HAW HAMBURG

BACHELORARBEIT

Codierte Datenübertragung per Licht und Erprobung im Rahmen einer drahtlosen Anwendung für Außenbereichsleuchten

vorgelegt am 13. April 2023

von Lennard Jönsson

1. Prüfer: Prof. Dr. Jan Mietzner 2. Prüfer: Reinhard Breuer

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG

Fachbereich Medientechnik

Finkenau 35 22081 Hamburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungen

API Application Programming Interface

CIM Color Intensity Modulation

CPU Central Processing Unit

CRI Color Rendering Index

CSK Color-shift Keying

FIFO First In, First Out

FSK Frequency Shift Keying

GPIO General Purpose Input/Output, oftmals einfach: Pin

HF Hochfrequenz

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

ISR Interrupt Service Routine

LED Light Emitting Diode

MOSFET Metalloxid-Halbleiter-Feldeffekttransistor

OOK On-Off-Keying

PAM Pulsamplitudenmodulation

PDM Pulsdauermodulation

PPM Pulspositionmodulation

PSK Frequency Shift Keying

PWM Pulsweitenmodulation

RAM Random Access Memory

ROM Read-Only Memory

RTOS Real-Time Operating System

RX empfangsseitig

SoC System on a chip

TX sendeseitg

UART Universal Asynchronous Receiver/Transmitter

VLC Visible Light Communications

Zusammenfassung

Diese Arbeit behandelt die Anwendung von Datenübertragung mittels sichtbaren Lichts in Kombination mit Mikrocontrollern. Es werden zwei Programmieransätze ausgeführt und als Programmbibliotheken für Folgearbeiten zur Verfügung gestellt. Eine Programmbibliothek benutzt *bit banging*, um die Input/Output-Pins des Mikrocontrollers direkt anzusteuern und eine Lichtquelle zu schalten beziehungsweise einen Lichtsensor auszulesen. Der zweite Ansatz verwendet die seriellen Schnittstellen des Mikrocontrollers für ebendiese Aufgaben. Weiterhin werden elektronische Schaltungen vorgestellt, die sich zum Senden und Empfangen der hochfrequenten Signale eignen. Abschließend erfolgen Untersuchungen mit Leuchtmitteln für Umgebungslicht, Auswertungen der Datenrate und der Reichweite.

Summary

This thesis approaches the application of data transmission by means of visible light in combination with microcontrollers. Two programming approaches are executed and made available as program libraries for potential follow-up work. One of the libraries uses a bit banging approach to directly control the input / output pins of the microcontroller and switch a light source or read a light sensor. The second approach uses the serial interfaces of the microcontroller for the same tasks. Furthermore, electronic circuits are presented that are suitable for transmitting and receiving the high-frequency signals. Finally, investigations are carried out with illuminants for ambient light, evaluations of the data rate and the range.

Kapitel 1

Einleitung

Die hochfrequente Datenübertragung mit sichtbarem Licht (oft abgekürzt mit VLC, *visible light communications*) stellt ein intensiv untersuchtes Forschungsthema dar, und wird zuweilen auch bereits in kommerziell erhältlichen Produkten als Alternative zu Funkwellen angewandt.[1, 2, 3, 4],[5, 6] Die Verwendung des breitbandigen Lichtspektrums im sichtbaren Bereich kann für hohe Datenraten genutzt werden und wird bereits in sogenannten „LiFi"-Produkten als Alternative zu dem WiFi-Standard eingesetzt. Neben der hohen Datenrate ergeben sich weitere Vorteile durch eine scharfe räumliche Begrenzung des Lichtkegels, die zu einer hohen Abhörsicherheit und der Vermeidung von Interferenzen mit nahegelegenen Systemen führt. Zudem können bereits vorhandene Lichtinstallationen mit verhältnismäßig geringem Aufwand aufgerüstet werden.

Die Modulation und die Codierung der Übertragungsstrecke sind entscheidende Faktoren für eine effiziente Datenübertragung. Durch die hohen Frequenzen im Terahertzbereich wären theoretisch sehr hohe Datenraten weit über 100 GBit/s möglich. Die Datenrate wird jedoch durch die Trägheit der elektronischen Bauteile, insbesondere durch das Leuchtmittel begrenzt. Gleichzeitig entstehen zusätzliche Anforderungen an den Sender, da eine gute Lichtqualität gewünscht und eine gesundheitliche Verträglichkeit vorausgesetzt wird. Der kluge Einsatz von Modu-

lations- und Codierverfahren berücksichtigt zum einen die Maximierung der Datenrate und zum anderen die Lichtqualität.

Ein simpler Aufbau einer VLC-Übertragung kann bereits mit zwei Mikrocontrollern, einer LED und einer Photodiode erfolgen. Dadurch ist ein schneller Einstieg in das Thema gegeben, wodurch die Hemmschwelle für die Lehre an Schulen und Universitäten niedrig gesetzt wird. Eine fortgeschrittene Nutzung kann durch die Nutzung der Schnittstellen des Mikrocontrollers und hardwarenahe Programmierung erfolgen. Darauf aufbauend können Codierverfahren implementiert und weiterführende Projekte, etwa die Nutzung in bestimmten Beleuchtungsszenarien, angesetzt werden. In dieser Arbeit sollen zwei Programmieransätze vorgestellt werden, die eine Maximierung der Datenrate mit dem weit verbreiteten ESP32 Mikrocontroller anstreben. Die Programmierung erfolgt in dem Arduino-Framework, das populär im Lehrbetrieb und in Anwendungen eingesetzt wird. Durch eine detaillierte Dokumentation und eine ausführliche Erklärung der angewandten Methoden soll ein leichter Einstieg für potentielle zukünftige Arbeiten ermöglicht werden. Zuletzt soll ein Laboraufbau mit LEDs erfolgen, die zur Umgebungsbeleuchtung verwendet werden können. Insbesondere im Kontext für Außenbereichsleuchten wird eine hohe Reichweite angestrebt, um die Kommunikation zweier benachbarter Leuchten zu ermöglichen. Vor diesem Hintergrund sollen Untersuchungen zur Reichweitenverbesserung der Übertragung erfolgen.

Kapitel 2

Einführungen in die Themen

Diese Arbeit verbindet eine Reihe an Themengebieten, die in diesem Kapitel eingeführt und wichtige Grundlagen erklärt werden sollen. Zunächst folgt eine Einführung in die Themen LiFi und VLC. Daran anschließend werden die Themen Modulation und Leitungscodierung behandelt. In Abschnitt 2.4 wird ein Überblick über die verwendeten elektronischen Bauteile gegeben. Die darauf folgenden Abschnitte befassen sich mit der Programmierung von Mikrocontrollern und insbesondere mit dem Arduino-Framework und dem ESP32.

2.1 LiFi und Datenübertragung mit sichtbarem Licht

Datenübertragung mit sichtbarem Licht – oder VLC - *visible light commnications* – bezeichnet die Nutzung der elektromagnetischen Strahlung in dem sichtbaren Bereich der Wellenlängen von etwa 400 nm bis 700 nm. In einem engeren Sinne beschreibt der Term die Punkt-zu-Punkt Verbindung zweier Geräte. Dagegen bezeichnet der Ausdruck „LiFi“ die Datenübertragung mit Datenraten im Megabit und Gigabitbereich im sichtbaren elektromagnetischen Spektrum und in den angrenzenden Gebieten der ultravioletten und der infraroten Strahlung. LiFi soll angelehnt an den WiFi-Standard als drahtloses Netzwerksystem mit Multibenutzer-

kommunikation entwickelt werden und ihn teilweise ergänzen oder ersetzen.[2]

Ermöglicht wurde diese Entwicklung durch die Erfindung und Weiterentwicklung von Leuchtdioden und Photodioden (siehe Abschnitte 2.4.1 und 2.4.2), die anstelle der für die im Bereich der Mikrowellenstrahlung verwendeten Antennen genutzt werden. Photodioden setzen jedoch anders als Antennen nicht die empfangene Strahlung in entsprechende Trägersignale um - diese hätten im Bereich des sichtbaren Lichts Frequenzen zwischen 430 THz und 750 THz - sondern generieren eine Spannung aufgrund der Strahlungsintensität auf dem Halbleitermaterial (siehe auch hierzu Abschnitt 2.4.2). Die Informationsübertragung findet demnach als Reaktion auf die Modulation der Strahlungsintensität statt. Spezielle Photodioden für hohe Geschwindigkeiten reagieren auf Änderungen mit Reaktionszeiten in der Größenordnung von wenigen Nanosekunden, wodurch Grenzfrequenzen im Gigahertzbereich möglich wären. Leuchtdioden sind bei der Erzeugung ihrer Leuchtkraft bei einem sprunghaften Spannungsanstieg am Eingang jedoch langsamer: Sie haben Grenzfrequenzen im hohen Megahertzbereich. Aus diesem Grund ist der kluge Einsatz von Modulationsverfahren ausschlaggebend für die Verbesserung der Datenrate bei VLC. Auf das Thema Modulation wird in Abschnitt 2.2 weiter eingegangen.[7]

2.1.1 Chancen und Potentiale

Die Verwendung sichtbaren Lichts bringt einige Vorzüge mit sich, die VLC und LiFi einzigartig im Feld der Datenübertragung machen. Das diskutierte hochfrequente Spektrum ist breitbandig, bisher unlizenziert und damit frei benutzbar. Die hohen Frequenzen im Terahertzbereich bieten Möglichkeiten für sehr hohe theoretische Datenraten, die bisher nur durch die Schnelligkeit der Dioden begrenzt werden. Eine mögliche Weiterentwicklung der Halbleiterbauteile könnte zukünftig unmittelbar zu einer Erhöhung der Datenrate führen. In diesem Zusammenhang wur-

den Forschungsansätze mit extrem kleinflächigen Leuchtdioden im Mikrometer- und Nanometermaßstab oder Leuchtdioden mit optischem Resonator vorgestellt, um eine schnellere Reaktionszeit zu erhalten.[8, 3, 1]

Die Lizenzierung der Radio- und Mikrowellenfrequenzen verhindert die ungewollte Interferenz von Sendern, wodurch ein ungestörter Empfang der Signale möglich wird. Diese Lizenzierung ist für das LiFi-Spektrum bisher nicht vorgesehen und könnte darüber hinaus auch nicht nötig sein, da eine scharfe räumliche Trennung der ausgesendeten Signale möglich ist. Anders als Strahlung im Megahertz- und Gigahertzspektrum verhält sich die kurzwellige Terahertzstrahlung so, wie es aus dem Alltag von Licht bekannt ist. Sie kann somit durch Wände und Hindernisse begrenzt werden und durch die Richtcharakteristik der Lichtquelle fokussiert werden. Darüber hinaus wird durch die räumliche Begrenzung die Möglichkeit einer absolut abhörsicheren Verbindung geschaffen, da sich das Empfangsgerät innerhalb des klar definierten und sichtbaren Lichtkegels befinden muss.

Weitere Vorteile ergeben sich daraus, dass bereits installierte Leuchtmittel für eine doppelte Nutzbarkeit um ein Vorschaltgerät ergänzt und somit in die bestehende Infrastruktur integriert werden könnten. Dadurch werden Kosten und Ressourcen für die Produktion der Leuchtmittel und Energie durch die doppelte Nutzung der Leuchten gespart. Die Modulation der Leuchten findet dabei in einer nicht wahrnehmbaren Schnelligkeit statt, sodass eine Beeinträchtigung der Beleuchtungsfunktion vermieden werden kann.[1]

2.1.2 Nachteile

Die räumliche Begrenzung bietet ein großes Potential für zukünftige Anwendungen, ist jedoch gleichzeitig ein limitierender Faktor, was eine großflächige Abdeckung eines LiFi-Signals betrifft. Die scharfen Grenzen führen dazu, dass Schatten

in der Signalabdeckung entstehen können, in denen Informationen nicht empfangen werden können. Dadurch, dass die Strahlung an den meisten Oberflächen reflektiert oder absorbiert wird, ist die vollständige Ausleuchtung eines Raumes mit komplexer Geometrie und einer Vielzahl an Hindernissen nicht trivial. Auch im Außenbereich kann die Strahlung etwa durch Gebäudewände oder Bäume blockiert werden, wodurch ein engmaschiges Netzwerk an Sendern für eine lückenlose Abdeckung nötig wäre.

Ein Datenlink kann nur aufgebaut werden, solange die sendende Einheit Licht abstrahlt. Somit ist ein weiterer Nachteil in Situationen gegeben, in denen die Helligkeit variiert werden soll, insbesondere wenn die Helligkeit stark verringert oder Dunkelheit geschaffen werden soll. Die Dimmung des Lichts führt dazu, dass sich das Signal-Rausch-Verhältnis verschlechtert oder die Datenrate verringert werden muss. Um eine konstante Übertragungseffizienz zu erreichen, wurden bereits Ansätze vorgestellt, die Datenworte in eine Zusammenstellung von Codewörtern zu überführen, die sich an die Dimmstufe anpassen.[9]

2.2 Modulation

Modulation bezeichnet im Allgemeinen die Überführung eines Nutzsignals im Basisband in ein für die Übertragung geeignetes Frequenzband durch die Kombination mit einem oder mehrerer Trägersignale. Im folgenden Abschnitt wird das Prinzip der Modulation anhand der Amplitudenmodulation verdeutlicht. Anschließend wird vor allem auf die für die Datenübertragung mit sichtbarem Licht relevanten Modulationsverfahren weiter eingegangen. Zum Verständnis wurde [10] herangezogen und an vielen Stellen sinngemäß übernommen. Daher erfolgt die Erwähnung am Ende eines Absatzes, der Erklärungen aus dem Buch enthält.

2.2.1 Amplitdudenmodulation

Bei der Amplitudenmodulation wird ein hochfrequentes, sinusförmiges Trägersignal mit dem tieffrequenten Nutzsignal moduliert. Mathematisch betrachtet entspricht dies der Multiplikation des modulierenden Nutzsignals mit dem Trägersignal. Es wird von Mischung des modulierenden Signals mit dem Trägersignal gesprochen.[10]

$$u_{\mathrm{M}}(t) = \underbrace{U_1 \cdot \cos(\omega_1 t)}_{\text{Basisbandsignalu}_1(\mathrm{t})} \cdot \underbrace{U_{\mathrm{T}} \cdot \cos(\omega_{\mathrm{T}} t)}_{\text{Trägersignalu}_{\mathrm{T}}(\mathrm{t})} \tag{2.1}$$

$$= \frac{U_1 U_{\mathrm{T}}}{2}[\cos([\omega_{\mathrm{T}} - \omega_1] \cdot t) + \cos([\omega_{\mathrm{T}} + \omega_1] \cdot t)] \tag{2.2}$$

Dabei ist $\omega_1 \ll \omega_{\mathrm{T}}$. Die Information des Nutzsignals befindet sich in der Amplitude des Trägersignals. Der trigonometrischen Beziehung bei der Multiplikation zweier Cosinus-Schwingungen entsprechend, entsteht ein neues Signal, das aus zwei Cosinus-Termen mit Frequenzen $\omega_{\mathrm{T}} - \omega_1$ und $\omega_{\mathrm{T}} + \omega_1$ besteht.[10]

Dieser Fall gilt für ein rein cosinusförmiges modulierendes Signal. Reale Signale bestehen dagegen in der Regel aus einer Überlagerung vieler Schwingungen mit unterschiedlichen Frequenzen und Amplituden. Dies führt dazu, dass sich im Übertragungsband ein Frequenzbereich um ω_{T} auftut, der $2 \cdot \omega_{\mathrm{g}}$ breit ist und symmetrisch zu ω_{T} ist, wobei ω_{g} die höchste auftretende Frequenz im Basisband, die Grenzfrequenz, ist. Der Bereich zu höheren Frequenzen als ω_{T} wird oberes Seitenband genannt und liegt im Bereich von ω_{T} bis $\omega_{\mathrm{T}} + \omega_{\mathrm{g}}$, während der Bereich zu kleineren Frequenzen von $\omega_{\mathrm{T}} - \omega_{\mathrm{g}}$ bis ω_{T} unteres Seitenband genannt wird.[10]

Die Demodulation stellt die Umkehr der Modulation und damit die Rückgewinnung des Nutzsignals am Empfänger dar. Bei der Amplitudendemodulation wird dazu das empfangene Signal mit dem Trägersignal gemischt.

$$u_2(t) = u_{\mathsf{M}}(t) \cdot cos(\omega_{\mathsf{T}} t) \tag{2.3}$$

$$= [U_0 + u_1(t)] \cdot cos^2(\omega_{\mathsf{T}} t) \tag{2.4}$$

$$= [U_0 + u_1(t)] \cdot \frac{1}{2} \cdot [1 + cos(2\omega_{\mathsf{T}} t)] \tag{2.5}$$

Wie in Gleichung 2.5 zu sehen ist, wird neben einem Gleichanteil U_0 und einer Schwingung bei der doppelten Trägerfrequenz $2\omega_{\mathsf{T}}$ auch das Nutzsignal $u_1(t)$ zurück gewonnen. $u_1(t)$ kann im weiteren Verlauf durch die Entfernung des Gleichanteils und Unterdrückung der Schwingung bei $2\omega_{\mathsf{T}}$ etwa mittels eines Filters isoliert werden. Diese Art der Signalrückgewinnung wird kohärent genannt, weil der Empfänger sich mit dem Empfangssignal synchronisieren muss, die Phase also bekannt sein muss. Es ist auch eine inkohärente Demodulation des Amplitudensignals ohne Kenntnis der Phasenlage möglich, auf die an dieser Stelle nicht eingegangen werden soll, da ein Verständnis des inkohärenten Demodulators für den praktischen Teil dieser Arbeit nicht relevant ist.[10]

2.2.2 Digitale Modulation

Digitale Signale zeichnen sich dadurch aus, dass sie diskrete, das heißt diskontinuierliche, Größen im Zeit- und im Wertebereich annehmen. In einem einfachen Fall besteht das Nutzsignal aus einer Abfolge von Einsen und Nullen, welche direkt verwendet werden können, um einen Sinusträger zu schalten. Bei dem sogenannten *On-Off Keying* (OOK) repräsentiert das Vorhandensein des Sinusträgers eine logische Eins, während die Abwesenheit des Trägers als logische Null interpretiert wird. Das resultierende Übertragungssignal ist in Abbildung 2.1 gezeigt. Das *Frequency-Shift Keying* (FSK) ist ein ähnliches Modulationsverfahren, bei dem je nach übertragenem Symbol die Frequenz des Trägers um einen Frequenzhub Δf angehoben oder abgesenkt wird. Beim *Phase-Shift Keying* (PSK) wird anstelle der Frequenz die Phase des Sinusträgers mit dem Nutzsignal moduliert.[10]

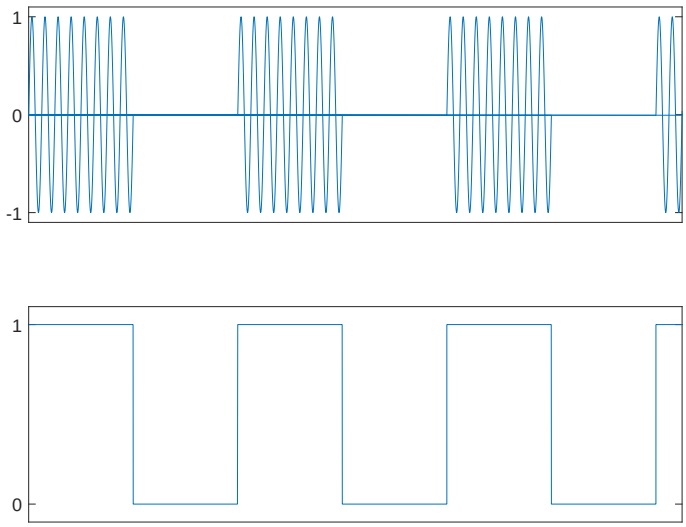


Abbildung 2.1: Grafische Darstellung des Modulationsverfahrens On-Off-Keying. Unten: Nutzsignal, welches das Trägersignal moduliert. Oben: Moduliertes Übertragungssignal.

Pulsamplitudenmodulation

OOK ist ein binäres Modulationsverfahren, bei dem ein Informationssymbol genau einem Bit entspricht. Bei mehrstufigen Modulationsverfahren können die Informationen mehrerer Bits in einem Informationssymbol zusammengefasst werden. Betrachten wir wieder den Fall des OOK, so ist auch denkbar, dass zusätzlich zu den Niveaus „0“ und „1“ auch negative Symbole, Werte größer als 1 oder Werte zwischen 0 und 1 übertragen werden können. Die Übertragung zweier Bits in einem Symbol wird beispielsweise durch die Verwendung von vier Niveaus ermöglicht. Das Verwenden mehrerer Niveaus beansprucht keine zusätzliche Bandbreite und geschieht lediglich durch Anpassung der Signalamplitude, weshalb dieses Verfahren als Pulsamplitudenmodulation (PAM) bezeichnet wird. Für die Anzahl der verwendeten Niveaus ist die Variable M geläufig, wodurch auch von *M-PAM* gesprochen wird. Um n Bits zu übertragen, sind $M = 2^n$ Niveaus nötig. Umgekehrt

können mit M Niveaus $n = \log_2(M)$ Bits übertragen werden. Bei einer asymmetrischen PAM befinden sich alle Niveaus bei Signalwerten ≥ 0. Dies ist besonders in Anwendungsfällen relevant, bei denen keine negativen Signalwerte übertragen werden können, wie bei der Übertragung von Licht. Um Bitfehlerraten zu erreichen, bei denen eine Informationsübertragung möglich ist, muss der Abstand der Niveaus im Verhältnis zur Empfangsleistung ausreichend groß sein, wodurch die maximale Anzahl der Niveaus begrenzt wird. Signalstörungen wie Rauschen oder Interferenzen können dazu führen, dass falsche Niveaus am Empfänger detektiert werden und die Information bei der Übertragung verloren geht.[10]

Pulsdauermodulation und Pulsphasenmodulation

Alternativ zur Signalamplitude kann auch die Zeitkomponente genutzt werden, um mehrere Bits in einem Symbol zu übertragen. Die Zeitdauer *T* eines Symbols kann in M Teile segmentiert werden, um 2^M Bits übertragen zu können. Die daraus resultierende Bitrate beträgt M/T. Bei der Pulsweitenmodulation (PWM) oder Pulsdauermodulation (PDM) entscheidet der Zeitpunkt t der Signalflanke im Intervall T über den Wert der zusammengefassten Bits. Da hierdurch die Dauer des Pulses moduliert wird, ist die Bezeichnung Pulsdauermodulation eigentlich physikalisch präziser, da der Begriff „Weite“ lediglich bei Betrachtung des Signals auf einem Oszilloskop einen Sinn ergibt. Anstelle der Signalflanke kann auch der Zeitpunkt t eines Signalpulses mit unveränderlicher Dauer $T_{\text{Puls}} \leq T/M$ im Intervall T variiert werden, um mehrere Bits in einem Symbol zu übertragen, was als Pulsphasenmodulation (PPM) bezeichnet wird. Zur Erkennung des Zeitpunktes t relativ zum Intervall T ist ein Taktsignal notwendig, das Sender und Empfänger synchronisiert. Alternativ dazu kann auch die zeitliche Differenz zweier Pulse verwendet werden, um den Wert der Bits zu übertragen (Puls-Pausen-Modulation). Dadurch entfällt die Notwendigkeit eines synchronisierenden Taktsignals.

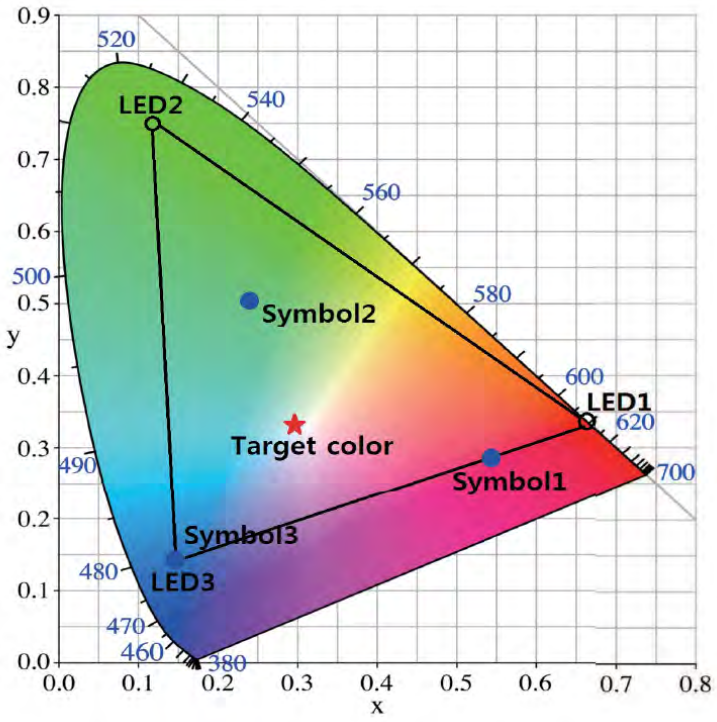


Abbildung 2.2: Abbildung der Informationssymbole in einem Farbraum, der durch drei einfarbige LEDs erzeugt wird. Grafik entnommen aus [4].

Color-Shift Keying

Die Mischung von rotem, grünem und blauem Licht aus LEDs erzeugt für das menschliche Auge weißes Licht. Diese Tatsache wird bei der Datenübertragung mit sichtbarem Licht genutzt, um ein spezielles Modulationsverfahren anzuwenden. Bei dem *color-shift keying* (CSK) werden Bitfolgen einer bestimmten Farbe zugeordnet, die am Sender emittiert wird. Der Empfänger besitzt drei Photodetektoren, die durch Filterung je eine der drei Farben detektieren können. Die Häufigkeit und Intensität der Farben darf nicht dem Zufall überlassen werden, um eine konstante Farbwahrnehmung zu erreichen. Diesem Zusammenhang wird bei der *color intensity modulation* (CIM) Rechnung getragen, indem die Übertragungssymbole entsprechend im Farbraum – der Summe aller möglichen Mischfarben aus den

drei farbigen LEDs – angeordnet werden.[4] In Abbildung 2.2 ist die Verteilung der Informationssymbole um die gewünschte Zielfarbe herum veranschaulicht (markiert durch die Beschriftung *target color*). CIM ermöglicht auch die Dimmung der Lichtquelle bei Maximierung der erreichbaren Datenrate. Beim CSK wird diese Idee weitergeführt und die zusätzliche Bedingung eingeführt, dass die Signaleinhüllende eine konstante Leistung haben muss.[11] Dadurch verringern sich die wahrnehmbaren Signalfluktuationen und gesundheitlichen Risiken, die etwa bei einer Epilepsie aufgrund von flackernden Lichtquellen auftreten können, wird vorgebeugt. Das *Institute of Electrical and Electronics Engineers* (IEEE), das eine Vielzahl an weltweit anerkannten Standards in den Ingenieurswissenschaften veröffentlicht hat, entwickelte bereits 2018 den Standard IEEE 802.15.7, der CSK mit bis zu vier Bits pro Symbol für die optische Datenübertragung definiert. Da für die Übertragung von vier Bits pro Symbol $2^4 = 16$ Punkte im Farbraum benötigt werden, wird von 16-CSK gesprochen.[12]

2.3 Leitungscodierung

Leitungscodierung dient dazu, das Sendesignal an die physikalischen Gegebenheiten bei der Übertragung anzupassen. Anforderungen, die aus den physikalischen Eigenschaften des Senders und des Empfängers resultieren können, umfassen etwa eine Gleichspannungsfreiheit, die Notwendigkeit eines Taktsignals, eine gewisse Störresistenz oder die Beschränkung auf eine bestimmte Bandbreite. Zudem kann eine hohe spektrale Effizienz und eine geringe Störempfindlichkeit angestrebt werden.[13]

Die optische Datenübertragung funktioniert, anders als die Datenübertragung mit elektrischen Spannungen per Leiterkabel, nur mit Signalwerten größer oder gleich null, weil keine „negativen Lichtintensitäten“ übertragen werden können.

Für die Leitungscodierung resultiert daraus, dass asymmetrische Codes verwendet werden müssen, wobei sich die Asymmetrie auf die Verteilung der Signalwerte um den Wert null herum bezieht. Bei der Anwendung von LiFi muss zudem der doppelten Verwendung des Senders als Signalüberträger und als Lichtquelle Rechnung getragen werden. Auch wenn der momentane Signalwert aufgrund der Trägheit des Auges bei Frequenzen weit über 100 Hz nicht wahrgenommen wird, ist dennoch ein zeitlicher Mittelwert über mehrere Informationssymbole wahrnehmbar, der abhängig vom Signal variieren und als störende Fluktuation der Lichtquelle auftreten kann. Abhilfe verschafft der Manchester-Code, der in dem folgenden Abschnitt erklärt werden soll. Anders als bei vielen elektronischen Übertragungsstrecken, stellt das Vorhandensein eines Mittelwertes an sich jedoch kein Problem dar.

2.3.1 Manchester-Code

Der Manchester-Code ist ein Leitungscode, der aus den zwei Eingangssymbolen „0“ und „1“ ein Signal bildet, das aus zwei verschiedenen Ausgangssymbolen besteht. Leitungscodes mit zwei Ausgangssymbolen werden als binäre Leitungscodes bezeichnet. In einer Anwendung, die auch als *Biphase-Code* bezeichnet wird, haben die Symbole die Werte „-1“ und „+1“ – der Wert „0“ kommt nicht vor. Der Biphase-Code wird in einigen elektronischen Übertragungsnetzwerken verwendet, ist jedoch aufgrund des negativen Signalwertes für die optische Datenübertragung ungeeignet. Anstelle des Wertes „-1“ kann jedoch der Wert „0“ verwendet werden,

Tabelle 2.1: Vorschriften der Manchester-Codierung.

Eingang	Manchester	Biphase
1	0 1	-1 +1
0	1 0	+1 -1

wodurch der Code asymmetrisch und der negative Anteil entfernt wird. Der Code funktioniert dann nach den simplen Ersetzungen, die in Tabelle 2.1 aufgeführt sind. Da aus einem Eingangssymbol zwei Ausgangssymbole resultieren, hat ein Ausgangssymbol nur noch die Hälfte der Symboldauer eines Eingangssymbols. Die benötigte Signalbandbreite wird durch die Codierung dementsprechend verdoppelt.[13]

2.4 Elektronik

Im Zusammenhang mit der hochfrequenten Datenübertragung von Licht sind einige elektronische Bauteile von besonderer Relevanz, die im Folgenden ausführlich beleuchtet werden sollen. Zur Erstellung dieses Kapitel wurden [7] und [14] herangezogen und einige Erklärungen sinngemäß übernommen. Die Referenzen tauchen daher erst am Ende des entsprechenden Absatzes auf.

2.4.1 Leuchtdioden

Leuchtdioden (englisch *light emitting diode*, LED) bestehen wie viele Halbleitbauteile aus einem Übergang zwischen einem n-dotierten und einem p-dotierten Halbleitermaterial, einem pn-Übergang. Beim Anlegen einer Spannung in Flussrichtung beginnt die LED ab der sogenannten Flussspannung U_F Licht auszusenden. Die Verwendung verschiedener Materialien führt zu verschiedenen Werten von U_F und verschiedenen ausgestrahlten Wellenlängen. Einige Materialien und ihre zugehörigen Werte sind in Tabelle 2.2 gezeigt.[7]

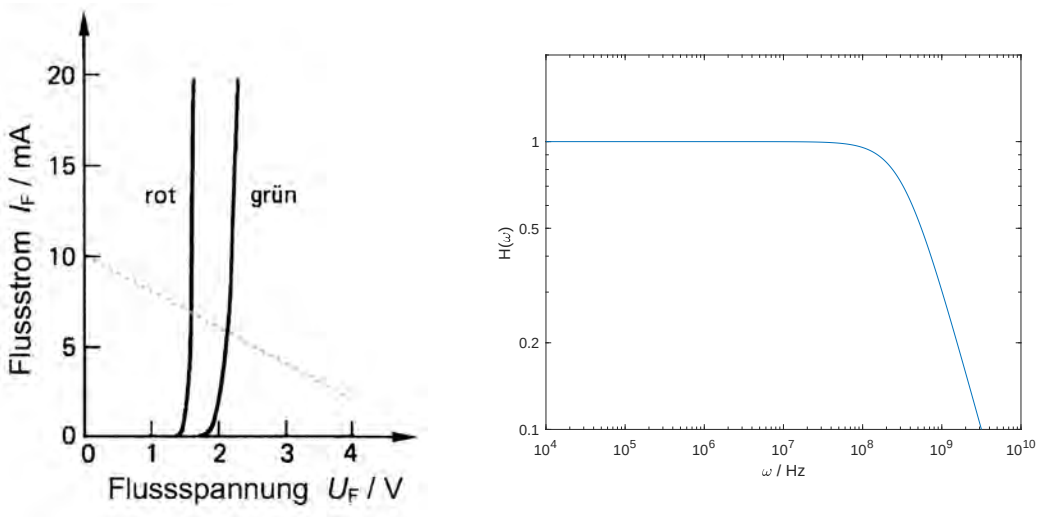
Der elektrische Strom, der die LED durchfließt wird als Flussstrom I_F bezeichnet. Die Leuchtkraft einer LED verändert sich mit dem Flussstrom. Als Maß für die Leistung der elektromagnetischen Strahlung wird die Strahlungsleistung Φ_e

Tabelle 2.2: LED-Materialien und ihre zugehörigen Kennwerte. Die Werte wurden aus [7] entnommen.

Material	**Farbe**	**Wellenlänge**	**Flussspannung**	**Lichtausbeute**
		λ / nm	U_F / V	η / (lm/W)
GaAs:Si	IR	930	1,3	-
$GaAs_{0,6}P_{0,4}$, AlGaInP	Rot	650	1,8	35
AlGaInP	Gelb	590	2,2	130
GaP:N, AlGaInN	Grün	570	2,4	100
InGaN, AlGaInN	Blau	470	3,5	25
InGaN + YAG:Ce	Weiß	-	3,5	110

mit der Einheit J/s verwendet. Für kleine Ströme $I_\mathrm{F} < 30\,\mathrm{mA}$ ergibt sich ein näherungsweise linearer Zusammenhang zwischen Flussstrom und Strahlungsleistung Φ_e. Für höhere Ströme flacht die Erhöhung der Strahlungsleistung dagegen ab oder wird sogar negativ. Da der differentielle Widerstand nach Überschreiten der Flussspannung sehr gering ist (in der Größenordnung von wenigen Ω), führt eine geringfügige Änderung der Spannung bereits zu einer großen Änderung des Flussstroms. Dieses Verhalten ist in Abbildung 2.3a dargestellt. Wird die LED jedoch mit kleinen Strömen betrieben, kann die Strahlungsleistung direkt über die Einstellung des Flussstroms moduliert werden. Die ausgesendeten Spektren sind dabei mit einer Bandbreite von etwa 40 nm sehr schmal im Vergleich zu anderen Leuchtmitteln. Der Abstrahlwinkel ist über die Gehäuseform der LED einstellbar und kann je nach Anwendungsfall variiert werden.[7]

Die Modulationsübertragungsfunktion $H(\omega)$ beschreibt das dynamische Verhalten einer Leuchtdiode. Sie gibt das Verhältnis der Strahlungsleistung bei einer Anregungsfrequenz $\Phi_1(\omega)$ zu einer Strahlungsleistung bei Gleichstrom $\Phi_1(\omega = 0)$ an.



(a) **(b)**

Abbildung 2.3: (a): Strom-Spannungsverhalten zweier Leuchtdioden aus unterschiedlichen Materialien mit eingezeichneter Vorwiderstandsgeraden (blau). Bearbeitet nach [7]. **(b)**: Modulationsübertragungsfunktion einer LED mit $\tau = 2\,\text{ns}$.

$$H(\omega) = \frac{\Phi_1(\omega)}{\Phi_1(\omega = 0)} = \frac{1}{\sqrt{1 + (\omega\tau)^2}} \tag{2.6}$$

τ bezeichnet die Lebensdauer der angeregten Ladungsträger und ist ein materialspezifischer Wert, der in vielen Datenblättern von Leuchtdioden zu finden ist. Mithilfe von Gleichung 2.6 kann eine 3-dB Grenzfrequenz definiert werden, die angibt, bei welcher Frequenz $\omega_{3\text{dB}}$ sich die Strahlungsleistung in Bezug auf Gleichstrom halbiert.[7]

$$H(\omega_{3\text{dB}}) = 0,5 = \frac{1}{\sqrt{1 + (\omega_{3\text{dB}}\tau)^2}} \quad \longrightarrow \quad \frac{\sqrt{3}}{\tau} = \omega_{3\text{dB}} \tag{2.7}$$

$$f_{3\text{dB}} = \frac{\sqrt{3}}{2\pi\tau} = \frac{0,276}{\tau} \tag{2.8}$$

Eine LED mit einer Lebensdauer der angeregten Ladungsträger $\tau = 2\,\text{ns}$ hätte somit eine 3-dB Grenzfrequenz von 137,8 MHz. Das Übertragungsverhalten einer solchen LED nach Gleichung 2.6 ist in Abbildung 2.3b dargestellt.

Verschaltung einer LED

Wie in Abbildung 2.3a zu erkennen ist, steigt der Flussstrom einer Leuchtdiode rapide an, nachdem die Flussspannung überschritten ist. Kleine Schwankungen der angelegten Spannungen führen zu großen Änderungen des Stroms. Der maximale Strom, der die LED durchfließen kann ist jedoch begrenzt und oftmals schon nach einigen zig oder wenigen hundert Milliampere erreicht.[15, 16, 17] LEDs, die für hohe Leistungen ausgelegt sind, können dagegen bis in den Ampere-Bereich arbeiten.[18] Zur Begrenzung des Stroms kann die LED mit einem Vorwiderstand betrieben werden, der seriell zur Spannungsquelle eingesetzt wird. Der Strom durch LED und Widerstand R_V beträgt dann [7]

$$I_\text{F} = \frac{U_\text{R}}{R_\text{V}} = \frac{U_\text{S} - U_\text{F}}{R_\text{V}}\ . \tag{2.9}$$

Dieser Gerade kann in die *U-I*-Kennlinie der LED übertragen werden und bildet mit dieser einen Schnittpunkt, aus dem der Arbeitspunkt der Diode abgelesen werden kann. In Abbildung 2.3a ist dies exemplarisch für einen Vorwiderstand von $R_\text{V} = 500\,\Omega$ gezeigt.

2.4.2 Photodiode

Die Photodiode wird umgekehrt zur Leuchtdiode dafür benutzt, Licht in einen elektrischen Strom zu wandeln. Dieselbe Umwandlung ist zudem auch mit Photowiderständen oder Phototransistoren möglich. Diese sollen an dieser Stelle jedoch

nicht behandelt werden, da sie weit niedrigere Grenzfrequenzen besitzen und deshalb für dynamische Anwendungen im Megahertz-Bereich nicht geeignet sind.[14]

Die Erzeugung einer Spannung, die letztendlich zu einem elektrischen Strom führt, geschieht durch die Absorption eines Photons, dessen Energie ein Elektron in das Leitungsband des Halbleiterkristalls anhebt. Das Elektron hinterlässt ein Loch im Valenzband und wird durch die intrinsische Spannung im Material sofort zur n-Seite abgeführt, während das Loch zur p-Seite abgeführt wird. Durch die ungleiche Ladungsverteilung wird eine Spannung zwischen den beiden Enden der Photodiode erzeugt, die sogenannte Leerlaufspannung U_L. Durch einen Kurzschluss der beiden Enden wird ein elektrischer Strom ermöglicht, der sogenannte Kurzschlussstrom I_K oder Photostrom I_Ph. Der Kurzschlussstrom hängt bei einer gegebenen Photodiode linear von der eingehenden Strahlungsleistung Φ_e ab - eine Eigenschaft, die ebenfalls in Analogie zur Leuchtdiode steht. Zudem wird der Kurzschlussstrom durch die Wahrscheinlichkeit der Absorption eines Photons bei einer gegebenen Wellenlänge beeinflusst, welche als Quantenausbeute $\eta(\lambda)$ bezeichnet wird.[7][14]

Der durch Lichteinstrahlung erzeugte Strom kann in Flussrichtung direkt genutzt werden, um eine Leistung zu erzeugen, wodurch die Photodiode als Photoelement benutzt wird. Wird dagegen eine Spannung an die Photodiode angelegt, die entgegen der Stromrichtung des Kurzschlussstroms abfällt, spricht man von einem Betrieb in Sperrrichtung. Es fließt ein geringer Dunkelstrom durch die Photodiode, der mit zunehmender Beleuchtungsstärke größer wird. Anders als bei dem Betrieb in Flussrichtung ändert sich der Strom bei Variation der Sperrspannung kaum. Durch diesen Betrieb wird die Sperrschichtkapazität C_S, die am pn-Übergang der Diode entsteht, abgebaut, wodurch die Diode schneller auf Änderungen der Strahlungsleistung reagiert. Das Schaltungsbild ist in Abbildung 2.4 gezeigt.[7]

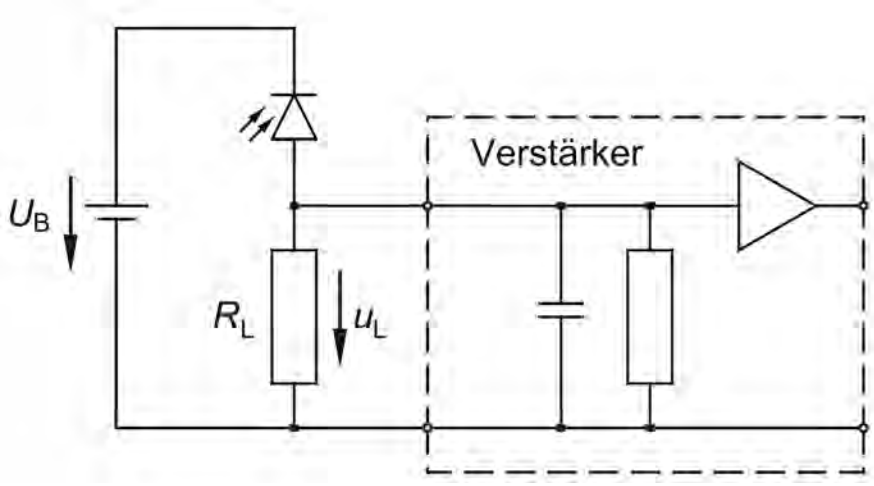


Abbildung 2.4: Photodiode mit Vorspannung in Sperrrichtung (Diodenbetrieb). Bearbeitet nach[7].

Eine schnelle Reaktion auf die Änderung der Strahlungsleistung ist bei einer Anwendung zur Übertragung von hochfrequenten Signalen entscheidend. Neben der zuvor beschriebenen Sperrschichtkapazität hat auch der in Abbildung 2.4 gezeigte Lastwiderstand R_L einen Einfluss auf die Umladung der Sperrschicht bei einer eingehenden Wechselspannung. Widerstand und Kapazität bilden zusammen eine Zeitkonstante

$$\tau_{RC} = C_\mathrm{S} R_\mathrm{L} \, . \tag{2.10}$$

Hinzu kommen Prozesse für den Ladungsträgertransport im Inneren der Photodiode, die zu einer Verzögerung führen. Diese Prozesse sind bauteilspezifisch und können nur durch das herstellerseitige Design der LED beeinflusst werden.[7]

pin-Photodiode

Herkömmliche Photodioden haben relativ hohe intrinsische Diffusionszeiten der Ladungsträger in der Größenordnung von 1 µs. Grenzfrequenzen in der Größenordnung von 200 kHz bis etwa 1 MHz sind die Folge. Die Diffusionszeit kann je-

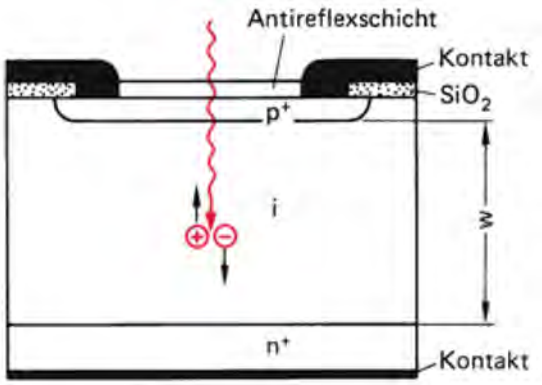


Abbildung 2.5: Aufbau einer pin-Photodiode. Die breite i-Zone führt dazu, dass p- und n-Zone sehr schmal gestaltet werden können.[7]

doch entscheidend verringert werden, wenn zwischen die p- und die n-Schicht der Photodiode ein intrinsisch leitendes Material (hochreiner Halbleiterkristall) eingeschlossen wird. In Bezug auf die intrinsische Leitfähigkeit des eingebundenen Materials wird die Zone i-Zone genannt. Sie sorgt durch eine gute Absorptionsfähigkeit dafür, dass viele Elektronen durch eine Photonabsorption in das Leitungband angeregt werden und schnell in die n-Zone (Löcher analog dazu in die p-Zone) abgeleitet werden. Da die p- und die n-Zone nun sehr dünn gestaltet werden können, benötigen die Ladungsträger kaum noch Zeit für die Diffusion in den p- und n-Materialien. Die sogenannte Transitzeit τ_D in der i-Zone wird ausschlaggebend für die zeitliche Verzögerung. Sie ist in der Größenordnung von Nanosekunden, sodass Grenzfrequenzen bis in den Gigahertzbereich möglich werden. Der Aufbau einer pin-Diode ist in Abbildung 2.5 gezeigt.[7] [19]

2.5 Mikrocontroller und SoCs

Mikrocontroller vereinen eine große Anzahl hochintegrierter, elektronischer Bauteile auf einer einzelnen Platine oder einem Chip. Kernstück eines Mikrocontrol-

lers ist eine zentrale Recheneinheit (*central processing unit, CPU*), die dazu in der Lage ist, sehr viele arithmetische und logische Operationen pro Sekunde durchzuführen, auf angebundene Speicher zuzugreifen, sowie Ein- und Ausgaben zu tätigen. Neben der CPU befinden sich auf einer Mikrocontrollerplatine weitere Bausteine wie ein Taktgeber, Speicher – insbesondere vom Typ *read-only memory* (ROM), *random access memory* (RAM) und Flash – Datenbusse, Ein- und Ausgabepins (sogenannte *general-purpose inputs/outputs*, GPIOs), Zeitgeber (*Timer*) und Register für Verarbeitungsunterbrechungen (*interrupt register*). Weitere Bausteine können ergänzt werden, die die Anwendungsgebiete eines Mikrocontrollers noch universeller machen und dem Anwender eine Vielzahl von Möglichkeiten eröffnen. Hersteller sprechen oftmals in Abgrenzung zu simpleren Ausführungen von *systems on a chip* (SoC), bei denen beispielsweise zusätzlich Grafikprozessoren, serielle Schnittstellen, Radiomodule oder Umweltsensoren ergänzt werden. SoCs besitzen nicht selten eine große Ähnlichkeit zu vollwertigen Computern. Die Bauteile des SoCs sind jedoch fest auf der Platine integriert und können nicht wie bei einem Motherboard-basierten Computer über Steckplätze ausgetauscht oder ergänzt werden. Die Programmierbarkeit der Mikrocontroller bietet dennoch eine Möglichkeit weitere Komponenten über die Ein- und Ausgabeschnittstellen der SoCs anzubinden, sofern ein geeignetes Datenaustauschverfahren gefunden wird, mit dem sowohl SoC als auch die Komponente kompatibel sind.

2.5.1 Interrupts

Mikrocontroller führen normalerweise ihren Programmcode zyklisch aus. Dies führt dazu, dass, abhängig von der Laufzeit des Programmcodes, mitunter sehr langsam auf äußere Einflüsse oder veränderte Variablen reagiert werden kann. Bestünde die Aufgabe eines Programms beispielsweise darin, so lange eine Reihe von Sensorwerten auszulesen bis ein Schalter gedrückt wird, so würde das Auslesen der Sensoren erst nach der Abfrage des Schalterzustands beendet werden. Benötigt das Auslesen eines Sensorwertes gegebenenfalls sogar einen signifikanten Anteil

der Programmlaufzeit könnte dies mit einer merkbaren zeitlichen Verzögerung einhergehen.

Unterbrechungen der Laufzeit, bezeichnet als *Interrupts*, können die Ausführung des Programms an der aktuellen Stelle pausieren und die Ausführung einer anderen Routine, der sogenannten *interrupt service routine* (ISR) einleiten. Nach Beendigung der ISR wird das Programm an der zuvor unterbrochenen Stelle im Hauptprogramm fortgesetzt. In dem beschriebenen Beispiel könnte die Betätigung des Schalters also einen Interrupt auslösen, dessen ISR die Abfrage weiterer Sensorwerte unterbricht.

Interrupts können hardwareseitig, etwa durch die Veränderung eines Spannungswertes an einem GPIO, oder softwareseitig, beispielsweise durch den Ablauf eines Timers, ausgelöst werden. Während der Ausführung einer ISR werden andere Interrupts ausgesetzt. Nachdem die ISR beendet wurde, werden Interrupts, die gegebenenfalls während der Ausführung der vorherigen ISR ausgelöst wurden in der Reihenfolge einer vorgegebenen Priorisierung abgearbeitet. ISRs sollten so wenig Zeit wie möglich beanspruchen, um die Ausführung anderer möglicherweise für die Funktionsfähigkeit des Mikrocontrollers relevanter Routinen nicht zu unterdrücken.[20, 21]

2.5.2 Timer

Zeitgeber (auf englisch *timer*) können hardware- und softwareseitig in einem Mikrocontroller implementiert sein. Timer besitzen eine gewisse Ungenauigkeit, die an die Präzision des Taktgebers auf dem SoC, üblicherweise ein Schwingquarz, gebunden ist. Bei der Entwicklung von zeitkritischen Anwendungen ist zu beachten, dass die geringen Abweichungen der Resonanzfrequenz der Taktgeber zu einem zeitlichen Drift führen, wenn mehrere taktgesteuerte Mikrocontroller miteinander

kommunizieren. Die geringe Abweichung der Taktgeber führt schon nach wenigen Sekunden dazu, dass sich mikrosekundengenaue Zeitschritte zueinander verschieben und nicht mehr zeitgleich stattfinden.

Timer können dazu genutzt werden, Programmabschnitte zeitlich zu strukturieren. Beispielsweise kann eine gewisse Zeit gewartet werden, bevor der nächste Programmschritt durchgeführt wird oder es kann untersucht werden, wie lange die Ausführung eines bestimmten Programmabschnittes gedauert hat.

Eine wichtige Funktion übernehmen Timer in Zusammenhang mit Interrupts. *Timed interrupts* lösen Interrupts aus, nachdem eine definierte Zeit abgelaufen ist. Wird der Timer nach Ablauf automatisch neu geladen, können auf diese Weise Routinen periodisch ausgeführt werden. Sensorwerte können in definierten zeitlichen Abständen ausgelesen werden, Buffer nach einer bestimmten Zeit mit Daten gefüllt oder Daten entnommen werden, wodurch zeitkritische Anwendungen erst möglich gemacht werden. Ohne einen Zeitgeber würde der zeitliche Abstand nur von der Ausführungsdauer des Programmcodes abhängen, die von vielen Faktoren abhängig und schwer vorhersagbar ist .

Timer können niemals genauer als der Taktgeber arbeiten, der sie mit dem Zeitsignal versorgt. Oftmals wird ein Teiler (*prescaler* oder *divider*) verwendet, der die atomaren Zeitschritte des Taktgebers in längere Zeitschritte des Timers übersetzt. Ein Timer mit einem Teiler von 10, der von einem 10 MHz-Taktgeber angetrieben wird, würde nur alle 10 Taktschritte um eins inkrementiert werden und hätte somit eine Taktrate von 1 MHz.[22]

2.5.3 UART

UART steht für *Universal Asynchronous Receiver/Transmitter* und bezeichnet ein integriertes Bauteil, das eine serielle Kommunikation über eine Datenleitung ermög-

Abbildung 2.6: Ein UART-Frame kann aufgrund der variablen Anzahl von Daten, Paritäts- und Stop Bits in der Länge variieren.

licht. Die Kommunikation findet asynchron statt, was bedeutet, dass keine Leitung vorhanden ist, die ein Taktsignal überträgt. Für eine erfolgreiche Kommunikation müssen vor der Übertragung gewisse Parameter beidseitig definiert sein:

- Baudrate, die Anzahl der Bits pro Sekunde
- Datenbits, die Anzahl der nutzbaren Bits nach der Übertragung
- Paritätsbit, ein Bit zur Fehlererkennung
- Stop-Bits, ein oder zwei Bits, die das Ende des Bitwortes markieren
- Datenflusssteuerung, Einflussnahme auf die Übertragung weiterer Bitworte

Die Summe aus Start Bit, Datenbits, Paritätsbit und Stop Bit(s) bildet ein UART-*Frame*. Ein anschauliche Darstellung ist in Abbildung 2.6 gezeigt.

Sender

Sendeseitig beginnt die Übertragung eines UART, sobald Daten im Schieberegister oder einem sogenannten *First In, First Out (FIFO)*-Buffer vorhanden sind. Ein FIFO-Buffer gibt diejenigen Daten als erstes aus, die auch zuerst in den Buffer gegeben wurden. Die Übertragung startet mit dem Versenden eines Start-Bits. Anschlie-

ßend werden die eingegebenen Datenbits mit einer Länge von 5 - 8 Bits übertragen.[23]

Ist das Versenden eines Paritätsbits eingestellt, wird nun das entsprechende Bit angefügt. Das Paritätsbit wird abhängig von der Anzahl der Einsen im Informationswort auf 0 oder 1 gesetzt. Bei der Einstellung *even parity* (gerade Parität) wird das Paritätsbit 1 gesetzt, wenn die Anzahl der Einsen im Informationswort ungerade ist. Umgekehrt ist es 0, wenn die Anzahl der Einsen im Informationswort gerade ist. Es sorgt somit dafür, dass immer eine gerade Anzahl von Einsen übertragen wird. Bei der Einstellung *odd parity* (ungerade Parität), verhält es sich genau umgekehrt: Die Anzahl der Einsen ist also stets ungerade.[24]

Anschließend an das Paritätsbit wird eine festgelegte Anzahl an Stop-Bits gesendet, welche das Ende der Übertragung markieren.[23, 25]

Empfänger

Der Empfänger wartet vor einer Datenübertragung so lange im inaktiven Zustand, bis das Start-Bit empfangen wird. Das Start-Bit gilt als empfangen, wenn nach der Hälfte einer Bitdauer noch derselbe Zustand auf der Datenleitung gemessen wird. Nun erfolgt der Empfang der Datenbits, die mit einem Vielfachen der Bitrate abgetastet werden. Nachdem die vordefinierte Anzahl der erwarteten Datenbits empfangen wurde, wird gegebenenfalls das Paritätsbit empfangen und mit dem erwarteten Wert abgeglichen. Zuletzt wird ein Stop-Bit empfangen oder, falls definiert, werden zwei Stop-Bits empfangen. [23, 26, 27]

Falls das Paritätsbit nicht mit dem erwarteten Wert übereinstimmt oder anstelle der Stop-Bits ein anderer Wert empfangen wird, wird die gesamte Nachricht als ungültig angesehen und verworfen. Im Kontext von Microcontrollern können hierdurch *Parity Error*-Interrupts (bei fehlerhaftem Paritätsbit) und *Frame Error*-Interrupts (bei fehlerhaftem Stop-Bit) ausgelöst werden.[28]

2.6 Programmierung in der Arduino IDE

„Arduino“ ist ein Hersteller von Mikrocontrollerboards, der eine eigene integrierte Programmierumgebung (integrated development environment, IDE) entwickelt hat, die einen leichten Einstieg in die Programmierung von Mikrocontrollerboards mithilfe einer simpleren Variante der Programmiersprache C++ verspricht. Die Software ist quelloffen und kann somit auch mit Boards verwendet werden, die von anderen Herstellern produziert wurden. Außerdem können dadurch Arduino-kompatible Boards auch in anderen Editoren programmiert werden, die über entsprechende Programmerweiterungen an die Programmierung von Arduinos angepasst wurden. Während die Platinen von Arduino eher für eine leichte Handhabung und einen simplen Einstieg gestaltet sind, zielen die Designs der Firma „Espressif“ mehr auf Leistungsstärke und kleine Abmessungen ab und können ebenfalls mittels der Arduino IDE programmiert werden.[29]

2.6.1 Aufbau eines Arduino Programms

Ein Arduino-Programm benötigt immer zwei grundlegende Funktionen: `setup` und `loop`. Der Programmcode wird in einem sogenannten *Sketch* gespeichert.
Die Funktion `loop` enthält das Hauptprogramm in einem Arduino-Sketch. Von ihr aus können weitere Funktionen aufgerufen werden, oder Anweisungen durchgeführt werden. Wird das Ende der `loop`-Funktion erreicht, springt das Programm automatisch wieder an den Anfang zurück, sodass es zyklisch durchlaufen wird.

Die `setup`-Funktion wird zu Beginn einmalig vor der `loop`-Funktion ausgeführt und enthält Anweisungen zur Konfiguration, die vor Beginn des Hauptprogramms ausgeführt werden sollen. Beispielsweise werden bestimmte GPIOs als Inputs oder Outputs definiert, Programmbibliotheken initialisiert oder die serielle Ausgabe gestartet.

Bei der Entwicklung komplexer Programme muss nicht jeder Entwickler den gesamten Quellcode neu schreiben. Es ist möglich, Programmbibliotheken in den Sketch einzubinden, die bereits vorgefertigte Funktionen zur Verfügung stellen. Genauer gesagt wird in den eingebundenen Bibliotheksdateien meistens eine Klasse definiert. Klassen sind ein Konstrukt aus der objektorientierten Programmierung. Da C++ eine objektorientierte Programmiersprache ist, unterstützt es auch die Definition von Klassen. Mithilfe von Klassen können Funktionen Objekten zugeordnet werden, die im Quellcode definiert werden. Solche Funktionen werden Methoden genannt. Beispielsweise kann eine Bibliothek mit dem Namen „Autos.h“ Methoden zur Verfügung stellen, die Merkmale von Autos verändern können. Ein Objekt der Klasse `Autos` kann dann beispielsweise den Namen „Limousine“ tragen. Eine Methode der Klasse `Autos` könnte die Funktion `Autos::setzeFarbe()` sein, mit welcher man das Attribut `Farbe` des Objekts `Limousine` verändern kann und zwar nach dem folgendem Schema:[30]

Codeblock 2.1: Einbindung einer Bibliothek und grundlegende Verwendung einer C++-Klasse.

```
1  #include <Autos.h>
2  Autos Limousine;  /* erstellt ein Objekt der Klasse 'Autos'
    mit dem Namen 'Limousine'*/
3  Limousine.setzeFarbe(schwarz);  /* ruft die Methode
    'Autos::setzeFarbe()' mit dem Parameter 'schwarz' auf */
```

In Codeblock 2.1 ist außerdem zu sehen, wie eine Programmbibliothek mithilfe des `#include`-Befehls eingebunden wird.[31]

2.6.2 Tasks

Ein *Task* ist ein Teilprogramm, das von dem Mikrocontroller im Kontext eines Betriebssystems (*Real-Time Operating System*, RTOS) ausgeführt wird. Die Funktionen `setup` und `loop` können als Tasks angesehen werden, die standardmäßig in einem Arduino-Programm laufen, und in den Fällen der als nächstes betrachteten Boards trifft dies auch genau zu. Der `setupTask` wird einmalig ausgeführt und nach Beenden der `setup`-Funktion terminiert. Der `loopTask` führt die `loop`-Funktion in einer Endlosschleife aus.[32]

Auf den Boards der Hersteller „Seeeduino“ und „Espressif“ ist das quelloffene Betriebssystem FreeRTOS implementiert, wodurch die Handhabe weiterer Tasks ermöglicht wird. Die SoCs des Herstellers Arduino laufen ohne Betriebssystem, wodurch keine zusätzlichen Tasks erstellt werden können.[1] Tasks können bestimmte Metainformationen wie Name, Priorität, *stack size*, Eingabeparameter und die Bezeichnung der CPU, auf der sie laufen, besitzen. Die *stack size* ist ein Maß für die Größe des Programms, das in dem Task ausgeführt wird. Ein *task handle* kann als Objekt übergeben werden, mithilfe dessen Metainformationen zum Task abgefragt und Tasks pausiert oder beendet werden können. Der wichtigste Parameter ist jedoch die Funktion, die ausgeführt werden soll, wenn der Task gestartet wird. In dem `loopTask` wird hier beispielsweise die Funktion `loop()` übergeben. In Systemen mit mehreren CPUs, wie beispielsweise dem ESP32, ermöglichen Task die parallele Ausführung von Programmcode in nahezu unabhängiger Weise (Multiprocessing). Werden mehrere Tasks auf einer CPU ausgeführt, muss darauf geachtet werden, dass die ausgeführte Funktion Methoden enthält, die die Ausführung der anderen Tasks ermöglicht (Multitasking).[34, 35, 36, 37]

[1] Die TaskScheduler Bibliothek ist jedoch für einige Arduino Boards verfügbar und bietet sehr ähnliche Möglichkeiten.[33]

2.6.3 Queues

In Abschnitt 2.6.2 wurden Tasks behandelt, die sich dafür eignen, Codeabschnitte nahezu unabhängig voneinander auszuführen und somit die Leistungsfähigkeit des Programms erhöhen können. *Queues* (Warteschlangen) erweitern die Leistungsfähigkeit von Tasks, indem sie einen Datenaustausch zwischen mehreren Tasks ermöglichen. Beispielsweise kann es passieren, dass Task 1 auf eine Ressource angewiesen ist, die in einem anderen Task 2 generiert wird. Eine ineffiziente Ausführung wäre es, wenn Task 2 regelmäßig überprüfen müsste, ob Task 1 die Ressource schon geliefert hat. Stattdessen kann Task 2 auf eine Queue horchen und die Ausführung solange pausieren, bis Task 1 die Ressource in die Queue geliefert hat. Task 2 kann dann in dem Moment mit der Ausführung fortfahren, sobald Task 1 die Ressource geliefert hat.

Umgekehrt können Queues dazu benutzt werden, eine Zeit lang damit zu warten, eine Ressource in einen Buffer zu schreiben, wenn dieser Buffer momentan noch voll ist. Der Task wartet dadurch ab, bis Speicher frei geworden ist, um die generierte Ressource zu verarbeiten. Queues geben zuerst diejenigen Informationen aus, die sie als erstes erhalten haben und arbeiten damit als FIFO-Buffer.[38][39]

Mehrere Tasks können auf dieselbe Queue zugreifen. Falls Tasks einen Wert aus der Queue lesen möchten, aber die Queue momentan leer ist, werden die Tasks in einen blockierenden Status versetzt. Sobald wieder ein Wert in der Queue vorhanden ist, wird dieser an den Task mit der höchsten Priorität weitergegeben. Haben beide Tasks dieselbe Priorisierung, erhält der Task den Wert, der als erstes in den blockierenden Status versetzt wurde. Analog dazu verhält es sich bei einer vollständig gefüllten Queue, in die mehrere Tasks schreiben möchten.[38][39]

Kapitel 3

Dokumentation und Durchführung der Arbeiten

In diesem Kapitel sollen die durchgeführten Arbeiten nachvollziehbar dokumentiert werden. Zum Verständnis der Programmierung werden entscheidende Funktionen in Abschnitt 3.1 erklärt. In dem anschließenden Abschnitt werden die programmierten Bibliotheken erklärt. In Abschnitt 3.3 werden die erstellten Platinen vorgestellt und ihre Wirkungsweise erklärt. In Abschnitt 3.4 folgen Informationen zur Fokussierung der Lichtquelle im Laboraufbau.

3.1 Verwendete APIs

Application Programming Interfaces (APIs) sind Software-Schnittstellen, die während der Programmierung von dem Entwickler verwendet werden können, um auf Funktionen einer weiteren Software zuzugreifen. Anders als Benutzer-Schnittstellen sind sie nicht dafür ausgelegt, vom Endbenutzer verwendet zu werden, sondern dazu gedacht, den Funktionsumfang für den Entwickler zu erweitern. Beispielsweise kann eine API Funktionen zur Verfügung stellen, die die Handhabe der Hardware auf einem Mikrocontroller erleichtern oder Informationen bei einem Web-

Server abrufen. In diesem Abschnitt werden die wichtigsten verwendeten APIs vorgestellt.

3.1.1 FreeRTOS Multiprocessing

FreeRTOS wurde bereits in Abschnitt 2.6.2 im Zusammenhang mit Tasks erwähnt. Es ist ein quelloffener Betriebssystemkern, der in weiten Teilen unter dem Namen ESP-IDF FreeRTOS in den Mikrocontrollerboards von Espressif implementiert ist. FreeRTOS stellt Funktionen zum Erstellen von Tasks bereit und ermöglicht damit Multiprocessing.[40, 34, 41]

Codeblock 3.1: Erstellen eines Tasks in ESP-IDF FreeRTOS.

```
1  void T_tx(void * parameter){
2    // Funktionsdefinition ...
3  }
4
5  void setup(){
6    xTaskCreate(T_tx, "TX Task", 1024, NULL, 0, NULL);
7  }
```

In Codeabschnitt 3.1 ist gezeigt, wie man einen Task mittels der Funktion `xTaskCreate()` erstellen kann. Als erster Parameter wurde die Funktion `T_tx` übergeben, welche von dem Task mit dem Namen `TX Task` (zweiter Übergabeparameter) ausgeführt wird. Als `stack size` (siehe Abschnitt 2.6.2) wurde der Wert `1024` übergeben. Die verbleibenden Parameter dienen dazu, der Funktion `T_tx` einen Parameter zu übergeben (`NULL`), dem Task eine Priorität zu geben (`0`), und einen *task handle* (siehe Abschnitt 2.6.2) mit dem Task zu verknüpfen (`NULL`). Der sogenannte Nullpointer `NULL` wird benutzt, um keinen Parameter und keinen Task Handle zu übergeben. Mit der Funktion `xTaskCreatePinnedToCore()` ist es alternativ mög-

lich, einen Task zu erstellen, der auf einem bestimmten Prozessorkern ausgeführt werden soll.[42]

Die übergebene Funktion `T_tx` darf keinen Rückgabewert per `return` erzeugen und darf nicht beenden, da ansonsten eine Fehlermeldung erzeugt wird. Aus diesem Grund werden die in Tasks verwendeten Funktionen oftmals als Endlosschleifen mittels `while(true)` oder `for(;;)` konzipiert. Soll die Ausführung des Tasks beendet werden, so kann die Funktion `vTaskDelete()` aufgerufen werden. Der Funktion kann ein Task Handle übergeben werden, welcher an die zu beendende Funktion geknüpft ist. Alternativ kann sich der Task per `vTaskDelete(NULL)` selbst beenden.[42]

3.1.2 Ring Buffer

Die Programmierumgebung von Espressif „ESP-IDF“ bietet eine eigene Variante der von FreeRTOS zur Verfügung gestellten Bibliotheken für Stream- und Nachrichten-Buffer, die *ring buffer* (Ringspeicher). Ringspeicher sind FIFO-Buffer (siehe Abschnitt 2.5.3), bei denen man sich den Speicherprozess wie auf einem Ring vorstellen kann: Neue Elemente werden auf einem Teil des Rings gespeichert, wobei ein neues Element stets dorthin gesetzt wird, wo das nächstältere aufhört. Ist der Speicher bis zum Ende gefüllt, werden neue Elemente an den Anfang des Speichers gesetzt. Dabei ist es auch möglich, dass Elemente aufgeteilt werden, weil ein Teil des Elements an das Ende des (in Wirklichkeit eindimensionalen) Speichers passt und der Rest an den Anfang gelegt wird. Ringspeicher sind insbesondere bei Streaming- und anderen Anwendungen relevant, bei denen in variabler Geschwindigkeit Daten generiert und verarbeitet werden, weil stets sichergestellt wird, dass die konsumierende Einheit – in der Regel ein Prozessor, der die Daten verarbeitet – die maximale Zeitdauer hat, die die Größe des Speichers hergibt bis Daten überschrieben werden.

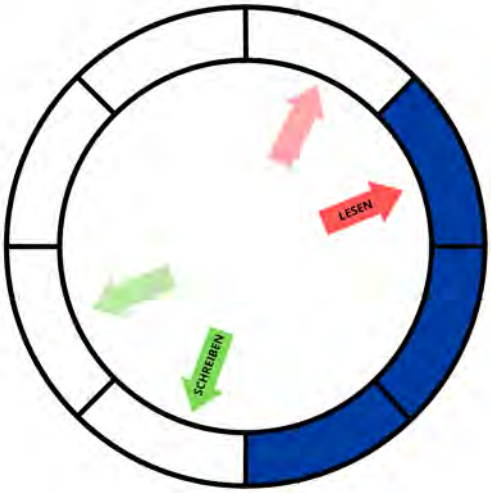


Abbildung 3.1: Darstellung eines Ringspeichers.

In Abbildung 3.1 ist das Konzept eines Ringspeichers dargestellt. Im Uhrzeigersinn wird der Speicher mit eingehenden Daten beschrieben, was durch den grünen Pfeil markiert ist. Der blassgrüne Pfeil markiert die Position, die als übernächstes beschrieben wird. Bereits beschriebene Positionen sind durch eine blaue Füllung markiert. An der Position des roten Pfeils wird als nächstes eine Information aus dem Buffer gelesen. An der Stelle des roséfarbenen Pfeils wurde zuletzt eine Information gelesen. Die fehlende blaue Füllung zeigt an, dass der Speicherplatz nach der Entnahme aus dem Ringspeicher wieder frei geworden ist.

Die Methoden zur Handhabe eines Ringspeichers sind in einem Beispiel in Codeabschnitt 3.2 gezeigt. Zunächst wird ein *ring buffer handle* deklariert, mithilfe dessen der Ringspeicher im weiteren Verlauf referenziert werden kann. Die Initialisierung des Buffers erfolgt im `setup` mithilfe der Methode `xRingbufferCreate`. Die Methode gibt einen Wert vom Typ `RingbufHandle_t` zurück und akzeptiert zwei Übergabeparameter: die Größe des Ringspeichers in Bytes und den Buffer-Typ.[43]

Codeblock 3.2: Handhabe eines Ringspeichers in der Ring Buffer API von ESP-IDF FreeRTOS.

```
1  RingbufHandle_t buf_handle;   // deklariere ring buffer handle
2
3  void T_tx(){
4    // ...
5    byte data = Serial2.read();
6    UBaseType_t res = xRingbufferSend(buf_handle, &data,
     sizeof(data), portMAX_DELAY); // Eingabe von Daten in den
     Ringspeicher
7    if(res != pdTRUE){
8      Serial.println("Failed to send item into ringbuffer");
9    }
10   // ...
11 }
12
13 void T_decode(){
14   size_t size;
15   byte * item = (byte *) xRingbufferReceiveUpTo(buf_handle,
     &size, pdMS_TO_TICKS(1000), 2);
16   if(item == NULL){
17     Serial.println("Could not receive item from the ringbuffer");
18   } else{
19     // item enthält nun Daten und kann hier verarbeitet werden
20     // ...
21
22     vRingbufferReturnItem(buf_handle, (void*)item); // Rückgabe
     des Elemtents, um Speicher freizugeben
23   }
24 }
25
26 void setup(){
27   buf_handle = xRingbufferCreate(2048, RINGBUF_TYPE_ALLOWSPLIT);
28   if(buf_handle == NULL){
29     Serial.println("error creating ringbuffer");
30   }
31 }
```

Es sind folgende Buffer-Typen verfügbar:

- Ein Buffer vom Typ `RINGBUF_TYPE_NOSPLIT` speichert Elemente nur in zusammenhängenden Speicherplätzen. Jedes Element hat einen Overhead von 8 Byte.
- `RINGBUF_TYPE_ALLOWSPLIT` zeichnet einen Buffer aus, in dem die Elemente aufgeteilt werden dürfen, wenn am Ende des Ringspeichers nicht genügend Platz für das ganze Element ist. Dieser Typ nutzt daher den Speicher gegebenenfalls effizienter aus. Ein aufgetrenntes Element erzeugt allerdings neben dem 8-Bytes Overhead pro Element einen zusätzlichen Overhead von 8 Bytes.
- Ein `RINGBUF_TYPE_BYTEBUF` speichert die Elemente als eine kontinuierliche Aneinanderreihung von Bytes. Die Elemente haben keinen Overhead und beim Auslesen des Buffers ist nicht bekannt, wie groß ein Element bei der Eingabe war. Stattdessen kann bei der Ausgabe eine beliebige Anzahl von Bytes ausgelesen werden. [43]

Mit der Methode `xRingbufferSend(RingbufHandle_t handle, const void * data_p, size_t size, TickType_t ticks_to_wait)` können Daten in den Ringspeicher geschrieben werden. Der Methode wird ein ring buffer handle, ein Zeiger auf die Daten `data_p` und die Anzahl der zu schreibenden Bytes `size` übergeben. Mit `ticks_to_wait` kann eine Dauer bestimmt werden, die abgewartet wird, bevor mit der Ausführung des Codes fortgefahren wird, falls der Ringspeicher aktuell vollständig gefüllt ist.[43]

Mit `void * xRingbufferReceive(RingbufHandle_t handle, size_t * size_p, TickType_t tick_to_wait)` können Daten aus dem Ringspeicher gelesen werden. `handle`, `size_p` und `ticks_to_wait` funktionieren analog zu `xRingbufferSend`, mit

den Unterschieden, dass `size_p` ein Zeiger zu einer Variable ist, in die die Anzahl der erhaltenen Bytes geschrieben wird und `ticks_to_wait` eine Zeitdauer angibt, die gewartet wird, falls keine Daten im Speicher vorhanden sind. Der Rückgabewert ist vom Typ `void *` und enthält den Zeiger auf die geschriebenen Daten. Eine Typenkonvertierung, in Beispiel 3.2 Zeile 15 durchgeführt mit `(byte *)`, kann nötig sein, um die Daten anschließend zu verarbeiten.[43]

Ist der Buffer vom Typ `RINGBUF_TYPE_ALLOWSPLIT`, reicht ein einmaliger Aufruf von `xRingbufferReceive()` nicht aus, um ein an den Enden des Buffers aufgetrenntes Element vollständig zu erhalten. Stattdessen kann bei diesem Buffer-Typ die Funktion `xRingbufferReceiveSplit` eingesetzt werden, um gegebenenfalls zwei Elemente zu erhalten. Für den Typ `RINGBUF_TYPE_BYTEBUF` kann die Methode `xRingbufferReceiveUpTo` verwendet werden, um die maximale Anzahl der erhaltenen Bytes zu begrenzen.[43]

Aus dem Buffer gelesene Elemente müssen unbedingt mittels `vRingbufferReturnItem` „zurückgegeben“ werden, damit der Speicher des gelesenen Elementes im Ringspeicher wieder freigemacht wird. Neben dem ring buffer handle wird nur ein Zeiger zum Datenobjekt des zuvor erhaltenen Elementes an die Methode übergeben.[43]

3.1.3 HardwareSerial

Mit der „HardwareSerial“ Bibliothek, die fester Bestandteil des Arduino Cores ist, kann auf die UART-Schnittstellen der Mikrocontrollerboards von Espressif und Arduino zugegriffen werden. Die Funktionsweise von UART wurde bereits in Abschnitt 2.5.3 behandelt. Während der ESP32 drei UART Schnittstellen besitzt, haben die Arduinos zwischen einer und vier Schnittstellen, abhängig von der Ausführung. Eine UART-Schnittstelle wird in der Regel, zumindest während des Entwicklungs-

prozesses, für die Kommunikation mit dem Rechner verwendet, an den das Board per USB angeschlossen ist.[24, 44]

Der häufigste Anwendungsfall der HardwareSerial Bibliothek ist die Verwendung des seriellen Monitors, der während der Verbindung vom Mikrocontroller zu einem Computer benutzt werden kann, um nützliche Daten beim Debugging auszugeben. Die hierzu verwendete Methode heißt `Serial.print`. `Serial` ist dabei ein Objekt der Klasse `HardwareSerial`. Neben `Serial`, das auf die Schnittstelle `UART0` zugreift gibt es noch `Serial1`, `Serial2` und `Serial3`, die auf je eine der anderen Schnittstellen zugreifen, sofern diese vorhanden sind.[45]

Codeblock 3.3: Verwendung der HardwareSerial Bibliothek: Die wichtigsten Methoden.

```
1  void setup(){
2    Serial2.begin(115200, SERIAL_8N1, 4, 2, false);
3  }
4
5  void loop(){
6    Serial2.write(example_array[1]); // Serielle Ausgabe
7
8    if(Serial2.available()){
9      byte data = Serial2.read(); // Serielle Eingabe
10     // ...
11   }
12 }
```

Die Initialisierung der seriellen Schnittstellen erfolgt im `setup` mithilfe der Methode `Serial.begin`. Der Funktion können eine Reihe von Parametern übergeben werden, die die serielle Kommunikation konfigurieren. Der erste Parameter gibt

die Baudrate an und ist notwendig. Typische Werte sind hier etwa 9600, 19200 oder 115200. Alle weiteren Parameter sind optional, da Standardwerte definiert sind. Der zweite Parameter definiert die Anzahl der Datenbits, Stop Bits und das Paritätsbit. Vordefinierte Compiler-Direktiven[1] können hierbei zur Konfiguration verwendet werden. Eine Auswahl der insgesamt 24 Bezeichner ist nachfolgend zur Veranschaulichung aufgeführt.[45]

- `SERIAL_8N1`: Ein UART-Frame wird konfiguriert als 8 Datenbits, keinem Paritätsbit `(N)` und einem Stop Bit.
- `SERIAL_5E1`: Ein UART-Frame wird konfiguriert als 5 Datenbits, einem Paritätsbit im Modus *even* `(E)` und einem Stop Bit.
- `SERIAL_7O2`: Ein UART-Frame wird konfiguriert als 7 Datenbits, einem Paritätsbit im Modus *odd* `(O)` und zwei Stop Bits.

Den Datenbits ist in jedem Fall noch ein Start Bit vorgeschaltet. Es können nicht weniger als fünf, jedoch nicht mehr als acht Datenbits, und nur entweder ein oder zwei Stop Bits verwendet werden. Außerdem sind alle drei möglichen Verwendungen des Paritätsbits gezeigt.[45]

Die folgenden zwei Parameter geben je einen GPIO an, der zum Empfangen (RX) beziehungsweise zum Senden (TX) von Nachrichten verwendet wird. Auch hier sind standardmäßig GPIOs definiert, die in den Hardware Spezifikationen des verwendeten Boards zu finden sind. Der letzte Parameter ist ein Boolean, der auf `true` gesetzt werden kann, falls gewünscht wird, dass das Signal logisch invertiert wird.[45]

Eine serielle Ausgabe einer UART-Nachricht erfolgt mit der Methode `Serial.write`. Der Funktion kann ein Wert, eine Variable oder ein Zeiger übergeben werden. Im Falle des Zeigers muss ein zweiter Parameter übergeben werden, der die Länge

[1] Das sind Bezeichner, die per Befehl `#define` im Quellcode definiert, und vom Compiler durch Werte ersetzt werden.[46]

des zu schreibenden Objektes in Bytes angibt. `Serial.write` erzeugt die Ausgabe von UART-Frames auf dem TX-Pin.[45]

Zum Empfang serieller Nachrichten per UART können die Methoden `Serial.available` und `Serial.read` verwendet werden. `Serial.available` gibt einen Boolean zurück, sofern neue Daten im RX-Buffer des UART vorhanden sind und kann somit auch als Bedingung in einer `if`-Klammer verwendet werden. `Serial.read` liest ein Byte aus und gibt es als Rückgabewert zurück. Es ist außerdem eine Verwendung möglich, bei der ein Zeiger zu einem Datenobjekt und die Anzahl der zu lesenden Bytes übergeben wird. In diesem Fall gibt die Methode die Anzahl der gelesenen Bytes zurück.[45]

Anstelle von `Serial` können in den obigen Fällen natürlich auch die anderen drei UART-Schnittstellen zum Aufruf der genannten Methoden verwendet werden. Die Verwendung von *flow control* (siehe Abschnitt 2.5.3) ist seitens der HardwareSerial Bibliothek zwar möglich, wird an dieser Stelle jedoch nicht behandelt, da es für den praktischen Teil nicht relevant ist.

3.1.4 Interrupts

Interrupts wurden in Abschnitt 2.5.1 theoretisch beschrieben. Methoden zur Initialisierung von Interrupts, die auf Veränderungen der Spannungswerte an den GPIOs reagieren, sind in dem Arduino Core des ESP32 standardmäßig integriert. Ein Interrupt kann mit der Methode `void attachInterrupt(uint8_t pin, std::function <void(void)> intRoutine, int mode)` an eine Pin-Veränderung gekoppelt werden. Der `mode` ist dabei die Art der Veränderung am GPIO `pin`, bei der folgende Arten möglich sind:[47, 48]

- `GPIO_INTR_DISABLE`: Interrupt deaktivieren.
- `GPIO_INTR_POSEDGE`: Interrupt bei steigender Flanke.
- `GPIO_INTR_NEGEDGE`: Interrupt bei fallender Flanke.
- `GPIO_INTR_ANYEDGE`: Interrupt bei steigender und fallender Flanke.
- `GPIO_INTR_LOW_LEVEL`: Interrupt bei gemessenem LOW am Pin.
- `GPIO_INTR_HIGH_LEVEL`: Interrupt bei gemessenem HIGH am Pin.

Mit dem zweiten Übergabeparameter kann die ISR übergeben werden, die an einer anderen Stelle im Code definiert sein muss und bei einem Interrupt ausgeführt werden soll. Die ISR sollte mit dem zusätzlichen Parameter `IRAM_ATTR` definiert sein, der die Funktion im IRAM des ESP32 speichert, wodurch schneller auf sie zugegriffen werden kann[2]. Dies ist bei Interrupts entscheidend, weil die Codeausführung an anderer Stelle durch die ISR unterbrochen wird, und daher so schnell wie möglich wieder fortgesetzt werden sollte, um Fehler zu vermeiden. Die Verwendung des `IRAM_ATTR` ist in einem nachfolgenden Beispiel in Codeabschnitt 3.4 in Zeile 19 gezeigt.[47, 48, 49]

3.1.5 Timer

Timer wurden im Zusammenhang mit Mikrocontrollern bereits kurz in Abschnitt 2.5.2 behandelt. Der Arduino Core für den ESP32 enthält eine API zur Verwendung der Hardware Timer auf dem ESP32. Die meisten Ausführungen der ESP32-Chips enthalten vier Hardware Timer. Diese Timer sind standardmäßig an einen Taktgeber mit 80 MHz angeschlossen und können in Wertebereiche bis zu einer maximalen Bitbreite von 64 Bit zählen. Der Taktquelle wird ein Prescaler mit einer 16-Bit Wortbreite vorgeschaltet, der somit Werte bis zu 65.535, nicht jedoch weniger als 2 annehmen kann.[50, 51]

[2] Standardmäßig werden Funktionen im langsameren Flash-Speicher gespeichert.[47]

Codeblock 3.4: Grundlegende Handhabe eines Timers in der Timer API.

```
1  hw_timer_t * sampleTimer = NULL; // Deklaration
2  sampleTimer = timerBegin(2, 8, true); // Initialisierung
3
4  // ...
5
6  uint64_t ticks = timerRead(sampleTimer); // Zähler auslesen
7  timerWrite(sampleTimer, 0); // Wert des Zählers ändern
8
9  // ...
10 #define RATE 10000
11 timerAttachInterrupt(sampleTimer, isr, false); // ISR
    anhängen
12 timerAlarmWrite(sampleTimer, RATE, true); // Timer
    Interrupts auslösen
13 timerAlarmEnable(sampleTimer);  // Timer Interrupts
    aktivieren
14
15 // ...
16
17 timerEnd(sampleTimer);  // Timer deinitialisieren
18
19 void IRAM_ARTR isr(){
20   // Definition der Interrupt Service Routine ...
21 }
```

In Codeabschnitt 3.4 sind die wichtigsten Funktionen zur Handhabe eines Timers gezeigt. Das Timer-Objekt `sampleTimer` wird durch die Funktion `timerBegin` initialisiert. Der Funktion wird übergeben, welcher der Hardware Timer dem Objekt zugeordnet werden soll, welcher Prescaler verwendet werden soll und ob der Zähler hoch (`true`) oder runterzählen soll. In dem Beispiel aus Zeile 2 wird dem Objekt der dritte Hardware Timer mit einem Prescaler von 8 zugeordnet. Der Zähler zählt demnach alle 8 Takte um eins aufwärts, was etwa der Dauer von 100 ns entspricht.

Nach der Initialisierung beginnt der Timer zu zählen. Der Wert kann per `timerRead` ausgelesen werden. Es existieren auch Funktionen, die direkt eine Konvertierung in Mikrosekunden oder Millisekunden vornehmen, falls weniger simple Umrechnungen als in dem Beispiel nötig sind. Der Wert des Zählers kann mittels `timerWrite` verändert werden.[51, 52]

In Abschnitt 2.5.2 wurde bereits die wichtige Funktion beschrieben, die Timer in Zusammenhang mit Interrupts erfüllen. In den folgenden Zeilen ab Zeile 11 ist gezeigt, wie eine ISR an den Timer gebunden werden kann. Zunächst muss eine entsprechende Funktion im Quellcode definiert sein, was ab Zeile 19 angedeutet ist. Mit `timerAttachInterrupt` kann dann die Funktion `isr` als interrupt service routine bei dem Timer registriert werden. Der Methode wird noch ein Boolean übergeben, der keine Auswirkung auf die Funktionalität hat, weil zu dem Zeitpunkt des Schreibens nur die Übergabe von `false` unterstützt wird.[51, 52]

Des Weiteren müssen *Alarms* des Timers nach einem Ablauf einer definierten Zeit aktiviert werden. Die Zeitspanne wird definiert mit der Methode `timerAlarmWrite`, bei der der zweite Parameter die Anzahl der Ticks angibt, nachdem der Timer einen Alarm auslöst. Da zuvor eine ISR registriert wurde, wird dieser Alarm einen Interrupt auslösen. Die Übergabe von `true` lässt den Timer im Anschluss an den Alarm sofort neu laden, sodass er nach einem erneuten Ablaufen der Zeitdauer wieder auslöst. Zunächst muss der Alarm aber noch mittels `timerAlarmEneable`

aktiviert werden. Falls der Timer nicht mehr benötigt wird, kann er mit `timerEnd` beendet werden.[51]

3.2 Programmierung der Mikrocontroller

In diesem Kapitel wird die Programmierung der Mikrocontroller zum Versenden und Empfangen von Nachrichten mit LEDs und Photodiode erläutert. Es wurden zwei verschiedene Ansätze verfolgt, die verschiedene Hardware und verschiedene APIs auf dem ESP32 benutzen. Zunächst wird in Abschnitt 3.2.1 und in Abschnitt 3.2.2 ein Ansatz vorgestellt, der mit Interrupts und dem hochfrequentem Lesen und Schreiben der GPIOs eine Verbindung herstellt. Dieser Ansatz umgeht das UART-Bauteil auf dem ESP32 und ersetzt es durch softwareseitig implementierte Funktionen (*bit banging*). In Abschnitt 3.2.3 und Abschnitt 3.2.4 wird dagegen das UART-Bauteil miteinbezogen und die HardwareSerial-Bibliothek (siehe Abschnitt 3.1.3) verwendet.[53]

3.2.1 Manchester Encoder / Transmitter Bibliothek

Die in diesem und dem nächsten Abschnitt beschriebenen Bibliotheken wurden speziell im Rahmen dieser Bachelorarbeit erstellt und basieren lediglich auf den vom Hersteller Espressif zur Verfügung gestellten Funktionen. Es wurde somit – abgesehen von dem Betriebssystem FreeRTOS, das vom Hersteller vorinstalliert ist – keine Software von Dritten verwendet. Dadurch soll eine hohe Transparenz und Nachvollziehbarkeit angestrebt werden, welche die zukünftige Weiterentwicklung und Verbesserung der Programmbibliothek erleichtern sollen. Ohne eine ausführlichen Diskussion, die in Abschnitt 5 erfolgt, vorwegzunehmen, sei erwähnt, dass der Ansatz in diesem Abschnitt nicht mit der in Abschnitt 3.2.3 und Abschnitt 3.2.4 erreichten Datenrate konkurrieren kann. Er bietet jedoch einen allgemeineren Zugriff auf die GPIOs des Mikrocontrollers, wodurch potentiell die Implemen-

tierung anderer Codierverfahren möglich wird. Die Erklärungen in diesem und dem folgenden Abschnitt sollen einen nachvollziehbaren Einstieg in die Programmierung der Programmbibliotheken ermöglichen.

Die Grundidee der Manchester Sender-Bibliothek folgt einem *bit-banging*-Ansatz.[53] Bit-banging bezeichnet im Kontext der Programmierung von Mikrocontrollern das Emulieren einer Hardware-Schnittstelle durch das direkte Ansteuern der GPIOs. Stehen dem Prozess genügend Ressourcen für eine fehlerfreie Funktionalität zur Verfügung, kann auf diese Weise ein Hardwareteil durch Software ersetzt werden. Des Weiteren kann die Schnittstelle flexibel umprogrammiert und angepasst werden, ohne dass der Austausch eines Hardwarebausteins nötig ist. Neben einer hohen Flexibilität und der Möglichkeit einer ferngesteuerten Nachrüstung durch Updates können hierdurch in einem industriellen Kontext auch Kosten gespart werden.

Ein Überblick über die Bibliothek

Das Programm besteht, wie die meisten Arduino-Programme, aus einer Hauptdatei `main.cpp` und einer Header-Datei `manTx.h`. In der Hauptdatei kann der allgemeine Funktionsablauf in einem weiteren Kontext als die Datenübertragung programmiert werden. Zur Funktionalität der Übertragung muss lediglich die Methode `ManchesterTX::start` mit einem Zeiger zu den gewünschten Daten und die Anzahl der Bytes als Übergabeparameter aufgerufen werden. Das Attribut `ManchesterTX::started` ist global lesbar und bleibt `true`, solange eine Übertragung läuft. Die Funktionsweise des Senders ist in der Header-Datei definiert. Die Klasse `ManchesterTX` hält alle Methoden bereit, die für die Übertragung benötigt werden. In der Datei wird außerdem der für die Übertragung verwendete GPIO in der Compiler-Direktive `TX_PIN` und die Dauer eines Symbols in `CYCLE_LENGTH` definiert.

Manchester-Codierung

Die Methode `start` initialisiert einen Task mit der Funktion `sendData` und einen Timer mit Prescaler 80, der aufwärts zählt. Dem Timer wird einer ISR mit dem Namen `flipTransition()` zugeordnet, welche lediglich zwei simple Operationen durchführt: sie kehrt den Wert des Boolean `transition` um und benachrichtigt dann den Task zu `sendData`. Dieser prozessiert daraufhin den neuen Zustand von `transition`. Durch den Zusatz `IRAM_ATTR` wird die ISR in dem RAM des ESP32 gespeichert und ist dadurch deutlich schneller abrufbar als aus dem Flash-Speicher. Nachdem `start` aufgerufen wurde, wird die Funktion `sendData` zyklisch ausgeführt. Der in `start` übergebene Zeiger ist vom Typ `void *`, was bedeutet, dass ein Zeiger zu jedem möglichen Datentyp akzeptiert wird. Dadurch ist der Datentyp für die Übertragung unerheblich. Die Größe des Datentyps in Bytes muss im Parameter `len` übergeben werden. Um eine Synchronisation mit dem Empfänger zu erreichen, muss das erste Byte in `data` den Wert `0b00000010` haben. Weitere Details zur Synchronisation folgen in Abschnitt 3.2.2.

`sendData` wird nach der Initialisierung einiger Variablen durch die Funktion `xTaskNotifyWait` in einen blockierenden Zustand versetzt, bis eine Veränderung von `transition` durch die ISR festgestellt wird. Die Methode speichert den Zustand von `transition` seit der letzten Ausführung ab und ruft die Methode `sendOne` oder `sendZero` abhängig vom momentan übertragenen Bit in `data` auf. Die beiden Methoden ändern jeweils den Zustand von `TX_PIN`, wobei je nach Zustand von `transition` die Spannung auf `HIGH` oder `LOW` gesetzt wird. Hierdurch wird auch ersichtlich, dass ein Datenbit die zweifache Länge von `CYCLE_LENGTH` benötigt, um übertragen zu werden. Nachdem in `sendData` detektiert wurde, dass sich der Wert von `transition` von `true` wieder auf `false` geändert hat, erfährt `data` einen Bitshift, sodass das nächste Bit übertragen wird. Nach dem achten Bit wird die Position

im `data`-Array inkrementiert und das nächste Byte übertragen. Schließlich endet die Übertragung nach dem `len` Bytes übertragen wurden.

Im Kopf der Funktion wird durch `syncedTransition` bei Beginn einer Übertragung abgewartet, dass `transition` wieder den Wert `false` bekommt. Dadurch wird sichergestellt, dass eine Übertragung nicht zufällig in dem Zeitraum zwischen zwei Umschaltungen von `transition` begonnen wird.

3.2.2 Manchester Decoder / Receiver Bibliothek

Auch das Programm der Empfängerseite besteht, wie schon die Sendeseite, aus einer Hauptdatei `main.cpp` und einer Header-Datei `manRX.h`. Die Hauptdatei ruft die Methode `ManchesterRX::start()` in der `setup`-Funktion auf, wodurch dem in der Compiler-Direktive `RX_PIN` definierten GPIO ein Interrupt bei einem Wechsel des Spannungspegels zugeordnet wird. Danach wird ein Task erstellt, der die Funktion `syncCheck` in einer Schleife ausführt. Ein zweiter Task für die Funktion `rxRoutine` wird erstellt, der ab dem Absatz „Startsequenz“ erklärt wird. Ein Timer `sampleTimer` wird ebenfalls initialisiert, aber erst zu einem späteren Zeitpunkt gestartet. Anders als dem Sender, muss dem Empfänger keine Taktrate definiert werden, weil diese in einem Synchronisationsprozess beim Start der Übertragung bestimmt wird.

Die ISR „onRXPinChange“

Die Methode `onRXPinChange` dient zur Protokollierung der Veränderungen des Spannungspegels am GPIO `RX_PIN`. Eine Timer-Instanz `cycleTimer` wird dazu genutzt, die vergangene Zeit seit der letzten Pin-Veränderung zu bestimmen. Die erfassten Werte werden in das `timerArray` geschrieben. Da zur Synchronisation nur 11 Werte benötigt werden (Erklärung folgt unter der Überschrift „Startsequenz“), wird

ein Fehler ausgegeben, falls mehr als 11 Werte in das `timerArray` geschrieben werden. Dies ist denkbar in einem Szenario, bei dem die ISR so schnell hintereinander aufgerufen wird, dass das `timerArray` zwischenzeitlich nicht auf einen Sendetakt analysiert werden konnte. Die Analyse des Arrays findet in dem zuvor gestarteten Task `syncCheck` statt.

Synchronisation

Der in `start` initiierte Task führt periodisch die Funktion `syncCheck` aus, die das `timerArray` auf ein Synchronisationssignal überprüft. Der Boolean `newValue` wird durch `onRXPinChange` auf `true` gesetzt, sobald ein neuer Wert in das `timerArray` geschrieben wurde. Nachdem festgestellt wurde, dass das `timerArray` mit elf Werten gefüllt ist, beginnt die Analyse des `timerArray` in der Funktion `checkArrayForClockSignal`. Die Funktion bestimmt den größten und den kleinsten Wert der Dauer zwischen zwei Pin-Veränderungen und gibt `true` zurück, falls die Differenz weniger als 800 Taktschritte (gemeint ist der Takt des Timers cycleTimer) ist. Dabei ist 800 ein Wert, der sich als funktional erwiesen hat, aber angepasst werden könnte, um strenger oder nachsichtiger bei der Bestimmung des Taktes zu sein.

Wenn keine größere Differenz als 800 Taktschritte festgestellt wird, gilt die Synchronisation als erfolgreich und die Taktrate wird mittels `calcClockRate` berechnet. Diese Funktion berechnet den Mittelwert der elf zuvor empfangenen Werte. Die von `syncCheck` verwendeten Variablen werden zurückgesetzt, der `cycleTimer` beendet und ein neuer Timer gestartet: Der in `start` initialisierte `sampleTimer` wird in Verbindung mit der ISR `bitsampling` dazu genutzt, während der Dauer eines Symbols mehrere Samples vom `RX_PIN` zu holen, aus denen später das Symbol bestimmt werden kann. Der Timer hat einen Prescaler von 80, wodurch er jede Mikrosekunde um eins inkrementiert wird. Der Timer wurde schon in `start` initialisiert, um an dieser Stelle während des Synchronisationsprozesses Zeit zu sparen.

Startsequenz

Die Routine `bitsampling` wird nun periodisch vom `sampleTimer` aufgerufen und schreibt den Wert von `TX_PIN` in ein Array namens `sampleArray`. Das Array fasst acht Samples und wird zyklisch beschrieben. Die Routine hat das `IRAM_ATTR`, um im RAM des ESP32 gespeichert zu werden und dadurch schneller abrufbar zu sein.

Die `rxRoutine` wird unmittelbar nach dem Starten in einen blockierenden Zustand versetzt und hat zunächst keine Funktion, bis die Blockierung durch die `bitsampling`-ISR aufgehoben wird. Dies geschieht jedes mal, sobald acht Samples in das `sampleArray` geschrieben wurden. Nach der Freigabe beginnt die `rxRoutine` vor dem eigentlichen Datenempfang damit, die Startsequenz `0b0110` zu empfangen, die nach den Synchronisationszeichen vom Sender übertragen wird. Die ganze Synchronisations- und Startsequenz hat den Wert `0b00000010`. Nachdem die Manchester-Codierung angewendet wurde, entspricht dies der Sequenz `0b10101010` `0b10100110` auf der Datenleitung. Der Sender überträgt also zwölf Flanken zwischen denen insgesamt elf Mal der gleiche Zeitabstand liegt. Der Zeitabstand entspricht etwa der mittels `calcClockRate` berechneten Signaltaktrate. Es verbleibt die Sequenz `0b0110`, die als Startsequenz definiert wurde. Nachdem die Startsequenz empfangen wurde, wird `rxStarted` auf `true` gesetzt und der Datenempfang beginnt.

Datenempfang

Die Funktion `getBitOffSpampleArray` verarbeitet das `sampleArray` und gibt `true` oder `false` zurück, je nachdem, ob mehr Bits im `LOW` oder im `HIGH`-Zustand gemessen wurden. Der Rückgabewert wird in der Funktion `intoDataline` in einem `datalineArray` gespeichert, das nur zwei Bits fasst. Sobald das zweite Bit beschrie-

ben wurde, gibt die Funktion den Wert `true` zurück, wodurch in `rxRoutine` die Funktion `datalineToRX` aufgerufen wird.

Diese Funktion führt die Decodierung der Manchester-Sequenzen aus. Abhängig von den Werten im `datalineArray` wird ein Buffer mit dem Namen `rxBuffer` mit Bits gefüllt. Ein Bitshift führt dazu, dass nacheinander 8 Bits in den `rxBuffer` geschrieben werden und das erhaltene Byte anschließend in einen weiteren Buffer `byteBuffer` überführt wird. Anschließend wird der `rxBuffer` zurückgesetzt und das nächste Byte kann empfangen werden. Es dürfen keine zwei Einsen oder zwei Nullen im `datalineArray` auftauchen, weil dies kein gültiges Symbol in der Manchester-Codierung ist. Falls vier Nullen hintereinander festgestellt wurden, wird die Übertragung als beendet angesehen und per Methode `terminateRX` beendet.

3.2.3 UART TX

Die Programme „UART TX“ und „UART RX“ verfolgen einen anderen Ansatz als die zuvor beschriebenen bit-banging-Programme. Es wird zur Übertragung am Sender und am Empfänger je eine der drei UART-Schnittstellen des ESP32 genutzt. Die Schnittstelle gibt die eingegebenen Daten per On-Off-Keying an dem zugewiesenen GPIO aus, der frei gewählt werden kann. Die Baudrate und der GPIO zum Senden werden in entsprechenden Compiler-Direktiven im Kopf der Datei definiert. Da eine unidirektionale Übertragung angestrebt wird, ist nur eine der beiden möglichen Datenleitungen in Benutzung. Für eine Datenübertragung mit Licht kann das Sendesignal direkt genutzt werden, um das Vorschaltgerät einer Lichtquelle zu betreiben.

Das Programm ist vollständig in der Hauptdatei `main.cpp` definiert, da aufgrund der Verwendung der HardwareSerial-API das Erstellen einer Klasse für die Übertragung nicht nötig ist. Der `loop` wird nicht benötigt, da alle Prozesse vollstän-

dig in anderen Tasks ablaufen. Es werden drei Tasks ausgeführt:

- „Acquire Data“ führt die Routine `T_pushData` aus, die regelmäßig Daten zur Übertragung bereitstellt. Dieser Task emuliert die Generierung von Daten etwa aus einem Stream oder nach einer Benutzereingabe.
- „Encoding Task“ führt `T_encode` aus, wodurch die zu übertragenen Daten nach dem Manchester-Verfahren codiert und in den Ring Buffer `txBuffer` geschrieben werden.
- „TX Task“ führt schließlich die Übertragung über die serielle Schnittstelle in der Funktion `T_tx` aus. Daten werden aus dem `txBuffer` gelesen und an die serielle Schnittstelle übergeben.

Diese Tasks bilden den vollständigen Prozess von der Eingabe der Daten, über die Codierung bis zum Versenden ab. Zur Simulation der Schnittstelle können Daten in einer Variable definiert werden, die in `T_pushData` in die `dataQueue` übergeben wird. In einem realen Anwendungsszenario würden diese Daten in einem separaten Prozess generiert und an dieser Stelle in die Queue übergeben werden.

3.2.4 UART RX

Das Programm zum Empfangen über die serielle Schnittstelle führt im wesentlichen zwei Tasks aus:

- „RX Task“ führt die Funktion `T_rx` aus, die kontinuierlich die serielle Schnittstelle ausliest und ankommende Bytes in einen Ring Buffer `rxBuffer` speichert.
- „Decoding Task“ führt die Funktion `T_decode` aus, die die Daten aus dem `rxBuffer` in einen weiteren Zwischenspeicher `encodedByte` schreibt. Der Zwischenspeicher ist vom Typ `uint16_t` und fasst somit zwei Bytes aus dem `rxBuffer`. Aus zwei Bytes kann somit ein decodiertes Byte generiert werden.

Die decodierten Bytes werden abschließend auf einem seriellen Monitor ausgegeben. In einem realen Anwendungsfall könnten die Daten, die in der Variable `decodedByte` erhalten werden, für die weiteren Prozesse genutzt werden.

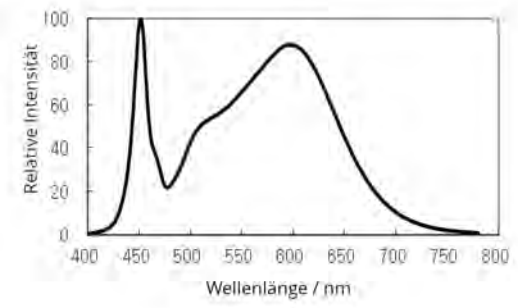
Anders als bei der beschriebenen Empfängerseite in Abschnitt 3.2.2 muss diesem Empfänger beim Start des Programms eine Baudrate übergeben werden, die mit der Sendeseite übereinstimmen muss. Darüber hinaus muss die UART-spezifische Übertragung definiert sein: Die Anzahl der Start- und Stop-Bits, sowie das Paritätsbit müssen im `setup` in der Methode `Serial2.begin` definiert werden und mit dem Empfänger übereinstimmen. Zudem können die Spannungswerte auf der Datenleitung invertiert werden. In Verbindung mit dem vorgeschlagenen Sendegerät muss die Sendeseite gegenüber dem Empfänger invertiert sein.

3.3 Verwendete Elektronik

Die elektronischen Bauteile am Sender und am Empfänger haben die Aufgabe, die hochfrequenten Signale möglichst unverändert und mit den geeigneten Strom- und Spannungswerten weiterzuleiten. Die angestrebten Signalfrequenzen befinden sich im Megahertzbereich, sodass das Hochfrequenzverhalten der Bauteile bei dem Entwurf der Platinen berücksichtigt werden muss. Im Folgenden werden die verwendeten Bauteile beschrieben und ihre Auswahl begründet.

3.3.1 Sender Hardware

Der ESP32 ist selbstständig nicht dazu in der Lage, LEDs mit hohen Leistungen und Spannungen über 3,3 V zu betreiben. Die GPIOs können kumuliert bis zu 1200 mA abgeben, pro Pin darf jedoch ein Strom von 40 mA nicht überschritten werden.[54] Die Bauteile des Senders haben deshalb zur Aufgabe, das vom ESP32 generierte Si-



(a)

(b)

Abbildung 3.2: (a) Relative Lichtintensität in Abhängigkeit von der Wellenlänge λ des Lichts der LED SM301B. Entnommen und nachbearbeitet aus [16].
(b) Produktbild der Platine LT-3182 der Firma LED-TECH.DE. Abgerufen von [55].

gnal auf geeignete Spannungswerte zu transformieren und genügend elektrische Leistung zur Verfügung zu stellen. Dabei soll das Signal möglichst unverformt bleiben, sodass ein unverzerrtes Signal von dem Leuchtmittel gesendet wird.

Leuchtmittel

Als Leuchtmittel eignen sich LEDs durch ihre schnelle Reaktionszeit, die Größenordnungen von 1 - 1000 ns annimmt. Aus Vorgängerarbeiten wurden die LEDs vom Typ LM301B übernommen, die sich durch eine hohe Leuchtkraft von 38 – 40 lm und eine gute Farbwiedergabe auszeichnen.[56] Der Farbwiedergabeindex (*color rendering index*, CRI), der ein Maß für die Farbtreue eines Leuchtmittels angibt, beträgt für die verwendete Ausführung der LM301B CRI $= 80$. Maximal kann ein Wert von 100 erreicht werden.[57, 58] Die LEDs können durchgängig mit einem Flussstrom von 200 mA betrieben werden und tolerieren Spannungen bis zu 2,9 V. Die Betriebsspannung liegt bei 2,7 V. Für den Versuchsaufbau wurde die Platine LT-3182 der Firma „LED-TECH.DE optoelectronics GmbH“ verwendet, auf der sieben SM301B-LEDs engmaschig angeordnet sind. Ein Vorwiderstand von 10 Ω wurde zur Strombegrenzung verwendet.

Hochfrequenzschalter

Zur Schaltung der Leuchtdioden ist ein Bauteil nötig, das zum einen die geforderten Ströme und Spannungen verträgt und zum anderen ausreichend schnell reagiert, um die Form des Signals nicht zu verfälschen. Für diesen Zweck wurde der Hochfrequenzschalter (HF-Schalter) SN75454B verwendet, der eine Last von $50\,\Omega$ bei einem Ausgangsstrom von $I_\mathrm{O} = 200\,\mathrm{mA}$ innerhalb von 8 ns von Low auf High, beziehungsweise innerhalb von 12 ns von High auf Low umschalten kann. Der Schalter wird mit einer 5 V-Spannungsversorgung betrieben und kann Ströme bis zu 400 mA führen. An seinen Eingängen werden Spannungen unterhalb von 0,8 V als Low und oberhalb von 2,0 V als High interpretiert, sodass er mit einem Ausgang des ESP32 kompatibel ist. Der Schalter hat eine NOR-Logik, wodurch das Eingangssignal am Ausgang invertiert wird.

3.3.2 Schaltbild des Senders

Das in Abbildung 3.3 gezeigte Schaltbild zeigt die Elektronik des Senders. Der 5 V-Pin des ESP32 Entwicklerboards wird dazu verwendet, die Spannungsversorgung des Schalters SN75454B bereitzustellen. Der Schalter benötigt einen Eingangsstrom von maximal 80 mA, der die Spannungsversorgung des ESP32 zusätzlich belastet. Der GPIO des ESP32, der das Datensignal ausgibt, ist an den Eingang des HF-Schalters angebunden. Der Schalter stellt zwei Eingänge zur Verfügung, von denen nur einer benutzt wird. Am Ausgang des Schalters ist die LED-Platine mit einem niederohmigen Vorwiderstand angebunden. Die Spannungsversorgung des HF-Schalters wird durch eine Kapazität nahe am Schalter stabilisiert. Außerdem werden Spannungsspitzen beim An- und Abschalten der LEDs durch einen Kondensator an der Spannungsversorgung der LEDs V_S abgeschwächt.

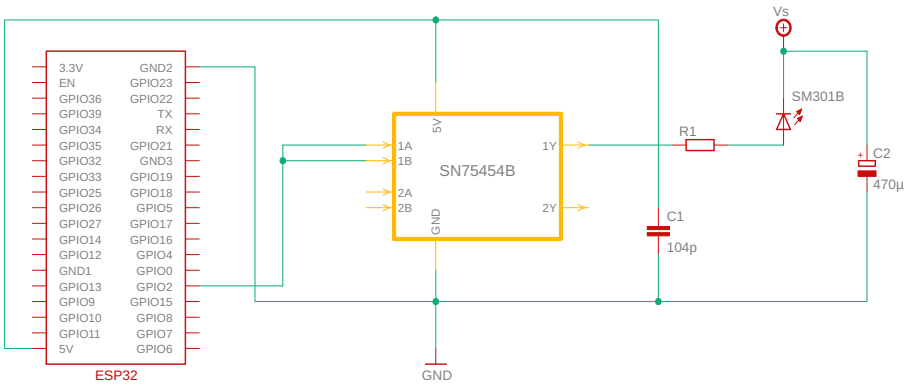


Abbildung 3.3: Schaltplan des Senders.

Bei der gezeigten Schaltung wird durch die Anordnung des Schalters nicht etwa die Versorgungsspannung V_S, sondern die Masse geschaltet. Die Ausgabe eines High-Pegels am GPIO des ESP32, was einem Spannungswert von 3,3 V entspricht, wird durch die NOR-Logik des HF-Schalters in ein Low gewandelt. Der Schalter wird dadurch hochohmig, sodass kein Strom mehr durch die LEDs fließen kann. Der Ausgang des HF-Schalters kann in diesem Zustand als Leerlauf betrachtet werden: Die LEDs leuchten nicht. Durch einen Low-Pegel des Mikrocontrollers schaltet der HF-Schalter die Masse durch und V_S kann vollständig über die LEDs und den Vorwiderstand abfallen. Es fließt ein Strom und die LEDs leuchten auf.

3.3.3 Empfänger

Die Empfängerhardware hat zur Aufgabe, das einfallende Lichtsignal in eine Spannung zu wandeln und diese in den Arbeitsbereich des Mikrocontrollers zu konvertieren. Da das Signal im Laufe der Signalkette verzerrt wird, muss es nach der Wandlung in elektrische Spannung verstärkt und verformt werden. Im Wesent-

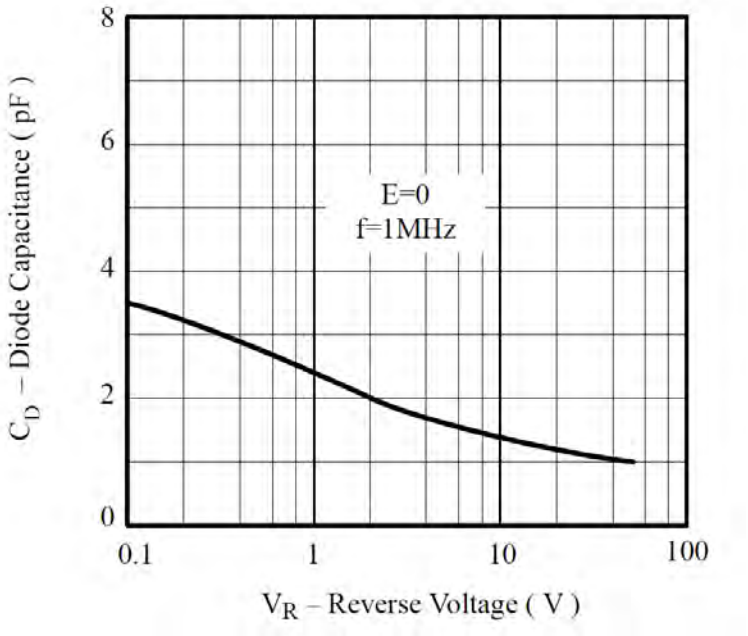


Abbildung 3.4: Auftragung von Messungen an der Photodiode BPW43 zur Sperrschichtkapazität C_S in Abhängigkeit zur Sperrspannung V_R aus dem Datenblatt [59].

lichen sind dazu neben Widerständen und Kondensatoren eine Photodiode und Operationsverstärker nötig. Die verwendeten Bauteile des Empfängers werden im Folgenden erklärt.

Photodiode

Die eingehenden Lichtimpulse werden durch eine Photodiode in eine elektrische Spannung gewandelt. Diese Photodiode muss dazu in der Lage sein, schnell genug auf das modulierte Lichtsignal zu reagieren und entsprechende elektrische Pulse auszugeben. Wie in Abschnitt 2.4.2 beschrieben wurde, haben Photodioden endliche Reaktionszeiten, die vom Aufbau der Diode abhängen. Zusätzlich wird die Auf- beziehungsweise Entladung der Sperrschichtkapazität, die maßgeblich zur

Zeitkonstante der Diode beiträgt, deutlich beschleunigt, wenn an die Diode eine Vorspannung in Sperrrichtung gelegt wird. Für die Empfängerschaltung wurde in dieser Arbeit die Photodiode mit der Bezeichnung BPW43 verwendet, die laut Datenblatt eine Anstiegszeit von 4 ns hat. Dabei wird die Anstiegszeit in der Literatur definiert als die Zeit, die die Photodiode benötigt, um bei einem eingehenden Lichtpuls die Ausgangsspannung von 10 % auf 90 % des maximalen Wertes ansteigen zu lassen.[60] Unter der Bedingung, dass 90 % der Spannung für mindestens die doppelte Dauer der Anstiegszeit ausgegeben werden muss, ergibt sich damit eine theoretische Grenzfrequenz von

$$f_{\mathrm{g}} = \frac{1}{3 \cdot 4\,\mathrm{ns}} = 83,3\,\mathrm{MHz}. \tag{3.1}$$

Die Anstiegszeit ist jedoch an weitere Größen gekoppelt: Die angegebene Zeit von 4 ns wurde eingebunden in eine Schaltung mit einem Lastwiderstand von $50\,\Omega$ und einer Sperrspannung von 10 V erreicht. Da sich die Sperrschichtkapazität C_{S} über dem Lastwiderstand R_{L} entlädt, ist dieser entscheidend für die Zeitkonstante τ, die sich aus dem Produkt von R_{L} und C_{S} ergibt und ein Maß für die Dauer der Auf- beziehungsweise Entladung ist. Gleichzeitig hat auch die Sperrspannung V_{R} einen Einfluss auf die Zeitkonstante, da sich die C_{S} mit der Erhöhung von V_{R} verringert. Messungen von C_{S} in Abhängigkeit von V_{R} aus dem Datenblatt der Photodiode BPW43 sind in Abbildung 3.4 aufgetragen. In Tabelle 3.1 sind entscheidende Kennwerte der Photodiode zusammengestellt. Dort ist auch zu sehen, dass die Photodiode breitbandig im sichtbaren Spektrum und auch im nahen Infrarotspektrum absorbiert.

Tabelle 3.1: Kennwerte der Photodiode BPW43 aus dem Datenblatt [59]. E_A bezeichnet die Beleuchtungsstärke während der Messung der Leerlaufspannung, λ die Wellenlänge des eingestrahlten Lichts.

Größe	Testbedingung	Symbol	Wert	Einheit
Maximale Sperrspannung		V_R	32	V
Dunkelstrom in Sperrrichtung	$V_R = 10\,V$, ohne Beleuchtung	I_{ro}	1	nA
Leerlaufspannung	$E_A = 1\,klx$	V_O	320	mV
Anstiegszeit	$V_R = 10\,V$, $R_L = 50\,\Omega$, $\lambda = 820\,nm$	t_r	4	ns
Kapazität der Diode	$V_R = 10\,V$, $f = 1\,MHz$, ohne Beleuchtung	C_D	1,3	pF
Spektrale Bandbreite		$\lambda_{0.5}$	550 ...1000	nm

Operationsverstärker

Das Signal der Photodiode muss ausreichend verstärkt werden, um für die weitere Signalkette verwendbar zu sein. Bei der Auswahl des Verstärkers muss beachtet werden, dass eine gute Verstärkung auch bei hohen Frequenzen über 1 MHz erreicht wird. Wird die Verstärkung eines Rechtecksignals mit einer Frequenz von 1 MHz angestrebt, so muss beachtet werden, dass die Fourier-Analyse des Signals spektrale Anteile bei ungeradzahligen Vielfachen der Grundfrequenz aufzeigt. Um den Rechteckcharakter des Signals weitgehend zu erhalten, muss die Bandbreite des Verstärker somit 1 MHz noch weit übersteigen. Da die Photodiode selbst jedoch eine gewisse Anstiegszeit benötigt, kann eine perfekte Rechteckschwingung nicht erreicht werden. Die Charakteristik der Rechteckschwingung wird dagegen in einem nachgeschalteten Bauteil in der Signalkette wiederhergestellt. In dieser Arbeit wurde der Operationsverstärker AD8055 verwendet, der eine -3 dB-

Tabelle 3.2: Kennwerte des Operationsverstärkers AD8055 aus dem Datenblatt [61]. G bezeichnet den Verstärkungsfaktor, V_e die Eingangsspannung(en).

Größe	Testbedingung	Wert	Einheit
-3 dB-Bandbreite	$G = 2$	150	MHz
Anstiegs- und	$V_\text{e} = 4\,V$ Schritt,		
Abfallzeit	$G = 2$	4	ns
Versorgungsspannung		± 4 ... ± 6	V
Ausgangsstrom max.	$V_\text{e} = \pm 2\,V$	60	mA
Ausgangsspannung max.		$\pm 2.9 \cdots \pm 3.1$	V

Grenzfrequenz von 150 MHz bei einem Verstärkungsfaktor von zwei aufweist. Weitere Kennwerte sind in Tabelle 3.2 aufgezeigt.

3.3.4 Schaltbild des Empfängers

In Abbildung 3.5 ist der elektronische Schaltplan des Empfängers gezeigt. Die Photodiode $D1$ befindet sich vor dem Operationsverstärker $IC1$ und ist mit einer Vorspannung von -5 V an der Anode angeschlossen, sodass die Photodiode dauerhaft sperrt. Wie in Abschnitt 2.4.2 erklärt, wird dadurch die Reaktionszeit der Diode verkürzt. Der Operationsverstärker $IC1$ wird als invertierender Transimpedanzverstärker betrieben und hat einen Rückkopplungswiderstand $R1 = 499\,\Omega$ zwischen dem Ausgang und dem invertierenden Eingang. Der Transimpedanzverstärker wandelt den von der Photodiode erzeugten Eingangsstrom in eine proportionale Ausgangsspannung um. Mit den Versorgungsspannungen von -5 V beziehungsweise +5 V wird ein ausreichender Arbeitsbereich des Verstärkers sichergestellt. Die Versorgungsspannungen sind über die Kondensatoren $C1$ und $C2$ gegen Störeinflüsse geschützt.

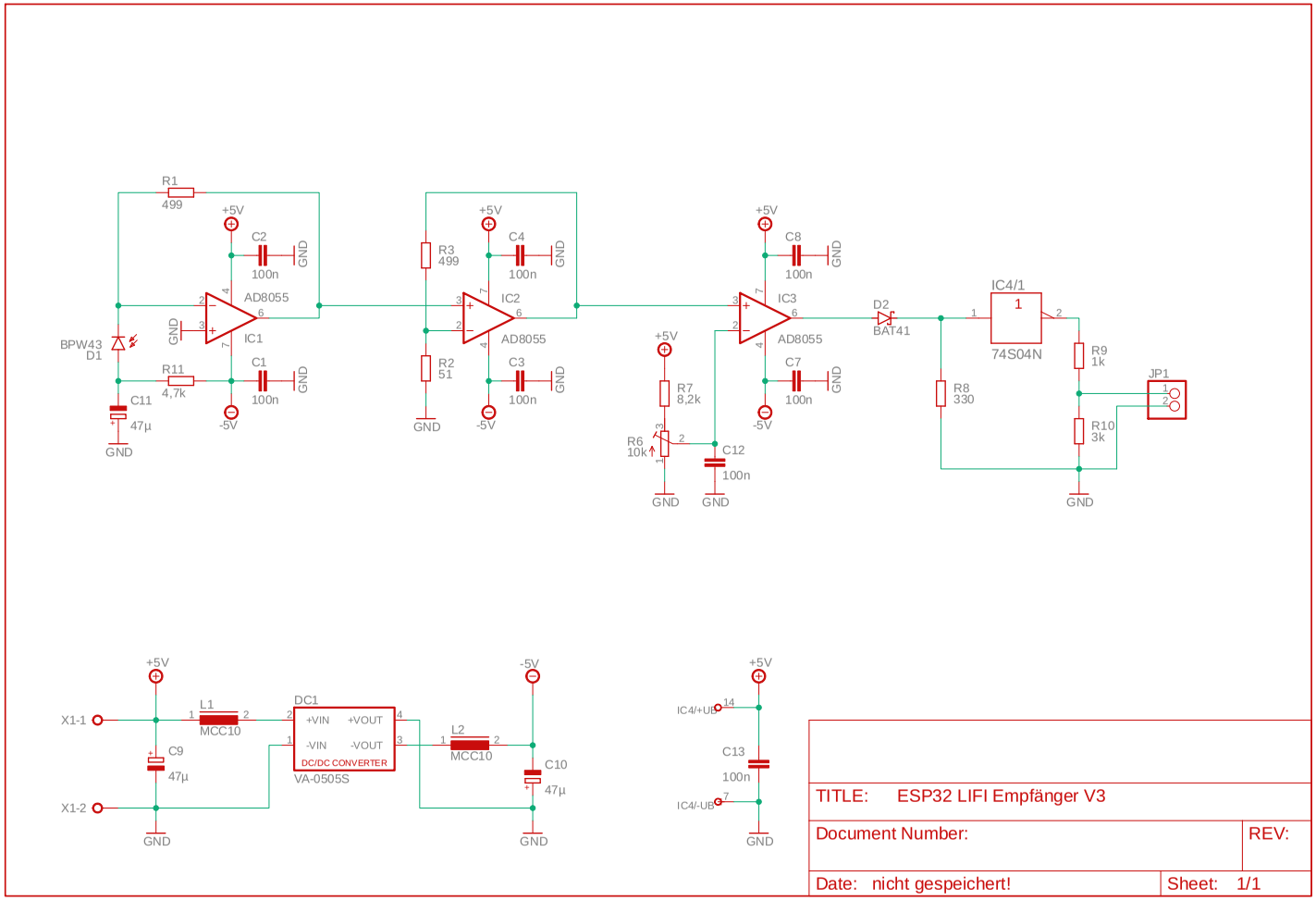


Abbildung 3.5: Schaltplan des Empfängers.

Der Ausgang des ersten Verstärkers ist direkt an den Eingang der zweiten Verstärkerstufe angeschlossen, der das vorverstärkte Signal nun auf noch höhere Pegel verstärken soll, die sich für den nachgeschalteten Komparator eignen. Der zweite Verstärker wird als nicht-invertierender Verstärker betrieben und hat ebenfalls einen Rückkopplungswiderstand von $R2 = 499\,\Omega$. Die Widerstandswerte von $R3$ und $R2$ bilden den Verstärkungsfaktor G (von dem Englischen *gain*), für den der folgende Zusammenhang gilt:[62]

$$G = 1 + \frac{R3}{R2} = \frac{499\,\Omega}{51\,\Omega} = 10,78 \tag{3.2}$$

Wie schon die erste Verstärkerstufe, wird auch dieser Verstärker mit $\pm 5\,\mathrm{V}$ versorgt, die durch zusätzliche Kondensatoren stabilisiert werden.

Der dritte Operationsverstärker, der ebenfalls vom Typ AD8055 ist, wird als Komparator verwendet. Der Komparator gibt die vollständige positive Sättigungsspannung $V_{\mathrm{S}+}$ des Operationsverstärkers aus, sobald der nicht-invertierende Eingang auf einem höheren elektrischen Potential liegt als der invertierende Eingang. Sinkt das Potential am nicht-invertierenden Eingang dagegen unter das Potential des invertierenden Eingangs, gibt der Komparator die negative Sättigungsspannung $V_{\mathrm{S}-}$ aus. Zwischen den beiden Spannungen $V_{\mathrm{S}+}$ und $V_{\mathrm{S}-}$ stellt sich kein stabiler Spannungspegel ein, sodass der Ausgang stets – innerhalb der durch die Anstiegszeit gegebenen Grenzen – eine der beiden Spannungen ausgibt. Da der Operationsverstärker bipolar mit den Versorgungsspannungen $V^+ = 5\,\mathrm{V} = -V^-$ betrieben wird, beträgt $V_{\mathrm{S}+} = -V_{\mathrm{S}-} \approx 3\,V$. Im weiteren Signalweg wird allerdings nur der positive Anteil des Signals weiterverwendet. Die Schwelle des Komparators wird am invertierenden Eingang bestimmt: Der Mittelabgriff eines Potentiometers, das an eine 5 V-Spannungsversorgung angeschlossen ist, dient zum Einstellen der Spannung am invertierenden Eingang. Somit kann die Schwelle an das Spannungsniveau am Ausgang der zweiten Verstärkerstufe angepasst werden. Um Signalfluk-

tuationen der Schwellspannung zu vermeiden, wird auch diese durch einen Kondensator stabilisiert.

Der Komparatorstufe ist eine BAT41 Schottky-Diode nachgeschaltet, die negative Signalanteile entfernt. Die Diode sperrt bei einem sprunghaften Übergang von einem Strom in Durchlassrichtung zu einem Sperrstrom innerhalb von 5 ns und ist deshalb für die angestrebten Datenraten geeignet. Hinter der Diode befindet sich ein Hex-Invertierer mit sechs unabhängig ansteuerbaren Signalinvertierern, von denen jedoch nur einer benötigt wird. Der Hex-Invertierer ist vom Typ SN74S04N und schaltet seinen Ausgang innerhalb von 4.5 ns von Low auf High, beziehungsweise innerhalb von 5 ns von High auf Low und ist deshalb für die Anwendung bei 1 MHz nutzbar. Hinter dem Ausgang des Invertierers folgt ein Spannungsteiler, der das Signal nach

$$U_{\mathsf{R10}} = \frac{\mathsf{R10}}{\mathsf{R9} + \mathsf{R10}} = \frac{3\,\mathrm{k\Omega}}{1\,\mathrm{k\Omega} + 3\,\mathrm{k\Omega}} = \frac{3}{4} \tag{3.3}$$

auf etwa Dreiviertel der Ausgangsspannung absenkt. Das Signal wird damit in den akzeptablen Bereich für den Eingang des Mikrocontrollers geregelt, der Spannungswerte bis zu 3,3 V toleriert.

3.4 Fokussierung der Lichtquelle

Zur Fokussierung der Lichtquelle auf den Lichtsensor wird eine Sammellinse vom Typ OM7 der Firma „Astromedia“ verwendet. Es handelt sich um eine plankonvexe Linse mit einem Durchmesser von 34,5 mm und einer Brennweite von 106 mm. Die Linse ist eingefasst in ein 100 mm langes Metallrohr, das den Einfluss von Umgebungslicht verringert. Das Gehäuse der Photodiode verleiht ihr eine Richtcharakteristik, die bei einem Einfallswinkel von $\pm 30°$ nur noch 20 % der relativen Em-

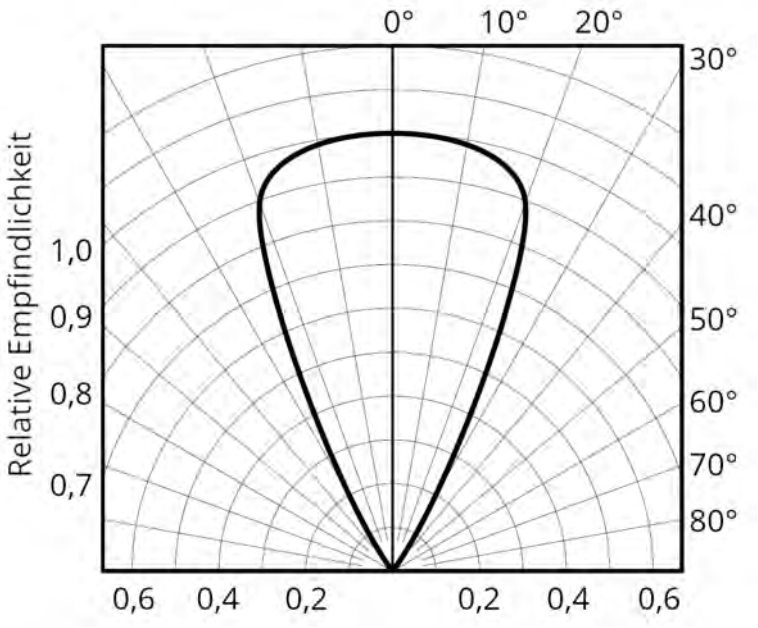


Abbildung 3.6: Richtcharakteristik des Gehäuses der Photodiode BPW43. Die relative Empfindlichkeit als Verhältnis zum Maximalwert ist in Abhängigkeit vom Einfallswinkel des Lichts gezeigt. Abbildung nachbearbeitet nach [16].

pflindlichkeit zulässt. Die Richtcharakteristik ist in Abbildung 3.6 gezeigt. Die Photodiode ist 1 cm hinter dem Austrittsloch im Mittelpunkt des Metallrohrs platziert, sodass der Einfluss des Umgebungslichtes zu vernachlässigen ist.

Sendeseitig wurden ebenfalls Maßnahmen getroffen, um möglichst viel Licht in die Richtung des Empfängers umzulenken. Eigens gestaltete computergestützte Modelle mehrerer Reflektorschirme, in denen die Platine LT-3182 integriert werden kann, wurden mittels additiver Fertigung hergestellt und mit einem breitbandig reflektierenden, hochglänzenden Lack der Firma „Dupli-Color“ behandelt. Dadurch konnten die hergestellten Reflektoren genutzt werden, um die breite Abstrahlcharakteristik der LED-Platine zu schmälern. Dieser Ansatz steht im Gegensatz zu einem in [56] vorgeschlagenen Ansatz, bei dem ein Anteil der Leuchtdioden auf einer Platine mit einer Seitenemitterlinse bestückt werden. Diese wird dazu

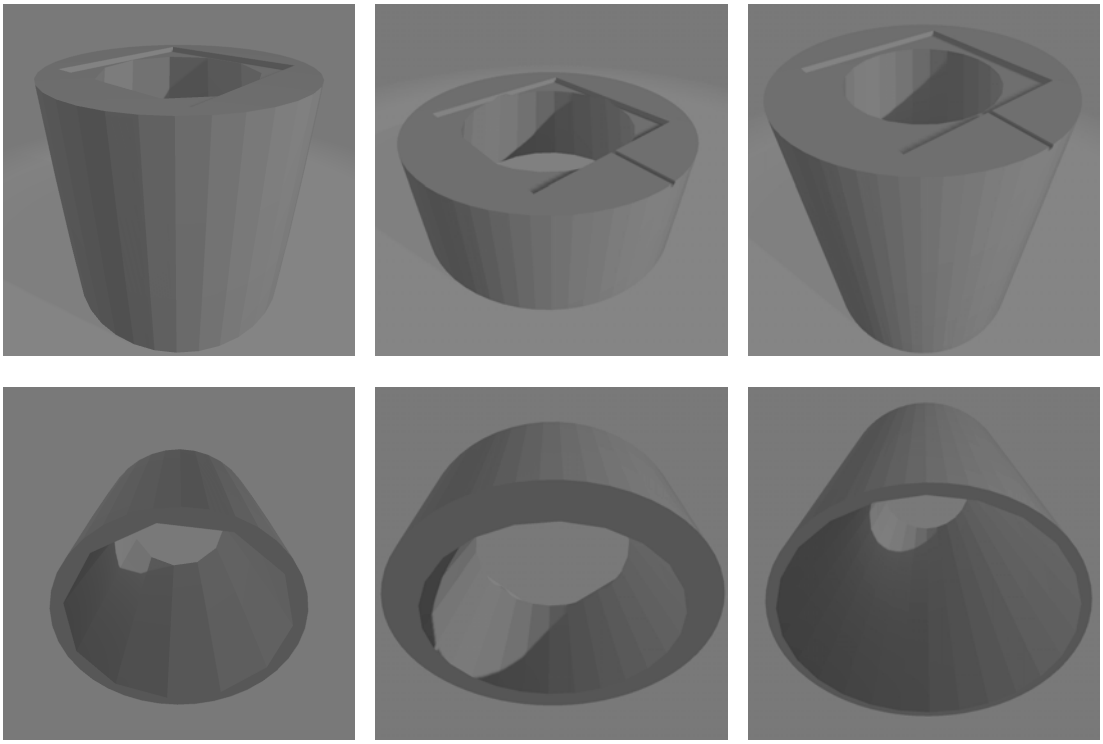


Abbildung 3.7: Draufsicht (obere Zeile) und Untersicht (untere Zeile) der Computermodelle der angefertigten Reflektoren. Die Reflektoren werden von links nach rechts als S1, S2 und S3 bezeichnet.

verwendet, die Hauptabstrahlrichtung stark zu verändern und auf den Sender zu richten, während die restlichen LEDs für die Funktion der Umgebungsbeleuchtung zur Verfügung stehen. Die Diskussion dieser Ansätze folgt in Kapitel 5. Die Computermodelle der Reflektoren sind in Abbildung 3.7 gezeigt.

Kapitel 4

Ergebnisse und Messungen

Die Arbeiten wurden zunächst mit den elektronischen Bauteilen der Vorgängerarbeit [56] begonnen. Dort konnten Datenraten bis zu 115200 Bit/s bei einer Entfernung von 4,1 m erzielt werden. Bis zu einem Abstand von 4,6 m waren noch Übertragungen mit einer hohen Fehlerrate möglich.

Der in [56] verwendete Photosensor iC-LQNP eignet sich gut für Signalfrequenzen zwischen 40 und 400 kHz, ist jedoch für die angestrebte Signalfrequenzen von über 1 MHz nicht passend. Die Bandpasscharakteristik des integrierten Verstärkers hat eine Mittenfrequenz bei 140 kHz und eine obere -3 dB-Eckfrequenz bei 400 kHz. Somit musste neben der Auswahl eines neuen Photosensors auch ein neues Konzept für die Verstärkung des Empfangssignals erstellt werden, da die Verstärkung bereits in den iC-LQNP integriert war. Auch die Elektronik des Sensors war für die angestrebten Datenraten nicht geeignet: Der in die sendeseitige Schaltung implementierte Metalloxid-Halbleiter-Feldeffekttransistor (MOSFET) L3705N, der als Schalter für die Leuchtdioden verwendet wurde, hat eine hohe Eingangskapazität von 3650 pF bei einer Frequenz von 1 MHz und einer Drain-Source-Spannung von 25 V. Bei einer geringeren Drain-Source-Spannung nimmt die Kapazität zudem weiter zu, was die Reaktionsfähigkeit weiter verschlechtert.[63] Messungen des MOSFETs mit Gegentaktverstärkung durch einen NPN- und einen PNP-

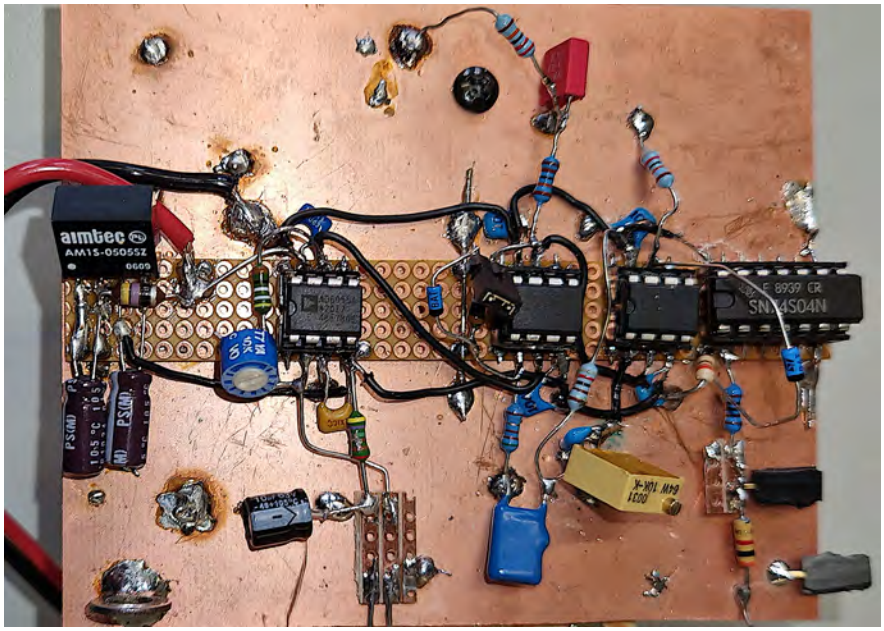


Abbildung 4.1: HF-Platine des Empfängers. Durch kurze Leiterbahnen konnten Störeinflüsse minimiert werden. Die Photodiode befindet sich mittig am unteren Bildrand, ihr Gehäuse ist jedoch auf dem Bild nicht zu sehen.

Transistor hatten in der Vorgängerarbeit [56] bei einer Rechteckschwingung mit einer Periodendauer von 4 µs bereits deutliche Signalverzerrungen gezeigt. Eigene Versuche bestätigten den Sachverhalt. Erste Versuche, im Labor vorhandene Transistoren als Schalter für die Leuchtdioden zu verwenden, erwiesen sich aufgrund derselben Problematik als wenig erfolgversprechend. Schließlich wurden hinreichend kurze Reaktionszeiten mit der integrierten Schaltung SN75454B erreicht.

Die ersten Aufbauten des Empfängers mit der Photodiode BPW43 wurden auf einer Steckplatine durchgeführt. Die Verwendung einer Steckplatine und von Dupont-Kabeln erwies sich als nicht zielführend: Parasitäre Kapazitäten und hochfrequente Störsignale führten dazu, dass der Operationsverstärker oszillierte und das Signal bis zur Unkenntlichkeit verzerrt wurde. Ein zweiter Aufbau auf einer

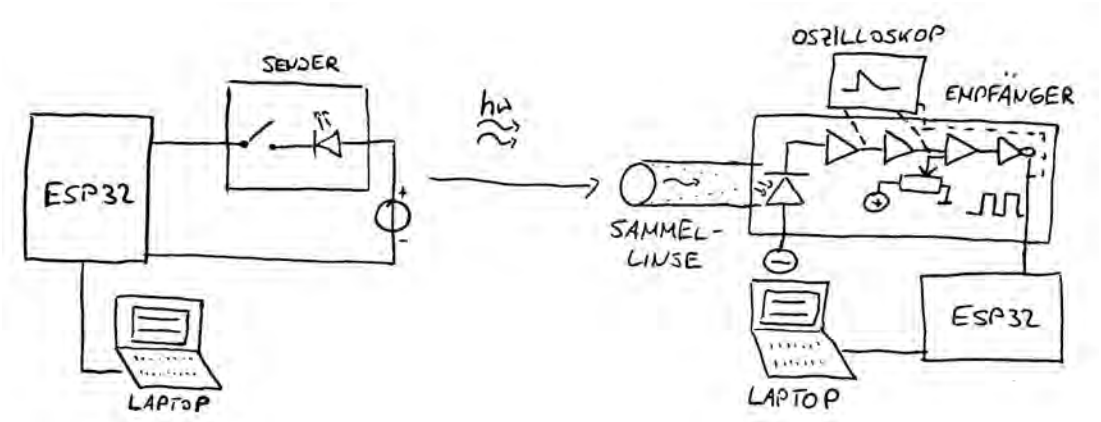


Abbildung 4.2: Skizze des Versuchsaufbaus mit vereinfachten Schaltbildern des Senders und des Empfängers. Die Messpunkte des Oszilloskops sind nach der ersten und der zweiten Verstärkerstufe sowie nach dem Invertierer gezeigt.

Lochrasterplatine führte zu einer deutlichen Verbesserung, jedoch waren immer noch deutliche Störsignale auf dem verstärkten Signal zu sehen. Schließlich erfolgte der Aufbau auf einer Hochfrequenz-Leiterplatte mit einer großflächigen Massefläche, die in Abbildung 4.1 gezeigt ist. Durch die kurzen Leiterbahnen und die große Massefläche konnten hiermit die besten Ergebnisse erzielt werden.

4.1 Versuchsaufbau

In Abbildung 4.2 ist der Versuchsaufbau einer Datenübertragung mit sichtbarem Licht und der vorgestellten Mikrocontroller-Programmierung skizziert. Die Schaltbilder von Sender und Empfänger sind vereinfacht dargestellt, um die wichtigsten Funktionen zu kennzeichnen und Messpunkte des Oszilloskops zu zeigen. Die Laptops wurden zur Programmierung der Mikrocontroller benutzt, etwa um die Datenrate zu variieren oder um die zu übermittelnde Nachricht festzulegen. Der Laptop am Empfänger wurde zudem als serieller Monitor zum Auslesen der eingehenden Daten verwendet. Als Testsequenz wurde die Zeichenfolge

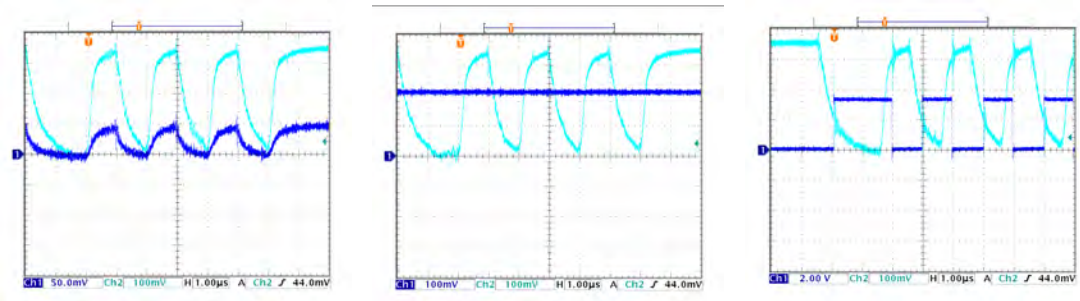


Abbildung 4.3: Oszilloskopische Messungen der Verstärkerstufen (links), der Schwellwertspannung (Mitte) und des Invertierer-Ausgangs (rechts).

```
0123456789\n0123456789\n0123456789\n
```

eingestellt, wobei das Zeichen `\n` einen Zeilenumbruch im seriellen Monitor bewirkt. Die restlichen Ziffern entsprechen ihrer Binärschreibweise im Oktett (beispielsweise 7 = 00000111 mit anschließender entsprechender Manchester-Codierung). Zur Übertragung wurden die eigens erstellten Programme UART TX und UART RX verwendet, da hiermit die größeren Datenraten erzielt werden konnten. Das Oszilloskop wurde zur Messung und Beurteilung des Eingangssignals am Empfänger hinter dem Ausgang der ersten Verstärkerstufe, der zweiten Verstärkerstufe, an der Schwellspannung des Komparators am invertierenden Eingang und am Ausgang des Invertierers angeschlossen. Der serielle Monitor diente zu Feststellung einer erfolgreichen Übertragung, wobei eine Übertragung als erfolgreich gewertet wurde, wenn die Sequenz in 9 von 10 Versuchen fehlerfrei übertragen wurde.

4.2 Anpassung der Schwellspannung

In Abbildung 4.3 sind exemplarisch Messpunkte an den zuvor beschriebenen Stellen in der Signalkette gezeigt. Im linken Bild ist die Verstärkung des Eingangssignals

durch die erste Verstärkerstufe zu sehen. Am Ausgang des Transimpedanzverstärkers wird ein Scheitelwert von 45 mV gemessen. Am Ausgang des zweiten Verstärkers beträgt der Scheitelwert 340 mV. Die in Gleichung 3.2 berechnete Verstärkung von $G = 10,78$ wird somit nicht erreicht. Ursachen hierfür könnten Ungenauigkeiten der Bauteileigenschaften und das reale Verhalten des Operationsverstärkers sein, das sich von dem in Gleichung 3.2 angenommenen idealen Verhalten unterscheidet. Die erreichte reale Verstärkung von $G_\mathrm{r} = 7,56$ ist für die weitere Signalkette jedoch ausreichend.

Das mittlere Bild zeigt die Ausgangsspannung der zweiten Verstärkerstufe und die Schwellwertspannung des Invertierers in einem Graphen. Die Skalierungsfaktoren der beiden Spannungen wurden bewusst gleich gewählt, um den Schwellwert mithilfe des Oszilloskops anhand der Eingangsspannung des Komparators wählen zu können. Da der zweite Operationsverstärker direkt an den Eingang des Komparators angebunden ist, entspricht die Ausgangsspannung des zweiten Verstärkers der Eingangsspannung des Komparators. Wie anhand von Abbildung 3.5 gezeigt wurde, kann das Potentiometer $R6$ dazu verwendet werden, den Schwellwert anzupassen und die Pulsweite der Rechteckschwingung am Ausgang des Komparators feinzujustieren. Dadurch kann die Komparatorstufe auf den Eingangspegel angepasst werden, der sich beispielsweise mit zunehmendem Abstand der Lichtquelle verringern könnte. In dem gezeigten Fall wurde eine Schwellspannung von 210 mV gewählt. Nach der Gleichrichtung durch die Diode $D2$ und der Invertierung durch $IC4$ wird das Signal in der rechten Abbildung von 4.3 gewonnen. Da in diesem Beispiel eine Baudrate von 1 MBaud gewählt wurde, haben die kurzen Taktschritte eine Dauer von 1 µs.

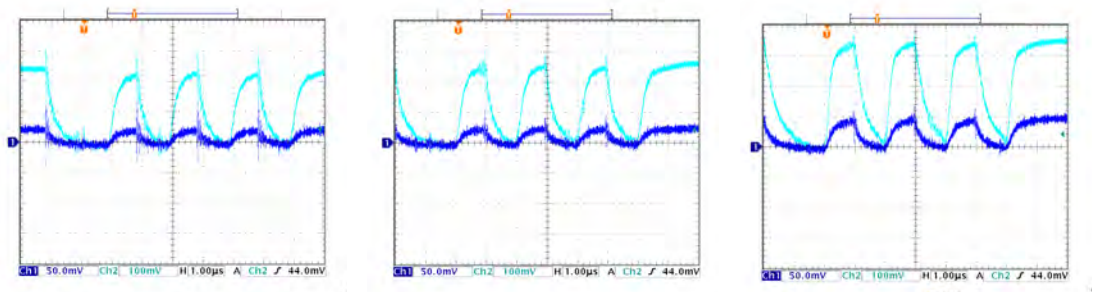


Abbildung 4.4: Ausgangsspannungen nach der ersten und der zweiten Verstärkerstufe mit unterschiedlichen Reflektoren. Links: Ohne Reflektor. Mitte: Reflektor S1. Rechts: Reflektor S3.

4.3 Verwendung der Reflektoren

Der Einfluss eines Reflektors auf eine mögliche Reichweitenverbesserung konnte aufgrund Zeitmangels leider nicht eingehend untersucht werden. Es wurde jedoch festgestellt, dass bei einer Übertragungsrate von 1 MBaud die größte Übertragungsdistanz von 5,4 m mit dem Reflektor S3 erreicht werden konnte, während eine Messung ohne Reflektor nur bis zu einer Distanz von 3 m möglich war. Der Vergleich zweier Messungen bei einer Entfernung von 2 m ergab zudem einen Spannungsgewinn von 7 dB nach der ersten Verstärkerstufe. Während ohne Reflektor ein Scheitelwert von 20 mV gemessen wurde, ergab die Messung mit dem Reflektor S3 einen Scheitelwert von 45 mV. Der Spannungsgewinn setzte sich entsprechend nach der zweiten Verstärkerstufe mit 230 mV gegen 340 mV fort. Der Reflektor S1 zeigte mit 25 mV nach der ersten und 250 mV nach der zweiten Verstärkerstufe einen niedrigeren Spannungsgewinn. Bei S2 wurde kein Spannungsgewinn festgestellt. Die Messungen sind in Abbildung 4.4 gezeigt.

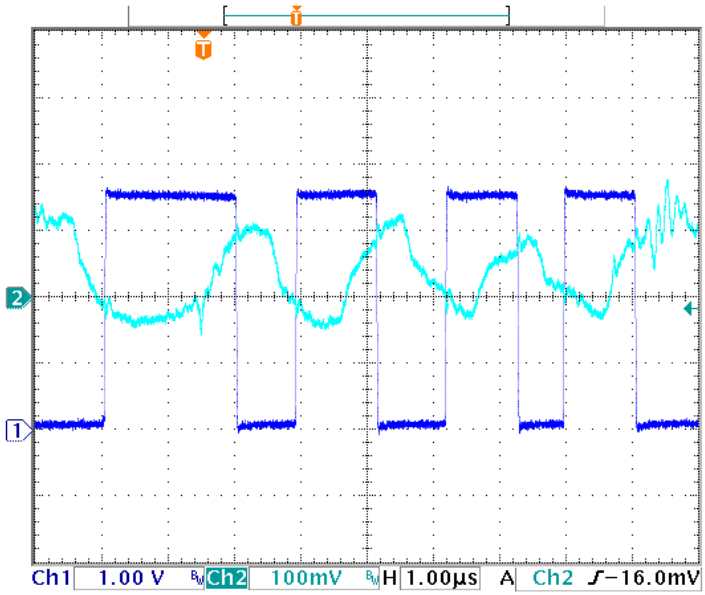


Abbildung 4.5: Aus einem stark deformierten Empfangssignal (türkis) kann durch die Verwendung des Komparators und des Invertierers auch bei einem Abstand von 5 m ein Rechtecksignal regeneriert werden. Das türkise Signal zeigt den Ausgang der zweiten Verstärkerstufe, das dunkelblaue Signal den Ausgang des Invertierers.

4.4 Untersuchung der Reichweite

Die oben genannte Übertragungssequenz wurde versuchsweise über verschiedene Distanzen gesendet und der Erfolg der Übertragung mit dem seriellen Monitor des Empfängers und einem Oszilloskop beurteilt. Die LED-Platine LT-3182 mit integriertem Vorwiderstand wurde mit 4,3 V versorgt. Die höchste Datenrate von 1,5 MBit/s, entsprechend einer Baudrate von 3 MBaud, konnte dabei in einem sehr engen Entfernungsbereich bei 1,5 m $\pm$0,3 m erzielt werden. Bei einer Einstellung von 2 MBaud war eine Übertragung bis zu 3 m Entfernung möglich. Eine Entfernung von 5,4 m wurde bei einer Baudrate von 1 MBaud erzielt. In Abbildung 4.5 ist gezeigt, wie ein stark deformiertes Eingangssignal bei einer Entfernung von 5 m

mittels Komparator und Invertierer noch immer dazu genutzt werden kann, ein verwertbares Rechtecksignal zu regenerieren.

4.5 Einfluss des Umgebungslichts

Der Einfluss des Umgebungslichts wurde während der Messungen aus Abschnitt 4.4 qualitativ durch das Ausschließen von Tageslicht mittels Jalousien und das An- und Abschalten der Deckenbeleuchtung des Labors untersucht. Der Einfluss des Tageslichts könnte sich als Gleichanteil auf den gemessenen Spannungen zeigen und den Arbeitspunkt des Transimpedanzverstärkers somit beeinflussen. Möglich wären zudem hochfrequente Interferenzen durch Vorschaltgeräte der Laborbeleuchtung. Die Spannungswerte wurden am Oszilloskop mit starker Umgebungsbeleuchtung und schwachem Restlicht bei verschiedenen Übertragungsdistanzen und Datenraten gemessen. Es konnte weder bei der Messung der Reichweite noch bei der Messung der Datenrate ein Einfluss des Umgebungslichts festgestellt werden. Auch die beobachteten Spannungswerte am Oszilloskop blieben durch die Veränderungen des Umgebungslichts unbeeinflusst.

Kapitel 5

Diskussion

In Abschnitt 3.4 wurde bereits die Verwendung der Reflektoren erklärt und eine Inkompatibilität mit dem in [56] besprochenen Ansatz der Verwendung einer Seitenemitterlinse festgestellt. Zu unterscheiden sind diese beiden Ansätze in einem Anwendungsszenario mit Außenbereichsleuchten, die in einem größeren Abstand zueinander stehen. Die Lichtkegel der Leuchten sind auf den Boden gerichtet, sodass nur ein sehr geringer Anteil bei der benachbarten Leuchte ankommt. Eine Seitenemiterlinse könnte dazu verwendet werden, einen Anteil des Lichts in Richtung des benachbarten Empfängers zu brechen. Dies ist in Abbildung 5.1 veranschaulicht. Die Verwendung einer Seitenemitterlinse hätte zum Vorteil, dass bereits bestehende Leuchtmittel um eine Linse und ein Vorschaltgerät ergänzt werden könnten und somit für die Datenübertragung tauglich wären – eine ausreichende Reaktionszeit der Leuchtmittel vorausgesetzt. Die Notwendigkeit eines zusätzlichen Leuchtmittels für die Datenübertragung, die bei einem dedizierten Überträger gegeben wäre, entfällt. Darüber hinaus würden zwei verschiedene Leuchtmittel eventuell als störend empfunden werden, wenn sie nicht dieselbe Farbtemperatur oder einen unterschiedlichen Farbwiedergabeindex hätten.

Für die Verwendung eines dedizierten Überträgers spricht jedoch, dass Umgebungslicht und Übertragungslicht völlig getrennt voneinander betrachtet wer-

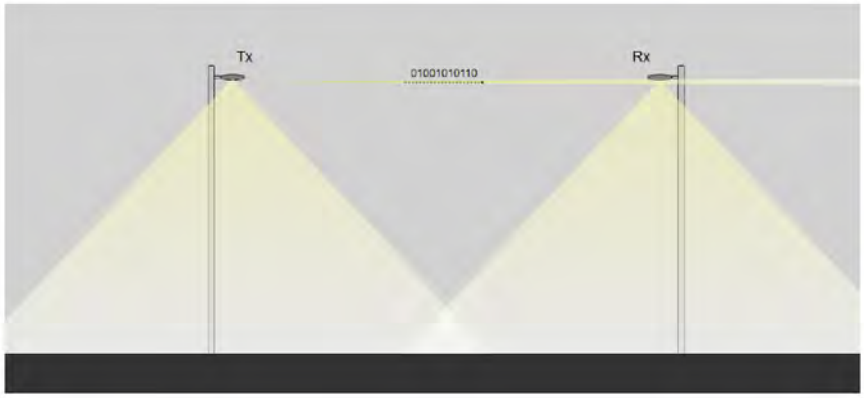


Abbildung 5.1: Veranschaulichung der Datenkommunikation zweier Außenbereichsleuchten, deren Lichtkegel auf den Boden gerichtet ist.[56]

den könnten. So wäre beispielsweise eine unabhängige Dimmung der Lichtquellen möglich. Darüber hinaus fällt die Fokussierung der Lichtquelle auf den Empfänger leichter, wenn der Sender ein unabhängiges Bauteil ist. Die plankonvexe Linse am Empfänger führt zwar zu einem starken Gewinn an Empfangsqualität und somit zu einer größeren Reichweite. Ab einer Entfernung von etwa 2,5 m wird die Lichtquelle jedoch so klein auf die Photodiode projiziert, dass die Fläche der Photodiode nicht mehr vollständig beleuchtet wird. Zudem führen kleinste Veränderungen der Geometrien, etwa durch ein Verschieben des Empfängers, dazu, dass das Lichtbündel nicht mehr auf die Photodiode fällt. Eine Linse hat eine starre Richtcharakteristik, die bei der Ausrichtung von Sender und Empfänger berücksichtigt werden muss, etwa durch eine verstellbare Vorrichtung für die Linse. Der Sender hat zwar ebenfalls eine starre Richtcharakteristik, kann jedoch als Ganzes bei der Ausrichtung auf den Empfänger eingestellt werden. Am schwersten wiegt jedoch, dass ein wesentlicher Anteil der Leuchtkraft bei der Brechung verloren geht und die Abstrahlung in einem breiten Winkel erfolgt. Dadurch könnte es schwerfallen, eine ausreichende Lichtintensität bei dem recht weit entfernten Empfänger zu generieren. Denkbar wäre jedoch für geringere Datenraten eine Bündelung des durch die Linse abgestrahlten Lichts mit einer präzisen Anordnung von

Reflektorspiegeln in Kombination mit einer sensiblen Photodetektion und einer ausreichenden Vorverstärkung am Empfänger. Um sowohl kleine als auch große Eingangssignale verwenden zu können, könnte insbesondere eine logarithmische Verstärkung zielführend sein. Versuche mit logarithmischen Verstärkern konnten während des Zeitraums dieser Arbeiten leider aufgrund mangelnder Verfügbarkeit der Bauteile nicht weiter untersucht werden.

5.1 Limitierung der Datenrate

Mit den Programmen UART TX / RX konnten Datenraten bis zu 1,5 MBit/s erzielt werden. Eine Steigerung der Datenrate wäre softwareseitig durch die Erhöhung des entsprechenden Parameters möglich. Die direkte Verbindung der Sende- und Empfangspins zweier ESP32-Entwicklerboards über Dupont-Kabel zeigte bis zu 20 MBaud eine erfolgreiche Übertragung. Limitierend für die Datenrate scheint das detektierte Signal an der Empfängerschaltung zu sein, das bei 1,5 MBit/s stark deformiert gemessen wird. Auch hier könnte die Verwendung eines logarithmischen Verstärkers zu einer Reichweiten- und Datenratenverbesserung führen. Eine Vergrößerung der Verstärkung des Signals der Photodiode durch eine Vergrößerung des Rückkopplungswiderstands $R1$ dürfte keine Verbesserung bringen, da hierdurch auch die Bandbreite des Operationsverstärkers reduziert wird.

Die Programme Manchester Encoder / Decoder konnten für eine Übertragung bis zu einer Datenrate von 333 Bit/s verwendet werden. Die Programmbibliotheken bieten Transparenz, um weitere Codierverfahren zu implementieren, und beispielsweise durch Kanalcodierung die Fehleranfälligkeit zu verringern. Limitierend für die Datenrate ist die Empfängerseite. Mit dem Oszilloskop wurde festgestellt, dass die Sendeseite bis zu einer Taktlänge von 50 µs zuverlässig das korrekte Signal ausgibt. Die Empfängerseite desynchronisiert jedoch bereits bei einer Takt-

länge von 1500 µs nach der Übertragung von etwa zwei Zeichen. Bei einer Taktlänge von 1000 µs scheitert schon die Synchronisierung. Abhilfe könnte es schaffen, sich kontinuierlich während einer Übertragung an den Sendetakt anzupassen, indem das `sampleArray` analysiert, und ein zeitlicher Drift korrigiert wird. Gleichzeitig scheint jedoch das häufige Sampling den Mikrocontroller an die Grenzen der Rechenkapazität zu bringen. Anstelle von 8 Samples pro Takt könnte die Samplingrate eventuell reduziert werden. Bisweilen rufen zudem die Methode `getBitOffSampleArray` und die ISR `bitsampling` auf das `sampleArray` zu, ohne dass ein Semaphor benutzt wird. Dadurch wäre es theoretisch möglich, dass Fehler entstehen, wenn beide Methoden gleichzeitig auf das Array zugreifen. Ein Semaphor würde das Array vor einem gleichzeitigen Zugriff schützen und so eventuelle Fehler ausräumen. Zusätzliche Rechenkapazität könnte zudem gewonnen werden, indem die Einbindung des zweiten Prozessorkerns bewusst genutzt wird. Momentan wird dem Mikrocontroller bei der Erstellung der Tasks die Wahl gelassen, auf welchem Kern ein Task laufen soll. Außerdem erfolgt das Sampling in der vorliegenden Fassung gleichverteilt über eine Taktlänge, obwohl Samples in der Mitte eines Taktschrittes mit höherer Wahrscheinlichkeit dem aktuell übertragenen Bitwert entsprechen. Zur Verbesserung der Effizienz des Samplings könnten Samples in der Mitte eines Taktschrittes stärker gewichtet werden als solche, die am Rand eines Taktschrittes genommen wurden.

Kapitel 6

Fazit

In dieser Arbeit wurden zwei Programmieransätze für den ESP32 entwickelt, die eine Datenübertragung mit sichtbarem Licht erlauben. Die Programmbibliothek Manchester Encoder / Receiver erreicht bisweilen eher niedrige Datenraten von 333 MBits/s, ist jedoch transparent programmiert, sodass weitere Codierverfahren implementiert werden könnten und basiert nicht auf dritten Programmbibliotheken. Zudem erlaubt die Bibliothek die selbstständige Synchronisierung des Empfängers auf den Sender, ohne dass eine Baudrate vorab angegeben werden muss. Verbesserungspotential des Programms wird vor allem bei der Synchronität nach Beginn der Übertragung und beim Bitsampling vermutet, das zum einen effizienter durch eine Auslagerung auf den zweiten Prozessorkern oder eine seltenere Ausführung und zum anderen sicherer durch die Verwendung eines Semaphors gestaltet werden könnte.

Deutlich höhere Datenraten von 1,5 MBit/s wurden mit den UART TX / RX Programmen erzielt, die die seriellen Schnittstellen des ESP32 benutzen. Möglich wurden diese Datenraten über den Entwurf eigener Sender- und einer Empfangsplatinen, die an die hohen Übertragungsfrequenzen angepasst sind. Bei der Übertragung mittels OOK werden Rechteckschwingungen übertragen, wodurch hohe Frequenzen als ungeradzahlige Harmonische der Grundfrequenz auftreten. Mit der

entworfenen Empfängerplatine können hohe Frequenzen empfangen und durch eine geschickte Komparatorschaltung auch deformierte Signale zu Rechteckschwingungen regeneriert werden. Eine Reichweite von 5,4 m bei einer Übertragungsrate von 1 MBaud wurde durch die Verwendung eines eigens erstellten Reflektors in Verbindung mit einer ausgewählten LED-Lichtquelle erreicht. Als Ursache für die Limitierung der Reichweite wird die Eingangsverstärkung vermutet. Eine Reichweitenverbesserung könnte durch die Verwendung eines logarithmischen Verstärkers als Eingangsverstärker erzielt werden. Die Sammellinse des Empfängers führt zu einer signifikanten Verbesserung der Reichweite, bündelt die Lichtstrahlen ab einer Entfernung von etwa 2,5 m aber so stark, dass die Photodiode nicht mehr vollständig ausgeleuchtet wird. Zudem führen kleinste Variationen der Geometrien zu einer Verschiebung des Fokuspunktes außerhalb der Fläche der Photodiode. Verbesserungen könnten somit zum einen dadurch erreicht werden, dass am Sender eine möglichst große Fläche in Richtung des Empfängers emittiert und zum anderen sollte eine Optik am Empfänger verwendet werden, die einen Lichtpunkt von der Größe der Photodiode bei der gewünschten Übertragungsdistanz erzeugt. Ein erster Ansatz wäre die Verwendung einer Linse mit einer größeren Brennweite. Um am Sender möglichst viel Licht in Richtung des Empfängers zu senden und die Fläche zu vergrößern, wurden drei Reflektoren gestaltet und verglichen. Dabei konnte bei Verwendung des Reflektors mit der Bezeichnung S3 ein Spannungsgewinn von 7 dB gegenüber der Lichtquelle ohne Reflektor beobachtet werden. Eine quantitative Untersuchung der Reflektoren war aus Zeitgründen jedoch leider nicht möglich, sodass dieser Versuch lediglich als Anhaltspunkt für weitere Versuche gewertet werden kann.

Literatur

[1] Peter Adam Hoeher. Visible Light Communications - Theoretical and Practical Foundations. Carl Hanser Verlag, 2019. Kap. Introduction, 3 ff.

[2] Chen Chen. “Special Issue on ’Visible Light Communication’”. In: *Visible Light Communication*. 2022.

[3] Rongqiao Wan et al. “Improving the Modulation Bandwidth of GaN-Based Light-Emitting Diodes for High-Speed Visible Light Communication: Countermeasures and Challenges”. In: *Advanced Photonics Research* (8. Dez. 2021).

[4] Jae Kyun Kwon Kang-Il Ahn. “Color Intensity Modulation for Multicolored Visible Light Communications”. In: *IEEE Photonics Technology Letters* (15. Dez. 2012).

[5] Artikelseite ”trulifi”, Firma Signify. 12. Apr. 2023. URL: `https://www.signify.com/de-at/innovation/trulifi`.

[6] Homepage ”pureLiFi”. 12. Apr. 2023. URL: `https://purelifi.com/`.

[7] Ekbert Hering. Elektronik für Ingenieure und Naturwissenschaftler. Springer Vieweg, 2014. Kap. Optoelektronik, 303 ff.

[8] Jonathan J. D. McKendry et al. “Visible-Light Communications Using a CMOS-Controlled Micro-Light Emitting-Diode Array”. In: *Journal of Lightwave Technology* (1. Jan. 2012).

[9] Jia-Ning Guo et al. “Constant Transmission Efficiency Dimming Control Scheme for VLC Systems”. In: *Photonics* (31. Dez. 2020).

[10] Martin Werner. Nachrichtentechnik - Eine Einführung für alle Studiengänge. Springer Vieweg, 2017. Kap. Modulation, 241 ff.

[11] Eric Monteiro. “Design and Implementation of Color-Shift Keying for Visible Light Communications”. In: *Journal of Lightwave Technology* (15. Mai 2014).

[12] IEEE Standard 802.15.7-2018. 5. Dez. 2018. URL: https://standards.ieee.org/ieee/802.15.7/6820/.

[13] Martin Werner. Nachrichtentechnik - Eine Einführung für alle Studiengänge. Springer Vieweg, 2017. Kap. Digitale Übertragsung im Basisband, 165 ff.

[14] Ulrich Tietze, Christoph Schenk und Eberhard Gamm. Halbleiter-Schaltungstechnik. Springer, 2022. Kap. Optoelektronische Bauelemente, 1168 ff.

[15] Datenblatt L-934MBTL. Kingbright, 14. März 2023. URL: https://cdn-reichelt.de/documents/datenblatt/A500/L-934MBTL%28V1%29.pdf.

[16] Datenblatt LM301B. Samsung, 14. März 2023. URL: https://cdn.samsung.com/led/file/resource/2022/04/Data_Sheet_LM301B_CRI80_Rev.10.2.pdf.

[17] Datenblatt LL-504BC2E-B4-2CC. Lucky Light, 14. März 2023. URL: https://cdn-reichelt.de/documents/datenblatt/A500/LL-504BC2E-B4-2CC.pdf.

[18] Datenblatt Luxeon Rebel. Lumileds, 14. März 2023. URL: https://lumileds.com/wp-content/uploads/files/DS64.pdf.

[19] Eigenleitung - Wikipedia. 14. März 2023. URL: https://de.wikipedia.org/wiki/Eigenleitung.

[20] attachInterrupt() - Arduino.cc. 2. März 2023. URL: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/.

[21] Interrupt allocation - ESP32 - ESP IDF Programming Guide. 2. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/intr_alloc.html.

[22] General Purpose Timer - ESP32 - ESP-IDF Programming Guide. 2. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/peripherals/timer.html.

[23] Universal asynchronous receiver - transmitter - Wikipedia. 7. März 2023. URL: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[24] Universal Asynchronous Receiver/Transmitter (UART) - ESP32 - ESP-IDF Programming Guide. 2. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/uart.html.

[25] Datenblatt Philips UART SCC2691. Philips, 2. März 2023. URL: https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf.

[26] Understanding UART - Rohde & Schwarz. 7. März 2023. URL: https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html#:~:text=UART%20stands%20for%20universal%20asynchronous,also%20have%20a%20ground%20connection..

[27] Determining Clock Accuracy Requirements for UART Communications. 7. März 2023. URL: https://pdfserv.maximintegrated.com/en/an/AN2141.pdf.

[28] ESP32 Technical Reference Manual. Kapitel 13. Version 4.8. 7. März 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf#uart.

[29] About Arduino. 7. März 2023. URL: https://www.arduino.cc/en/about.

[30] Dr. Bert Klöppel. Objektorientierte Modellierung und Programmierung mit C++. R. Oldenbourg, 1997, 105 ff.

[31] #include - Arduino Reference. 8. März 2023. URL: https://www.arduino.cc/reference/en/language/structure/further-syntax/include/.

[32] ESP32 Dual Core with Arduino IDE - Random Nerd Tutorials. 8. März 2023. URL: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/.

[33] Github - askhipenko / TaskScheduler. 8. März 2023. URL: https://github.com/arkhipenko/TaskScheduler.

[34] FreeRTOS (Overview) - ESP32. 8. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html.

[35] FreeRTOS for Arduino Boards. 8. März 2023. URL: https://wiki.seeedstudio.com/Software-FreeRTOS/.

[36] FreeRTOS FAQ relating to FreeRTOS memory management and usage. 8. März 2023. URL: https://www.freertos.org/FAQMem.html#StackSize.

[37] Multitasking - Wikipedia. 8. März 2023. URL: https://de.wikipedia.org/wiki/Multitasking.

[38] ESP32 ESP-IDF FreeRTOS Queue Tutorial. 8. März 2023. URL: https://esp32tutorials.com/esp32-esp-idf-freertos-queue-tutorial/.

[39] FreeRTOS - Queues for task and interrupt message passing in FreeRTOS real time embedded software applications. 8. März 2023. URL: https://www.freertos.org/Embedded-RTOS-Queues.html.

[40] FreeRTOS - Market leading RTOS (Real Time Operating System for microcontrollers. 9. März 2023. URL: https://www.freertos.org/.

[41] API - Wikipedia. 9. März 2023. URL: https://en.wikipedia.org/wiki/API.

[42] FreeRTOS (ESP-IDF) - ESP32. 9. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos_idf.html.

[43] FreeRTOS (Supplemental Features) - ESP32. 9. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos_additions.html.

[44] Serial - Arduino Reference. 10. März 2023. URL: https://www.arduino.cc/reference/en/language/functions/communication/serial/.

[45] Programmbibliothek HardwareSerial - abgerufen bei GitHub. 10. März 2023. URL: https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/HardwareSerial.cpp.

[46] #define directive (C/C++ - Microsoft Learn. 10. März 2023. URL: https://learn.microsoft.com/en-us/cpp/preprocessor/hash-define-directive-c-cpp?view=msvc-170.

[47] ESP32 with PIR Motion Sensor using Interrupts and Timers. 11. März 2023. URL: https://randomnerdtutorials.com/esp32-pir-motion-sensor-interrupts-timers/.

[48] GPIO - Arduino ESP32 2.0.6 documentation. 11. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/intr_alloc.html.

[49] Interrupt allocation - ESP32. 11. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/intr_alloc.html.

[50] General Purpose Timer - ESP32. 10. März 2023. URL: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/peripherals/timer.html.

[51] Timer - Arduino-ESP32 2.0.6 documentation. 10. März 2023. URL: https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/api/timer.html.

[52] Programmbibliothek ESP32 Timer - abgerufen bei GitHub. 11. März 2023. URL: https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-timer.c.

[53] Bit banging - Wikipedia. 11. März 2023. URL: https://en.wikipedia.org/wiki/Bit_banging.

[54] Datenblatt ESP32 Series. Espressif, 8. Apr. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[55] Produktbild LT-3182. 8. Apr. 2023. URL: https://www.led-tech.de/de/7x-Samsung-LM301B-Plug-and-Light-Star-4000k#.

[56] Tim Christiansen. "Konzeption und Erprobung einer Outdoor-LiFi Anwendung zur automatischen Dimmung von Außenbereichsleuchten". Bachelorarbeit. Hochschule für angewandte Wissenschaften Hamburg, 6. Mai 2022.

[57] J. Stapel et al. Welches Licht darf es sein? 2014. URL: https://www.ifte.de/mitarbeiter/reifegerste-Dateien/LICHT2014.pdf.

[58] Patrick Schnabel. Lichtstrom Phi (luminous flux, luminous power). 8. Apr. 2023. URL: https://www.elektronik-kompendium.de/sites/grd/2205181.htm.

[59] Datenblatt BPW43. Vishay Telefunken, 20. Mai 1999.

[60] Govind P Agrawal. Fiber-Optic Communication Systems. Wiley, 2021, S. 109.

[61] Datenblatt AD8055. Analog Devices, 2006.

[62] Ekbert Hering. Elektronik für Ingenieure und Naturwissenschaftler. Springer Vieweg, 2014. Kap. Operationsverstärker, 424 ff.

[63] Datenblatt L3705N. kein Datum. VBsemi.

Anhang

Auf dem angefügten USB-Stick befinden sich die Programmcodes für die Programme Manchester Encoder / Transmitter, Manchester Decoder / Receiver, UART TX und UART RX. Zudem sind die STL-Dateien der Modelle der Reflektoren S1, S2 und S3 vorhanden.

Danksagung

Ich danke Herrn Prof. Dr. Jan Mietzner für die Übernahme der Erstprüferschaft, für die freundliche Beratung und kompetente Betreuung. Ich bin sehr dankbar für die zahlreichen Vorlesungen in Elektrotechnik, Digitale Signalverarbeitung, Nachrichtentechnik & Telekommunikation, die ich bei ihm besuchen durfte und bei denen ich, selbst in Zeiten von Corona, vieles gelernt habe.

Ich danke Herrn Reinhard Breuer für die freundliche Betreuung während des praktischen Teils der Arbeit, für die große Hilfe im Elektrotechniklabor und für die vielen Anregungen bei der Erstellung der Schaltplatinen. Selbst in Zeiten, in denen abseits der Hochschule viel wichtigere Dinge als eine Abschlussarbeit anstanden, konnte ich mich auf seine Unterstützung verlassen. Ich wünsche ihm und allen Studierenden des Fachbereichs, dass er noch lange am Department mitwirken kann.

Ich danke Magdalena Appelhans, Sophie Schmidt und Julia Gießmann für die Geduld beim Korrekturlesen. Ich hoffe, es war nicht zu schlimm.

Ich danke Simon, Johanna und Kai für die Freundschaften, die ich in diesem Studium gewonnen habe. Danke für unzählbare Abende, mit Tränen vor Lachen in den Augen, für schlechte Wortwitze, lange Unterhaltungen, für Herz an Herz, für griechischen Wein, für 5 km/h, … Ohne euch wäre das Studium nicht dasselbe gewesen.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Codierte Datenübertragung per Licht und Erprobung im Rahmen einer drahtlosen Anwendung für Außenbereichsleuchten

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 13. April 2023