

BACHELORTHESIS
Jonas Ernsting

Funktionsdemonstrator für die optische Messdatenübertragung in Fahrzeugbatterien

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

Jonas Ernsting

Funktionsdemonstrator für die optische Messdatenübertragung in Fahrzeugbatterien

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.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Paweł Buczek

Eingereicht am: 12. Oktober 2020

Jonas Ernsting

Thema der Arbeit

Funktionsdemonstrator für die optische Messdatenübertragung in Fahrzeugbatterien

Stichworte

Mikrocontroller, Sensor, Messung, messen, Batterie, Zelle, BMS, UART, Transceiver, IrDA, SIR, geometrische Optik, optisch, Lichtleitkörper, Lichtwellenleiter, Lichtleiter, Datenübertragung, Kommunikation, Demonstrator, Demonstrationsaufbau, C-Quelltext

Kurzzusammenfassung

Dieser Arbeit befasst sich mit der optischen Datenübertragung mittels infrarot-Transceiver-Modulen und eines Lichtleitkörpers. Hierfür wird ein Lichtleitkörper entworfen und ein Demonstrationsaufbau erstellt. Die Anordnung und Abmessungen sind dabei dabei eine realen Batterie orientiert. Die Übertragung orientiert sich an der IrDA SIR Spezifikation und erfolgt nach dem Master-Slave-Prinzip. Dabei steht vor allem die reine Funktionsweise der Übertragungsmöglichkeit im Vordergrund, nicht der übertragbare Inhalt.

Jonas Ernsting

Title of Thesis

Demonstration model for the optical transmission of measurement data in vehicle batteries

Keywords

microcontroller, sensor, measurement, measure, battery, cell, BMS, UART, transceiver, IrDA, SIR, geometrical optics, optical, light conductor, optical waveguide, light guide, data transfer, communication, demonstration model, C source code

Abstract

This thesis deals with optical data transmission by means of infrared transceiver modules and a light guide body. For this purpose, a light guide body is designed and a demonstration model is created. The arrangement and dimensions are based on a real battery. The transmission is based on the IrDA SIR specification and takes place according to the master-slave principle. The main focus here is on the pure functionality of the data transfer, not the transferred data.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ausgangssituation und Problemstellung	1
1.2	Übersicht und Vergleich der Kommunikationslösungen	3
1.2.1	Datenübertragung mittels elektrischer Leitungen	3
1.2.2	Datenübertragung mittels Hauptstromleitung	4
1.2.3	Datenübertragung mittels elektromagnetischer Strahlung	5
1.2.4	Datenübertragung mittels Lichtleitkörper	6
1.2.5	Vergleich der Optionen	7
1.3	Inhalt und Ziel dieser Arbeit	9
2	Grundlagen und Konzeption	10
2.1	Grundkonzept und Systemaufbau	10
2.2	Eingesetzte Komponenten	14
2.3	Kommunikationsprotokoll	17
2.4	Lichtleitkörper	24
2.4.1	Optische Eigenschaften	24
2.4.2	Maße des Lichtleitkörpers und Positionen der Transceiver-Module	27
2.4.3	Konzepte des Lichtleitkörpers	30
3	Entwicklung und Implementierung	36
3.1	Lichtleitkörper	36
3.1.1	Lichtleitkörpermaterial	36
3.1.2	Linsenberechnung	36
3.1.3	Alternative Linsenberechnung	41
3.1.4	Berechnung der Reflektorflächen	43
3.1.5	Simulation und Fertigung des Lichtleitkörpers	53
3.2	Hardware	57
3.2.1	Verfügbarer Strom für externe Komponenten	58

3.2.2	Messmodul	62
3.2.3	Modulüberwachung	74
3.2.4	Mechanische Konstruktion	76
3.3	Software	80
3.3.1	Limitierung der DAVE-APPs	80
3.3.2	Implementierte Befehle	81
3.3.3	Flussdiagramme	84
3.3.4	Kommunikationsabläufe	88
3.3.5	Bedienung	99
4	Test der Komponenten und Erprobung des Demonstrationsaufbaus	103
4.1	Optik	103
4.1.1	Funktionsfähigkeit ohne Lichtleitkörper	103
4.1.2	Funktionsfähigkeit mit Lichtleitkörper	104
4.1.3	Verschiebung des Lichtleitkörpers	104
4.2	Hardware	105
4.2.1	Transceiver-Modul	105
4.2.2	Fertigungstoleranzen	110
4.2.3	Robustheit des Demonstrationsaufbaus	111
4.2.4	Bedienbarkeit	111
4.2.5	Stromaufnahme des Überwachungs- und der Messmodule	112
4.2.6	Messgenauigkeit der Analog-zu-Digital-Wandler	113
4.3	Software	113
4.3.1	Kommunikation zwischen dem Überwachungs- und den Messmodulen	113
4.3.2	Kommunikation zwischen dem Überwachungsmodul und einem Com- puter	116
4.4	Ausstehende Tests	116
5	Bewertung, Zusammenfassung, Offene Punkte und Ausblick	117
5.1	Ergebnisse und Bewertung	117
5.2	Offene Punkte und Ansätze zur Weiterführung	118
5.2.1	Lichtleitkörper	118
5.2.2	Hardware	119
5.2.3	Software	119
5.3	Ausblick	119
	Abbildungsverzeichnis	121

Tabellenverzeichnis	124
Abkürzungen	127
Literaturverzeichnis	128
A Anhänge	133
A.1 Konfiguration	133
A.1.1 APP Abhängigkeit der DAVE-APPs des Überwachungsmoduls als Master	134
A.1.2 APP Abhängigkeit der DAVE-APPs des Messmoduls als Slave . . .	135
A.1.3 Hardwaresignalverbindungen der DAVE-APPs des Überwachungs- moduls als Master	136
A.1.4 Hardwaresignalverbindungen der DAVE-APPs des Messmoduls als Slave	137
A.1.5 Konfiguration der DAVE-APPs des Überwachungsmoduls als Master	138
A.1.6 Konfiguration der DAVE-APPs des Messmoduls als Slave	151
A.2 Quelltexte	157
A.2.1 main.c des Überwachungsmoduls als Master	157
A.2.2 main.c des Messmoduls als Slave	185
Selbstständigkeitserklärung	194

1 Einleitung

1.1 Ausgangssituation und Problemstellung

Der Anteil an zugelassenen Elektrofahrzeugen im deutschen Straßenverkehr steigt stetig an [8]. Elektrofahrzeuge, deren Energiespeicher eine Batterie aus Lithium-Ionen-Zellen ist, benötigen eine Möglichkeit, den Zustand jeder einzelnen Zelle für einen sicheren Betrieb zu überwachen. Hierfür dient ein Batteriemanagementsystem (BMS) oder Überwachungsmodul, welches die Zellspannung und gegebenenfalls weitere Daten wie zum Beispiel Zelltemperatur und -strom erfasst. Das Überwachungsmodul wird in dieser Arbeit auch teilweise mit Modulüberwachung bezeichnet.

Um den erforderlichen Strom für die gewünschte Abgabeleistung und somit das Gewicht durch Verringerung des Leiterquerschnitts zu senken, sind in einer Batterie zur Anhebung der Gesamtspannung einzelne oder Pakete aus parallel geschalteten Zellen in Reihe geschaltet. Dadurch entsteht bei der Erfassung der Zellspannungen jedoch das Problem, dass diese höhere Spannung auch vom BMS oder Überwachungsmodul verarbeitet werden müssen, was die Messgenauigkeit reduziert. Auch ist der Verdrahtungsaufwand hoch, besonders wenn noch weitere Sensoren, zum Beispiel für die Zelltemperatur verbaut werden sollen, da die entsprechenden Leitungen zu jeder Zelle gelegt werden müssen. Mit steigender Anzahl der Leitungen steigen das Gewicht, der Platzbedarf, der Fertigungsaufwand, die Störungsanfälligkeit und die Kosten. Dieser Systemaufbau ist in Abbildung 1.1 gezeigt.

Eine Alternative ist der Einsatz von Messmodulen mit einer Kommunikationsschnittstelle auf jeder einzelnen Zelle, deren Daten das BMS oder Überwachungsmodul ausliest. Die Messmodule werden durch die Zelle versorgt und die Kommunikation geschieht galvanisch getrennt. Dadurch ist die Messung dezentralisiert, was die Komplexität der Messelektronik reduziert, die Messgenauigkeit zu erhöht und keine zusätzliche Leitungen für weitere Sensoren erfordert. Hierfür gibt es verschiedene Konzepte, welche nachfolgend erläutert und verglichen werden.

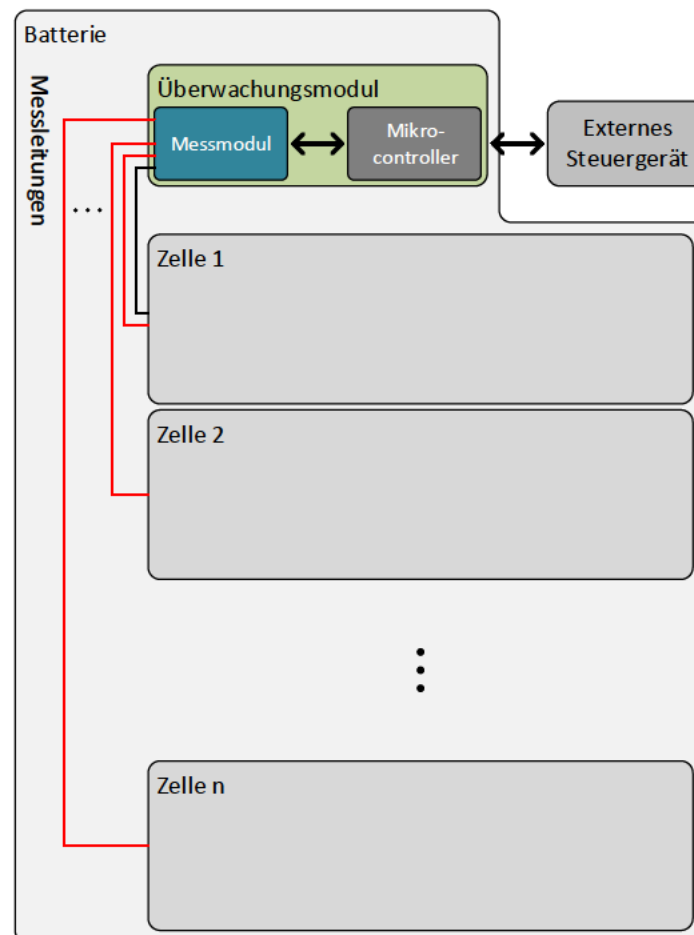


Abbildung 1.1: Systemaufbau bei zentraler Messung mittels elektrischer Leitungen im BMS bzw. Überwachungsmodul. Für die Spannungsmessung ist zu jeder Zelle eine Messleitung gelegt. Das BMS misst alle Spannungen relativ zur gemeinsamen Masse und berechnet daraus die Zellspannung. Jeder zusätzliche Sensor benötigt mindestens zwei weitere Leitungen für die Verbindung mit dem BMS.

1.2 Übersicht und Vergleich der Kommunikationslösungen

1.2.1 Datenübertragung mittels elektrischer Leitungen

In einer Daisy-Chain-Topologie sind elektrische Leitungen immer zwischen zwei nebeneinander liegenden Busteilnehmern gelegt, sodass an jeden Busteilnehmer höchstens zwei angeschlossen sind. Durch potentialtrennende Bausteine sind die Busteilnehmer mit diesen Datenleitungen verbunden.

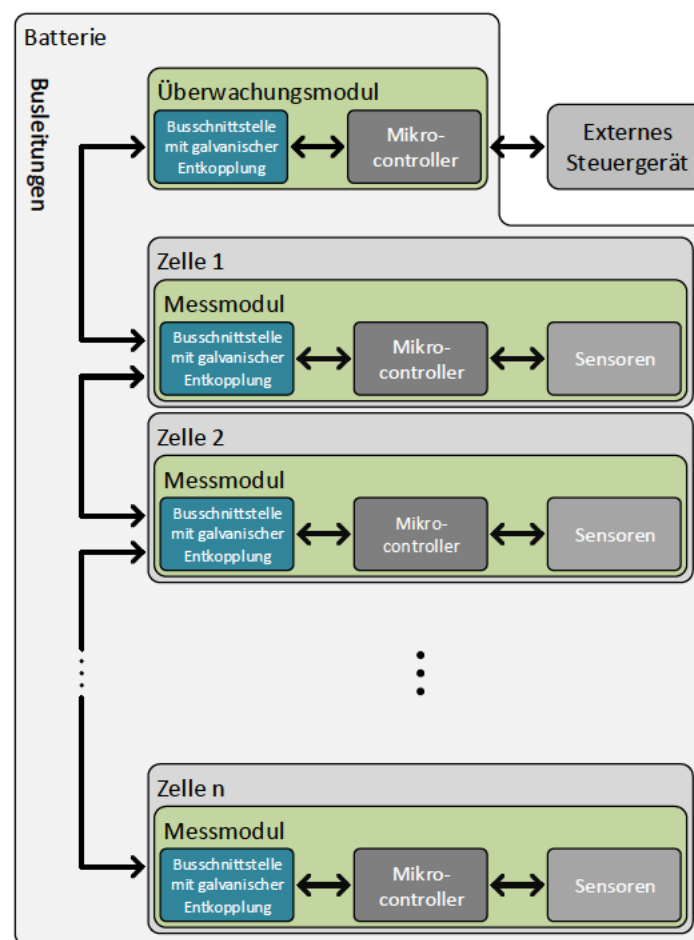


Abbildung 1.2: Systemaufbau mit Datenübertragung über elektrische Verbindung. Jeder Busteilnehmer ist galvanisch getrennt an die gemeinsame Busleitung angeschlossen, welche sich in Form einer Daisy-Chain-Topologie aus den Leitungsstücken zwischen den Busteilnehmern zusammensetzt.

1.2.2 Datenübertragung mittels Hauptstromleitung

Bei der sogenannten Powerline Communication (PLC) dient die Hauptstromleitung der Batterie als Übertragungsmedium, in welches das zu übertragende Signal kapazitiv oder induktiv von den Sendern ein- und an den Empfängern ausgekoppelt wird.

Weitere Informationen zu diesem Thema können in dem Whitepaper der HomePlug Powerline Alliance, Inc. [12] und auf dem Poster von Thomas Landinger [40] nachgelesen werden.

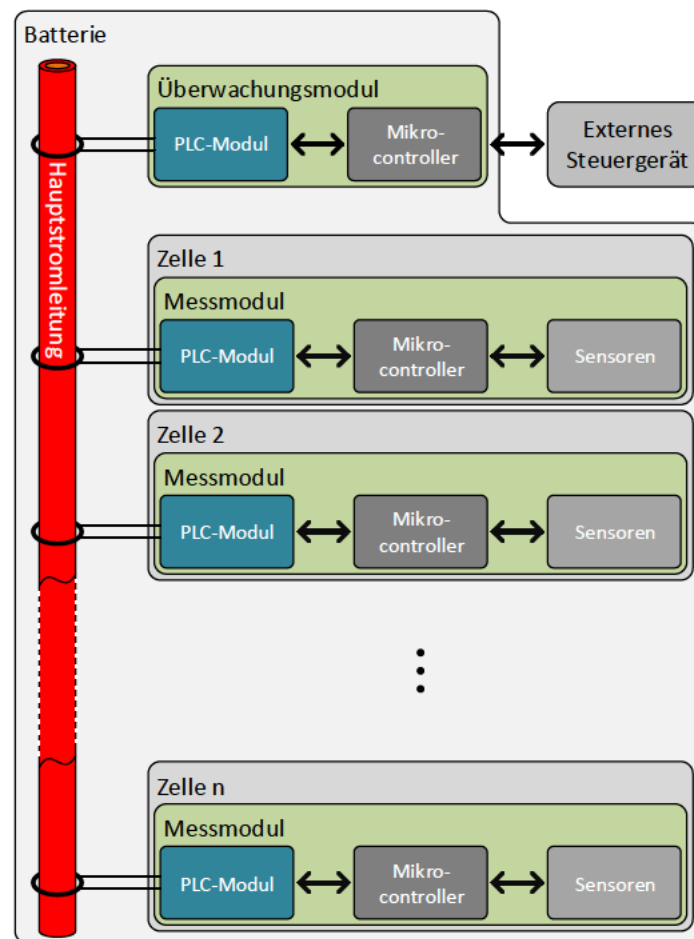


Abbildung 1.3: Systemaufbau mit Datenübertragung über Hauptstromleitung. Alle Bus-teilnehmer sind galvanisch getrennt induktiv oder kapazitiv an die Hauptstromleitung der Batterie gekoppelt und können das Übertragungssignal in diese ein- und auskoppeln.

1.2.3 Datenübertragung mittels elektromagnetischer Strahlung

Das BMS und die Messmodule besitzen je eine Antenne, über welche die Daten durch elektromagnetische Wellen übertragen werden.

Weitere Informationen zu diesem Thema können in den Abschlussarbeiten von Nico Sassano [36] [37] und Eike Mense [2] nachgelesen werden.

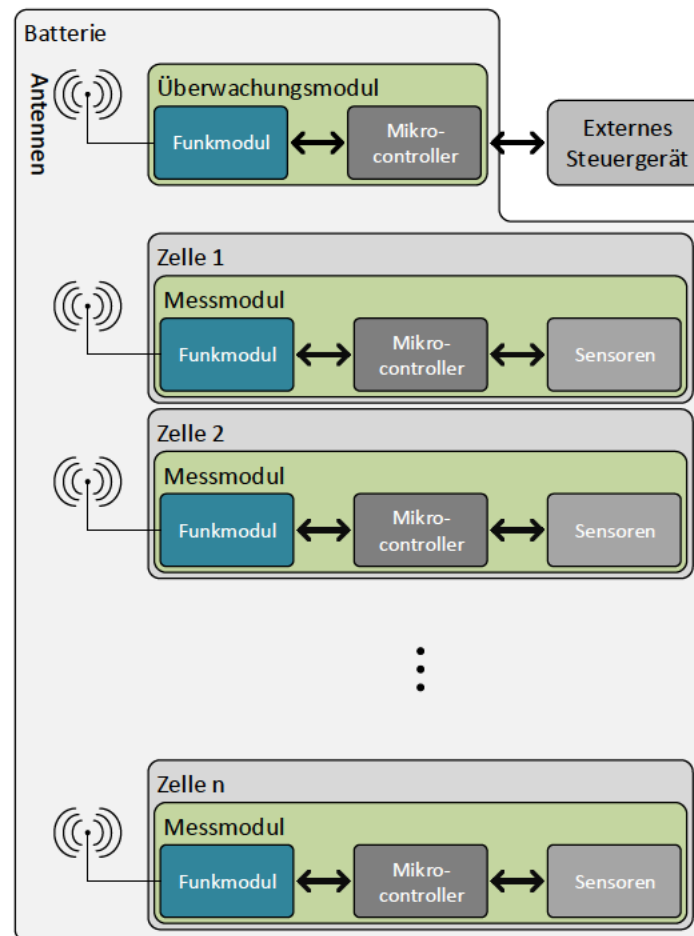


Abbildung 1.4: Systemaufbau mit Datenübertragung über elektromagnetische Verbindung. Die Busteilnehmer sind galvanisch getrennt durch Funktechnologie miteinander verbunden.

1.2.4 Datenübertragung mittels Lichtleitkörper

Alle Busteilnehmer verfügen über je einen optischen Transceiver-Baustein, welche durch einen Lichtleitkörper verbunden sind und durch Lichtsignale Daten übertragen.

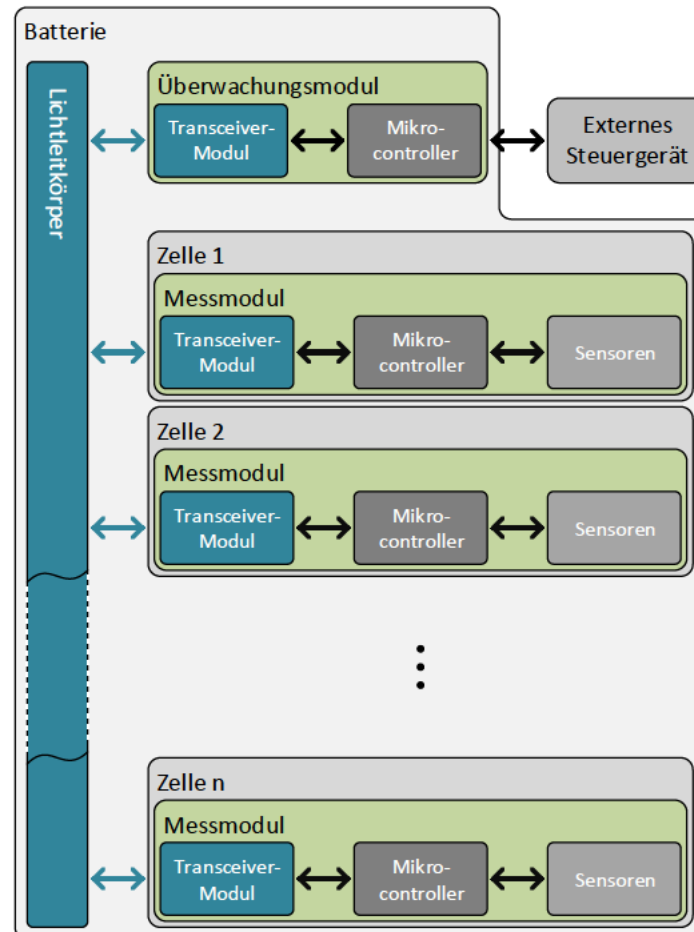


Abbildung 1.5: Systemaufbau mit Datenübertragung über optischen Lichtleitkörper. Jeder Busteilnehmer ist mit einem optischen Transceiver-Baustein ausgestattet, welcher alle galvanisch getrennt in einen Lichtleitkörper verbunden sind.

1.2.5 Vergleich der Optionen

Eine hohe Robustheit gegenüber externen elektromagnetischen Feldern ist in der Nähe von Antriebskomponenten eines Fahrzeugs besonders wichtig, da dort hohe und schnell wechselnden Ströme keine Ausnahme sind. Die drahtlose und PLC-Variante sind in diesem Umfeld besonders störanfällig. Erstere muss ausreichend gegen Außeneinflüsse, aber auch gegen Abstrahlung nach außen abgeschirmt werden. Die leitungsgebundene und mehr noch die optische Variante weisen eine hohe elektromagnetische Verträglichkeit auf, sodass die Schirmung innerhalb der Batterie minimal gehalten werden kann.

Der Verkabelungsaufwand hält sich bei der PLC-Variante in Grenzen und entfällt bei den drahtlosen und optischen Varianten gänzlich. Die leitungsgebundene Variante hingegen hat für die gemeinsamen Masse- und Datenleitung viermal so viele Steckverbindungen wie Busteilnehmer minus eins. Dies sorgt für eine hohe Fehleranfälligkeit gegenüber den anderen Varianten. Für die optische Variante ist erforderlich, dass die Ein-/Ausstrahlflächen des Lichtleitkörpers und die Linsen der Transceiver staubfrei und unverschmutzt bleiben, damit die Signalstärke nicht reduziert wird. Dieses Problem gibt es bei den anderen Varianten nicht.

Tabelle 1.1: Vergleich der Kommunikationskonzepte

Datenübertragung mittels ...				
Vergleichsbereich	elektrischer Leitungen	Hauptstromleitung	elektromagnetischer Strahlung	Lichtleiterkörper
Elektromagnetische Vertäglichkeit	Bei geeigneter Verlegung auch ohne Schirmung robust gegen elektromagnetische Störungen.	Stromschwankungen durch Belastung können die Übertragung stören.	Der Kommunikationsraum muss gut gegen externe elektromagnetische Strahlung, aber auch gegen Abstrahlung geschirmt sein.	Unempfindlich gegenüber elektromagnetischen Störungen.
Installationsaufwand	Zwischen allen nebeneinander liegenden Busteilnehmern muss eine Kommunikationsleitung gelegt werden.	Das Einkoppellement jeder Zelle muss an die Hauptstromleitung angebracht werden.	Keine Installation eines Übertragungsmediums erforderlich.	Der Lichtleiterkörper muss präzise eingelegt und gut fixiert werden.
Störanfälligkeit	Viele (Steck-) Verbindungen erhöhen Anfälligkeit für Störungen.	Lastbedingte Stromschwankungen und elektromagnetische Strahlung könnten für Störungen sorgen.	Elektromagnetische Strahlung kann die Übertragung von Daten stören.	Mechanische Einflüsse oder Verschmutzung könnten stören.

1.3 Inhalt und Ziel dieser Arbeit

Ziel dieser Arbeit ist es, ein Konzept der optischen Datenübertragung mittels eines geeigneten Lichtleitkörpers in Form eines Demonstrationsaufbaus zu realisieren. Es sollen daher vielfach einfache aber funktionale Lösungen zur Umsetzung verwendet werden.

Die Anordnung und Abmessungen sollen dabei einer realen Batterie beziehungsweise einem realen Batteriemodul entsprechen. Das Modul umfasst zwölf Batteriezellen, weshalb Messdaten von zwölf Messmodulen erfasst und über einen entsprechend dimensionierten und geformten Lichtleitkörper an eine Modulüberwachung übertragen werden sollen. Die Kommunikation soll nach dem Master-Slave-Prinzip erfolgen und die übertragenen Messdaten durch Bedienelemente an den Messmodulen simuliert werden. Hierbei steht vor allem die reine Funktionsweise der Übertragungsmöglichkeit im Vordergrund, nicht der übertragbare Inhalt.

Der Lichtleitkörper ist zu entwerfen und im Demonstrationsaufbau zu testen.

In den folgenden Kapiteln werden die Themen der System- und Kommunikationskonzeption, geometrische Optik, sowie Hardware und Software bearbeitet. Genutzte externe Vorlagen, Quellen und Materialien sind entsprechend gekennzeichnet.

2 Grundlagen und Konzeption

Diese Kapitel beschreibt und erläutert grundsätzliche Konzepte und Ansätze für die Kommunikation über optische Signale.

2.1 Grundkonzept und Systemaufbau

Das zu untersuchende System setzt sich im Wesentlichen aus den Hauptkomponenten Modulüberwachung, Lichtleitkörper und Messmodule zusammen, welche ein Kommunikationsnetzwerk bilden und die Überwachung der Zellen in einer Batterie durch dezentrale Messwerterfassung ermöglichen. Ein Schema des Batterieaufbaus ist in Abbildung 2.1 gezeigt.

In einer solchen Batterie ist jede zweite Zelle um 180° verdreht, damit diese durch kurze Brückenverbinder elektrisch in Reihe geschaltet werden können. Um die Anzahl an unterschiedlichen Komponenten minimal zu halten, sind alle Zellen und deren Messmodule gleich aufgebaut. Dementsprechend befinden sich die Messmodule im Wechsel auf beiden Seiten des Lichtleitkörpers. Die Zellen versorgen jeweils ihr Messmodul.

Der Lichtleitkörper besteht aus einem lichtleitfähigem Material in geeigneter Form und dient als Kommunikationskanal für alle Busteilnehmer, über welchen Befehle und Daten zwischen Modulüberwachung und Messmodulen durch Lichtsignale ausgetauscht werden. Durch ein geeignetes Übertragungsprotokoll wird eine Kollision mehrerer Übertragungen vermieden. Die Modulüberwachung erfasst die Daten aller Messmodule sowie den Batteriestrom und kann diese über eine eigene Kommunikationsschnittstelle an ein externes Gerät, wie zum Beispiel ein übergeordnetes Zentralsteuergerät weitergeben.

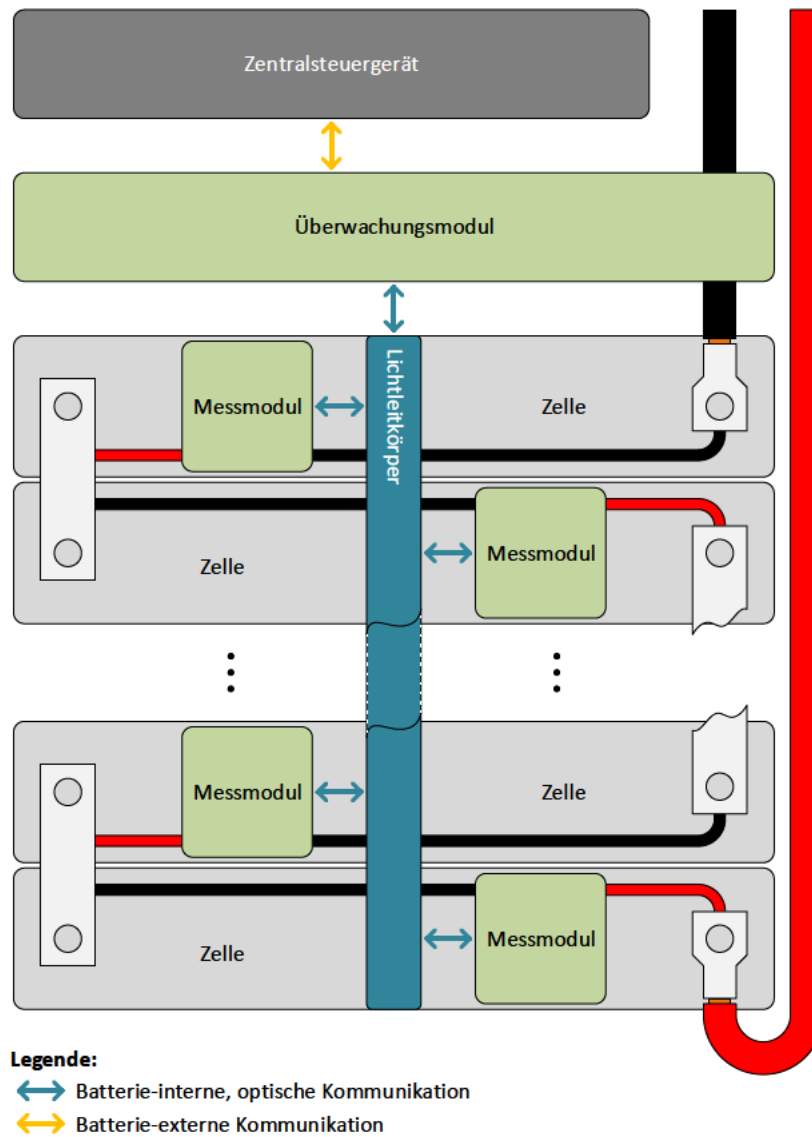


Abbildung 2.1: Systemaufbau mit Datenübertragung über optischen Lichtleitkörper. Auf jeder Zelle der Batterie sitzt ein Messmodul, welches über den Lichtleitkörper mit der Modulüberwachung am Ende der Batterie verbunden ist. Die Pfeile symbolisieren den Datenverkehr.

Demonstrationsaufbau

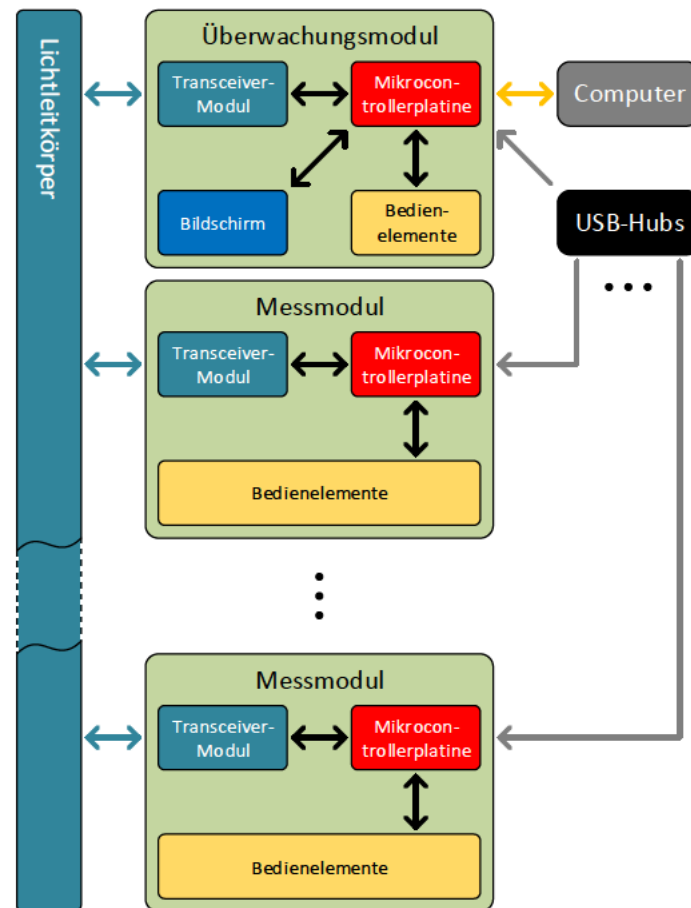
Zur optischen Kommunikation sollen die in Tabelle 2.1 zu findenden Transceiver-Module TFDU4101 sowie die Mikrocontrollerplatine XMC4700 / XMC4800 Relax Kit Series-V1 für die Modulüberwachung beziehungsweise XMC 2Go XMC1100 V1 für die Messmodule eingesetzt werden. Ausschlaggebend für diese Auswahl war, dass Infineon ein Projektpartner ist und die Transceiver-Module optimal alle Funktionen zur Kommunikation nach der bewährten Infrared Data Association (IrDA) Serial Infrared (SIR) Spezifikation in einer Komponente vereinen.

Für den Demonstrationsaufbau wird der Systemaufbau etwas abgeändert, wobei der optische Teil sich mechanisch sehr nahe an einem Batteriemodulrahmen orientieren soll, da der Schwerpunkt dieser Arbeit in der Realisierung und Erprobung der optischen Kopplung liegt.

Da in dem Demonstrationsaufbau keine Leistung von der Batterie abverlangt werden soll, können die Brückenverbinder zur Reihenschaltung der Zellen entfallen, was die Versorgung aller Komponenten durch ein Netzteil anstatt einzelner Zellen ermöglicht und somit für einen geringeren Betriebsaufwand sorgt. Dadurch besteht zwar keine galvanische Trennung zwischen den Busteilnehmern, dies ist wegen der entfallenen Zellverbinder aber auch nicht mehr erforderlich. Aus diesem Grund werden die Mikrocontrollerplatinen, welche jeweils all ihre angeschlossenen Komponenten versorgen sollen, durch zwei USB-Hubs über ihren USB-Mikro-B-Anschluss versorgt.

Anstelle von realen Messwerten sollen möglichst einfach veränderbare Werte erzeugt werden, da auf reale Zellen verzichtet wurde. Die so erzeugten Daten und Informationen sollen dann durch die Messmodule erfasst und von der Modulüberwachung abgefragt werden, sodass diese in Abhängigkeit der Nutzerauswahl durch Bedienelemente an der Modulüberwachung auf einem eingebauten Bildschirm dargestellt und zusätzlich über die im Mikrocontroller integrierte USB-zu-Seriell-Schnittstelle abgefragt werden können.

Ein Blockschaltbild des Demonstrationsaufbaus als Übersicht ist in Abbildung 2.2 zu sehen.



Legende:

- ↔ Batterie-interne, serielle, optische Kommunikation
- ↔ Batterie-externe, serielle, elektrische Kommunikation
- ↔ interne Kommunikation oder Verbindung
- elektrische Versorgung

Abbildung 2.2: Blockschaltbild des Demonstrationsaufbaus. Die Zellen entfallen und stattdessen werden Messwerte über Bedienelemente an den Messmodulen generiert. Alle Mikrocontrollerplatinen werden über zwei USB-Hubs versorgt. Die Modulüberwachung verfügt über Bedienelemente und einen Bildschirm, welcher Daten und Informationen anzeigen kann.

2.2 Eingesetzte Komponenten

Für die Umsetzung der Arbeit wurden die in Tabelle 2.1 aufgelisteten Komponenten bereitgestellt. Im Verlaufe der Entwicklung kamen weitere hinzu, welche in Tabelle 2.2 zu finden sind.

Tabelle 2.1: Liste der bereitgestellten Komponenten für den Demonstrationsaufbau


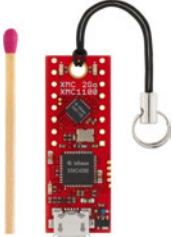
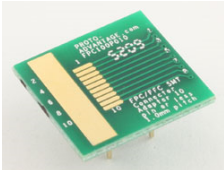

Komponente	Beschreibung	Abbildung
<p>Mikrocontrollerplatine: Infineon Technologies AG XMC4700 Relax Kit V1 <i>KIT_XMC47_RELAX_V1</i></p>	<p>Mikrocontroller-Evaluierungsplatine des XMC4700-F144K2048 Mikrocontrollers mit On-Board-Debugger, General-Purpose Input / Output (GPIO), Light-Emitting Diode (LED), Tastern, zwei Universal Serial Bus (USB) Mikro-B-Anschlüssen, microSD-Karteneinschubschacht und Ethernetanschluss.</p>	
<p>Mikrocontrollerplatine: Infineon Technologies AG XMC 2Go XMC1100 V1 <i>KIT_XMC_2GO_XMC1100_V1</i></p>	<p>Mikrocontroller-Evaluierungsplatine des XMC1100 Mikrocontrollers mit On-Board-Debugger, GPIO, LED und USB-Mikro-B-Anschluss.</p>	
<p>Adapterplatine: Proto Advantage FPC/FFC SMT Connector (1 mm pitch, 10 pin or less) DIP Adapter <i>FPC100P010</i></p>	<p>Adapterplatine mit zehn Lötflächen für einen FPC/FFC Anschluss mit 1 mm Pinabstand auf der Oberseite und zehn zugehörige Lötflächen für zwei Pinreihen auf der Unterseite.</p>	
<p>Transceiver-Modul: Vishay Intertechnology, Inc. Infrarot-Transceiver-Modul <i>TFDU4101</i></p>	<p>Infrarot-Transceiver-Modul für optische Kommunikation nach IrDA-Standards.</p>	

Tabelle 2.2: Liste der während der Entwicklung ergänzten Komponenten

Komponente	Beschreibung	Abbildung
<p>Bildschirm: 2004A Liquid Crystal Display (LCD)-Modul mit Inter-Integrated Circuit (I2C)-Schnittstelle</p>	<p>LCD-Modul mit 4 Zeilen, welche je 20 Zeichen darstellen können, auf Basis einer Hitachi HD44780 Steuereinheit mit Ein-/Ausgangserweiterungsmodul für die Ansteuerung über eine I2C-Schnittstelle.</p>	 <p>[29]</p>
<p>Bedienelemente: Grayhill Drehschalter <i>94HBB16T</i> mit Griff <i>947705-019</i>, C&K Drucktaster <i>D6R10F2LFS</i>, TT Electronics PLC Potentiometer <i>P170NP1-QC15BR50K</i> und Vishay Intertechnology, Inc. LED <i>TLLE4401</i></p>	<p>Drehschalter mit 16 Positionen und Standard-Binär-Code-Ausgang (oben rechts), Drucktaster mit Schließerkontakt (unten rechts), Potentiometer mit 50 kΩ (mittig links) und eine gelbe LED (oben mittig)</p>	
<p>USB-Hub: Markenloser USB-Hub</p>	<p>USB-Hub mit sieben abschaltbaren USB-3.0-Typ-A-Anschlüssen. Die Verbindung zu einem Computer wird über einen USB-3.0-Micro-B-Anschluss hergestellt. Für die Versorgung der angeschlossenen Geräte ist ein männlicher Hohlsteckeranschluss mit 1,3 mm Mittelstiftdurchmesser (positives Potential) für einen weiblichen Hohlstecker mit 3,5 mm Außendurchmesser vorhanden.</p>	 <p>[30]</p>

2.3 Kommunikationsprotokoll

Alle Busteilnehmer sind an den einteiligen Lichtleitkörper angebunden und empfangen somit jede Übertragung, auch wenn diese nicht an den entsprechenden Busteilnehmer gerichtet sind. Mehrere gleichzeitige Lichtsignale überlagern sich, sodass nur im Wechselbetrieb (Halbduplex) kommuniziert werden darf, damit übertragende Daten erhalten bleiben.

Auf Grund dessen und wegen des Systemkonzepts bietet sich eine Kommunikationskoordination nach dem Master-Slave-Prinzip an. Nur auf einen Befehl der Modulüberwachung als Master führt das angesprochene Messmodul als Slave diesen aus und sendet eine entsprechende Antwort zurück. Das Master wartet für eine definierte Zeit auf eine Antwort und fährt bei Ausbleiben einer solchen und nach Ablauf der Wartezeit mit dem nächsten Slave fort. Während dieser Wartezeit darf kein anderer Busteilnehmer außer dem Angesprochenen senden. Dadurch ist eine Kollisionsfreiheit sichergestellt.

Da in dieser Arbeit die optische Übertragung im Vordergrund steht, wurde ein einfaches, serielles Protokoll gewählt. Für die Kommunikation zwischen den Busteilnehmern wurden die Universal-Serial-Interface-Channel (USIC) der Mikrocontroller als Universal-Asynchronous-Receive-Transmit (UART) mit der Data/Parity/Stop (DPS) Notation $8E1$ konfiguriert, wobei 8 für die Anzahl der Datenbits, E für gerade Parität und 1 für die Anzahl der Stoppbits steht.

Damit jeder Busteilnehmer erkennen kann, ob eine Nachricht für ihn bestimmt ist, muss eine Adresse mitgesendet werden. Auf Grund der insgesamt 13 Busteilnehmer (zwölf Messmodule und eine Modulüberwachung) bietet sich zur Maximierung der übertragbaren Datenbits eine Adresse mit 4 Bit an, welche einen Adressbereich von 0 bis 15 abdeckt und 4 Bit für Befehle und Daten lässt.

Die Adresszuweisung kann der Tabelle 2.3 entnommen werden. Eine Besonderheit hier ist, dass mit der letzten Adresse 15 alle Slaves angesprochen werden, welche dann jedoch nicht antworten dürfen, damit es keine Signalüberlagerung gibt. Eine Antwort aller Slaves nacheinander in Zeitfenstern entsprechend ihrer Adresse wäre denkbar, wurde aber nicht vorgesehen.

Mit dieser Aufteilung ergibt sich der Sendesignalverlauf einer UART-Standardübertragung wie in Abbildung 2.3 zu sehen.

Tabelle 2.3: Adresszuweisung der Busteilnehmer

Adressbereich	Anwendung
0	Adresse zur direkten Ansprache des Masters (Modulüberwachung)
1...12	Adresse zur direkten Ansprache eines Slaves (Messmodule)
13...14	Unvergebene Adressen
15	Broadcast an alle Slaves (Messmodule)

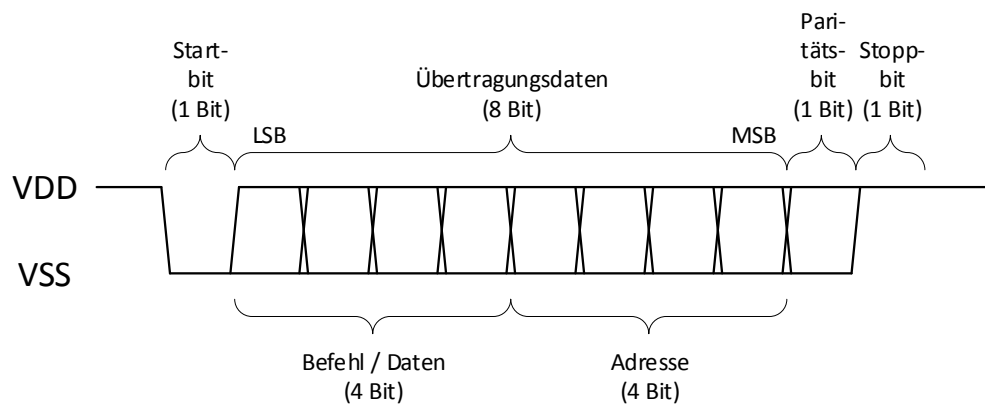


Abbildung 2.3: Paketaufbau und Signalverlauf einer UART-Standardübertragung. Signalpegel und -verlauf entsprechen den üblichen Eigenschaften. Die acht Datenbits beinhalten die Zieladresse und den Befehl oder die Daten.

Anpassung des Signalverlaufs für die Transceiver-Module

Da das Transceiver-Modul *TFDU4101* für Übertragungen nach der IrDA SIR Spezifikation ausgelegt ist [34, S. 1], muss die Bildung des Sendesignals so angepasst werden, dass der Leerlaufpegel 0 V ist und ein hoher elektrischer Pegel immer vor dem Ende der Bitdauer wieder auf 0 V zurückkehrt [34, S. 2], da das Transceiver-Modul die interne Sendediode bei länger als $50 \mu\text{s}$ andauernden Sendesignalpulsen automatisch abschaltet [34, S. 2, 7] und sie erst bei der nächsten steigenden Sendesignalflanke wieder einschaltet. Das Empfangssignal hat im Leerlauf einen hohen elektrischen Pegel und generiert beim Empfangen eines Lichtpulses einen von der Dauer des Lichtpulses unabhängig langen Signalpuls am Ausgang mit 0 V. Diese Empfangssignalpulsdauer ist konstant und liegt zwischen $1,65 \mu\text{s}$ und $3 \mu\text{s}$ [34, S. 5]. Im Datenblatt wird diese Dauer für einen eingehenden Lichtpuls ab mehr als $1,2 \mu\text{s}$ angegeben [34, S. 5], weswegen folgend diese Zeit als Mindestlichtsignalpulsdauer angesehen wird. Für die Zeit zwischen den Sendesignalpulsen ist keine Mindestzeit angegeben.

Beide Mikrocontroller unterstützen sogenanntes *pulse shaping* [20, S. 342 bis 343] [21, S. 1809 bis 1810], wobei im Sendesignal ein hoher elektrischer Pegel vor Ablauf der Bitdauer wieder zum niedrigen wechselt. Der Abtastzeitpunkt kann ebenfalls wie vorgegeben eingestellt werden [20, S. 340 bis 341] [21, S. 1807 bis 1808], allerdings ist hier darauf zu achten, dass der Abtastmodus nicht im Mehrheitsmodus mit den Werten der letzten drei Zeitabschnitte, sondern im Einzelabtastmodus mit nur dem aktuellen Wert arbeitet, da der Empfangssignalpuls ebenfalls kürzer als die Bitdauer ist und durch Takt- oder Synchronisationsabweichung der Auswertzeitpunkt relativ zum ersten Bit einer Übertragung mit jedem weiteren Bit weiter verschoben werden kann, sodass möglicherweise Abtastwerte noch vor dem Wechsel zum hohen elektrischen Pegel ausgewertet werden.

Somit ergibt sich der angepasste Sendesignalverlauf wie in Abbildung 2.4 zu sehen.

Bei der maximalen Datenrate von 115200 Bit/s [34, S. 1] beträgt die Bitdauer $8,681 \mu\text{s}$. In der IrDA SIR Spezifikation wird die Dauer eines hohen elektrischen Pegels im Sendesignal mit $3/16$ der Bitdauer vorgegeben [11, S. 2, Kapitel *How IrDA Transmission Works*], was bei dieser Datenrate $1,628 \mu\text{s}$ entspricht.

Da ein Empfangssignalpuls von mindestens $1,65 \mu\text{s}$ etwa $3/16$ einer Bitdauer entspricht, wird für den Messzeitpunkt zur Erfassung des Eingangswertes der Zeitpunkt $2/16$ der Bitdauer gewählt.

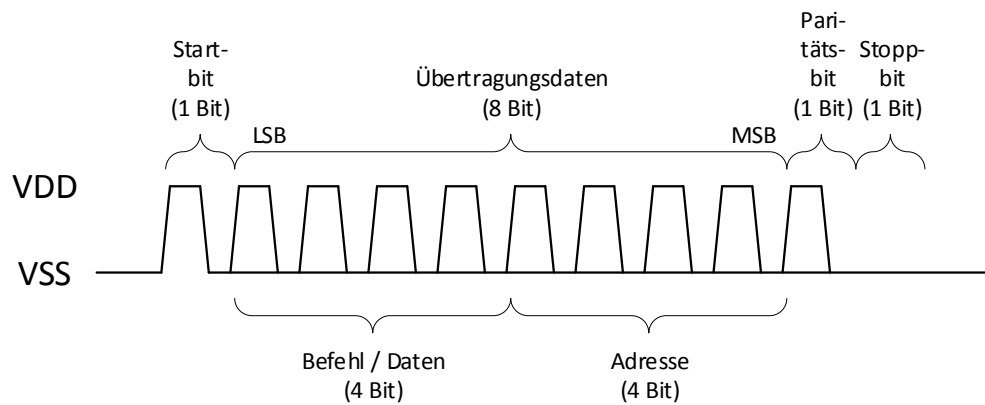


Abbildung 2.4: Paketaufbau und Verlauf eines Sende- oder invertierten Empfangssignals einer UART-Übertragung in Anlehnung an die IrDA SIR Spezifikation. Der Sendesignalpegel ist im Vergleich zu einer UART-Standardübertragung invertiert und die Bitdauer ist gekürzt. Die acht Datenbits beinhalten die Zieladresse und den Befehl oder die Daten.

Mit diesen Einstellungen verläuft die Übertragung eines Bits mit logisch 0 wie in Abbildung 2.5 links vereinfacht dargestellt.

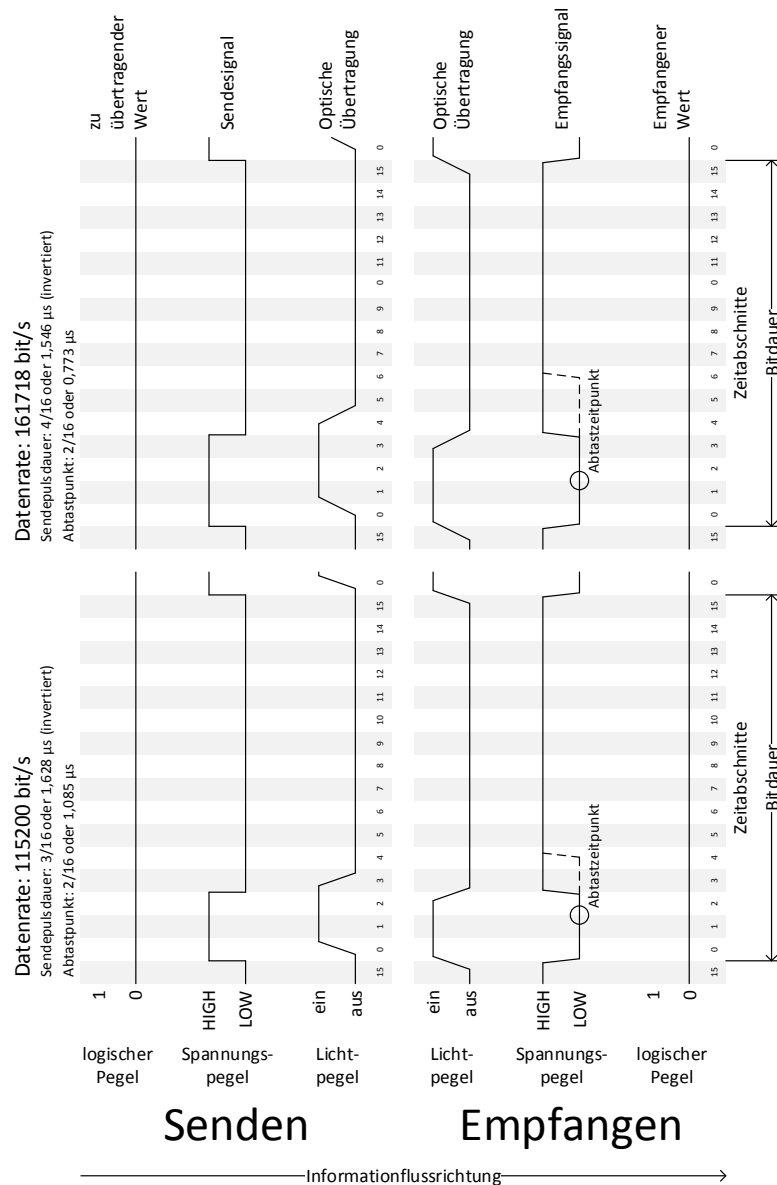


Abbildung 2.5: Theoretischer Ablauf der Übertragung eines Bits von einem zum anderen Busteilnehmer. In diesen Verläufen wurden die Empfängerlatenz, die Anstiegs- und Abfallzeiten der Controllerein- und -ausgänge, die Auswertzeit der Controllerein- und -ausgänge und die Lichtpulsverlängerung durch Streckenunterschiede des reflektierten Lichts vernachlässigt. Weiterhin wurde vereinfachend angenommen, dass das Empfangssignal und der empfangene Wert bei einem eingehenden Lichtstärke- beziehungsweise Spannungspegel von 50 % ihren Zustand wechselt.

Einfluss von Taktabweichung und Jitter

Kleine Taktabweichungen zwischen Sender und Empfänger können für eine fehlerhafte Übertragung sorgen, da der Abtastzeitpunkt mit jedem weiteren Bit vor oder hinter den Sendesignalpuls wandern kann.

Im schlimmsten Fall weicht der Takt des XMC1100 um 4 % nach oben [19, S. 50] und der des XMC4700 um 0,0025 % nach unten [35, S. 1] ab. Entsprechend gestaltet sich auch die Abweichung der Bitdauern nach Gleichung 2.1.

Diese Verschiebung addiert sich mit jedem weiteren Bit nach dem Startbit auf. Hinzu kommt ein Jitter des Transceiver-Moduls von $t_{bitJitter} = 0,25 \mu s$ [34, S. 5], sodass sich die zeitliche Verschiebung zwischen dem Startbit und dem Paritätsbit nach Gleichung 2.3 berechnet. Das Stoppbit wird hier nicht berücksichtigt, da es keine Information zum Abtasten enthält. Somit ergibt sich die Anzahl der zu berücksichtigenden Bits zu $n_{bitn} = 10$.

$$dt_{bitWcErr} = |t_{bitWcErr} - t_{bitWcErr}|$$

$$= \left| \frac{1}{f_{bitSetXMC1100} \cdot (1 + dev_{WcErrXMC1100})} - \frac{1}{f_{bitSetXMC4700} \cdot (1 + dev_{WcErrXMC4700})} \right|$$
(2.1)

$$= \left| \frac{s}{115234 \cdot (1 + 0,04)} - \frac{s}{115204 \cdot (1 - 0,000025)} \right|$$

$$= 336,246 ns$$
(2.2)

$$dt_{bitn} = (n_{bit} - 1) \cdot dt_{bitWcErr} + t_{bitJitter}$$
(2.3)

$$= (10 - 1) \cdot 336,246 ns + 0,25 \mu s$$

$$= 3,276 \mu s$$

Dieser Wert darf nur halb so lang wie die Empfangsignalpulsdauer sein, damit der Abtastzeitpunkt noch innerhalb des Signalpulses liegt. Dies jedoch nach Gleichung 2.4 nicht erfüllt.

$$dt_{bitn} < \frac{t_{pulseLength}}{2} \quad (2.4)$$
$$3,276 \mu s \not< \frac{1,65 \mu s}{2} = 0,825 \mu s$$

Anhebung der Datenrate zur Reduzierung des Einflusses der Taktabweichung

Da die Empfangssignalpulsdauer nicht verlängert werden kann, muss zur effektiven Verlängerung dessen die Übertragungsrate angehoben werden. Zur Einhaltung der Mindestdauer des Lichtsignalpulses von $1,2 \mu s$, muss die Sendesignalpulsdauer entsprechend angepasst werden.

In den Tests aus Kapitel 4.2.1 ergab sich die höchste Datenrate ohne Fehler im Empfangssignal zu 216008 Bit/s. Um Fehler in der Kommunikation zu vermeiden und Toleranzen zwischen den Transceiver-Modulen zu berücksichtigen, wird die ermittelte Taktrate um mindestens 25 % auf

$$f_{MAX} \leq 216008 \text{ Bit/s} \cdot 0,75 = 162006 \text{ Bit/s}$$

reduziert.

Letztendlich wurde die Datenrate auf 161718 Bit/s eingestellt, da die *UART-APP Version 4.1.12* in der Programmier- und Debugsoftware *DAVE Version 4.4.2* von *Infineon Technologies India Pvt. Ltd* zur Konfiguration der UART-Schnittstellen für beide Mikrocontroller exakt diesen Wert einstellen kann. Dies soll helfen, Übertragungsfehler durch Unterschiede in den Datenraten zu vermeiden.

Aus dieser Datenrate ergibt sich die Bitdauer zu $6,184 \mu s$. Für eine Dauer des Sendesignalpulses von $1,546 \mu s$ wurde diese auf $4/16$ der eines Bits gelegt, womit die Mindestpulsdauer von $1,2 \mu s$ eingehalten wird. Zum Ausgleich von Taktabweichungen der Busteilnehmer in beide Richtungen, wurde der Abtastzeitpunkt möglichst nahe an die zeitliche Mitte des Empfangssignals mit $2/16$ bei $0,773 \mu s$ nach der steigenden Empfangssignalfanke gesetzt.

Die vereinfacht dargestellte Übertragung mit diesen Einstellungen ist in Abbildung 2.5 rechts zu sehen.

Wiederholt man mit dieser Datenrate die Berechnung der zeitlichen Verschiebung zwischen dem Startbit und dem Paritätsbit nach Gleichung 2.3, zeigt sich jedoch in Gleichung 2.5, dass diese Taktanhebung nicht ausreicht, um die größtmöglich Taktabweichung auszugleichen.

$$\begin{aligned} dt_{bitWcErr} &= \left| \frac{s}{161718 \cdot (1 + 0,04)} - \frac{s}{161718 \cdot (1 - 0,000025)} \right| \\ &= 237,985 \text{ ns} \\ \\ dt_{bitn} &= (10 - 1) \cdot 237,985 \text{ ns} + 0,25 \mu\text{s} \\ &= 2,392 \mu\text{s} \not\leq 0,825 \mu\text{s} \end{aligned} \tag{2.5}$$

Ein besserer Ansatz als die Anhebung der Datenrate wäre die Reduzierung der Taktabweichung. Da der Demonstrationsaufbau jedoch nur bei Raumtemperatur eingesetzt wird, wurde die Anhebung der Datenrate beibehalten und keine weiteren Maßnahmen ergriffen.

2.4 Lichtleitkörper

Die Aufgabe des Lichtleitkörpers ist es, eine optische Verbindung zwischen dem Transceiver-Modul der Modulüberwachung als Master und denen der Messmodule als Slaves zu ermöglichen. Dafür muss dieser das Licht zwischen dem Master- und den Slave-Transceiver-Modulen leiten können.

2.4.1 Optische Eigenschaften

Snelliusschen Brechungsgesetz und Totalreflexion

Für die Leitung von Licht ist es erforderlich, dass die Richtung der Lichtstrahlen geändert wird. Hierfür kann eine für die genutzte Lichtwellenlänge reflektierende Oberfläche oder ein Übergang zwischen Medien mit unterschiedlichem Brechungsindex genutzt werden. Je flacher der Eintrittswinkel eines Lichtstrahls auf einen Medienübergang, desto geringer fällt die Abstrahlung in das andere Medium (sogenannte Transmission [33]) aus. Ab

einem ausreichend flachen Eintrittswinkel wird das gesamte Licht reflektiert (sogenannte Totalreflexion [33]). Dieser sogenannte kritischer Winkel [9] kann mit dem Snelliusschen Brechungsgesetz 2.6 berechnet werden, welches zur Berechnung des Austrittswinkels eines Lichtstrahls nach einem Medienübergang dient [9]. α ist der Winkel des eingehenden und β der des ausgehenden Lichtstrahls zur Senkrechten des Medienübergangs. n_1 und n_2 sind der jeweilige Brechungsindex der zwei Medien. Wird der Winkel α vergrößert, vergrößert sich β ebenfalls. Ab $\beta \geq \beta_{TR} = 90^\circ$ herrscht Totalreflexion [9], sodass alle eingehenden Lichtstrahlen mit mindestens dem zugehörigen Winkel $\alpha \geq \alpha_{TR}$ vollständig reflektiert werden [9]. Die Formel zur Berechnung dieses Winkels ist in Gleichung 2.7 zu finden.

Snelliussches Brechungsgesetz [9] [6, S. 82]:

$$\begin{aligned} n_1 \cdot \sin(\alpha) &= n_2 \cdot \sin(\beta) \\ \Rightarrow \alpha &= \arcsin\left(\frac{n_2}{n_1} \cdot \sin(\beta)\right) \end{aligned} \quad (2.6)$$

Mit $\beta = \beta_{TR} = 90^\circ$:

$$\alpha_{TR} = \arcsin\left(\frac{n_2}{n_1}\right) \quad (2.7)$$

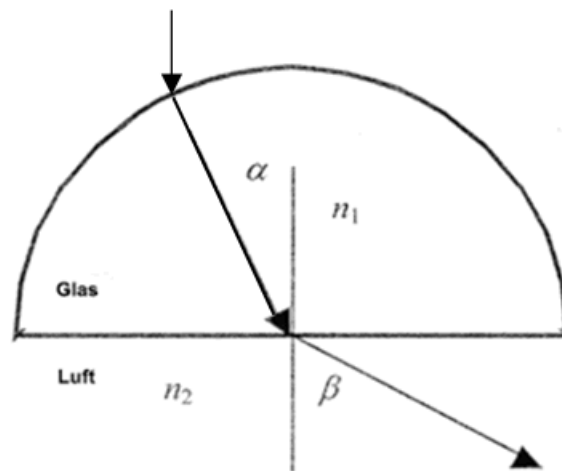


Abbildung 2.6: [9] Snelliusschen Brechungsgesetz. Die Pfeile beschreiben den Pfad und die Richtung des Lichtstrahls, α und β den Winkel zur Senkrechten und n_1 und n_2 den Brechungsindex der Medien.

Beispiel für Lichtumleitung

Ein Beispiel für die genutzten optischen Effekte in einem Lichtleitkörper zur Umleitung der Lichtstrahlen ist in Abbildung 2.7 zu sehen. Hier sendet die Lichtquelle (3) Lichtstrahlen (8) in das Umgebungsmedium mit niedrigem Brechungsindex (1) aus, welche, von einer Linse (5) durch die Brechungsindexdifferenz parallel ausgerichtet, in das Lichtleitkörpermedium mit hohem Brechungsindex (2) gelangen. Dort treffen sie auf einen glatten Medienübergang (6), an welchem sie mit ihrem Eintrittswinkel gerichtet reflektiert werden (9). Auf Grund des relativ spitzen Eintrittswinkels wird ein Teil des Lichts (10) in das Umgebungsmedium (1) transmittiert. Anschließend werden die gerichtet reflektierten Lichtstrahlen (9) an einem rauhen Medienübergang (7) ungerichtet reflektiert und somit zerstreut. Einige Lichtstrahlen (9) werden dabei teilweise und andere nahezu vollständig transmittiert (10), da sie in einem relativ spitzen Eintrittswinkel auf den Medienübergang (7) treffen. Die anderen verbleiben, durch Transmission (10) in das Umgebungsmedium (1) geschwächt, in dem Lichtleitkörpermedium (2). Ein kleiner Teil dieser reflektierten Lichtstrahlen (9) trifft mit einem ausreichend spitzen Eintrittswinkel auf den glatten Medienübergang vor dem Lichtsensor (4), um zurück in das Umgebungsmedium (1) transmittiert und vom Lichtsensor (4) erfasst zu werden.

Verdrehung jeder zweiten Zelle um 180° befinden sich die Transceiver-Module auf beiden Seiten des Lichtleitkörpers, was die verfügbare Breite für den Lichtleitkörper reduziert. Zusätzlich wirkt jeder Versatz des Kanals von der Mitte dadurch doppelt, sodass sich die verfügbare Breite um 10 mm reduzieren würde. Da ein zukünftiger Batterierahmen ohne diesen Versatz konstruiert werden kann, wurde dieser ignoriert.

Die Transceiver-Module haben eine Breite von 4,7 mm, eine Höhe von 9,9 mm und eine Tiefe von 4 mm mit einem Radius der halbkugelförmigen Linsen von 1,7 mm und einem Linsenmittelpunktsabstand von 5,95 mm [34, S. 9]. Zur Positionierung auf den Zellen wurde deren Koordinatenpunkt mittig zwischen den Linsenmittelpunkten gewählt. Auf der gleichen nach außen verlängerten Linie wurden Löcher zur Befestigung des Lichtleitkörpers mit einem Durchmesser von 3,2 mm und einem vertikalen Abstand von 19 mm vorgesehen. Dies lässt für den Schraubenkopf einen Durchmesser von 7 mm, ohne dass die Befestigungspunkte über die Grundfläche der Zelle hinaus stehen. Der Abstand zwischen dem Linsenmittelpunkt und der Transceiver-Modul-Rückseite beträgt 3 mm [34, S. 9]. Mit einem Abstand zur Grundflächenaußengrenze von 1 mm für die Lötflächen der Platine, beträgt der horizontale Abstand zwischen den gegenüberliegenden Befestigungslöchern und Linsenmittelpunkten der Transceiver-Module 30 mm. Vertikal werden die Transceiver-Module mittig auf der Zelle positioniert.

Das Transceiver-Modul für die Modulüberwachung (Master) wurde, ohne Berücksichtigung der Batterierahmenaußenmaße, horizontal mittig und 0,5 mm unter der Außenkante der ersten Zelle gesetzt.

Der Abstand zwischen den Linsen und dem Lichtleitkörper soll 0,5 mm betragen. Somit ergibt sich die maximale Breite zu 25,6 mm. Die Höhe ist durch die Linsen des Master-Transceiver-Moduls und die Außenkante der letzten Zelle auf 321,3 mm eingeschränkt. Bei dem Demonstrationsaufbau ist die Stärke des Lichtleitkörpers nicht begrenzt.

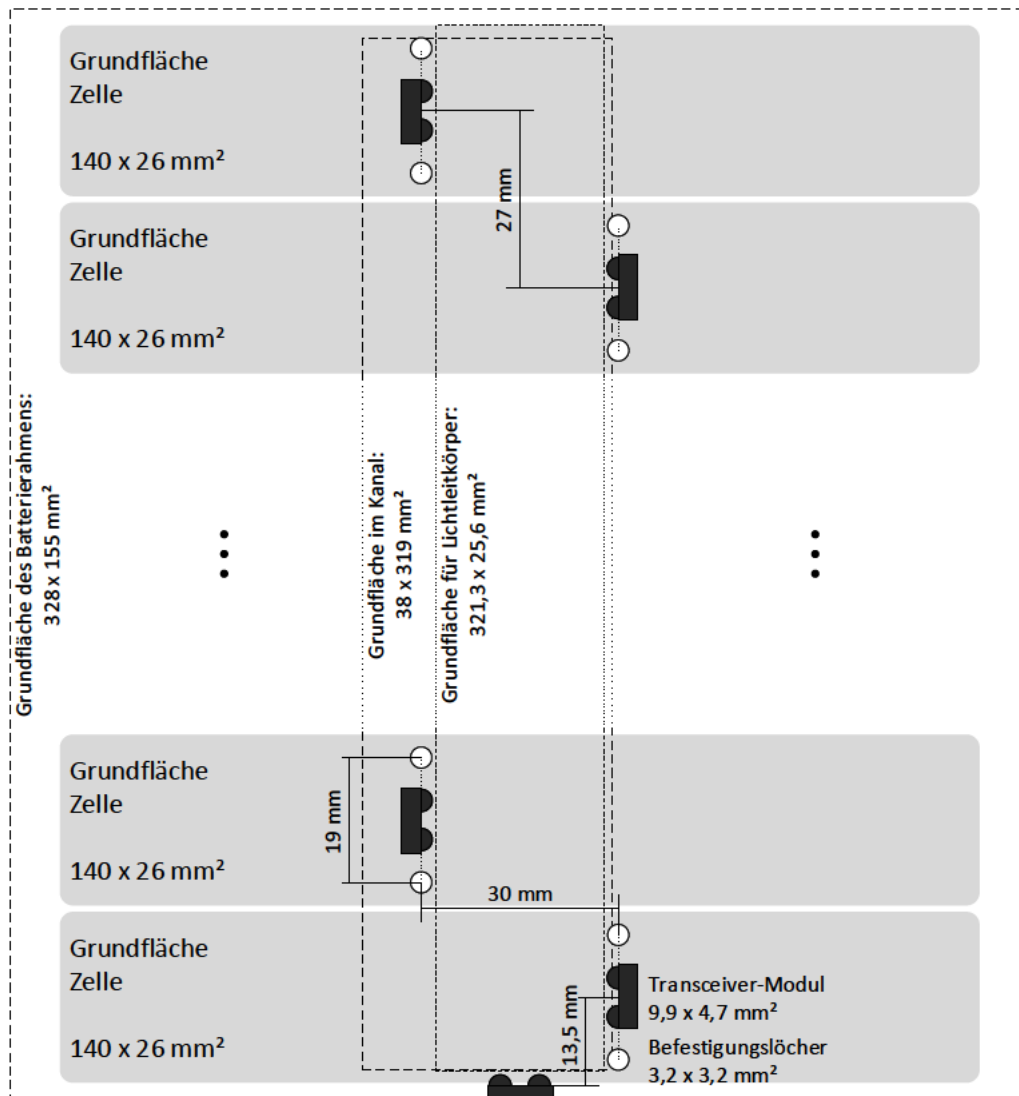


Abbildung 2.8: Verfügbare Grundfläche für den Lichtleitkörper. Eingezeichnet sind Grundflächen der Zellen, Transceiver-Module, Befestigungslöcher für den Lichtleitkörper, Außengrenzen des Batterierahmens und dessen integrierten Kanals sowie die vorgesehene Grundfläche für den Lichtleitkörper zwischen den Transceiver-Modulen.

2.4.3 Konzepte des Lichtleitkörpers

Zur Umleitung der Lichtsignale gibt es verschiedene Formen, welche unterschiedliche Eigenschaften nutzen. In dieser Arbeit werden nur zweidimensionale Entwürfe betrachtet.

Streukammer

Dieser simple Entwurf sieht eine rechteckförmige Kammer ohne Details vor. Die Oberflächen könnten dabei rau oder glatt sein. Hierbei wird sich darauf verlassen, dass ausreichend Licht an den benötigten Stellen transmittiert wird.

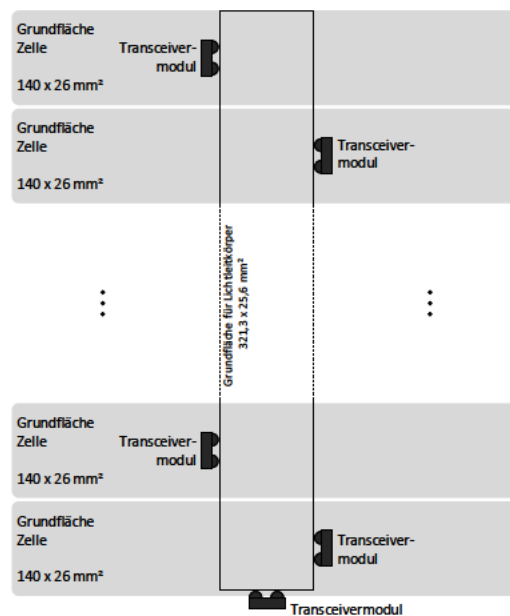


Abbildung 2.9: Lichtleitkörperkonzept der Streukammer.

Reflexionsbohrungen

Eine rechteckförmige Kammer wird mit kleinen Bohrungen vor den Linsen der Slave-Transceiver-Module versehen, welche das auftreffende Licht zerstreuen. Damit der senkrechte Lichtpfad der Slave-Transceiver-Module mit 90° in Richtung des Masters reflektiert wird, sitzen diese Bohrungen um die Distanz $d = \frac{r}{\sqrt{2}}$ weiter entfernt von diesem, wobei

r der Bohrungsradius ist. Die Linsen haben dabei, von der Seite des Master-Transceiver-Moduls ausgehend, einen abnehmenden Abstand zu den Slave-Transceiver-Modul-Linsen, damit sie nicht das reflektierte Licht der dahinter liegenden Bohrung blockieren. Vor den Slave-Transceiver-Modul-Senderlinsen könnten Linsen im Lichtleitkörper integriert sein, welche die ausgesendeten Lichtstrahlen parallel ausrichten, damit mehr Licht auf die Bohrungen trifft. Die Oberflächen könnten dabei rau oder glatt sein, um die Lichtstrahlen mehr oder weniger zu streuen.

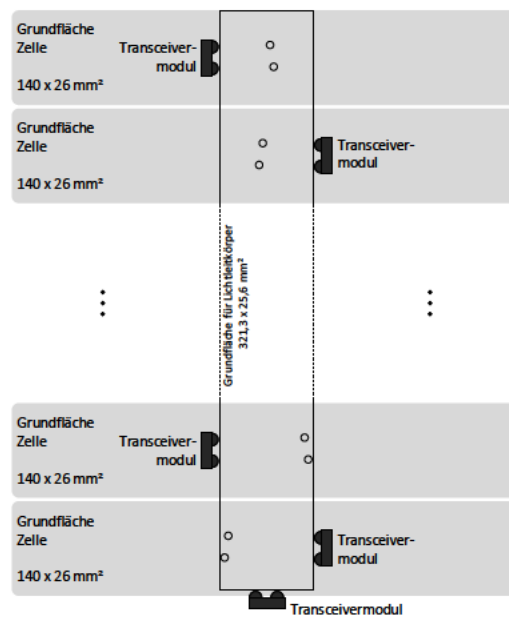


Abbildung 2.10: Lichtleitkörperkonzept mit Reflexionsbohrungen.

Streuende Reflektorflächen

Der Lichtkanal hat am Master-Transceiver-Modul die maximale Breite. Vor jeder Slave-Transceiver-Modul-Linse befindet sich ein Viertelkreis, welcher die Breite des Lichtleitkörpers nach oben um seinen Radius reduziert und einen kleinen Teil der Lichtstrahlen so reflektieren soll, dass diese auf die Linsen der Transceiver-Module treffen. Hierfür befindet sich der Viertelkreismittelpunkt nicht direkt vor den Slave-Transceiver-Modul-Linsen, sondern ist um die Distanz $d = \frac{r}{\sqrt{2}}$ weiter entfernt von Master platziert, wobei r der Viertelkreisradius ist. Dabei werden die von den Slave-Transceiver-Modulen ausgesendeten

Lichtstrahlen durch eine Linse parallel ausgerichtet, damit mehr auf dem Viertelkreis landen. Die Oberflächen sollten für eine gerichtete Reflexion glatt sein.

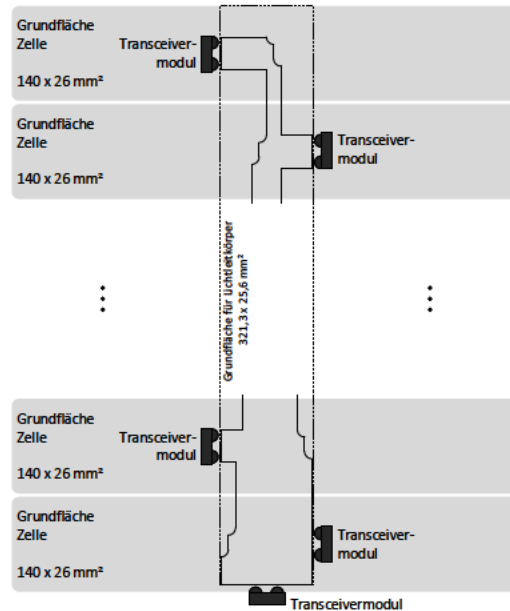


Abbildung 2.11: Lichtleitkörperkonzept mit streuenden Reflektorflächen.

Ausgerichtete Reflektorflächen

Vor jeder Transceiver-Modul-Linse der Messmodule befindet sich eine auf die zugehörige Master-Transceiver-Modul-Linse ausgerichtete Fläche, welche die Breite des Lichtleitkörpers nach oben reduziert und alle Lichtstrahlen auf die entsprechenden Linsen des Kommunikationspartners reflektieren soll. Die von den Slave-Transceiver-Modulen ausgesendeten Lichtstrahlen werden durch eine Linse parallel ausgerichtet, damit mehr auf der Reflektorfläche landen. Für die gerichtete Reflexion sollten die Oberflächen glatt sein.

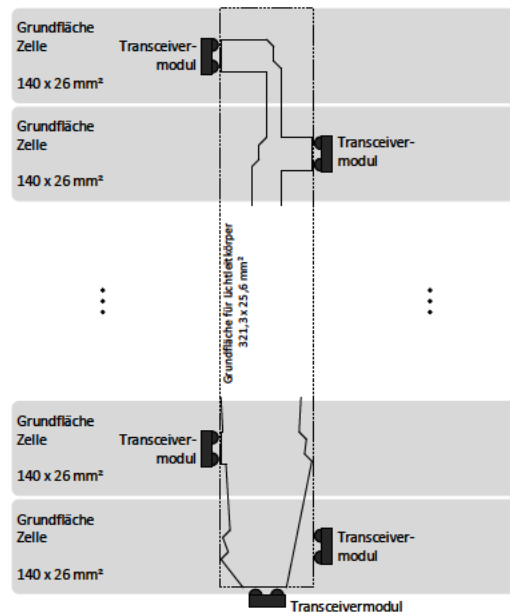


Abbildung 2.12: Lichtleitkörperkonzept mit ausgerichteten Reflektorflächen.

Lichtleitbahnen

Von den Linsen des Master-Transceiver-Moduls verläuft jeweils eine schmale Lichtleiterbahn, welche in einer 90°-Kurve vor die letzten Slave-Transceiver-Modul-Linsen münden. Aus diesen zwei Lichtleiterbahnen führen vor jedem Messmodul zwei der Signalrichtung entsprechende 90°-Kurven zu den Transceiver-Modul-Linsen. Durch die flachen Eintrittswinkel der Lichtstrahlen sollten diese in Totalreflexion verlustarm geleitet werden. Dies erfordert eine glatte Oberfläche des Lichtleitkörpers.

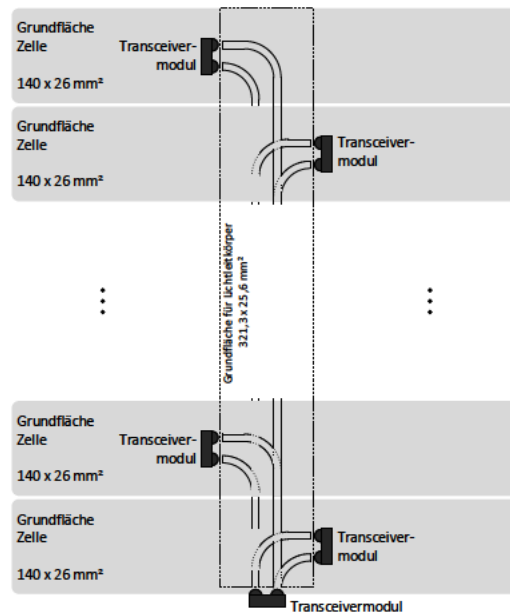


Abbildung 2.13: Lichtleitkörperkonzept mit Lichtleitbahnen.

Auswahl eines Lichtleitkörperkonzeptes

Zum Vergleich der Konzepte wurden mögliche Vor- und Nachteile ermittelt, welche in der Tabelle 2.4 zu finden sind.

Wegen der großen erwarteten Verluste beim Konzept der Streukammer und der geringen Reichweite bei den Konzepten der Reflexionsbohrungen und streuenden Reflektorflächen wurden diese nicht für die Umsetzung ausgewählt.

In diesen theoretischen Betrachtungen stellt sich das Konzept der Lichtleiterbahnen als vermutlich beste Option heraus. Auf Grund von fertigungstechnischen Schwierigkeiten wurde jedoch das Konzept der ausgerichteten Reflektorflächen für die erste Version des Lichtleitkörpers ausgewählt.

Tabelle 2.4: Vor- und Nachteile der verschiedenen Lichtleitkörper

Lichtleitkörperkonzept	Vorteile	Nachteile
Streukammer	- sehr große Positionstoleranz der Transceiver-Module	- sehr geringe Effizienz durch hohe Transmission an den Seiten gegenüber der Transceiver-Modul-Linse
Reflexionsbohrungen	- ohne Linsen große Positionstoleranz der Transceiver-Module	- geringe Effizienz durch Transmission an den Seiten gegenüber der Transceiver-Modul-Linse - kleine Reflexionsflächen - große Streuung und damit limitierte Reichweite
Streuende Reflektorflächen	- ohne Linsen große Positionstoleranz der Transceiver-Module	- geringe Effizienz durch etwas Transmission bei den steilen Eintrittswinkeln an den Viertelkreisen reduziert Effizienz - kleine Reflexionsflächen - große Streuung und damit limitierte Reichweite
Ausgerichtete Reflektorflächen	- keine Streuung und entsprechend große Reichweite	- etwas Transmission bei den steilen Eintrittswinkeln an den Reflexionsflächen - auch ohne Linsen sehr kleine Positionstoleranz der Transceiver-Module - erfordert geringe Fertigungstoleranzen - reduzierte Effizienz wegen Transmission durch geringen Eintrittswinkel bei kleinem Abstand zwischen Master- und Slave-Transceiver-Modul - großer Rechenaufwand
Lichtleitbahnen	- geringer Materialbedarf - hohe Effizienz, da basierend auf Totalreflexion und dadurch kaum Transmission - geringe Positionstoleranz der Transceiver-Module - keine Streuung	- mechanisch empfindlich - auf Grund der feinen Strukturen und spitzen Innenwinkel schwierig zu fertigen

3 Entwicklung und Implementierung

3.1 Lichtleitkörper

In diesem Kapitel wird die Berechnung, Simulation und Fertigung des Lichtleitkörper mit ausgerichteten Spiegelflächen beschrieben.

3.1.1 Lichtleitkörpermaterial

Auf Grund des gewählten Transceiver-Moduls muss der Lichtleitkörper für infrarotes Licht mit einer Wellenlänge zwischen 880 nm und 900 nm [34, S. 5] hinreichend gute optische Eigenschaften aufweisen. Da *Plexi-* oder *Acrylglas* aus Polymethylmethacrylat je nach Additiven auch für diese Wellenlängen Lichtdurchlässig ist [31] und gute mechanische Eigenschaften aufweist, wurde dieses Material ausgewählt.

Alternativ ist auch Makrolon aus Polycarbonat geeignet, da es spezielle Varianten zur Lichtleitung gibt [18, S. 3].

3.1.2 Linsenberechnung

Zur parallelen Ausrichtung der Lichtstrahlen beim Eintreten in den Lichtleitkörper werden Linsen in die Oberfläche integriert.

Für die Berechnung wurden diese in vertikale Abschnitte mit der Höhe dH unterteilt, sodass die Linsenform iterativ von der Mitte aus berechnet werden kann. In Abbildung 3.1 ist die Geometrie zur Berechnung zu sehen.

D ist die horizontal und H die vertikale Distanz zwischen dem Fokuskreismittelpunkt und dem aktuell zu berechnenden Abschnittsanfangspunkt. Die Fokuskreise liegen auf dem Mittelpunkt der Transceiver-Modul-Linsen und geben mit ihrem Durchmesser die

Höhe der Linsen und Reflektorflächen vor. In diesem Bereich sollen die Lichtstrahlen der Transceiver-Modul-Sendediode parallel verlaufen.

Da der Winkel α an den Abschnittsanfangspunkte berechnet wird, entsteht ein Fehler, der die Lichtstrahlen auseinander laufen lässt und die Linse insgesamt etwas breiter wird. Dieser Fehler soll durch Aufteilung der Höhe in kleine Abschnitte dH minimiert werden.

Damit ist H_n durch dH vorgegeben, wodurch mit dem Abstand D_n der horizontale Abstand dx zum folgenden Abschnittsanfangspunkt berechnet wird.

Auf Grund der Achsensymmetrie der Linse ist es ausreichend, nur die obere Hälfte der Linse zu berechnen und diese um den vertikalen Mittelpunkt zu spiegeln.

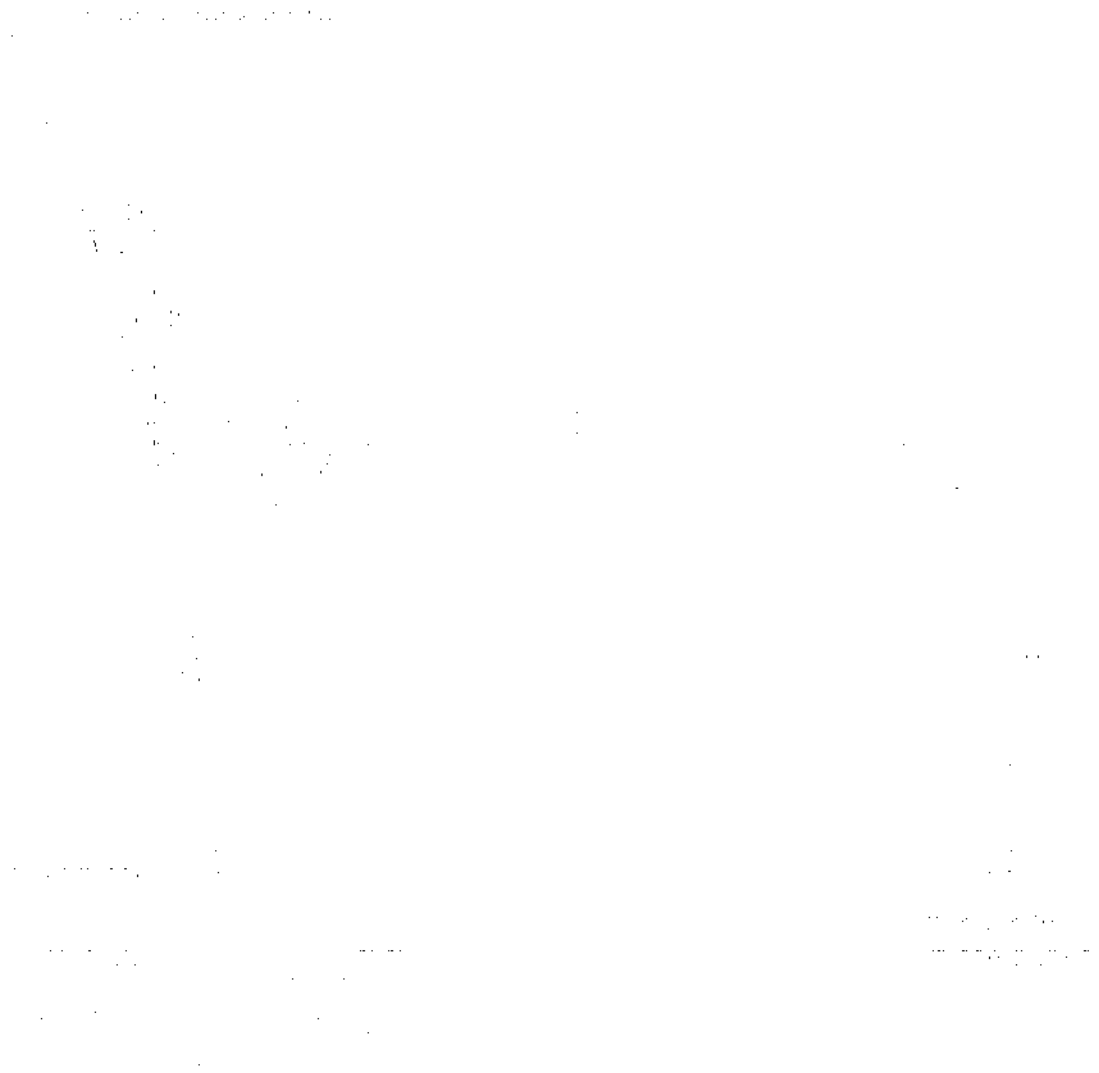


Abbildung 3.1: Geometrische Zeichnung für die Berechnung der Linsen.

Gleichungen

Die Höhe des Linsenmittelpunktes, von dem aus berechnet wird, ist auf 0 gesetzt. Somit entspricht $H_1 = dH$. Die horizontale Distanz des mittleren Punktes zum Fokuskreismitelpunkt D_1 ist durch den Abstand zwischen Lichtleitkörper D_{LLK-L} und dem Radius der Transceiver-Modul-Linsen r_L vorgegeben.

$$H_1 = dH \quad (3.1)$$

$$D_1 = D_{LLK-L} + r_L \quad (3.2)$$

Die Berechnung von H_n für jeden weiteren Punkt erfolgt nach Gleichung 3.3.

$$H_n = H_{n-1} + dH \quad (3.3)$$

Mit H_n und D_n kann γ_n nach Gleichung 3.4 und damit α_n nach Gleichung 3.6 berechnet werden. Hierfür werden die Brechungsindizes des Lichtleitkörpermateri als n_1 und der Umgebungsluft n_2 benötigt, da dies die Form der Linse beeinflusst.

$$\gamma = \arctan\left(\frac{H}{D}\right) \Rightarrow \gamma_n = \arctan\left(\frac{H_n}{D_n}\right) \quad (3.4)$$

Snelliussches Brechungsgesetz [9], [6, S. 82]:

$$\sin(\alpha) \cdot n_1 = \sin(\beta) \cdot n_2 \quad (3.5)$$

Mit $\beta = \alpha + \gamma$:

$$\begin{aligned} \sin(\alpha) \cdot n_1 &= \sin(\alpha + \gamma) \cdot n_2 && | \cdot \frac{1}{n_2} \\ \sin(\alpha) \cdot \frac{n_1}{n_2} &= \sin(\alpha + \gamma) \end{aligned}$$

Mit $\sin(\alpha + \gamma) = \sin(\alpha) \cdot \cos(\gamma) + \cos(\alpha) \cdot \sin(\gamma)$ [10, Gl. 14]:

$$\begin{aligned}
 \sin(\alpha) \cdot \frac{n_1}{n_2} &= \sin(\alpha) \cdot \cos(\gamma) + \cos(\alpha) \cdot \sin(\gamma) && | \cdot \frac{1}{\sin(\alpha)} \\
 \frac{n_1}{n_2} &= \cos(\gamma) + \cos(\alpha) \cdot \frac{\sin(\gamma)}{\sin(\alpha)} \\
 \frac{n_1}{n_2} &= \cos(\gamma) + \sin(\gamma) \cdot \frac{\cos(\alpha)}{\sin(\alpha)} \\
 \frac{n_1}{n_2} &= \cos(\gamma) + \sin(\gamma) \cdot \cot(\alpha) && | - \cos(\gamma) \\
 \frac{n_1}{n_2} - \cos(\gamma) &= \sin(\gamma) \cdot \cot(\alpha) && | \cdot \frac{1}{\sin(\gamma)} \\
 \frac{n_1}{n_2} \cdot \frac{1}{\sin(\gamma)} - \frac{\cos(\gamma)}{\sin(\gamma)} &= \cot(\alpha) \\
 \frac{n_1}{n_2} \cdot \frac{1}{\sin(\gamma)} - \cot(\gamma) &= \cot(\alpha) \\
 \operatorname{arccot}\left(\frac{n_1}{n_2} \cdot \frac{1}{\sin(\gamma)} - \cot(\gamma)\right) &= \alpha \\
 \Rightarrow \alpha_n &= \operatorname{arccot}\left(\frac{n_1}{n_2} \cdot \frac{1}{\sin(\gamma_n)} - \cot(\gamma_n)\right) \tag{3.6}
 \end{aligned}$$

Sind diese Werte bekannt, kann nach Gleichung 3.7 der folgende horizontale Abstand zum Fokuskreismittelpunkt D_{n+1} berechnet werden.

$$D_{n+1} = D_n + dx_n$$

Mit $dx_n = \tan(\alpha_n) \cdot dH$:

$$D_{n+1} = D_n + \tan(\alpha_n) \cdot dH \tag{3.7}$$

Simulation

Aus den so berechneten Koordinaten wurde eine simulationsfähige Datei für den zwei-dimensionalen Optiksimulator *Ray Optics Simulation* von ricktu288 [39] erstellt, welche das in Abbildung 3.2 zu sehende Ergebnis brachte. Zur Berechnung wurden die Brechungsindizes $n_1 = 1,49$ für *Plexiglas* aus Polymethylmethacrylat und $n_2 = 1,000292$ für Luft [32] bei einer Wellenlänge des Lichts von 589 nm eingesetzt. In der Simulation

wurde eine punktförmige Lichtquelle angenommen, welche in alle Richtungen die gleich Leuchtstärke besitzt. Dies entspricht jedoch nicht der Lichtausstrahlung der Transceiver-Modul-Sendediode, weil diese bei $\pm 24^\circ$ [34, S. 5] nur noch die halbe Leuchtstärke haben. Da die Anordnung der Transceiver-Module und somit der Sendediode so gewählt ist, dass diese mittig vor der Linse sitzen und sich die Linsen somit in der relevanten Richtung der Lichtausbreitung befinden, kann dies vernachlässigt werden.

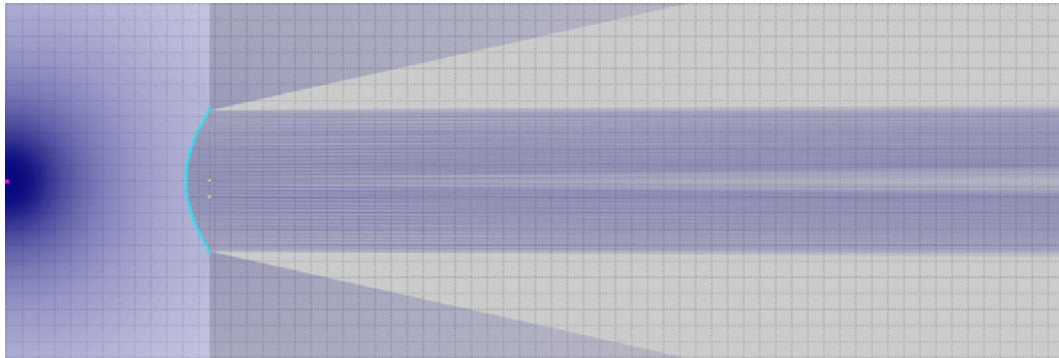


Abbildung 3.2: Simulationsergebnis der Linsen. Die türkisen Punkte zeigen die Endpunkte der Streckenabschnitte des Lichtleitkörperumrisses sowie die Position der Punktlichtquellen. Links ist eine Punktlichtquelle, deren Lichtstrahlen durch die Linse in der Lichtleitkörperoberfläche nahezu parallel ausgerichtet werden. Diese Simulation wurde mit dem zweidimensionalen Optiksimulator *Ray Optics Simulation* von ricktu288 [39] erstellt.

Zu sehen ist, dass die bei der Linse auftreffenden Lichtstrahlen im Lichtleitkörper näherungsweise parallel ausgerichtet sind. Auf Grund der Berechnungsweise laufen die Lichtstrahlen etwas auseinander, wodurch sich in der Mitte ein dunkler Bereich herausbildet.

3.1.3 Alternative Linsenberechnung

Zur Vermeidung des dunklen Bereichs in den ausgerichteten Lichtstrahlen der Linse nach der vorherigen Berechnung, wurde die Berechnung der Linsenform ebenfalls mit der Gleichung 3.8 nach Willig [3] durchgeführt.

$$z(H) = \frac{H^2}{R_0 \cdot (1 + \sqrt{1 + (\frac{H}{R_0})^2})} \quad (3.8)$$

$z(H)$ entspricht dem Abstand des aktuell berechneten Punktes bei der Höhe H und R_0 ist der Radius im Scheitel der Linse auf der optischen Achse [3] beziehungsweise der Abstand der Linse zu Fokuskreismittelpunkt. In Abbildung 3.3 ist die Geometrie zu sehen.

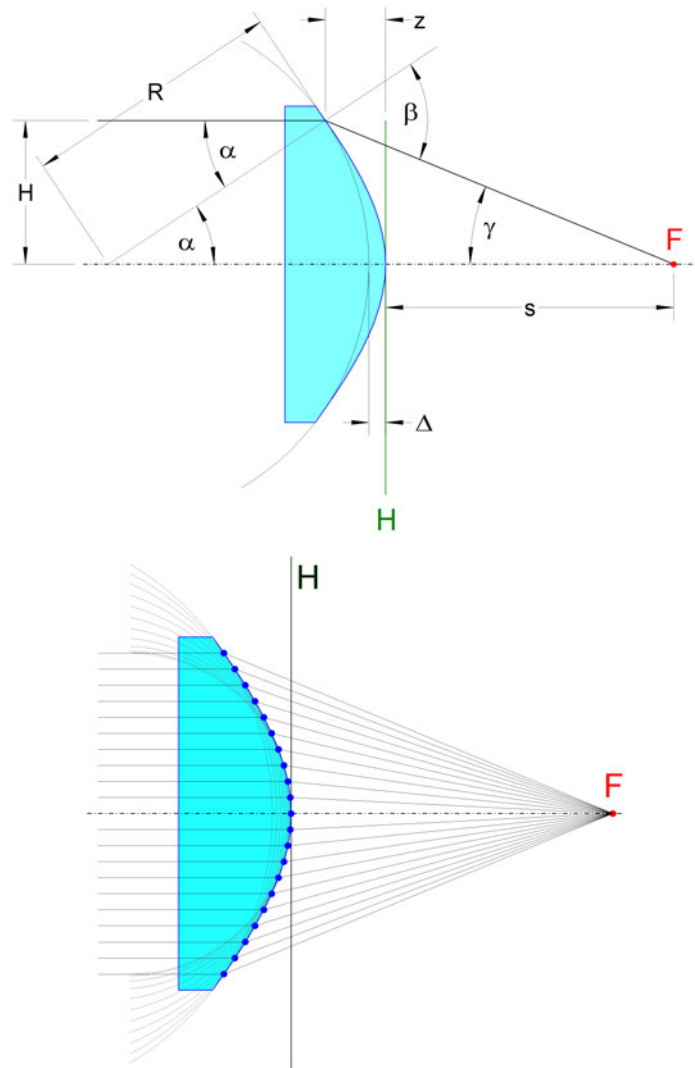


Abbildung 3.3: [3] Geometrie zur Berechnung nach Willig.

Simulation

Wird eine nach dieser Formel berechnete Linse simuliert, ist zu erkennen, dass die ausgerichteten Lichtstrahlen gleichmäßiger als mit der vorherigen verteilt sind und sich kein

dunkler Bereich in der Mitte herausbildet. Das Simulationsergebnis ist in Abbildung 3.4 dargestellt.

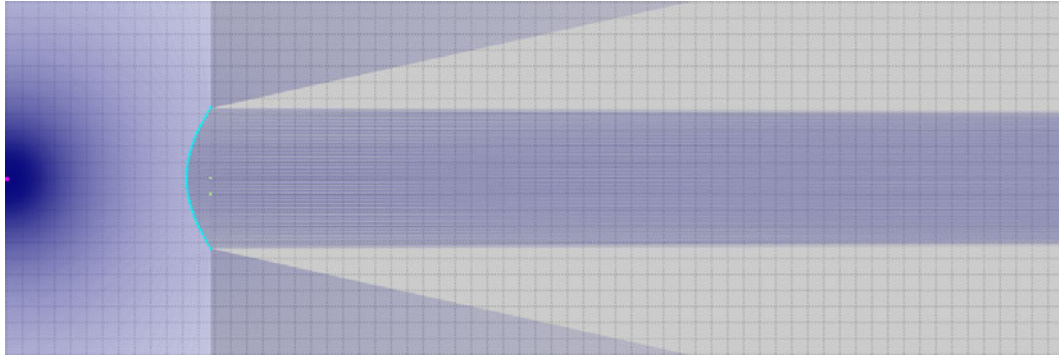


Abbildung 3.4: Simulationsergebnis der Linsen nach der alternativen Berechnung. Die türkisen Punkte zeigen die Endpunkte der Streckenabschnitte des Lichtleitkörperumrisses sowie die Position der Punktlichtquellen. Links ist eine Punktlichtquelle, deren Lichtstrahlen durch die Linse in der Lichtleitkörperoberfläche nahezu parallel ausgerichtet werden. Diese Simulation wurde mit dem zweidimensionalen Optiksimulator *Ray Optics Simulation* von ricktu288 [39] erstellt.

Zum Zeitpunkt der Formulierung der alternative Linsenberechnung war der Lichtleitkörper bereits nach der ersten Version gefertigt worden. Da die Unterschiede nicht signifikant für die Funktionalität des Demonstrationsaufbaus sind, wurde auf eine Neuanfertigung verzichtet.

3.1.4 Berechnung der Reflektorflächen

Alle Berechnungen und Angabe von Positionen erfolgen in einem zweidimensionalen Koordinatensystem und es werden die in Tabelle 3.1 aufgelisteten Abkürzungen benutzt.

Die Berechnung der Reflektorflächen erfolgt vom obersten (letzten) zum untersten (ersten) Slave-Transceiver-Modul und es werden zuerst die Reflektorendpunkte (oben) und anschließend die Reflektoranfangspunkte (unten) berechnet.

Die Slave-Transceiver-Module sind von $1 \leq n \leq N$ durchnummeriert, wobei $N = 12$ die Gesamtanzahl ist.

Weiterhin wird angenommen, dass die Lichtstrahlen einer Lichtquelle im Lichtleitkörper auf einer Breite des Fokuskreisdurchmessers parallel laufen und der Winkel zwischen der

Tabelle 3.1: Abkürzungen und deren Bedeutung für die Berechnung der Reflektorflächen

Abkürzung	Bedeutung
x_{ME}	x-Koordinate Master Empfänger
y_{ME}	y-Koordinate Master Empfänger
x_{MS}	x-Koordinate Master Sender
y_{MS}	y-Koordinate Master Sender
x_{S_nE}	x-Koordinate Slave n Empfänger
y_{S_nE}	y-Koordinate Slave n Empfänger
x_{S_nS}	x-Koordinate Slave n Sender
y_{S_nS}	y-Koordinate Slave n Sender
x_{S_nERA}	x-Koordinate Slave n EmpfängerReflektorAnfang
y_{S_nERA}	y-Koordinate Slave n EmpfängerReflektorAnfang
x_{S_nERE}	x-Koordinate Slave n EmpfängerReflektorEnde
y_{S_nERE}	y-Koordinate Slave n EmpfängerReflektorEnde
x_{S_nSRA}	x-Koordinate Slave n SenderReflektorAnfang
y_{S_nSRA}	y-Koordinate Slave n SenderReflektorAnfang
x_{S_nSRE}	x-Koordinate Slave n SenderReflektorEnde
y_{S_nSRE}	y-Koordinate Slave n SenderReflektorEnde

y-Achse und den Reflektorflächen α immer größer als 0° und kleiner als 90° ist. Wenn die Master-Transceiver-Modul-Linsen nicht weiter oben als die unterste Linse des untersten Slave-Transceiver-Moduls sitzen, ist dies durch die möglichen Positionen der Master- und Slave-Transceiver-Module gegeben.

Auf Grund der geraden Anzahl von Slave-Transceiver-Modulen lässt sich der Lichtleitkörper in zwei gleichgroße Hälften links und rechts unterteilen, auf welchen sich jeweils die Reflektorflächen für jedes zweite Slave-Transceiver-Modul befinden.

Als Referenz für die Berechnung wurde die Abbildung 3.5 erstellt. Dort sind die Fokuskreise mit Radius $L1$ des Master- (unten mittig) und des Slave-Transceiver-Moduls (mittig rechts), sowie die verbindende (mittig links) und darüber liegende (oben links) Reflektorfläche zu sehen.

Die Reflektorfläche hat die gleiche Höhe, wie der Fokuskreisdurchmesser von $2 \cdot L1$. Anhand der Distanz zwischen dem Reflektorflächenanfangspunkt und dem Fokuskreismitelpunkt des Master-Transceiver-Moduls $L2$ und $L3$ kann der Winkel der Reflektorfläche α und daraus Streckendifferenz $L8$ berechnet werden, welche zusammen mit dem Fokuskreisdurchmesser die Koordinaten des Reflektorflächenendpunktes bestimmt.

Der Endpunkt der darüber liegenden Reflektorfläche dient zur Berechnung des Versatzes der verbindenden auf der x-Achse näher zur Lichtleitkörpermitte, wodurch die verfügbare Reflektorflächenhöhe ansteigt.

Als Referenz sind in Tabelle 3.2 die Funktionen der Slave-Transceiver-Modul-Linsen abhängig von ihrer Positionierung aufgelistet.

Tabelle 3.2: Funktion der Transceiver-Modul-Linsen bei Positionierung des Transceiver-Moduls auf der linken oder rechten Seite des Lichtleitkörpers

Position	links	rechts
Funktion oben	Empfänger	Sender
Funktion unten	Sender	Empfänger

Abbildung 3.5: Geometrische Zeichnung für die Berechnung der Reflektoranfangs- und -endpunkte. In dieser Zeichnung sind nur die Punktbezeichnungen der rechten Transceiver-Module eingetragen.

Gleichungen

Für die Berechnung müssen die Länge $L1$ und die Koordinaten aller Slave- sowie die der Master-Transceiver-Modul-Linsen $x_{ME}, y_{ME}, x_{MS}; y_{MS}, x_{S_nE}, y_{S_nE}, x_{S_nS}$ und y_{S_nS} gegeben sein. Die Koordinaten können aus denen der Transceiver-Module in Kapitel 2.4.2 ermittelt werden. Die Länge $L1$ hingegen muss so gewählt werden, dass die aus der Rechnung resultierende Lichtleitkörperbreite nicht den maximal verfügbaren Raum überschreitet. Da dieser Prozess eine schrittweise Annäherung erfordert, wurde die folgende Berechnung in einem Tabellendokument mit der Software *Excel* der Firma Microsoft [26] automatisiert.

Zur Vereinfachung der Berechnungen nehmen alle Reflektoren auf der y-Achse eine Höhe von $2 \cdot c \cdot L1$ ein und haben daher, abhängig von ihrem Winkel α_n , unterschiedliche Längen. Die y-Koordinaten der Reflektoranfangs- und -endpunkte können somit aus denen der Slaves nach den Gleichungen aus Tabelle 3.3 berechnet werden.

Tabelle 3.3: Berechnung der y-Koordinaten der Reflektoranfangs- und -endpunkte

Empfänger:	$y_{S_nERA} = y_{S_nE} - L1$
	$y_{S_nERE} = y_{S_nE} + L1$
Sender:	$y_{S_nSRA} = y_{S_nS} - L1$
	$y_{S_nSRE} = y_{S_nS} + L1$

Anschließend wird die x-Koordinate des Endpunkts vom aktuell zu berechnenden Reflektor ermittelt. Diese Berechnung entfällt für den obersten Punkt auf der jeweiligen Seite wie in Tabelle 3.4, da die Reflektoren von der Mitte aus nach außen wandern.

Tabelle 3.4: x-Koordinate der obersten Reflektorendpunkte auf beiden Seiten

Links:	$x_{S_NERE} = 0$
Rechts:	$x_{S_{N-1}SRE} = 0$

Für alle anderen Reflektorflächen wird der Endpunkt vom Startpunkt des auf der gleichen Seite darüber liegenden Reflektors wie in Gleichung 3.9 beziehungsweise 3.10 berechnet.

$$\tan(\iota_{S_n}) = \frac{L6_{S_n}}{L7_{S_n}} \Rightarrow L6_{S_n} = \tan(\iota_{S_n}) \cdot L7_{S_n}$$

Mit $\iota_{S_n} = \beta_{S_{n+2}}$:

$$L6_{S_{n+2}E} = \tan(\beta_{S_{n+2}E}) \cdot L7_{S_{n+2}E} \quad (3.9)$$

$$L6_{S_{n+2}S} = \tan(\beta_{S_{n+2}S}) \cdot L7_{S_{n+2}S} \quad (3.10)$$

Die äußersten reflektierten Lichtstrahlen für den aktuell zu berechnenden (innersten) Reflektorendpunkt können entweder von den folgenden (darüber liegenden) zwei Reflektorflächen auf der gleichen oder den letzten (obersten) zwei auf der gegenüberliegenden Seite stammen. Daher muss die Länge $L6$ für all diese Reflektoren berechnet werden. Zwei auf jeder Seite zur Berücksichtigung beider Kommunikationsrichtungen. Für die Berechnung von $L6$ ist $L7$ erforderlich, was nach den Gleichungen in Tabelle 3.5 berechnet wird.

Tabelle 3.5: Formeln zur Berechnung der unterschiedlichen Längen $L7$ für einen Reflektorflächenendpunkt

Empfänger links (oben)	$L7_{S_{n+2}ERA} = y_{S_{n+2}ERA} - y_{S_nERE}$
	$L7_{S_{n+2}SRA} = y_{S_{n+2}SRA} - y_{S_nERE}$
	$L7_{S_{N-1}ERE} = y_{S_{N-1}ERE} - y_{S_nERE}$
	$L7_{S_{N-1}SRE} = y_{S_{N-1}SRE} - y_{S_nERE}$
Sender links (unten)	$L7_{S_nERA} = y_{S_nERA} - y_{S_nSRE}$
	$L7_{S_{n+2}SRA} = y_{S_{n+2}SRA} - y_{S_nSRE}$
	$L7_{S_{N-1}ERE} = y_{S_{N-1}ERE} - y_{S_nSRE}$
	$L7_{S_{N-1}SRE} = y_{S_{N-1}SRE} - y_{S_nSRE}$
Empfänger rechts (unten)	$L7_{S_nSRA} = y_{S_nSRA} - y_{S_nERE}$
	$L7_{S_{n+2}ERA} = y_{S_{n+2}ERA} - y_{S_nERE}$
	$L7_{S_NERE} = y_{S_nERE} - y_{S_nERE}$
	$L7_{S_NSRE} = y_{S_nSRE} - y_{S_nERE}$
Sender rechts (oben)	$L7_{S_{n+2}ERA} = y_{S_{n+2}ERA} - y_{S_nSRE}$
	$L7_{S_{n+2}SRA} = y_{S_{n+2}SRA} - y_{S_nSRE}$
	$L7_{S_NERE} = y_{S_nERE} - y_{S_nSRE}$
	$L7_{S_NSRE} = y_{S_nSRE} - y_{S_nSRE}$

Entsprechend angepasst, ergeben sich die Gleichungen für $L6$ in Tabelle 3.6.

Tabelle 3.6: Formeln zur Berechnung der unterschiedlichen Längen $L6$ für einen Reflektorflächenendpunkt

Empfänger links	$L6_{S_{n+2}E} = \tan(\beta_{S_{n+2}E}) \cdot L7_{S_{n+2}ERA}$
	$L6_{S_{n+2}S} = \tan(\beta_{S_{n+2}S}) \cdot L7_{S_{n+2}SRA}$
	$L6_{S_{N-1}E} = \tan(\beta_{S_{N-1}E}) \cdot L7_{S_{N-1}ERE}$
	$L6_{S_{N-1}S} = \tan(\beta_{S_{N-1}S}) \cdot L7_{S_{N-1}SRE}$
Sender links	$L6_{S_nE} = \tan(\beta_{S_nE}) \cdot L7_{S_nERA}$
	$L6_{S_{n+2}S} = \tan(\beta_{S_{n+2}S}) \cdot L7_{S_{n+2}SRA}$
	$L6_{S_{N-1}E} = \tan(\beta_{S_{N-1}E}) \cdot L7_{S_{N-1}ERE}$
	$L6_{S_{N-1}S} = \tan(\beta_{S_{N-1}S}) \cdot L7_{S_{N-1}SRE}$
Empfänger rechts	$L6_{S_nSRA} = \tan(\beta_{S_nS}) \cdot L7_{S_nSRA}$
	$L6_{S_{n+2}ERA} = \tan(\beta_{S_{n+2}E}) \cdot L7_{S_{n+2}ERA}$
	$L6_{S_NERE} = \tan(\beta_{S_NE}) \cdot L7_{S_NERE}$
	$L6_{S_NSRE} = \tan(\beta_{S_NS}) \cdot L7_{S_NSRE}$
Sender rechts	$L6_{S_{n+2}ERA} = \tan(\beta_{S_{n+2}E}) \cdot L7_{S_{n+2}ERA}$
	$L6_{S_{n+2}SRA} = \tan(\beta_{S_{n+2}S}) \cdot L7_{S_{n+2}SRA}$
	$L6_{S_NERE} = \tan(\beta_{S_NE}) \cdot L7_{S_NERE}$
	$L6_{S_NSRE} = \tan(\beta_{S_NS}) \cdot L7_{S_NSRE}$

Zur Bestimmung der x-Koordinate des Reflektorflächenendpunktes wird geprüft, wie nahe die reflektierten Lichtstrahlen der vorherigen Sender- und Empfänger-Reflektorflächen auf Höhe der y-Koordinate des aktuellen Reflektorflächenendpunktes an der x-Koordinate 0 vorbei laufen.

Dafür wird die resultierende x-Koordinate des Reflektorflächenendpunktes mit allen unterschiedlichen Längen $L6$ betragsmäßig berechnet und der Höchste aus diesen Werten und 0 als x-Koordinate verwendet. Dies stellt sicher, dass der Reflektor ausreichend weit außen platziert ist. Das Vorzeichen wird anhand der Transceiver-Modul-Position ermittelt und eingesetzt. Die Gleichungen zur Berechnung sind in Tabelle 3.7 zu finden.

Wie oben beschrieben, entfällt diese Berücksichtigung bei den obersten (letzten) Reflektorflächen der jeweiligen Seite, da keine weiteren Reflektorflächen darüber liegen.

Tabelle 3.7: Formeln zur Berechnung der x-Koordinate eines Reflektorflächenendpunktes

Empfänger links (oben)	$x_{S_n ERE} = \text{MAX}(\begin{aligned} &0; \\ & x_{S_{n+2} ERA} - L6_{S_{n+2} ERA}; \\ & x_{S_{n+2} SRA} - L6_{S_{n+2} SRA}; \\ &L6_{S_{N-1} ERE} - x_{S_{N-1} ERE} ; \\ &L6_{S_{N-1} SRE} - x_{S_{N-1} SRE} \end{aligned}) \cdot \text{WENN}(0 < x_{S_n E}; -1; 1)$
Sender links (unten)	$x_{S_n SRE} = \text{MAX}(\begin{aligned} &0; \\ & x_{S_n ERA} - L6_{S_n ERA}; \\ & x_{S_{n+2} SRA} - L6_{S_{n+2} SRA}; \\ &L6_{S_{N-1} ERE} - x_{S_{N-1} ERE} ; \\ &L6_{S_{N-1} SRE} - x_{S_{N-1} SRE} \end{aligned}) \cdot \text{WENN}(0 < x_{S_n S}; -1; 1)$
Empfänger rechts (unten)	$x_{S_n ERE} = \text{MAX}(\begin{aligned} &0; \\ & x_{S_n SRA} - L6_{S_n SRA}; \\ & x_{S_{n+2} ERA} - L6_{S_{n+2} ERA}; \\ &L6_{S_N ERE} - x_{S_n ERE} ; \\ &L6_{S_N SRE} - x_{S_n SRE} \end{aligned}) \cdot \text{WENN}(0 < x_{S_n E}; -1; 1)$
Sender rechts (oben)	$x_{S_n SRE} = \text{MAX}(\begin{aligned} &0; \\ & x_{S_{n+2} ERA} - L6_{S_{n+2} ERA}; \\ & x_{S_{n+2} SRA} - L6_{S_{n+2} SRA}; \\ &L6_{S_N ERE} - x_{S_n ERE} ; \\ &L6_{S_N SRE} - x_{S_n SRE} \end{aligned}) \cdot \text{WENN}(0 < x_{S_n S}; -1; 1)$

Damit können die Abstände zwischen Fokuskreismittel- und Reflektorendpunkt nach den Gleichungen in den Tabellen 3.8 und 3.9 berechnet werden.

Tabelle 3.8: Formeln zur Berechnung der Länge $L2$ eines Reflektorflächenanfangspunktes

Empfänger links (oben)	$L2 = x_{S_nERE} - x_{MS}$
Sender links (unten)	$L2 = x_{S_nSRE} - x_{ME}$
Empfänger rechts (unten)	$L2 = -(x_{S_nERE} - x_{MS})$
Sender rechts (oben)	$L2 = -(x_{S_nSRE} - x_{ME})$

Tabelle 3.9: Formeln zur Berechnung der Länge $L3$ eines Reflektorflächenanfangspunktes

Empfänger	$L3 = y_{S_nERE} - y_{MS}$
Sender	$L3 = y_{S_nSRE} - y_{ME}$

Daraus wiederum lassen sich die Länge $L5$ nach Gleichung 3.11 sowie die Winkel β und γ nach Gleichung 3.12 beziehungsweise 3.13 berechnen.

$$L5 = \sqrt{L4^2 - L1^2}$$

Mit: $L4 = \sqrt{L2^2 + L3^2}$:

$$L5 = \sqrt{L2^2 + L3^2 - L1^2} \quad (3.11)$$

$$\tan(\beta) = \frac{L2}{L3} \Rightarrow \beta = \arctan\left(\frac{L2}{L3}\right) \quad (3.12)$$

$$\tan(\gamma) = \frac{L1}{L5} \Rightarrow \gamma = \arctan\left(\frac{L1}{L5}\right) \quad (3.13)$$

Mit diesen Werten lässt sich der Winkel α nach Gleichung 3.15, damit die Länge $L8$ nach Gleichung 3.15 und letztendlich die x-Koordinate des Reflektorflächenanfangspunktes nach der zutreffenden Gleichung aus Tabelle 3.10 berechnen.

$$\alpha = \frac{90^\circ - \beta - \gamma}{2} \quad (3.14)$$

$$\tan(\alpha) = \frac{L8}{2 \cdot L1} \Rightarrow L8 = \tan(\alpha) \cdot 2 \cdot L1 \quad (3.15)$$

Tabelle 3.10: Formeln zur Berechnung der x-Koordinate eines Reflektorflächenanfangspunktes

Empfänger links (oben)	$x_{S_n ERA} = x_{S_n ERE} + L8$
Sender links (unten)	$x_{S_n SRA} = x_{S_n SRE} + L8$
Empfänger rechts (unten)	$x_{S_n ERA} = x_{S_n ERE} - L8$
Sender rechts (oben)	$x_{S_n SRA} = x_{S_n SRE} - L8$

Wie oben schon erwähnt, wurde diese Berechnung für alle zwölf Slave-Transceiver-Module in einem Tabellendokument automatisiert und die Länge $L1$ manuell nach dem Wägevorgang so lange angepasst, bis die Breite des Lichtleitkörpers dem in Kapitel 2.4.2 definierten Wert von 25,6 mm abzüglich der Linsenbreite beider Seiten entsprach.

3.1.5 Simulation und Fertigung des Lichtleitkörpers

Zur Simulation des im Tabellendokument berechneten Lichtleitkörpers wurde dieses so erweitert, dass der Inhalt für eine simulationsfähige Datei des zweidimensionalen Optiksimsulators *Ray Optics Simulation* von ricktu288 [39] in Form von American Standard Code for Information-Interchange (ASCII) Zeichen erstellt wird. Der Inhalt mehrerer Zellen wird kopiert und in der Simulationsdatei zusammengeführt. Das Ergebnis ist in Abbildung 3.6 zu sehen.

Wie bei der Simulation der Linsen, wurde aus Aufwandsgründen eine punktförmige Lichtquelle mit der gleichen Leuchtstärke alle Richtungen eingesetzt, was nicht der Lichtverteilungskurve der Transceiver-Modul-Sendedioden entspricht, welche bei $\pm 24^\circ$ nur noch die halbe Leuchtstärke haben [34, S. 5]. Auf Grund der mittigen Anordnung der Slave-Transceiver-Modul-Linsen vor den Linsen des Lichtleitkörpers, kann dies vernachlässigt werden.

Der Öffnungswinkel dieser punktförmigen Lichtquellen wurde bei den Slave-Transceiver-Modulen auf den mit der halbe Leuchtstärke von auf 48° begrenzt [34, S. 5]. Bei der Master-Transceiver-Modul-Sendediode wurde dieser auf 96° verdoppelt, damit die Lichtstrahlen auch die Reflektorflächen der nächsten Slave-Transceiver erreichen. Ob die dort eintreffende Leuchtstärke ausreicht, wird sich in praktischen Tests zeigen.

In der Abbildung 3.6a ist zu erkennen, dass alle gebündelten und reflektierten Lichtstrahlen der Slave-Transceiver-Modul-Sendedioden in Richtung des Master-Transceiver-Modul-Empfängers verlaufen.

Weiterhin ist die Transmission der Lichtstrahlen an den ersten fünf Reflektorflächen zu sehen. Auf Grund der kurzen Distanz zum Master-Transceiver-Empfänger könnte die eintreffende Leuchtstärke der reflektierten Lichtstrahlen ausreichen. Ab dem sechsten Slave-Transceiver gibt es keine Transmission an der Reflektorfläche. Demnach tritt hier Totalreflexion auf.

Betrachtet man nur die Lichtstrahlen der Master-Transceiver-Sendediode in Abbildung 3.6b, ist zu erkennen, dass bei allen Slave-Transceiver-Modul-Empfängern die eingehenden Lichtstrahlen von den eingesetzten Linsen auf den Empfängerpunkt gebündelt werden.

In Abbildung 3.6d ist der dunkle Bereich in der Mitte der gebündelten Lichtstrahlen durch die Berechnungsweise der Linsen zu erkennen.

Zuletzt ist in der Abbildung 3.6c zu erkennen, dass die reflektierten Lichtstrahlen der Slave-Transceiver-Modul-Sendediode beim Medienübergang vor dem Master-Transceiver-Modul-Empfänger umgelenkt werden und diesen somit nicht direkt treffen. Da dieser jedoch über eine eigene Linse verfügt, in welche die Lichtstrahlen eindringen, wurde der Entwurf so belassen. Diese Ablenkung reduziert sich mit steigendem Abstand der Reflektorflächen, da die Lichtstrahlen in einem steileren Winkel auf den Medienübergang treffen.

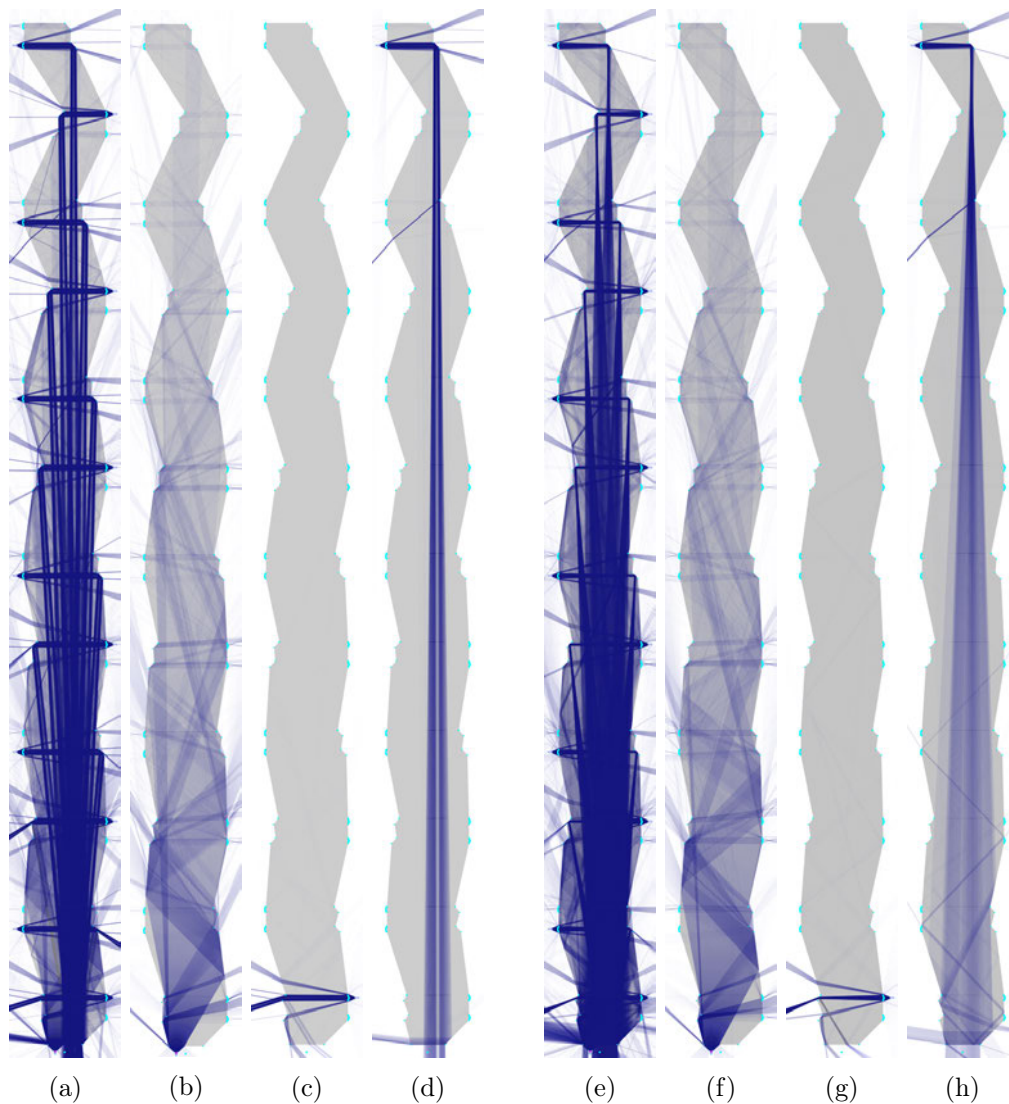


Abbildung 3.6: Simulationsergebnis des berechneten Lichtleitkörpers. Die türkisen Punkte zeigen die Endpunkte der Streckenabschnitte des Lichtleitkörperumrisses sowie die Position der Punktlichtquellen. In den Bildern a bis d ist der Brechungsindex des Lichtleitkörpers auf den von *Plexiglas* mit 1,49 [32] und e bis h auf den von *Makrolon* 1,587 [18, S. 26] gesetzt. In a und e sind alle Transceiver-Modul-Sendediode aktiv, in b und f die des Masters, in c und g die des ersten Slaves und in d und h die des letzten. Diese Simulation wurde mit dem zweidimensionalen Optiksimulator *Ray Optics Simulation* von ricktu288 [39] erstellt.

Fertigung

Der Lichtleitkörper wurde im Wasserstrahlverfahren gefertigt, bei dem eine Düse computergesteuert in zwei Dimensionen verfährt und der unter hohem Druck austretende und mit schleifenden Partikeln versetzte Wasserstrahl das Material abträgt.

Dazu musste eine Datei im Drawing Interchange File Format (DXF) angefertigt werden, wofür das Tabellendokument entsprechend erweitert wurde. Wegen der Zeichenlimitierung pro Zelle durch die Software *Excel* auf maximal 32767 wurde der Dateinhalt auf mehrere Zellen verteilt und musste nachträglich in einer Datei zusammengefügt werden.

Da der so generierte Umriss des Lichtleitkörpers keine Befestigungspunkte besitzt, wurden diese in der erstellten DXF-Datei mit der Software *LibreCAD* [27] manuell ergänzt. Diese sollten konzeptbedingt keinen Einfluss auf die Übertragung nehmen, da diese entfernt von den Reflektorflächen und Linsen in die Außenflächen integriert wurden. Für andere Lichtleitkörperkonzepte müssen möglicherweise getrennte Halter angefertigt werden, um kein Licht in den Haltern zu transmittieren. Der somit entstandene Umriss des Lichtleitkörpers ist in Abbildung 3.7 zu sehen.

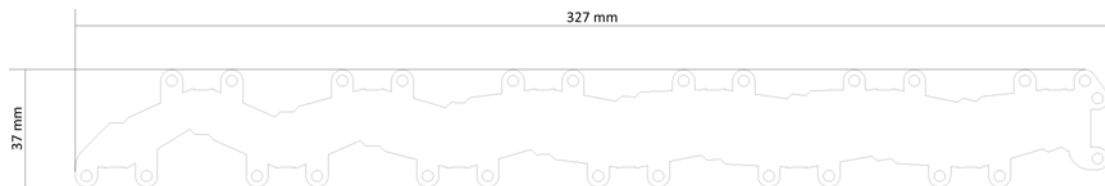


Abbildung 3.7: Umriss des Lichtleitkörpers aus der DXF-Datei.

Anhand des gefertigten Teils wurde in Kapitel 4.2.2 eine Abweichung von $+0,05\text{ mm}$ bis $+0,30\text{ mm}$ zum Sollumriss gemessen. Die Schnittfläche ist rau und mit etwa $2,5^\circ$ Neigung nicht senkrecht. Glättungsversuche der Oberfläche an einem Verschnittteil durch Erwärmung mit einem Heißluftföhn und einer offenen Flamme erzeugten Blasen im Material, bevor die Oberfläche glatt wurde. Abschleifen mit Sandpapier von Hand und Politurpaste auf einen Rotationswerkzeugaufsatz brachten ebenfalls keinen Erfolg. Daher wurde der Lichtleitkörper mit seiner rauen Oberfläche belassen.

Weiterhin wurde als Grundmaterial *Makrolon* genutzt, dessen Brechungsindex zwischen 1,584 und 1,587 [18, S. 25–26] liegt und damit höher als der für die Linsenberechnung

angenommene Wert von 1,49 ist. Die Auswirkungen werden nachfolgend anhand einer Simulation untersucht.

Angepasste Simulation des Lichtleitkörpers für das Material Makrolon

Zur Überprüfung des Einflusses des höheren Brechungsindex vom Grundmaterial *Makrolon*, wurde dieser in der Simulationsdatei auf den höchsten im Datenblatt aufgeführten Wert von 1,587 [18, S. 26] angepasst und erneut simuliert. Das Ergebnis ist in Abbildung 3.6 zu sehen.

In Abbildung 3.6g und 3.6h ist erkennbar, dass die Lichtstrahlen durch den höheren Brechungsindex zusammen laufen. Nachdem sich die Lichtstrahlen gekreuzt haben, laufen diese mit steigendem Abstand zwischen dem Slave- und dem Master-Transceiver-Modul immer weiter auseinander, was für eine Reduktion der Leuchtstärke beim Master-Transceiver-Modul-Empfänger sorgt.

Bei Betrachtung der Abbildung 3.6e ist erkennbar, dass es durch den höheren Brechungsindex des Lichtleitkörpermaterials bereits beim vierten Slave-Transceiver keine Transmission an der Reflektorfläche mehr auftritt.

Durch den höheren Brechungsindex ist allerdings die Ablenkung der Lichtstrahlen der ersten Slave-Transceiver-Sendediode am Medienübergang vor dem Master-Transceiver-Empfänger größer, was in Abbildung 3.6g zu beobachten ist.

Bei den Tests aus Kapitel 4.1.2 zeigte sich, dass der gefertigte Lichtleitkörper funktionsfähig ist. Daher wurde die Fertigung eines verbesserten Lichtleitkörpers späteren Arbeiten vorbehalten.

3.2 Hardware

In diesem Kapitel wird die Auslegung und Konstruktion des Demonstrationsaufbaus auf Grundlage der in Tabelle 2.1 aus Kapitel 2.2 aufgelisteten Komponenten beschrieben. Zur Realisierung und Funktionserweiterung wurden im Verlaufe der Entwicklung weitere Komponenten ergänzt, welche in Tabelle 2.2 zu finden sind.

3.2.1 Verfügbarer Strom für externe Komponenten

Die Sendedioden der Transceiver-Module, die Bedienelemente und der LCD (nur beim BMS) sollen über die Linearspannungsregler auf den Mikrocontrollerplatinen versorgt werden. Der Sendediodenstrom wird durch den integrierten Stromregler auf maximal 350 mA [34, S. 5] begrenzt. Zur Verringerung dieses Stromes kann ein Vorwiderstand an den Pin V_{CC2} angeschlossen werden [34, S. 6], was erforderlich ist, da die Linearspannungsregler *IFX54211MB V33* [14, S. 14] der Infineon Technologies AG auf den Mikrocontrollerplatinen *XMC 2Go XMC1100 V1* nur einen Strom von maximal $I_{LVRMAX} = 150 \text{ mA}$ [15, S. 2] [14, S. 9] bereitstellen können. Der Linearspannungsregler *IFX1117MEV33* [17, S. 24] der Infineon Technologies AG des *XMC4700 Relax Kit V1* kann einen Strom von bis zu $I_{LVRMAX} = 1 \text{ A}$ [13, S. 1] liefern. Da die Linearspannungsregler beider Mikrocontrollerplatinen alle Komponenten auf diesen versorgen, steht nur ein begrenzter Strom für angeschlossene Komponenten zur Verfügung.

LED Ströme

Auf beiden Mikrocontrollerplatinen befinden sich die gleichen grünen und roten LED [14, S. 14] [17, S. 24], deren Strom I_{LED} mit dem ohmschen Gesetz 3.16 und den Messwerten der Spannung U_{RLED} und des Widerstandswertes R_{LED} der jeweiligen Vorwiderstände ermittelt wurde. Alle Messungen wurden mit einem Unitec 45647 Multimeter an einer *XMC 2Go XMC1100 V1* Mikrocontrollerplatine durchgeführt. Die Messwerte sowie Ergebnisse befinden sich in Tabelle 3.11.

$$I_{LED} = \frac{U_{RLED}}{R_{LED}} \quad (3.16)$$

Tabelle 3.11: Messwerte zur Berechnung der LED Ströme und die Ergebnisse

grüne LED	rote LED
$U_{RLEDgn} = 1,370 \text{ V}$	$U_{RLEDgn} = 1,477 \text{ V}$
$R_{LEDgn} = 680 \Omega$	$R_{LEDgn} = 677 \Omega$
$I_{LEDgn} = 2,015 \text{ mA} \approx 2,1 \text{ mA}$	$I_{LEDrt} = 2,182 \text{ mA} \approx 2,2 \text{ mA}$

Strom für externe Komponenten

Zur Berechnung des maximalen für externe Komponenten verfügbaren Stromes I_{ExtMAX} wird die Differenz aus dem maximalen Strom der Linearspannungsregler I_{LVRMAX} und der Summe aus den typischen Strömen Mikrocontrollerplatinen I_{BrdTyp} , den typischen Strömen der Mikrocontrollerperipherie und den LED Strömen gebildet. Der Strom der Mikrocontrollerperipherie wird hier durch die Differenz aus der Mikrocontrollerstromaufnahme mit aktivierter $I_{DDPAEMAX}$ (Maximalewert) und mit deaktivierter Peripherie I_{DDPAD} berechnet. Es wird davon ausgegangen, dass die Ströme bei den vorliegenden Betriebsbedingungen nicht höher als angegeben sind.

Für die *XMC 2Go XMC1100 V1* gilt [19, S. 41]:

$$\begin{aligned} I_{ExtMAX} &\leq I_{LVRMAX} - (I_{BrdTyp} + (I_{DDPAEMAX} - I_{DDPAD}) + 1 \cdot I_{LEDgrn} + 2 \cdot I_{LEDrot}) \\ &\leq (150 - (75 + (11 - 4, 7) + 1 \cdot 2, 1 + 2 \cdot 2, 2)) \cdot mA = 62, 2 mA \end{aligned} \quad (3.17)$$

Für das *XMC4700 Relax Kit V1* gilt [22, S. 71]:

$$\begin{aligned} I_{ExtMAX} &\leq I_{LVRMAX} - (I_{BrdTyp} + (I_{DDPAE} - I_{DDPAD}) + 3 \cdot I_{LEDgrn} + 2 \cdot I_{LEDrot}) \\ &\leq (1000 - (250 + (135 - 86) + 3 \cdot 2, 1 + 2 \cdot 2, 2)) \cdot mA = 690, 3 mA \end{aligned} \quad (3.18)$$

Maximale Umgebungstemperatur

Der Linearspannungsregler des Mikrocontrollerboards *XMC 2Go XMC1100 V1* kann demnach etwa 62 mA und der des *XMC4700 Relax Kit V1* etwa 690 mA für externe Komponenten zur Verfügung stellen. Voraussetzung hierfür ist, dass die Sperrschichttemperatur T_{JMAX} im Bereich von $-40 \dots 125^\circ\text{C}$ bzw. $0 \dots 125^\circ\text{C}$ bleibt. [15, S. 7] [13, S. 5] Zur Prüfung der Einhaltung der Maximaltemperatur kann mit dem thermischen Widerstand R_{thJA} und der maximalen Linearspannungsreglerleistung P_{LVRMAX} die Temperaturdifferenz dT_{JA} zwischen Sperrschicht- und Umgebungstemperatur und somit die maximale Umgebungstemperatur T_{AmbMAX} bei gegebenen Maximalstrom I_{LVRMAX} berechnet werden.

Bei dem *XMC 2Go XMC1100 V1* muss der höchste angegebene thermische Widerstandswert verwendet werden, da der Linearspannungsregler dort nicht über die Platine und durch keinen aktiven Luftstrom gekühlt wird. Der des *XMC4700 Relax Kit V1* hingegen wird über eine insgesamt mindestens 250 mm² große Kupferfläche gekühlt, weswegen der niedrigere Wert zur Berechnung verwendet wurde.

$$\begin{aligned} P_{LVRMAX} &= U_{LVR} \cdot I_{LinRegMax} \\ &= (U_{LVRIIn} - U_{LVROut}) \cdot I_{LVRMax} \end{aligned} \quad (3.19)$$

$$dT_{JA} = R_{thJA} \cdot P_{LVRMAX} \quad (3.20)$$

$$T_{AmbMAX} \leq T_{JMAX} - dT_{JA} \quad (3.21)$$

Tabelle 3.12: Berechnungswerte und -ergebnisse der maximalen Umgebungstemperatur bei maximalem Strom der Linearspannungsregler

XMC 2Go XMC1100 V1	XMC4700 Relax Kit V1
$U_{LVRIIn} = 4,75 V$	$U_{LVRIIn} = 4,63 V$
$U_{LVROut} = 3,32 V$	$U_{LVROut} = 3,29 V$
$I_{LinRegMax} = 150 mA$	$I_{LinRegMax} = 1 A$
$P_{LVRMAX} = 214,5 mW$	$P_{LVRMAX} = 1,34 W$
$R_{thJA} = 217 K/W$	$R_{thJA} = 81 K/W$
$dT_{JA} \approx 46,547 K$	$dT_{JA} = 108,54 K$
$T_{JMAX} = 125 ^\circ C$	$T_{JMAX} = 125 ^\circ C$
$T_{AmbMAX} \leq 78,453 ^\circ C$	$T_{AmbMAX} \leq 16,46 ^\circ C$

Mit etwa 78°C liegt die maximale Umgebungstemperatur für den *XMC 2Go XMC1100 V1* weit über der zu erwartenden, womit der Linearspannungsregler den oben berechneten Strom ohne zusätzliche Kühlung den externen Komponenten zur Verfügung stellen kann. Beim *XMC4700 Relax Kit V1* hingegen darf die Umgebungstemperatur maximal etwa 16°C betragen, wenn der maximale Strom fließt. Daher wurde der maximale Strom für externe Komponenten $I_{ExtMAX25}$ für eine übliche Raumtemperatur von $T_{Amb25} = 25 ^\circ C$, wie in Gleichung 3.25 zu sehen, berechnet.

$$T_{Amb25} \leq T_{JMAX} - dT_{JA25} \Rightarrow dT_{JA25} \leq T_{JMAX} - T_{Amb25} \quad (3.22)$$

$$dT_{JA25} = R_{thJA} \cdot P_{LVR25} \Rightarrow P_{LVR25} = \frac{dT_{JA}}{R_{thJA}} \quad (3.23)$$

$$P_{LVR25} = U_{LVR} \cdot I_{LinReg25} \Rightarrow I_{LinReg25} \leq \frac{P_{LVR25}}{U_{LVR}}$$

Mit Gleichung 3.23 und 3.22:

$$I_{LinReg25} \leq \frac{T_{JMAX} - T_{Amb25}}{R_{thJA} \cdot U_{LVR}} \leq \frac{125^\circ C - 25^\circ C}{81 K/W \cdot 1,34 V} = 0,921 A \quad (3.24)$$

$$\begin{aligned} I_{ExtMAX25} &\leq I_{ExtMAX} - (I_{LinRegMAX} - I_{LinReg25}) \\ &\leq 690,3 mA - (1 A - 0,921 A) = 611,3 mA \end{aligned} \quad (3.25)$$

Somit bleibt beim *XMC4700 Relax Kit V1* ein Strom von etwa 611 mA für extern angeschlossene Komponenten, wenn die Umgebungstemperatur von 25 °C eingehalten wird.

Dimensionierung des Vorwiderstands der Transceiver-Modul-Sendediode

Die Mikrocontrollerplatinen *XMC 2Go XMC1100 V1* geben den maximal zur Verfügung stehenden Strom mit $I_{ExtMAX} \leq 62,2 mA$ vor, da der des *XMC4700 Relax Kit V1* deutlich höher ist. Für eine Sicherheit von 25 % liegt der für die Sendediode der Transceiver-Module verfügbare Strom bei $I_{TRXMAX} \leq 46,65 mA$. Diese Sicherheit soll die Fertigungstoleranzen der Sendedioden und Vorwiderstände ausgleichen sowie ausreichend Strom für die Transceiver-Modul-Elektronik und die Bedienelemente lassen.

Den Werten aus Tabelle 4.2 nach, ist für einen Sendediodenstrom im Bereich $40,0\Omega \leq I_{IRED} \leq 45,0\Omega$ ein Vorwiderstand von $43,6\Omega \leq R_{IRED} \leq 49,3\Omega$ erforderlich. Daher wurde ein Vorwiderstandswert von $R_{IRED} = 47\Omega$ ausgewählt, sodass der Sendediodenstrom unter $I_{IREDMAX} \leq 45mA$ bleiben sollte. Mit diesen Werten beträgt die umgesetzte Leistung im Vorwiderstand $P_{R_{IRED}} = I_{IREDMAX}^2 \cdot R_{IRED} \leq (45 \cdot 10^{-3} A)^2 \cdot 47\Omega = 95,175mW$. Die durchschnittliche Stromaufnahme der Transceiver-Modul-Elektronik beim Senden an Anschlusspin 6 (V_{CC1}) wird mit maximal $I_{VCC1MAX} \leq 2,5mA$ [34, S. 4] angegeben.

Für andere extern angeschlossene Komponenten, wie zum Beispiel die Bedienelemente, bleibt bei der entsprechenden Mikrocontrollerplatine nach Gleichung 3.26 noch der Strom in Tabelle 3.13. Der LCD wird durch den USB-Micro-B-Anschluss der Mikrocontrollerplatine *XMC4700 Relax Kit V1* versorgt [16] und belastet den Linearspannungsregler daher nicht.

$$I_{ExtIO} \leq I_{ExtMAX} - I_{IREDMAX} - I_{VCC1MAX} \quad (3.26)$$

Tabelle 3.13: Berechnungswerte und -ergebnisse des maximalen, für Bedienelemente zur Verfügung stehenden Stroms der Mikrocontrollerplatinen

XMC 2Go XMC1100 V1	XMC4700 Relax Kit V1
$I_{ExtMAX} = 62,2mA$	$I_{ExtMAX} = 611,3mA$
$I_{IREDMAX} = 45,0mA$	$I_{IREDMAX} = 45,0mA$
$I_{VCC1MAX} = 2,5mA$	$I_{VCC1MAX} = 2,5mA$
$I_{ExtIO} = 14,7mA$	$I_{ExtIO} = 563,8mA$

3.2.2 Messmodul

Die Messmodule setzen sich jeweils aus einem Mikrocontroller, Bedienelementen und einem optischen Transceiver-Modul zusammen. Als Mikrocontrollerboard dient ein *XMC 2Go XMC1100 V1* der Infineon Technologies AG, welcher über ein TFDU4101 Transceiver-Modul von Vishay Intertechnology, Inc. kommuniziert und dessen Messwerte über die Bedienelemente simuliert werden.

Insgesamt entstanden zwei Versionen. Eine, welche die Adapterplatine *FPC100P010* von Proto Advantage nutzt, um das Transceiver-Modul und die Mikrocontrollerplatine zu verbinden und eine zweite, bei der alle Komponenten auf einer eigens angefertigten Platine mit den Abmessungen der Aufstandsfläche einer Batteriezelle montiert sind.

Voruntersuchung mit einer Adapterplatine

Zunächst wurde geprüft, ob eine Proto Advantage *FPC100P010* Adapterplatine [38] genutzt werden kann, um das Transceiver-Modul direkt auf die Mikrocontrollerplatine aufzusetzen. Die drei in Abbildung 3.8 oben zu sehenden, auf ihre Anschlusspins reduzierten Komponenten wurden übereinander gelegt und es wurde geprüft, in welcher Anordnung dieser Aufbau funktionieren könnte. Es ergab sich die in Abbildung 3.8 unten dargestellte Anordnung, bei der die Adapterplatine mit dem Transceiver-Modul von unten in die Mikrocontrollerplatine eingesetzt wird. Zur Umverlegung des Transceiver-Modul-Anschlusses 4 (*RXD*) für das Empfangssignal auf den Mikrocontrollerplattenanschluss 5 (*P2.11*) ist eine Drahtbrücke zwischen den Anschlussflächen 4 und 10 auf der Adapterplattenunterseite bestimmt. Ebenso ist die Positionierung des Sendediodenvorwiderstandes zwischen Anschlussfläche 1 und 6 vorgesehen.

Anhand eines Aufbaus dieses Entwurfs und einer weiteren Adapterplatine mit Transceiver-Modul wurde die Funktion bestätigt. Fotos der Adapterplatten sind in Abbildung 3.9 und Fotos des Gesamtaufbaus in Abbildung 3.10 zu finden. Wie in Abbildung 3.10 zu sehen ist, befindet sich an der Mikrocontrollerplatine eine Buchsenleiste mit den Anschlüssen 5 (*VSS*), 13 (*P0.14*), 14 (*P0.15*), 15 (*P2.0*) und 16 (*P2.6*), über welche Bedienelemente angeschlossen werden können.

Zur Planung des Demonstrationsaufbaus wurden 3D-Modelle der Komponenten in der Software *DesignSpark Mechanical 4.0* von Ansys, Inc. und RS Components [1] erstellt. In Abbildung 3.11 sind diese aus verschiedenen Perspektiven zu sehen.

Wie in Abbildung 3.12 zu sehen, ist bei diesem Aufbau der Zugang zu den USB-Mikro-B-Anschlüssen durch die nächste Mikrocontrollerplatine eingeschränkt.

Weiterhin sollte das Messmodul nur die Adapterplatine für höhere Positionspräzision der Transceiver-Module befestigt werden, da das präzise Anlöten an die Mikrocontrollerplatten schwierig ist. Auf Grund fehlender Löcher an der Adapterplatine müsste ein

Einschub- oder Klemmsystem entwickelt werden, welches die Linsen des Transceiver-Moduls nicht verdeckt und ein Aufliegen des Lichtleitkörpers auf der Adapterplatine ermöglicht. Außerdem gibt es hier noch keine Bedienelemente, welche durch eine externe Platine ergänzt werden müssten.

Aus diesen Gründen wurde dieser Ansatz nicht weiter verfolgt und stattdessen eine eigene Platine für alle Komponenten entwickelt.

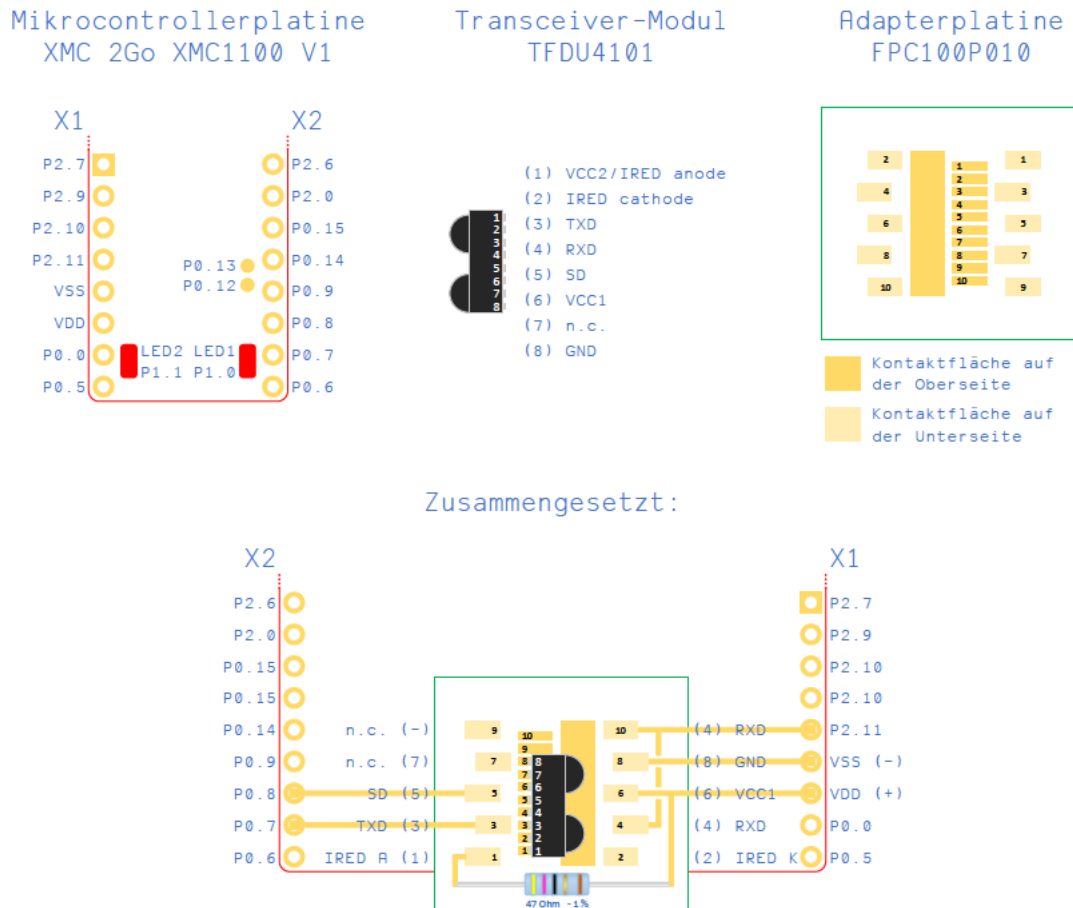


Abbildung 3.8: Vereinfachte Darstellung der Komponenten für das Messmodul mit aufgesetzter Adapterplatine und vereinfachtem Anschlussplan. Links oben ist der Mikrocontrollerplatinenteil mit den Anschlusspins, mittig oben das Transceiver-Modul mit Anschlussbeschriftung, rechts oben die Adapterplatine und mittig unten alle drei Komponenten im vereinfachten Anschlussplan zusammen gesetzt zu sehen. Die Funktionen der Mikrocontroller- und Transceiver-Modul-Anschlüsse können dem jeweiligen Datenblatt [19, S. 22–26] [34, S. 2, S. 7] entnommen werden.

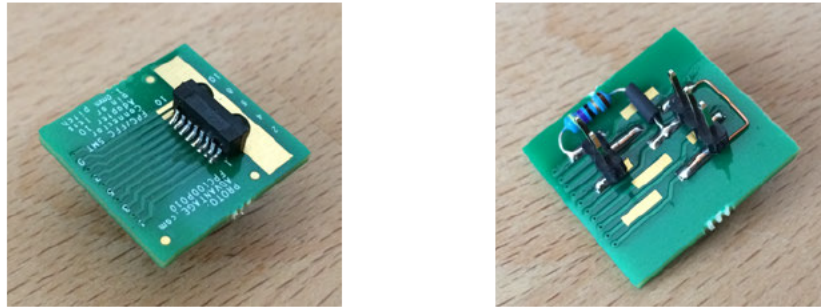


Abbildung 3.9: Fotos der aufgebauten Adapterplatine mit Transceiver-Modul, Drahtbrücke und Vorwiderstand. Links ist die Ober- und rechts die Unterseite zu sehen.

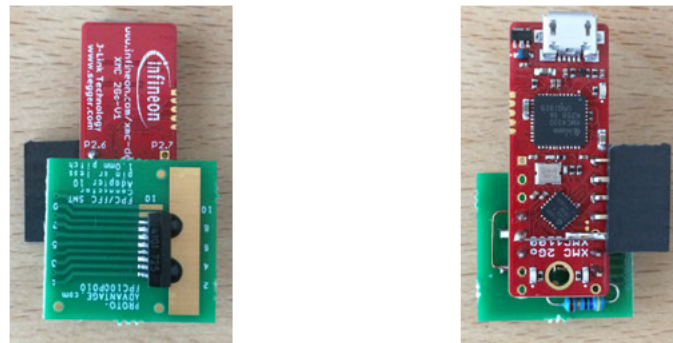


Abbildung 3.10: Fotos des Messmodulaufbaus mit Adapterplatine. Die bestückte Adapterplatine ist am Mikrocontroller über die Anschlusspins angelötet.

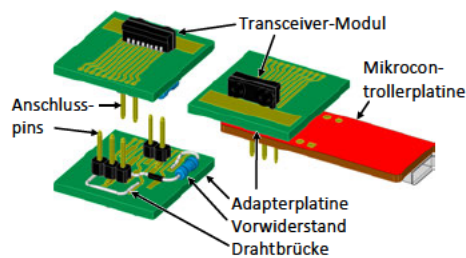


Abbildung 3.11: 3D-Modelle der zusammengesetzten Komponenten im Aufbau mit Adapterplatine. Links ist die Adapterplatine mit Anschlusspins, Vorwiderstand, Drahtbrücke und Transceiver-Modul von oben und unten zu sehen. Rechts ist diese in die Mikrocontrollerplatine eingesetzt.

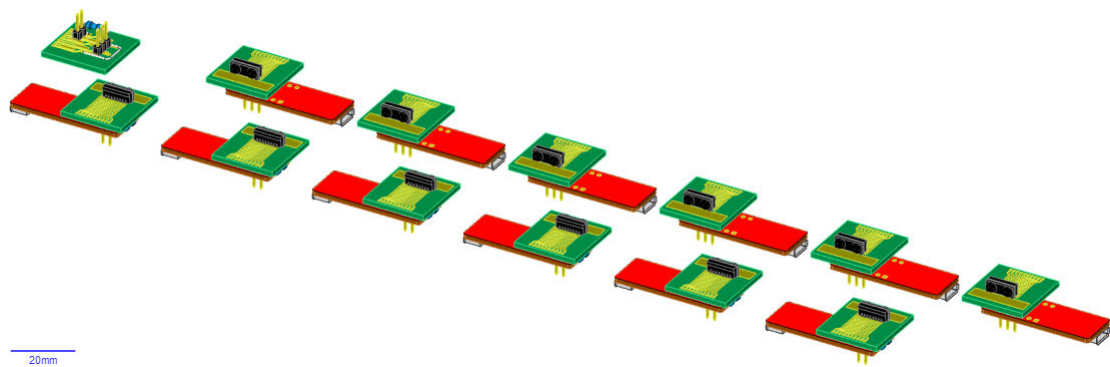


Abbildung 3.12: 3D-Modelle der Messmodule mit Adapterplatine zusammengesetzt in Batterieanordnung. Links oben ist keine Mikrocontrollerplatine vorhanden, da diese Adapterplatine an die Mikrocontrollerplatine des Masters angeschlossen wird.

Platinenentwicklung

Für eine präzise und einfache Montage aller Komponenten und Bedienelemente in einer Einheit wurde eine Platine mit zugehörigem Schaltplan in der Software *KiCad 5.1.6-1* vom *KiCad Developers Team* [7] entworfen. Der Schaltplan ist in Abbildung 3.14 und den Platinenentwurf in Abbildung 3.13 zu finden. Tabelle 3.14 zeigt die Teileliste. Die folgend den Komponentennamen nachgestellten Bezeichner beziehen sich auf diese drei Dokumente.

Die Außenmaße der Platine entsprechen den in Kapitel 2.4.2 ermittelten der Zellen-grundfläche von 26 mm Breite und 140 mm Länge. Es ist kein Platz für Zellanschlüsse vorgesehen, da diese Platine, wie in Kapitel 2.1 beschrieben, nicht auf Zellen aufgesetzt werden muss.

Vier Befestigungslöcher mit einem Durchmesser von 3,2 mm sind mit einem Abstand von 3 mm zu den Platinenrändern in allen Ecken platziert. Um die Platinenmitte angeordnet liegen vier Befestigungslöcher für den Lichtleitkörper, wobei die Löcher eines Paares einen Abstand von 19 mm und die Paare zueinander 30 mm haben.

Mit einem Distanz von 38 mm sind parallel zu den kurzen Platinenkanten zwei 2 mm breite Streifen markiert, welche die Außengrenzen im Kanal des Batterierahmens darstellen.

Die Transceiver-Module ($U2$) befinden sich an den in Kapitel 2.4.2 definierten Positionen. Die Bezeichner in Klammern beziehen sich hierbei und im weiteren Verlauf auf den Schaltplan in Abbildung 3.13, den Platinenentwurf in Abbildung 3.14 und die Teileliste in Tabelle 3.14. Auf der gleiche Seite neben dem Lichtleitkörperbereich ist auch die Mikrocontrollerplatine ($U1$) platziert. Zusätzliche Anschlussreihen ($J1$, $J2$) ermöglicht den nachträglichen Anschluss externer Komponenten. Weiterhin gibt es Anschlüsse für einen Temperatursensor ($TH1$) und externe Versorgung ($J3$). Letzterer ist durch eine Verpolungsschutzdiode ($D1$) mit der Platinenversorgung verbunden, um die Komponenten bei Verpolung zu schützen sowie bei der Versorgung mehrerer Platinen durch ein Netzteil und der Versorgung einer Mikrocontrollerplatine durch deren USB-Mikro-B-Anschluss einen Rückstrom in das Netzteil oder bei nicht verbundenem oder abgeschaltetem Netzteil die Versorgung der anderen angeschlossenen Platinen zu verhindern.

Auf der gegenüberliegenden Seite ist der Raum für die Bedienelemente vorgesehen. Ein Drehschalter ($SW1$), ein Drucktaster ($SW2$), eine LED ($D3$) und zwei Potentiometer

(*RV1*, *RV2*) finden hier Platz. Die Mikrocontrollerplatine wertet all diese Eingänge aus beziehungsweise steuert den Ausgang an. Die Peripheriekomponenten des Tasters (*SW2*) dienen zur Entprellung. Der Kondensator (*C3*) wird bei unbetätigtem Taster (*SW2*) über den Widerstand (*R3*) und die Überbrückungsdiode (*D2*) geladen. Wird der Taster (*SW2*) gedrückt, entlädt sich der Kondensator (*C3*) über den Widerstand (*R4*). Die Überbrückungsdiode (*D2*) kann auch entfallen, wenn die Steigung der steigenden Signalflanke nicht so steil sein muss.

Anstelle der zwei Potentiometer können auch ein Spannungsteiler aus den Widerständen *R6* und *R7* zur Messung der Versorgungsspannung, sowie ein temperaturabhängiger Widerstand an Anschluss *TH1* mit dem Widerstand *R8* zur Temperaturmessung verbaut werden. Die Zenerdioden *D4* und *D5* dienen als Überspannungsschutz. In den eingesetzten Platinen wurden jedoch nur die Potentiometer bestückt.

Um alle Schraublöcher herum wurde die Massefläche auf der Unterseite der Platine für Schraubköpfe mit 6 mm Durchmesser ausgespart. Auf der Oberseite wurden alle Leiterbahnen möglichst weit entfernt von den Schraublöchern platziert, allerdings werden die Leiterbahnen neben den Löchern beim Transceiver-Modul (*U2*) durch einen Schraubkopf abgedeckt, sodass hier zur Vermeidung eines Kurzschlusses keine Schrauben direkt auf der Platine aufliegen dürfen. Da hier der Lichtleitkörper zwischen Schraubkopf und Platine liegt, stellt dies kein Problem dar.

Des Weiteren wurde eine Sollbruchstelle in Form von Bohrungen vor den Transceiver-Modul-Linsen (*U2*) hinzugefügt, damit diese Platine kompakter ist, wenn sie für die Modulüberwachung (Master) eingesetzt wird. Auf dem abgetrennten Teil befindet sich zur Weiterverwendbarkeit die Anschlussleiste *J4*, womit dieser Teil für die Bedienelemente der Modulüberwachung (Master) dienen kann.

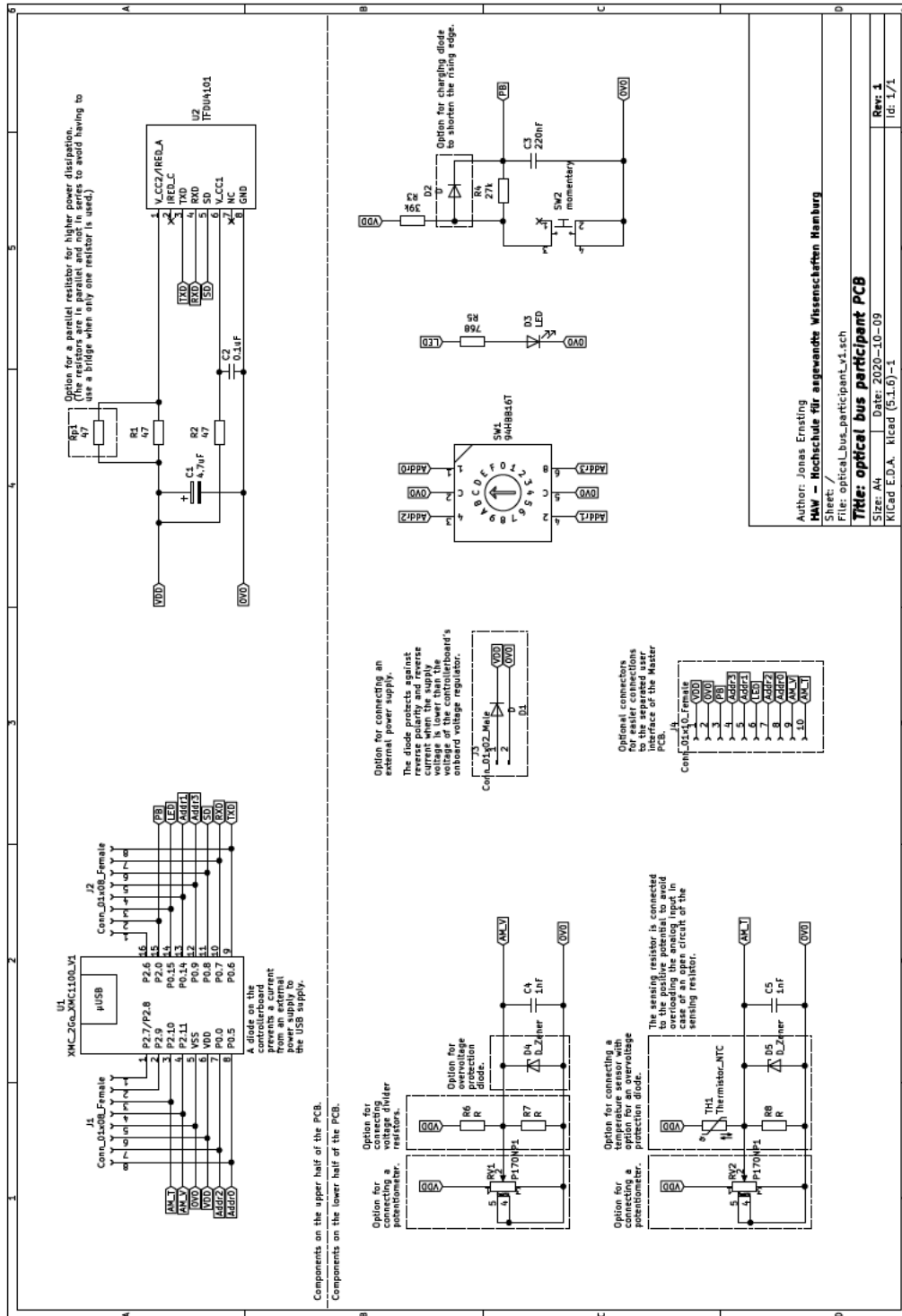


Abbildung 3.13: Schaltplan der Messmodulplatine.

Tabelle 3.14: Teileliste der Busteilnehmer

Komponente	Referenz	Hersteller	Hersteller-Teilenummer	Stück / PCB	Stück, ges.	Beschreibung
Mikrocontrollerboard	-	Infineon	KIT_XMCA7_RELAX_V1	1	1	
Mikrocontrollerboard	U1	Infineon	KIT_XMC_2Go_XMCC100_V1	1	12	
LED	-	-	31601	1	1	
Pull-Up-Widerstände für I2C-Bus	-	-	-	2	2	4,7kOhm
Prototypplatine	-	-	-	1	1	
Messmodulplatine	-	-	-	1	13	
Koillierter Drehschalter	SW1	Grayhill	94HBBJ6T	1	13	13 TH1, 6-sch auflegend, 16 Positionen (eol HEX), Bedienstange (lfd 6,86x3,56mm), 10NDC, Dauerstrom 0,1A@30V, Schaltstrom 30mA@30V
Griff für kodierten Drehschalter	-	Grayhill	94705-019	1	13	13 Drehknopf, grau, h. 5,33mm, mit Pfeil
IPDA-Transceiver	U2	Vishay	TFD4101-TB	1	13	13 SMT, M41 (328 metrisch), Tantalum, 4,7µF±10%, 16VDC, ESR 2,90hm
Kondensator für IRED LED	C1	Vishay	293D475X9016B2TE3	1	13	13 SMT, 0603 (1608 metrisch), Keramik, 0,1µF±5%, 50VDC
Kondensator für V_CCC Filter	C2	Vishay	VJ0603F104MAGV1BC	1	26	26 SMT, 1206 (3216 metrisch), Thick Film, 470nm±1%, 100ppm, -55...155°C, 200V, 250mW
Vor- und Filterwiderstand für IRED LED und V_CCC	RL, Rp1, R2	Vishay	CRCW12064700KEA	2	13	13 TH1, 6-sch auflegend, SPST, OFF-ON), grau, Bedienstange (rund, lfd 7,6mm)
Taster	SW2	C&K	D6K102LFS	1	13	13 SMT, 0603 (1608 metrisch), Thick Film, 39kOhm±1%, 100ppm, -55...155°C, 75V, 100mW
Pull-Up-Widerstand für Tasterentpörrung	R3	Vishay	CRCW0603390KKEA	1	13	13 SMT, 0603 (1608 metrisch), Thick Film, 27kOhm±1%, 100ppm, -55...155°C, 75V, 100mW
Pull-Down-Widerstand für Tasterentpörrung	R4	Vishay	CRCW0603270KKEA	1	13	13 SMT, 0603 (1608 metrisch), Thick Film, 27kOhm±1%, 100ppm, -55...155°C, 75V, 100mW
Kondensator für Tasterentpörrung	C3	Vishay	VJ0603F224MCPV1BC	1	13	13 SMT, 0603 (1608 metrisch), Keramik, 0,22µF±5%, 10VDC
LED	D3	Vishay	TLE4401	1	13	13 TH1, T-1 (3mm), gelb (589nm), 1,8V, 2mA, 60° (diffus), 17mcd
Vorwiderstand für LED	R5	Vishay	CRCW0603768RKEA	1	13	13 SMT, 0603 (1608 metrisch), Thick Film, 768Ohm±1%, 100ppm, -55...155°C, 75V, 100mW
Potentiometer für Zellspannungs- und Temperaturersatzwert	RV1, RV2	TT Electronics	P170NP1-CC15BR50K	2	24	24 TH1, 6-sch auflegend, Conductive Plastic, 50kOhm±20%, 200VAC, 100mW, linear, 260°±10°, Bedienstange (lfd 15,6mm)
Potentiometer für beide Analogeingänge	C4, C5	Vishay	VJ0603A1020APV1BC	2	24	24 SMT, 0603 (1608 metrisch), Keramik, 10µF±2%, 50VDC
Pinreihen für Controllerboard	-	Amphenol FO	10129378-909001BLF	2	24	24 TH1, 2,54mm Lochabstand, Pins, 1x8 Kontakte, 5,84mm Einseitstiefe, 1,10V, 3A, vergoldet
Buchsenreihen für Controllerboard/Pis line	J3, J4	Amphenol FO	79515-3081F	2	24	24 TH1, 2,54mm Lochabstand, Buchse, lfd Kontakte, 3,6...6,81mm Einseitstiefe, 1xV, 2A, vergoldet
Pinreihe zum Trennen des Linearspannungreglers	J2	Amphenol FO	10129378-902001BLF	1	12	12 TH1, 2,54mm Lochabstand, Pins, 1x2 Kontakte, 5,84mm Einseitstiefe, 1,10V, 3A, vergoldet
Steckbrücke für zum Trennen des Linearspannungreglers	-	Harwin	M7967-05	1	12	12 Steckbrücke, 2,54mm Lochabstand, 1x2 Kontakte, ca 5,5...6mm Einseitstiefe, 230V, 3A, vergoldet, schwarz
Verpörrungsschutzdiode	D1	Vishay	SL13-EJ/SAT	1	12	12 SMA (DO-214AC), S, V f 0,485V, rrr 30V, I f 1,5A, I frr 50A, I r 0,2mA

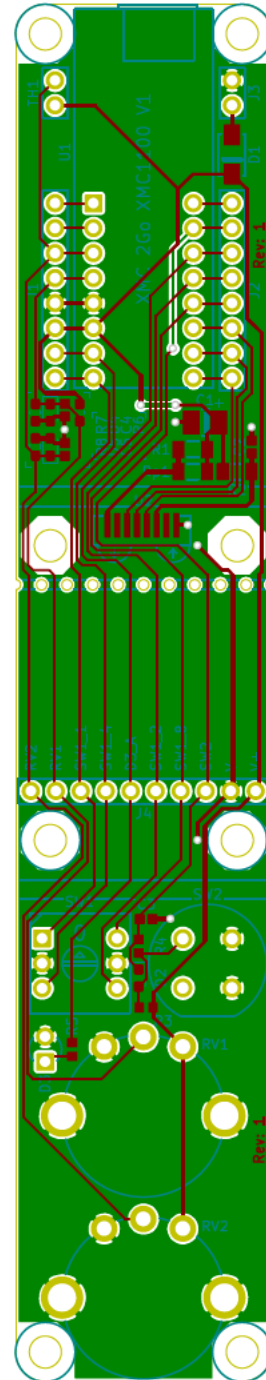


Abbildung 3.14: Entwurf der Messmodulplatine.

Dimensionierung des Kondensators C3 und der Widerstände R3 und R4

Für eine Entprellung des Tasters *SW2* wurden die Widerstände *R3* und *R4* sowie der Kondensator *C3* eingesetzt. Die Diode *D2* kann zur Verkürzung der steigenden Signalflanke hinzugefügt werden, entfiel jedoch bei den eingesetzten Platinen.

In einer Simulation mit der Software *LTSpice* [28] wurde die Entprellschaltung simuliert.

Dabei wurden die Werte $R3 = 39\text{ k}\Omega$, $R4 = 27\text{ k}\Omega$ und $C3 = 0,22\text{ }\mu\text{F}$ für die Prellzeit des Tasters von $10\text{ }\mu\text{s}$ [23, S. 1] ermittelt.

Dimensionierung der Kondensatoren C4 und C5

Der Widerstandswert der Potentiometer beträgt $50\text{ k}\Omega$. Dieser Wert wurde gewählt, um den Linearspannungsregler der Mikrocontrollerplatine nicht zu belasten. Ein solch hoher Widerstandswert verlängert die erforderliche Messdauer zum Laden der Analogeingangskapazität. Um diese so kurz wie möglich zu halten, wurden Kondensatoren mit der Kapazität 1 nF parallel zum Analogeingang vorgesehen. Dies entspricht der hundertfachen Kapazität eines Analogeingangs mit 10 pF [22, S. 36], womit die Messabweichung nach Gleichung 3.29 weniger als 1% betragen sollte. Da die Präzision der Messung bei diesem Demonstrationsaufbau nicht entscheidend ist, wurde der Umladewiderstand und die zugehörige -zeit vernachlässigt. Um den Einfluss der Leiterbahnimpedanz niedrig zu halten, sind die Kondensatoren nahe an den Analogeingängen platziert.

$$Q_{Ges} = Q_C + Q_{ADCINPUT} \quad (3.27)$$

Mit $Q = C \cdot U$:

$$\begin{aligned} C_{Ges} \cdot U_{Ges} &= C_C \cdot U_C + C_{ADCINPUT} \cdot U_{ADCINPUT} && | \cdot \frac{1}{C_{Ges}} \\ U_{Ges} &= \frac{C_C \cdot U_C + C_{ADCINPUT} \cdot U_{ADCINPUT}}{C_{Ges}} && (3.28) \end{aligned}$$

Die maximale Abweichung vom Messwert dU beträgt damit:

$$\begin{aligned}
 dU &= \frac{U_C - U_{Ges}}{U_C} = 1 - \frac{U_{Ges}}{U_C} \\
 dU &= 1 - \frac{C_C \cdot U_C + C_{ADCINPUT} \cdot U_{ADCINPUT}}{C_{Ges} \cdot U_C} \\
 dU &= 1 - \frac{1 \text{ nF} \cdot 3,3 \text{ V} + 0,010 \text{ nF} \cdot 0 \text{ V}}{1,01 \text{ nF} \cdot 3,3 \text{ V}} \approx 0,99\%
 \end{aligned} \tag{3.29}$$

Die externe LED $D3$ soll durch die Ausgänge der Mikrocontrollerplatinen versorgt werden, welche maximal einen Strom von 5 mA [19, S. 31] [22, S. 46] zur Verfügung stellen können. Daher wurde eine LED mit einem nominalen Strom von $I_{FLED} = 2 \text{ mA}$ [24, S. 1] ausgewählt. Mit einer Versorgungsspannung $U_{DD} = 3,32 \text{ V}$ und einer Vorwärtsspannung $U_F = 1,8 \text{ V}$ [24, S. 1] ergibt sich der Vorwiderstandswert zu $R_{LED} = \frac{3,32-1,8}{2 \cdot 10^{-3}} \cdot \frac{\text{V}}{\text{A}} = 760 \Omega$. Als daher wurde ein Standardwiderstand mit $R_{LED} = 768 \Omega$ ausgewählt. Die Leistung über dem Widerstand liegt bei etwa $P_{RLED} = I_F^2 \cdot R_{LED} = (2 * 10^{-3} \text{ A})^2 \cdot 768 \Omega \approx 3 \text{ mW}$.

Belastung des Linearspannungsreglers durch die Bedienelemente

Der durch die Bedienelemente abverlangte Strom vom Linearspannungsregler der Mikrocontrollerplatinen beträgt nach Gleichung 3.30 $I_{UI} \approx 2,5 \text{ mA}$, was deutlich unter dem in Kapitel 3.2.1 berechneten von $I_{ExtIO} = 14,7 \text{ mA}$ liegt, womit keine Überlastung auftreten sollte. Dies wurde durch die Test in Kapitel 4.2.5 bestätigt.

$$\begin{aligned}
 I_{UI} &= 2 \cdot I_{RV} + I_{FLED} + I_{FLED} + I_{BTN} \\
 &= 2 \cdot \frac{U_{DD}^2}{R_V} + I_{FLED} + \frac{U_{DD}^2}{R_3}
 \end{aligned} \tag{3.30}$$

$$\begin{aligned}
 &= 2 \cdot \frac{(3,32 \text{ V})^2}{50 \text{ k}\Omega} + 2 \text{ mA} + \frac{(3,32)^2}{39 \text{ k}\Omega} \approx 2,5 \text{ mA}
 \end{aligned} \tag{3.31}$$

3.2.3 Modulüberwachung

Die Modulüberwachung besteht aus Mikrocontrollerboard, LCD, Bedienelementen und optischem Transceiver-Modul. Für das Mikrocontrollerboard wurde ein *XMC4700 Relax Kit V1* verwendet, welcher Informationen auf einem 2004A LCD-Modul mit Ein-/Ausgangserweiterung über I2C-Anschluss anzeigen kann. Als Bedienelemente sind ein Taster (*SW2*), ein Drehschalter (*SW1*) und eine LED (*D3*) verbaut, welche auf einer Bedienelementhälfte einer zertrennten Messmodulplatine sitzen und über eine Anschlussleiste (*J4*) mit der Mikrocontrollerplatine verbunden sind. Kommuniziert wird mittels eines *TFDU4101* Transceiver-Moduls (*U2*), welches auf der anderen Messmodulplatinenhälfte montiert ist und ebenfalls durch Leitungen an den zwei Anschlussleisten (*J1*, *J2*) mit der Mikrocontrollerplatine verbunden ist. Die Komponenten in Klammern nachgestellten Bezeichner beziehen sich auf den Schaltplan in Abbildung 3.13, den Platinenentwurf in Abbildung 3.14 und die Teileliste in Tabelle 3.14.

Zur Versorgung aller externen Komponenten und für die Pull-Up-Widerstände des I2C-Busses wurde ein kleines Verteilermodul aus einer Prototypenplatine angefertigt. Der vereinfachte Anschlussplan der Modulüberwachung und eine Detailansicht des Verteilermoduls ist in Abbildung 3.15 zu sehen.

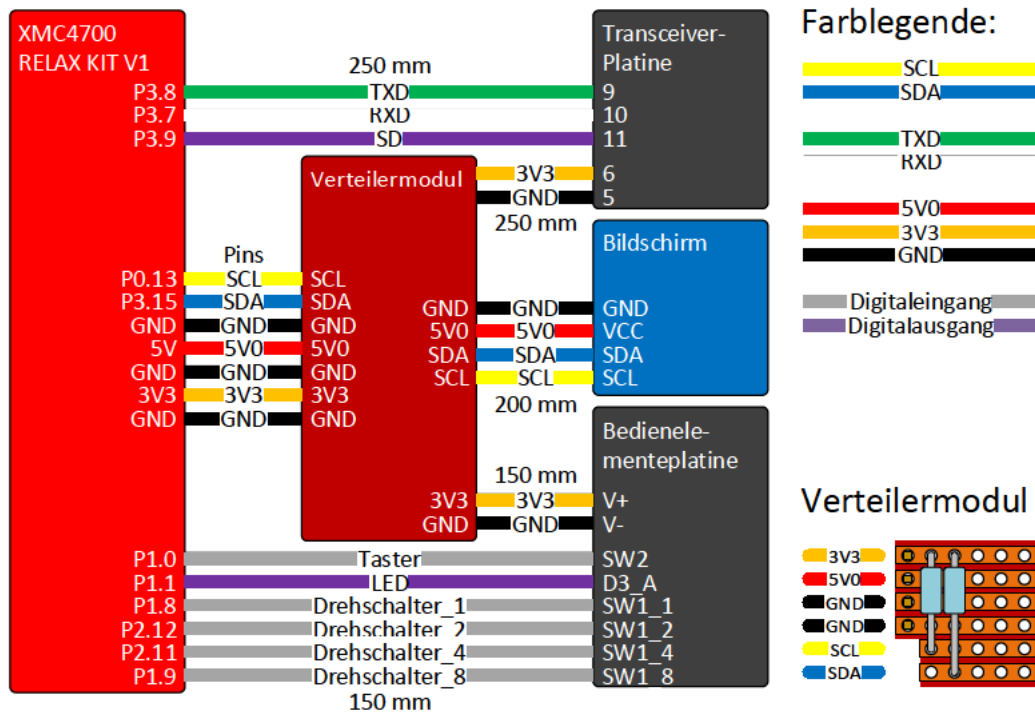


Abbildung 3.15: Vereinfachter Anschlussplan der Modulüberwachung. Unter oder über den Leitungen steht deren Länge. Rechts unten ist eine Detailansicht des Verteilermoduls abgebildet.

Belastung des Linearspannungsreglers durch die Bedienelemente

Die Bedienelemente, bestehend aus der gelben LED ($D3$) und dem Taster ($SW2$), belasten den Linearspannungsregler des *XMC4700 Relax Kit V1* mit einem Strom von $I_{UI} = I_{FLED} + I_{FLED} + I_{BTN} = +I_{FLED} + \frac{U_{DD}^2}{R_3} = 2 \text{ mA} + \frac{(3,32 \text{ V})^2}{39 \text{ k}\Omega} \approx 2,3 \text{ mA}$, welcher weit unter dem in Kapitel 3.2.1 berechneten von $I_{ExtIO} = 563,8 \text{ mA}$ liegt und somit keine Überlastung des Linearspannungsreglers auftreten sollte.

Der LCD wird über den Anschluss 5V der Mikrocontrollerplatine versorgt, welcher wiederum über Dioden durch die USB-Mikro-B-Anschlüsse versorgt wird [16].

3.2.4 Mechanische Konstruktion

Für einen stabilen Demonstrationsaufbau werden alle Komponenten auf einer Grundplatte aus Holz mit den Außenmaßen von etwa 50 x 28 x 1,8 cm befestigt.

Für den Entwurf der Halter für die Messmodule, USB-Hubs und Leitungen sowie des Gehäuses für das Überwachungsmodul zur Montage an der Holzplatte wurde erneut die Software *DesignSpark Mechanical 4.0* von Ansys, Inc. und RS Components [1] genutzt. In den Abbildungen 3.16, 3.17 und 3.18 sind diese dargestellt. Die versorgenden USB-Leitungen werden von den USB-Hubs durch Löcher in der Holzplatte unter diese geführt, dort verlegt und gelangen durch Löcher wieder nach oben zu den Verbrauchern.

Alle Formteile wurden mit einem 3D-Drucker hergestellt und anschließend damit alle Komponenten auf der Grundplatte montiert. Der fertiggestellte Versuchsaufbau ist in Abbildung 3.19 zu sehen.

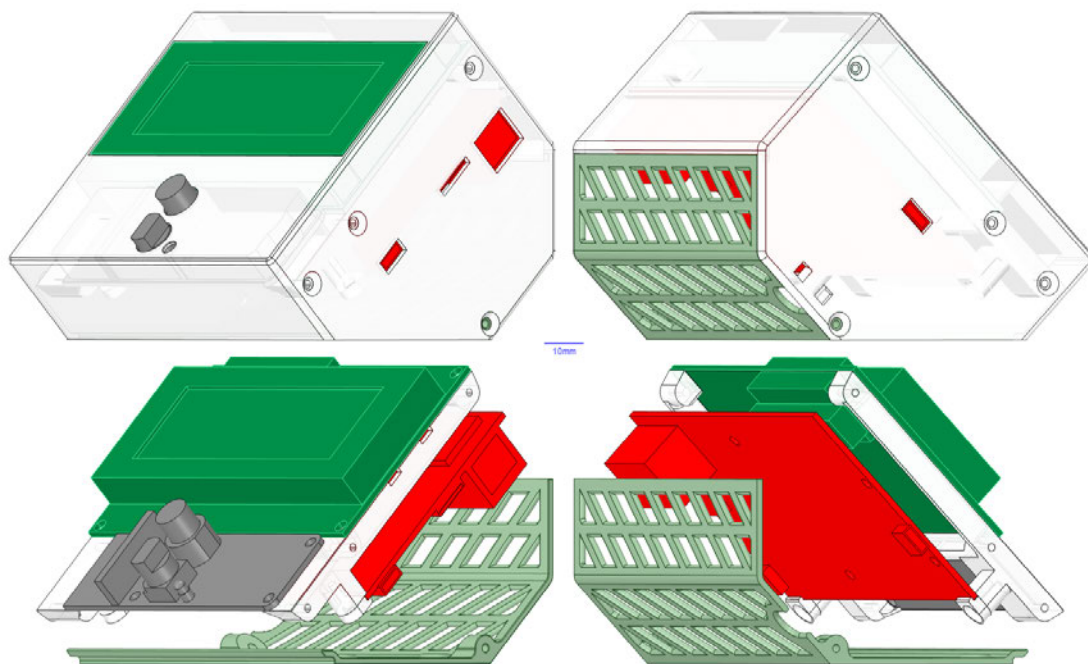


Abbildung 3.16: 3D-Modell des Überwachungsmodulgehäuses. Oben ohne und unten mit Außengehäuse. In grün ist das LCD-Modul, grau die Messmodulplatinehälfte mit den Bedienelementen und in rot die Mikrocontrollerplatine.

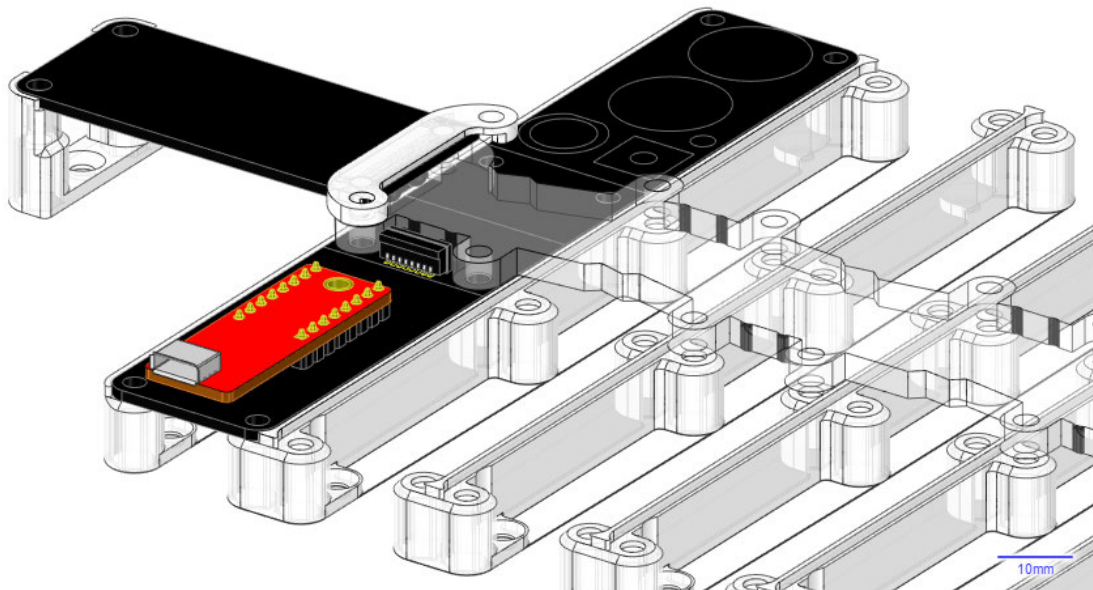


Abbildung 3.17: 3D-Modelle der Messmodule. Links oben befindet sich die Messmodulplatinehälfte mit dem Transceiver-Modul des Überwachungsmoduls. Der Halter links stützt die Platine, während der Bügel auf der anderen Seite die Platine am Lichtleitkörper ausrichtet. Mittig befinden sich die Messmodulplatinen mit Mikrocontrollerplatine und Transceiver-Modul, welche durch die Träger befestigt sind. Alle Schraublöcher sind mit M3-Muttern versehen.

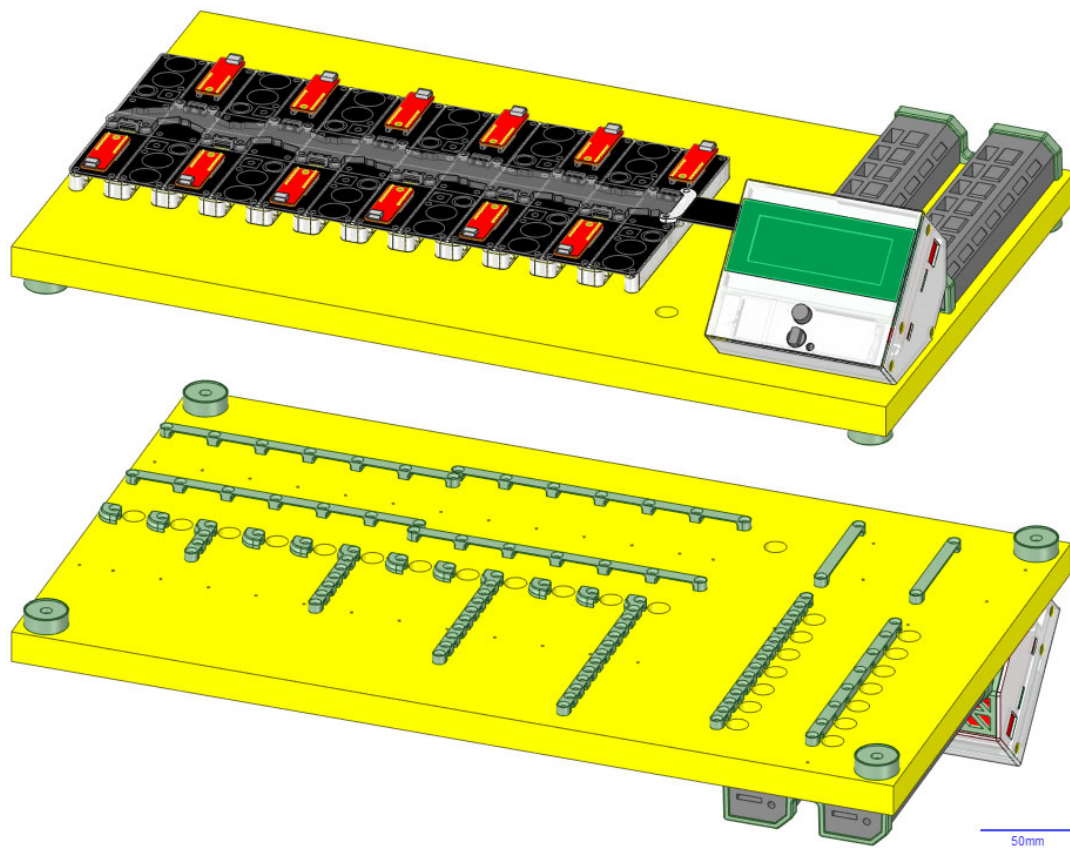


Abbildung 3.18: 3D-Modell des Demonstrationsaufbaus. Oben ist die Oberseite und unten die Unterseite der Grundplatte zu sehen.

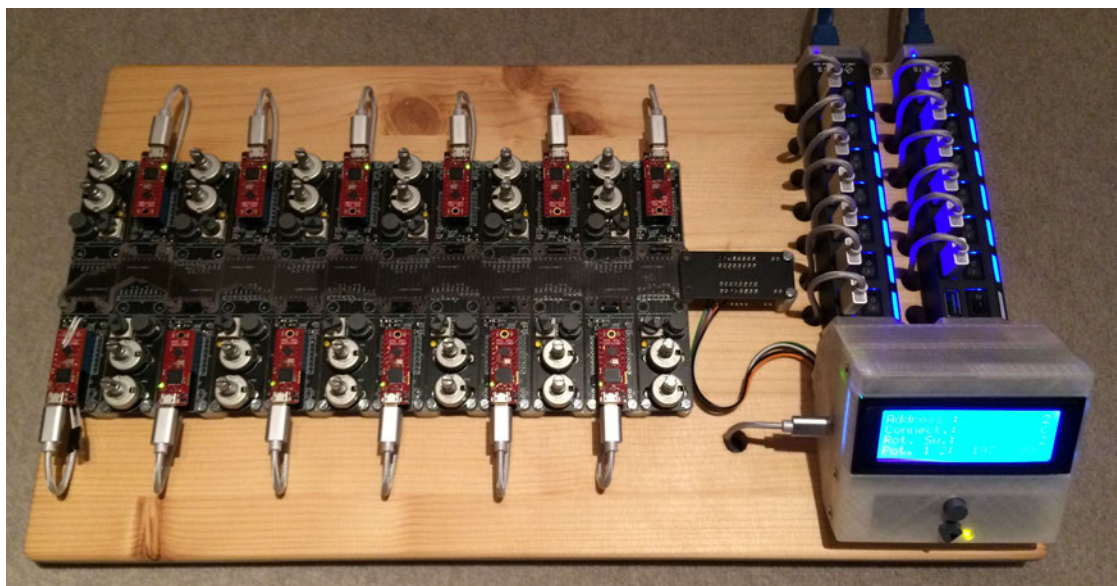


Abbildung 3.19: Foto des fertiggestellten Versuchsaufbaus. Auf diesem Bild ist der Lichtleitkörper nicht befestigt.

3.3 Software

Die Programmierung der Mikrocontroller erfolgte mit der Programmier- und Debugsoftware *DAVE Version 4.4.2* von *Infineon Technologies India Pvt. Ltd.* Diese ermöglicht eine einfache Konfiguration der Mikrocontroller über *DAVE-APPs*, mit denen die Einstellungen der Mikrocontrollerkomponenten über grafische Benutzeroberfläche ausgewählt und vorgegeben werden kann. Aus diesen wird der Initialisierungs-Code in der Programmiersprache C generiert, sodass im Hauptprogramm nur eine Funktion zur Initialisierung aller Mikrocontrollerkomponenten aufgerufen werden muss. Die generierten Code-Dateien können manuell angepasst werden, allerdings muss dies nach jeder Generierung erneut geschehen.

Die Programmierung des Hauptprogramms geschieht ebenfalls in der Programmiersprache C. Die Code-Dateien sind im Anhang in Kapitel A.2 zu finden.

3.3.1 Limitierung der DAVE-APPs

Die Konfiguration der Mikrocontrollerkomponenten durch die grafische Oberfläche der *DAVE-APPs* und die automatische Generierung des Initialisierungs-Codes aus diesen kann den Programmieraufwand reduzieren, allerdings sind nicht immer alle konfigurierbaren Funktionen über die *DAVE-APPs* einstellbar. So kann in der *UART-APP Version 4.1.12* nicht die Pulslänge und der Abtastmodus sowie -zeitpunkt eingestellt werden. Um die Konfiguration entsprechend Kapitel 2.3 anzupassen, wird der folgende Code-Teil in der generierten Initialisierungsfunktion der UART-Schnittstelle

```
UART_Internal_init()
```

unter dem Dateipfad

```
Dave\Generated\UART\uart_conf.c
```

nach der Funktion

```
XMC_UART_CH_Init(...);
```

hinzugefügt:

```
// manually inserted code for IrDA transceiver
// set sample mode to one sample per bit and the sample point
// to 2/16 of the bit length (after time quanta 1)
UART_Internal.channel->PCR_ASCMode =
    (uint32_t) ((UART_Internal.channel->PCR_ASCMode &
        (~USIC_CH_PCR_ASCMode_SMD_Msk) &
        (~USIC_CH_PCR_ASCMode_SP_Msk)) |
        (((uint32_t) 0UL) << USIC_CH_PCR_ASCMode_SMD_Pos) |
        (((uint32_t) 1UL) << USIC_CH_PCR_ASCMode_SP_Pos));
// enable pulse output and set pulse length to 4/16 of the bit
// length
UART_Internal.channel->PCR_ASCMode =
    (uint32_t) ((UART_Internal.channel->PCR_ASCMode &
        (~USIC_CH_PCR_ASCMode_PL_Msk)) |
        (((uint32_t) 3UL) << USIC_CH_PCR_ASCMode_PL_Pos));
// invert data output
UART_Internal.channel->SCTR =
    (uint32_t) (UART_Internal.channel->SCTR &
        (~USIC_CH_SCTR_DOCFG_Msk)) |
    (uint32_t) XMC_USIC_CH_DATA_OUTPUT_MODE_INVERTED;
```

Dieser konfiguriert die UART-Schnittstelle entsprechend der definierten Vorgaben um, bevor sie durch die Funktion

```
XMC_UART_CH_Start(...);
```

aktiviert wird.

3.3.2 Implementierte Befehle

Für die Steuerung des Überwachungsmoduls über die serielle Schnittstelle und der Messmodule sind verschiedene Befehle implementiert, auf die es entsprechende Antworten gibt. Diese und deren Funktion sowie Bedeutung sind in Tabelle 3.15 beziehungsweise 3.16 aufgeführt.

Tabelle 3.15: Implementierte Befehle und Antworten im Überwachungsmodul und deren Funktion

Befehl oder Antwort	Binär-Code	Beschreibung
Unbekannter Befehl	0b 0001	Diese Antwort sendet der Master nach dem Empfangen eines undefinierten Befehls von der seriellen Schnittstelle.
Ping	0b 0010	Befehl der seriellen Schnittstelle für eine Antwort des Masters mit dem gleichen Inhalt.
Slave antwortet nicht	0b 0011	Antwort des Masters nach dem Weiterleiten einer Nachricht an die Slaves und Ausbleiben einer Antwort durch die Slaves.
Befehl ignoriert	0b 0100	Antwort des Masters, wenn ein Befehl von der seriellen Schnittstelle eintrifft, dieser aber wegen einer aktuell ausgeführten Aktion ignoriert wird.
Abbruch	0b 0101	Befehl von der seriellen Schnittstelle an den Master zum Abbrechen der aktuell ausgeführten Aktion. Der Master antwortet nach dem Abbruch mit gleichem Inhalt.
Sende Adressen aller verfügbaren Slaves	0b 0111	Befehl von der seriellen Schnittstelle an den Master, die Adressen aller momentan verfügbaren Slaves zu senden.
Sende Daten aller verfügbaren Slaves	0b 1000	Befehl von der seriellen Schnittstelle an den Master, die Daten aller momentan verfügbaren Slaves in Form von ASCII-Zeichen tabellarisch zu senden.
Sende Daten aller Slaves	0b 1001	Befehl von der seriellen Schnittstelle an den Master, die Daten aller Slaves in Form von ASCII-Zeichen tabellarisch zu senden.
Schalte automatisches Senden der Daten aller Slaves um	0b 1010	Befehl von der seriellen Schnittstelle an den Master zum Ein- und Ausschalten des automatischen Sendens der Daten aller Slaves in Form von ASCII-Zeichen in tabellarischer Anordnung.

Tabelle 3.16: Implementierte Befehle und Antworten im Messmodul und deren Funktion

Befehl oder Antwort	Binär-Code	Beschreibung
Unbekannter Befehl	0b 0001	Diese Antwort sendet der Slave nach dem Empfangen eines undefinierten Befehls vom Master.
Ping	0b 0010	Befehl des Masters für eine Antwort des Slaves mit dem gleichen Inhalt.
Abbruch	0b 0011	Befehl des Masters zum Abbrechen der aktuell ausgeführten Aktion. Der Slave antwortet nach dem Abbruch mit gleichem Inhalt.
Sende Daten	0b 0100	Befehl des Masters an den Slave, alle Daten zu senden, mit welchen der Slave unmittelbar im Anschluss antwortet.

3.3.3 Flussdiagramme

In diesem Kapitel sind die vereinfachten Grundabläufe der Programme in dem Überwachungs- und den Messmodulen aufgeführt. Abbildung 3.20 zeigt den vereinfachten Grundablauf des Programms im Messmodul und 3.21 und 3.22 den des Programms im Überwachungsmodul.

Nach der Initialisierung wartet das Messmodul als Slave auf einen Befehl des Überwachungsmoduls als Master. Sobald ein solcher eintrifft, wird die Adresse geprüft und nach Erkennen der Eigenen dieser ausgeführt und entsprechend geantwortet.

In der aktuellen Software wird das Paritätsbit einer empfangenen Nachricht nicht überprüft. Dementsprechend werden einfache Übertragungsfehler nicht erkannt.

Die Datenerfassung geschieht nach Erhalt des Befehls zum Senden der Daten.

Im Überwachungsmodul wird nach der Initialisierung auf Befehle von der externen seriellen Schnittstelle, Benutzereingaben an den Bedienelementen und Ablauf eines zyklischen Timers gewartet. Nach Erhalt eines Befehls über die externe serielle Schnittstelle wird, wie beim Messmodul auch, die Adresse geprüft. Bei der Eigenen wird der empfangene Befehl ausgeführt und entsprechend geantwortet. Sollte die empfangene Adresse nicht die Eigene sein, wird der Befehl mit Adresse ohne Veränderung an die Messmodule weitergeleitet. Wenn ein Messmodul antwortet, wird die erste empfangene Nachricht an zurück über die externe serielle Schnittstelle übertragen. Andernfalls wird nur kommuniziert, dass kein Messmodul geantwortet hat.

Sobald der zyklische Timer abläuft, werden die Daten aller Messmodule nacheinander angefragt und anschließend der LCD aktualisiert. Wenn die Flagge entsprechend gesetzt ist, werden diese aktuellen Daten in Tabellenform als ASCII-Zeichenkette über die externe serielle Schnittstelle übertragen.

Bei Änderung der Bedienelemente am Überwachungsmodul wird der LCD ebenfalls aktualisiert.

Der genaue Ablauf und Informationen zu den unterliegenden Funktionen können den angehängten, kommentierten Code-Dateien im Anhang dem Kapitel A.2 entnommen werden.

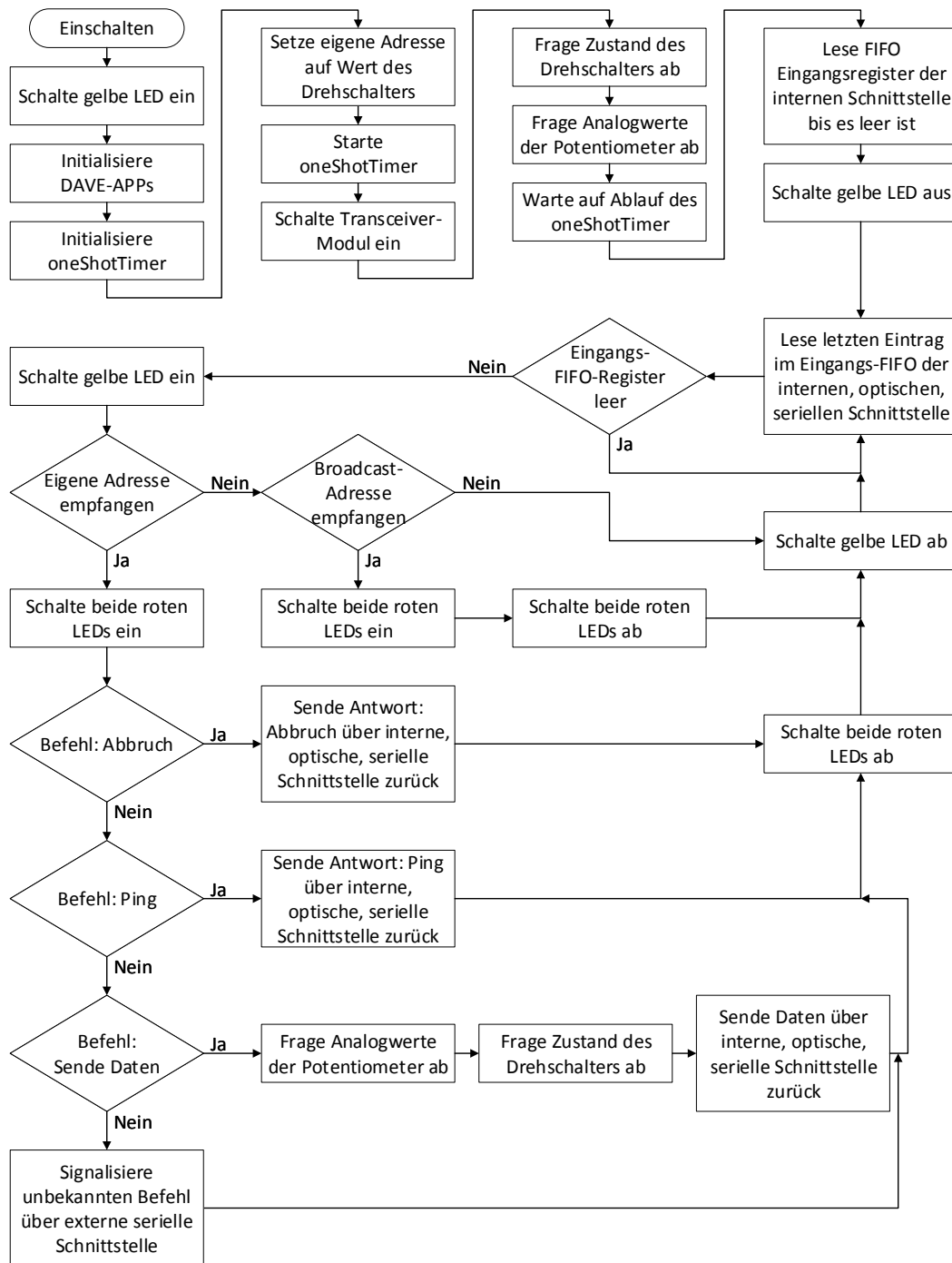


Abbildung 3.20: Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Messmodul als Slave.

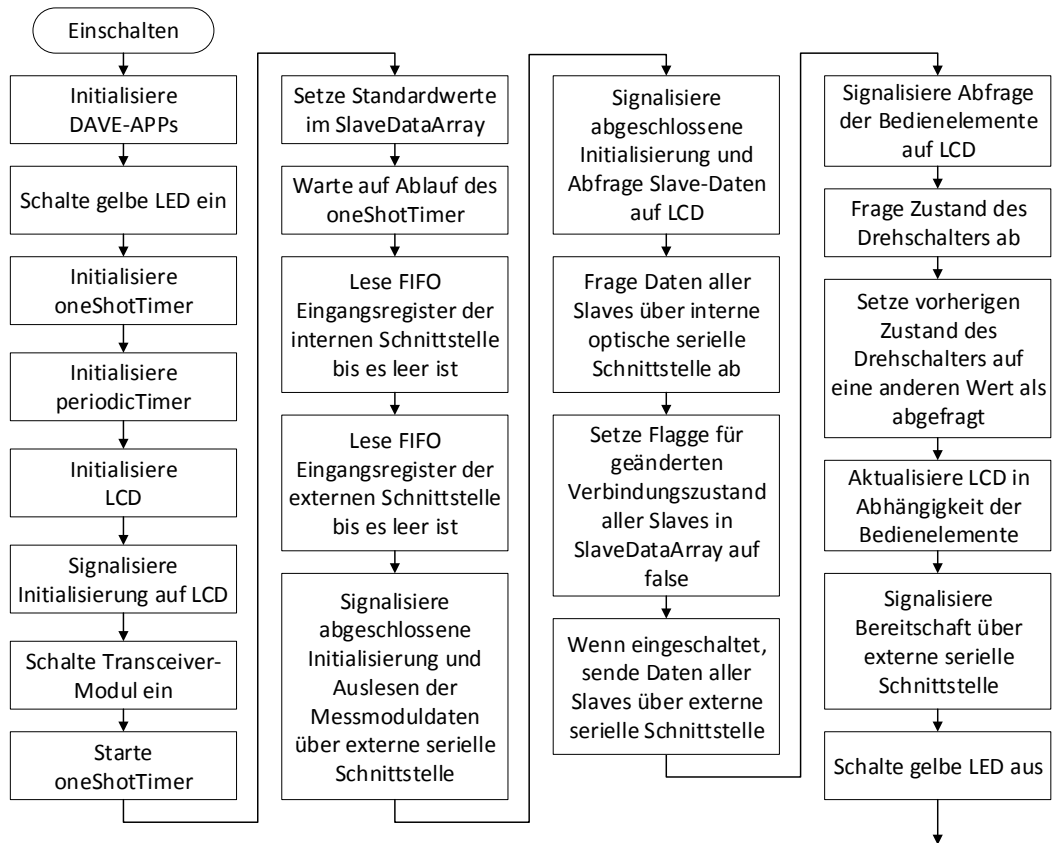


Abbildung 3.21: Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Überwachungsmodul als Master Teil 1.

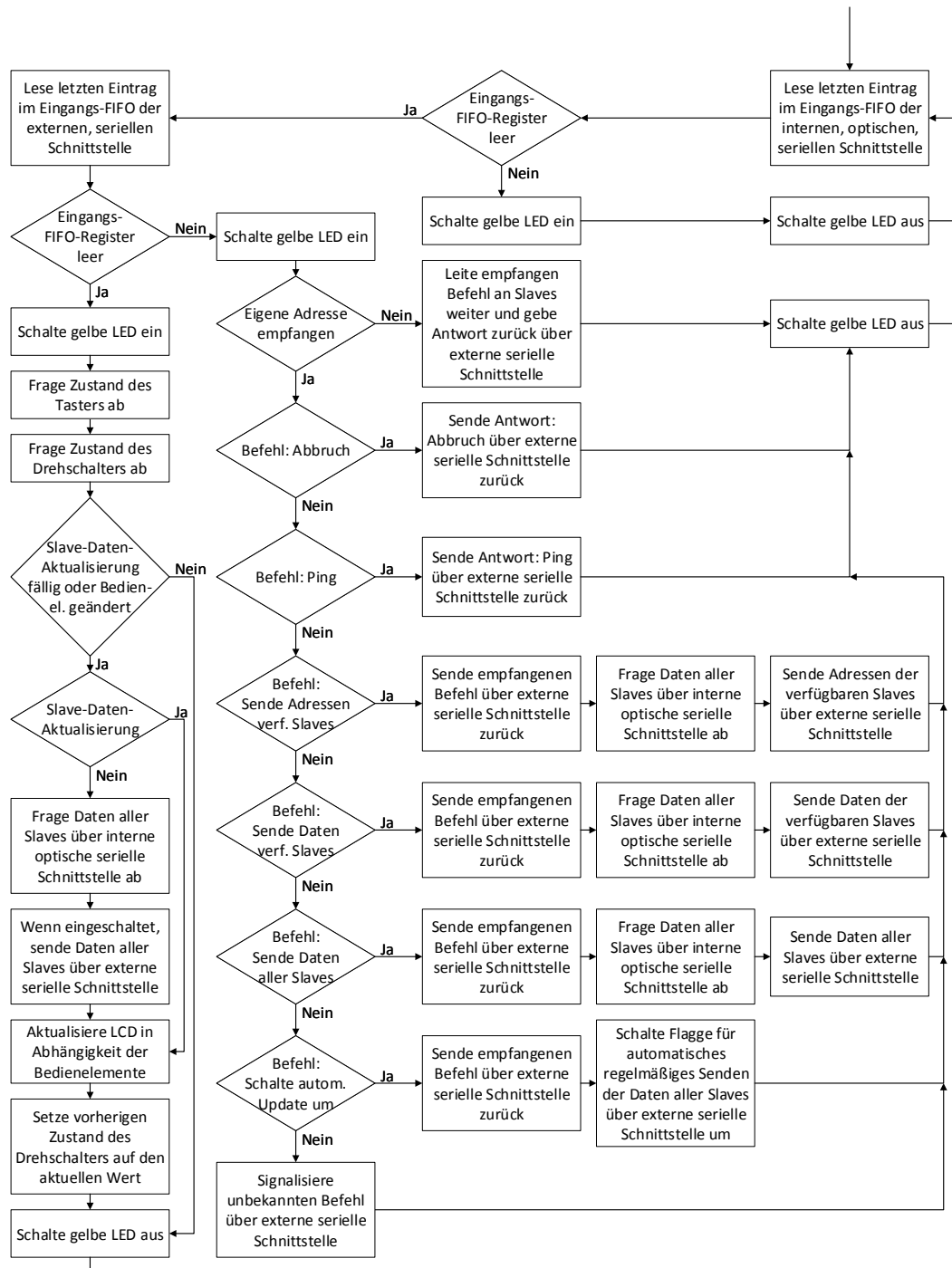


Abbildung 3.22: Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Überwachungsmodul als Master Teil 2.

3.3.4 Kommunikationsabläufe

Zur Beschreibung der Kommunikationsabläufe wurden Kommunikationsdiagramme erstellt, welche dem Programmablauf entsprechen.

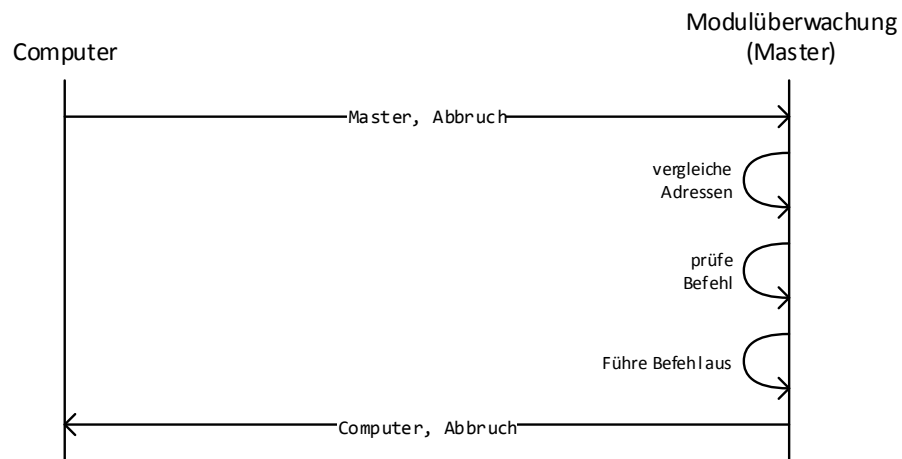


Abbildung 3.23: Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl *Abbruch*.

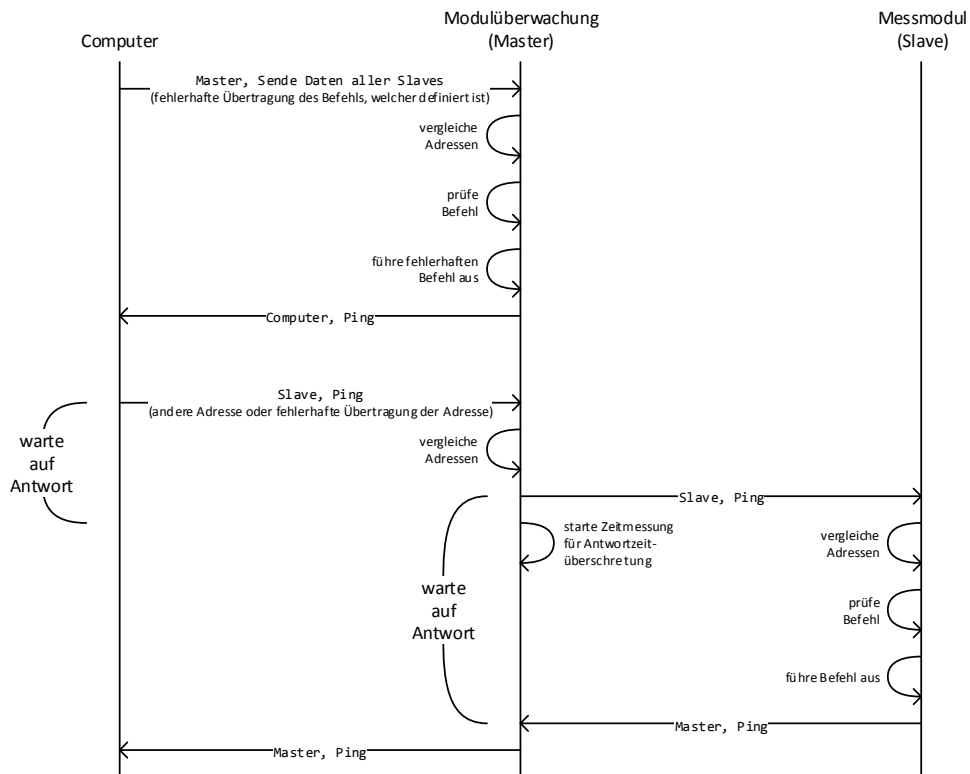


Abbildung 3.24: Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Kommunikationsfehlern oder absichtlich anderer Adresse.

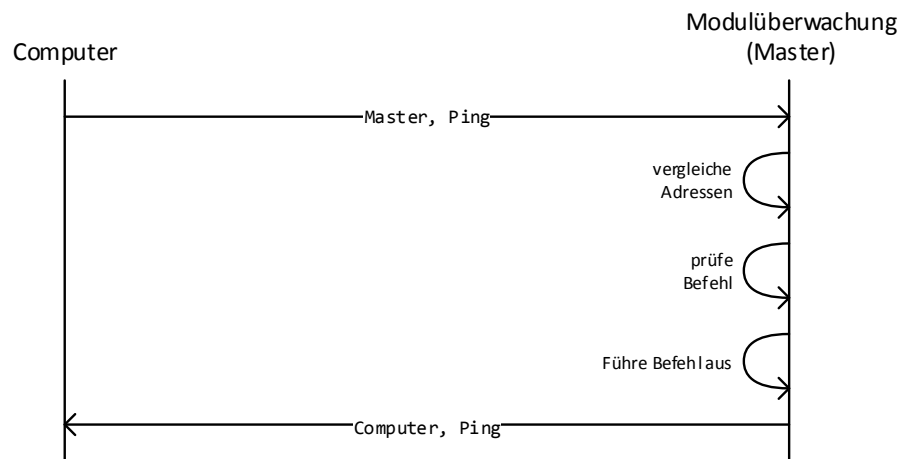


Abbildung 3.25: Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl *Ping*.

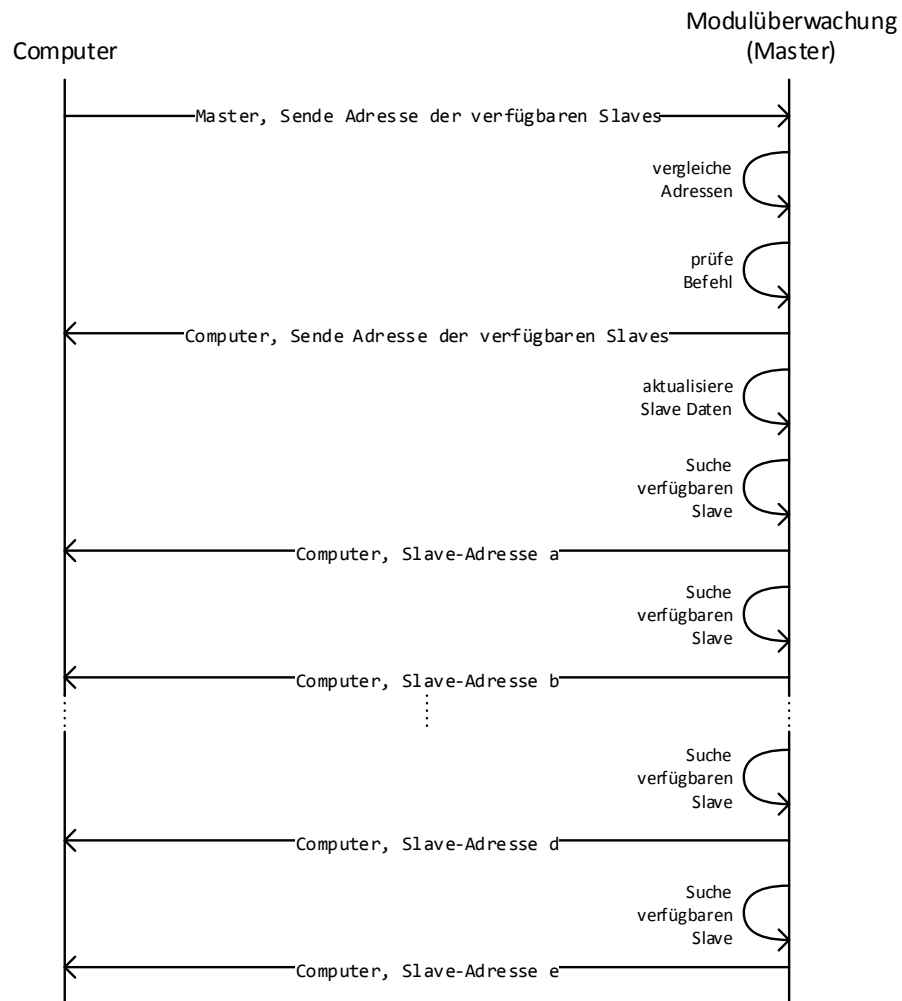


Abbildung 3.26: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl *Sende Adressen aller verfügbaren Slaves*.

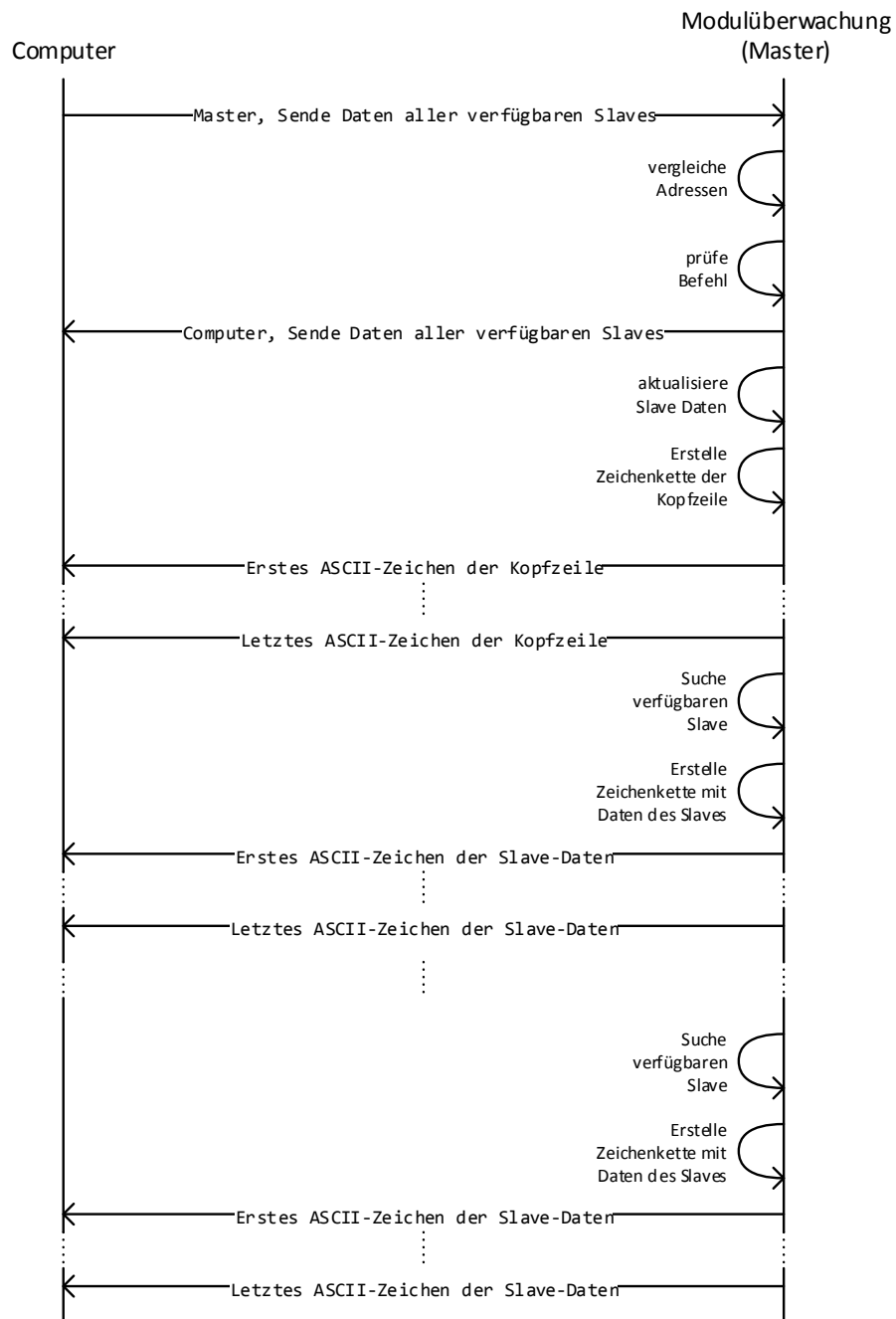


Abbildung 3.27: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl *Sende Daten aller verfügbaren Slaves*.

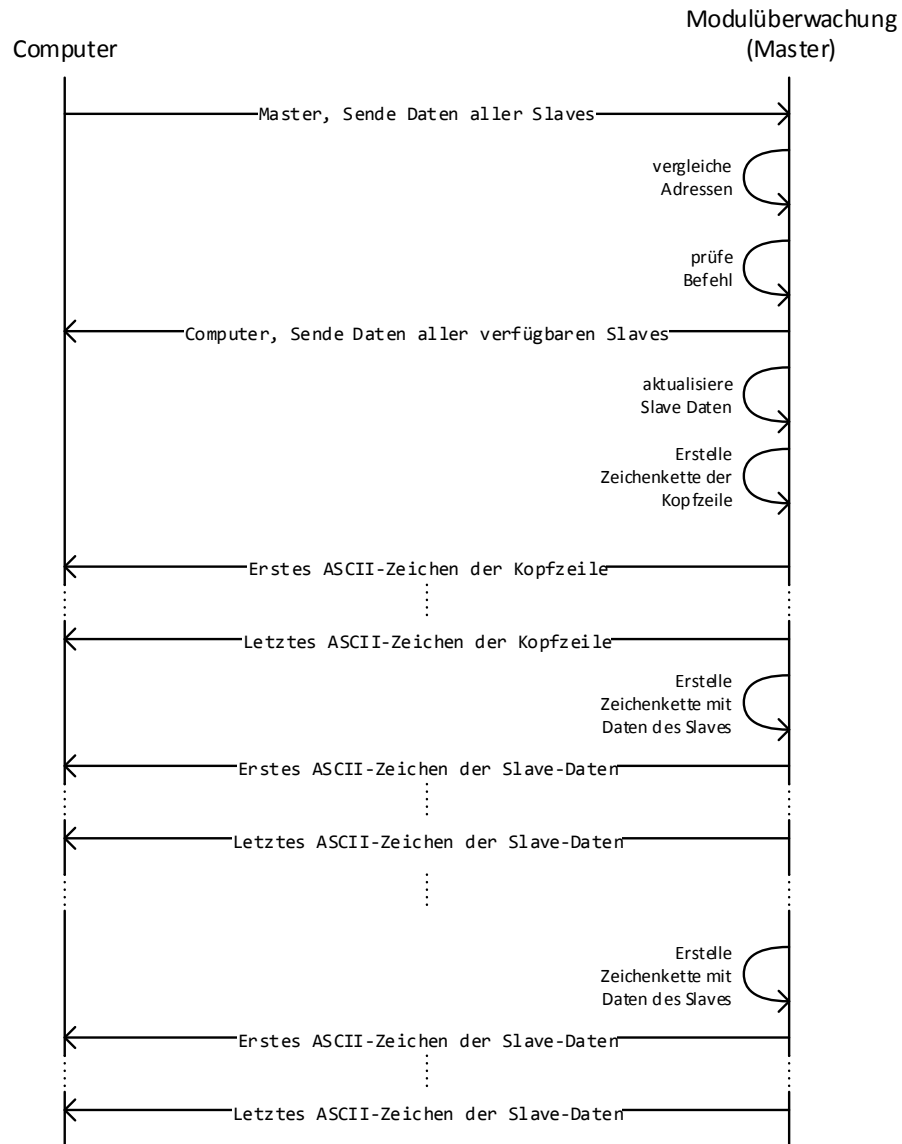


Abbildung 3.28: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl *Sende Daten aller Slaves*.

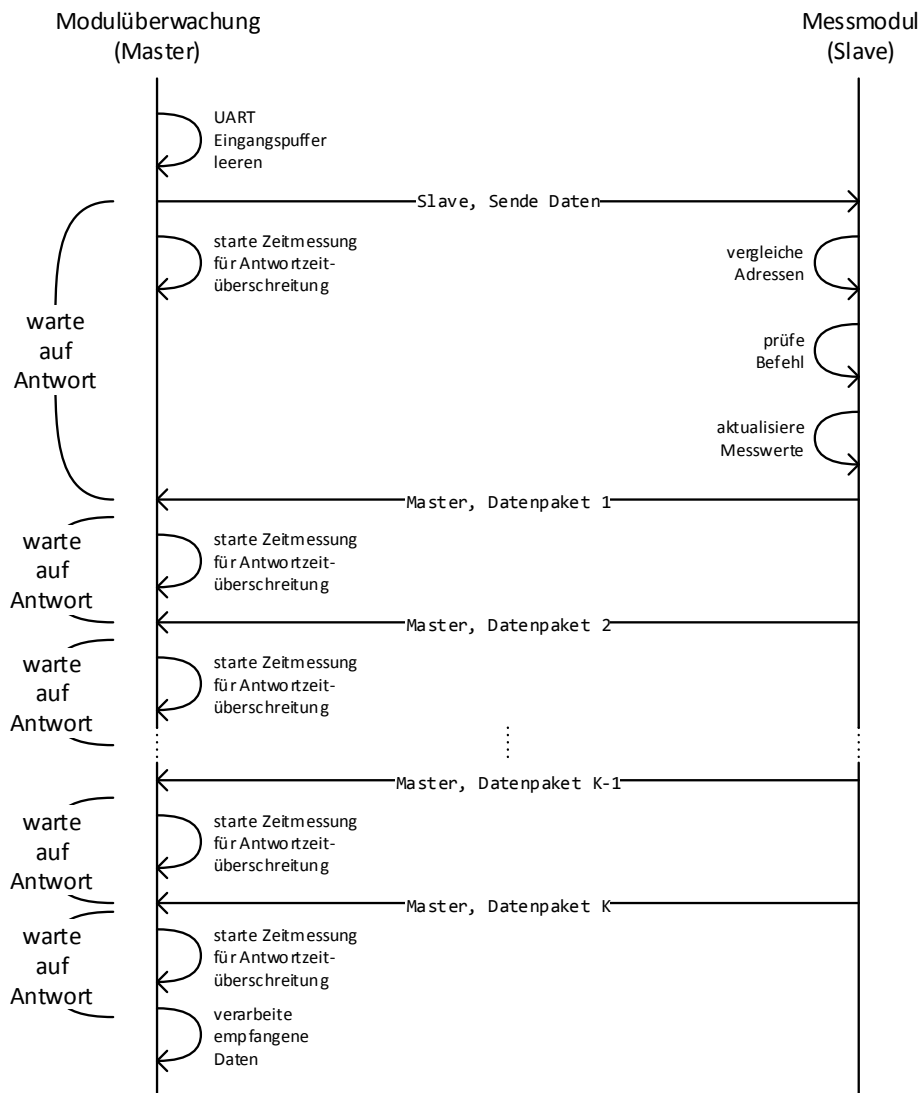


Abbildung 3.29: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl *Sende Daten*.

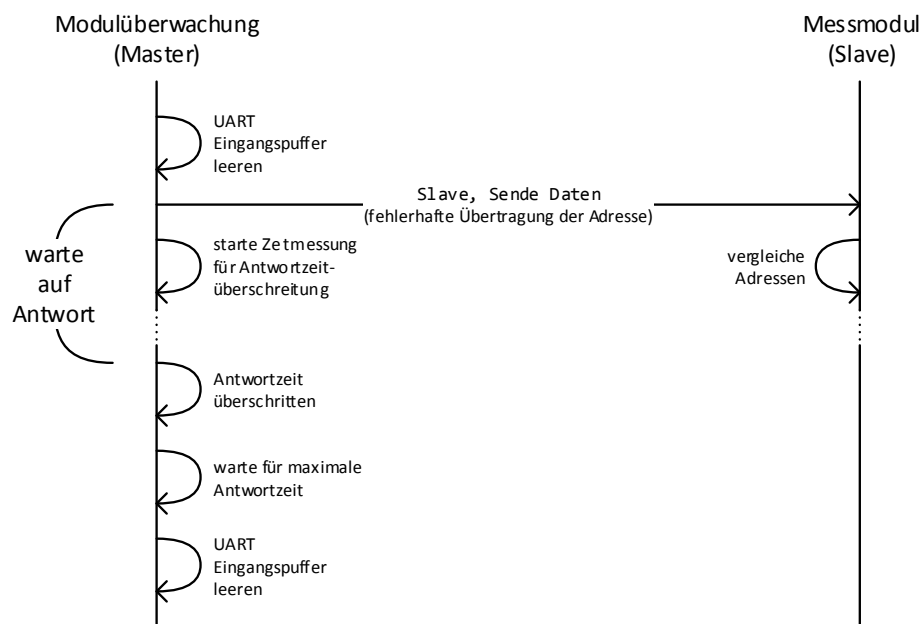


Abbildung 3.30: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl *Sende Daten* mit Fehler bei Adressübertragung.

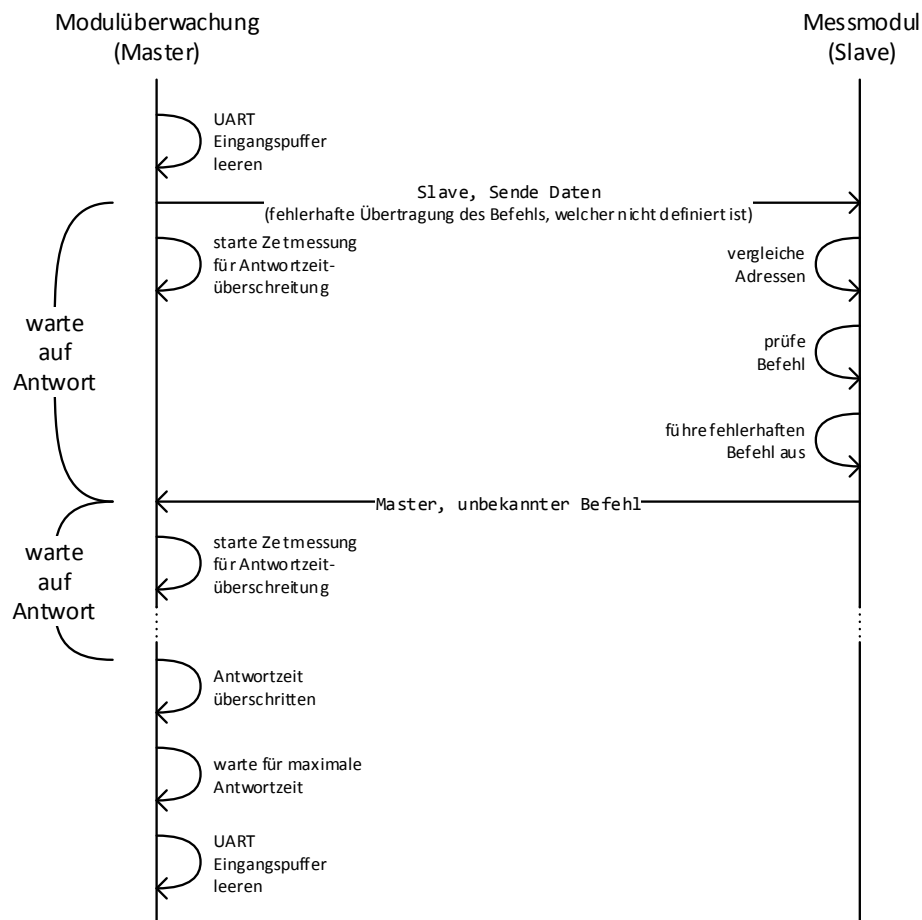


Abbildung 3.31: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl `Sende Daten` mit Übertragungsfehler beim Befehl (bekannt).

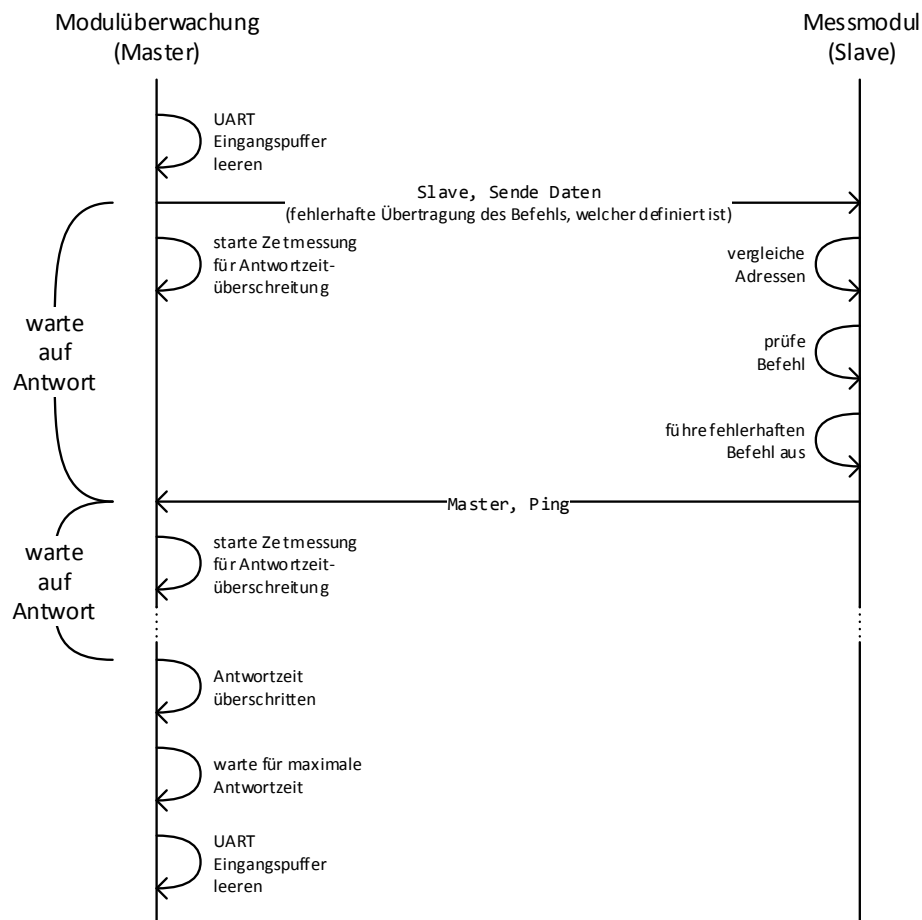


Abbildung 3.32: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Übertragungsfehler beim Befehl (unbekannt).

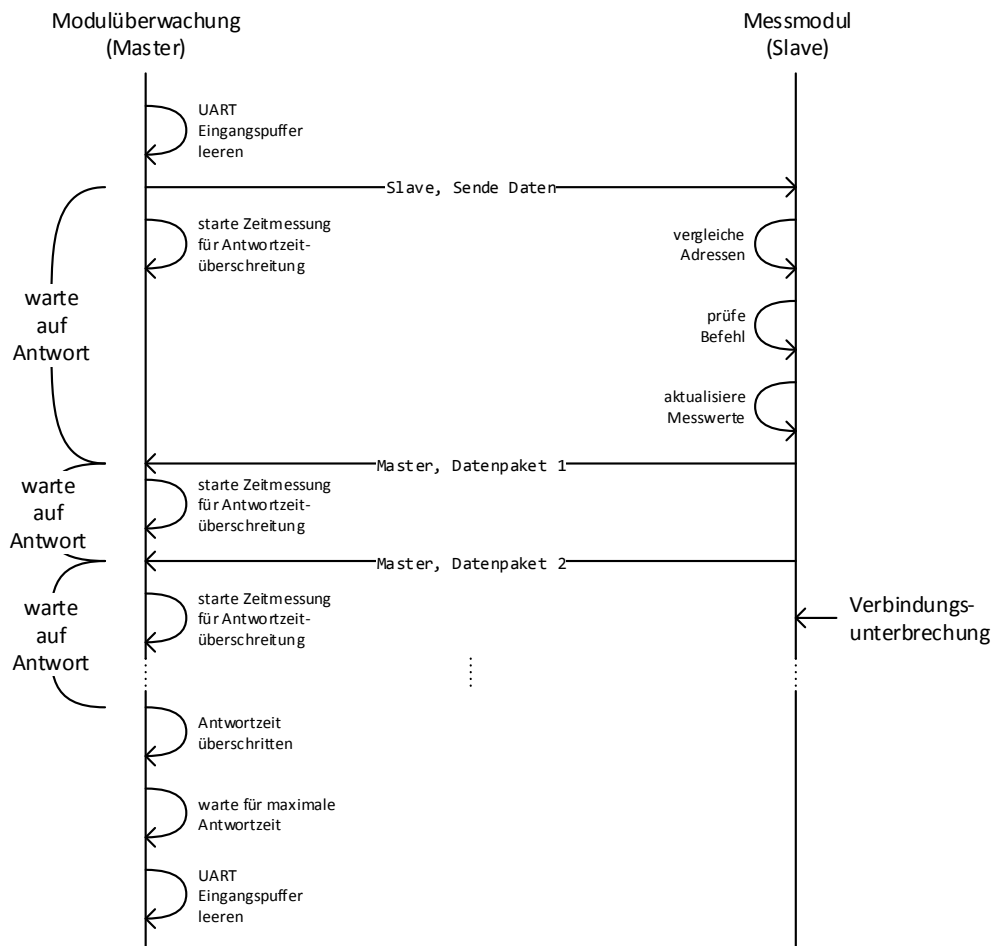


Abbildung 3.33: Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Verbindungsunterbrechung.

3.3.5 Bedienung

Folgend wird beschrieben, wie das Überwachungsmodul und die Messmodule bedient sowie letztere adressiert werden können und was die verschiedenen LED und der LCD anzeigen.

Messmodule

Beim Start der Messmodule ist es erforderlich, dass eine gültige Adresse aus dem in Kapitel 2.3 in Tabelle 2.3 definierten Bereich am Drehschalter ausgewählt ist. Dieser Wert wird als eigene Adresse eingestellt. Liegt der ausgewählte Wert außerhalb des definierten Bereichs, blinken die roten LED der Mikrocontrollerplatine im Wechsel. In diesem Fall muss eine gültige Adresse ausgewählt und durch Drücken des Tasters bestätigt werden. Nach der Initialisierung ist für den Drucktaster keine Funktion implementiert.

Sobald ein Messmodul eine Übertragung empfangen hat, leuchtet die gelbe LED bis diese fertig verarbeitet wurde. Wenn dabei die eigene Adresse erkannt wird, leuchten beide roten LED der Mikrocontrollerplatine ebenfalls.

Durch Verstellen des Drehschalters und der Potentiometer können die zugehörigen übertragenen Werte verändert werden.

Die grüne LED der Mikrocontrollerplatine dient zur Anzeige einer vorhandenen Spannungsversorgung und blinkt solange keine Computer mit installiertem Treiber am USB-Mikro-B-Anschluss verbunden ist [14, S. 9].

Bedienung des Überwachungsmoduls über serielle Schnittstelle

Zur Bedienung des Überwachungsmoduls über die UART-Schnittstelle kann ein Computer mit einer entsprechender Software an den USB-Mikro-B-Anschluss des on-board-Debuggers [17, S. 9, S. 13] angeschlossen werden. Die Übertragung erfolgt mit einer Datenrate von 115200 Bit/s und der Einstellung *8E1* nach DPS-Notation. *8* steht für die Anzahl der Datenbits, *E* für gerade Parität und *1* für die Anzahl der Stoppbits. Übertragen wird zuerst das niederwertigste Datenbit. Wie bei der Kommunikation mit den Messmodulen, besteht ein Datenpaket aus der Zieladresse und dem Befehl. Die Adresse besteht ebenfalls aus vier Bit und ist für das Überwachungsmodul auf 0b 0000 gesetzt. Die Adresse des Kommunikationspartners ist 0b 0001.

Das Überwachungsmodul wird auf einen Befehl zuerst mit einem Datenpaket bestehend aus der Adresse des Kommunikationspartners und dem empfangenen Befehl antworten. Ausgenommen sind hier die Befehle *Abbruch* und *Ping* sowie bei unbekanntem Befehl oder neuem Befehl, während eine Aktion ausgeführt wird. Weitere Informationen sind im Kapitel 3.3.2 in Tabelle 3.15 zu finden.

Bei Befehlen mit einer Antwort in Form einer ASCII-Zeichenkette wird keine Adresse übertragen und die empfangenen Datenpakete können als ASCII-Zeichen interpretiert werden.

Nach der Initialisierung wird das Überwachungsmodul seine Bereitschaft durch die Übertragung der ASCII-Zeichenkette `\nReady.\n` mitteilen. Anschließend können die Befehle aus Tabelle 3.15 in Kapitel 3.3.2 übertragen werden.

Bedienung des Überwachungsmoduls über Bedienelemente und LCD

Zur Bedienung des Überwachungsmoduls unabhängig von einem seriellen Kommunikationspartner, wurde eine Benutzeroberfläche für den eingebauten LCD erstellt. Je nach Drehschalterstellung wird eine Gesamt- oder Detailansicht aller beziehungsweise eines Messmoduls angezeigt. Die Funktionen der 16 Drehschalterstellungen sind in Tabelle 3.17 erläutert. Bei jeglicher Aktivität des Überwachungsmoduls leuchtet die gelbe LED.

Tabelle 3.17: Benutzeroberflächenfunktionen

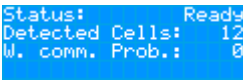
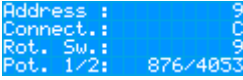


Dreh- schalter- stellung	Bildschirm	Beschreibung
0		Angezeigt wird der Systemstatus, die Anzahl an Slaves, welche die letzte Anfrage des Masters beantwortet haben und die Anzahl von diesen mit Kommunikationsproblemen.
1...12		In dieser Detailanzeige des ausgewählten Slaves wird die Adressen, Verbindungszustand, Dreh- schalterstellung und die Analog-to-Digital Converter (ADC)-Werte der zwei Potentiometer angezeigt. Informationen zu den Abkürzungen des Verbindungszustands sind in Tabelle 3.18 zu finden.
13...14		Für diese Schalterstellungen ist noch keine Funktion implementiert.
15		Auf diesem Bildschirm sind die Verbindungszustände aller Slaves zu sehen. Beim Drücken des Tasters wird die dezimale Adresse der Slaves anstelle des Verbindungszustandes angezeigt. Informationen zu den Abkürzungen sind in Tabelle 3.18 zu finden.

Tabelle 3.18: Bedeutung der Abkürzungen für den Verbindungszustand

Abkürzung	Bedeutung	Beschreibung
C	Connected	Der Slave hat die letzte Anfrage des Masters beantwortet und es wurden alle Datenpakete ohne die maximale Anzahl an Kommunikationsversuchen zu überschreiten übertragen.
ERR C	ERRor, Connected	Der Slave hat die letzte Anfrage des Masters beantwortet, allerdings wurden nach der maximalen Anzahl an Kommunikationsversuchen nicht alle Datenpakete übertragen.
LST	LoST	Der Slave hat die vorletzte Anfrage des Masters noch beantwortet, allerdings nicht mehr die Aktuelle.
NC	Not Connected	Der Slave hat mindestens die letzten zwei Anfragen des Masters nicht beantwortet.
NW	NeW	Der Slave antwortete auf die aktuelle Anfrage, allerdings nicht auf die Vorherige(n).

4 Test der Komponenten und Erprobung des Demonstrationsaufbaus

4.1 Optik

In diesem Kapitel werden die durchgeführten Tests und deren Ergebnisse im Bereich der Optik beschrieben.

4.1.1 Funktionsfähigkeit ohne Lichtleitkörper

Zur Prüfung in wie weit der Demonstrationsaufbau ohne Lichtleitkörper funktionsfähig ist, wurden die folgenden Tests durchgeführt. Alle Slaves und der Master werden über die USB-Hubs und ein *47597 GEA-005* USB-Steckernetzteil des Herstellers Unitech versorgt.

Für den ersten Test wird der Lichtleitkörper demontiert und alle Komponenten befinden sich in ihrer vorgesehenen Position. Der Master fragt zyklisch die Daten aller Slaves nacheinander an und stellt den Verbindungsstatus auf dem LCD dar.

Ohne den Lichtleitkörper erkennt der Master nur die nächsten sechs Slaves, von den anderen wird es keine Rückmeldung empfangen. Der siebte Slave erkennt das Signal des Masters und die eigene Adresse mit dem korrekten Befehl, allerdings scheint dessen Antwort nicht den Master zu erreichen. Slave acht und neun empfangen zwar Lichtsignale, erkennen jedoch nicht ihre eigene Adresse.

Wird das Master-Transceiver-Modul um 20 mm angehoben, erkennt der achte Slave teilweise seine Adresse, allerdings nicht den Befehl.

Eine Änderung des Master-Transceiver-Modul-Winkels um etwa 5° nach unten hat keinen Einfluss auf das Ergebnis, allerdings werden nach einer Änderung des Winkels in

gleichem Maße nach oben die Slaves vier und fünf nicht mehr vom Master erkannt. Beide empfangen noch eine Nachricht, allerdings erkennt nur Slave sechs seine Adresse mit einem falschen Befehl.

Wird das Master-Transceiver-Modul um weitere 20 mm auf 40 mm über seiner vorgesehenen Position angehoben, empfängt der Master nach wie vor nur Rückmeldungen von den ersten sechs Slaves. Der siebte Slave erkennt die eigene Adresse mit dem korrekten Befehl, allerdings scheint dessen Antworten nur teilweise bei geringer Änderung des Master-Transceiver-Modul-Winkels nach unten vom Master erkannt zu werden. In beiden Winkeln erkennt der achte Slave seine Adresse und den korrekten Befehl, während der neunte Slave zwar eine Nachricht erkennt, allerdings nicht seine Adresse. Bei geringer Änderung des Master-Transceiver-Modul-Winkels nach oben wird vom Master der Verlust von Paketen für den dritten Slave sowie ein Verbindungsverlust der Slaves fünf und sechs angezeigt.

Der Einsatz eines senkrechten Glasspiegels auf der dem Master-Transceiver-Modul gegenüberliegenden Seite hinter dem letzten Slave hat nur bei angehobenem Master-Transceiver-Modul auf 20 mm einen Einfluss auf die Testergebnisse gezeigt. Hier wurde der Befehl beim achten Slave sporadisch richtig erkannt.

4.1.2 Funktionsfähigkeit mit Lichtleitkörper

Mit dem in der vorgegebenen Position verbautem Lichtleitkörper kann der Master alle Slaves erreichen und deren Antworten empfangen. Seltenen treten vereinzelt meist bei den gleichen Slaves Übertragungsfehler auf. Welche dies sind, variiert mit der Montageposition. Nach Tauschen von den Adressen der auffälligen Slaves mit näher am Master liegenden traten die Übertragungsfehler nach wie vor bei den gleichen Slaves unabhängig von der eingestellten Adresse auf.

4.1.3 Verschiebung des Lichtleitkörpers

Zum Testen des Einflusses der Verschiebung des Lichtleitkörpers auf die Erreichbarkeit der Slaves und Anzahl der Übertragungsfehler wurden alle Schrauben zur Befestigung des Lichtleitkörpers entfernt und dieser auf der Ebene um etwa 0,5 mm in alle Richtungen verschoben und soweit möglich verdreht. Dabei wurde keine Verbindung unterbrochen

und es konnte keine Änderung in der Anzahl an Übertragungsfehlern festgestellt werden.

Bei der eingestellten Sendeleistung der Transceiver-Module erwies sich der gefertigte Lichtleitkörper somit als sehr tolerant gegenüber Verschiebungen.

4.2 Hardware

In diesem Kapitel werden die durchgeführten Tests und deren Ergebnisse im Bereich der Hardware beschrieben.

4.2.1 Transceiver-Modul

Datenraten und Pulsdauerlänge

Für die Arbeiten in Kapitel 2.3 sollte die maximale übertragungsfehlerfreie Datenrate ermittelt werden. Für diese Tests wurden zwei der Aufbauten mit Adapterplatinen aus Kapitel 3.2.2 auf je einer Steckplatine platziert. Die Mikrocontrollerplatine *XMC4700 Relax Kit V1* wurde verwendet, um das sendende Transceiver-Modul und dessen Sendediode zu versorgen und die Daten 0b 0000 0000 zu übertragen, damit zehn aufeinanderfolgende Lichtpulse an das empfangende Transceiver-Modul, versorgt durch einen *XMC 2Go XMC1100 V1*, übermittelt wurden. Nach der Angabe des Widerstandswertes für Niederleistungsbetrieb im Datenblatt [34, S. 6], wurde die Sendediode des sendenden Transceiver-Moduls über einen Widerstand von 56Ω versorgt.

Als Ausgangspunkt zur Ermittlung der maximalen übertragungsfehlerfreien Datenrate wurden 250 kBit/s ausgewählt, da die Empfangssignalpulsdauer von maximal $3 \mu\text{s}$ addiert mit dessen Anstiegs- und Abfallszeit von je maximal 100 ns [34, S. 5] weniger als die resultierende Bitdauer von 4μ beträgt. Bei einer Sendesignalpulsdauereinstellung von $8/16$ der Bitdauer ist der Lichtpuls $2 \mu\text{s}$ lang. Dies liegt innerhalb der Spezifikation von mindestens $1,2 \mu\text{s}$ und maximal $50 \mu\text{s}$.

Die Messung von Sende- und Empfangssignal des sendenden Transceiver-Moduls, Kathodenspannung der Sendediode und Empfangssignal des empfangenden Transceiver-Moduls mit einem *Tektronix MSO3034* Oszilloskop ergab die in Abbildung 4.1 zu sehenden Spannungsverläufe.

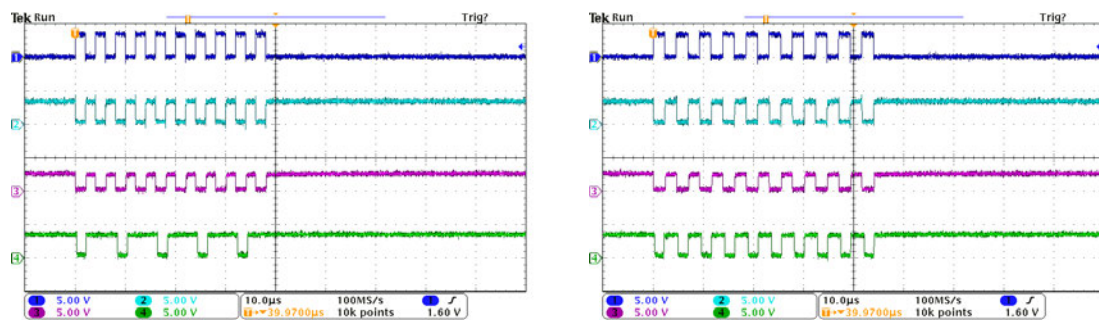


Abbildung 4.1: Vergleich des Signalverlaufs einer Übertragung bei 250000 Bit/s (links) und 216001 Bit/s (rechts). Die Dauer eines Sendesignalpulses beträgt in beiden Bildern 8/16 der eines Bits. Kanal 1 misst den Sendesignalverlauf, Kanal 2 den Empfangssignalverlauf und Kanal 3 die Kathodenspannung der Sendediode. Alle drei Kanäle werden am sendenden Transceiver-Modul gemessen. Der Kanal 4 misst den Empfangssignalverlauf des empfangenden Transceiver-Moduls. Es ist zu erkennen, dass bei der höheren Datenrate (links) nur bei jedem zweiten Licht- ein Empfangssignalspuls ausgegeben wird und somit fehlerhafte Daten empfangen werden. Mit der niedrigeren Datenrate (rechts) gibt es für jeden Licht- einen Empfangssignalspuls, womit alle Daten korrekt empfangen werden.

Bei einer Datenrate von 250000 Bit/s ist am Spannungsverlauf der Sendediodenkathode des sendenden Transceiver-Moduls zu erkennen, dass diese einschaltet, allerdings wird nur nach jedem zweiten Lichtpuls ein Empfangssignalspuls am empfangenden Transceiver-Modul ausgegeben. Dieser Test wurde mit verschiedenen Datenraten und Sendesignalspulsdauern wiederholt. Die Ergebnisse sind in Tabelle 4.1 zu finden.

Zunächst wurde die Datenrate 250000 Bit/s getestet, was Fehler in der Übertragung ergab. Nach einer Reduzierung auf 200000 Bit/s traten keine Fehler mehr auf. Mit einer auf den Mittelwert von 225000 Bit/s angehobenen Datenrate traten wieder Fehler auf. Nach einer Reduzierung auf 215010 Bit/s war die Übertragung erneut fehlerfrei. Diese Datenrate wurde dann in etwa 1 kBit/s Schritten angehoben, bis Fehler auftraten, was schon nach der zweiten Erhöhung bei 217022 Bit/s der Fall war. Somit ergab sich die höchste Datenrate ohne Fehler im Empfangssignal zu 216008 Bit/s.

In weiteren Tests, deren Ergebnisse ebenfalls in Tabelle 4.1 zu finden sind, zeigte sich, dass die Sende- und somit Lichtsignalspulsdauer keinen Einfluss auf das Empfangsergebnis am empfangenden Transceiver-Modul hat. In Abbildung 4.2 ist zu sehen, dass die gleichen Übertragungsfehler auch mit einer halbierten Lichtsignalspulsdauer auftreten. Die Sende-

Datenrate [Bit/s]	Sendsignal- pulsdauer [Anteil der Bit- dauer]	Ergebnis
250000	2/8	Jedes zweite Bit wurde nicht ausgegeben.
250000	4/8 und 8/16	Jedes zweite Bit wurde nicht ausgegeben.
225000	8/16	Das 3., 5., 7, und 9. Bit wurde nicht ausgegeben.
220000	8/16	Das 4., 6., 8, und 10. Bit wurde nicht ausgegeben.
220001	4/16	Das 4., 6., 8, und 10. Bit wurde nicht ausgegeben.
217022	8/16	Einzelne Bits wurden sporadisch nicht ausgegeben.
216008	8/16	Alle Bits wurden am Ausgang ausgegeben.
215010	8/16	Alle Bits wurden am Ausgang ausgegeben.
200000	7/8	Alle Bits wurden am Ausgang ausgegeben.
200000	6/8	Alle Bits wurden am Ausgang ausgegeben.
200000	4/8 und 8/16	Alle Bits wurden am Ausgang ausgegeben.
200000	2/8	Alle Bits wurden am Ausgang ausgegeben.

Tabelle 4.1: Liste der getesteten Datenraten mit den zugehörigen Sendsignalpulsdauern als Anteil der Bitdauer und das Testergebnis.

signalpulsdauer liegt dabei mit $1,136 \mu\text{s}$ etwas unterhalb der in Kapitel 2.3 definierten minimalen von $1,2 \mu\text{s}$.

Zuletzt wurde bei allen getesteten Einstellungen eine gleichbleibende Dauer der Empfangssignale von etwa $1,73 \mu\text{s}$ gemessen.

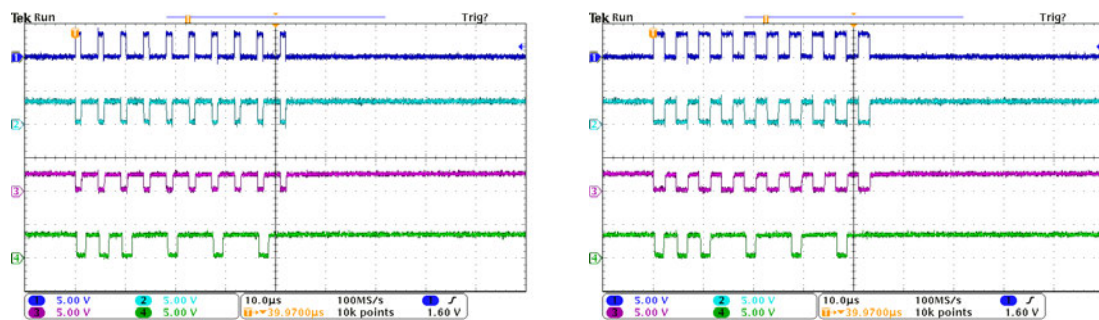


Abbildung 4.2: Vergleich des Signalverlaufs einer Übertragung bei 220001 Bit/s mit unterschiedlichen Sendesignalpulsängen. Die Dauer eines Sendesignalpulses beträgt im linken Bild $4/16$ und im rechten $8/16$ der eines Bits. Kanal 1 misst den Sendesignalverlauf, Kanal 2 den Empfangssignalverlauf und Kanal 3 die Kathodenspannung der Sendediode. Alle drei Kanäle werden am sendenden Transceiver-Modul gemessen. Der Kanal 4 misst den Empfangssignalverlauf des empfangenden Transceiver-Moduls. Der zeitliche Abstand zwischen Abfällen des Lichtpulses und Wiederanstieg des selben scheint keinen Einfluss auf die Übertragungsfehler zu haben, da bei beiden Konfigurationen die gleichen Übertragungsfehler auftreten.

Kennlinie der Transceiver-Modul-Sendediode

Zur Auswahl eines Vorwiderstandes für die Sendediode des Infrarot-Transceiver-Moduls *TFDU4101* des Herstellers *Vishay Intertechnology, Inc.* in Kapitel 3.2.1 wurde deren Kennlinie aufgenommen. Die dabei zu berücksichtigten Testbedingungen waren, dass die Versorgungsspannung der Sendediode zwischen $-0,5\text{ V}$ und 6 V bleibt sowie ein Dauerstrom von maximal 80 mA fließen darf [34, S. 3], was höher als der verfügbare Strom des Linearspannungsreglers *IFX54211MB V33* auf den Mikrocontrollerplatinen *XMC 2Go XMC1100 V1* ist, weshalb eine Messung bis zu diesem Strom ausreicht. Bei einer Messung mit höheren Strömen wäre zu beachten, dass ein Strom von maximal 400 mA nur für weniger als $90\text{ }\mu\text{s}$ und mit einem Tastverhältnis von weniger als 20% [34, S. 3] fließen darf.

Als Versorgung diente ein *PeakTech 6225 A* Labornetzteil, zum Messen der Sendediodenspannung ein *Voltcraft VC135* Multimeter und für das Messen des Sendediodenstroms ein *Unitec 45647* Multimeter. Die Messung ergab die in Tabelle 4.2 aufgelisteten Werte und die in Abbildung 4.3 zu sehende Kennlinie. Zusätzlich zur gemessenen Sendediodenspannung und -strom wurden die Werte für Sendediodenleistung, Vorwiderstandsspannung, Vorwiderstandswert und Vorwiderstandsleistung berechnet, welche dort ebenfalls

eingetragen sind. Für die Berechnung des Vorwiderstandswertes wurde eine Versorgungsspannung der Sendediode am Anschlusspin 1 (V_{CC2} oder *IRED anode*) des Transceiver-Moduls von $U_{CC2} = 3,32\text{ V}$ angenommen.

Ver- sorgungs- spannung U_{IRED} [V]	Dioden- strom I_{IRED} [mA]	Dioden- leistung P_{IRED} [mW]	Vor- wider- stands- spannung U_{RIRED} [V]	Vor- wider- stands- wert R_{IRED} [Ω]	Vor- wider- stands- leistung P_{RIRED} [mW]
1,100	0,032	0,035	2,22	70476,2	0,1
1,200	0,626	0,751	2,12	3386,6	1,3
1,300	14,80	19,24	2,02	136,5	29,9
1,325	24,2	32,065	1,995	82,4	48,3
1,335	30,7	40,985	1,985	64,7	60,9
1,340	32,8	43,952	1,98	60,4	64,9
1,345	38,3	51,514	1,975	51,6	75,6
1,350	40,0	54	1,97	49,3	78,8
1,355	45,0	61,029	1,965	43,6	88,5
1,400	90,4	126,56	1,92	21,2	173,6

Tabelle 4.2: Messwerte der Sendediodenkennlinie eines Transceiver-Moduls. Zusätzlich zur gemessenen Sendediodenspannung und -stromverlauf wurden die Werte für Sendediodenleistung, Vorwiderstandswert und Vorwiderstandsleistung berechnet.

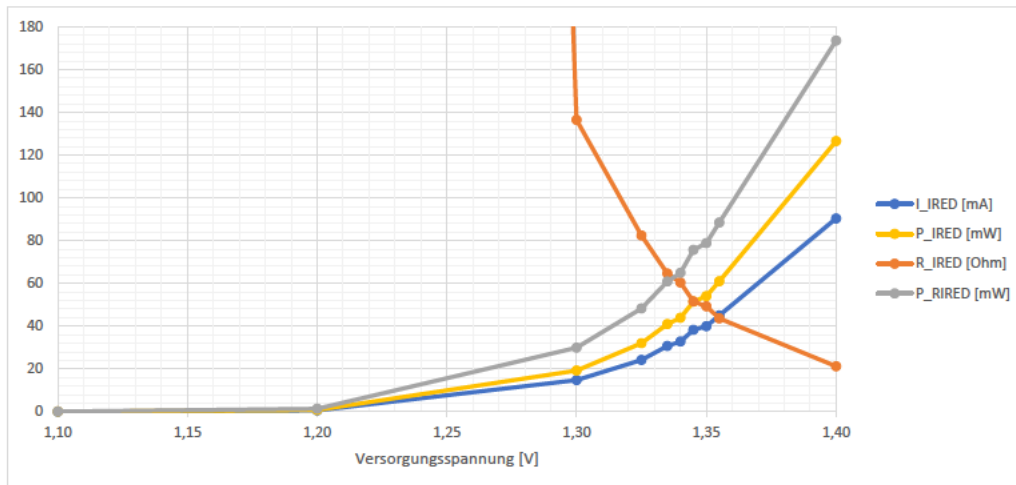


Abbildung 4.3: Kennlinie der Sendediode eines Transceiver-Moduls. Zusätzlich zum gemessenen Stromverlauf sind die berechneten Werte von Sendediodenleistung, Vorwiderstandswert und Vorwiderstandsleistung eingezeichnet.

4.2.2 Fertigungstoleranzen

Lichtleitkörper

Die Messung mehrerer Befestigungspunkte, welche ein Solldurchmesser von 7 mm haben, ergab an der Oberseite 7,05 mm und an der Unterseite 7,30 mm. Mit der Materialstärke von 5,90 mm ergibt dieser Größenunterschied eine Neigung der Außenkanten von $\delta = \arctan\left(\frac{7,30\text{ mm} - 7,05\text{ mm}}{5,9\text{ mm}}\right) \approx 2,4^\circ$.

Über die gesamte Länge ist der Lichtleitkörper 327,5 mm beziehungsweise 328,1 mm lang, was im Vergleich mit der Solllänge von 327 mm einer Vergrößerung um 0,153 % beziehungsweise 0,336 % bedeutet. Diese Abweichungen sind sehr klein und haben daher voraussichtlich keinen nennenswerten Einfluss auf die Übertragung.

Versuchsaufbau

Alle Bohrungen in der Grundplatte wurden von Hand angezeichnet und gebohrt. Dadurch entstanden Positionsabweichungen von bis zu 1 mm, welche durch Aufbohren der Löcher in den mit dem 3D-Drucker hergestellten Teilen ausgeglichen wurden. Diese Teile

waren um etwa 0,2% zu klein. Bei den Maßen der Messmodulplatinen konnten keine Abweichungen von den Sollmaßen ermittelt werden.

Im Gesamtaufbau ergab sich die Distanz der äußersten Kanten von den über die lange Seite aneinandergereihten Messmodulplatinen zu den vorgegebenen 323 mm. Demnach wurden die Fertigungstoleranzen durch den geringen Abstand zwischen den Messmodulen und deren Haltern von 0,1 mm ausgeglichen. Die seitliche Abweichung wurde durch die Konstruktion allerdings nicht begrenzt und beträgt etwa $\pm 0,25$ mm.

Die Positionsabweichung der Slave-Transceiver-Module auf den Messmodulplatinen beträgt maximal 0,25 mm in beide Richtungen und das Master-Transceiver-Modul sitzt etwa 0,5 mm zu nahe am Lichtleitkörper.

4.2.3 Robustheit des Demonstrationsaufbaus

Größtenteils ist der Demonstrationsaufbau ausreichend robust, um häufig gehandhabt zu werden. Die Leitungen sind gut und kompakt unter der Grundplatte befestigt und sollten sich nicht lösen. Eine etwas stärkere Dimensionierung der USB-Hub-Halterung hätte das Herausziehen der Stecker ohne Gegenhalten ermöglicht. Auch ist das äußerste Messmodul recht nahe am Grundplattenrand befestigt, sodass kaum geschützt ist.

Das genutzte Material des Lichtleitkörpers *Makrolon* erwies sich nur begrenzt widerstandsfähig gegen Kratzer, allerdings scheint es etwas elastisch zu sein, sodass es nicht einfach bricht. Demnach sollte der Lichtleitkörper eine starre Montage auch bei Temperaturschwankungen ohne Beschädigung überstehen können.

4.2.4 Bedienbarkeit

Die Bedienelemente an den Messmodulen sind gut zugänglich und ermöglichen eine einfache Änderung der übertragenen Werte. Die am Messmodul angezeigte Information kann über den Drehschalter einfach eingestellt werden. An den versorgenden USB-Hubs lassen sich einzelne Komponenten nach Bedarf abschalten.

Zukünftige Lichtleitkörper können mit M3-Schrauben an den eingelassenen M3-Muttern einfach, sicher und präzise befestigt werden.

4.2.5 Stromaufnahme des Überwachungs- und der Messmodule

Zur Ermittlung der Stromaufnahme am USB-Anschluss der Mikrocontrollerplatinen I_{USB} wurde ein Widerstand mit einem Wert von $R_{mess} = 1 \Omega$ in die versorgende USB-Leitung integriert. Die Toleranz dieses Widerstands ist unbekannt und wird daher mit $\pm 10\%$ angenommen. Die Spannung über diesem Widerstand wurde mit einem *Tektronix MSO3034* Oszilloskop aufgenommen und mit dem Widerstandswert zum Strom umgerechnet.

Während das Messmodul sendet, sind dessen gelbe und beide roten LED eingeschaltet. Die übertragenen Daten sind unbekannt, allerdings ist bekannt, dass die Zieladresse die des Masters ist, wofür vier aufeinanderfolgende Lichtpulse übertragen werden.

Wenn das Überwachungsmodul sendet, ist ebenfalls dessen gelbe LED und LCD aktiv. Die übertragenen Daten beim aufgenommenen Stromverlaufs sind jedoch unbekannt.

In Abbildung 4.4a ist der Stromverlauf eines Messmoduls während des Sendens und in Abbildung 4.4b der des Überwachungsmoduls zu sehen.

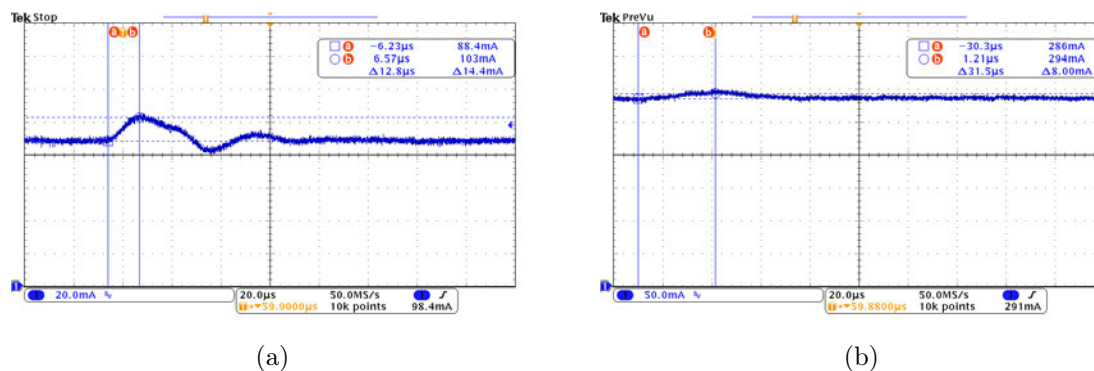


Abbildung 4.4: Simulationsergebnis des berechneten Lichtleitkörpers. Die türkisen Punkte zeigen die Endpunkte der Streckenabschnitte des Lichtleitkörperumrisses sowie die Position der Punktlichtquellen. In den Bildern a bis d ist der Brechungsindex des Lichtleitkörpers auf den von *Plexiglas* mit 1,49 [32] und e bis h auf den von *Makrolon* 1,587 [18, S. 26] gesetzt. In a und e sind alle Transceiver-Modul-Sendedioden aktiv, in b und f die des Masters, in c und g die des ersten Slaves und in d und h die des letzten. Diese Simulation wurde mit dem zweidimensionalen Optiksimulator *Ray Optics Simulation* von ricktu288 [39] erstellt.

Wird die maximale angenommene Toleranz des Messwiderstandes von -10% in die Messwerte eingerechnet, so ergibt sich der maximale Strom des Messmodul zu $I_{LVRMAXmess} =$

$\frac{100\%}{90\%} \cdot 88,4\text{ mA} \approx 98,2\text{ mA}$ und der des Überwachungsmoduls zu $I_{LVRMAXmess} = \frac{100\%}{90\%} \cdot 294\text{ mA} \approx 327\text{ mA}$.

Der so gemessene Strom entspricht beim Messmodul dem des Linearspannungsreglers $I_{LVR} = I_{USB}$ und liegt deutlich unter dem maximalen von $I_{LVRMAX} = 150\text{ mA}$ [15, S. 2] [14, S. 9], sodass dieser nicht überlastet wird.

Beim Strom des Überwachungsmoduls ist der des LCD I_{LCD} ebenfalls enthalten, sodass der Gesamtstrom $I_{USB} = I_{LVR} + I_{LCD}$ nicht dem des Linearspannungsreglers I_{LVR} entspricht. Da der ermittelt Strom jedoch weit unter dem maximalen Strom des Linearspannungsreglers von $I_{LVRMAX} = 1\text{ A}$ [13, S. 1] liegt, kann sicher gesagt werden, dass dieser ebenfalls nicht überlastet wird.

4.2.6 Messgenauigkeit der Analog-zu-Digital-Wandler

Die Ermittlung der Messgenauigkeit der ADC entfiel, da diese hier nur zur Generierung eines zu übertragenen Wertes aus den Potentiometern dienen und daher keine hohe Genauigkeit erforderlich ist.

4.3 Software

In diesem Kapitel werden die durchgeführten Tests und deren Ergebnisse im Bereich der Software beschrieben.

4.3.1 Kommunikation zwischen dem Überwachungs- und den Messmodulen

Die Kommunikation zwischen dem Überwachungsmodul als Master und den Messmodulen als Slave funktioniert mit nur wenigen Übertragungsfehlern. Fehlende Pakete und verlorene, sowie neue Verbindungen werden vom Überwachungsmodul erkannt.

In Videoaufnahmen in Zeitlupe ist zu erkennen, wie die gelbe LED aller Messmodule gleichzeitig einschaltet und die Roten entsprechend der eingestellten Adresse nacheinander. Dies zeigt, dass alle Messmodule allen Datenverkehr und ihre eigenen Adresse erkennen.

Zur Überprüfung des Signalverlaufs wurden zunächst die Daten 0b 0000 0000 für zehn aufeinanderfolgende Sendesignalpulse vom Master an den Slave übertragen und die Send- und Empfangssignale beider Transceiver-Module mit einem *Tektronix MSO3034* Oszilloskop aufgenommen. Es ergab sich der Verlauf in Abbildung 4.5.

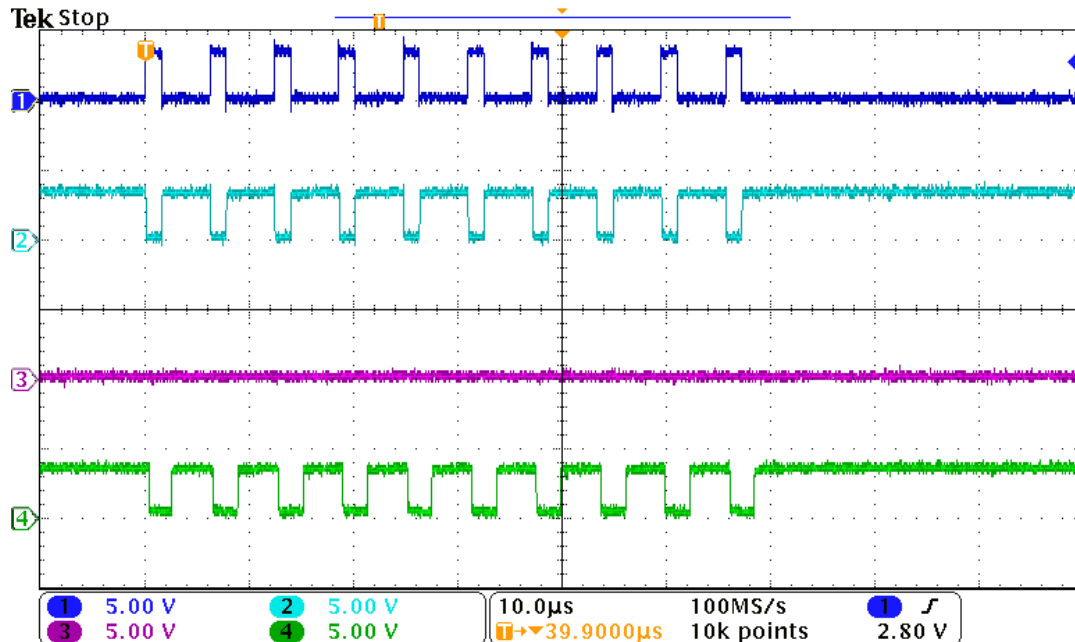


Abbildung 4.5: Signalverlauf einer Übertragung mit 161718 Bit/s. Die Dauer eines Sendesignalpulses beträgt in beiden Bildern 4/16 der eines Bits. Kanal 1 misst den Sendesignalverlauf und Kanal 2 den Empfangssignalverlauf des Überwachungsmoduls. Kanal 3 misst den Sendesignalverlauf und Kanal 4 den Empfangssignalverlauf eines Sendemoduls. Es ist zu erkennen, dass es für jeden Licht- einen Empfangssignalpuls gibt, womit alle Daten korrekt empfangen werden.

Die Sendesignalpulsdauer wurde bei beiden Mikrocontrollerplatinen mit $1,55 \mu\text{s}$ gemessen. Die Empfangssignalpulsdauer unterschied sich etwas. Bei dem Transceiver-Modul des Überwachungsmoduls betrug sie $2,32 \mu\text{s}$ und bei dem des Messmoduls $2,16 \mu\text{s}$.

Ein vollständiger Kommunikationsablauf zwischen dem Überwachungsmodul und einem Messmodul ist in Abbildung 4.6 gezeigt.

Nach der Anfrage des Überwachungsmoduls sind die sieben Antwortpaket zu sehen. Der gesamte Datenaustausch nimmt eine Zeit von etwa $0,78 \text{ ms}$ in Anspruch.

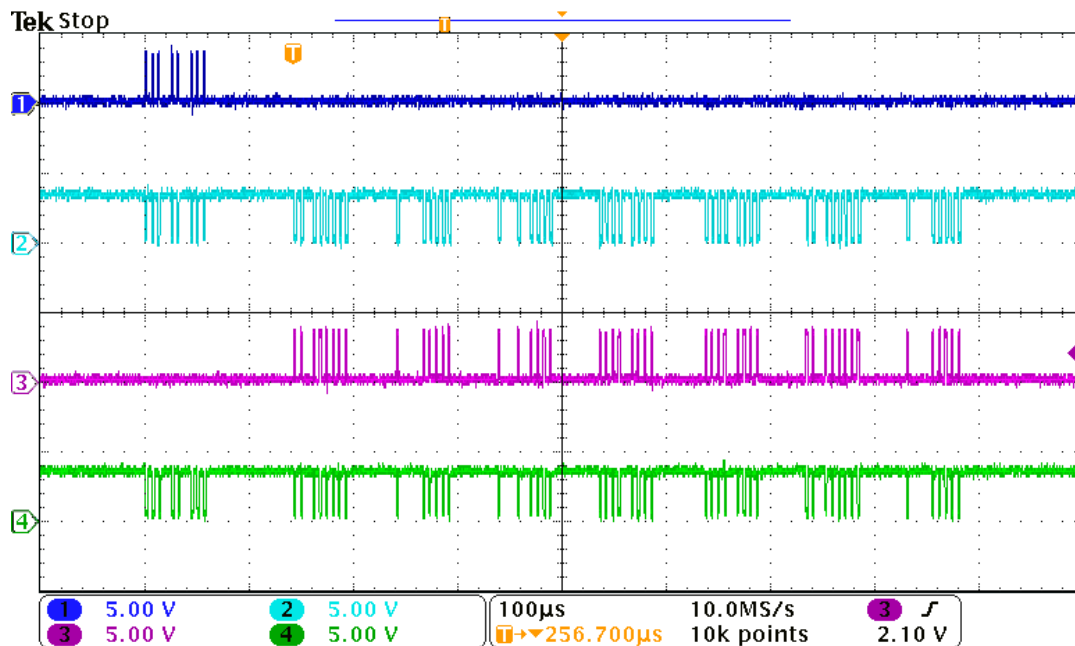


Abbildung 4.6: Signalverlauf einer Übertragung mit 161718 Bit/s. Die Dauer eines Sendesignalpulses beträgt in beiden Bildern $4/16$ der eines Bits. Kanal 1 misst den Sendesignalverlauf und Kanal 2 den Empfangssignalverlauf des Überwachungsmoduls. Kanal 3 misst den Sendesignalverlauf und Kanal 4 den Empfangssignalverlauf eines Sendemoduls.

Bei doppelter Vergabe einer Adresse antworten die Messmodule nahezu gleichzeitig mit unterschiedlichen Daten. In diesem Fall erkennt das Überwachungsmodul eine Antwort, allerdings und die empfangene Information entspricht aber nicht der Eingestellten an den Messmodulen, da sie sich überlagert. In einem solchen Fall wurden die Send- und Empfangssignale der Transceiver-Module eines der Messmodule mit gleicher Adresse und des Überwachungsmoduls aufgenommen und es ergab sich der Verlauf in Abbildung 4.5.

Abbildung 4.7a zeigt nur die erste Übertragung eines Messmoduls, während Abbildung 4.7b den gesamten Datenübertragungsablauf beinhaltet. Unmittelbar vor dem ersten Bit ist das Signal des anderen Messmoduls zu erkennen. Am Empfangssignalverlauf des Transceiver-Moduls vom Überwachungsmodul zu sehen, dass sich die beiden Lichtsignale der Messmodule überlagern und die Übertragung des anderen Messmoduls um etwas weniger als eine Bitdauer früher anfängt. Dies ist jedoch nicht bei jeder Antwort der Messmodule so. Bei wiederholten Messungen zeigte sich, dass sich die beiden Signale der teilweise genau überlagern.

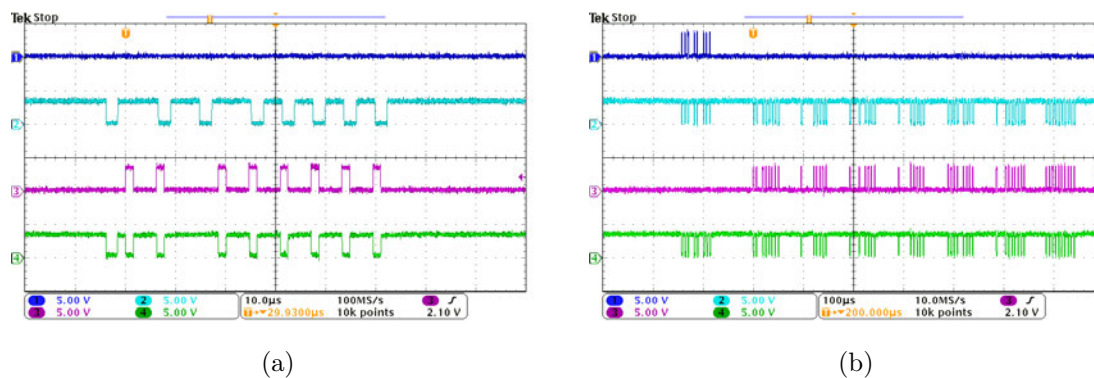


Abbildung 4.7: Signalverlauf einer Übertragung mit 161718 Bit/s bei doppelt belegter Adresse. Die Dauer eines Sendesignalpulses beträgt in beiden Bildern $4/16$ der eines Bits. Kanal 1 misst den Sendesignalverlauf und Kanal 2 den Empfangssignalverlauf des Überwachungsmoduls. Kanal 3 misst den Sendesignalverlauf und Kanal 4 den Empfangssignalverlauf eines Sendemoduls.

4.3.2 Kommunikation zwischen dem Überwachungsmodul und einem Computer

Zum Test der Kommunikation zwischen einem Computer und dem Überwachungsmodul über die externe serielle UART-Schnittstelle wurde die Software *Docklight Version 2.3.25* von *Flachmann & Heggelbacher* [25] genutzt.

Nacheinander wurde die Ausführung und Antwort jedes definierten Befehls geprüft.

Alle Befehle funktionieren den Erwartungen entsprechend und ermöglichen die Datenübertragung vom Überwachungsmodul an den Computer.

4.4 Ausstehende Tests

Die durchgeführten Tests betrafen bisher im Wesentlichen nur die grundsätzliche Funktion der Komponenten und des Demonstrationsaufbaus. Umfangreichere Tests zu der Stromaufnahme, dem betriebsfähigen Spannungsbereich, der Messgenauigkeit der ADC, den verschiedenen Lichtleitkörpermaterialien und -konzepten, dem Einfluss von Verschmutzung der Lichtleitkörperoberfläche, der Übertragungsfehlerrate, der Behandlung von Übertragungsfehlern sowie verschiedenen Oberflächenbeschaffenheiten des Lichtleitkörpers wird Bestandteil zukünftiger Arbeiten sein.

5 Bewertung, Zusammenfassung, Offene Punkte und Ausblick

5.1 Ergebnisse und Bewertung

In dieser Arbeit konnte erfolgreich ein funktionsfähiger modularer Demonstrationsaufbau umgesetzt werden, welcher das Konzept der optischen Kommunikation mittels Infrarot-Transceiver-Module über einen geeigneten Lichtleitkörper zeigt. Hierbei wurden optische Grundlagen vorgestellt, welche für die Berechnung des Lichtleitkörpers erforderlich waren. Der entstandene Lichtleitkörper ermöglichte die Kommunikation nach Master-Slave-Prinzip in einer normalerweise ungünstigen Anordnung der 13 Busteilnehmer. Eine Platine mit den Abmessungen einer Batteriegrundfläche wurde entwickelt, welche sowohl für die Messmodule, als auch das Überwachungsmodul eingesetzt werden konnte und die Möglichkeit der Messwerterfassung oder -simulation bietet.

Beim aktuellen Entwurf des Lichtleitkörpers besteht am Medienübergang vor dem Master-Transceiver-Modul Optimierungsbedarf für eine gezieltere Transmission. Dies bezieht sich auf die bei Simulation des Lichtleitkörpers in Kapitel 3.1.5 festgestellte Ablenkung der Lichtstrahlen von den Transceiver-Modulen der Messmodule. Weiterhin entspricht die raue Oberflächenbeschaffenheit nicht der gewünschten glatten, wofür eine Erprobung noch aussteht. Zudem ist die Positionspräzision des Demonstrationsaufbaus durch die Handfertigung recht niedrig. Bessere Ergebnisse könnte das Bohren aller Schraublöcher durch eine computergesteuerte Fräse in einem homogeneren Grundplattenmaterial oder die Fertigung eines einteiligen Trägers für alle Messmodulplatinen und die Transceiver-Modul-Platinenhälfte des Überwachungsmoduls liefern.

Das Konzept hat sich als viel versprechend erwiesen, sodass eine weitere Bearbeitung dieses Konzepts sinnvoll erscheint.

5.2 Offene Punkte und Ansätze zur Weiterführung

In dieser Arbeit konnten nicht alle Bereiche vollständig ausgearbeitet werden und es blieben einige Fragen offen. So stehen noch die folgende Punkte aus.

5.2.1 Lichtleitkörper

Bei den Berechnung in Kapitel 3.1.2 des Lichtleitkörpers wurde ein Brechungsindex für ein anderes Material und eine andere Wellenlänge verwendet. Hier sollte eine erneute Berechnung mit dem korrekten Brechungsindex des Materials für die genutzte Lichtwellenlänge erfolgen. Der so berechnete Lichtleitkörper sollte anschließend in einem Verfahren gefertigt werden, welches eine glatte Oberfläche hinterlässt, um die Funktion und insbesondere die Verschiebungstoleranz des in dieser Arbeit bearbeiteten Konzepts zu überprüfen. Letztere könnte ebenfalls in einer Simulation überprüft werden. Weiteren Simulationen könnte die mit steigendem Winkel abnehmende Leuchtstärke und die Leistung der Transceiver-Modul-Sendediode berücksichtigen, sodass die Simulationen nicht nur qualitative, sondern auch quantitative Ergebnisse liefern.

Des Weiteren steht die Erprobung weiterer Lichtleitkörperformen, -materialien und -fertigungsverfahren aus.

Eine weitere Idee wäre, den Lichtleitkörper in zwei optisch getrennte, übereinander liegende Schichten zur Ermöglichung eines Vollduplex-Gegenbetriebs. Dadurch würde ebenfalls die erforderliche Breite des in dieser Arbeit bearbeiteten Lichtleitkörperkonzepts reduziert oder eine Vergrößerung der Reflektorflächen erreicht.

Zusätzlich sollten Tests über den Einfluss von externem Licht mit ähnlicher Wellenlänge durchgeführt werden, um das Verhalten bei externen Störeinflüssen zu überprüfen.

Zuletzt könnten alle reflexionsbasierten Lichtleitkörper ebenfalls als verspiegelte Kammer in den Batterierahmen integriert werden. Hierbei müsste besonders geprüft werden, in wie weit Staub und andere Verschmutzungen die Lichtstärke reduzieren können, da dieses Konzept dafür anfälliger sein könnte als eine Lichtleitkörper.

5.2.2 Hardware

Eine grundsätzliche Analyse der verschiedenen Kommunikationskonzepte wurde in der Einleitung vorgestellt, jedoch sind in diesem Bereich weitere Ausführung erforderlich.

Weiterhin muss die Stromaufnahme durch Anpassung der Sendeleistung optimiert werden.

Darüber hinaus sollte die Präzision des Mikrocontrollertakts zur Vermeidung von Übertragungsfehlern verbessert werden. Hierfür ist vermutlich eine neue Taktquelle erforderlich.

5.2.3 Software

In der aktuellen Software wird das Paritätsbit einer empfangenen Nachricht nicht überprüft, was für eine Behandlung von Übertragungsfehlern jedoch hilfreich ist.

Wenn Übertragungsfehler erkannt werden können, könnte ein Programm geschrieben werden, welches die Übertragungsfehlerrate ermittelt.

Weiterhin könnte ein Slave nach empfangen einer Nachricht, die nicht ihn bestimmt war, alle weiteren Nachrichten für eine vorbestimmte Zeit ignorieren. Dadurch könnte die Adressübertragung der Masteradresse in den Slave-Antworten entfallen, was die Datenrate beim aktuellen Datenübertragungsprotokoll verdoppeln würde. Der Master weiß, von wem das Antwortpaket kommt und wie viele zu erwarten sind.

In einem alternativen Übertragungsprotokoll könnte mit anderen optischen Transceivern, welche keinen Echo-Modus [34, S. 2] haben, eine adressbasierte, bitweise Arbitrierung implementiert werden. Sobald mehrere Busteilnehmer gleichzeitig senden, entscheiden die zuerst gesendeten Adressbits, welcher Busteilnehmer senden darf.

5.3 Ausblick

In dieser Arbeit konnte gezeigt werden, dass die Kommunikation mittels optischen Lichtwellenleiters grundsätzlich möglich ist. An dem entstandenen Demonstrationsaufbau können verschiedene Lichtleitkörper verbaut und getestet werden. Zur besseren Beurteilung verschiedener Lichtleitkörperkonzepte sind weitere Arbeiten im Bereich der Software und

Optik erforderlich. Auch müssen noch weitere Lichtleitkörperkonzepte und -materialien untersucht werden. Zudem sollte die Umsetzung des Transceiver-Moduls in diskreten Komponenten zu Reduktion der Kosten geprüft werden.

Abbildungsverzeichnis

1.1	Systemaufbau bei zentraler Messung mittels elektrischer Leitungen im BMS bzw. Überwachungsmodul	2
1.2	Systemaufbau mit Datenübertragung über elektrische Verbindung	3
1.3	Systemaufbau mit Datenübertragung über Hauptstromleitung	4
1.4	Systemaufbau mit Datenübertragung über elektromagnetische Verbindung	5
1.5	Systemaufbau mit Datenübertragung über optischen Lichtleitkörper	6
2.1	Vereinfachter Systemaufbau mit Datenübertragung über optischen Lichtleitkörper	11
2.2	Blockschaltbild des Demonstrationsaufbaus	13
2.3	Paketaufbau und Signalverlauf einer UART-Standardübertragung	18
2.4	Paketaufbau und Verlauf eines Sende- oder invertierten Empfangssignals einer UART-Übertragung in Anlehnung an die IrDA SIR Spezifikation	20
2.5	Theoretischer Ablauf der Übertragung eines Bits von einem zum anderen Busteilnehmer	21
2.6	Snelliusschen Brechungsgesetz	25
2.7	Beispiel optischer Effekte	27
2.8	Verfügbare Grundfläche für den Lichtleitkörper	29
2.9	Lichtleitkörperkonzept der Streukammer	30
2.10	Lichtleitkörperkonzept mit Reflexionsbohrungen	31
2.11	Lichtleitkörperkonzept mit streuenden Reflektorflächen	32
2.12	Lichtleitkörperkonzept mit ausgerichteten Reflektorflächen	33
2.13	Lichtleitkörperkonzept mit Lichtleitbahnen	34
3.1	Geometrische Zeichnung für die Berechnung der Reflektoranfangs- und -endpunkte	38
3.2	Simulationsergebnis der Linsen	41
3.3	Geometrie zur Berechnung nach Hans-Peter Willig.	42
3.4	Simulationsergebnis der Linsen nach der alternativen Berechnung	43

3.5	Geometrische Zeichnung für die Berechnung der Reflektoranfangs- und -endpunkte	46
3.6	Simulationsergebnis des berechneten Lichtleitkörpers	55
3.7	Umriss des Lichtleitkörpers aus der DXF-Datei	56
3.8	Vereinfachte Darstellung der Komponenten für das Messmodul mit aufgesetzter Adapterplatine und vereinfachtem Anschlussplan	65
3.9	Fotos der aufgebauten Adapterplatine	66
3.10	Fotos des Messmodulaufbaus mit Adapterplatine	66
3.11	3D-Modelle der zusammengesetzten Komponenten im Aufbau mit Adapterplatine	66
3.12	3D-Modelle der Messmodule mit Adapterplatine zusammengesetzt in Batterieanordnung	67
3.13	Schaltplan der Messmodulplatine	70
3.14	Entwurf der Messmodulplatine	71
3.15	Vereinfachter Anschlussplan der Modulüberwachung	75
3.16	3D-Modell des Überwachungsmodulgehäuses	76
3.17	3D-Modelle der Messmodule	77
3.18	3D-Modell des Demonstrationsaufbaus	78
3.19	Foto des fertiggestellten Versuchsaufbaus	79
3.20	Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Messmodul als Slave	85
3.21	Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Überwachungsmodul als Master Teil 1	86
3.22	Vereinfachtes Flussdiagramm des Hauptprogrammablaufs im Überwachungsmodul als Master Teil 2	87
3.23	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl Abbruch	88
3.24	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Kommunikationsfehlern oder absichtlich anderer Adresse	89
3.25	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl Ping	90
3.26	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl Sende Adressen aller verfügbaren Slaves	91
3.27	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl Sende Daten aller verfügbaren Slaves	92

3.28	Kommunikationsdiagramm zwischen Computer und Überwachungsmodul (Master) bei Befehl Sende Daten aller Slaves	93
3.29	Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten	94
3.30	Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Fehler bei Adressübertragung	95
3.31	Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Übertragungsfehler beim Befehl (bekannt)	96
3.32	Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Übertragungsfehler beim Befehl (unbekannt)	97
3.33	Kommunikationsdiagramm zwischen Überwachungsmodul (Master) und Messmodul (Slave) bei Befehl Sende Daten mit Verbindungsunterbrechung	98
4.1	Vergleich des Signalverlaufs einer Übertragung bei 250000 Bit/s und 216001 Bit/s	106
4.2	Vergleich des Signalverlaufs einer Übertragung bei 220001 Bit/s mit unterschiedlichen Sendesignalpulsängen	108
4.3	Kennlinie der Sendediode eines Transceiver-Moduls	110
4.4	Simulationsergebnis des berechneten Lichtleitkörpers	112
4.5	Signalverlauf einer Übertragung mit den festgelegten Einstellungen bei 161718 Bit/s	114
4.6	Signalverlauf einer Übertragung mit 161718 Bit/s	115
4.7	Signalverlauf einer Übertragung mit 161718 Bit/s bei doppelt belegter Adresse	116

Tabellenverzeichnis

1.1	Vergleich der Kommunikationskonzepte	8
2.1	Liste der bereitgestellten Komponenten für den Demonstrationsaufbau . .	15
2.2	Liste der während der Entwicklung ergänzten Komponenten	16
2.3	Adresszuweisung der Busteilnehmer	18
2.4	Vor- und Nachteile der verschiedenen Lichtleitkörper	35
3.1	Abkürzungen und deren Bedeutung für die Berechnung der Reflektorflächen	44
3.2	Funktion der Transceiver-Modul-Linsen bei Positionierung des Transceiver- Moduls auf der linken oder rechten Seite des Lichtleitkörpers	45
3.3	Berechnung der y-Koordinaten der Reflektoranfangs- und -endpunkte . . .	47
3.4	x-Koordinate der obersten Reflektorendpunkte auf beiden Seiten	47
3.5	Formeln zur Berechnung der unterschiedlichen Längen $L7$ für einen Re- flektorflächenendpunkt	48
3.6	Formeln zur Berechnung der unterschiedlichen Längen $L6$ für einen Re- flektorflächenendpunkt	49
3.7	Formeln zur Berechnung der x-Koordinate eines Reflektorflächenendpunktes	50
3.8	Formeln zur Berechnung der Länge $L2$ eines Reflektorflächenanfangspunktes	51
3.9	Formeln zur Berechnung der Länge $L3$ eines Reflektorflächenanfangspunktes	51
3.10	Formeln zur Berechnung der x-Koordinate eines Reflektorflächenanfangs- punktes	52
3.11	Messwerte zur Berechnung der LED Ströme und die Ergebnisse	58
3.12	Berechnungswerte und -ergebnisse der maximalen Umgebungstemperatur bei maximalem Strom der Linearspannungsregler	60
3.13	Berechnungswerte und -ergebnisse des maximalen, für Bedienelemente zur Verfügung stehenden Stroms der Mikrocontrollerplatinen	62
3.14	Teileliste der Busteilnehmer	71
3.15	Implementierte Befehle und Antworten im Überwachungsmodul und deren Funktion	82

3.16	Implementierte Befehle und Antworten im Messmodul und deren Funktion	83
3.17	Benutzeroberflächenfunktionen	101
3.18	Bedeutung der Abkürzungen für den Verbindungszustand	102
4.1	Liste der getesteten Datenraten mit den zugehörigen Sendesignalpulsdauern als Anteil der Bitdauer und das Testergebnis.	107
4.2	Messwerte der Sendediodenkennlinie eines Transceiver-Moduls	109
A.1	Konfiguration der DAVE-APP SYSTIMER mit dem Namen SYSTIMER	138
A.2	Konfiguration der DAVE-APP CPU_CTRL_XMC4 mit dem Namen CPU_CTRL_XMC4_0139	
A.3	Konfiguration der DAVE-APP I2C_MASTER mit dem Namen I2C_LCD	140
A.4	Konfiguration der DAVE-APP GLOBAL_CCU4 mit dem Namen GLOBAL_CCU4_0	141
A.5	Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 1	141
A.6	Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 2	142
A.7	Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 3	143
A.8	Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 4	144
A.9	Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 5	145
A.10	Konfiguration der DAVE-APP UART mit dem Namen UART_External Teil 1	146
A.11	Konfiguration der DAVE-APP UART mit dem Namen UART_External Teil 2	147
A.12	Konfiguration der DAVE-APP UART mit dem Namen UART_Internal Teil 1	148
A.13	Konfiguration der DAVE-APP UART mit dem Namen UART_Internal Teil 2	149
A.14	Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DO_TransceiverShutDown, DO_LED1, DO_LED2 und DO_LED_Y	150
A.15	Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DI_BTN, DI_RSW_1, DI_RSW_2, DI_RSW_4 und DI_RSW_8	150
A.16	Konfiguration der DAVE-APP SYSTIMER mit dem Namen SYSTIMER	151

A.17 Konfiguration der DAVE-APP CPU_CTRL_XMC1 mit dem Namen CPU_CTRL_XMC1_0151	
A.18 Konfiguration der DAVE-APP CLOCK_XMC1 mit dem Namen CLOCK_XMC1_0152	
A.19 Konfiguration der DAVE-APP UART mit dem Namen UART_Internal	
Teil 1	153
A.20 Konfiguration der DAVE-APP UART mit dem Namen UART_Internal	
Teil 2	154
A.21 Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DO_TRX_SD, DO_LED1, DO_LED2 und DO_LED_Y	155
A.22 Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DI_RSW_1, DI_RSW_2, DI_RSW_4 und DI_RSW_8	155
A.23 Konfiguration der DAVE-APPs PIN_INTERRUPT mit dem Namen DI_INT_BTN	155
A.24 Konfiguration der DAVE-APPs GLOBAL_ADC mit den Namen GLO- BAL_ADC_0	156
A.25 Konfiguration der DAVE-APPs ADC_MEASUREMENT mit den Namen ADC_MEASUREMENT_0	156

Abkürzungen

ADC Analog-to-Digital Converter.

ASCII American Standard Code for Information-Interchange.

BMS Batteriemangagementsystem.

DPS Data/Parity/Stop.

DXF Drawing Interchange File Format.

GPIO General-Purpose Input / Output.

I2C Inter-Integrated Circuit.

IrDA Infrared Data Association.

LCD Liquid Crystal Display.

LED Light-Emitting Diode.

PLC Powerline Communication.

SIR Serial Infrared.

UART Universal-Asynchronous-Receive-Transmit.

USB Universal Serial Bus.

USIC Universal-Serial-Interface-Channel.

Literaturverzeichnis

- [1] DESIGNSPARK: *Internetseite der Software DesignSpark Mechanical*. 2020. – URL <https://www.rs-online.com/designspark/mechanical-software-de>. – Zugriffsdatum: 2020-09-04
- [2] EIKE MENSE: *Hard- und Softwareentwicklung für einen drahtlosen Batterie-Zellen-Sensor zur elektrochemischen Impedanzspektroskopie*. Hochschule für Angewandte Wissenschaften Hamburg, 2014. – URL http://edoc.sub.uni-hamburg.de/haw/frontdoor.php?source_opus=2881&la=de
- [3] HANS-PETER WILLIG: *Internetseite über die Berechnung von asphärischen Linsen*. 2020. – URL https://physik.cosmos-indirekt.de/Physik-Schule/Asph%C3%A4rische_Linse. – Zugriffsdatum: 2020-09-29
- [4] INFINEON TECHNOLOGIES AG: *Internetseite der Evaluationsplatine XMC4700 / XMC4800 Relax Kit Series-V1*. 2020. – URL https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc47_relax_v1/. – Zugriffsdatum: 2020-08-28
- [5] INFINEON TECHNOLOGIES AG: *Internetseite der Evaluationsplatine XMC4700 / XMC4800 Relax Kit Series-V1*. 2020. – URL https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc47_relax_v1/. – Zugriffsdatum: 2020-08-28
- [6] JOACHIM HEINTZE ; PETER BOCK (Hrsg.): *Lehrbuch zur Experimentalphysik Band 4: Wellen und Optik*. URL <https://link.springer.com/book/10.1007/978-3-662-54492-1>, 2017. – ISBN 978-3-662-54492-1
- [7] KICAD DEVELOPERS TEAM: *Internetseite der Software KiCad*. 2020. – URL <https://kicad-pcb.org/>. – Zugriffsdatum: 2020-09-21

- [8] KRAFTFAHRT-BUNDESAMT: *Internetseite der Jahresbilanz - Bestand*. 2020. – URL https://www.kba.de/DE/Statistik/Fahrzeuge/Bestand/Jahresbilanz/b_jahresbilanz_inhalt.html. – Zugriffsdatum: 2020-08-23
- [9] MATHE ONLINE: *Internetseite für Snelliussches Brechungsgesetz*. 2020. – URL <https://www.av.ph.tum.de/Experiment/3000/Beschreibungen/ver3021.php>. – Zugriffsdatum: 2020-09-14
- [10] MATHE ONLINE: *Internetseite für Winkelfunktionen*. 2020. – URL <https://www.mathe-online.at/mathint/wfun/i.html>. – Zugriffsdatum: 2020-09-14
- [11] N. N.: *Infrared Data Communication According to IrDA Standard Part 1: Physical Layer Rev. 1.4, 20-Sep-06, Document Number: 82513*. Vishay Intertechnology, Inc., 2006. – URL <http://www.vishay.com/ir-transceivers/list/product-81288/tab/documents/>
- [12] N. N.: *Home Plug Green PHY - The Standard For In-Home Smart Grid - Powerline Communications*. HomePlug Powerline Alliance, Inc., 2010. – URL [nichtverfügbar](#)
- [13] N. N.: *Voltage Regulator IFX1117 Data Sheet Rev. 1.0, 2011-02-24*. Infineon Technologies AG, 2011. – URL <https://www.infineon.com/cms/de/product/power/linear-voltage-regulator/linear-voltage-regulators-for-industrial-applications/ifx1117me-v33/>
- [14] N. N.: *XMC 2Go Kit with XMC1100 Kit Version 1.0 Board User's Manual Revision 1.0, 2014-02-20*. Infineon Technologies AG, 2014. – URL https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc_2go_xmc1100_v1/
- [15] N. N.: *IFX54211MB V33 Data Sheet Rev. 1.0, 2015-08-26*. Infineon Technologies AG, 2015. – URL <https://www.infineon.com/cms/de/product/power/linear-voltage-regulator/linear-voltage-regulators-for-industrial-applications/ifx54211mb-v33/>
- [16] N. N.: *XMC4700 Relax-V1 Schematic 13.11.2015*. Infineon Technologies AG, 2015. – URL https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc47_relax_v1/

- [17] N. N.: *Evaluation Board For XMC4000 Family Board User's Manual Revision 1.2, 2016-06-16*. Infineon Technologies AG, 2016. – URL https://www.infineon.com/cms/en/product/evaluation-boards/kit_xmc47_relax_v1/
- [18] N. N.: *Produktdatenblatt, August 2016 Makrolon® GP Massivplatten aus Polycarbonat*. Covestro Deutschland AG, 2016. – URL https://www.plexiglas-hecker.de/wp-content/uploads/2016/10/Datenblatt_Makrolon.pdf
- [19] N. N.: *XMC1100 AB-Step Data Sheet V1.8 2016-09*. Infineon Technologies AG, 2016. – URL <https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc1000-industrial-microcontroller-arm-cortex-m0/>
- [20] N. N.: *XMC1100 AB-Step Reference Manual V1.3 2016-08*. Infineon Technologies AG, 2016. – URL <https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc1000-industrial-microcontroller-arm-cortex-m0/>
- [21] N. N.: *XMC4700 / XMC4800 Reference Manual V1.3 2016-07*. Infineon Technologies AG, 2016. – URL <https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc4000-industrial-microcontroller-arm-cortex-m4/>
- [22] N. N.: *XMC4700 / XMC4800 Data Sheet V1.1 2018-09*. Infineon Technologies AG, 2018. – URL <https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc4000-industrial-microcontroller-arm-cortex-m4/>
- [23] N. N.: *D6 SPST Momentary Key Switches*. C and K Switches, 2020. – URL <https://www.mouser.de/datasheet/2/60/d6-1382571.pdf>
- [24] N. N.: *High Intensity LED in Ø 3 mm Tinted Diffused Package*. Vishay Intertechnology, Inc., 2020. – URL <https://www.mouser.de/datasheet/2/427/t11e4401-1767394.pdf>

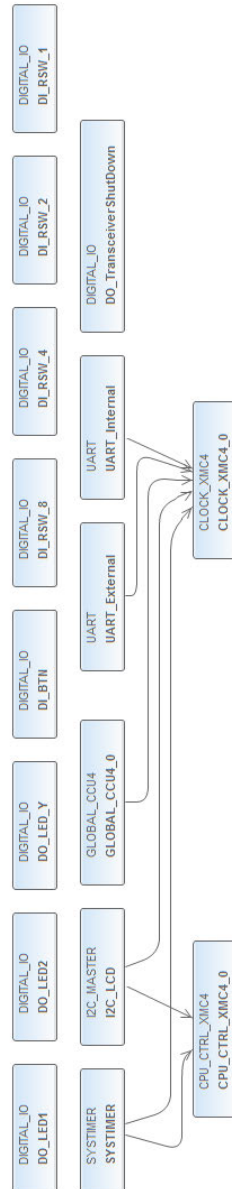
- [25] N. N.: *Internetseite der Software Docklight*. 2020. – URL <https://docklight.de/>. – Zugriffsdatum: 2020-10-10
- [26] N. N.: *Internetseite der Software Excel*. 2020. – URL <https://www.microsoft.com/de-de/microsoft-365/excel>. – Zugriffsdatum: 2020-10-05
- [27] N. N.: *Internetseite der Software LibreCAD*. 2020. – URL <https://librecad.org/>. – Zugriffsdatum: 2020-10-01
- [28] N. N.: *Internetseite der Software LTSpice*. 2020. – URL <https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html>. – Zugriffsdatum: 2020-10-10
- [29] N. N.: *Internetseite des LCD-Moduls 2004A*. 2020. – URL <https://www.amazon.de/LCD-Display-Zeichen-Zeilen-I2C-Schnittstellenmodul-HD44780-Chipsatz/dp/B077K8ZJ6M>. – Zugriffsdatum: 2020-08-28
- [30] N. N.: *Internetseite des USB-Hubs*. 2020. – URL <https://www.ebay.de/itm/293497644926>. – Zugriffsdatum: 2020-08-28
- [31] N. N.: *Internetseite zu Plexiglas/Acrylglas (PMMA)*. 2020. – URL https://hm-kunststofftechnik.de/plexiglas-acrylglas_im_detail.html. – Zugriffsdatum: 2020-10-07
- [32] N. N.: *Internetseite zum Brechungsindex*. 2020. – URL <https://de.wikipedia.org/wiki/Brechungsindex>. – Zugriffsdatum: 2020-10-01
- [33] N. N.: *Internetseite zur Reflexion*. 2020. – URL [https://de.wikipedia.org/wiki/Reflexion_\(Physik\)](https://de.wikipedia.org/wiki/Reflexion_(Physik)). – Zugriffsdatum: 2020-10-10
- [34] N. N.: *TFDU4101 Datasheet Rev. 2.0, 03-Mar-2020, Document Number: 81288*. Vishay Intertechnology, Inc., 2020. – URL <http://www.vishay.com/ir-transceivers/list/product-81288/tab/documents/>
- [35] N. N.: *NX3225SA Datasheet*. Nihon Dempa Kogyo Co., Ltd., Veröffentlichungsjahr unbekannt. – URL https://www.ndk.com/en/products/search/crystal/1190855_1494.html
- [36] NICO SASSANO: *Hard- und Softwareentwicklung für einen drahtlos kommunizierenden Batterie-Zellensensor mit funksynchronisierter Messung*. Hochschule für

- Angewandte Wissenschaften Hamburg, 2013. – URL http://edoc.sub.uni-hamburg.de/haw/frontdoor.php?source_opus=2262&la=de
- [37] NICO SASSANO: *Entwicklung eines Messsystems zur funksynchronisierten elektrochemischen Impedanzspektroskopie an Batterie-Zellen*. Hochschule für Angewandte Wissenschaften Hamburg, 2015. – URL http://edoc.sub.uni-hamburg.de/haw/frontdoor.php?source_opus=3561&la=de
- [38] PROTO ADVANTAGE: *Internetseite der Adapterplatine Proto Advantage FPC100P010*. 2020. – URL https://www.proto-advantage.com/store/product_info.php?products_id=3400021. – Zugriffsdatum: 2020-09-04
- [39] RICKTU288: *Internetseite der Software Ray Optics Simulation*. 2020. – URL <https://ricktu288.github.io/ray-optics/>. – Zugriffsdatum: 2020-09-29
- [40] THOMAS LANDINGER, STEFAN MIKL, FATIMA RASRAS, GÜNTER HOFER, GÜNTER SCHWARZBERGER, MATTHIAS ROSE, ANDREAS JOSSEN: *Interference Scenarios during Single Cell Impedance Measurements in Automotive Battery Packs*. Thomas Landler, 2020
- [41] VISHAY INTERTECHNOLOGY, INC.: *Internetseite des infrarot-Transceiver-Moduls TFDU4101*. 2020. – URL <https://www.vishay.com/product?docid=81288>. – Zugriffsdatum: 2020-08-28

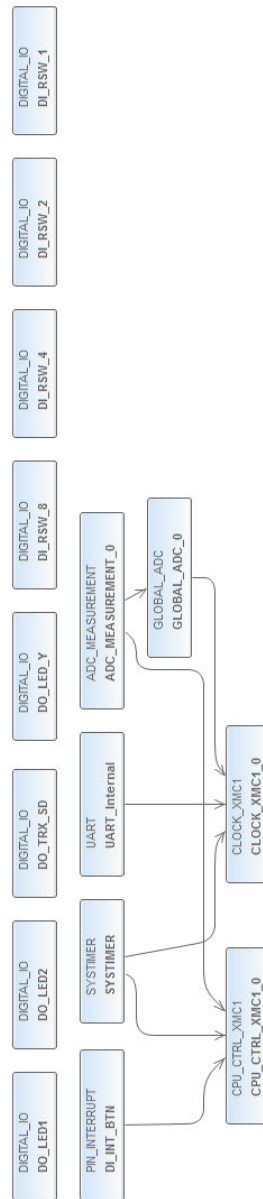
A Anhänge

A.1 Konfiguration

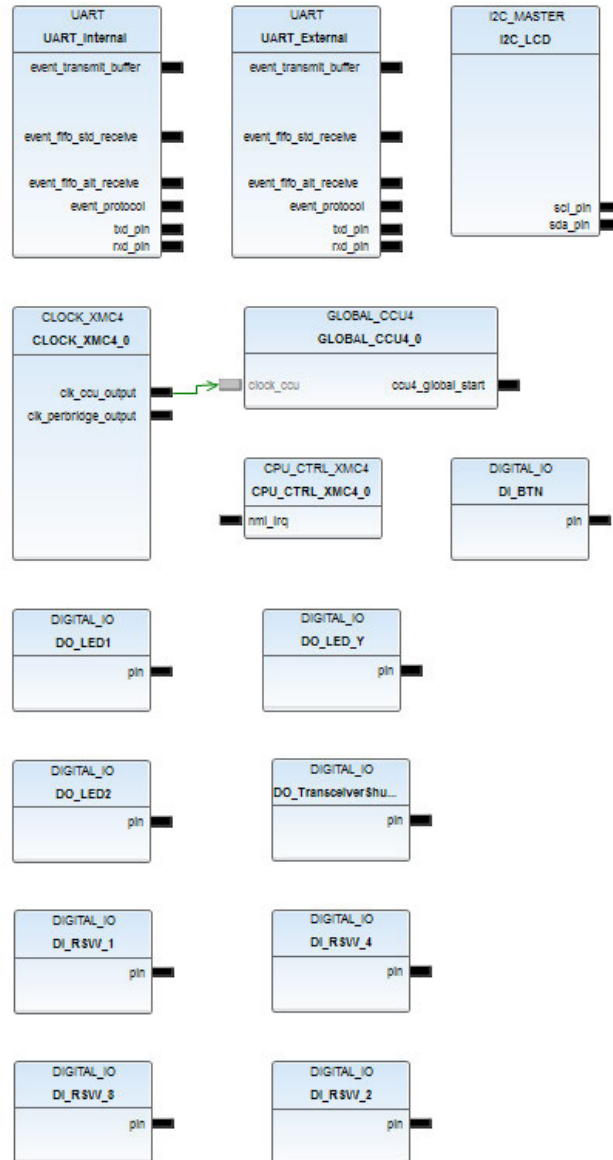
A.1.1 APP Abhängigkeit der DAVE-APPs des Überwachungsmoduls als Master



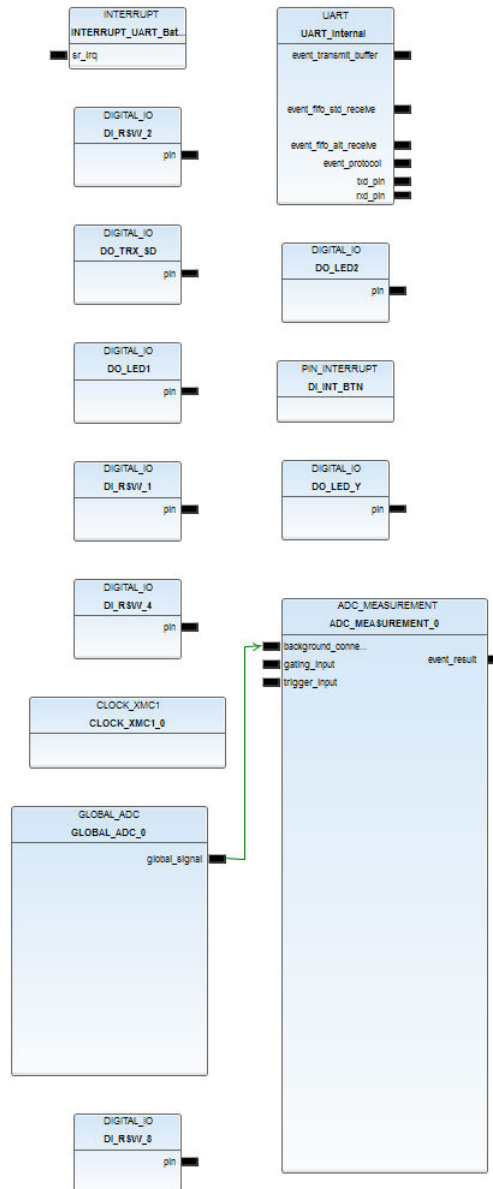
A.1.2 APP Abhängigkeit der DAVE-APPs des Messmoduls als Slave



A.1.3 Hardwaresignalverbindungen der DAVE-APPs des Überwachungsmoduls als Master



A.1.4 Hardwaresignalverbindungen der DAVE-APPs des Messmoduls als Slave



A.1.5 Konfiguration der DAVE-APPs des Überwachungsmoduls als Master

Tabelle A.1: Konfiguration der DAVE-APP SYSTIMER mit dem Namen SYSTIMER

Registerkarte	Einstellung	Wert
General Settings	SysTick timer period [us]	1
General Settings	Number of software timers	2
Interrupt Settings	SysTick Interrupt Settings Preemption	63
Interrupt Settings	SysTick Interrupt Settings Subpriority	0

Tabelle A.2: Konfiguration der DAVE-APP CPU_CTRL_XMC4 mit dem Namen CPU_CTRL_XMC4_0

Registerkarte	Einstellung	Wert
General Settings	Number of priority bits for priority grouping	6
General Settings	Debug interface	Disabled
General Settings	Disable write buffer	No
Exception Settings	Enable hard fault debugging support	No
Exception Settings	Enable trap on divide by zero	No
Exception Settings	Enable trap on unaligned access	No
Exception Settings	Enable memory management fault	No
Exception Settings	Memory Management Fault Priority Preemption priority	0
Exception Settings	Memory Management Fault Priority Subpriority	0
Exception Settings	Enable usage fault	No
Exception Settings	Usage Fault Priority Preemption priority	0
Exception Settings	Usage Fault Priority Subpriority	0
Exception Settings	Enable bus fault	No
Exception Settings	Bus Fault Priority Preemption priority	0
Exception Settings	Bus Fault Priority Subpriority	0

Tabelle A.3: Konfiguration der DAVE-APP I2C_MASTER mit dem Namen I2C_LCD

Registerkarte	Einstellung	Wert
General Settings	Desired bus speed [KHz]	100
General Settings	Actual bus speed [KHz]	100
General Settings	Multi-master Settings Enable multi-master	No
General Settings	Multi-master Settings Own address	0
Advanced Settings	Transmit/Receive Handling Transmit mode	Interrupt
Advanced Settings	Transmit/Receive Handling Receive mode	Interrupt
Advanced Settings	FIFO Settings Enable Tx FIFO	Yes
Advanced Settings	FIFO Settings Size	16
Advanced Settings	FIFO Settings Enable Rx FIFO	Yes
Advanced Settings	FIFO Settings Size	16
Interrupt Settings	Transmit Interrupt Priority Preemption priority	63
Interrupt Settings	Transmit Interrupt Priority Subpriority	0
Interrupt Settings	Transmit End of transmit callback	No
Interrupt Settings	Receive Interrupt Priority Preemption priority	63
Interrupt Settings	Receive Interrupt Priority Subpriority	0
Interrupt Settings	Receive End of transmit callback	No
Interrupt Settings	Error Handling Callback Functions Nack received	Yes: I2C_NACK_CALLBACK
Interrupt Settings	Error Handling Callback Functions Arbitration lost	No
Interrupt Settings	Error Handling Callback Functions Error detected	No
Pin Settings	Enable noise filter	No
Pin Settings	Enable advanced pin configurations	No

Tabelle A.4: Konfiguration der DAVE-APP GLOBAL_CCU4 mit dem Namen GLOBAL_CCU4_0

Registerkarte	Einstellung	Wert
General Settings	Clock frequency [MHz]	144
General Settings	Multi channel mode shadow transfer	Period and Compare

Tabelle A.5: Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 1

Registerkarte	Einstellung	Wert
General Settings	fOHP [MHz]	12
General Settings	fOSC [MHz]	12
General Settings	fUSBPLL [MHz]	192
General Settings	fPLL [MHz]	288
General Settings	fOFI [MHz]	24
General Settings	fSTDBY [kHz]	32.768
General Settings	fULP [kHz]	32.768
General Settings	fOSI [kHz]	32.768
General Settings	fCPU [MHz]	144
General Settings	fDMA [MHz]	144
General Settings	fPERIPH [MHz]	144
General Settings	fCCU [MHz]	144
General Settings	fUSB [MHz]	48
General Settings	fSDMMC [MHz]	48
General Settings	fETH [MHz]	72
General Settings	fEBU [MHz]	288
General Settings	fWDT [MHz]	24
General Settings	fEXT [MHz]	144
General Settings	fRTC [MHz]	32.768

Tabelle A.6: Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 2

Registerkarte	Einstellung	Wert
Clock Generation Settings	High Precision Oscillator Settings Operation mode	External Crystal Mode
Clock Generation Settings	High Precision Oscillator Settings External clock frequency [MHz]	12
Clock Generation Settings	Internal Fast Oscillator Settings Calibration	Factory Calibration
Clock Generation Settings	Main PLL Settings Enable main PLL	Yes
Clock Generation Settings	Main PLL Settings PLL clock source	External Crystal High Precision Oscillator
Clock Generation Settings	Main PLL Settings PLL operating mode	Normal Mode
Clock Generation Settings	Main PLL Settings Requested PLL frequency [MHz]	288
Clock Generation Settings	Main PLL Settings Actual PLL frequency [MHz]	288
Clock Generation Settings	USB PLL Settings Requested USB PLL frequency [MHz]	192
Clock Generation Settings	USB PLL Settings Actual USB PLL frequency [MHz]	192
Standby Clock Generation Settings	Standby Clock (fSTDBY) Clock source	Internal Slow Oscillator
Standby Clock Generation Settings	RTC Clock (fRTC) Clock source	Internal Slow Oscillator

Tabelle A.7: Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 3

Registerkarte	Einstellung	Wert
Clock Generation Settings	High Precision Oscillator Settings Operation mode	External Crystal Mode
Clock Generation Settings	High Precision Oscillator Settings External clock frequency [MHz]	12
Clock Generation Settings	Internal Fast Oscillator Settings Calibration	Factory Calibration
Clock Generation Settings	Main PLL Settings Enable main PLL	Yes
Clock Generation Settings	Main PLL Settings PLL clock source	External Crystal High Precision Oscillator
Clock Generation Settings	Main PLL Settings PLL operating mode	Normal Mode
Clock Generation Settings	Main PLL Settings Requested PLL frequency [MHz]	288
Clock Generation Settings	Main PLL Settings Actual PLL frequency [MHz]	288
Clock Generation Settings	USB PLL Settings Requested USB PLL frequency [MHz]	192
Clock Generation Settings	USB PLL Settings Actual USB PLL frequency [MHz]	192

Tabelle A.8: Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 4

Registerkarte	Einstellung	Wert
Standby Clock Generation Settings	Standby Clock (fSTDBY) Clock source	Internal Slow Oscillator
Standby Clock Generation Settings	RTC Clock (fRTC) Clock source	Internal Slow Oscillator
Clock Selection Settings	System Clock (fSYS) Clock Source	Main PLL Clock
Clock Selection Settings	System Clock (fSYS) Clock Divider	2
Clock Selection Settings	System Clock (fSYS) Actual frequency [MHz]	144
Clock Selection Settings	CPU Clock (fCPU) Clock Divider	1
Clock Selection Settings	CPU Clock (fCPU) Actual frequency [MHz]	144
Clock Selection Settings	Peripheral Bus Clock (fPERIPH) Clock Divider	1
Clock Selection Settings	Peripheral Bus Clock (fPERIPH) Actual frequency [MHz]	144
Clock Selection Settings	CCU Clock (fCCU) Enable CCU clock	Yes
Clock Selection Settings	CCU Clock (fCCU) Clock Divider	1
Clock Selection Settings	CCU Clock (fCCU) Actual frequency [MHz]	144
Clock Selection Settings	USB Clock (fUSB) and SDMMC Clock (fSDMMC) Enable USB / SDMMC clock	Yes
Clock Selection Settings	USB Clock (fUSB) and SDMMC Clock (fSDMMC) Clock source	USB PLL Clock
Clock Selection Settings	USB Clock (fUSB) and SDMMC Clock (fSDMMC) Clock Divider	4
Clock Selection Settings	USB Clock (fUSB) and SDMMC Clock (fSDMMC) Actual frequency [MHz]	48

Tabelle A.9: Konfiguration der DAVE-APP CLOCK_XMC4 mit dem Namen CLOCK_XMC4_0 Teil 5

Registerkarte	Einstellung	Wert
Clock Selection Settings	EBU Clock (fEBU) Enable EBU clock	Yes
Clock Selection Settings	EBU Clock (fEBU) Clock Divider	1
Clock Selection Settings	EBU Clock (fEBU) Actual frequency [MHz]	288
Clock Selection Settings	External Clock (fEXT) Enable external clock out- put	No
Clock Selection Settings	External Clock (fEXT) Clock source	System Clock
Clock Selection Settings	External Clock (fEXT) Clock Divider	1
Clock Selection Settings	External Clock (fEXT) Actual frequency [MHz]	144
Clock Selection Settings	Ethernet Clock (fETH) Enable Ethernet clock	Yes
Clock Selection Settings	Ethernet Clock (fETH) Clock source	System Clock
Clock Selection Settings	Ethernet Clock (fETH) Clock Divider	2
Clock Selection Settings	Ethernet Clock (fETH) Actual frequency [MHz]	72
Event Settings	OSC_HP oscillator wat- chdog trap	No
Event Settings	USB VCO lock trap	No
Event Settings	System VCO lock trap	No
Event Settings	OSC_ULP oscillator wat- chdog trap	No

Tabelle A.10: Konfiguration der DAVE-APP UART mit dem Namen UART_External
Teil 1

Registerkarte	Einstellung	Wert
General Settings	Operation mode	Full Duplex
General Settings	Desired speed [baud]	115200
General Settings	Actual speed [baud]	115204
General Settings	Data bits	8
General Settings	Stop bit	1 Stop Bit
General Settings	Parity selection	Even Parity
Advanced Settings	Protocol Handling Transmit mode	Direct
Advanced Settings	Protocol Handling Receive mode	Direct
Advanced Settings	Protocol Handling Timing Settings Oversampling	16
Advanced Settings	FIFO Settings Enable transmit FIFO	No
Advanced Settings	Size	16
Advanced Settings	FIFO Settings Enable receive FIFO	Yes
Advanced Settings	Size	16
Advanced Settings	Shift Settings MSB first	No
Advanced Settings	Shift Settings Enable data input inversion	No
Advanced Settings	Shift Settings Enable data output inversion	No

Tabelle A.11: Konfiguration der DAVE-APP UART mit dem Namen UART_External
Teil 2

Registerkarte	Einstellung	Wert
Interrupt Settings	Transmit Interrupt Priority Preemption priority	63
Interrupt Settings	Transmit End of transmit callback	No
Interrupt Settings	Receive Interrupt Priority Preemption priority	63
Interrupt Settings	Receive End of transmit callback	No
Interrupt Settings	Error Handling Interrupt Priority Preemption priority	63
Interrupt Settings	Error Handling Interrupt Priority Subpriority	0
Interrupt Settings	Error Handling Callback Functions Receiver noise detection	No
Interrupt Settings	Error Handling Callback Functions Format error in stop bit 0	No
Interrupt Settings	Error Handling Callback Functions Format error in stop bit 1	No
Interrupt Settings	Error Handling Callback Functions Synchronization break detected	No
Pin Settings	Enable advanced pin characteristics	Yes
Pin Settings	Transmit Mode	Push Pull
Pin Settings	Transmit Driver strength	Don't Care
Pin Settings	Receive Mode	Pull Up

Tabelle A.12: Konfiguration der DAVE-APP UART mit dem Namen UART_Internal
Teil 1

Registerkarte	Einstellung	Wert
General Settings	Operation mode	Full Duplex
General Settings	Desired speed [baud]	161718
General Settings	Actual speed [baud]	161718
General Settings	Data bits	8
General Settings	Stop bit	1 Stop Bit
General Settings	Parity selection	Even Parity
Advanced Settings	Protocol Handling Transmit mode	Direct
Advanced Settings	Protocol Handling Receive mode	Direct
Advanced Settings	Protocol Handling Timing Settings Oversampling	16
Advanced Settings	FIFO Settings Enable transmit FIFO	No
Advanced Settings	Size	16
Advanced Settings	FIFO Settings Enable receive FIFO	Yes
Advanced Settings	Size	16
Advanced Settings	Shift Settings MSB first	No
Advanced Settings	Shift Settings Enable data input inversion	No
Advanced Settings	Shift Settings Enable data output inversion	No

Tabelle A.13: Konfiguration der DAVE-APP UART mit dem Namen UART_Internal
Teil 2

Registerkarte	Einstellung	Wert
Interrupt Settings	Transmit Interrupt Priority Preemption priority	63
Interrupt Settings	Transmit End of transmit callback	No
Interrupt Settings	Receive Interrupt Priority Preemption priority	63
Interrupt Settings	Receive End of transmit callback	No
Interrupt Settings	Error Handling Interrupt Priority Preemption priority	63
Interrupt Settings	Error Handling Interrupt Priority Subpriority	0
Interrupt Settings	Error Handling Callback Functions Receiver noise detection	No
Interrupt Settings	Error Handling Callback Functions Format error in stop bit 0	No
Interrupt Settings	Error Handling Callback Functions Format error in stop bit 1	No
Interrupt Settings	Error Handling Callback Functions Synchronization break detected	No
Pin Settings	Enable advanced pin characteristics	Yes
Pin Settings	Transmit Mode	Push Pull
Pin Settings	Transmit Driver strength	Don't Care
Pin Settings	Receive Mode	Pull Up

Tabelle A.14: Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DO_TransceiverShutDown, DO_LED1, DO_LED2 und DO_LED_Y

Registerkarte	Einstellung	Wert
General Settings	Pin direction	Input/Output
General Settings	Input Settings Mode	Tristate
General Settings	Output Settings Mode	Push Pull
General Settings	Output Settings Initial output level	Low
General Settings	Output Settings Driver strength	Don't Care

Tabelle A.15: Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DI_BTN, DI_RSW_1, DI_RSW_2, DI_RSW_4 und DI_RSW_8

Registerkarte	Einstellung	Wert
General Settings	Pin direction	Input
General Settings	Input Settings Mode	Pull Up
General Settings	Output Settings Mode	Push Pull
General Settings	Output Settings Initial output level	Low
General Settings	Output Settings Driver strength	Don't Care

A.1.6 Konfiguration der DAVE-APPs des Messmoduls als Slave

Tabelle A.16: Konfiguration der DAVE-APP SYSTIMER mit dem Namen SYSTIMER

Registerkarte	Einstellung	Wert
General Settings	SysTick timer period [us]	500
General Settings	Number of software timers	2
Interrupt Settings	SysTick Interrupt Settings Preemption	3

Tabelle A.17: Konfiguration der DAVE-APP CPU_CTRL_XMC1 mit dem Namen CPU_CTRL_XMC1_0

Registerkarte	Einstellung	Wert
General Settings	Number of priority bits for priority grouping	2
General Settings	Debug interface	None
Exception Settings	Enable hard fault debug- ging support	No

Tabelle A.18: Konfiguration der DAVE-APP CLOCK_XMC1 mit dem Namen CLOCK_XMC1_0

Registerkarte	Einstellung	Wert
Clock Control Settings	DCO1 [MHz]	64
Clock Control Settings	DCO2 [kHz]	32.768
Clock Control Settings	PCLK [MHz]	64
Clock Control Settings	MCLK [MHz]	32
Clock Control Settings	RTC_CLOCK [kHz]	32.768
Clock Control Settings	WDT_CLOCK [kHz]	32.768
General Settings	Clock control unit source	DCO1
General Settings	Main clock (MCLK) [MHz]	32.0
General Settings	Actual setting [MHz]	32
General Settings	Fast peripheral clock (PCLK) [MHz]	2 x MCLK
General Settings	Actual setting [MHz]	64
General Settings	RTC clock source	DCO2
Event Settings	Enable loss of DCO1 clock event	No
Event Settings	Enable standby clock failure event	No

Tabelle A.19: Konfiguration der DAVE-APP UART mit dem Namen UART_Internal
Teil 1

Registerkarte	Einstellung	Wert
General Settings	Operation mode	Full Duplex
General Settings	Desired speed [baud]	161718
General Settings	Actual speed [baud]	161718
General Settings	Data bits	8
General Settings	Stop bit	1 Stop Bit
General Settings	Parity selection	Even Parity
Advanced Settings	Protocol Handling Transmit mode	Direct
Advanced Settings	Protocol Handling Receive mode	Direct
Advanced Settings	Protocol Handling Timing Settings Oversampling	16
Advanced Settings	FIFO Settings Enable transmit FIFO	No
Advanced Settings	Size	16
Advanced Settings	FIFO Settings Enable receive FIFO	Yes
Advanced Settings	Size	2
Advanced Settings	Shift Settings MSB first	No
Advanced Settings	Shift Settings Enable data input inversion	No
Advanced Settings	Shift Settings Enable data output inversion	No

Tabelle A.20: Konfiguration der DAVE-APP UART mit dem Namen UART_Internal
Teil 2

Registerkarte	Einstellung	Wert
Interrupt Settings	Transmit Interrupt Priority Preemption priority	3
Interrupt Settings	Transmit End of transmit callback	No
Interrupt Settings	Receive Interrupt Priority Preemption priority	3
Interrupt Settings	Receive End of transmit callback	No
Interrupt Settings	Error Handling Interrupt Priority Preemption priority	3
Interrupt Settings	Error Handling Error Handling Callback Functions Receiver noise detection	No
Interrupt Settings	Error Handling Error Handling Callback Functions Format error in stop bit 0	No
Interrupt Settings	Error Handling Error Handling Callback Functions Format error in stop bit 1	No
Interrupt Settings	Error Handling Error Handling Callback Functions Synchronization break detected	No
Pin Settings	Enable advanced pin characteristics	Yes
Pin Settings	Transmit Mode	Push Pull
Pin Settings	Receive Mode	Pull Up
Pin Settings	Receive Hysteresis	Standard

Tabelle A.21: Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DO_TRX_SD, DO_LED1, DO_LED2 und DO_LED_Y

Registerkarte	Einstellung	Wert
General Settings	Pin direction	Input/Output
General Settings	Input Settings Mode	Tristate
General Settings	Input Settings Hysteresis	Standard
General Settings	Output Settings Mode	Push Pull
General Settings	Output Settings Initial output level	Low

Tabelle A.22: Konfiguration der DAVE-APPs DIGITAL_IO mit den Namen DI_RSW_1, DI_RSW_2, DI_RSW_4 und DI_RSW_8

Registerkarte	Einstellung	Wert
General Settings	Pin direction	Input
General Settings	Input Settings Mode	Pull Up
General Settings	Input Settings Hysteresis	Standard
General Settings	Output Settings Mode	Push Pull
General Settings	Output Settings Initial output level	Low

Tabelle A.23: Konfiguration der DAVE-APPs PIN_INTERRUPT mit dem Namen DI_INT_BTN

Registerkarte	Einstellung	Wert
General Settings	Enable during initialization	Yes
General Settings	Input Settings Mode	Tristate
General Settings	Input Settings Hysteresis	Standard
General Settings	Interrupt Settings Generate interrupt on	Falling Edge
General Settings	Interrupt Settings Interrupt handler	InterruptHandler_BTN
General Settings	Interrupt Settings Interrupt Priority Preemption priority	3

Tabelle A.24: Konfiguration der DAVE-APPs GLOBAL_ADC mit den Namen GLOBAL_ADC_0

Registerkarte	Einstellung	Wert
General Settings	Clock Settings Peripheral bus clock [MHz]	32
General Settings	Clock Settings Desired analog clock [MHz]	30
General Settings	Clock Settings Actual analog clock [MHz]	32
General Settings	Clock Settings Digital clock	fADC
General Settings	Clock Settings Actual digital clock [MHz]	32
General Settings	Enable start up calibration	Yes
General Settings	Analog reference voltage	External reference if VDD \geq 3.0V (upper supply range)

Tabelle A.25: Konfiguration der DAVE-APPs ADC_MEASUREMENT mit den Namen ADC_MEASUREMENT_0

Registerkarte	Einstellung	Wert
General Settings	Measurement Settings Number of measurements	2
General Settings	Measurement Settings Trigger edge selection	No External Trigger
General Settings	Measurement Settings Enable continuous conversion	No
General Settings	Measurement Settings Start conversion after initialization	No
General Settings	Conversion class Settings Conversion Mode	12 Bit Conversion
General Settings	Conversion class Settings Actual sample time [nsec]	125
General Settings	Conversion class Settings Total conversion time [nsec]	1406.25

A.2 Quelltexte

A.2.1 main.c des Überwachungsmoduls als Master

```

/*
TODO Setup for IrDA communication.
After every code generation modify the generated code as follows:

In "Dave\Generated\UART\uart_conf.c"
in function "UART_STATUS_t UART_Internal_init()"
add after "XMC_UART_CH_Init(XMC_USIC_CH_t *channel, const XMC_UART_CH_CONFIG_t *const config);"
and before "XMC_UART_CH_Start(XMC_USIC_CH_t *const channel);"
the following code block:

// manually inserted code for IrDA transceiver
// set sample mode to one sample per bit and the sample point
// to 2/16 of the bit length (after time quanta 1)
UART_Internal.channel->PCR_ASCMode =(uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_SMD_Msk)
& (~USIC_CH_PCR_ASCMode_SP_Msk))
| (((uint32_t) 0UL) << USIC_CH_PCR_ASCMode_SMD_Pos)
| (((uint32_t) 1UL) << USIC_CH_PCR_ASCMode_SP_Pos));
// enable pulse output and set pulse length to 4/16 of the bit length
UART_Internal.channel->PCR_ASCMode = (uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_PL_Msk))
| (((uint32_t) 3UL) << USIC_CH_PCR_ASCMode_PL_Pos));
// invert data output
UART_Internal.channel->SCTR = (uint32_t) (UART_Internal.channel->SCTR
& (~USIC_CH_SCTR_DOCFG_Msk))
| (uint32_t) XMC_USIC_CH_DATA_OUTPUT_MODE_INVERTED;

Alternative configurations and options:

// set sample mode to majority decision and the sample point to 4/16 of the bit length
// (after time quanta 3)
//UART_Internal.channel->PCR_ASCMode = (uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_SMD_Msk)
& (~USIC_CH_PCR_ASCMode_SP_Msk))
| (((uint32_t) 1UL) << USIC_CH_PCR_ASCMode_SMD_Pos)
| (((uint32_t) 3UL) << USIC_CH_PCR_ASCMode_SP_Pos));

// invert data output
//XMC_USIC_CH_SetDataOutputMode(UART_Internal.channel, XMC_USIC_CH_DATA_OUTPUT_MODE_INVERTED);
*/

#include <DAVE.h> //Declarations from DAVE Code Generation (includes SFR declaration)
#include <stdio.h> // for sprintf, (char *__restrict, const char *__restrict, ...)

// 0,5 ms
// time the transceiver needs after the SD (shutdown) signal goes low before it can operate
#define TRANSCEIVER_STARTUP_TIME_IN_US ((uint32_t) 500)

// 0,5 s
// time between slave data update (time before flag for update request is set to true again)
#define SLAVE_DATA_UPDATE_TIME_IN_US ((uint32_t) 500000)

```

A Anhänge

```
// All information about the LCD and the IO-expander is taken from their respective datasheets
// found under the following url:
// https://eater.net/datasheets/HD44780.pdf , accessed on 01.08.2020
// https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf , accessed on 01.08.2020

// instructions of HD44780 LCD controller
// 0b 0000 0001; execution time: ??? s
// Clears entire display and sets DDRAM address 0 in address counter.
#define LCD_CLEAR_DISPLAY ((uint8_t) 1)
// 0b 0000 0010; execution time: 1.52 ms
// Sets DDRAM address 0 in address counter.
// Also returns display from being shifted to original position. DDRAM contents remain unchanged.
#define LCD_RETURN_HOME ((uint8_t) 2)
// 0b 0000 0100; execution time: 37 us
// Sets cursor move direction and specifies display shift.
// These operations are performed during data write and read.
#define LCD_ENTRY_MODE_SET ((uint8_t) 4)
// 0b 0000 1000; execution time: 37 us
// Sets entire display (D) on/off, cursor on/off (C), and
// blinking of cursor position character (B).
#define LCD_DISPLAY_ON_OFF_CONTROL ((uint8_t) 8)
// 0b 0001 0000; execution time: 37 us;
// Moves cursor and shifts display without changing DDRAM contents.
#define LCD_CURSOR_DISPLAY_SHIFT ((uint8_t) 16)
// 0b 0010 0000; execution time: 37 us
// Sets interface data length (DL), number of display lines (N), and character font (F).
#define LCD_FUNCTION_SET ((uint8_t) 32)
// 0b 0100 0000; execution time: 37 us
// Sets CGRAM address. CGRAM data is sent and received after this setting.
#define LCD_SET_CGRAM_ADDRESS ((uint8_t) 64)
// 0b 1000 0000; execution time: 37 us
// Sets DDRAM address. DDRAM data is sent and received after this setting.
#define LCD_SET_DDRAM_ADDRESS ((uint8_t) 128)

// flags for instruction display entry mode
// I/D: Increments (I/D = 1) or decrements (I/D = 0) the DDRAM address by 1 when
// a character code is written into or read from DDRAM. The cursor or blinking moves
// to the right when incremented by 1 and to the left when decremented by 1. The same
// applies to writing and reading of CGRAM.
// 0b 0000 0000
// I/D-bit = 0 = decrement; characters appear from right to left
#define LCD_ENTRY_MODE_CHARACTER_ORDER_RIGHT_TO_LEFT ((uint8_t) 0)
// 0b 0000 0010
// I/D-bit = 1 = increment; characters appear from left to right
#define LCD_ENTRY_MODE_CHARACTER_ORDER_LEFT_TO_RIGHT ((uint8_t) 2)
// S: Shifts the entire display either to the right (I/D = 0) or to the left (I/D = 1)
// when S is 1. The display does not shift if S is 0. If S is 1, it will seem as if the
// cursor does not move but the display does. The display does not shift when reading from
// DDRAM. Also, writing into or reading out from CGRAM does not shift the display.
// 0b 0000 0000
// S-bit; cursor shifts and display stays in place
#define LCD_ENTRY_MODE_SHIFT_CHARACTERS_RIGHT ((uint8_t) 0)
// 0b 0000 0001
// S-bit; cursor stays in place and display shifts to the right (I/D = 0) or to the left (I/D = 1)
#define LCD_ENTRY_MODE_SHIFT_CHARACTERS_LEFT ((uint8_t) 1)

// flags for instruction display on/off control
// D: The display is on when D is 1 and off when D is 0. When off, the display data remains
// in DDRAM, but can be displayed instantly by setting D to 1.
// 0b 0000 0100
// D-bit; turn on display
#define LCD_DISPLAY_ON ((uint8_t) 4)
// 0b 0000 0000
// D-bit; turn off display (data remains in memory)
#define LCD_DISPLAY_OFF ((uint8_t) 0)
// C: The cursor is displayed when C is 1 and not displayed when C is 0. Even if the cursor
// disappears, the function of I/D or other specifications will not change during display
// data write. The cursor is displayed using 5 dots in the 8th line for 5 x 8 dot character
// font selection and in the 11th line for the 5 x 10 dot character font selection (Figure 13).
```

```

// 0b 0000 0010; C-bit
// display cursor
#define LCD_CURSOR_ON ((uint8_t) 2)
// 0b 0000 0000; C-bit;
// don't display cursor
#define LCD_CURSOR_OFF ((uint8_t) 0)
// B: The character indicated by the cursor blinks when B is 1 (Figure 13). The blinking is
// displayed as switching between all blank dots and displayed characters at a speed of
// 409.6 ms intervals when fcp or fOSC is 250 kHz. The cursor and blinking can be set to
// display simultaneously. (The blinking frequency changes according to fOSC or the reciprocal
// of fcp. For example, when fcp is 270 kHz, 409.6 x 250/270 = 379.2 ms.)
// 0b 0000 0001; B-bit
// blink character that is indicated cursor
#define LCD_BLINK_ON ((uint8_t) 1)
// 0b 0000 0000; B-bit
// don't blink character that is indicated cursor
#define LCD_BLINK_OFF ((uint8_t) 0)

// flags for instruction cursor or display shift
// S/C: Select whether display shifts or the cursor moves.
// 0b 0000 0000; S/C-bit;
#define LCD_CURSOR_DISPLAY_CURSOR_SHIFT ((uint8_t) 0)
// 0b 0000 1000; S/C-bit;
#define LCD_CURSOR_DISPLAY_DISPLAY_SHIFT ((uint8_t) 8)
// R/L: Select what direction the display or cursor should move.
// 0b 0000 0100; R/L-bit;
#define LCD_CURSOR_DISPLAY_MOVE_RIGHT ((uint8_t) 4)
// 0b 0000 0000; R/L-bit;
#define LCD_CURSOR_DISPLAY_MOVE_LEFT ((uint8_t) 0)

// flags for instruction function set
// DL: Sets the interface data length. Data is sent or received in 8-bit lengths (DB7 to DB0)
// when DL is 1, and in 4-bit lengths (DB7 to DB4) when DL is 0. When 4-bit length is selected,
// data must be sent or received twice.
// 0b 0001 0000; DL-bit
// set number of data bits to 8
#define LCD_FUNCTION_DATA_BITS_8 ((uint8_t) 16)
// 0b 0000 0000; DL-bit
// set number of data bits to 4
#define LCD_FUNCTION_DATA_BITS_4 ((uint8_t) 0)
// N: Sets the number of display lines.
// 0b 0000 1000; N-bit
// set number of lines on the display to 2
#define LCD_FUNCTION_LINES_2 ((uint8_t) 8)
// 0b 0000 0000; N-bit
// set number of lines on the display to 1
#define LCD_FUNCTION_LINES_1 ((uint8_t) 0)
// F: Sets the character font.
// 0b 0000 0100; F-bit
// set character array size in dots/pixels to 5 wide and 10 tall
#define LCD_FUNCTION_DOTS_5X10 ((uint8_t) 4)
// 0b 0000 0000; F-bit
// set character array size in dots/pixels to 5 wide and 8 tall
#define LCD_FUNCTION_DOTS_5X8 ((uint8_t) 0)

// execution times
// execution time in microseconds for instruction return home
#define LCD_INSTRUCTION_EXECUTION_TIME_RETURN_HOME_IN_US ((uint32_t) 1520)
// execution time in microseconds for instruction entry mode set
#define LCD_INSTRUCTION_EXECUTION_TIME_ENTRY_MODE_SET_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction display on/off control
#define LCD_INSTRUCTION_EXECUTION_TIME_DISPLAY_ON_OFF_CONTROL_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction cursor shift
#define LCD_INSTRUCTION_EXECUTION_TIME_CURSOR_DISPLAY_SHIFT_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction function set
#define LCD_INSTRUCTION_EXECUTION_TIME_FUNCTION_SET_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction set CGRAM address
#define LCD_INSTRUCTION_EXECUTION_TIME_SET_CGRAM_ADDRESS_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction set DDRAM address

```

A Anhänge

```
#define LCD_INSTRUCTION_EXECUTION_TIME_SET_DDRAM_ADDRESS_IN_US ((uint32_t) 37)
// execution time in microseconds for instruction read busy flag & address
#define LCD_INSTRUCTION_EXECUTION_TIME_READ_BUSY_FLAG_AND_ADDRESS_IN_US ((uint32_t) 0)
// execution time in microseconds for instruction write to CGRAM or DDRAM
#define LCD_INSTRUCTION_EXECUTION_TIME_WRITE_DATA_TO_CG_OR_DDRAM_IN_US ((uint32_t) 41)
// execution time in microseconds for instruction read from CGRAM or DDRAM
#define LCD_INSTRUCTION_EXECUTION_TIME_READ_DATA_FROM_CG_OR_DDRAM_IN_US ((uint32_t) 41)
// execution time in microseconds for instruction clear display
// write space code (20H) into all DDRAM addresses and then set DDRAM address 0 and return
// the display to its original status if it was shifted
#define LCD_INSTRUCTION_EXECUTION_TIME_CLEAR_DISPLAY_IN_US ((uint32_t)
(LCD_INSTRUCTION_EXECUTION_TIME_SET_DDRAM_ADDRESS_IN_US
+ 80U * LCD_INSTRUCTION_EXECUTION_TIME_WRITE_DATA_TO_CG_OR_DDRAM_IN_US
+ LCD_INSTRUCTION_EXECUTION_TIME_RETURN_HOME_IN_US))

// the execution time is multiplied by this value (if the busy flag is not checked the
// datasheet asks to wait longer than the execution instruction time)
#define LCD_BUSY_FLAG_WAITING_TIME_FACTOR ((uint32_t) 2)
// the datasheet asks to wait for at least 450 ns
#define LCD_ENABLE_PULSE_LENGTH_IN_US ((uint32_t) 1)
// the datasheet asks to wait for at least t_BUF >= 4,7 us
#define LCD_I2C_PAUSE_LENGTH_IN_US ((uint32_t) 5)

// PINOUT
// LCD EXP Function
// VSS GND power 0 V supply
// VDD VCC power 5 V supply
// V0 VR contrast setting
// RS P0 register select
// RW P1 read / not write
// E P2 enable
// D0 NC DB0
// D1 NC DB1
// D2 NC DB2
// D3 NC DB3
// D4 P4 DB4
// D5 P5 DB5
// D6 P6 DB6
// D7 P7 DB7
// A VCC supply for back light via jumper
// K K open circuit when P3 is pulled low

// Bit order:
// MSB LSB
// D7 D6 D5 D4 BL E RW RS

// code used for verification:
//data = 0U; // none
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 1U; // RS
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 2U; // RnW
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 4U; // EN
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 8U; // screen backlight
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 16U; // DB4
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 32U; // DB5
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 64U; // DB6
//i2c_transmit(&data, LCD_I2C_ADDRESS);
//data = 128U; // DB7
//i2c_transmit(&data, LCD_I2C_ADDRESS);

// number of data bits
```

A Anhänge

```
#define LCD_NUMBER_OF_BITS_DATA ((uint32_t) 4)
// 0b 1111 0000; mask for data bits
#define LCD_MASK_DB_BITS ((uint8_t) 240)
// 0b 0000 1000; mask for backlight (function of IO-expander)
#define LCD_MASK_BACKLIGHT_BIT ((uint8_t) 8)
// 0b 0000 0100; mask for enable bit
#define LCD_MASK_EN_BIT ((uint8_t) 4)
// 0b 0000 0010; mask for read / not write bit
#define LCD_MASK_RnW_BIT ((uint8_t) 2)
// 0b 0000 0001; mask for register select bit
#define LCD_MASK_RS_BIT ((uint8_t) 1)

#define LCD_I2C_ADDRESS ((uint8_t) 39) // 0b 0010 0111 I2C address of LCD

#define LCD_NUMBER_OF_COLUMNS ((uint32_t) 20) // 20; number of columns on the LCD
#define LCD_NUMBER_OF_ROWS ((uint32_t) 4) // 4; number of rows on the LCD

#define LCD_ENTRIES_PER_ROW ((uint32_t) 3U) // number of entries per row
// number characters available for each entry in a row
#define LCD_CHARACTERS_PER_ENTRY (LCD_NUMBER_OF_COLUMNS / LCD_ENTRIES_PER_ROW)

#define EXTERNAL_UART_NUMBER_OF_BITS_ADDRESS ((uint32_t) 4) // number of address bits
#define EXTERNAL_UART_NUMBER_OF_BITS_DATA ((uint32_t) 4) // number of data bits

// (2 ^ EXTERNAL_UART_NUMBER_OF_BITS_ADDRESS - 1U) << EXTERNAL_UART_NUMBER_OF_BITS_DATA
// mask for address
#define EXTERNAL_UART_MASK_ADDRESS ((uint8_t) 240)
// 2 ^ EXTERNAL_UART_NUMBER_OF_BITS_DATA - 1U
// mask for data
#define EXTERNAL_UART_MASK_DATA ((uint8_t) 15)

#define EXTERNAL_UART_ADDRESS_MASTER ((uint8_t) 0) // address of master transceiver
#define EXTERNAL_UART_ADDRESS_EXTERNAL ((uint8_t) 1) // address of external device

// message for unknown command
#define EXTERNAL_UART_MESSAGE_UNKNOWN_COMMAND ((uint8_t) 1)
// command and message for requesting a response
#define EXTERNAL_UART_COMMANDMESSAGE_PING ((uint8_t) 2)
// message for no response received within timeout time
#define EXTERNAL_UART_MESSAGE_SLAVE_DOESNT_RESPOND ((uint8_t) 3)
// message for not executed and ignored command
#define EXTERNAL_UART_MESSAGE_COMMAND_IGNORED ((uint8_t) 4)
// command and message for aborting current action
#define EXTERNAL_UART_COMMANDMESSAGE_ABORT ((uint8_t) 5)
// command for sending addresses of all currently connected slaves
#define EXTERNAL_UART_COMMAND_SHOW_ALL_AVAILABLE_SLAVE_ADDRESSES ((uint8_t) 7)
// command for sending data of all currently connected slaves
#define EXTERNAL_UART_COMMAND_SHOW_DATA_OF_ALL_AVAILABLE_SLAVES ((uint8_t) 8)
// command for sending data of all slaves
#define EXTERNAL_UART_COMMAND_SHOW_DATA_OF_ALL_SLAVES ((uint8_t) 9)
// command for toggling periodic sending of all slave data after it was updated
#define EXTERNAL_UART_COMMAND_TOGGLE_AUTO_UPDATE_SLAVE_DATA ((uint8_t) 10)
// ...
// ...
// #define EXTERNAL_UART_ ((uint8_t) 15)

// transmission time = (1 + 8 + 1 + 1) / (115200 1/s) = 95,486 us
// 1000 us; maximum time a slave has to respond to any command
#define EXTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US ((uint32_t) 1000)

#define INTERNAL_UART_NUMBER_OF_BITS_ADDRESS ((uint32_t) 4) // number of address bits
#define INTERNAL_UART_NUMBER_OF_BITS_DATA ((uint32_t) 4) // number of data bits

// (2 ^ INTERNAL_UART_NUMBER_OF_BITS_ADDRESS - 1U) << INTERNAL_UART_NUMBER_OF_BITS_ADDRESS
// mask for address
#define INTERNAL_UART_MASK_ADDRESS ((uint8_t) 240)
```


A Anhänge

```
// 2 ^ INTERNAL_UART_NUMBER_OF_BITS_DATA - 1U
// mask for data
#define INTERNAL_UART_MASK_DATA ((uint8_t) 15)

// address of master transceiver
#define INTERNAL_UART_ADDRESS_MASTER ((uint8_t) 0)
// lowest address of first cell slave
#define INTERNAL_UART_ADDRESS_SLAVE_START ((uint8_t) 1)
// highest address of last cell slave
#define INTERNAL_UART_ADDRESS_SLAVE_END ((uint8_t) 12)
// 2 ^ INTERNAL_UART_NUMBER_OF_BITS_ADDRESS - 1U
// broadcast address to reach all bus participants
#define INTERNAL_UART_ADDRESS_BROADCAST ((uint8_t) 15)

// message for unknown command
#define INTERNAL_UART_MESSAGE_UNKNOWN_COMMAND ((uint8_t) 1)
// command and message for requesting a response
#define INTERNAL_UART_COMMANDMESSAGE_PING ((uint8_t) 2)
// command and message for aborting current action
#define INTERNAL_UART_COMMANDMESSAGE_ABORT ((uint8_t) 3)
// command for sending all available data
#define INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES ((uint8_t) 4)
//...
// command for ...
// #define INTERNAL_UART_ (INTERNAL_UART_MASK_DATA)

// number of retries of command for sending all data packages
#define INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES_MAXIMUM_TRIES ((uint32_t) 2)

// transmission time of one character = (1 + 8 + 1 + 1) / (115200 1/s) = 95,486 us
// 1000 us; maximum time a slave has to respond to any command
#define INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US ((uint32_t) 1000)

// number of bits sent by the slave for the rotary switch position
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH ((uint32_t) 4)
// number of bits sent by the slave for the ADC value of potentiometer 1 (or supply voltage)
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1 ((uint32_t) 12)
// number of bits sent by the slave for the ADC value of potentiometer 2 (or temperature)
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2 ((uint32_t) 12)

// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH - 1U
// mask for rotary switch position bits to be send to the master
#define SLAVE_DATA_MASK_ROTARY_SWITCH ((uint32_t) 255)
// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1 - 1U
// mask for ADC value of potentiometer 1 (or supply voltage) bits to be send to the master
#define SLAVE_DATA_MASK_POTENTIOMETER_1 ((uint32_t) 4095)
// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2 - 1U
// mask for ADC value of potentiometer 2 (or temperature) bits to be send to the master
#define SLAVE_DATA_MASK_POTENTIOMETER_2 ((uint32_t) 4095)

// number of transmissions from the slave after COMMAND_SEND_ALL_DATA_PACKAGES
#define SLAVE_DATA_NUMBER_OF_PACKAGES ((SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1)
/ INTERNAL_UART_NUMBER_OF_BITS_DATA)

bool ui_buttonStatePrevious = false; // previous state of the button
bool ui_buttonState = false; // current state of the button
uint8_t ui_rotarySwitchPositionPrevious = 0U; // previous position of the rotary switch
uint8_t ui_rotarySwitchPosition = 0U; // current position of the rotary switch

// only read and write in UART receive event handler; buffer for received data
```

A Anhänge

```
volatile uint8_t externalUART_inputBuffer = 0U;
// only read and write in UART transmission function; buffer for transmission data
uint8_t externalUART_outputBuffer = 0U;

uint8_t externalUART_inputAddress = 0U; // source address were the received data came from
uint8_t externalUART_inputData = 0U; // data received from source

uint8_t externalUART_outputAddress = 0U; // destination address were data will be send to
uint8_t externalUART_outputData = 0U; // data to send to destination

// only read and write in UART receive event handler; buffer for received data
volatile uint8_t internalUART_inputBuffer = 0U;
// only read and write in UART transmission function; buffer for transmission data
uint8_t internalUART_outputBuffer = 0U;

uint8_t internalUART_inputAddress = 0U; // source address were the received data came from
uint8_t internalUART_inputData = 0U; // data received from source

uint8_t internalUART_outputAddress = 0U; // destination address were data will be send to
uint8_t internalUART_outputData = 0U; // data to send to destination

struct struct_slave {
    uint8_t address; // address of slave
    bool connectionState; // state of connection: false: not connected | true: connected
    // state of connection changed:
    // false: no change
    // true: if connectionState == true then newly connected else connection lost
    bool connectionStateChanged;
    // indicates problems with the communication, if a connection is established
    bool connectionProblems;
    uint8_t rotarySwitch; // position of rotary switch
    // ADC value of potentiometer 1; this analog input can also be used for supply voltage measurement
    uint16_t potentiometer1ADC;
    // ADC value of potentiometer 2; this analog input can also be used for temperature measurement
    uint16_t potentiometer2ADC;
};

// struct array for all slave data
struct struct_slave slaveData[INTERNAL_UART_ADDRESS_SLAVE_END
- INTERNAL_UART_ADDRESS_SLAVE_START + 1U] = {};
// number of elements in slaveData
uint32_t slaveDataNumberOfElements = sizeof(slaveData) / sizeof(slaveData[0]);

// flag that indicates whether all slave data shall be send to the external device via UART
// after it was updated
bool externalUART_autoUpdateSlaveData = false;

uint32_t slavDataUpdatePeriodicTimerID = 0U; // timer ID for controlling the timer
volatile bool slaveDataUpdateFlag = false; // true if an update of slave data is required

struct struct_oneShotTimer {
    uint32_t id; // timer ID for controlling one the timer
    volatile bool running; // true while one shot timer is running
};

struct struct_oneShotTimer oneShotTimer = {.id = 0U, .running = false}; // one shot timer parameters

/**
 * restart software timer and check whether software timer restart failed
 */
void oneShotTimerStart(uint32_t timeToWait, bool waitForTimerToElapse)
{
```

```

    if(SYSTIMER_RestartTimer(oneShotTimer.id, timeToWait) == SYSTIMER_STATUS_SUCCESS)
    {
        oneShotTimer.running = true;
    }
    else // error handler code
    {
        XMC_DEBUG("\nTimer_start_failed\n");
        while(true);
    }

    // wait for one shot timer to elapse
    if(waitForTimerToElapse)
    {
        while(oneShotTimer.running);
    }
}

/**
 * callback function of on shot timer
 * stop timer and indicate that timer has elapsed
 */
void oneShotTimerElapsed(void)
{
    // stop software timer
    SYSTIMER_StopTimer(oneShotTimer.id);

    oneShotTimer.running = false;
}

/**
 * callback function of on periodic timer
 * set flag to indicate that the periodic timer has elapsed
 */
void slaveDataUpdatePeriodicTimerElapsed(void)
{
    slaveDataUpdateFlag = true; // set flag
}

/*
 * callback for NACK
 * the transmission is aborted if no acknowledge is received
 */
void I2C_NACK_CALLBACK(void)
{
    I2C_MASTER_AbortTransmit(&I2C_LCD);
}

void i2c_transmit(uint8_t* data, uint8_t slaveAddress)
{
    I2C_MASTER_Transmit(&I2C_LCD, true, slaveAddress << 1U, data, 1U, true);
    while(I2C_MASTER_IsTxBusy(&I2C_LCD));
}

/**
 * transmit 4 control bits and 4 data bits to the LCD via I2C
 * backlight will be enabled
 * data: data of which the 4 MSB will be transmitted via I2C
 * dataNotInstruction: 0 for instruction, 1 for data

```

A Anhänge

```
* executionTime: time to wait in microseconds after transmission finished
*/
void i2c_lcdTransmit4MSB(uint8_t data, bool dataNotInstruction, uint32_t executionTime)
{
    // mask 4 MSB
    data = (data & LCD_MASK_DB_BITS) | LCD_MASK_BACKLIGHT_BIT;

    // enable backlight
    data = data | LCD_MASK_BACKLIGHT_BIT;

    // set Register Select bit
    data = ((dataNotInstruction) ? (data | LCD_MASK_RS_BIT) : (data & ~LCD_MASK_RS_BIT));

    // 1: send via I2C with enable bit low
    data = data & ~LCD_MASK_EN_BIT;
    i2c_transmit(&data, LCD_I2C_ADDRESS);

    // wait for I2C bus pause time time to elapse
    // start software timer and wait for it to elapse
    oneShotTimerStart(LCD_I2C_PAUSE_LENGTH_IN_US, true);

    // 2: send via I2C with enable bit high
    data = data | LCD_MASK_EN_BIT;
    i2c_transmit(&data, LCD_I2C_ADDRESS);

    // wait for I2C bus pause time time to elapse
    // start software timer and wait for it to elapse
    oneShotTimerStart((LCD_ENABLE_PULSE_LENGTH_IN_US < LCD_I2C_PAUSE_LENGTH_IN_US)
        ? (LCD_I2C_PAUSE_LENGTH_IN_US) : (LCD_ENABLE_PULSE_LENGTH_IN_US), true);

    // 3: send via I2C with enable bit low
    data = data & ~LCD_MASK_EN_BIT;
    i2c_transmit(&data, LCD_I2C_ADDRESS);

    // wait for execution time to elapse (or I2C bus pause time if longer)
    // start software timer and wait for it to elapse
    oneShotTimerStart((executionTime < LCD_I2C_PAUSE_LENGTH_IN_US)
        ? (LCD_I2C_PAUSE_LENGTH_IN_US) : (executionTime), true);
}

/**
 * transmit instruction via I2C to the LCD in 4 bit mode
 * data: data to transmit to the LCD via I2C
 * dataNotInstruction: 0 for instruction, 1 for data
 * executionTime: time to wait in microseconds after transmission finished
 */
void i2c_lcdTransmit(uint8_t data, bool dataNotInstruction, uint32_t executionTime)
{
    // send the first 4 bits
    i2c_lcdTransmit4MSB(data & LCD_MASK_DB_BITS, dataNotInstruction, LCD_I2C_PAUSE_LENGTH_IN_US);
    // send the last 4 bits
    i2c_lcdTransmit4MSB((data << LCD_NUMBER_OF_BITS_DATA)
        & LCD_MASK_DB_BITS, dataNotInstruction, executionTime);
}

/**
 * set the cursor to the desired position
 */
void lcd_setCursorPosition(uint8_t row, uint8_t column)
{
    // offset to add to the address to display the character in the desired row
    uint8_t row_offsets[] = { 0U, 64U, 20U, 84U };

    if(row > LCD_NUMBER_OF_ROWS)
    {

```

```

        row = LCD_NUMBER_OF_ROWS; // set last row
    }

    if(column > LCD_NUMBER_OF_COLUMNS)
    {
        column = LCD_NUMBER_OF_COLUMNS; // set last column
    }

    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than the
    // execution instruction time)
    i2c_lcdTransmit(LCD_SET_DDRAM_ADDRESS | (column - 1U + row_offsets[row - 1U]), false ,
LCD_BUSY_FLAG_WAITING_TIME_FACTOR * LCD_INSTRUCTION_EXECUTION_TIME_SET_DDRAM_ADDRESS_IN_US);
}

/**
 * transmit string via I2C to the LCD at the set cursor position
 */
void lcd_print(char *pointer, uint8_t row, uint8_t column)
{
    // set cursor position
    lcd_setCursorPosition(row, column);

    // iterate over characters of string and send individual characters to the LCD
    while (*pointer)
    {
        // transmit data and wait for double the instruction execution time
        // (if the busy flag is not checked the datasheet asks to wait longer than the
        // execution instruction time)
        i2c_lcdTransmit(*pointer, true, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_WRITE_DATA_TO_CG_OR_DDRAM_IN_US);
        *pointer++;
    }
}

/**
 * executes the clear LCD function of the LCD
 */
void lcd_clearDisplay(void)
{
    // 0b 0000 0001
    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than the
    // execution instruction time)
    i2c_lcdTransmit(LCD_CLEAR_DISPLAY, false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_CLEAR_DISPLAY_IN_US);
}

/**
 * writes space to all visible characters of the specified row on the LCD
 * (does not reset cursor)
 */
void lcd_clearRow(uint32_t row)
{
    // set cursor position
    lcd_setCursorPosition(row, 1U);
    // write space (decimal 32) to all characters of the row
    for(uint32_t column = 1U; column <= LCD_NUMBER_OF_COLUMNS; column += 1U)
    {
        // transmit data and wait for double the instruction execution time
        // (if the busy flag is not checked the datasheet asks to wait longer than the
        // execution instruction time)
        i2c_lcdTransmit(32U, true, LCD_BUSY_FLAG_WAITING_TIME_FACTOR

```

A Anhänge

```
        * LCD_INSTRUCTION_EXECUTION_TIME_WRITE_DATA_TO_CG_OR_DDRAM_IN_US);
    }
}

// LCD initialization in 4 bit mode as described on page 46 of the datasheet
void lcd_init()
{
    // the datasheet asks to wait for more than 15 ms after VCC is 4,5 V or higher
    oneShotTimerStart(50000U, true); // start software timer with 50 ms and wait for it to elapse

    // set lcd to 8 data bit mode
    // 0b 0011 0000
    // transmit the 4 MSBs of the instruction and wait for 5 ms
    // (the datasheet asks to wait for more than 4,1 ms)
    i2c_lcdTransmit4MSB(LCD_FUNCTION_SET | LCD_FUNCTION_DATA_BITS_8, false, 5000U);

    // set lcd to 8 data bit mode a second time
    // 0b 0011 0000
    // transmit the 4 MSBs of the instruction a second time and wait for 0,2 ms
    // (the datasheet asks to wait for more than 0,1 ms)
    i2c_lcdTransmit4MSB(LCD_FUNCTION_SET | LCD_FUNCTION_DATA_BITS_8, false, 200U);

    // set lcd to 8 data bit mode a third time
    // 0b 0011 0000
    // transmit the 4 MSBs of the instruction a third time and wait for double the
    // instruction execution time (if the busy flag is not checked the datasheet asks
    // to wait longer than the execution instruction time)
    i2c_lcdTransmit4MSB(LCD_FUNCTION_SET | LCD_FUNCTION_DATA_BITS_8, false,
LCD_BUSY_FLAG_WAITING_TIME_FACTOR * LCD_INSTRUCTION_EXECUTION_TIME_FUNCTION_SET_IN_US);

    // set lcd to 4 data bit mode
    // 0b 0010 0000
    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than
    // the execution instruction time)
    i2c_lcdTransmit4MSB(LCD_FUNCTION_SET | LCD_FUNCTION_DATA_BITS_4, false,
LCD_BUSY_FLAG_WAITING_TIME_FACTOR * LCD_INSTRUCTION_EXECUTION_TIME_FUNCTION_SET_IN_US);

    // set number display lines and character font (dots or pixels per character)
    // 0b 0000 1000
    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than the
    // execution instruction time)
    i2c_lcdTransmit(LCD_FUNCTION_LINES_2 | LCD_FUNCTION_DOTS_5X8, false,
LCD_BUSY_FLAG_WAITING_TIME_FACTOR * LCD_INSTRUCTION_EXECUTION_TIME_FUNCTION_SET_IN_US);

    // turn display off, turn cursor off, turn blinking character off
    // 0b 0000 1000
    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than the
    // execution instruction time)
    i2c_lcdTransmit(LCD_DISPLAY_ON_OFF_CONTROL | LCD_DISPLAY_OFF | LCD_CURSOR_OFF | LCD_BLINK_OFF,
false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_DISPLAY_ON_OFF_CONTROL_IN_US);

    // clear display
    lcd_clearDisplay();

    // set text display direction
    // 0b 0000 0111
    // transmit instruction and wait for double the instruction execution time
    // (if the busy flag is not checked the datasheet asks to wait longer than the
    // execution instruction time)
    i2c_lcdTransmit(LCD_ENTRY_MODE_SET | LCD_ENTRY_MODE_CHARACTER_ORDER_LEFT_TO_RIGHT
| LCD_ENTRY_MODE_SHIFT_CHARACTERS_RIGHT, false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_ENTRY_MODE_SET_IN_US);
```

A Anhänge

```
// initialization complete
// continue setup

// set cursor shift and direction of shift
// 0b 0001 0100
// transmit instruction and wait for double the instruction execution time
// (if the busy flag is not checked the datasheet asks to wait longer than the
// execution instruction time)
i2c_lcdTransmit(LCD_CURSOR_DISPLAY_SHIFT | LCD_CURSOR_DISPLAY_CURSOR_SHIFT
| LCD_CURSOR_DISPLAY_MOVE_RIGHT, false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_CURSOR_DISPLAY_SHIFT_IN_US);

// set cursor to home position
// 0b 0000 0010
// transmit instruction and wait for double the instruction execution time
// (if the busy flag is not checked the datasheet asks to wait longer than the
// execution instruction time)
i2c_lcdTransmit(LCD_RETURN_HOME, false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_RETURN_HOME_IN_US);

// turn display on, turn cursor off, turn blinking character off
// 0b 0000 1000
// transmit instruction and wait for double the instruction execution time
// (if the busy flag is not checked the datasheet asks to wait longer than the
// execution instruction time)
i2c_lcdTransmit(LCD_DISPLAY_ON_OFF_CONTROL | LCD_DISPLAY_ON | LCD_CURSOR_OFF | LCD_BLINK_OFF,
false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
* LCD_INSTRUCTION_EXECUTION_TIME_DISPLAY_ON_OFF_CONTROL_IN_US);

// set cursor visible for testing; turn display on, turn cursor on, turn blinking character on
// 0b 0000 1010
// transmit instruction and wait for double the instruction execution time
// (if the busy flag is not checked the datasheet asks to wait longer than the
// execution instruction time)
//i2c_lcdTransmit(LCD_DISPLAY_ON_OFF_CONTROL | LCD_DISPLAY_ON | LCD_CURSOR_ON | LCD_BLINK_ON,
//false, LCD_BUSY_FLAG_WAITING_TIME_FACTOR
// * LCD_INSTRUCTION_EXECUTION_TIME_DISPLAY_ON_OFF_CONTROL_IN_US);
}

/**
 * If the UART input buffer is not empty, read the first (oldest) data from the UART input buffer
 * to extract and save the address and first (oldest) data.
 * returns true if data was available
 */
bool externalUART_receiveFirst()
{
    bool dataAvailable = false;

    // read oldest data from receive FIFO and thereby shift all receive FIFO data
    if (!UART_IsRXFIFOEmpty(&UART_External))
    {
        externalUART_inputBuffer = UART_GetReceivedWord(&UART_External);
        externalUART_inputAddress = (uint8_t) ((externalUART_inputBuffer
& EXTERNAL_UART_MASK_ADDRESS) >> EXTERNAL_UART_NUMBER_OF_BITS_DATA);
        externalUART_inputData = (uint8_t) (externalUART_inputBuffer & EXTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate that no data has been received
}

/**
 * First check whether reception is in progress and if so, wait until it's finished.
 * Then, if the UART input buffer is not empty, read the first (oldest) data from the
 * UART input buffer until it is empty to save the last (newest) data.
 */
```

A Anhänge

```
* This discards all older received data in the UART input buffer.
* returns true if data was available
*/
bool externalUART_receiveLast()
{
    bool dataAvailable = false;

    while(UART_IsRxBusy(&UART_External)); // wait for any running reception to finish

    // read oldest data from receive FIFO and thereby shift all receive FIFO data until
    // the receive FIFO is empty to get the latest received data
    while(!UART_IsRXFIFOEmpty(&UART_External))
    {
        externalUART_inputBuffer = UART_GetReceivedWord(&UART_External);
        externalUART_inputAddress = (uint8_t) ((externalUART_inputBuffer
        & EXTERNAL_UART_MASK_ADDRESS) >> EXTERNAL_UART_NUMBER_OF_BITS_DATA);
        externalUART_inputData = (uint8_t) (externalUART_inputBuffer & EXTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate whether data has been received
}

/**
 * transmit one symbol via UART
 */
void externalUART_transmit(void)
{
    externalUART_outputBuffer = (externalUART_outputAddress << EXTERNAL_UART_NUMBER_OF_BITS_DATA)
    | externalUART_outputData; // set output buffer
    while(UART_IsTxBusy(&UART_External)); // wait for previous transmission to finish
    UART_Transmit(&UART_External, &externalUART_outputBuffer, 1U);
}

/**
 * transmit a string via UART
 */
void externalUART_transmitString(char *pointer)
{
    UART_Transmit(&UART_External, (uint8_t*) pointer, strlen(pointer));
}

/**
 * If the UART input buffer is not empty, read the first (oldest) data from the
 * UART input buffer to extract and save the address and first (oldest) data.
 * returns true if data was available
 */
bool internalUART_receiveFirst()
{
    bool dataAvailable = false;

    // read oldest data from receive FIFO and thereby shift all receive FIFO data
    if(!UART_IsRXFIFOEmpty(&UART_Internal))
    {
        internalUART_inputBuffer = UART_GetReceivedWord(&UART_Internal);
        internalUART_inputAddress = (uint8_t) ((internalUART_inputBuffer
        & INTERNAL_UART_MASK_ADDRESS) >> INTERNAL_UART_NUMBER_OF_BITS_DATA);
        internalUART_inputData = (uint8_t) (internalUART_inputBuffer & INTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate that no data has been received
}
}
```


A Anhänge

```
/**
 * First check whether reception is in progress and if so, wait until it's finished.
 * Then, if the UART input buffer is not empty, read the first (oldest) data from the
 * UART input buffer until it is empty to save the last (newest) data.
 * This discards all older received data in the UART input buffer.
 * returns true if data was available
 */
bool internalUART_receiveLast()
{
    bool dataAvailable = false;

    while(UART_IsRxBusy(&UART_Internal)); // wait for any running reception to finish

    // read oldest data from receive FIFO and thereby shift all receive FIFO data until the
    // receive FIFO is empty to get the latest received data
    while(!UART_IsRXFIFOEmpty(&UART_Internal))
    {
        internalUART_inputBuffer = UART_GetReceivedWord(&UART_Internal);
        internalUART_inputAddress = (uint8_t) ((internalUART_inputBuffer
        & INTERNAL_UART_MASK_ADDRESS) >> INTERNAL_UART_NUMBER_OF_BITS_DATA);
        internalUART_inputData = (uint8_t) (internalUART_inputBuffer & INTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate that no data has been received
}

/**
 * transmit one symbol via UART
 * Any received data which was received while sending data is discarded.
 */
void internalUART_transmit(void)
{
    /*
     * The while loops "while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal))"
     * before and after the function
     * "UART_Transmit(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * shouldn't be necessary when receive mode is set to "direct" which blocks the CPU
     * until all data is sent or received.
     * (See description of function
     * "UART_lStartTransmitPolling(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * and
     * "UART_lStartReceivePolling(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * in "Dave\Generated\UART\uart.c".)
     */
    internalUART_outputBuffer = (internalUART_outputAddress << INTERNAL_UART_NUMBER_OF_BITS_DATA)
    | internalUART_outputData; // set output buffer
    // wait for previous transmission or reception to finish
    while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal));
    // transmit contents of output buffer
    UART_Transmit(&UART_Internal, &internalUART_outputBuffer, 1U);
    // wait for transmission or reception to finish
    while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal));
    // wait for echoed data from the IrDA transmitter and discard it
    while(!internalUART_receiveLast());
}

/*
 * send all slave addresses to the external device via UART
 */
void showAllAvailableSlaveAddresses(void)
{
```

```

// iterate over all slaves
for(uint32_t idx = 0U; idx < slaveDataNumberOfElements; idx += 1U)
{
    // check and handle newly received command from the external device
    if(internalUART_receiveFirst()) // check for new received data
    { // check command
        // abort execution
        if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_ABORT)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_ABORT;
            externalUART_transmit();
            break; // exit for loop
        }
        // inform external device that command is being ignored
        else
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = EXTERNAL_UART_MESSAGE_COMMAND_IGNORED;
            externalUART_transmit();
        }
    }
    // transmit all known slave addresses
    else if(slaveData[idx].connectionState)
    {
        externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
        externalUART_outputData = slaveData[idx].address;
        externalUART_transmit();
    }
}
}

/**
 * forward the message from the external device via external UART to the slaves via
 * internal UART and return the first answer of slave or a timeout message if the
 * slave didn't respond
 */
void forwardCommandAndWaitForAnswer(void)
{
    internalUART_outputAddress = externalUART_inputAddress;
    internalUART_outputData = externalUART_inputData;
    internalUART_transmit();

    // start timer for response timeout
    oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, false);

    // wait for answer of any slave or until INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US has
    // elapsed
    while(true)
    {
        // check and handle newly received command from the external device
        if(internalUART_receiveFirst()) // check for new received data
        {
            // abort execution
            if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_ABORT)
            {
                oneShotTimerElapsed(); // stop timer
                externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
                externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_ABORT;
                externalUART_transmit();
                break; // exit while loop
            }
            // inform external device that command is being ignored
            else
            {
                externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
                externalUART_outputData = EXTERNAL_UART_MESSAGE_COMMAND_IGNORED;
            }
        }
    }
}

```

```

        externalUART_transmit();
    }
}
// transmission received from the slave
else if(internalUART_receiveLast()) // check for new received data
{
    oneShotTimerElapsed(); // stop timer
    externalUART_outputAddress = internalUART_inputAddress;
    externalUART_outputData = internalUART_inputData;
    externalUART_transmit();
    break; // exit while loop
}
// timeout; slave didn't answer
else if (!oneShotTimer.running)
{
    externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
    externalUART_outputData = EXTERNAL_UART_MESSAGE_SLAVE_DOESNT_RESPOND;
    externalUART_transmit();
    break; // exit while loop
}
}
}

/**
 * request information from slave and save it if the slave responds
 */
void slaveDataUpdate(uint8_t *address)
{
    uint32_t commandTryCounter = 0U; // number of times the command has been sent
    uint32_t receivedDataBuffer; // buffer for data from received transmissions
    uint32_t packageCounter; // number of received transmissions from the slave
    // index of the slave in the slaveData array with the given address
    uint32_t slaveDataArrayIndex;

    // search for correct address and save the index
    for(slaveDataArrayIndex = 0U; slaveDataArrayIndex < slaveDataNumberOfElements;
        slaveDataArrayIndex += 1U)
    {
        if(slaveData[slaveDataArrayIndex].address == *address)
        {
            break; // exit for loop
        }
        // if the slave address was not found in slaveData handle error
        else if(slaveDataArrayIndex == (slaveDataNumberOfElements - 1U))
        {
            // error handler code
            XMC_DEBUG("\nSlave_not_found.\n");
            while(true);
        }
    }

    // check command retry count
    while(commandTryCounter < INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES_MAXIMUM_TRIES)
    {
        commandTryCounter += 1U; // increment number of command tries
        receivedDataBuffer = 0U; // reset received data
        packageCounter = 0U; // reset number of received data packages

        internalUART_receiveLast(); // clear UART input buffer by reading it until it is empty

        internalUART_outputAddress = *address;
        internalUART_outputData = INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES;
        internalUART_transmit();
        // start timer for response timeout
        oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, false);

        // receive data packages and update slave data array

```

```
while(packageCounter < SLAVE_DATA_NUMBER_OF_PACKAGES)
{
    // check and handle newly received command from the slave
    if(internalUART_receiveFirst()) // check for new received data
    {
        oneShotTimerElapsed(); // stop timer
        packageCounter += 1U; // increment data package count
        // save received data to local buffer
        receivedDataBuffer |= internalUART_inputData & INTERNAL_UART_MASK_DATA;

        /**
         * save data and clear receivedDataBuffer
         * or
         * shift receivedDataBuffer for the data from the next transmission
         */
        switch(packageCounter)
        {
            case (SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH /
INTERNAL_UART_NUMBER_OF_BITS_DATA):
                // save the rotary switch position
                slaveData[slaveDataArrayIndex].rotarySwitch =
                (uint8_t) (receivedDataBuffer & SLAVE_DATA_MASK_ROTARY_SWITCH);
                receivedDataBuffer = 0U; // clear buffer
                break;
            case ((SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1)
/ INTERNAL_UART_NUMBER_OF_BITS_DATA):
                // save the potentiometer 1 ADC value
                slaveData[slaveDataArrayIndex].potentiometer1ADC =
                (uint16_t) (receivedDataBuffer & SLAVE_DATA_MASK_POTENTIOMETER_1);
                receivedDataBuffer = 0U; // clear buffer
                break;
            case ((SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2)
/ INTERNAL_UART_NUMBER_OF_BITS_DATA):
                // save the potentiometer 2 ADC value
                slaveData[slaveDataArrayIndex].potentiometer2ADC =
                (uint16_t) (receivedDataBuffer & SLAVE_DATA_MASK_POTENTIOMETER_2);
                receivedDataBuffer = 0U; // clear buffer
                break;
            default:
                // shift bits to make room for the data from the next transmission
                receivedDataBuffer <<= INTERNAL_UART_NUMBER_OF_BITS_DATA;
                break;
        }

        // there are still packages left to receive
        if(packageCounter < SLAVE_DATA_NUMBER_OF_PACKAGES)
        {
            // start timer for response timeout of a following package
            oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, false);
        }
        // all packages received
        else
        {
            // wait for any follow-up packages to detect a faulty communication
            // restart timer for response timeout and wait for it to elapse
            oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, true);
            // if data is available, the communication had problems and the
            // received data is likely incorrect
            if(internalUART_receiveLast())
            {
                // increment data package count to indicate a incorrect amount of
                // received packages
                packageCounter += 1U;
                break; // exit inner while loop
            }
        }
    }
}
```

```
        {
            // exit outer while loop since all packages have been received
            // correctly and therefore no retry is required
            commandTryCounter =
                INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES_MAXIMUM_TRIES;
        }
    }
}
// check if INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US has elapsed since the
// last data package was received from the slave
else if(!oneShotTimer.running)
{
    // start timer for response timeout and wait for it to elapse
    oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, true);
    // The following line is only required if ...
    // ...at least one transmission has been received while waiting for the timeout,
    // ...no retry will follow and
    // ...it was the last slave to be contacted.
    internalUART_receiveLast(); // clear FIFO input buffer
    break; // exit inner while loop
}
}
}

// update connection state
// slave responded
if(packageCounter > 0U)
{
    // slave responded with the correct number of packages
    if(packageCounter == SLAVE_DATA_NUMBER_OF_PACKAGES)
    {
        // indicate no connection problems
        slaveData[slaveDataArrayIndex].connectionProblems = false;
    }
    // slave responded with a incorrect number of packages
    else
    {
        // indicate connection problems
        slaveData[slaveDataArrayIndex].connectionProblems = true;
    }

    // update connection state: existing/new connection
    if(!slaveData[slaveDataArrayIndex].connectionState)
    {
        // indicate connection
        slaveData[slaveDataArrayIndex].connectionState = true;
        // indicate new connection
        slaveData[slaveDataArrayIndex].connectionStateChanged = true;
    }
    else
    {
        // if slave was connected before, indicate no change
        slaveData[slaveDataArrayIndex].connectionStateChanged = false;
    }
}
// slave did not respond
else
{
    // indicate no connection problems
    slaveData[slaveDataArrayIndex].connectionProblems = false;

    // update connection state: no/lost connection
    if(slaveData[slaveDataArrayIndex].connectionState)
    {
        // indicate no connection
        slaveData[slaveDataArrayIndex].connectionState = false;
        // indicate lost connection
        slaveData[slaveDataArrayIndex].connectionStateChanged = true;
    }
}
```

```

        else
        {
            // if slave was not connected before, indicate no change
            slaveData[slaveDataArrayIndex].connectionStateChanged = false;
        }
    }
}

/**
 * update all slave information
 * check each slave and save the data sent by them unless aborted by the external device or
 * slave connection loss
 */
void slaveDataUpdateAll(void)
{
    for(uint8_t idx = 0U; idx < slaveDataNumberOfElements; idx += 1U)
    {
        // check and handle newly received command from the external device
        if(externalUART_receiveFirst() // check for new received data
        {
            // abort execution
            if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_ABORT)
            {
                externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
                externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_ABORT;
                externalUART_transmit();

                return; // exit function
            }
            // inform external device that command is being ignored
            else
            {
                externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
                externalUART_outputData = EXTERNAL_UART_MESSAGE_COMMAND_IGNORED;
                externalUART_transmit();
            }
        }
        else
        {
            slaveDataUpdate(&slaveData[idx].address);
        }
    }
}

/**
 * send all slave data as a string (table format)
 */
void sendSlaveDataString(bool onlyPrintIfSlaveAvailable)
{
    char uart_outputBuffer[128U] = {};
    // print header
    sprintf(uart_outputBuffer,
        "\nAddress|Connection|Rotary_Switch|Potentiometer|Potentiometer");
    externalUART_transmitString(uart_outputBuffer);
    sprintf(uart_outputBuffer,
        "\n|State|Position|1_ADC_Value|2_ADC_Value");
    externalUART_transmitString(uart_outputBuffer);
    sprintf(uart_outputBuffer,
        "\n-----|-----|-----|-----|-----");
    externalUART_transmitString(uart_outputBuffer);
    // print table content
    for(uint32_t idx = 0U; idx < slaveDataNumberOfElements; idx += 1U) {
        // check and handle newly received command from the external device
        if(internalUART_receiveFirst() // check for new received data
        {

```

```

// abort execution
if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_ABORT)
{
    externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
    externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_ABORT;
    externalUART_transmit();

    return; // exit function
}
// inform external device that command is being ignored
else
{
    externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
    externalUART_outputData = EXTERNAL_UART_MESSAGE_COMMAND_IGNORED;
    externalUART_transmit();
}
}
else if(!onlyPrintIfSlaveAvailable || slaveData[idx].connectionState)
{
    sprintf(uart_outputBuffer, "\n0b%u%u%u|_%s_%s|_%13hu|_%13hu|_%13hu",
        (slaveData[idx].address & 8U) ? (1U) : (0U),
        (slaveData[idx].address & 4U) ? (1U) : (0U),
        (slaveData[idx].address & 2U) ? (1U) : (0U),
        (slaveData[idx].address & 1U) ? (1U) : (0U),
        (slaveData[idx].connectionProblems) ? ("ERR_") : (""),
        (slaveData[idx].connectionState)
        ? ((slaveData[idx].connectionStateChanged)
        ? ("New") : ("C")) : ((slaveData[idx].connectionStateChanged)
        ? ("Lost") : ("NC")),
        (unsigned short) slaveData[idx].rotarySwitch,
        (unsigned short) slaveData[idx].potentiometer1ADC,
        (unsigned short) slaveData[idx].potentiometer2ADC);
    externalUART_transmitString(uart_outputBuffer);
}
}

/**
 * update LCD contents based on user input
 */
void lcdUpdate(void)
{
    // index of the slave in the slaveData array with the given address
    uint32_t slaveDataArrayIndex;
    uint32_t entryNumber = 0U; // entry number of the current row
    uint32_t rowNumber = 0U; // row number to write the row buffer to
    char rowBuffer[LCD_NUMBER_OF_COLUMNS] = {}; // array for characters of one row of the LCD
    // array of characters of one entry of a row of the LCD
    char entryBuffer[LCD_CHARACTERS_PER_ENTRY] = {};

    // fill LCD with content
    // 123456_123456_123456
    // 12345678901234567890

    // show system status and how many slaves were detected
    if(ui_rotarySwitchPosition == 0U)
    {
        uint32_t numberOfDetectedSlaves = 0U; // number of currently detected slaves
        // number of currently detected slaves that have communication problems
        uint32_t numberOfDetectedSlavesWithProblems = 0U;

        // print basic screen contents, if a different menu was show previously
        if(ui_rotarySwitchPosition != ui_rotarySwitchPositionPrevious)
        {
            // set row buffer contents
            sprintf(rowBuffer, "Status:");
            // display row buffer contents in row number on the LCD

```

```

        lcd_print(rowBuffer, 1U, 1U);
        // set row buffer contents
        sprintf(rowBuffer, "Detected_Cells:");
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 2U, 1U);
        // set row buffer contents
        sprintf(rowBuffer, "W_comm_prob:");
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 3U, 1U);
        // clear contents of unused row
        lcd_clearRow(4U);
    }

    // set entry buffer contents
    sprintf(rowBuffer, "Reading...");
    // display row buffer contents in row number on the LCD
    lcd_print(rowBuffer, 1U, 8U);

    // iterate over slave data array and count how many are connected
    for(slaveDataArrayIndex = 0U; slaveDataArrayIndex < slaveDataNumberOfElements;
        slaveDataArrayIndex += 1U)
    {
        if(slaveData[slaveDataArrayIndex].connectionState)
        {
            numberOfDetectedSlaves += 1U; // count detected slave
        }
        if(slaveData[slaveDataArrayIndex].connectionProblems)
        {
            // count detected slave with communication problems
            numberOfDetectedSlavesWithProblems += 1U;
        }
    }

    // set entry buffer contents
    sprintf(rowBuffer, "Ready");
    // display row buffer contents in row number on the LCD
    lcd_print(rowBuffer, 1U, 8U);
    // set entry buffer contents
    sprintf(rowBuffer, "%4lu", numberOfDetectedSlaves);
    // display row buffer contents in row number on the LCD
    lcd_print(rowBuffer, 2U, 17U);
    // set entry buffer contents
    sprintf(rowBuffer, "%4lu", numberOfDetectedSlavesWithProblems);
    // display row buffer contents in row number on the LCD
    lcd_print(rowBuffer, 3U, 17U);
}

// show connection state of all slaves (or address if button is pressed)
else if(ui_rotarySwitchPosition == 15U)
{
    // iterate over all possible slave addresses
    for(uint8_t searchAddress = INTERNAL_UART_ADDRESS_SLAVE_START;
        searchAddress <= INTERNAL_UART_ADDRESS_SLAVE_END; searchAddress += 1U)
    {
        // search for correct address and save the index
        for(slaveDataArrayIndex = 0U; slaveDataArrayIndex < slaveDataNumberOfElements;
            slaveDataArrayIndex += 1U)
        {
            if(slaveData[slaveDataArrayIndex].address == searchAddress)
            {
                break; // exit for loop
            }
            // if the slave address was not found in slaveData handle error
            else if(slaveDataArrayIndex == (slaveDataNumberOfElements - 1U))
            {
                // error handler code
                XMC_DEBUG("\nSlave_not_found.\n");
                while(true);
            }
        }
    }
}

```



```

}

if(ui_buttonState) // print address instead of the connection state
{
    // set entry buffer contents
    sprintf(entryBuffer, "%4hu", slaveData[slaveDataArrayIndex].address);
}
else // print connection state
{
    // set entry buffer contents
    sprintf(entryBuffer, "%s%s", (slaveData[slaveDataArrayIndex].connectionProblems)
? ("ERR") : ("..."), (slaveData[slaveDataArrayIndex].connectionState)
? ((slaveData[slaveDataArrayIndex].connectionStateChanged)
? ("_NW") : ("_C")) : ((slaveData[slaveDataArrayIndex].connectionStateChanged)
? ("LST") : ("_NC")));
}

// append entry buffer to row buffer
strncat(rowBuffer, entryBuffer, sizeof(entryBuffer));
// increment entry number
entryNumber += 1U;
// append one space to the row buffer
if(entryNumber < LCD_ENTRIES_PER_ROW)
{
    strncat(rowBuffer, "_", 1U);
}
else // display the row buffer on the LCD
{
    entryNumber = 0U; // reset entry number
    rowNumber += 1U; // increment row number
    lcd_clearRow(rowNumber); // clear row contents
    // display row buffer contents in row number on the LCD
    lcd_print(rowBuffer, rowNumber, 1U);
    // empty row buffer: iterate over characters of string and set each to
    // NUL character
    for(uint32_t stringIndex = 0U;
        stringIndex < sizeof(rowBuffer); stringIndex += 1U)
    {
        rowBuffer[stringIndex] = '\0';
    }
}
}

// show data of the selected slave
else if((ui_rotarySwitchPosition >= INTERNAL_UART_ADDRESS_SLAVE_START)
&& (ui_rotarySwitchPosition <= INTERNAL_UART_ADDRESS_SLAVE_END))
{
    // print basic screen contents, if a different menu was show previously
    if(ui_rotarySwitchPosition != ui_rotarySwitchPositionPrevious)
    {
        sprintf(rowBuffer, "Address:"); // set entry buffer contents
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 1U, 1U);
        sprintf(rowBuffer, "Connect:"); // set entry buffer contents
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 2U, 1U);
        sprintf(rowBuffer, "Rot.Sw:"); // set entry buffer contents
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 3U, 1U);
        sprintf(rowBuffer, "Pot. 1/2:"); // set entry buffer contents
        // display row buffer contents in row number on the LCD
        lcd_print(rowBuffer, 4U, 1U);
    }

    // search for correct address and save the index
    for(slaveDataArrayIndex = 0U; slaveDataArrayIndex < slaveDataNumberOfElements;
        slaveDataArrayIndex += 1U)
    {

```

```

        if(slaveData[slaveDataArrayIndex].address == ui_rotarySwitchPosition)
        {
            break; // exit for loop
        }
        // if the slave address was not found in slaveData handle error
        else if(slaveDataArrayIndex == (slaveDataNumberOfElements - 1U))
        {
            // error handler code
            XMC_DEBUG("\nSlave_not_found.\n");
            while(true);
        }
    }

    // update LCD
    // set entry buffer contents
    sprintf(rowBuffer, "%4hu", slaveData[slaveDataArrayIndex].address);
    lcd_print(rowBuffer, 1U, 17U); // display row buffer contents in row number on the LCD
    sprintf(rowBuffer, "%s%s", (slaveData[slaveDataArrayIndex].connectionProblems)
? ("_ERR_") : ("_C"), (slaveData[slaveDataArrayIndex].connectionState)
? ((slaveData[slaveDataArrayIndex].connectionStateChanged)
? ("_NW") : ("_C")) : ((slaveData[slaveDataArrayIndex].connectionStateChanged)
? ("LST") : ("_NC"))); // set entry buffer contents
    lcd_print(rowBuffer, 2U, 13U); // display row buffer contents in row number on the LCD
    // set entry buffer contents
    sprintf(rowBuffer, "%4hu", slaveData[slaveDataArrayIndex].rotarySwitch);
    lcd_print(rowBuffer, 3U, 17U); // display row buffer contents in row number on the LCD
    sprintf(rowBuffer, "%4hu/%4hu", slaveData[slaveDataArrayIndex].potentiometer1ADC,
slaveData[slaveDataArrayIndex].potentiometer2ADC); // set entry buffer contents
    lcd_print(rowBuffer, 4U, 12U); // display row buffer contents in row number on the LCD
}
else
{
    // print basic screen contents, if a different menu was show previously
    if(ui_rotarySwitchPosition != ui_rotarySwitchPositionPrevious)
    {
        // clear contents of LCD
        lcd_clearDisplay();
        // update LCD
        lcd_print("Undefined_function.", 1U, 1U);
    }
}
}

/**
 * @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.
 */
int main(void)
{
    // variables for initialization of DAVE APPs
    DAVE_STATUS_t dave_init_status;

    // initialize DAVE APPs
    dave_init_status = DAVE_Init();

    // check whether initialization of DAVE APPs was successful
    if(dave_init_status != DAVE_STATUS_SUCCESS)
    {
        // error handler code
        XMC_DEBUG("\nDAVE_APPS_initialization_failed\n");
        while(true);
    }
}

```

```

// indicate activity
DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

// create a software timer which generates a single callback event after
// TRANSCEIVER_STARTUP_TIME_IN_US
oneShotTimer.id = SYSTIMER_CreateTimer(TRANSCEIVER_STARTUP_TIME_IN_US,
SYSTIMER_MODE_ONE_SHOT, (void*)oneShotTimerElapsed, NULL);

// check whether software timer was created successfully
if(oneShotTimer.id == 0U) // error handler code
{
    XMC_DEBUG("\nTimer_creation_failed\n");
    while(true);
}

// create a software timer which generates a cyclic callback event after
SLAVE_DATA_UPDATE_TIME_IN_US
slavDataUpdatePeriodicTimerID = SYSTIMER_CreateTimer(SLAVE_DATA_UPDATE_TIME_IN_US,
SYSTIMER_MODE_PERIODIC, (void*)slavDataUpdatePeriodicTimerElapsed, NULL);

// check whether software timer was created successfully
if(slavDataUpdatePeriodicTimerID == 0U) // error handler code
{
    XMC_DEBUG("\nTimer_creation_failed\n");
    while(true);
}

// initialize LCD
lcd_init();

// update LCD
// not required after initialization lcd_clearDisplay();
lcd_print("Initializing...", 1U, 1U);
lcd_print("starting_transceiver", 2U, 1U);

// enable transceiver
DIGITAL_IO_SetOutputLow(&DO_TransceiverShutDown);

// start software timer
oneShotTimerStart(TRANSCEIVER_STARTUP_TIME_IN_US, false);

// initialize slave data array while waiting for transceiver to start
for(uint32_t idx = 0U, address = INTERNAL_UART_ADDRESS_SLAVE_START;
idx < slaveDataNumberOfElements; idx += 1U, address += 1U)
{
    slaveData[idx].address = (uint8_t) address;
    slaveData[idx].connectionState = false;
    slaveData[idx].connectionStateChanged = false;
    slaveData[idx].rotarySwitch = 0U;
    slaveData[idx].potentiometer1ADC = 0U;
    slaveData[idx].potentiometer2ADC = 0U;
}

// wait for one shot timer to elapse and transceiver is ready
while(oneShotTimer.running);

// clear UART receive FIFO
internalUART_receiveLast();
externalUART_receiveLast();

// communicate initialization complete and reading of slave data
externalUART_transmitString("\nInitialization_complete.\n");
externalUART_transmitString("\nReading_slave_data...");

```

```

lcd_clearDisplay ();
lcd_print("Initialization", 1U, 1U);
lcd_print("complete.", 2U, 1U);
lcd_print("Reading_slave_data..", 3U, 1U);

// look for available slaves and get their data and set
// slaveData[idx].connectionStateChanged = false; as this was the first execution
slaveDataUpdateAll();
for(uint32_t idx = 0; idx < slaveDataNumberOfElements; idx += 1) {
    slaveData[idx].connectionStateChanged = false;
}

// start software timer and check whether software timer start failed
// error handler code
if(SYSTIMER_StartTimer(slavDataUpdatePeriodicTimerID) != SYSTIMER_STATUS_SUCCESS)
{
    XMC_DEBUG("\nTimer_start_failed\n");
    while(true);
}

// send updated slave data to the external device if enabled
if(externalUART_autoUpdateSlaveData)
{
    sendSlaveDataString(false);
}

// update LCD
lcd_print("user_input..", 3U, 9U);

// update user input
ui_rotarySwitchPosition = (uint8_t)
    ( (!DIGITAL_IO_GetInput(&DI_RSW_8) << 3)
      | (!DIGITAL_IO_GetInput(&DI_RSW_4) << 2)
      | (!DIGITAL_IO_GetInput(&DI_RSW_2) << 1)
      | (!DIGITAL_IO_GetInput(&DI_RSW_1) ) );
// assign a different value to the previous rotary switch position than the current one,
// to force a complete update of the LCD contents
if(ui_rotarySwitchPosition == 0U)
{
    ui_rotarySwitchPositionPrevious = 1U;
}
else
{
    ui_rotarySwitchPositionPrevious = 0U;
}

// finally update LCD contents based on user input
// and signal that master is ready for commands
lcdUpdate();
externalUART_transmitString("done.\n");
externalUART_transmitString("\nReady.\n");

// stop indicating activity
DIGITAL_IO_SetOutputLow(&DO_LED_Y);

// cyclic application code
while (true)
{
    // check whether any slave sent something when it wasn't requested to ensure a free bus
    if(internalUART_receiveLast()) // check for new received data
    {
        // indicate activity
        DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

        /*

```

```

* This has the potential for an endless waiting loop if a slave keeps sending data,
* but the bus must be free as it is only half-duplex.
* Sending an abort message probably doesn't work as the bus is only half-duplex
* and the slave might receive corrupted data or nothing at all, since it ignores
* any data received while transmitting.
* Waiting for the communication cycle time could also cause an unresponsive
* user interface.
*/
// start timer for response timeout and wait for it to elapse
//oneShotTimerStart(INTERNAL_UART_COMMAND_RESPONSE_TIMEOUT_IN_US, true);

// stop indicating activity
DIGITAL_IO_SetOutputLow(&DO_LED_Y);
}
// check and handle newly received command from the external device
else if(externalUART_receiveFirst()) // check for new received data
{
    // indicate activity
    DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

    // check target address
    if(externalUART_inputAddress == EXTERNAL_UART_ADDRESS_MASTER)
    {
        // check received command
        // abort command
        if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_ABORT)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_ABORT;
            externalUART_transmit();
        }
        // ping command
        else if(externalUART_inputData == EXTERNAL_UART_COMMANDMESSAGE_PING)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = EXTERNAL_UART_COMMANDMESSAGE_PING;
            externalUART_transmit();
        }
        // show addresses of all available slaves
        else if(externalUART_inputData
        == EXTERNAL_UART_COMMAND_SHOW_ALL_AVAILABLE_SLAVE_ADDRESSES)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = externalUART_inputData;
            externalUART_transmit(); // echo command
            slaveDataUpdateAll();
            showAllAvailableSlaveAddresses();
        }
        // show data of all available slaves
        else if(externalUART_inputData
        == EXTERNAL_UART_COMMAND_SHOW_DATA_OF_ALL_AVAILABLE_SLAVES)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = externalUART_inputData;
            externalUART_transmit(); // echo command
            slaveDataUpdateAll();
            sendSlaveDataString(true);
        }
        // show data of all slaves
        else if(externalUART_inputData == EXTERNAL_UART_COMMAND_SHOW_DATA_OF_ALL_SLAVES)
        {
            externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
            externalUART_outputData = externalUART_inputData;
            externalUART_transmit(); // echo command
            slaveDataUpdateAll();
            sendSlaveDataString(false);
        }
        // toggle flag that indicates whether all slave data shall be send to the
        // external device after it was updated

```

```

else if(externalUART_inputData
== EXTERNAL_UART_COMMAND_TOGGLE_AUTO_UPDATE_SLAVE_DATA)
{
    externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
    externalUART_outputData = externalUART_inputData;
    externalUART_transmit(); // echo command
    externalUART_autoUpdateSlaveData = !externalUART_autoUpdateSlaveData;
}
// unknown command
else
{
    externalUART_outputAddress = EXTERNAL_UART_ADDRESS_EXTERNAL;
    externalUART_outputData = EXTERNAL_UART_MESSAGE_UNKNOWN_COMMAND;
    externalUART_transmit();
}
}
// since the master was not addressed, forward message to internal UART and
// return answer of slave or timeout message
else
{
    forwardCommandAndWaitForAnswer();
}

// stop indicating activity
DIGITAL_IO_SetOutputLow(&DO_LED_Y);
}
// if no new data has been received, check whether the user input changed or the timer
// for update interval has elapsed and update the external device (if enabled) and the LCD
else
{
    // indicate activity
    DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

    // update user inputs
    ui_buttonState = !DIGITAL_IO_GetInput(&DI_BTN);
    ui_rotarySwitchPosition = (uint8_t)
        ( (!DIGITAL_IO_GetInput(&DI_RSW_8) << 3)
          | (!DIGITAL_IO_GetInput(&DI_RSW_4) << 2)
          | (!DIGITAL_IO_GetInput(&DI_RSW_2) << 1)
          | (!DIGITAL_IO_GetInput(&DI_RSW_1)) );

    // check whether slave data needs to be updated or user inputs have changed
    // and update the LCD contents and the external device if enabled
    if(slaveDataUpdateFlag
    || (ui_rotarySwitchPositionPrevious != ui_rotarySwitchPosition)
    || (ui_buttonStatePrevious != ui_buttonState))
    {
        // if timer for update interval has elapsed update slave data
        if(slaveDataUpdateFlag)
        {
            slaveDataUpdateAll(); // get current data from all slaves

            slaveDataUpdateFlag = false; // reset flag

            // send updated slave data to the external device if enabled
            if(externalUART_autoUpdateSlaveData)
            {
                sendSlaveDataString(false);
            }
        }

        // finally update LCD contents based on user input
        lcdUpdate();

        // set previous input states to detect changes
        ui_buttonStatePrevious = ui_buttonState;
        ui_rotarySwitchPositionPrevious = ui_rotarySwitchPosition;
    }
}

```

```
        // stop indicating activity
        DIGITAL_IO_SetOutputLow(&DO_LED_Y);
    }
}
}
```

A.2.2 main.c des Messmoduls als Slave

```

/*
TODO Setup for IrDA communication.
After every code generation modify the generated code as follows:

In "Dave\Generated\UART\uart_conf.c"
in function "UART_STATUS_t UART_Internal_init()"
add after "XMC_UART_CH_Init(XMC_USIC_CH_t *channel, const XMC_UART_CH_CONFIG_t *const config);"
and before "XMC_UART_CH_Start(XMC_USIC_CH_t *const channel);"
the following code block:

// manually inserted code for IrDA transceiver
// set sample mode to one sample per bit and the sample point
// to 2/16 of the bit length (after time quanta 1)
UART_Internal.channel->PCR_ASCMode = (uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_SMD_Msk)
& (~USIC_CH_PCR_ASCMode_SP_Msk)
| (((uint32_t) 0UL) << USIC_CH_PCR_ASCMode_SMD_Pos)
| (((uint32_t) 1UL) << USIC_CH_PCR_ASCMode_SP_Pos));
// enable pulse output and set pulse length to 4/16 of the bit length
UART_Internal.channel->PCR_ASCMode = (uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_PL_Msk)
| (((uint32_t) 3UL) << USIC_CH_PCR_ASCMode_PL_Pos));
// invert data output
UART_Internal.channel->SCTR = (uint32_t) (UART_Internal.channel->SCTR
& (~USIC_CH_SCTR_DOCFG_Msk)
| (uint32_t) XMC_USIC_CH_DATA_OUTPUT_MODE_INVERTED;

Alternative configurations and options:

// set sample mode to majority decision and the sample point to 4/16 of the bit length
// (after time quanta 3)
//UART_Internal.channel->PCR_ASCMode = (uint32_t) ((UART_Internal.channel->PCR_ASCMode
& (~USIC_CH_PCR_ASCMode_SMD_Msk)
& (~USIC_CH_PCR_ASCMode_SP_Msk)
| (((uint32_t) 1UL) << USIC_CH_PCR_ASCMode_SMD_Pos)
| (((uint32_t) 3UL) << USIC_CH_PCR_ASCMode_SP_Pos));

// invert data output
//XMC_USIC_CH_SetDataOutputMode(UART_Internal.channel, XMC_USIC_CH_DATA_OUTPUT_MODE_INVERTED);
*/

#include <DAVE.h> //Declarations from DAVE Code Generation (includes SFR declaration)

// 0,5 ms
// time the transceiver needs after the SD (shutdown) signal goes low before it can operate
#define TRANSCEIVER_STARTUP_TIME_IN_US (500U)

#define INTERNAL_UART_NUMBER_OF_BITS_ADDRESS (4U) // number of address bits; must be 2 ^ n
#define INTERNAL_UART_NUMBER_OF_BITS_DATA (4U) // number of data bits; must be 2 ^ m

// (2 ^ INTERNAL_UART_NUMBER_OF_BITS_ADDRESS - 1U) << INTERNAL_UART_NUMBER_OF_BITS_DATA
// mask for address
#define INTERNAL_UART_MASK_ADDRESS (240U)
// 2 ^ INTERNAL_UART_NUMBER_OF_BITS_DATA - 1U
// mask for data
#define INTERNAL_UART_MASK_DATA (15U)

#define INTERNAL_UART_ADDRESS_MASTER (0U) // address of master transceiver

```


A Anhänge

```
#define INTERNAL_UART_ADDRESS_SLAVE_START (1U) // lowest address of first cell slave
#define INTERNAL_UART_ADDRESS_SLAVE_END (12U) // highest address of last cell slave
// 2 ^ INTERNAL_UART_NUMBER_OF_BITS_ADDRESS - 1U
// broadcast address to reach all bus participants
#define INTERNAL_UART_ADDRESS_BROADCAST (15U)

// target of last command doesn't know the command
#define INTERNAL_UART_MESSAGE_UNKNOWN_COMMAND (1U)
// command for requesting a response and response message from the target
#define INTERNAL_UART_COMMANDMESSAGE_PING (2U)
// command for requesting abortion of current action and response message from the target
// if action was aborted
#define INTERNAL_UART_COMMANDMESSAGE_ABORT (3U)
// command for sending all available data
#define INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES (4U)
//...
// #define COMMAND_ (15U) // command for

// transmission time of one character = (1 + 8 + 1 + 1) / (115200 1/s) = 95,486 us
// 10 ms; time window the slave has to respond to any command
#define INTERNAL_UART_COMMAND_RESPONSE_TIME_WINDOW_IN_US (10000U)

// number of bits sent by the slave for the rotary switch position
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH (4U)
// number of bits sent by the slave for the ADC value of potentiometer 1 (or supply voltage)
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1 (12U)
// number of bits sent by the slave for the ADC value of potentiometer 2 (or temperature)
// (Should be multiple of INTERNAL_UART_NUMBER_OF_BITS_DATA.)
#define SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2 (12U)

// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH - 1U
// mask for rotary switch position bits to be send to the master
#define SLAVE_DATA_MASK_ROTARY_SWITCH ((uint32_t) 255)
// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1 - 1U
// mask for ADC value of potentiometer 1 (or supply voltage) bits to be send to the master
#define SLAVE_DATA_MASK_POTENTIOMETER_1 ((uint32_t) 4095)
// 2 ^ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2 - 1U
// mask for ADC value of potentiometer 2 (or temperature) bits to be send to the master
#define SLAVE_DATA_MASK_POTENTIOMETER_2 ((uint32_t) 4095)

// number of transmissions from the slave after COMMAND_SEND_ALL_DATA_PACKAGES
#define SLAVE_DATA_NUMBER_OF_PACKAGES ((SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2
+ SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1)
/ INTERNAL_UART_NUMBER_OF_BITS_DATA)

// 2 s; time between slave data update (time before flag for update request is set to true again)
#define SLAVE_DATA_UPDATE_TIME_IN_US (2000000U)

#define NUMBER_OF_ADC_CHANNELS (2U) // number of ADC channels used

// array for ADC values
// index = 0: potentiometer 1 (or supply voltage); index = 1: potentiometer 2 (or temperature)
XMC_VADC_RESULT_SIZE_t measuredADCvalues[NUMBER_OF_ADC_CHANNELS];
uint8_t adcChannelIndex = 0U; // index for saving the measured ADC values to an array
volatile bool adcMeasuring = false; // flag to indicate that ADC measurement is in progress

struct struct_slave {
    uint8_t address; // address of slave
    uint8_t rotarySwitch; // position of rotary switch
    // ADC value of potentiometer 1; this analog input can also be used for supply voltage measurement
    uint16_t potentiometer1ADC;
    // ADC value of potentiometer 2; this analog input can also be used for temperature measurement
    uint16_t potentiometer2ADC;
```

A Anhänge

```
};

struct struct_slave slaveData = {}; // struct for all slave data

// only read and write in UART receive event handler; buffer for received data
volatile uint8_t internalUART_inputBuffer = 0U;
// only read and write in UART transmission function; buffer for transmission data
uint8_t internalUART_outputBuffer = 0U;

uint8_t internalUART_inputAddress = 0U; // source address were the received data came from
uint8_t internalUART_inputData = 0U; // data received from source

uint8_t internalUART_outputAddress = 0U; // destination address were data will be send to
uint8_t internalUART_outputData = 0U; // data to send to destination

uint32_t oneShotTimerID = 0U; // timer ID for controlling one the timer
volatile bool oneShotTimerRunning = false; // true while one shot timer is running

/**
 * restart software timer and check whether software timer restart failed
 */
void oneShotTimerStart(uint32_t timeToWait, bool waitForTimerToElapse)
{
    if(SYSTIMER_RestartTimer(oneShotTimerID, timeToWait) == SYSTIMER_STATUS_SUCCESS)
    {
        oneShotTimerRunning = true;
    }
    else // error handler code
    {
        XMC_DEBUG("\nTimer_start_failed\n");
        while(true);
    }

    // wait for one shot timer to elapse
    if(waitForTimerToElapse)
    {
        while(oneShotTimerRunning);
    }
}

/**
 * callback function of one shot timer
 * stop timer and indicate that timer has elapsed
 */
void oneShotTimerElapsed(void)
{
    // stop software timer
    SYSTIMER_StopTimer(oneShotTimerID);

    oneShotTimerRunning = false;
}

/**
 * If the UART input buffer is not empty, read the first (oldest) data from the
 * UART input buffer to extract and save the address and first (oldest) data.
 * returns true if data was available
 */
bool internalUART_receiveFirst()
{
    bool dataAvailable = false;

    // read oldest data from receive FIFO and thereby shift all receive FIFO data
    if(!UART_IsRXFIFOEmpty(&UART_Internal))
```

```

    {
        internalUART_inputBuffer = UART_GetReceivedWord(&UART_Internal);
        internalUART_inputAddress = (uint8_t) ((internalUART_inputBuffer
        & INTERNAL_UART_MASK_ADDRESS) >> INTERNAL_UART_NUMBER_OF_BITS_DATA);
        internalUART_inputData = (uint8_t) (internalUART_inputBuffer & INTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate that no data has been received
}

/**
 * If the UART input buffer is not empty, read the first (oldest) data from the
 * UART input buffer until it is empty to save the last (newest) data.
 * This discards all older received data in the UART input buffer.
 * returns true if data was available
 */
bool internalUART_receiveLast()
{
    bool dataAvailable = false;

    while(UART_IsRxBusy(&UART_Internal)); // wait for any running reception to finish

    // read oldest data from receive FIFO and thereby shift all receive FIFO data until
    // the receive FIFO is empty to get the latest received data
    while(!UART_IsRXFIFOEmpty(&UART_Internal))
    {
        internalUART_inputBuffer = UART_GetReceivedWord(&UART_Internal);
        internalUART_inputAddress = (uint8_t) ((internalUART_inputBuffer
        & INTERNAL_UART_MASK_ADDRESS) >> INTERNAL_UART_NUMBER_OF_BITS_DATA);
        internalUART_inputData = (uint8_t) (internalUART_inputBuffer & INTERNAL_UART_MASK_DATA);
        dataAvailable = true; // indicate that data has been received
    }

    return dataAvailable; // indicate whether data has been received
}

/**
 * transmit one symbol via UART
 * Any received data which was received while sending data is discarded.
 */
void internalUART_transmit(void)
{
    /*
     * The while loops "while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal))"
     * before and after the function
     * "UART_Transmit(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * shouldn't be necessary when receive mode
     * is set to "direct" which blocks the CPU until all data is sent or received.
     * (See description of function
     * "UART_lStartTransmitPolling(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * and
     * "UART_lStartReceivePolling(const UART_t * const handle, uint8_t * data_ptr, uint32_t count)"
     * in "Dave\Generated\UART\uart.c".)
     */
    internalUART_outputBuffer = (internalUART_outputAddress << INTERNAL_UART_NUMBER_OF_BITS_DATA)
    | internalUART_outputData; // set output buffer
    // wait for previous transmission or reception to finish
    while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal));
    // transmit contents of output buffer
    UART_Transmit(&UART_Internal, &internalUART_outputBuffer, 1U);
    // wait for transmission or reception to finish
    while(UART_IsTxBusy(&UART_Internal) || UART_IsRxBusy(&UART_Internal));
    // wait for echoed data from the IrDA transmitter and discard it
    while(!internalUART_receiveLast());
}

```

```
/**
 * get rotary switch position
 */
void updateRotarySwitchPosition()
{
    slaveData.rotarySwitch = (uint8_t)
        ( (!DIGITAL_IO_GetInput(&DI_RSW_8) << 3)
          | (!DIGITAL_IO_GetInput(&DI_RSW_4) << 2)
          | (!DIGITAL_IO_GetInput(&DI_RSW_2) << 1)
          | (!DIGITAL_IO_GetInput(&DI_RSW_1) ) );
}

/**
 * callback function after each finished measurement to save the measured ADC value
 */
void ADC_Measurement_Handler(void)
{
    measuredADCvalues[adcChannelIndex] = ADC_MEASUREMENT_GetGlobalResult();
    adcChannelIndex += 1; // increment index
    // when the last ADC measurement completed, indicate so and reset index
    if(adcChannelIndex == NUMBER_OF_ADC_CHANNELS)
    {
        adcChannelIndex = 0; // reset index
        adcMeasuring = false; // reset flag
    }
}

/**
 * measure potentiometer values
 */
void updatePotentiometerADCvalues()
{
    // measure the analog values
    adcMeasuring = true; // set flag
    // start measurement for all used ADC channels
    ADC_MEASUREMENT_StartConversion(&ADC_MEASUREMENT_0);
    while(adcMeasuring); // wait for measurement to finish

    // save the measured values
    slaveData.potentiometer1ADC = measuredADCvalues[0]; // save the measured value
    slaveData.potentiometer2ADC = measuredADCvalues[1]; // save the measured value
}

/**
 * no function implemented
 */
void InterruptHandler_BTN(void)
{
    // insert code here
}

/**
 * sends (in order) rotary switch position, potentiometer 2 ADC value and
 * potentiometer 1 ADC value to the master
 * transmission order is MSB to LSB in package sizes of INTERNAL_UART_NUMBER_OF_BITS_DATA
 */
void sendAllData()
{
    uint32_t valueBuffer = 0U; // buffer for currently selected value
    uint32_t numberOfPackages = 0U; // number of transmissions for currently selected value

    // set master address as destination address
    internalUART_outputAddress = INTERNAL_UART_ADDRESS_MASTER;
}
```

```

for(uint8_t valueCounter = 0U; valueCounter < 3U; valueCounter += 1U)
{
    switch(valueCounter)
    {
        case 0U: // rotary switch position
            // calculate the number of data packages to transmit the rotary switch position
            numberOfPackages = SLAVE_DATA_NUMBER_OF_BITS_ROTARY_SWITCH
            // INTERNAL_UART_NUMBER_OF_BITS_DATA;
            valueBuffer = slaveData.rotarySwitch;
            break; // exit switch case
        case 1U: // potentiometer 1 ADC value
            // calculate the number of data packages to transmit the potentiometer 1 ADC value
            numberOfPackages = SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_1
            // INTERNAL_UART_NUMBER_OF_BITS_DATA;
            valueBuffer = slaveData.potentiometer1ADC;
            break; // exit switch case
        case 2U: // potentiometer 2 ADC value
            // calculate the number of data packages to transmit the potentiometer 1 ADC value
            numberOfPackages = SLAVE_DATA_NUMBER_OF_BITS_POTENTIOMETER_2
            // INTERNAL_UART_NUMBER_OF_BITS_DATA;
            valueBuffer = slaveData.potentiometer2ADC;
            break; // exit switch case
        default: // valueCounter out of range
            // error handler code
            XMC_DEBUG("\nvalueCounter_out_of_range.\n");
            while(true);
            //return; // exit function
    }

    // send data packages and check for new commands from the master
    for(uint32_t packageNumber = 1U; packageNumber <= numberOfPackages; packageNumber += 1U)
    {
        // check and handle newly received command from the master
        // Receiving commands from the master is only possible in the time
        // after the slave cleared its input buffer from the echoed data
        // and before the slave checks the input buffer again.
        // Therefore the usefulness of this check is questionable.
        if(internalUART_receiveLast()) // check for new received data
        {
            // abort execution
            if(internalUART_inputData == INTERNAL_UART_COMMANDMESSAGE_ABORT)
            {
                // set data
                internalUART_outputData = INTERNAL_UART_COMMANDMESSAGE_ABORT;
                internalUART_transmit(); // transmit data
                return; // exit function
            }
        }
        // send data package
        else
        {
            internalUART_outputData = (uint8_t) ((valueBuffer >>
            ((numberOfPackages - packageNumber) * INTERNAL_UART_NUMBER_OF_BITS_DATA))
            & INTERNAL_UART_MASK_DATA); // get all the bits that fit into one transmission
            internalUART_transmit(); // transmit data
        }
    }
}

/**
 * @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.

```

```

*/
int main(void)
{
    // variables for initialization of DAVE APPs
    DAVE_STATUS_t dave_init_status;

    // initialize DAVE APPs
    dave_init_status = DAVE_Init();

    // check whether initialization of DAVE APPs was successful
    if(dave_init_status != DAVE_STATUS_SUCCESS)
    {
        // error handler code
        XMC_DEBUG("\nDAVE_APPS_initialization_failed\n");
        while(true);
    }

    // indicate activity
    DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

    // create a software timer which generates a single callback event
    // after TRANSCEIVER_STARTUP_TIME_IN_US
    oneShotTimerID = SYSTIMER_CreateTimer(TRANSCEIVER_STARTUP_TIME_IN_US,
    SYSTIMER_MODE_ONE_SHOT, (void*)oneShotTimerElapsed, NULL);

    // check whether software timer was created successfully
    if(oneShotTimerID == 0U) // error handler code
    {
        XMC_DEBUG("\nTimer_creation_failed\n");
        while(true);
    }

    // set address to rotary switch position
    updateRotarySwitchPosition(); // update rotary switch position
    slaveData.address = slaveData.rotarySwitch; // set slave address

    /**
     * check whether slave address is in the valid range
     * if not indicate invalid address by flashing both red LEDs alternately
     * the user must set a valid address using the rotary switch and confirm it by pressing the button
     */
    DIGITAL_IO_SetOutputHigh(&DO_LED1); // turn on LED1
    DIGITAL_IO_SetOutputLow(&DO_LED2); // turn off LED2
    oneShotTimerStart(250000, false); // start oneShotTimer with 0,25 seconds
    while(!(INTERNAL_UART_ADDRESS_SLAVE_START <= slaveData.address
    && slaveData.address <= INTERNAL_UART_ADDRESS_SLAVE_END))
    {
        if(!PIN_INTERRUPT_GetPinValue(&DI_INT_BTN))
        {
            updateRotarySwitchPosition(); // update rotary switch position
            slaveData.address = slaveData.rotarySwitch; // set address to rotary switch position
        }
        else if(!oneShotTimerRunning)
        {
            DIGITAL_IO_ToggleOutput(&DO_LED1); // toggle LED1
            DIGITAL_IO_ToggleOutput(&DO_LED2); // toggle LED2
            oneShotTimerStart(250000, false); // start oneShotTimer with 0,25 seconds
        }
    }
    DIGITAL_IO_SetOutputLow(&DO_LED1); // turn off LED1
    DIGITAL_IO_SetOutputLow(&DO_LED2); // turn off LED2
    oneShotTimerElapsed(); // stop oneShotTimer timer
    // wait until button is released to avoid unintended input
    while(!PIN_INTERRUPT_GetPinValue(&DI_INT_BTN));

    // enable transceiver

```

```

DIGITAL_IO_SetOutputLow(&DO_TRX_SD);

// start software timer for transceiver start up time
oneShotTimerStart(TRANSCEIVER_STARTUP_TIME_IN_US, false);

// update slave data while waiting for transceiver to start
updateRotarySwitchPosition(); // update rotary switch position
updatePotentiometerADCvalues(); // update rotary switch position

// wait for one shot timer to elapse and transceiver is ready
while(oneShotTimerRunning);

// clear UART receive FIFO
internalUART_receiveLast();

// stop indicating activity
DIGITAL_IO_SetOutputLow(&DO_LED_Y);

// cyclic application code
while (true)
{
    if(internalUART_receiveLast()) // check for new received data
    {
        // indicate that the slave is handling the received data
        DIGITAL_IO_SetOutputHigh(&DO_LED_Y);

        // check target address
        // own address
        if(internalUART_inputAddress == slaveData.address)
        {
            // indicate that this slave was addressed
            DIGITAL_IO_SetOutputHigh(&DO_LED1);
            DIGITAL_IO_SetOutputHigh(&DO_LED2);

            // check and react to input buffer
            // abort command
            if(internalUART_inputData == INTERNAL_UART_COMMANDMESSAGE_ABORT)
            {
                internalUART_outputAddress = INTERNAL_UART_ADDRESS_MASTER;
                internalUART_outputData = INTERNAL_UART_COMMANDMESSAGE_ABORT;
                internalUART_transmit();
            }
            // ping command
            else if(internalUART_inputData == INTERNAL_UART_COMMANDMESSAGE_PING)
            {
                internalUART_outputAddress = INTERNAL_UART_ADDRESS_MASTER;
                internalUART_outputData = INTERNAL_UART_COMMANDMESSAGE_PING;
                internalUART_transmit();
            }
            // update all data and send all data packages
            else if(internalUART_inputData == INTERNAL_UART_COMMAND_SEND_ALL_DATA_PACKAGES)
            {
                updatePotentiometerADCvalues();
                updateRotarySwitchPosition();
                sendAllData();
            }
            // unknown command
            else
            {
                internalUART_outputAddress = INTERNAL_UART_ADDRESS_MASTER;
                internalUART_outputData = INTERNAL_UART_MESSAGE_UNKNOWN_COMMAND;
                internalUART_transmit();
            }
        }


        // stop indicating that this slave was addressed
        DIGITAL_IO_SetOutputLow(&DO_LED1);
        DIGITAL_IO_SetOutputLow(&DO_LED2);
    }
}

```

```
    }  
    // broadcast address  
    else if(internalUART_inputAddress == INTERNAL_UART_ADDRESS_BROADCAST)  
    {  
        // indicate that this slave was addressed  
        DIGITAL_IO_SetOutputHigh(&DO_LED1);  
        DIGITAL_IO_SetOutputHigh(&DO_LED2);  
  
        // insert code here  
  
        // stop indicating that this slave was addressed  
        DIGITAL_IO_SetOutputLow(&DO_LED1);  
        DIGITAL_IO_SetOutputLow(&DO_LED2);  
    }  
  
    // stop indicating that the slave is handling the received data  
    DIGITAL_IO_SetOutputLow(&DO_LED_Y);  
} }  
}
```


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