapiont')
529         ylabel('Intensity')
530     else
531         data_ir = readmatrix('SensorDataIR20.csv');
532         data_out_ir = readmatrix('SensorDataOutIR20.csv');
533 
534         plot(x,data_ir(1,:),'m',x,data_out_ir(1,:),'k')
535         title('Rote LED Roter Sensor')
536         xlabel('Datapiont')
537         ylabel('Intensity')
538     end
539 end
```

A.4 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
Dr. rer. nat. Florian Rittweger

24.07.2020

Bachelorthesis Simon von Riegen

Controllersteuerung eines faseroptischen Sensors für Lithium-Ionen-Batterien

Motivation

Für eine zukünftige Verwendung von Lithium-Ionen-Batterien im Automobilbereich ist eine verbesserte Überwachung des Batteriezustandes, insbesondere des Ladezustandes (*state of charge*) wünschenswert. Dabei gibt es verschiedene Möglichkeiten die bisherigen elektrischen Messmethoden zu erweitern, z. B. durch die elektrochemische Impedanzspektroskopie (EIS), als auch komplementäre, nicht-elektrische Messmethoden zu implementieren.
Eine Methode stellt dabei die in-situ Batterieüberwachung mit Hilfe optischer Lichtleitfasern dar. Dabei wird der aus Vorarbeiten bekannte Zusammenhang zwischen elektrischen und optischen Eigenschaften der Batteriezelle ausgenutzt. Bringt man eine Lichtleitfaser in die Batterieelektrode ein und misst das Transmissionssignal des eingespeisten Lichtes, kann eine Veränderung der gemessenen Lichtleistung in Abhängigkeit des Ladezustandes der Batterie ermittelt werden. Typischerweise erfolgt eine solche Messung noch unter Laborbedingungen mittels breitbandiger Halogenlichtquelle und Spektrometer.
An der HAW wird im Rahmen eines BMWi-Verbundprojektes mit Industriepartnern an optischer Sensorik für Fahrzeugbatterien gearbeitet.

Aufgabe

Ziel der Arbeit ist es deshalb, ein mobiles und miniaturisiertes System in Hard- und Software eines solchen Laboraufbaus zu entwickeln. Für eine Realisierung sollen ersatzweise Multicolor-RGB sowie Infrarot-LEDs als Lichtquelle und ein Phototransistor sowie ein Lichtsensor als Lichtempfänger verwendet werden. Hierfür ist die Entwicklung einer geeigneten Platine notwendig. Eine Steuerung der Messung soll auf Mikrocontrollerbasis realisiert werden. Die Funktionalität des Sensorsystems wird an einer Lichtleitfaser, welche sich in Lösungen verschiedener Konzentration befindet, in Ersatz für eine reale Batterie getestet. Die gemessene Lichtleistung ist dabei abhängig von der Konzentration. Für die Abschlussarbeit sind die folgenden Arbeitspakete geplant:

1. Einarbeitung und Recherche
    - Einordnung in die Projektzielstellung an der HAW, Konzepte für Batteriemanagement, Erläuterung Forschungsbedarf
    - Einarbeitung in die Grundlagen faseroptischer Sensoren
    - Recherche von vergleichbaren Vorarbeiten an der HAW, bspw. Klockmann, Nazimzada und andere.
    - Inbetriebnahme der Entwicklungsumgebung für ARM-Mikrocontroller, bspw. DAvE (Infineon)
    - Grundlagenrecherche zu Bauelementen und Schnittstellen

2. Konzeption

    - Spezifikation der Anforderungen
    - Design der Faserhalterung mit Biegung (Sonde)
    - Systemarchitektur als Übersicht (Blockschaltbild, etc.)
    - Grundsätzlicher Aufbau des Systems mit Schnittstellen
    - Entwurf der Steuerungs- und Softwarestruktur
    - Festlegung von Dateiformaten, Bedienoberfläche und Funktionsumfang

3. Konstruktion des Messaufbaus

    - CAD-Konstruktion der Adapter zur Ankopplung der Lichtleitfasern
    - Fertigung mit 3D-Druck
    - Montage und Erprobung, ggfs. Korrektur

4. Entwicklung der Hardware

    - Auswahl geeigneter Bauelemente als Lichtquelle und -sensor
    - Entwurf der Analogbaugruppen und Stromversorgung
    - Definition der Schnittstellen zu den Controllerboards
    - Platinenentwurf und Bestückung
    - Inbetriebnahme und Test

5. Entwicklung der Controllersoftware

    - Ansteuerung der Lichtquellen
    - Digitale Schnittstelle für integrierten Lichtsensor/Farbsensor
    - Schnittstelle zum PC, Auswahl und Begründung
    - Modul- und Gesamtfunktionstest gemäß Testplan

6. Entwicklung der PC-Software

    - Umsetzung als Matlab-Skript
    - Schnittstelle zum Controller
    - Visualisierung von Messdaten
    - Aufzeichnung der Messdaten mit Zeitstempel in Vorbereitung einer Synchronisation mit einem Zykliersystem
    - Modul- und Gesamtfunktionstest gemäß Testplan

7. Erprobung des gesamten Messsystems

    - Planung von Messreihen
    - Durchführung und Auswertung der Messungen
    - Untersuchung des Zusammenhangs zwischen Konzentration der Lösung und gemessener Lichtleistung
    - Auswertung von Einflussgrößen und Streuungen, Kalibrierung bzw. Referenzmessungen für Auswertung
    - Untersuchung des Aufbaus auf Reproduzierbarkeit

8. Einordnung, Bewertung und Ausblick

    - Zusammenfassung der Ergebnisse, Beurteilung des Messkonzeptes
    - Bewertung der gewählten Konzepte und Lösungsvarianten
    - Offene Punkte und einschränkende Erfahrungen und Beobachtungen

Hinweise zur Durchführung

Hardware

- Konstruktion, Bestückung und In-Betriebnahme einer Platine, welche im Wesentlichen die Ansteuerung verschiedener LED-Lichtquellen (Multicolor-RGB, IR) eines Phototransistor und eines Lichtsensor ermöglicht. Die Ansteuerung soll dabei mit dem Mikrocontroller XMC1100 erfolgen. Dieser soll eine UART-Schnittstelle zu PC oder einem übergeordneten Controller bieten.
- Die Platine soll so aufgebaut sein, dass sie auf den Mikrocontroller XMC4700 aufgesetzt werden kann und die Steuerung der Messung dann vom XMC4700 aus via UART möglich ist.
- Konstruktion und Fertigung von Anschlussmöglichkeiten optischer F-SMA Stecker (SMA 905) mittels 3D-Druck, welche über den LEDs bzw. Lichtsensoren auf der Platine platziert werden.
- Konstruktion und Fertigung einer Sonde mittels 3D-Druck, in welche eine optische Lichtleitfaser gespannt und welche anschließend in einem mit Flüssigkeit gefüllten Gefäß platziert werden kann.

Software

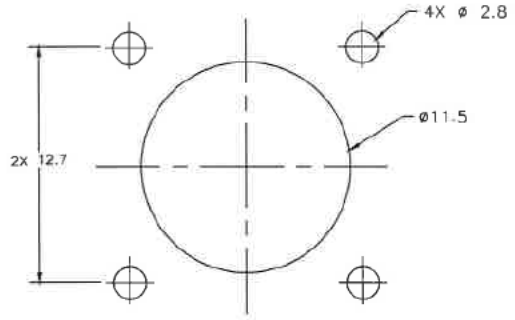
- Die Steuerung der Messung der Lichttransmissionsleistung der optischen Faser soll wahlweise mit den Mikrocontrollern XMC1100 erfolgen.
- Verwendung der Enwicklungsumgebung DAvE (Infineon).
- Codeentwicklung für die Ansteuerung der LEDs sowie das Auslesen des Phototransistors bzw. des Lichtsensors.
- Die on-board Elektronik ist direkt mit XMC1100 verbunden. Für eine stand-alone Nutzung soll die Nutzerbedienung über UART vom PC erfolgen. Im Falle der Verwendung über den XMC4700 ist der PC über Ethernet an das Sensorsystem angeschlossen.
- Bereitstellung und Speicherung der Messdaten in geeignetem Format auf dem PC. Zu beachten ist, dass Messungen typischerweise mindestens eine Woche Laufzeit haben.
- Darstellung des Messergebnisses auf dem PC, nach Möglichekeit als Live-Update.

Dokumentation

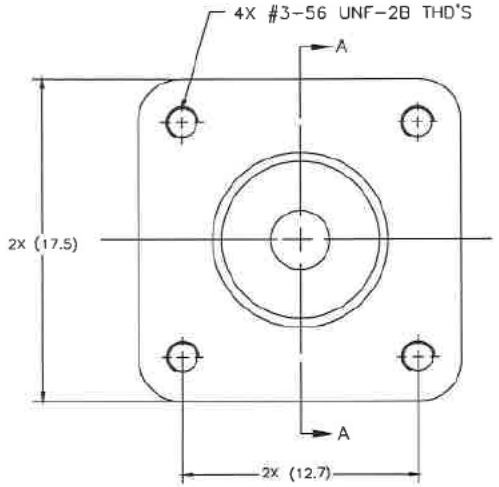
Die Fachliteratur und die kommerziellen Unterlagen bzw. Datenblätter sind zielgerichtet zu recherchieren. Dabei sind insbesondere wichtige Grundlagen der Batterieeffekte und der Lichtleiteroptik näher zu betrachten. Die gesetzten Rahmenbedingungen, die gewählten Lösungen und die Funktionsweise sind gut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

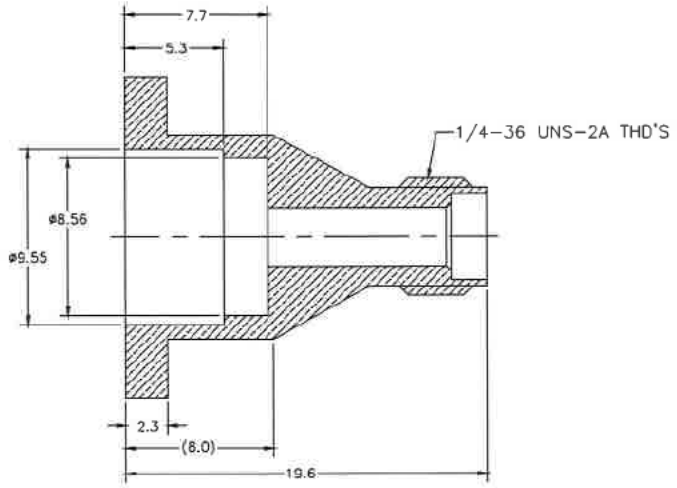
A.5 Wichtige Datenblätter

GEN NOTES:
1. AS SHIPPED, THE P/N CONTAINS THE FOLLOWING PIECES: RECEPTACLE BODY AND PROTECTIVE CAP, UNLESS OTHERWISE SPECIFIED.
2. DUST CAP IS SHOWN EXPLODED. DUST CAP SHOULD BE ASSEMBLED ON THE RECEPTACLE.
3. NOTED COMPONENT MATERIALS:
   -RECEPTACLE BODY: BRASS C38000
   -PROTECTIVE CAP: VINYL
4. COMPONENTS SHALL BE RoHS COMPLIANT & MEET AMPHENOL REGULATED MATERIAL SPECIFICATIONS 949-1604.

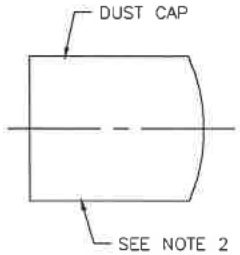


PANEL MOUNTING CUTOUT





SECTION A-A



REV.	DESCRIPTION	APVD	DATE	ECO No
B	ADD PANEL MOUNTING CUTOUT AND DIMENSIONS UPDATED NOTES AND ADD RoHS	SG	06/30/11	E11-0249
A	FIRST ISSUE	GJS	09/09/87	101-87

UNLESS OTHERWISE SPECIFIED TOLERANCE TO BE
LINEAR: X.X ± 0.1, X.XX ± 0.05, X.XXX ± 0.010
ANGULAR: X° ± 1°, X'X' ± 30', X"X"X" ± 15"
BREAK ALL SHARP EDGES AND REMOVE ALL BURRS
DO NOT SCALE
DIMENSIONS IN MILLIMETERS
THIRD ANGLE PROJECTION

MATL:	SEE NOTE 3
FINISH:	---
REF.:	
DRN:	R.FARHADIEH DATE:08-28-87
CKD:	P.CHANG DATE:08/31/87
APVD:	G.SELLERS DATE:09/03/11
SCALE:	NST
PART No:	905-117-5000
SIZE:	A3

TITLE: RECEPTACLE FLANGE MOUNT

Amphenol
FIBER OPTIC PRODUCTS
2000 Bradner Court, Suite 300, Lisle, IL 60532, USA

DRAWING NO. 905-117-5000
SHEET 1 OF 1

TCS3400

Color Light-to-Digital Converter

General Description

The TCS3400 device provides color and IR (red, green, blue, clear and IR) light sensing. The color sensing provides for improved accuracy lux and color temperature measurements typically used to adjust the backlight intensity and correct the display color gamut. Additionally it can be used for light source type detection as it reports the IR content of the light.

Ordering Information and Content Guide appear at end of datasheet.

Key Benefits & Features

The benefits and features of TCS3400, Color Light-to-Digital Converter are listed below:

Figure 1:
Added Value of Using TCS3400

Benefits	Features
• Single Device Integrated Optical Solution	• RGBC and ALS Support • Power Management Features
• Color Temperature and Ambient Light Sensing	• Programmable Gain & Integration Time • 1000000:1 Dynamic Range
• Equal Response to 360 degree Incident Light	• Circular Segmented RGBC Photodiode
• Ideal for Operation Behind Dark Glass	• Very High Sensitivity
• Light Source Detection	• RGBC + IR sensor

Applications

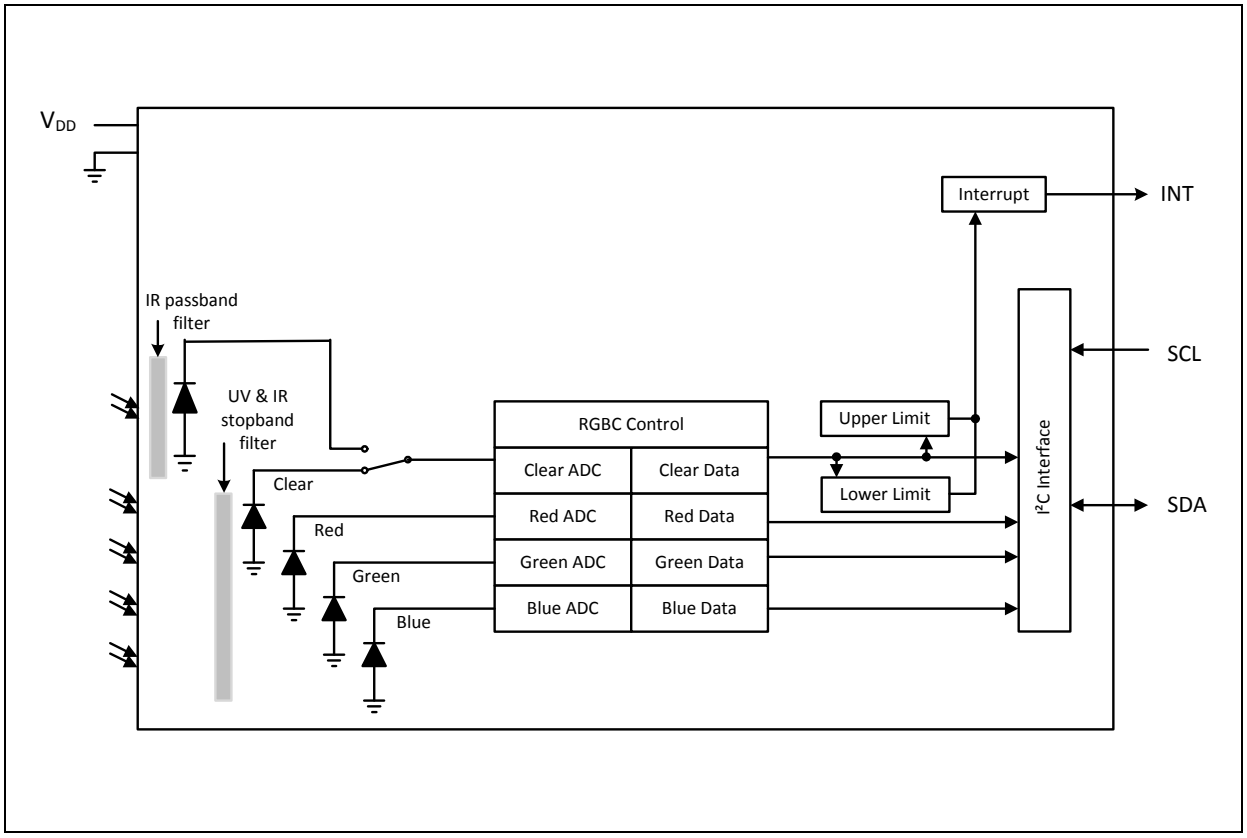
The TCS3400 applications include:

- Ambient light sensing
- Color temperature sensing
- Industrial process control
- Medical diagnostics

Block Diagram

The functional blocks of this device are shown below:

Figure 2:
TCS3400 Block Diagram



Pin Assignment

The TCS3400 pin assignments are described below.

Figure 3:
Pin Diagram

Pin Diagram (Top View):
Package FN Dual Flat No-Lead.
Package Drawing is not to scale.

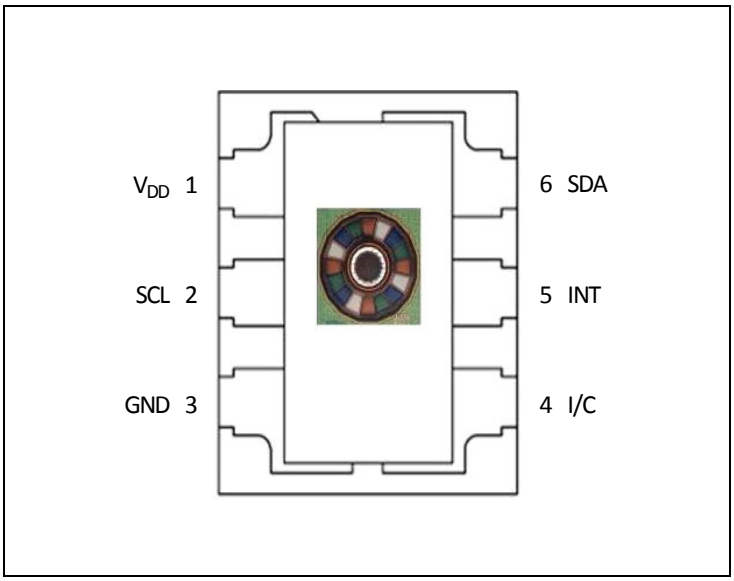


Figure 4:
Pin Description

Pin Number	Pin Name	Description
1	V_{DD}	Supply voltage
2	SCL	I^2C serial clock input terminal
3	GND	Power supply ground. All voltages are referenced to GND.
4	I/C	Internal connection, connect to ground or leave floating.
5	INT	Interrupt — open drain output (active low)
6	SDA	I^2C serial data I/O terminal – open drain

Absolute Maximum Ratings

Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only. Functional operation of the device at these or any other conditions beyond those indicated under Recommended Operating Conditions is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Figure 5:
Absolute Maximum Ratings

Parameter	Min	Max	Units	Comments
Supply voltage, V_{DD}		3.8	V	All voltages are with respect to GND
Input terminal voltage	-0.5	3.8	V	
Output terminal voltage	-0.5	3.8	V	
Output terminal current (SDA, INT)	-1	20	mA	
Storage temperature range, T_{STRG}	-40	85	°C	
Input current (latch up immunity) JEDEC JESD78D Nov 2011	CLASS 1			
Electrostatic discharge HBM S-001-2014	±2000		V	
Electrostatic discharge CDM JEDEC JESD22-C101F Oct 2013	±500		V	

Electrical Characteristics

All limits are guaranteed. The parameters with min and max values are guaranteed with production tests or SQC (Statistical Quality Control) methods.

Figure 6:
Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Units
V_{DD}	Supply voltage	2.7	3	3.6	V
T_A	Operating free-air temperature (1)	-40		70	°C

Note(s):

1. While the device is operational across the temperature range, functionality will vary with temperature. Specifications are stated at 25°C unless otherwise noted.

Figure 7:
Operating Characteristics, V_{DD}=3V, T_A=25°C (unless otherwise noted)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
I_{DD}	Supply current	Active		235	330	µA
		Wait state		60		
		Sleep state - no I^2C activity		1.0	10	
V_{OL}	INT, SDA output low voltage	3 mA sink current	0		0.4	V
		6 mA sink current	0		0.6	
I_{LEAK}	Leakage current, SDA, SCL, INT pins		-5		5	µA
V_{IH}	SCL, SDA input high voltage	TCS34001, TCS34005	0.7 V_{DD}			V
		TCS34003, TCS34007	1.26			
V_{IL}	SCL, SDA input low voltage	TCS34001, TCS34005			0.3 V_{DD}	V
		TCS34003, TCS34007			0.54	

Figure 8:
Optical Characteristics (Clear Channel), V_{DD} = 3V, T_A = 25°C, AGAIN = 16x, ATIME = 0xF6 (27.8ms)

Parameter	Test Conditions	Min	Typ	Max	Unit
R_e Irradiance Responsivity (Clear Channel)	White LED, CCT = 2700K(1)	11.2	14.0	16.8	counts/ ($\mu W/cm^2$)
	Blue LED, λ_D = 465 nm(2)	9.5	11.8	14.2	
	Green LED, λ_D = 525 nm(3)	11.6	14.5	17.4	
	Red LED, λ_D = 615 nm(4)	13.6	17.0	20.4	

Note(s):

1. The white LED irradiance is supplied by a warm white light-emitting diode with a nominal color temperature of 2700K.
2. The 465 nm input irradiance is supplied by an InGaN light-emitting diode with the following typical characteristics: dominant wavelength λ_D = 465 nm, spectral halfwidth $\Delta\lambda$½ = 22 nm.
3. The 525 nm input irradiance is supplied by an InGaN light-emitting diode with the following typical characteristics: dominant wavelength λ_D = 525 nm, spectral halfwidth $\Delta\lambda$½ = 35 nm.
4. The 615 nm input irradiance is supplied by an AlInGaP light-emitting diode with the following typical characteristics: dominant wavelength λ_D = 615 nm, spectral halfwidth $\Delta\lambda$½ = 15 nm.

Figure 9:
Optical Characteristics (IR Channel), VDD = 3V, T_A = 25°C, AGAIN = 16x, ATIME = 0xF6 (27.8ms)

Parameter	Test Condition	Min	Typ	Max	Unit
R_e Irradiance Responsivity (IR Channel)	λ_P = 850 nm(1)	10.0	13.3	16.6	counts/ ($\mu W/cm^2$)

Note(s):

1. The 850 nm input irradiance is supplied by an AlGaAs light-emitting diode with the following characteristics: peak wavelength λ_P = 850 nm, spectral halfwidth $\Delta\lambda$1/2 = 42 nm.

Figure 10:
Optical Characteristics, V_{DD}=3V, T_A=25°C

Parameter	Test Conditions	Red / Clear Channel		Green / Clear Channel		Blue / Clear Channel		IR / Clear Channel	
		Min	Max	Min	Max	Min	Max	Min	Max
Color ADC count value ratio: Color / Clear	λ_D = 465 nm (1)	0%	13%	10%	38%	70%	91%		
	λ_D = 525 nm (2)	3%	22%	59%	86%	10%	40%		
	λ_D = 615 nm (3)	80%	110%	0%	15%	3%	26%	0%	5%
	λ_P = 850 nm (4)							667%	

Note(s):

1. The 465 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 465 nm, spectral halfwidth $\Delta\lambda$½ = 22 nm.
2. The 525 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 525 nm, spectral halfwidth $\Delta\lambda$½ = 35 nm.
3. The 615 nm input irradiance is supplied by a AlInGaP light-emitting diode with the following characteristics: dominant wavelength λ_D = 615 nm, spectral halfwidth $\Delta\lambda$½ = 15 nm.
4. The 850 nm input irradiance is supplied by an AlGaAs light-emitting diode with the following characteristics: peak wavelength λ_P = 850 nm, spectral halfwidth $\Delta\lambda$1/2 = 42 nm.

Figure 11:
RGBC Characteristics, V_{DD} = 3V, T_A = 25°C, AGAIN = 16x, AEN = 1 (unless otherwise noted)

Parameter	Conditions	Min	Typ	Max	Units
Dark ADC count value (Clear and RGB Channels)	E_e = 0, AGAIN = 64x, ATIME = 0xB8 (200ms)	0	1	4	counts
		0		2	counts (1)
Dark ADC count value (IR Channel)		0	1	6	counts
		0		4	counts (1)
Integration time step size		2.65	2.78	2.93	ms
Number of integration steps		1		256	steps
ADC count value	ATIME = 0xFF (2.78ms) to 0xC1 (175ms) (1 to 63 steps)	0		1024	counts/ step
	ATIME = 0xC0 (178ms) to 0x00 (712ms) (64 to 256 steps)	0		65535	counts
Gain scaling, relative to 16× gain setting	1x: AGAIN = 00	0.936	0.985	1.065	×
	4x: AGAIN = 01	3.66	3.85	4.16	
	16x: AGAIN = 10		16.0		
	64x: AGAIN = 11	59.6	62.7	67.8	

Note(s):

1. Based on typical 3-sigma distribution. Not 100% tested.

Figure 12:
Wait Characteristics, V_{DD} = 3V, T_A = 25°C, WEN = 1 (unless otherwise noted)

Parameter	Conditions	Min	Typ	Max	Units
Wait step size	WTIME = 0xFF		2.78		ms

Timing Characteristics

The timing characteristics of TCS3400 are given below.

Figure 13:
AC Electrical Characteristics, V_{DD} = 3V, T_A = 25°C (unless otherwise noted)

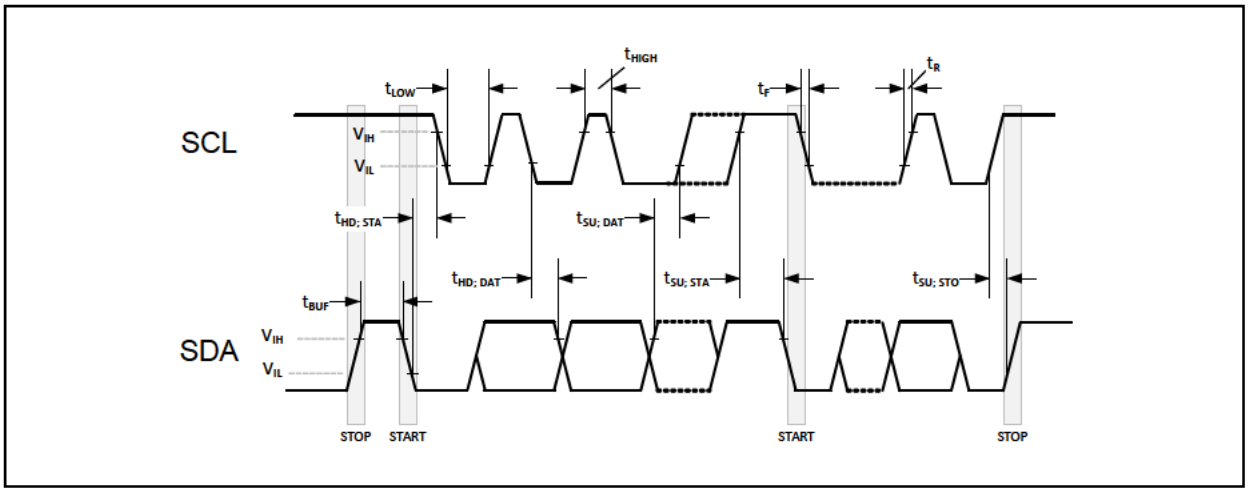
Parameter(1)	Conditions	Min	Max	Unit
f_{SCL}	Clock frequency (I^2C only)	0	400	kHz
t_{BUF}	Bus free time between start and stop condition	1.3		µs
$t_{HD;STA}$	Hold time after (repeated) start condition. After this period, the first clock is generated.	0.6		µs
$t_{SU;STA}$	Repeated start condition setup time	0.6		µs
$t_{SU;STO}$	Stop condition setup time	0.6		µs
$t_{HD;DAT}$	Data hold time	60		ns
$t_{SU;DAT}$	Data setup time	100		ns
t_{LOW}	SCL clock low period	1.3		µs
t_{HIGH}	SCL clock high period	0.6		µs
t_F	Clock/data fall time		300	ns
t_R	Clock/data rise time		300	ns
C_i	Input pin capacitance		10	pF

Note(s):

1. Specified by design and characterization; not production tested.

Timing Diagram

Figure 14:
Parameter Measurement Information



Typical Operating Characteristics

Figure 15:
Spectral Responsivity

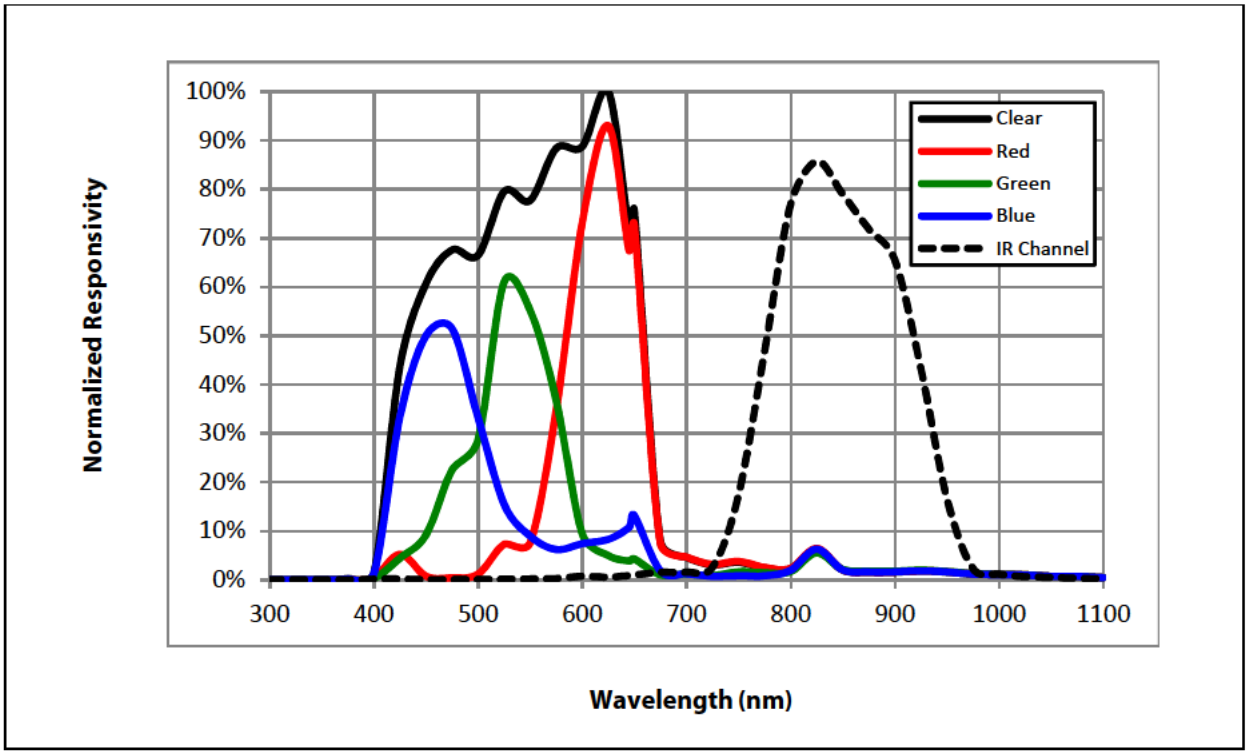


Figure 16:
Normalized Responsivity vs. Angular Displacement

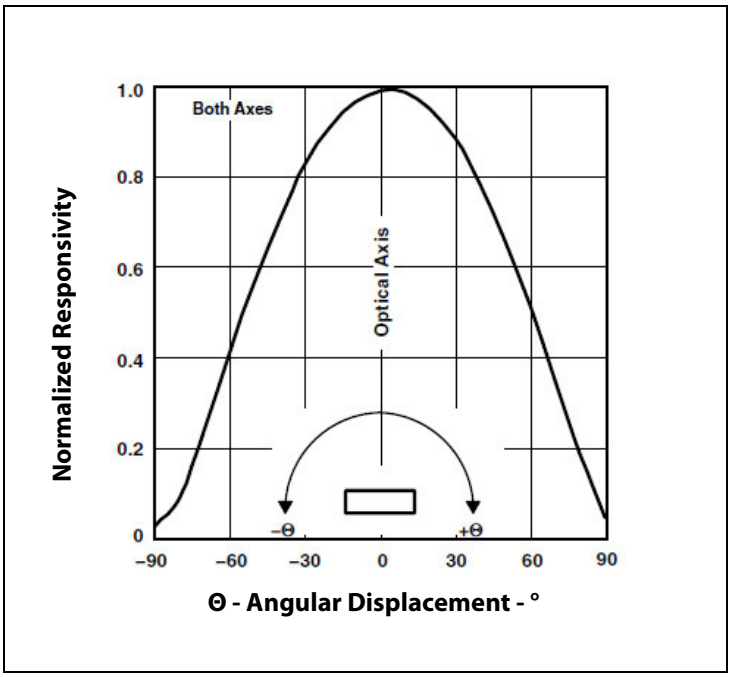


Figure 17:
Responsivity Temperature Coefficient

Wavelength	Temperature Coefficient
400 – 670nm	250 ppm/°C
850nm	2500 ppm/°C
950nm	5500 ppm/°C

Functional Description

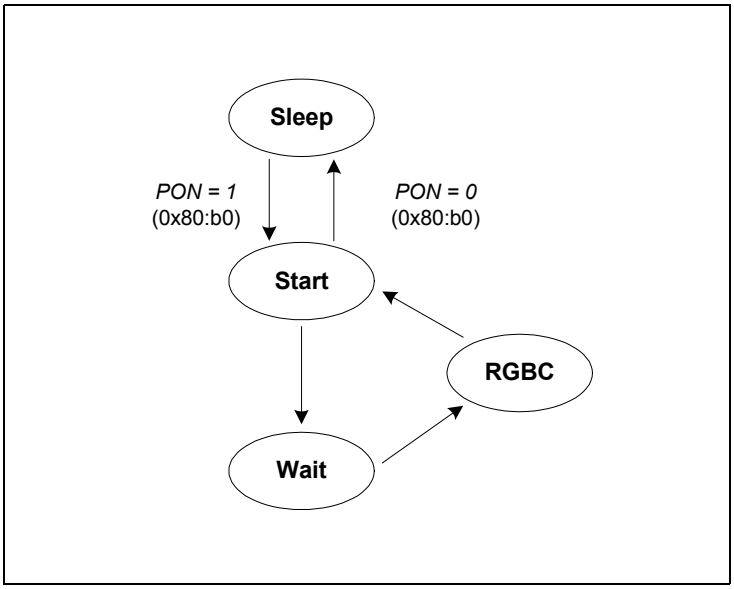
The TCS3400 device provides ambient light sensing and color temperature sensing. The internal state machine manages the operation of the device. It controls the ALS functionality and power down modes. Average power consumption is managed via control of variable endurance low power wait cycles.

The interrupt feature improves system efficiency by eliminating the need to poll the sensor. Two interrupt sources (ALS, ALS saturation) can activate the open drain output pin. Each interrupt source is enabled independently. ALS interrupts appear when upper or lower thresholds are exceeded for a consecutive number of sample readings.

The advanced digital color light sensor portion of the TCS3400 contains a segmented circular photodiode array used for color measurements. This architecture provides stable color sensing independent of the incident angle of light. Four integrating analog-to-digital converters (ADCs) integrate light energy from photodiodes simultaneously.

Communication with the device is accomplished through a fast (up to 400 kHz) two wire I^2C serial bus for easy connection to a microcontroller or embedded controller. The device typically draws only 235µA in color operation and 1uA during power down.

Figure 18:
Simplified ALS State Machine



Register Description

The device is controlled and monitored by registers accessed through the I^2C serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in the figure below.

Figure 19:
Register Map

Address	Register Name	R/W	Register Function	Reset Value
0x80	ENABLE	R/W	Enables states and interrupts	0x00
0x81	ATIME	R/W	RGBC integration time	0xFF
0x83	WTIME	R/W	Wait time	0xFF
0x84	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x85	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x86	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x87	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x8C	PERS	R/W	Interrupt persistence filter	0x00
0x8D	CONFIG	R/W	Configuration	0x40
0x8F	CONTROL	R/W	Gain control register	0x00
0x90	AUX	R/W	Auxiliary control register	0x00
0x91	REVID	R	Revision ID	Rev
0x92	ID	R	Device ID	ID
0x93	STATUS	R	Device status	0x00
0x94	CDATAL	R	Clear / IR channel low data register	0x00
0x95	CDATAH	R	Clear / IR channel high data register	0x00
0x96	RDATAL	R	Red ADC low data register	0x00
0x97	RDATAH	R	Red ADC high data register	0x00
0x98	GDATAL	R	Green ADC low data register	0x00
0x99	GDATAH	R	Green ADC high data register	0x00

Address	Register Name	R/W	Register Function	Reset Value
0x9A	BDATAL	R	Blue ADC low data register	0x00
0x9B	BDATAH	R	Blue ADC high data register	0x00
0xC0	IR	R/W	Access IR Channel	0x00
0xE4	IFORCE	W	Force Interrupt	0x00
0xE6	CICLEAR	W	Clear channel interrupt clear	0x00
0xE7	AICLEAR	W	Clear all interrupts	0x00

Enable Register (ENABLE 0 x 80)

The Enable Register is used primarily to power the device ON/OFF, and enable functions and interrupts.

Figure 20:
Enable Register

7	6	5	4	3	2	1	0
Reserved	SAI	Reserved	AIEN	WEN	Reserved	AEN	PON

Field	Bits	Description
Reserved	7	**Reserved.** Write as 0.
SAI	6	**Sleep After Interrupt.** When asserted, the device will power down at the end of a RGBC cycle if an interrupt is generated.
Reserved	5	**Reserved.** Write as 0.
AIEN	4	**ALS Interrupt Enable.** When asserted permits ALS interrupts to be generated, subject to the persist filter.
WEN	3	**Wait Enable.** This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	**Reserved.** Write as 0.
AEN	1	**ADC Enable.** This bit activates the four-channel (RGBC) ADC. Writing a 1 enables the ADC. Writing a 0 disables the ADC.
PON	0	**Power ON.** This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator and puts the part into a low power sleep mode. During reads and writes over the I^2C interface, this bit is temporarily overridden and the oscillator is enabled, independent of the state of PON.

RGBC Integration Time Register (ATIME 0x81)

The ATIME register controls the internal integration time of the RGBC channel ADCs. Upon power up, the RGBC time register is set to 0xFF.

The maximum (or saturation) count value can be calculated based upon the integration time cycles as follows:

*min [CYCLES * 1024, 65535]*

Figure 21:
RGBC Integration Time Register

Field	Bits	Description			
		Value	Cycles	Time	Max Count
ATIME	7:0	0xFF	1	2.78 ms	1024
		0xF6	10	27.8 ms	10240
		0xDB	37	103 ms	37888
		0xC0	64	178 ms	65535
		0x00	256	712 ms	65535

Wait Time Register (WTIME 0x83)

The WTIME controls the amount of time in a low power mode. It is set 2.78 ms increments unless the WLONG bit is asserted in which case the wait times are 12× longer. WTIME is programmed as a 2's complement number. Upon power up, the wait time register is set to 0xFF.

Figure 22:
Wait Time Register

Field	Bits	Description			
		Register Value	Wait Time	Time (WLONG=0)	Time (WLONG=1)
WTIME	7:0	0xFF	1	2.78 ms	0.03 s
		0xAB	85	236 ms	2.84 s
		0x00	256	712 ms	8.54 s

Note(s):

1. The wait time register should be configured before AEN is asserted.

Clear Channel Interrupt Threshold Register (0x84 - 0x87)

The Clear Channel Interrupt Threshold Registers provide 16 bit values to be used as the high and low thresholds for comparison to the 16 bit CDATA values. If AIEN (0x80:b4) is enabled and CDATA is not between AILT and AIHT for the number of consecutive samples specified in APERS (0x8C:b[3:0]) an interrupt is asserted on the interrupt pin.

Figure 23:
Clear Channel Interrupt Threshold Registers

Register	Address	Bits	Description
AILTL	0x84	7:0	Clear Channel low threshold lower byte
AILTH	0x85	7:0	Clear Channel low threshold upper byte
AIHTL	0x86	7:0	Clear Channel high threshold lower byte
AIHTH	0x87	7:0	Clear Channel high threshold upper byte

Interrupt Register (0x8C)

The Interrupt Register controls the interrupt capabilities of the device.

Figure 24:
Interrupt Register

7	6	5	4	3	2	1	0
Reserved				APERS			

Field	Bits	Description	
Reserved	7:4	**Reserved.** Write as 0.	
APERS	3:0	**Clear Interrupt Persistence.** Controls rate of Clear channel interrupt to the host processor.	
		Field Value	**Persistence**
		0000	Every RGBC cycle generates an interrupt
		0001	Any value outside of threshold range
		0010	2 consecutive values out of range
		0011	3 consecutive values out of range
		0100	5 consecutive values out of range
		0101	10 consecutive values out of range
		0110	15 consecutive values out of range
		0111	20 consecutive values out of range
		1000	25 consecutive values out of range
		1001	30 consecutive values out of range
		1010	35 consecutive values out of range
		1011	40 consecutive values out of range
		1100	45 consecutive values out of range
		1101	50 consecutive values out of range
		1110	55 consecutive values out of range
		1111	60 consecutive values out of range

Configuration Register (CONFIG 0x8D)

The CONFIG register sets the wait long time. The registers is set 0x40 at power up.

Figure 25:
Configuration Register

7	6	5	4	3	2	1	0
Reserved	Reserved1	Reserved				WLONG	Reserved

Field	Bits	Description
Reserved	7	**Reserved.** Write as 0.
Reserved (1)	6	**Reserved.** Write as 1.
Reserved	5:2	**Reserved.** Write all as 0.
WLONG	1	**Wait Long.** When asserted, the wait cycles are increased by a factor 12× from that programmed in the WTIME register.
Reserved	0	**Reserved.** Write as 0.

Note(s):

1. Bit 6 is reserved and has to be programmed = 1.

Control Register (CONTROL 0x8F)

Figure 26:
Control Register

7	6	5	4	3	2	1	0
Reserved						AGAIN	

Field	Bits	Description	
Reserved	7:2	**Reserved.** Write all as 0.	
AGAIN	1:0	**RGBC Gain Control.**	
		FIELD VALUE	**RGBC GAIN VALUE**
		00	1X Gain
		01	4X Gain
		10	16X Gain
		11	64X Gain

Auxiliary Register (AUX 0x90)

The AUX register enables the ALS saturation detection interrupt. If ASIEN = 1 and an interrupt occurs it is cleared by accessing the Clear Interrupt registers at 0XE6 or 0XE7.

Figure 27:
Auxiliary Register

7	6	5	4	3	2	1	0
Reserved		ASIEN	Reserved				

Field	Bits	Description
Reserved	7:6	**Reserved.** Write all as 0.
ASIEN	5	0 disables, 1 enables ALS Saturation Interrupt
Reserved	4:0	**Reserved.**

Revision ID Register (REVID 0x91)

This read-only register identifies the die revision level.

Figure 28:
Revision ID Register

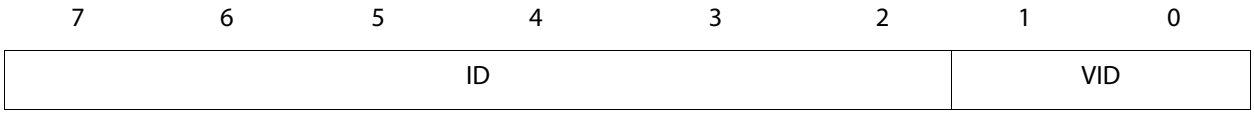
7	6	5	4	3	2	1	0
Reserved				RevID			

Field	Bits	Description
Reserved	7:4	**Reserved.**
RevID	3:0	Wafer die revision level

ID Register (ID 0x92)

The read-only ID register provides the device identification.

Figure 29:
ID Register

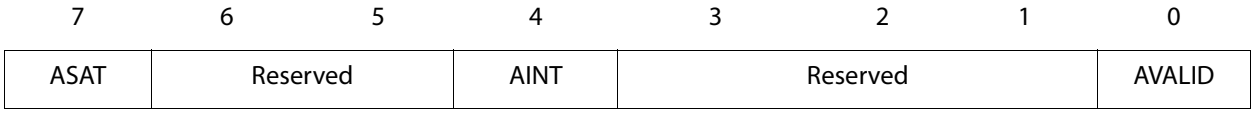


Field	Bits	Description
ID	7:2	Device Identification = 100100
VID	1:0	00b for TCS34001 & TCS34005 11b for TCS34003 & TCS34007

Status Register (STATUS 0x93)

The read-only Status Register provides the internal status of the device.

Figure 30:
Status Register



Field	Bits	Description
ASAT	7	**ALS Saturation.** When asserted, the analog sensor was at the upper end of its dynamic range. The bit can be de-asserted by sending a clear channel interrupt command (0xE6 CICLEAR) or by disabling the ALS ADC (AEN=0). ATIME and AGAIN are controls that can be adjusted to set when saturation happens. This bit triggers an interrupt if ASIEN in AUX is set.
Reserved	6:5	**Reserved.**
AINT	4	**ALS Interrupt.** If AEN is set, indicates that an ALS event that met the programmed ALS thresholds (AILT or AIHT) and persistence (APERS) occurred.
Reserved	3:1	**Reserved.**
AVALID	0	**RGBC Valid.** Indicates that the RGBC cycle has completed since AEN was asserted.

RGBC Data Registers (0x94 - 0x9B)

Clear, red, green, and blue data is stored as 16-bit values. The read sequence must read byte pairs (low followed by high) starting on an even address boundary (0x94, 0x96, 0x98, or 0x9A) inside the RGBC Data Register block. When the lower byte register is read, the upper eight bits are stored into a shadow register, which is read by a subsequent read to the upper byte. The upper register will read the correct value even if additional ADC integration cycles end between the reading of the lower and upper registers.

Figure 31:
RGBC Data Registers

Register	Address	Bits	Description
CDATAL	0x94	7:0	Clear / IR data low byte
CDATAH	0x95	7:0	Clear / IR data high byte
RDATAL	0x96	7:0	Red data low byte
RDATAH	0x97	7:0	Red data high byte
GDATAL	0x98	7:0	Green data low byte
GDATAH	0x99	7:0	Green data high byte
BDATAL	0x9A	7:0	Blue data low byte
BDATAH	0x9B	7:0	Blue data high byte

IR Register (0xC0)

Access to IR channel; allows mapping of IR channel on clear channel.

Figure 32:
IR Register

7	6	5	4	3	2	1	0
IR	Reserved						

Field	Bits	Description
IR	7	**IR Sensor access.** If this bit is set the clear channel reports the measurement from the IR sensor (center diode).
Reserved	6:0	**Reserved.** Always write as 0.

Clear Interrupt Registers (0xE3, 0xE7)

Any dummy data byte (0x00 recommended) written to the specified register will clear the indicated interrupt.

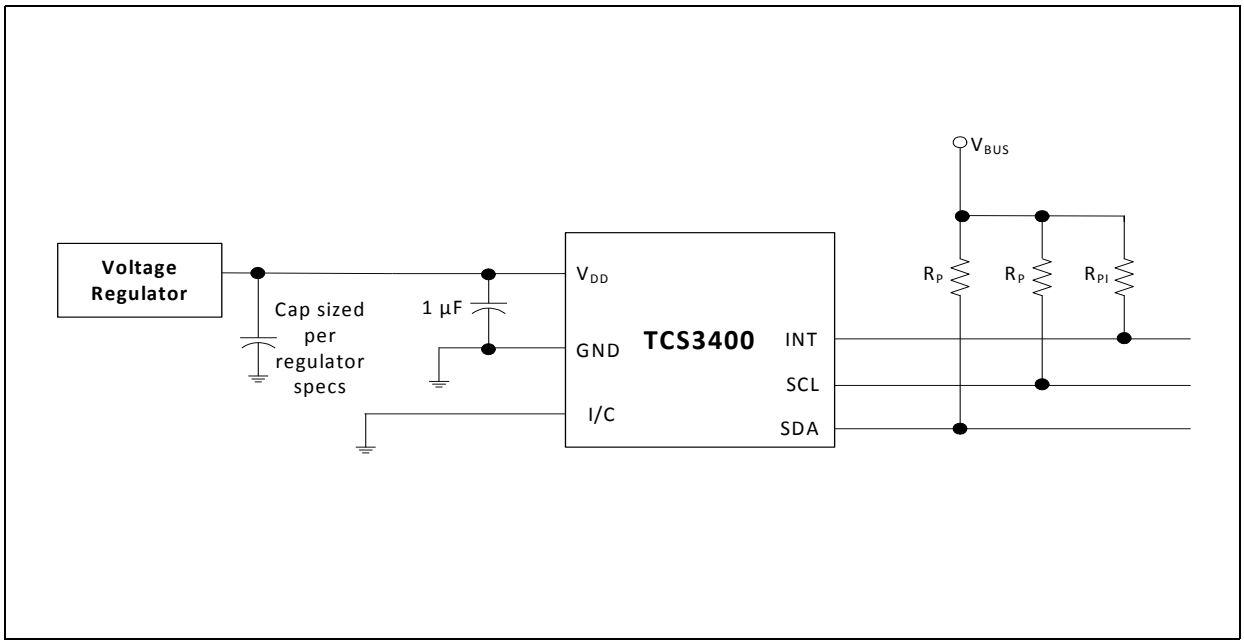
Figure 33:
Clear Interrupt Registers

Register	Address	Bits	Description
IFORCE	0xE4	7:0	Forces an interrupt (any value)
CICLEAR	0xE6	7:0	Clear channel interrupt clear (any value)
AICLEAR	0xE7	7:0	Clears all interrupts (any value)

Power Supply Considerations

Place a 1-µF low-ESR decoupling capacitor as close as possible to the V_{DD} pin.

Figure 34:
Typical Application Hardware Circuit



V_{BUS} in the above figures refers to the I^2C bus voltage which is either V_{DD} or 1.8V. Be sure to apply the specified I^2C bus voltage shown in the Ordering & Contact Information for the specific device being used.

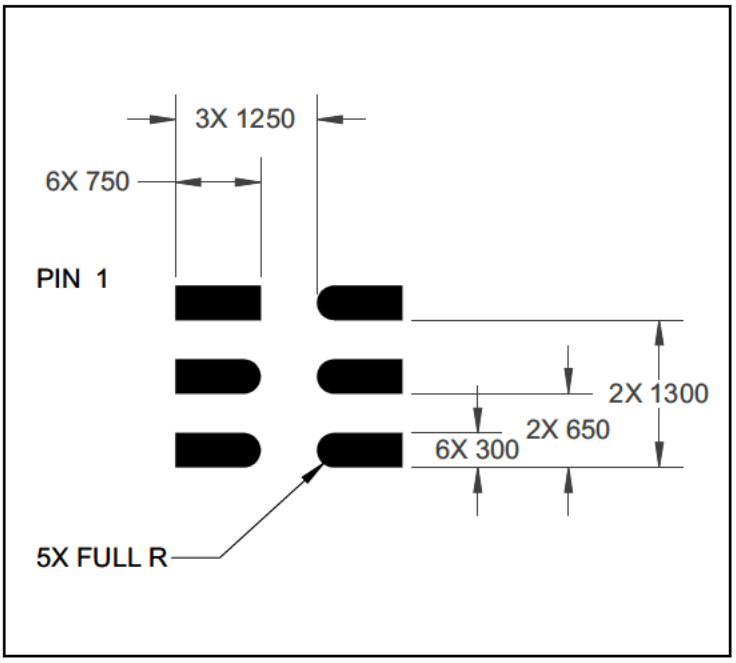
The I^2C signals and the Interrupt are open-drain outputs and require pull–up resistors. The pull-up resistor (RP) value is a function of the I^2C bus speed, the I^2C bus voltage, and the capacitive load. The **ams** EVM running at 400 kbit/s, uses 1.5-kΩ resistors. A 10-kΩ pull-up resistor (RPI) can be used for the interrupt line.

PCB Pad Layout

Suggested PCB pad layout guidelines for the surface mount module are shown. Flash Gold is recommended as a surface finish for the landing pads.

Figure 35:
Suggested PCB Layout

PCB Layout: Suggested land pattern based on the IPC-7351B Generic Requirements for Surface Mount Design and Land Pattern Standard (2010) for the small outline no-lead (SON) package.

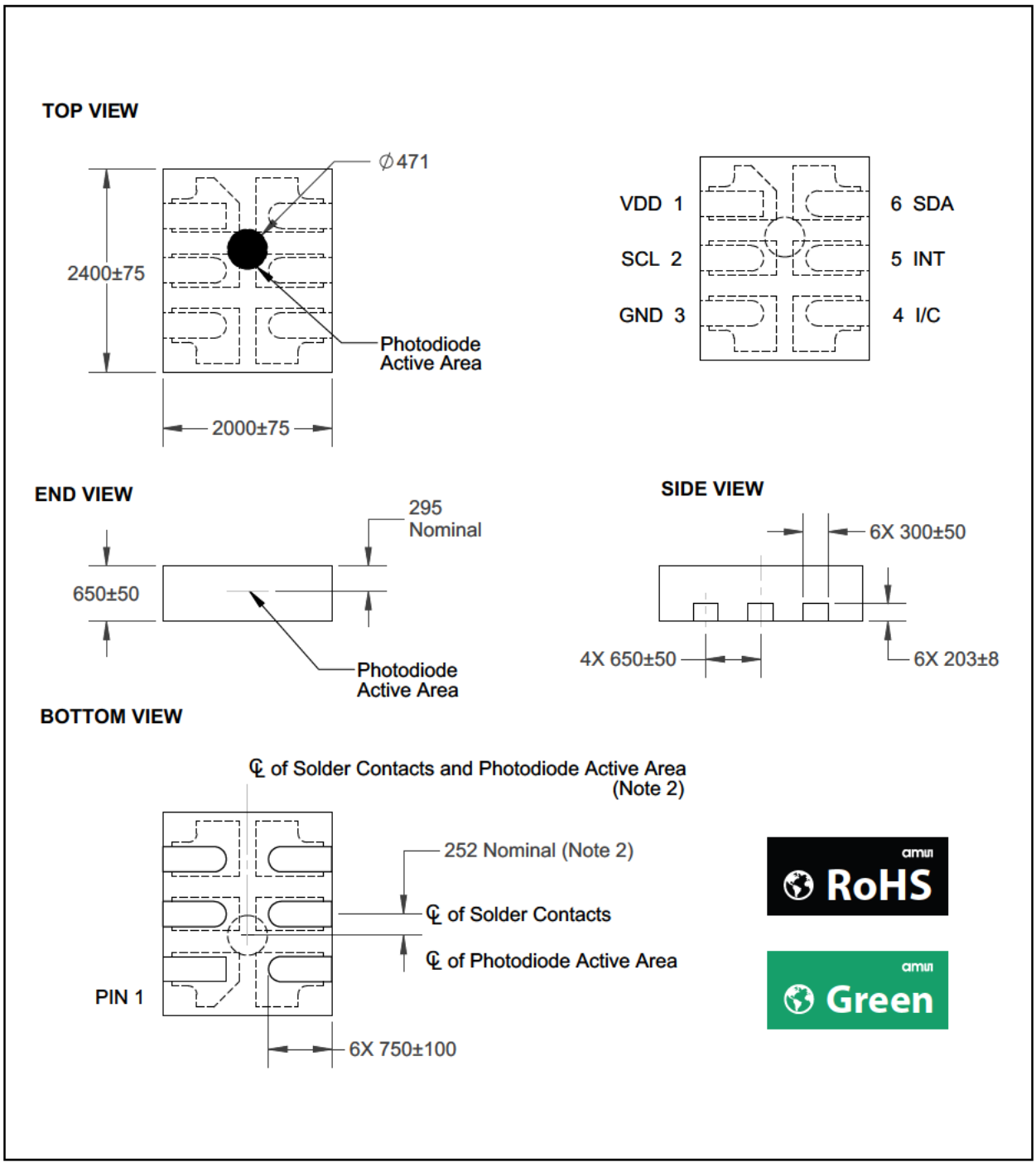


Note(s):

1. All linear dimensions are in millimeters.
2. This drawing is subject to change without notice.

Package Drawings & Markings

Figure 36:
IC Package Mechanical Drawing

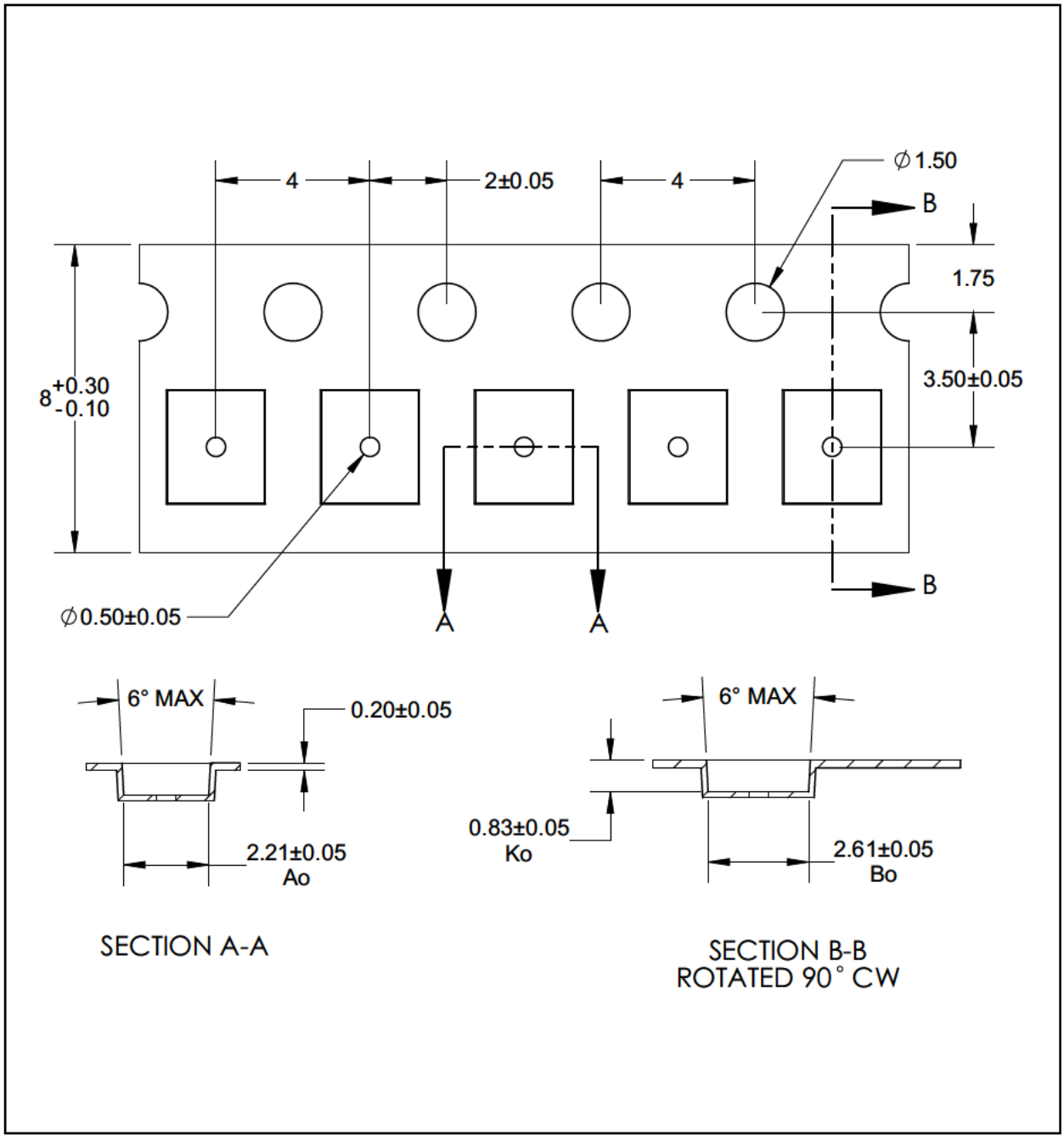


Note(s):

1. All linear dimensions are in micrometers. Dimension tolerance is ±20 µm unless otherwise noted.
2. The die is centered within the package within a tolerance of ±75 µm.
3. Package top surface is molded with an electrically non-conductive clear plastic compound having an index of refraction of 1.55.
4. Contact finish is Copper Alloy A194 with pre-plated NiPdAu lead finish.
5. This package contains no lead (Pb).
6. This drawing is subject to change without notice.

Package Mechanical Data

Figure 37:
Carrier Tape & Reel Information



Note(s):

1. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
2. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
3. Symbols on drawing Ao, Bo, and Ko are defined in ANSI EIA Standard 481–B 2001.
4. Each reel is 330 millimeters in diameter.
5. Packaging tape and reel conform to the requirements of EIA Standard 481–B.
6. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
7. This drawing is subject to change without notice.
8. The device pin 1 is located in the upper left corner inside the T&R pockets.

Soldering & Storage Information

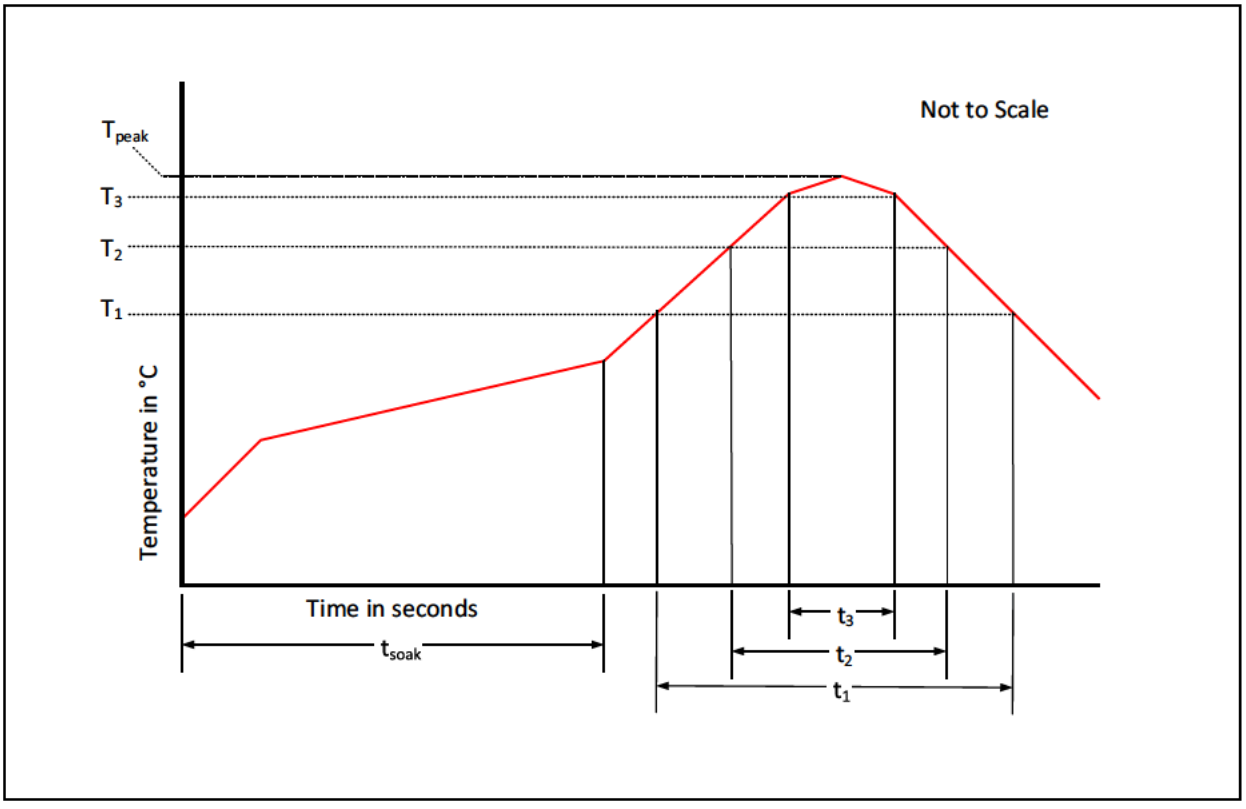
The FN package has been tested and has demonstrated an ability to be reflow soldered to a PCB substrate.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Figure 38:
Solder Reflow Profile

Parameter	Reference	Device
Average temperature gradient in preheating		2.5 °C/s
Soak time	t_{soak}	2 to 3 minutes
Time above 217 °C (T_1)	t_1	Max 60 s
Time above 230 °C (T_2)	t_2	Max 50 s
Time above T_{peak} - 10 °C (T_3)	t_3	Max 10 s
Peak temperature in reflow	T_{peak}	260 °C
Temperature gradient in cooling		Max -5 °C/s

Figure 39:
Solder Reflow Profile Graph



Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package. To ensure the package contains the smallest amount of absorbed moisture possible, each device is baked prior to being dry packed for shipping.

Devices are dry packed in a sealed aluminized envelope called a moisture-barrier bag with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

Shelf Life

The calculated shelf life of the device in an unopened moisture barrier bag is 12 months from the date code on the bag when stored under the following conditions:

- Shelf Life: 12 months
- Ambient Temperature: < 40°C
- Relative Humidity: < 90%

Rebaking of the devices will be required if the devices exceed the 12 month shelf life or the Humidity Indicator Card shows that the devices were exposed to conditions beyond the allowable moisture region.

Floor Life

The FN package has been assigned a moisture sensitivity level of MSL 3. As a result, the floor life of devices removed from the moisture barrier bag is 168 hours from the time the bag was opened, provided that the devices are stored under the following conditions:

- Floor Life: 168 hours
- Ambient Temperature: < 30°C
- Relative Humidity: < 60%

If the floor life or the temperature/humidity conditions have been exceeded, the devices must be rebaked prior to solder reflow or dry packing.

Rebaking Instructions

When the shelf life or floor life limits have been exceeded, rebake at 50°C for 12 hours.

Ordering & Contact Information

The device is packaged in a small OFN (Optical FN) package which is 2mm x 2.4mm.

Figure 40:
Ordering Information

Ordering Code	Address	Interface	Delivery Form	Delivery Quantity
TCS34001FN	0x39	I^2C $V_{BUS} = V_{DD}$ Interface	FN-6	12000 pcs/reel
TCS34001FNM	0x39	I^2C $V_{BUS} = V_{DD}$ Interface	FN-6	500 pcs/reel
TCS34003FN	0x39	I^2C bus = 1.8V Interface	FN-6	12000 pcs/reel
TCS34003FNM	0x39	I^2C bus = 1.8V Interface	FN-6	500 pcs/reel
TCS34005FN (1)	0x29	I^2C $V_{BUS} = V_{DD}$ Interface	FN-6	12000 pcs/reel
TCS34005FNM (1)	0x29	I^2C $V_{BUS} = V_{DD}$ Interface	FN-6	500 pcs/reel
TCS34007FN	0x29	I^2C bus = 1.8V Interface	FN-6	12000 pcs/reel
TCS34007FNM	0x29	I^2C bus = 1.8V Interface	FN-6	500 pcs/reel

Note(s):

1. Contact **ams** for availability.

Buy our products or get free samples online at:
www.ams.com/ICdirect

Technical Support is available at:
www.ams.com/Technical-Support

Provide feedback about this document at:
www.ams.com/Document-Feedback

For further information and requests, e-mail us at:
ams_sales@ams.com

For sales offices, distributors and representatives, please visit:
www.ams.com/contact

Headquarters
ams AG
Tobelbader Strasse 30
8141 Premstaetten
Austria, Europe

Tel: +43 (0) 3136 500 0

Website: www.ams.com

RoHS Compliant & ams Green Statement

RoHS: The term RoHS compliant means that ams AG products fully comply with current RoHS directives. Our semiconductor products do not contain any chemicals for all 6 substance categories, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, RoHS compliant products are suitable for use in specified lead-free processes.

ams Green (RoHS compliant and no Sb/Br): ams Green defines that in addition to RoHS compliance, our products are free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information: The information provided in this statement represents ams AG knowledge and belief as of the date that it is provided. ams AG bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. ams AG has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. ams AG and ams AG suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

Copyrights & Disclaimer

Devices sold by ams AG are covered by the warranty and patent indemnification provisions appearing in its General Terms of Trade. ams AG makes no warranty, express, statutory, implied, or by description regarding the information set forth herein. ams AG reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with ams AG for current information. This product is intended for use in commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by ams AG for each application. This product is provided by ams AG “AS IS” and any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

ams AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of ams AG rendering of technical or other services.

Document Status

Document Status	Product Status	Definition
Product Preview	Pre-Development	Information in this datasheet is based on product ideas in the planning phase of development. All specifications are design goals without any warranty and are subject to change without notice
Preliminary Datasheet	Pre-Production	Information in this datasheet is based on products in the design, validation or qualification phase of development. The performance and parameters shown in this document are preliminary without any warranty and are subject to change without notice
Datasheet	Production	Information in this datasheet is based on products in ramp-up to full production or full production which conform to specifications in accordance with the terms of ams AG standard warranty as given in the General Terms of Trade
Datasheet (discontinued)	Discontinued	Information in this datasheet is based on products which conform to specifications in accordance with the terms of ams AG standard warranty as given in the General Terms of Trade, but these products have been superseded and should not be used for new designs

Revision Information

Changes from 1-05 (2016-Aug-11) to current revision 1-06 (2017-Oct-10)	Page
Updated Figure 40	29

Note(s):

1. Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
2. Correction of typographical errors is not explicitly mentioned.

Content Guide

1 General Description
1 Key Benefits & Features
1 Applications
2 Block Diagram

3 Pin Assignment
4 Absolute Maximum Ratings
5 Electrical Characteristics

9 Timing Characteristics
9 Timing Diagram

10 Typical Operating Characteristics
12 Functional Description

13 Register Description
14 Enable Register (ENABLE 0 x 80)
15 RGBC Integration Time Register (ATIME 0x81)
15 Wait Time Register (WTIME 0x83)
16 Clear Channel Interrupt Threshold Register (0x84 - 0x87)
17 Interrupt Register (0x8C)
18 Configuration Register (CONFIG 0x8D)
18 Control Register (CONTROL 0x8F)
19 Auxiliary Register (AUX 0x90)
19 Revision ID Register (REVID 0x91)
20 ID Register (ID 0x92)
20 Status Register (STATUS 0x93)
21 RGBC Data Registers (0x94 - 0x9B)
22 IR Register (0xC0)
22 Clear Interrupt Registers (0xE3, 0xE7)

23 Power Supply Considerations
24 PCB Pad Layout
25 Package Drawings & Markings
26 Package Mechanical Data

27 Soldering & Storage Information
28 Moisture Sensitivity
28 Shelf Life
28 Floor Life
28 Rebaking Instructions

29 Ordering & Contact Information
30 RoHS Compliant & ams Green Statement
31 Copyrights & Disclaimer
32 Document Status
33 Revision Information

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original