



TECHNISCHE UNIVERSITÄT BERLIN
FAKULTÄT IV - ELEKTROTECHNIK UND INFORMATIK
FACHGEBIET LICHTTECHNIK

Bachelorarbeit

Veranschaulichung digitaler Steuerprotokolle in der Lichttechnik

Name: Klaus, Conrad Yosemite
Matrikelnummer: 401498
Studiengang: Technische Informatik
Abgabedatum: 25. April 2023

Betreuer: Frithjof Hansen, M.Sc.
Gutachter/innen: Prof. Dr.-Ing. Stephan Völker
Prof. Dr.-Ing. Sibylle Dieckerhoff

Berlin, 2023

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit "Veranschaulichung digitaler Steuerprotokolle in der Lichttechnik" ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt.



....., Berlin, 25. April 2023

(Klaus, Conrad Yosemite)

Abstrakt

In der Lichttechnik haben sich seit den 80er-Jahren verschiedene Standards durchgesetzt. Je nach Anwendungsfall werden verschiedene Protokolle präferiert. Die Protokolle DMX, Dali, und seit Neuerem auch ZigBee haben sich besonders in der Industrie etabliert. Im Hobbybereich findet auch vermehrt das MIDI Protokoll Anwendung.

In der Bachelorarbeit werden die Stärken und Schwächen der Protokolle miteinander verglichen. Dabei werden Faktoren wie die Geschwindigkeit, Nachrüstbarkeit und Benutzerfreundlichkeit näher untersucht.

In der Bühnentechnik hat sich das DMX Protokoll als Industriestandard bewährt. Der Aufbau des Protokolls ermöglicht es, Datenpakete einfach mitzuschneiden. In der Bachelorarbeit wird für die Lehre ein Projekt gebaut, um Studierende das DMX Protokoll anschaulich zu erklären. Dabei wird das DMX Protokoll sowohl auf der Anwenderschicht intuitiv erklärt, als auch auf der physischen Übertragungsschicht aufgeschlüsselt.

Inhaltsverzeichnis

1 Zielstellung und Aufbau der Arbeit	1
2 Protokolle Lichttechnik	2
2.1 DALI	2
2.2 DMX	6
2.3 MIDI	9
2.4 ZigBee	11
2.5 Vergleich	14
3 Technische Details vom DMX Protokoll	15
3.1 Verwendung des UART-Protokolls	15
3.2 Aufbau des DMX-Pakets	17
3.3 Zeitdiagramm	18
4 Hardware	20
4.1 Raspberry Pi	21
4.2 LED-Matrix	21
4.3 Platine DMX Breakout	23
4.4 Logikanalysator	26
4.5 Gehäuse	30
5 Software	33
5.1 Verwendete Technologien	33
5.2 Vergleich: Rust vs C	34
5.3 Bildschirm Treiber	36
5.4 Logikanalysator	37
5.5 Entwicklung mithilfe Simulation	37
5.6 Signal Decodierung	40
5.7 Signal Darstellung	42
5.7.1 Reset Sequenz	42

5.7.2 Kanal 1	43
5.7.3 Anwendungsbeispiel	44
5.8 Ergebnisse	45
5.9 Kontrolle DMX Signalanalyse	46
6 Fazit	50
Abbildungsverzeichnis	52
Tabellenverzeichnis	54
Literaturverzeichnis	55

1. Zielstellung und Aufbau der Arbeit

In der Lichttechnik haben sich je nach Anwendungsbereich verschiedene Lichtsteuerprotokolle durchgesetzt. In dieser Bachelorarbeit werden verschiedene Lichtsteuerprotokolle miteinander verglichen und die spezifischen Anwendungsgebiete herausgearbeitet.

Für die Lehre wird ein DMX-Analysator gebaut. Diese Bachelorarbeit setzt daher den Schwerpunkt auf das DMX Protokoll. Dieser Analysator funktioniert eigenständig und benötigt keine externe Hard- oder Software. Dieses Projekt wird eingesetzt, um Studierende das DMX Protokoll zu erklären und die Stärken und schwächen des Protokolls zu zeigen.

Das DMX Protokoll ist in der Bühnentechnik ein etabliertes Protokoll. Es gibt bereits einige gute Internetseiten, die das DMX Protokoll aus der Anwendersicht verständlich beschreiben. In dieser Bachelorarbeit wird jedoch der Fokus nicht nur auf die Anwenderschicht gesetzt. In dieser Bachelorarbeit wird auch die genaue Funktionsweise des Protokolls beschrieben, wie die Daten im Kabel übertragen und wie die Informationen decodiert werden.

2. Protokolle Lichttechnik

In der Lichttechnik haben sich mehrere Standards durchgesetzt. Abhängig vom Anwendungsfall werden meistens verschiedene Standards bevorzugt. Bei der Gebäudebeleuchtung wird DALI häufig genutzt. Aber auch neuere Technologien wie z.B. ZigBee oder Philips Hue setzen sich bei der privaten SmartHome Integration durch. In der Bühnentechnik hat sich der DMX Standard als industrieller Standard durchgesetzt. Der DMX Standard ist ebenfalls offen und kann von jedem implementiert werden. Allerdings sind die Leuchten, die auf dem freien Markt erhältlich sind, oft teuer. Für die Signalsteuerung ist entweder ein DMX Controller oder DMX Software erforderlich. Auch diese sind oft teuer oder lizenpflichtig. In der Hobbyszene wird daher gerne ein anderes Protokoll für die Lichtkoordination umfunktioniert: MIDI. Das Protokoll, wurde ursprünglich für die Übertragung von Musik erstellt, kann aber auch genutzt werden, um Lichtinformationen zu übermitteln.

2.1. DALI

Das Lichtsteuerungsprotokoll DALI (*Digital Addressable Lighting Interface*) ist in der Gebäudebeleuchtung weitverbreitet. Vor allem in Bürogebäuden,

Hotels oder auch im Einzelhandel wird es oft verwendet. Einzelne Leuchten können spezifisch gesteuert oder zu Gruppen zusammengefasst werden.

DALI wurde erstmals in den späten 1990er [1] von der europäischen Lichtindustrie entworfen. Viele Hersteller verwenden seither das Protokoll weltweit für die Steuerung von deren Produkten. Dadurch können Vorschaltgeräte vor den Leuchten von verschiedenen Hersteller nebeneinander eingesetzt werden, ohne dass der Anwender sich mit technischen Unterschieden auseinandersetzen muss [2, S.2, ch. 3.1].

Eines der Hauptvorteile von DALI ist die genaue Steuerung einzelner oder zu Gruppen zusammengefassten Leuchten. Jede Leuchte kann eine eindeutige Adresse zugewiesen bekommen. Die einzelnen Leuchten können dadurch separat voneinander gesteuert werden. Das Protokoll ermöglicht es, die Leuchten zu dimmen und somit maßgeschneidert an die speziellen Bedürfnisse der Umgebung anzupassen. Helles Arbeitslicht, Umgebungslicht als auch eine Akzentbeleuchtung kann damit gesteuert werden. Es können auch verschiedene Lichtszenen vorprogrammiert werden, z.B. eine Szene für die Tag- und Nachtbeleuchtung, (vgl.2.1).



(A) Szene Schauraum Tag



(B) Szene Schauraum Nacht

FIGURE 2.1: Beispiele für Lichtszenen [3, S.6]

DALI ist ein digitales Protokoll, welches eine bidirektionale Kommunikation zwischen Leuchten und Controller ermöglicht[2, S.1]. Dadurch können auch Fehlermeldungen der Leuchten ausgelesen werden. Eine kaputte Leuchte kann so einfach gefunden und ausgetauscht werden. Aber auch andere Sensoren wie Tageslichtsensoren oder Bewegungssensoren können ins Netzwerk eingeschlossen werden. Damit kann die Helligkeit beispielsweise durch das Tageslicht oder durch die Anwesenheit von Menschen gesteuert werden. Der Stromverbrauch kann so massiv reduziert werden, da die Leuchten nur benutzt werden, wenn sie auch tatsächlich gebraucht werden.

DALI basiert auf einem einfachen zwei Kabel Bussystem. Das Bussystem kann am Ende des Busses fortgeführt werden, um es nachträglich zu erweitern. Der DALI Bus besteht aus zwei Leitungen. Ist die Verwendung von DALI gewünscht, so müssen ggf. diese Leitungen nachgerüstet werden. Wird dies bereits beim Bau eingeplant, können die freien Adern in einer Netzeleitung benutzt werden[2, S. c.3.2.2] (vgl.2.2). Auf die Polarität der DALI Signale muss nicht geachtet werden [3, p.3].

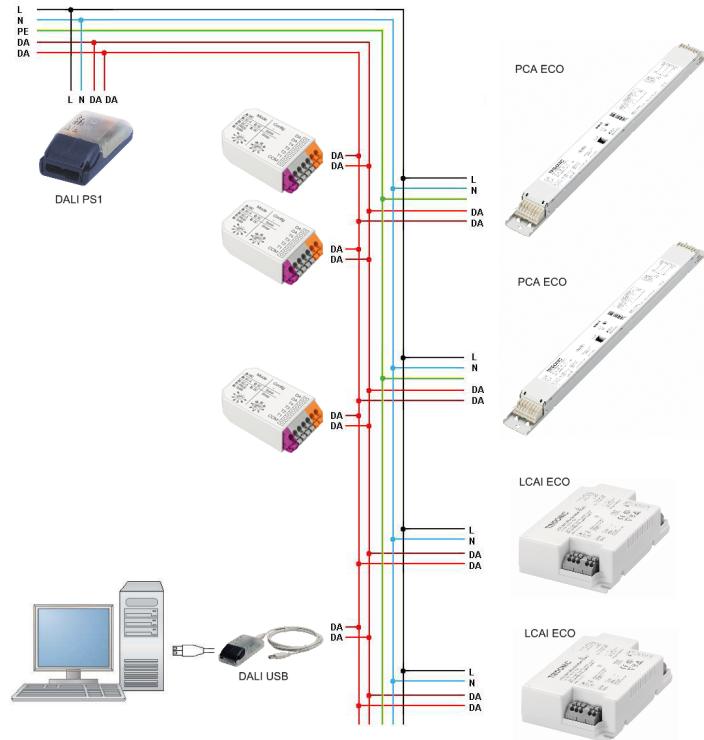


FIGURE 2.2: Verdrahtungsdiagramm [3, S.64]

Weiterhin können die DALI Controller einfach programmiert werden, um eine spätere Änderung der Leuchten einzustellen.

Neben den vielen Vorteilen hat das DALI Protokoll auch einige Limitierungen. Ein DALI Bus ist insgesamt auf max. 64 Adressen begrenzt [3, S.7]. Jede Adresse hat einen Wertebereich von 8 Bit [2, S.1]. Eine Leuchte kann daher z.B. auf max. 256 verschiedene Helligkeitsstufen eingestellt werden. Es können maximal 16 Gruppen und 16 verschiedene Lichtszenen programmiert werden. Die geringe Datenrate von $1,2 \frac{\text{kbit}}{\text{s}}$ limitiert das Protokoll auf eine Aktualisierungsrate von ca. 25-30 Herz. Weiterhin können die Kabel maximal 300 Meter lang verlegt werden, bevor sie wieder verstärkt werden müssen [3, S.3].

Im Vergleich zu einem analogen System, hat DALI keine großen Probleme mit Interferenzen oder Störungen von externen Quellen. Das Protokoll ist selbst auf langen Strecken sehr zuverlässig.

Insgesamt ist DALI ein leistungsstarkes Lichtsteuerprotokoll, das durch seine Flexibilität, Genauigkeit und Energieeffizienz viele Vorteile gegenüber einem analogen traditionellen Steuersystem bietet. Die Adaption der Lichtindustrie hat es zu einem Industriestandard für die Gebäudebeleuchtung aller Art gemacht.

2.2. DMX

DMX (*Digital Multiplex*) ist ebenfalls wie DALI ein Lichtsteuerprotokoll. Der Anwendungsfall ist meist jedoch ein anderer. DMX ist der Industriesstandard in der Bühnentechnik und Unterhaltungsindustrie. DMX wird benutzt, um steuerbare Leuchten zu koordinieren, wie z.B. bewegbare Leuchten und LED Farbleuchten. Dabei können alle Leuchten zentral von einem dedizierten physischen Controller oder über Software gesteuert werden (vgl. 2.3).

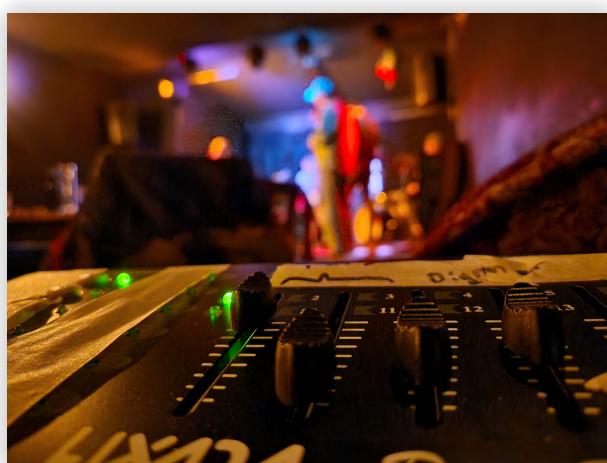


FIGURE 2.3: DMX gesteuerte Lichtszene in einer Berliner Bar

DMX wurde erstmals im Jahr 1986 vorgestellt [4, S.5, ch. 2.1]. Seither wurde es der verbreitetste Industriestandard für koordinierte Lichtshows. Genau wie DALI, ist DMX ebenfalls ein digitales Steuerprotokoll. Über ein DMX Kabel kann jeweils ein DMX Universum übertragen werden. Ein DMX Universum kann mehrere Kanäle gleichzeitig steuern. Ein Kanal kann dabei, z.B. auf die Farbe einer Leuchte, oder die Helligkeit einer Leuchte abgebildet werden. Das DMX Protokoll basiert auf einer seriellen Datenübertragung. Das Protokoll baut auf dem RS-485 Standard[4, S.11, ch. 3.2] auf.

Eines der Hauptvorteile von DMX ist die zentrale Steuerung von einer großen Anzahl von Leuchten über eine Konsole. Komplexere Leuchtkonstellationen mit vielen unterschiedlichen Leuchten sind dadurch umsetzbar. Mehrere Leuchten können so zusammengeschaltet werden, um komplizierte Effekte und Lichtszenen aufzubauen. Oftmals werden verschiedene Animationsmuster vorprogrammiert und als Lichtsequenz abgespeichert. Lichtgestalter/-innen können dadurch einfach und schnell verschiedene Animationen und Konfigurationen aufrufen, um die Lichtszenerie auf die verschiedenen Phasen der Show abzustimmen.

Die Verkabelung von DMX ist einfach. In der Regel wird das Datensignal getrennt von der Stromversorgung geführt. Das Datensignal kann mithilfe einer Gänseblümchenkette (*Daisy Chain*) (vgl. 2.4), einfach von Leuchte zu Leuchte verkabelt werden. Am Ende der Gänseblümchenkette, ist ein DMX-Terminator erforderlich, um die Datenintegrität innerhalb des Busses zu gewährleisten. Bei größeren Distanzen (ab 300 Meter) muss das DMX Signal verstärkt werden, um eine sichere Datenübertragung zu gewährleisten.

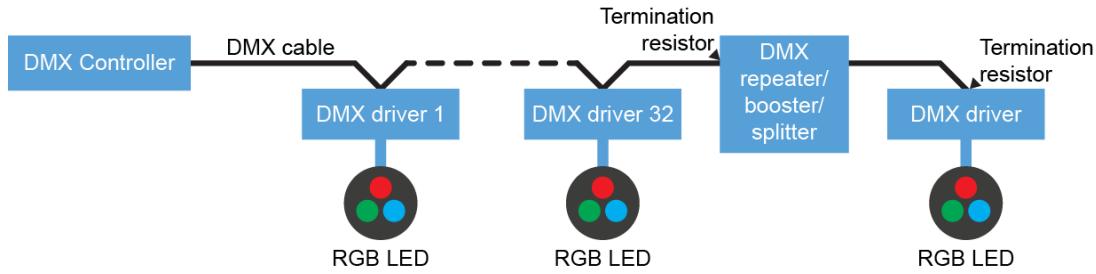


FIGURE 2.4: Verdrahtungsdiagramm [5, S.64]

DMX ist unter anderem auch sehr flexibel und konfigurierbar. Das Protokoll erlaubt die Ansteuerung von selbst gebauten Leuchten. Hersteller können für ihre Produkte auch bereits vorgefertigte DMX Konfigurationen mitliefern. Lichtgestalter/-innen bekommen dadurch eine genaue Kontrolle über die Funktionalitäten der einzelnen Leuchten, wie z.B. die Farbeinstellungen oder Filterauswahl.

Ein weiterer Vorteil von DMX, ist die Möglichkeit mehrere Systeme zusammenzuschalten. Eine große Anzahl an DMX Universen können über eine Konsole gesteuert werden, um eine größere Anzahl von Leuchten zu steuern. DMX kann auch mit Audio oder Video Systemen integriert werden, um eine synchrone Licht und Sound Show mit visuellen Effekten aufzubauen.

Neben den vielen Vorteilen hat DMX auch einige Einschränkungen. Ein unverstärktes Signal kann maximal 300 Meter lang werden, bevor das Signal durch die Signalverschlechterung erneut verstärkt werden muss. Daher müssen für längere Strecken zwingend DMX Verstärker verwendet werden, um die Signalstärke beizubehalten. Ein DMX Universum erlaubt eine Kontrolle von bis zu 512 Kanälen. Die Datenpakete werden mit einer Datenrate von insgesamt $250 \frac{\text{kbit}}{\text{s}}$ gesendet. Sind alle Kanäle ausgeschöpft, ist die maximale Aktualisierungsrate $44 \frac{\text{Aktualisierungen}}{\text{s}}$ [6, S.18, table6]

Zusammengefasst ist das DMX Protokoll ein verbreitetes Lichtsteuerungsprotokoll, welches in der Industrie oft zu finden ist. Die Möglichkeit mehrere Leuchten getrennt voneinander ansteuern zu können, und die Integrierbarkeit mit anderen Systemen, hat es eine wesentliche Komponente für Lichtgestalter/-innen gemacht.

2.3. MIDI

Das MIDI (*Musical Instrument Digital Interface*) Protokoll ist ein Musikprotokoll, welches primär in der Musikszene genutzt wird. MIDI erlaubt es elektronische Musikinstrumente, Computer, oder auch andere Geräte wie z.B. Synthesizer, miteinander zu interagieren. Musizierende können dadurch den Sound eines Instrumentes während des Spielens anpassen, und neue Effekte auf kreative Art und Weise aufzubauen.

MIDI wurde zuerst in den frühen 1980er Jahren [7, S.1] vorgestellt, und hat sich schnell als neuer Standard für die elektronische Musikproduktion durchgesetzt. Das Protokoll erlaubt eine Übertragung von musikalischen Daten, wie Ton-An, Ton-Aus, Tonhöhe, Dämpfung, Modulation etc. zwischen MIDI-Kompatiblen Geräten [8, S.9]. Dadurch können Künstler/-innen Musik mit einem elektronischen Klavier aufnehmen und die Töne, zusammen mit Metainformationen, wie dem Tastendruck, Dauer des Tastenanschlages usw. aufnehmen, und auf einem anderen Gerät abspielen oder auch im Nachhinein verändern.

MIDI kann auch benutzt werden, um virtuelle Instrumente zu simulieren und diese klanglich mit Software Erweiterungen stark zu verändern. Künstler/-innen haben dadurch eine Vielzahl von verschiedenen Geräuschen und Effekten zur Auswahl.

Ein weiterer Vorteil des MIDI Protokolls ist die große Anzahl von MIDI kompatiblen Geräten. MIDI kompatible Geräte sind unter anderem elektronische Tasteninstrumente, Synthesizer, Trommelmaschinen und viele weitere. Künstler/-innen können einfach eine Vielfalt von Geräten zusammen benutzen, um neue kreative und raffinierte musikalische Werke zu produzieren.

Im Vergleich zu DMX werden die Daten im MIDI Protokoll auf einer anderen Art und Weise verschickt. DMX sendet jedes Mal alle aktiven Kanäle. Sollte ein Paket nicht beim Empfänger ankommen, ist dies kein Problem, da das nächste Paket wieder alle Informationen enthält. Dadurch werden Informationen, die sich nicht ändern, jedes Mal neu versendet.

Bei MIDI werden nicht kontinuierlich alle Informationen gesendet. MIDI hat einen anderen Ansatz. Signale, wie z.B. Ton-An beim Tastenanschlag, werden nur einmal gesendet [8, S.3]. Dadurch werden keine Informationen redundant versendet.

Das MIDI Protokoll wurde erweitert, um neben den Musikinformationen, zusätzlich auch Lichtinformationen versenden zu können, [9, S.1] (vgl. 2.5). Die Leuchten, können programmiert werden, um auf Ton-An und Ton-Aus Signale zu hören [9, S.4]. Über Signale mit Effekt Informationen können Farben und weitere Optionen eingestellt werden [9, S.6, ch. 2.2.1.2].

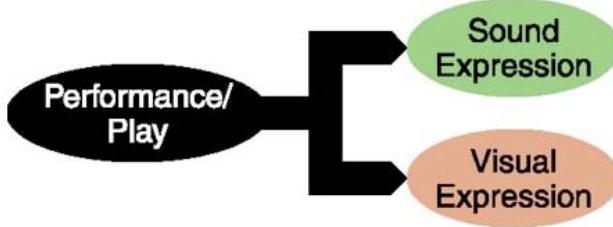


FIGURE 2.5: MIDI Licht Integration [9, S. 1]

Neben den vielen Vorteilen hat MIDI aber auch einige Einschränkungen. MIDI ist nicht in der Lage, rohe Audiodateien zu versenden. MIDI kann nur digitale Signale verschicken, welche die Parameter und Benutzung der Musikinstrumente beschreibt. Daher ist auch ein separates Audiosignal erforderlich, um die Musik aufzunehmen, die mithilfe von MIDI erstellt wird. MIDI läuft mit ca. $31 \frac{\text{kbit}}{\text{s}}$ [8, S. 1]. DMX hat im Vergleich dazu eine 8-mal höhere Datenrate.

Die genaue und flexible Einstellungen hat es für Musizierende eine wesentliche Komponente für die Musikproduktion gemacht.

2.4. ZigBee

ZigBee ist ein Kabelloses-Protokoll, welches in den letzten Jahren stark an Popularität gewonnen hat. Es wird gerne für die Smarthomeautomatisierung eingesetzt. Philips Hue ist eine Smarthome Produktgruppe, welche über das ZigBee Protokoll läuft, um Benutzer/-innen die Steuerung von Leuchten z.B. in einer Wohnung über das Mobiltelefon zu ermöglichen. Auch eine Integration mit anderen Systemen ist möglich. So kann das Lichtbild an die Musik

angepasst werden, oder Schalosien, am Abend automatisch heruntergefahren werden.

ZigBee und IEEE 802.15.4 sind standardisierte Protokolle, welche die Netzwerkinfrastruktur bereitstellen. Dabei definiert der IEEE 802.15.4 Standard den physikalische Layer, inklusive der MAC Adressen. ZigBee hingegen definiert die Netzwerk- und Anwendungsschicht [10, S.5].

ZigBee hat einen geringen Stromverbrauch mit entsprechend verlangsamten Datenraten. Dadurch können auch batteriebetriebene Produkte lange durchhalten. Das Protokoll wurde entwickelt, um mit allen möglichen IoT (*Internet of Things*) Geräten zu kommunizieren.

ZigBee Geräte kommunizieren über einer Mesh-Netzwerk-Topologie [10, S.10]. Das bedeutet, jedes Gerät (oder auch jeder Knotenpunkt) kann als Verstärker fungieren, um die Netzwerkreichweite zu vergrößern. Dadurch können ZigBee Geräte durch die Wohnung verteilt werden, um ein robustes und zuverlässiges System aufzubauen.

Philips Hue ist ein Produkt, welches sich auf intelligenten LED Leuchten fokussiert. Es kann auch mit anderen ZigBee kompatiblen Geräten verschaltet werden. Philips Hue kann per App einfach übers Mobiltelefon gesteuert werden. Die Helligkeit, Farbwärme oder auch ein Flackern wie bei einer Kerze kann eingestellt werden. Philips Hue kann unter anderem auch über Sprachassistenten wie Alexa oder dem Google Assistenten gesteuert werden.

Ein weiterer Vorteil vom Hue System, ist der einfache Aufbau und Einrichtung. Mithilfe der Hue App können Lichtszenen einfach erstellt und abgerufen werden. Timer können einprogrammiert werden, und mit einer

Internetverbindung von überall, auch außer Hause, ferngesteuert werden. Auch können Bewegungsmelder, Dimmer oder Batterie betriebene Schalter benutzt werden und überall im Hause verteilt werden, um die Lichter entspannt von überall aus steuern zu können. Auch intelligente Steckdosen können mit dem System kontrolliert werden.

Die Frequenzbänder des ZigBee Protokolls überlappen sich mit den Frequenzbändern des Bluetooth und WLAN Standards (vgl. 2.6).

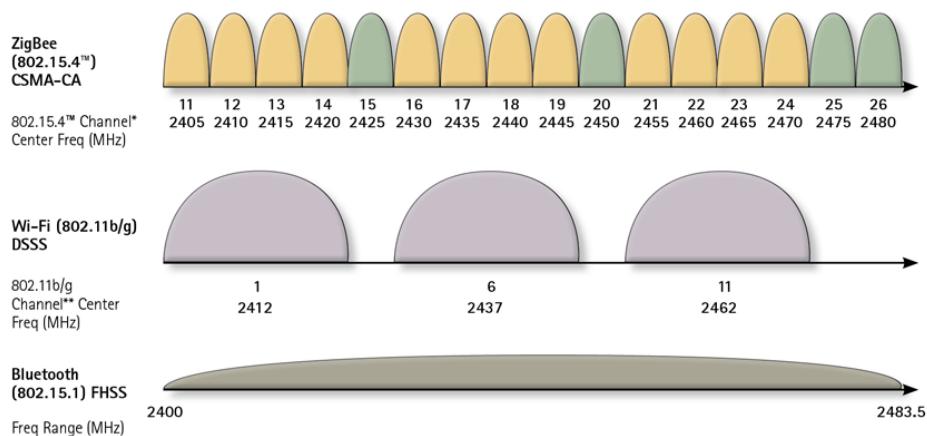


FIGURE 2.6: ZigBee Frequenzüberlappung [10, S.23]

Die grauen Bänder haben weniger Überlappungen mit den WLAN-Kanälen. Sie können gewählt werden, um weniger Interferenzen zwischen den beiden Protokollen zu erhalten.

Durch die kabellose Kommunikation ist keine Nachrüstung von Kabelsystemen erforderlich. Dies erleichtert den Einstieg für Konsumenten erheblich. Sie müssen keine neuen Kabel verlegen, oder sich mit der Elektronik auskennen. ZigBee ist ein allgemeines Protokoll, welches für IoT Geräte aller Art entwickelt worden ist. Es wird oft bei Lichtinstallationen in privaten Wohnungen genutzt.

Zusammengefasst hat das ZigBee Protokoll und die Philips Hue Produkte eine neue Nische im vernetzten und intelligenten Zuhause gefunden. Vor allem die einfache Installation und Nachrüstung macht es für den Verbraucher sehr einfach, auf Philips Hue umzuwechseln.

2.5. Vergleich

Die nachfolgende Tabelle 2.1 stellt die bereits angesprochenen Eigenschaften der Protokolle anschaulich da.

Kategorie / Protokoll	DALI	DMX	MIDI	ZigBee
Aktualisierungsrate	30 Hz	44 Hz	Signal basiert	Netzwerk abhängig
Datenrate in $\frac{\text{kbit}}{\text{s}}$	1,2	250	31.25	40 - 250
Unterschiedliche Kanäle	64 Adressen	512 Kanäle	16 Geräte	65,534 Geräte
Topologie	Bus	Bus	Bus	Mesh (<i>Peer-To-Peer</i>)
Exemplarische Verwendung	Häuser	Bühnen	Hobby- / Musikszene	Intelligentes Zuhause

TABLE 2.1: Lichtsteuer Protokolle im Vergleich

3. Technische Details vom DMX Protokoll

Ein DMX-Paket besteht aus verschiedenen Segmenten. Jedes Paket beginnt mit dem Halten von zwei Logiksignalen: Ein logisches Aus, gefolgt von einem logischen An. Das Logische An wird als Pause (*Break*) im DMX-Protokoll definiert. Das logische Aus wird als *MAB* (*Mark After Break*) definiert. Das Pausesignal markiert den Anfang des Pakets. Nach dem Break folgen insgesamt bis zu 513 decodierte Bytes. Das erste Byte bestimmt den *Startcode*. Der *Startcode* beschreibt den Inhalt der nachfolgenden Bytes. Ein Standard DMX Paket fängt mit einem *Startcode* gleich 0 an. Die restlichen Bytes im DMX-Protokoll werden für die bis zu 512 unterstützten Kanäle verwendet[6, S.14]. Das Pausesignal zusammen mit dem *Startcode* wird als Reset Sequenz bezeichnet[6, S.17].

3.1. Verwendung des UART-Protokolls

Die Decodierung aller Bytes erfolgen mithilfe der Verwendung des UART-Protokolls. UART ist ein serielles Datenprotokoll, bei dem die Bits nicht parallel über mehrere Kabel, sondern zeitlich nacheinander in der Leitung übertragen werden.

Das UART-Protokoll verwendet eine zuvor beim Sender und Empfänger definierte Geschwindigkeit für die Datenübertragung [11, S.2]. Das UART-Protokoll unterstützt optionale Start- und Stopbits, um die Stabilität zu erhöhen und die Datenintegrität zu prüfen[11, S.2]. Die Start- und Stopbits werden vor bzw. hinter dem gesendeten Byte verschickt. Das Startbit ist immer ein logische Aus, während die Stopbits aus einem logischen An bestehen[11, S.2]. Zusätzlich können Paritätsbits mitgesendet werden, um die Fehlertoleranz zu erhöhen[11, S.2]. Die Bits eines Bytes können beim UART-Protokoll in zwei verschiedenen Reihenfolgen gesendet werden: Entweder wird mit dem höchstwertigen Bit (*Most Significant Bit - MSB*), oder das niedrigstwertige Bit (*Least Significant Bit - LSB*) zuerst übertragen[11, S.2]. Sender und Empfänger müssen sich vor der Kommunikation über die Konfiguration abstimmen.

Das DMX-Protokoll basiert auf dem UART-Protokoll und verwendet ein Startbit, zwei Stopbits und fängt mit dem höchstwertigen Bit an. Die Bits werden mit einer Datenrate von $250\frac{\text{kbit}}{\text{s}}$ übertragen.

In einem DMX-Paket werden mehrere Bytes zu einem Paket zusammengefasst. Ein Paket wird durch die Reset Sequenz markiert. Das Break Signal in der Reset Sequenz ist jedoch nicht im UART-Protokoll definiert. Versucht man also, ein vollständiges DMX-Paket nur mit einem standardmäßigen UART-Empfänger zu empfangen, können nicht alle Teile des Signals erkannt werden, und das Paket kann nicht decodiert werden[12, S.29].

3.2. Aufbau des DMX-Pakets

Im DMX Diagramm 3.1 ist der Aufbau des DMX-Pakets dargestellt. Die einzelnen Elemente werden in den folgenden Unterabschnitten genauer erläutert.

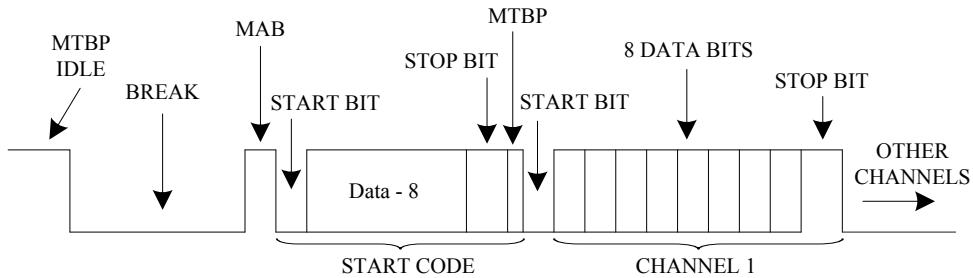


FIGURE 3.1: DMX Paketdiagramm [13, S.107]

Das Paket beginnt mit einem Pause Signal, welches für mindestens 92 µs gehalten werden muss [6, S.18]. Eine typische Länge beträgt 176 µs [6, S.18]. Nach dem Pause Signal folgt die *Mark After Break (MAB)*. Die Mindestlänge des *MAB* Signals beträgt 12 µs [6, p.18].

Nach dem Break Signal werden bis zu 513 Bytes übertragen. Der erste Byte ist der *Startcode*. Das DMX-Paket beginnt mit einem *Startcode* von 0. DMX wurde als unidirektionale Kommunikation entwickelt. Es gibt jedoch DMX-Erweiterungen wie z.B. RDM, die eine bidirektionale Kommunikation zwischen DMX Konsole und Leuchte ermöglichen. Ein RDM Signal kann beispielsweise einen anderen *Startcode* enthalten. Das Break- und *MAB* Signal zusammen mit dem *Startcode* wird als Reset Sequenz bezeichnet. Die Reset Sequenz wird benötigt, um den Anfang eines Datenpakets zu beschreiben und die verschiedenen Datenpakete voneinander zu unterscheiden.

Nach der Reset-Sequenz folgen die Kanäle des DMX-Protokolls. Es können zwischen 1 und 512 Kanäle übertragen werden. Jeder Kanal wird von einem Byte repräsentiert. Daher kann jeder Kanal einen Wert zwischen 0 und 255 annehmen.

3.3. Zeitdiagramm

Im DMX Paket Diagramm 3.2 ist die genaue Struktur des DMX Protokolls veranschaulicht.

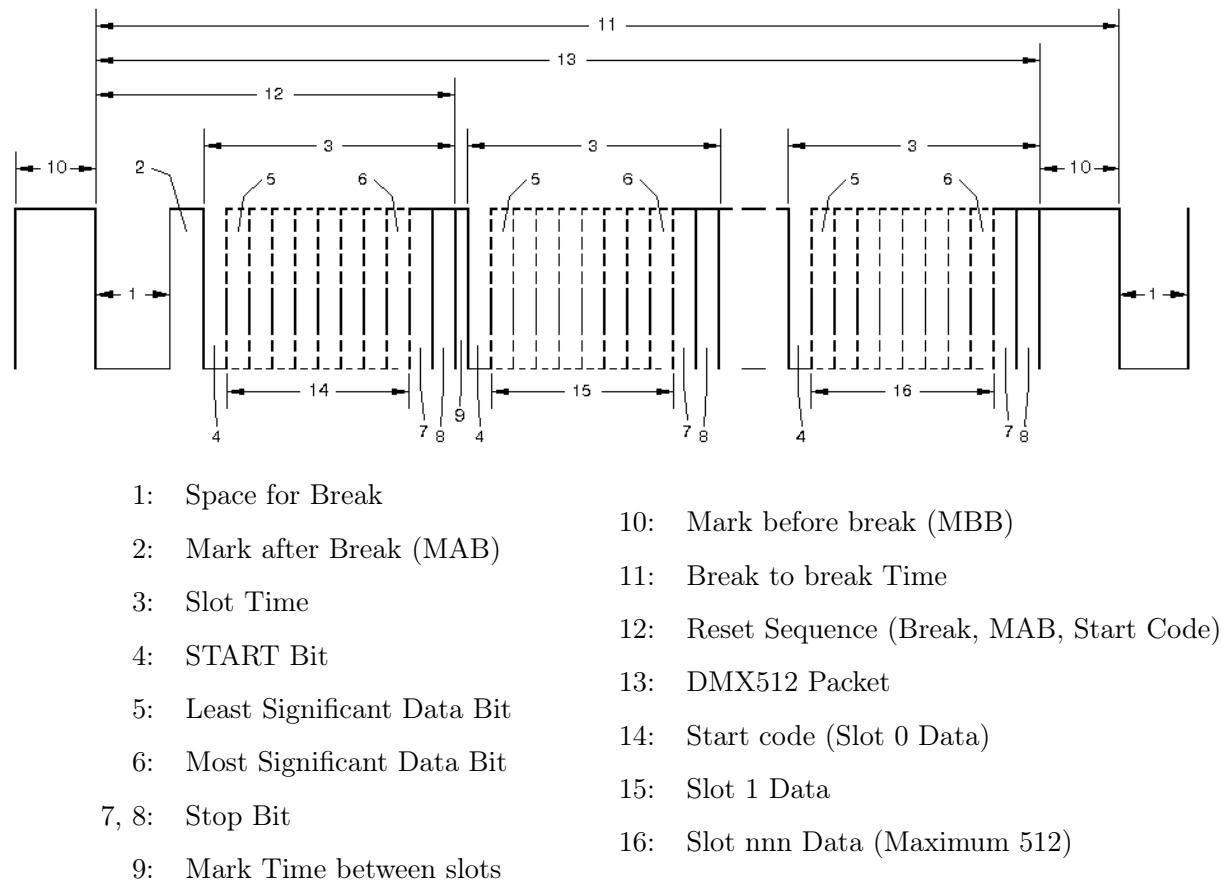


FIGURE 3.2: DMX Paket [6, S.17]

Die Zeitabstände, die im DMX-Protokoll verwendet werden, sind in der Tabelle 3.1 aufgeschlüsselt. Die Zeitabstände sind detailliert aufgeführt und

zeigen die minimalen, typischen und maximalen erlaubten Zeitabstände innerhalb des DMX Standards.

Nr.	Beschreibung	Min	Typ	Max	Einheit
-	Bit Rate	245	250	255	$\frac{\text{kbit}}{\text{s}}$
-	Bit Time	3.92	4	4.08	μs
-	Minimum Update Time for 513 slots	-	22.7	-	ms
-	Maximum Refresh Rate for 513 slots	-	44	-	$\frac{\text{Aktualisierungen}}{\text{s}}$
1	Space for Break	92	176	-	μs
2	Mark After Break (MAB)	12 -	-	- < 1.00	$\mu\text{s} \mid \text{s}$
9	Mark Time between slots	0	-	< 1.00	s
10	Mark Before BREAK (MBB)	0	-	< 1.00	s
11	Break to Break Time	1204 -	-	- 1.00	$\mu\text{s} \mid \text{s}$
13	DMX512 Packet	1204 -	-	- 1.00	$\mu\text{s} \mid \text{s}$

TABLE 3.1: Zeitdiagramm [6, S.18]

Es ist wichtig, diese Werte zu berücksichtigen, um sicherzustellen, dass die DMX Übertragung korrekt empfangen werden kann. Wird dies nicht berücksichtigt, kann es passieren, dass einige Geräte in der DMX-Kette die Daten nicht decodieren können.

Zusammenfassend basiert das DMX-Protokoll auf dem UART-Protokoll und verwendet eine Reihe von Zeitabständen, welche die Kommunikation zwischen DMX Geräten definiert. Es ist entscheidend, diese Elemente und ihre jeweiligen Zeitanforderungen zu verstehen, um ein korrekt funktionierendes DMX-System aufzubauen und zu betreiben. Diese Zeitabstände werden in dem gebauten DMX Analysator auch genauer untersucht.

4. Hardware

Der Kernpunkt der Bachelorarbeit ist die Analyse des DMX-Signals. Das Projekt wird für die Lehre eingesetzt, um das DMX Protokoll anschaulich und intuitiv zu erklären. Die Struktur des DMX Paketes soll daher heruntergebrochen und verständlich dargestellt werden.

Für die Signalverarbeitung und Bildschirmausgabe wird ein Raspberry Pi 4 in der 8 GB RAM Variante verwendet. Der Raspberry Pi 4, kann mithilfe einer Stiftleiste leicht mit externer Hardware kommunizieren.

Eine LED Matrix-Anzeige zeigt, und stellt das verarbeitete Signal dar. Es kann zwischen verschiedene Bildschirmanzeigen gewechselt werden, um verschiedene Aspekte des Protokolls genauer zu veranschaulichen. Die LED Matrix wird vom Raspberry Pi gesteuert. Allerdings wird eine Erweiterungsplatine, für den Raspberry Pi benötigt, um die LED Matrix direkt ansprechen zu können.

Das DMX-Signal wird als differenzielles (oder auch symmetrisches) Datensignal übertragen, um eine gute Rauschunterdrückung zu erhalten. Dieses symmetrische Datensignal wird in ein unsymmetrisches Datensignal gewandelt, um das Analysieren des Signals zu vereinfachen. Außerdem empfiehlt der DMX-Standard eine elektrische Isolierung zwischen dem DMX Bus und dem

Rest der elektronischen Komponenten. Eine Platine wurde daher für dieses Projekt angefertigt.

Im Projekt arbeiten mehrere Komponenten zusammen, um das DMX-Signal zu verarbeiten und auf dem Bildschirm auszugeben. Die Elektronik muss von äußerlichen Einwirkungen geschützt werden. Im Projekt wurde dafür ein Gehäuse modelliert und mithilfe eines 3D-Druckers erstellt.

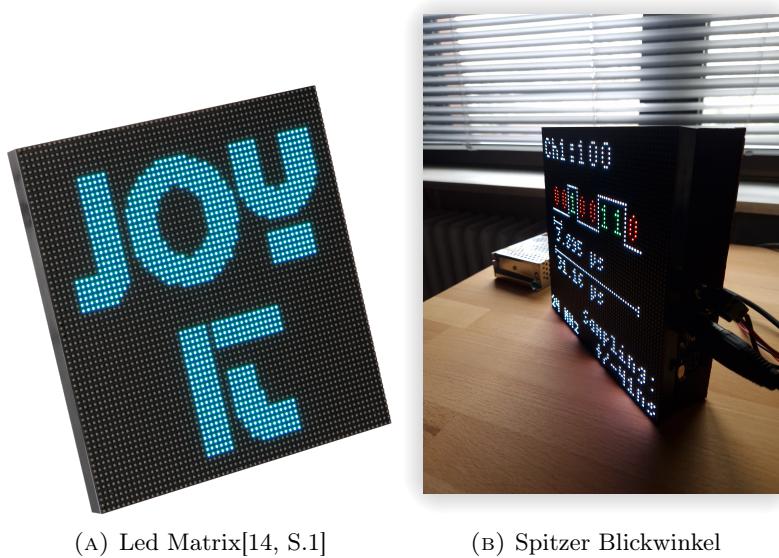
Für das Projekt ist es wichtig, das DMX-Signal zu analysieren und auch die genauen Zeitabstände abzugreifen. Dafür wird ein USB Logikanalysator benutzt. Das Signal kann mit bis zu 24 MHz aufgenommen werden.

4.1. Raspberry Pi

Der Raspberry Pi bietet ausreichend Rechenleistung für die benötigten Anforderungen. Ein Linux-Betriebssystem wird verwendet, um eine Vielzahl von bestehenden Bibliotheken zu benutzen. Es gibt bereits Bibliotheken für die Signalverarbeitung und für die Bildschirmausgabe, die im Projekt eingebunden werden können.

4.2. LED-Matrix

Für die Darstellung des Signals wird eine 64x64 LED Matrix 4.1 verwendet. Der Bildschirm kann gut aus mehreren Blickwinkeln betrachtet werden und ist groß genug, um ihn auch aus mehreren Metern Abstand gut sehen können. Dies ist erforderlich, da mehrere Studierende gleichzeitig auf den Bildschirm schauen werden.



(A) Led Matrix[14, S.1]

(B) Spitzer Blickwinkel

FIGURE 4.1: Gehäuse

Ein gepuffertes Signal für den Bildschirm ist wünschenswert, um ein mögliches Zerreißen des Bildes auf der LED Matrix zu vermeiden. Dafür wird eine handelsübliche Platine 4.2 für den Raspberry Pi verwendet. Der Raspberry Pi kann mithilfe der Platine bis zu 4 Bildschirme hintereinander schalten und bis zu 3 Bildschirme parallel steuern. Daher können theoretisch bis zu 12 Bildschirme gleichzeitig angesteuert werden.



FIGURE 4.2: Raspberry Pi LED Matrix Erweiterungsplatine

[14, S.1]

4.3. Platine DMX Breakout

Die DMX Spezifikation empfiehlt eine galvanische Trennung vom DMX Bus und dem Rest der Platine[6, S.22]. Die DMX Kabel können mehrere 100 Meter lange Signale weiterleiten. Elektrostatische Aufladungen, können die maximale spezifizierten Spannungen übersteigen. Eine elektronische Isolierung 4.3, des Busses zu anderen Komponenten ist vorteilhaft, um mögliche Schäden zu minimieren.

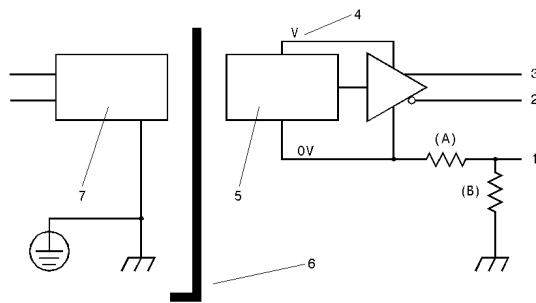


FIGURE 4.3: Zeitdiagramm [6, S.22]

Das Datensignal besteht aus einem symmetrischen Kabelpärchen. Dies hat den Vorteil, dass elektromagnetische Störeinflüsse, einen deutlich kleineren Einfluss auf die Übertragungsqualität haben. Der benutzte Logikanalysator benötigt, jedoch ein einzelnes, nicht symmetrisches Datensignal.

Für das Projekt wurde dafür eine eigene Platine entwickelt. Im Schaltplan 4.4 ist der Aufbau der Platine aufgeschlüsselt.

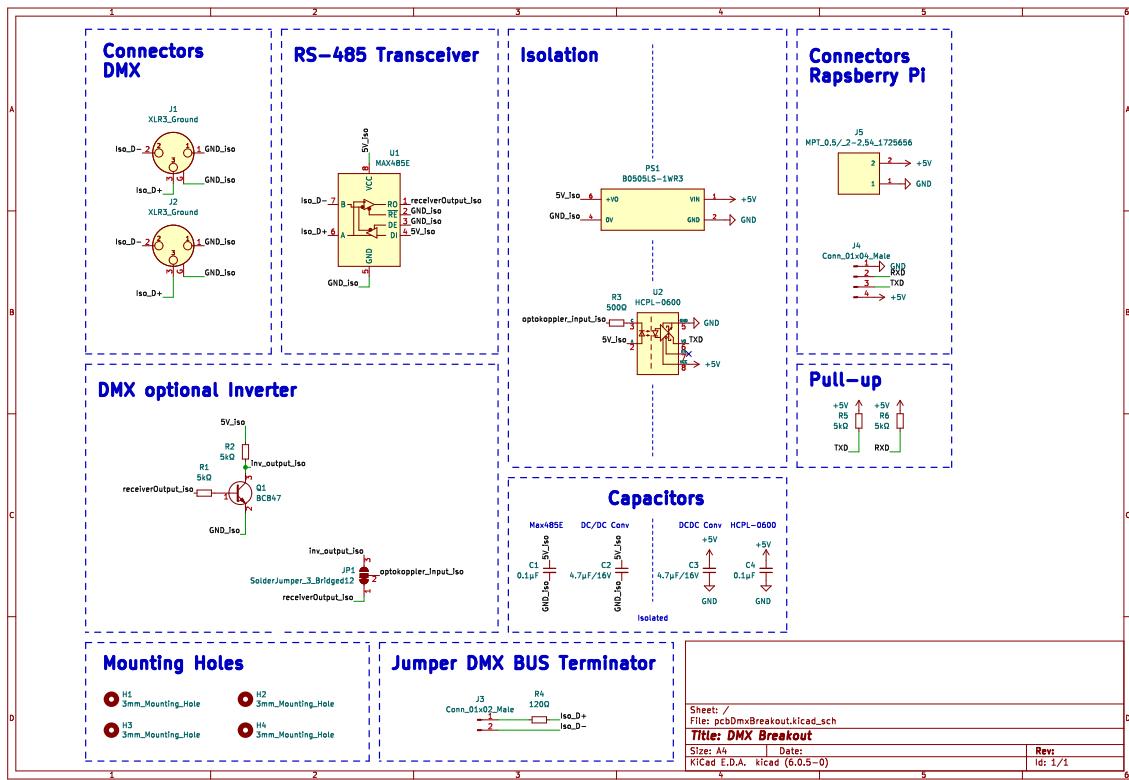


FIGURE 4.4: Schaltplan: DMX Platine

Auf der linken Seite befinden sich die Eingänge. Das Signal wird in der Mitte verarbeitet und galvanisch getrennt. Auf rechten Seiten des Schaltplanes befinden sich die Ausgangssignale. Das symmetrische Signal wird mithilfe eines *MAX485E RS-485 Transceiver* zu einem unsymmetrischen Signal gewandelt. Ein *HCPL-0600 Optokoppler* wird verwendet, um die galvanische Trennung des Datensignals zu erreichen. Der blaue, gepunktete Strich in der Mitte des Schaltplans deutet auf die galvanische Trennung hin.

Die Belegung des XLR-Steckers für das DMX Datensignal, ist nicht bei allen Herstellern gleich. Der Hersteller Martin Professionell hat bei alten Geräten vor dem Baujahr 2000 die symmetrischen Datensignale "DMX+" und "DMX-" vertauscht. Auch wenn nur noch wenige Geräte die Datensignale

vertauschen, ist es trotzdem eine gute Vorgehensweise, auch diesen Fall zu berücksichtigen. Dafür befindet sich ein DMX Inverter auf der Platine, der durch eine Lötbrücke, ein- oder ausgeschaltet werden kann.

Das Platinendesign ist im Aufbau 4.5 gezeigt.

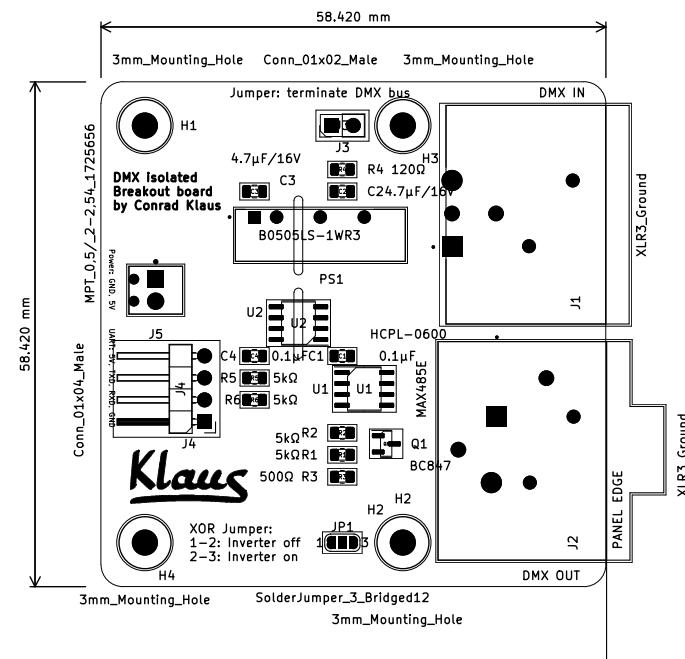


FIGURE 4.5: Aufbau: DMX Platine

Die Platine 4.5 benötigt keine besonders starke elektromagnetisches Rauschunterdrückung. Trotzdem ist es empfehlenswert, gängige Vorgehensweisen einzuhalten, wenn diese keine wesentlichen Extrakosten verursachen. Der Hersteller der Platinen, hatte ungefähr den gleichen Preis (bei dieser Platinengröße) für eine 2- oder 4-schichtige Platine. Bei einer 4-schichtigen Platine können zwei Schichten für die Masse genutzt werden, um eine dritte Schicht vor elektromagnetisches Rauschen zu schützen. Diese 3. Schicht wird zwischen den beiden Schichten mit der Masse eingeführt und befindet sich dadurch in einem Fahrradeschen Käfig. Die Platine basiert daher auf einem 4-schichtigen Entwurf, um das Datensignal des DMX Busses rauschfrei zu verarbeiten.

Die 3D gerenderte Platine 4.6 zeigt den Aufbau mit den Komponenten.

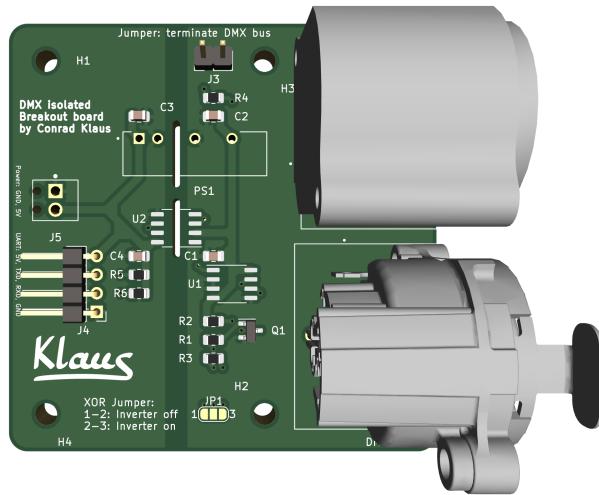


FIGURE 4.6: Aufbau mit Komponenten: DMX Platine

Die elektrische Isolierung ist gut sichtbar. Alle Schichten sind in der Mitte unterbrochen. Der Optokoppler und der DC-DC Konvertierer, dienen für die Signalübertragung und Stromübertragung zwischen den isolierten Platinenteilen. Unter den integrierten Schaltkreisen sind zusätzlich Aussparungen in der Platine eingebohrt, um die elektronische Isolierung zu verstärken.

4.4. Logikanalysator

Die Signalverarbeitung wird mithilfe eines handelsüblichen Logikanalysators 4.7 umgesetzt.



FIGURE 4.7: Logikanalysator: MCU123 Saleae Logic Klone

Die Vorgaben, für das Projekt, sehen vor, dass nicht nur der logische Inhalt des DMX Datenpaket entschlüsselt wird, sondern auch die exakten Zeitabstände zwischen den Signalfanken ausgelesen werden.

Neben der Verwendung des Logikanalysators wurden auch noch andere Optionen betrachtet: Die CPU könnte genutzt werden, um das Signal schnell hintereinander auszulesen und so das Signal aufzunehmen. Das verwendete Linux-Betriebssystem ist jedoch kein Echtzeitbetriebssystem. Das bedeutet, dass nicht garantiert werden kann, dass das Signal kontinuierlich ausgelesen wird, da das Betriebssystem zu jedem Zeitpunkt ein anderes Programm, die CPU zuweisen kann. Eine Möglichkeit, um dies zu umgehen, wäre es den Prozess über das Betriebssystem auf einem Kern zu fesseln. Tatsächlich macht dies der Treiber für die Bildschirmsteuerung bereits. Noch einen weiteren Kern nur für das Auslesen, des Signals zu blockieren, würde die Geschwindigkeit des Raspberry Pi stark verlangsamen. Der Raspberry Pi 4 hat insgesamt 4 Kerne. Die Hälfte der CPU Kerne zu blockieren, ist sehr Ressourcen verschwenderisch.

Eine weitere Überlegung war es den Raspberry Pi Pico zu verwenden: Der Raspberry Pi Pico, ist im Gegensatz zum Raspberry Pi 4 ein Mikrocontroller. Dieser besitzt eine programmierbare Ein- und Ausgangs Peripherie (*Programmable Input Output - PIO*). Diese Peripherie kann genutzt werden, um Signale per Hardware in Echtzeit zu verarbeiten. Eine Möglichkeit wäre es, die Bits in den RAM zu schreiben, und diese dann im Programm auszulesen [15, S. 43]. Auf dem Pico läuft jedoch kein richtiges Betriebssystem, da er ein Mikrocontroller ist. Daher würden viele Bibliotheken, wie z.B. der

Displaytreiber, der Signaldecoder und viele Rust Bibliotheken nicht mehr funktionieren. Daher ist auch diese Option nicht geeignet.

Auch würde überprüft, ob die UART Schnittstelle des Raspberry Pi 4 verwendet werden kann, weil das DMX Signal im Wesentlichen aus einem UART Signal besteht. Für die Markierung des Anfanges eines DMX Paketes wird, das UART Protokoll jedoch abgewandelt, um ein *Break* und ein *MarkAfter-Break* Signal einzubinden. Der Raspberry Pi kommt leider nicht mit dieser Abwandlung zurecht[12, S.20]. Die UART Schnittstelle, bietet auch keine Möglichkeit, um den zeitlichen Ablauf des Signals nachzuvollziehen. Die UART Schnittstelle kann deswegen auch nicht für die Signalabtastung verwendet werden.

Eine weitere Möglichkeit, ist die SPI Schnittstelle des Raspberry Pi auszunutzen. Dies wurde für diese Arbeit auch getestet. Das Signal konnte mit dem Raspberry Pi 1 mit bis zu 2 MHz aufgenommen werden [12][30].

Diese Option ließ aber einige Fragen auf. Das DMX Signal konnte oft nicht komplett aufgenommen. Möchte man immer das komplette DMX Signal aufnehmen, ist nur noch eine Abtastrate von 1 MHz möglich. Der Raspberry Pi 4 kann insgesamt 65536 (2^{16}) Abtastungen aufnehmen. Ein komplettes DMX Signal ist min. 22,624 ms lang[16]. Wenn wir immer ein komplettes Signal aufnehmen wollen, müssen wir $22.624 \text{ ms} * 2 \approx 50 \text{ ms}$ aufnehmen.

Berechnen wir nun die maximale Aufnahmefrequenz für den Raspberry Pi ist dies:

$$\frac{65\,536 \text{ Abtastungen}}{50 \text{ ms}} \approx 1.3 \text{ MHz}$$

Die zeitliche Auflösung beträgt daher:

$$\frac{1}{1.3 \text{ MHz}} \approx 760 \text{ ns}$$

Das DMX Protokoll besteht überwiegend aus dem UART Protokoll mit einer Übertragungsgeschwindigkeit von $250 \frac{\text{kbit}}{\text{s}}$. Das bedeutet die Übertragungslänge eines Bits beträgt 4 us [16].

Mit unserer Abtastrate können wir ca. 5 Abtastungen pro Bit aufnehmen:

$$\frac{4 \mu\text{s}}{760 \text{ ns}} \approx 5$$

Für die genaue zeitliche Analyse sind jedoch mehr Abtastungen pro Bit wünschenswert. Deswegen ist auch diese Option nicht ausreichend. Außerdem, wäre diese Lösung, spezifisch für den Raspberry Pi 4. Ältere oder zukünftige Modelle könnten andere Hardwarekonfigurationen benötigen. Das Programm würde dann bei diesen Modellen ggf. nicht laufen.

Der USB Logik Analysator kann bis zu 24MHz aufnehmen. Diese Auflösung ist für unseren Zweck optimal. Die zeitliche Auflösung beträgt:

$$\frac{1}{24 \text{ MHz}} \approx 42 \text{ ns}$$

Das bedeutet wir können ca. 96 Samples pro Bit aufnehmen:

$$\frac{4 \mu\text{s}}{42 \text{ ns}} \approx 96$$

Diese Auflösung ist gut geeignet.

4.5. Gehäuse

Das Projekt soll mobil, einfach aufzubauen und robust gegenüber äußerlichen Einwirkungen sein. Im Idealfall kann die verwendete Hardware direkt ins Gehäuse integriert werden. Für dieses Projekt wurde das Gehäuse daher in einer CAD Software modelliert und anschließend mithilfe eines 3D-Druckers produziert.

Die gerenderte Seiten- und Topansicht 4.8 zeigt den Aufbau des Gehäuses ohne die Abdeckung.

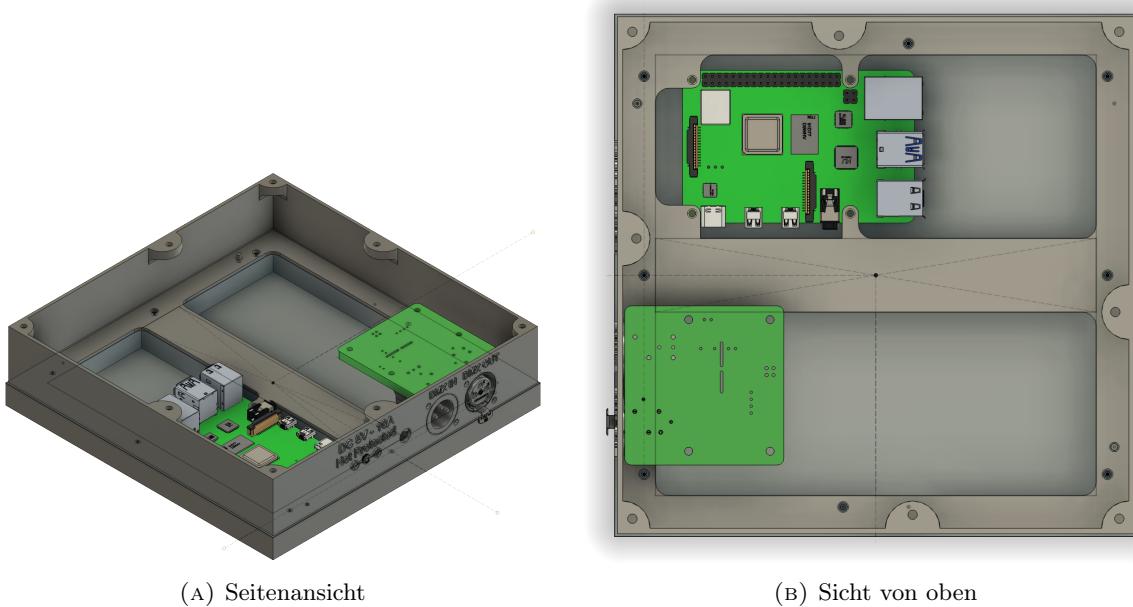
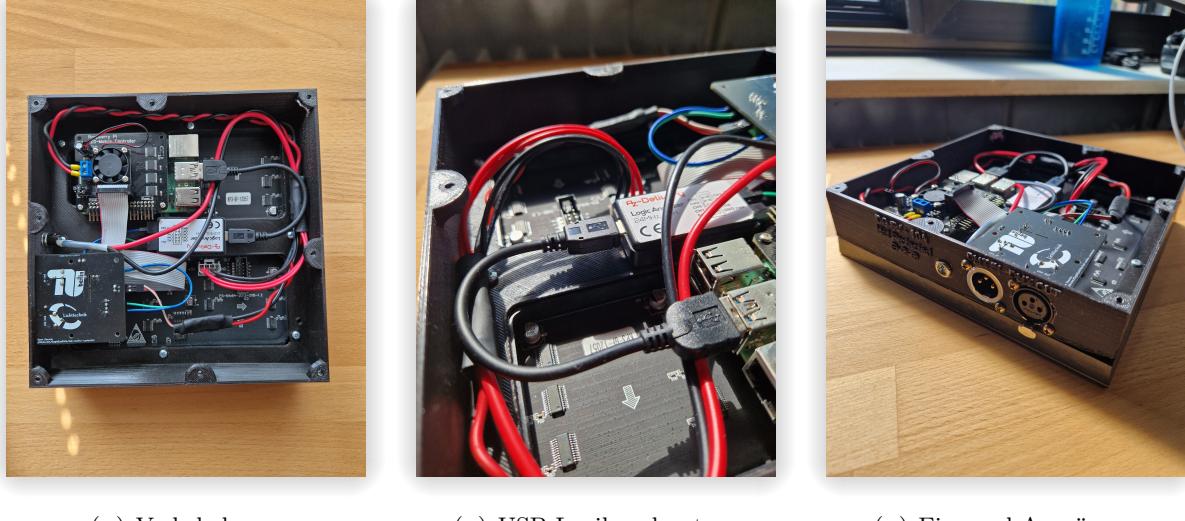


FIGURE 4.8: DMX Analysator Gehäuse

Die LED Matrix befindet sich in der Seitenansicht 4.8a unter dem Gehäuse. Die LED Matrix kann direkt an das Gehäuse geschraubt werden, um einen nahtlosen Übergang zwischen Bildschirm und Gehäuse zu erzeugen. Die exakten CAD-Zeichnungen [17, S.1] der LED Matrix haben dies ermöglicht.

Auch der Raspberry Pi kann direkt ins Gehäuse verschraubt werden. Der Raspberry Pi schwebt so in der Aussparung der LED Matrix. Das Gehäuse ist dadurch etwas schmäler. Die DMX Platine kann direkt in der Seitenwand verschraubt werden. Alle Datensignale und auch die Steckverbindungen für die Stromversorgung, befinden sich auf einer Seite, um den Anschluss des DMX Analysators möglichst einfach zu gestalten. Neben dem Stromanschluss befinden sich auf der Seitenwand auch die benötigte Spannung und Stromstärke sowie eine Beschreibung der Polarität des Stromanschlusses. Das Gehäuse ist breit genug, damit es ohne externe Stützen stabil stehen kann.

Die CAD-Zeichnungen beinhaltet nicht die Verkabelung der einzelnen Komponenten. Die Abbildung 4.9 zeigt den Strom- und Datenkabelaufbau.



(A) Verkabelung

(B) USB Logikanalysator

(C) Ein- und Ausgänge

FIGURE 4.9: Verkabelung

Für die Modellierung wurde Fusion360 benutzt. Fusion 360 verwendet einen parametrische Modellierungsansatz. Die Modellierung ist dabei abhängig von vorher definierten Merkmalen und Einschränkungen. Dabei können

Skizzen mithilfe mathematischer Beziehungen erstellt werden und ein 3 dimensionales Model aufgebaut werden. Der Vorteil der parametrischen Modellierung ist, dass definierte Merkmale (wie z.B. die Wanddicke), im späteren Verlauf verändert werden können, und alle darauf aufbauenden Merkmale neu errechnet werden.

Das Gehäuse wird von einer verschraubbaren Abdeckung 4.10c von äußerlichen Einflüssen geschützt. Das zusammengesetzte Projekt mit der LED Matrix und der DMX Gehäuse Abdeckung ist der Abbildung 4.10 zu sehen.

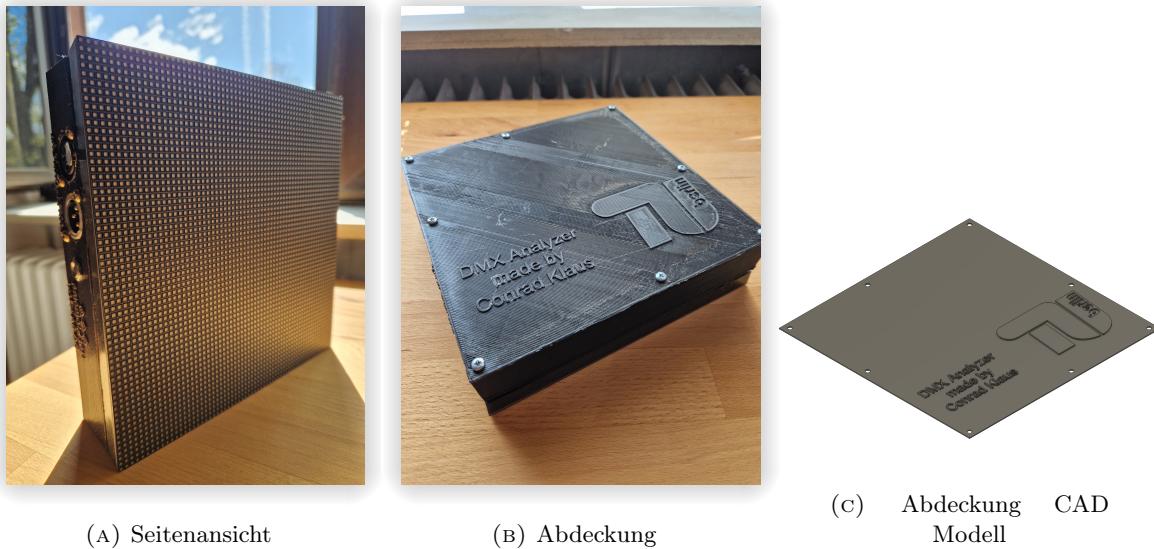


FIGURE 4.10: Gehäuse

5. Software

Die Hardware alleine, reicht nicht aus, um den Bildschirm mit den analysierten DMX Paketen zu füttern. Die Hardware gibt uns nur die rohen aufgenommenen Daten und eine physische Verbindung zwischen Bildschirm und dem Raspberry Pi. In dem Projekt wurde ein Programm geschrieben, um beide Elemente zu verbinden. Die Bachelorarbeit beschreibt auch, die Motivation, hinter den Entscheidungen der verwendeten Softwarelösungen. Die Auswahl der verwendeten Technologien wird im Folgenden beschrieben.

5.1. Verwendete Technologien

Als Basis wird das Linux OS für den Raspberry Pi verwendet: Raspberry Pi OS.

Ein aufgenommenes DMX Paket mit einer Abtastrate von 24Mhz umfasst 1.2 Millionen Abtastpunkte:

$$50 \text{ ms} \cdot 24 \cdot 10^6 \frac{\text{Abtastungen}}{\text{s}} \approx 1.2 \cdot 10^6 \text{ Abtastungen}$$

Die Verarbeitung des DMX Signales muss daher schnell erfolgen. Auch die Bildschirmsteuerung muss möglichst schnell laufen, damit der Bildschirm

auch Animationen flüssig darstellen kann. Außerdem muss das Programm direkt mit dem USB Logikanalysator kommunizieren. Die verwendete Programmiersprache muss also schnell sein, direkt mit Hardware umgehen können und direkt mit dem Betriebssystem kommunizieren können.

Die Programmiersprache sollte daher eine kompilierte Sprache und keine interpretierte Sprache sein. Bei einer kompilierten Sprache wird der Quellcode bereits vor dem Ausführen in Maschinensprache übersetzt. Eine interpretierte Sprache interpretiert den Quellcode erst während der Ausführung und ist daher generell langsamer.

Programme mit diesen Anforderungen wurden früher oft in C, C++ oder vergleichbaren Programmiersprachen geschrieben. Seit einigen Jahren gewinnt eine neue Programmiersprache deutlich an Popularität: Rust. Für dieses Projekt, ist der Großteil der Logik in Rust geschrieben. Für die Kommunikation mit dem USB Logikanalysator, die Analysierung des DMX Paketes und die Ansteuerung der LED Matrix werden an bereits existierenden Lösungen angeknüpft. Diese Interaktion kann mithilfe von externen Bibliotheken umgesetzt werden.

5.2. Vergleich: Rust vs C

Rust revolutioniert die Memory management. In der Vergangenheit wurde der Arbeitsspeicher entweder direkt vom Programmierer mit malloc (in c) reserviert und freigeben oder von einem Garbage collector (z.B. Java) gerichtet. Garbage Collectors sind langsamer und kümmern sich während der Laufzeit um den Speicher. Der Programmfluss eines Programmes kann also

während der Laufzeit unterbrochen werden, um den Speicher zu optimieren. Der Programmierer hat darauf keinen Einfluss. Dies ist für Zeitlich wichtige Abläufe ein Problem. In C hat der Progammierbar volle Kontrolle über den Speicher. Jedoch ist Speichermanagent sehr Fehler anfällig. Und führt oft zu Sicherheitslücken, nicht wieder freigeebner Speicher und zu Programmabstüzen von dereferenizerten dangling Pointers.

Rust, ist genau wie C sehr schnell, löst jedoch das Speicherproblem auf einer sehr elegante Art und Weise. Rust führt die Idee von Ownership und Borrowing ein. Der benutzte Speicher wird dabei immer von genau einer Variable besitzt. Fällt diese Variable out of Scope, so kann der Speicher ohne Probleme wieder frei gegeben werden. Um den Progamierwer mehr flexibilait zu geben, können der Speicher einer Variable auch an eine andere ausgleihen werden, oder der Besitzt weitergegeben werden. Die Regeln sind so definiert, dass der Speicher zu jeden Zeitpunkt auf genau eine Variable zurück geführt werden kann. Somit kann Rust bereits zur Kompilezeit den Speicherverbrauch mit einbeziehen und Fehler bereits zur Kompiletime erkennen. Segmentation Faults können so in reihnem Rust nicht mehr auftreten.

Möchte der Progammeirer in Rust, aber z.B. direkt mit der Hardware kommunizieren und den Status der Hardware auslesen, kann der Kompiler nicht zur Kompilezeit, also vor der Laufzeit voraussagen, welchen Zustand die Hardware haben wird. Für dises und ähnliche Scenaires unterstützt Rust ein unsafe Feature. Ist Code innerhalb eines unsafe blocks, werden viele Kompilerchecks deaktiviert, um den Progammier die benötigte flexiblität zu geben. In Rust sind diese Blöcke explizit gekennzeichtent. Dies erleichtert das Debugging bei einer Fehlersuche. In C ist alles unsicher.

Rust bietet aber noch viele weitere Vorteile im Vergleich zu C. In Rust ist die einbindung von externen Libaries deutlich leichter als in C. Auch unterstützt Rust eine vielzahl von Programmierparadigmen wie z.B. funktionale Programmierung, die in C nur mit Hacks umsetzbar sind. Rust ist Typsicher. C nicht. Das führt dazu das Fehler in Rust bereits zur Kompilzeit (als vor der Programmausführung) entdeckt werden. Bei C entstehen keine Fehler zur Compilezeit. Jedoch führen diese Fehler entweder zu undefiereten Verhalten des Programms bzw. eines Programmabsturz.

Es existieren viele alte Projekt die in C geschrieben sind. Um den Wechsel von C auf Rust zu vereinfachen, bietet Rust ein einfaches Interface an, um C code innerhalb von Rust und anderherum auszuführen. So kann c legacy code, trotzdem mit Rust verwendet werden.

5.3. Bildschirm Treiber

Für die Steuerung des Bildschirmes auf dem Pi wird die C Bibliothek *rpi rgb led matrix* [18] benutzt. Es gab bereits eine von der Gemeinschaft bereitgestellte Rust Schnittstelle für diese C Bibliothek. Die grafischen Bausteinelemente wie z.B. Text, Linien und andere Formen werden mithilfe der Bibliothek *embedded graphics* [19] umgesetzt. Dadurch muss nicht jeder Pixel einzeln gesetzt werden. Es können verschiedene Bausteinelemente abstrahiert auf den Bildschirm gebracht werden.

5.4. Logikanalysator

Für die Kommunikation mit dem Logikanalysator wird die Bibliothek *sigrok* [20] verwendet. Die Bibliothek ist in C geschrieben. Leider gab es keine geeignete Rust Schnittstelle, um mit dieser Bibliothek zu interagieren. Daher habe ich meine eigene C Bibliothek geschrieben, welche die *sigrok* C Bibliothek verwendet, um mit dem USB Logikanalysator zu kommunizieren. Für meine eigene C Bibliothek habe ich dann ein Interface aufgebaut, um sie aus meinem Rust Quellcode heraus aufzurufen.

Die empfangenen Daten vom USB Logikanalysator müssen jedoch noch verarbeitet werden. Dafür wird eine weitere Bibliothek verwendet: *ibsigrokDecode*[21]. Diese Bibliothek wurde nicht für Echtzeit Analysen ausgelegt. Die Bibliothek ist in Python geschrieben. Python ist eine interpretierte Sprache und daher langsamer. Bei der Anzahl der Abtastpunkte, die bei 24 Mhz aufgenommen werden, kommt der Raspberry Pi an seine Grenzen. Der Raspberry Pi braucht ungefähr 4 Sekunden um ein DMX Paket zu analysieren.

Mein geschriebenes Programm unterstützt verschiedene Abtastraten. Ist eine schnelleres und reaktives Verhalten gewünscht, und ungenauere Zeitmessungen in Ordnung, kann die Samplingrate bis zu 2 MHz problemlos heruntergestellt werden.

5.5. Entwicklung mithilfe Simulation

Der Raspberry Pi ist im Vergleich zu einem normalen Computer relativ langsam. Das komplette Programm neu auf dem Pi zu kompilieren braucht

etwa 1-2 Minuten. Beim Update von kleinen Quellcode-Veränderungen benötigt der Pi einige Sekunden. Der Entwicklungsprozess wird dadurch erheblich verlangsamt. Es wurden mehrere Ansätze getestet, um den Entwicklungsprozess zu vereinfachen.

Zunächst war die Überlegung das Programm auf einen normalen Computer zu kompilieren und nur das fertige Programm auf den Raspberry Pi zu kopieren und dort auszuführen. Dieses Verfahren nennt sich quer kompilieren (*cross compile*). Dafür ist es erforderlich eine komplette Werkzeugkette (*toolchain*) aufzubauen. Diese Werkzeugkette muss für den verwendeten Raspberry Pi maßgeschneidert werden. Alle Bibliotheken müssen ebenfalls über diese Werkzeugkette kompiliert werden. Am Anfang des Projektes, gab es noch wenige Abhängigkeiten. Die Abhängigkeiten wurden jedoch schnell zu viele, um Werkzeugkette mit geringen Aufwand zu erhalten.

Es wurde auch probiert Docker zu verwenden, um die Abhängigkeiten besser kontrollieren zu können. Das Kompilieren während des Bauens eines Docker Containers war jedoch ebenfalls nicht schnell genug. Jedoch wurden auch nicht alle Konfigurationsmöglichkeiten genauer getestet.

Für das Projekt wurde ein anderer Ansatz gewählt: Simulation.

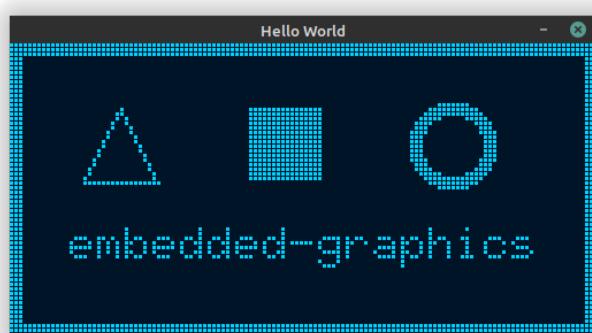


FIGURE 5.1: Simulierte LED Matrix [22]

Die Bibliothek, welche die grafischen Bausteinelemente auf der LED Matrix umsetzt, unterstützt ebenfalls die Simulation eines Bildschirmes auf einem PC [22]. Die simulierte LED Matrix muss etwas anders im Quellcode eingebunden werden, als die Hardware LED Matrix. Der Vorteil jedoch ist, dass die Logik für das Rendern des Bildes exakt gleich ist, und derselbe Quellcode genutzt wird.

Für die Kommunikation mit dem USB Logikanalysator wird die Bibliothek *sigrok* benutzt. *Sigrok* unterstützt nicht nur die Kommunikation mit dem USB Gerät, sondern auch das Abspielen eines vorher mit dem USB Logikanalysator aufgenommenen Signals. So kann ein DMX Paket aufgenommen werden, und beim Entwickeln einfach zum Testen genutzt werden. Beim Entwickeln der Software ist es hilfreich, nicht jedes mal die gesamte Hardware aufzubauen zu müssen. Der Computer, der zum Entwickeln benutzt wird, benötigt daher auch alle Abhängigkeiten und Bibliotheken, die das Programm verwendet. Dies ist bei Linux mit 'sudo apt install libsigrok' möglich. Auf dem Mac ist dies mithilfe 'brew install libsigrok' möglich. Leider hat Windows keinen eigenen Paketmanager. Auf Windows ist dies auch möglich, erfordert aber etwas mehr Arbeit.

Durch die Unterstützung einer Simulation, können andere Entwickler das Programm austesten, ohne selber die Hardware zur Verfügung zu haben. Entwickler, die sich einen genaueren Eindruck vom Projekt machen wollen, können dies mithilfe der Simulation sehr einfach machen.

5.6. Signal Decodierung

Das Signal wird in Echtzeit vom Logikanalysator erfasst. Diese Rohdaten werden dann für die Signaldecodierung mithilfe der *sigrokDecode* Bibliothek decodiert. Die Aufnahme der Rohdaten als auch die Signaldecodierung wird asynchron in einem separaten Ausführungsstrang (*thread*) umgesetzt. Die Bibliothek, welche zum Decodieren verwendet wird, gruppiert jedoch nicht das gesamte DMX Paket, sondern streamt die einzelnen decodierten Ereignisse wie z.B. das *Break* Signal, *Mark After Break (MAB)* oder die einzelnen Bits.

Die asynchron laufenden Ausführungsstränge müssen für den Datenaustausch synchronisiert werden. Für die Kommunikation werden sogenannte Rust Kanäle verwendet.

Das DMX Pakets wird mithilfe eines Zustandsautomaten zusammengefasst:

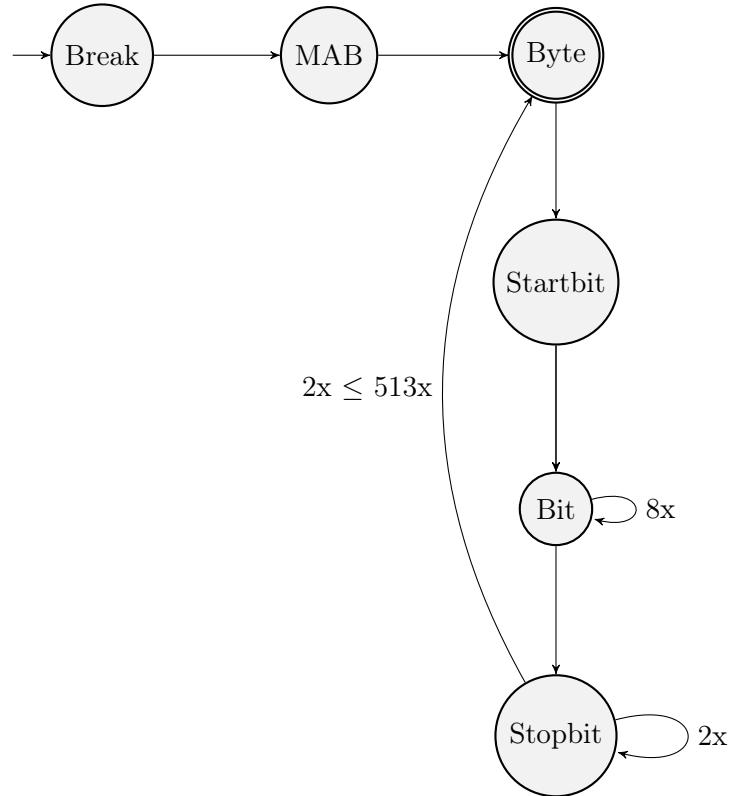


FIGURE 5.2: DMX Zustandsautomat

Der Zustandsautomaten empfängt die einzelnen decodierten Ereignisse, prüft die Reihenfolge und fasst ein komplett empfangenes Paket zusammen. Das Paket fängt mit der *Break* und *Mark After Break (MAB)* an. Danach folgen bis zu 513 Bytes. Der erste Byte ist der *Startcode*. Danach folgt mindestens ein und bis zu 512 weitere Kanäle.

Die Zusammenfassung der DMX Pakete und das Rendern des Bildes werden ebenfalls in separierten Ausführungsstränge berechnet, da beide Berechnungen sich nicht gegenseitig blockieren sollten. Ist ein DMX Paket komplett zusammengefasst, so wird dieses auch wiederum über einen Rust Kanal asynchron zum Ausführungsstrang des Bildschirmrenderers übertragen.

5.7. Signal Darstellung

Das analysierte Signal wird als Wellenform auf dem Bildschirm dargestellt. Aus dem Signal werden kleinere Ausschnitte ausgewählt und auf verschiedenen Bildschirmansichten dargestellt, weil der Bildschirm eine relativ kleine Auflösung aufweist.

5.7.1 Reset Sequenz

Jedes DMX Paket fängt mit der Reset Sequenz an.



FIGURE 5.3: Reset Sequenz

Diese Sequenz besteht aus dem *Break*, *Mark After Break (MAB)* und dem *Startcode*. Der *Startcode* ist bei DMX Paketen immer 0. Jeder Byte, der übertragen wird, enthält einen *Startcode* (in grün) und zwei Stopbits (in rot). Die Timings werden unter der Wellenform dargestellt.

Aufgrund der kleinen Auflösung sind die Zeitabstände nicht maßstabsgetreu. Ein Bit ist eigentlich nur $4\mu\text{s}$ lang. Auf dem Bildschirm sieht er jedoch deutlich länger aus. Am unterem Bildschirm Rand befindet sich die Abtastrate, mit der das Signal aufgenommen wird. Aus der Abtastrate leitet sich auch die Messunsicherheit der Zeitabstände ab.

5.7.2 Kanal 1

Die Dekodierung eines Bytes wird auf einer anderen Bildschirmansicht dargestellt.



FIGURE 5.4: Kanal 1

Die Bits, abgesehen von den Start- und Stopbits, werden auch als Wellenform abgebildet. Der Wert des Kanals ist oben im Bild als Dezimalwert angegeben: 101. In Binär ist dies 0b1100101. Beim Dmx Protokoll wird jedoch das höchstwertigen Bit zuerst versendet. Daher ist die Reihenfolge in der Wellenansicht umgedreht.

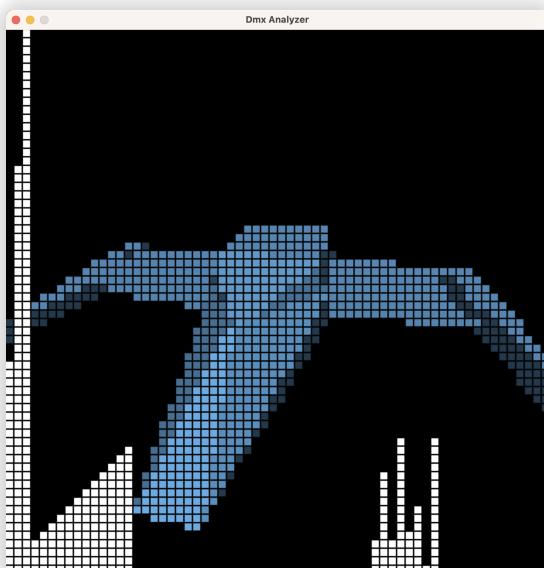
5.7.3 Anwendungsbeispiel

Das Ziel des Projektes ist es, Studierenden das DMX Paket auf einer anschaulichen Art und Weise zu zeigen. Im Idealfall bekommen die Studierenden ein intuitives Gefühl für die Anwendungsmöglichkeiten des Protokolls.

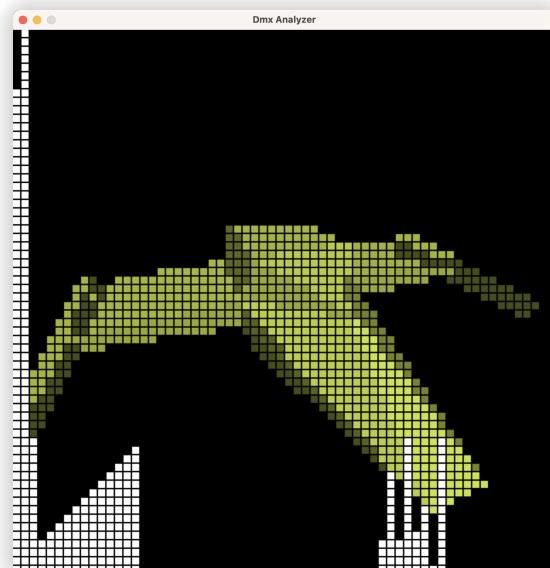
Auf einer Bühne werden die verschiedenen Aspekte der Leuchten über die DMX Kanäle kontrolliert. Ein Kanal kann dabei die Helligkeit oder auch die Farbe der Leuchten oder ähnliches steuern.

In dem Projekt möchte ich den Studierenden auch ein Gefühl mitgeben, wie die Kanäle für die Lichtsteuerung eingesetzt werden können.

Dafür habe ich eine eigene *3D Render Engine* in Rust aufgebaut, ohne die Verwendung von externen Bibliotheken¹.



(A) Farbe: Blau/ Cyan



(B) Farbe: Gelb

FIGURE 5.5: Farbe gesteuert von Kanal 1-3

¹Die Implementierung ist an einer bereits existierender Render Engine angelehnt. Weitere Details sind im Quellcode kommentiert.

Im Hintergrund befindet sich ein rotierendes Raumschiff. Im Vordergrund sind einige weiße Balken zu sehen. Die Balken bilden die ersten 64 Kanäle entlang der x-Achse ab. Der Wertebereich eines Kanals (0-255) wird auf die Höhe des Bildschirms (64 Pixel) abgebildet.

Die Farbe des Raumschiffes kann mithilfe der DMX Kanäle gesteuert werden. Im Beispiel kontrollieren die ersten 3 Kanäle die *RGB* Werte des Raumschiff. Im ersten Bild 5.5a ist der 2. (Grün) und 3. Kanal (Blau) dominierend. Die resultierende Farbe des Raumschiffes ist daher ein Cyan blau. Im zweiten Bild 5.5b dominieren die Kanäle 1 und 2. Daher ist die resultierende Farbe eine Mischung aus Rot und Grün: Gelb.

Zur Demonstrierung sind im Bild 5.5 noch weitere Kanäle angezeigt. Diese Kanäle verändern keine weiteren Eigenschaften des Raumschiffes, und deuten an, dass eine andere Leuchte, die dasselbe Signal erhält, von diesen anderen Kanälen kontrolliert werden könnte.

Die Studierenden sollend dadurch ein Verständnis bekommen, wie die Kanäle die Aspekte einer Leuchte steuern können.

5.8. Ergebnisse

Die simulierten Bildschirmansichten funktionieren ebenfalls auf der echten Hardware. In den Bildern 5.6 sind die verschiedenen Ansichten aufgezeigt:



FIGURE 5.6: Bildschirmansichten auf der Hardware

5.9. Kontrolle DMX Signalanalyse

Das analysierte Signal wird mithilfe eines Oszilloskope überprüft, um die dargestellte Wellenform zu verifizieren. Das Oszilloskop ermittelt den Durchschnitt von 4 Aufnahmen, um das rauschfreie Signal abzubilden.

Die analysierten Zeitabstände zeigen einen Unterschied zwischen Logik Analysator und dem Oszilloskope.

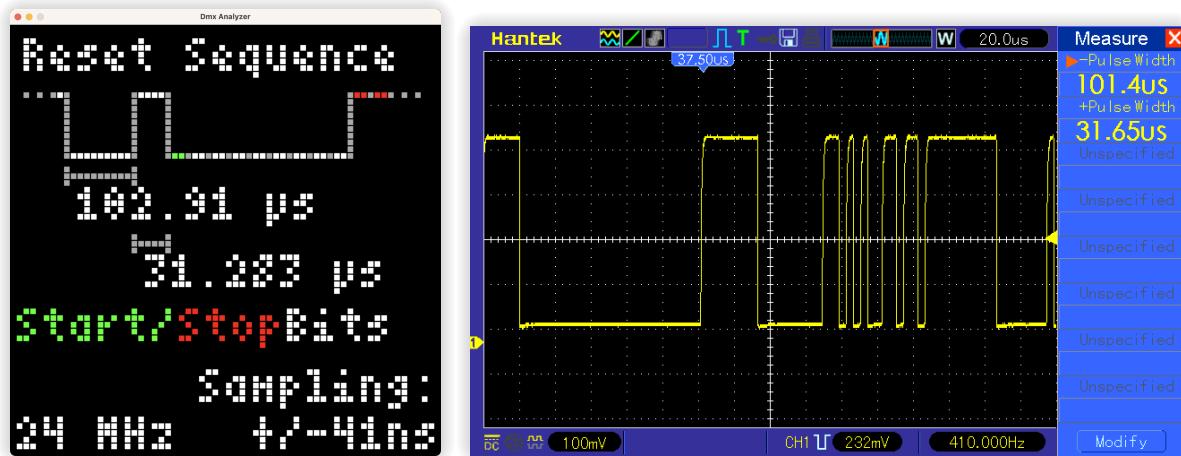
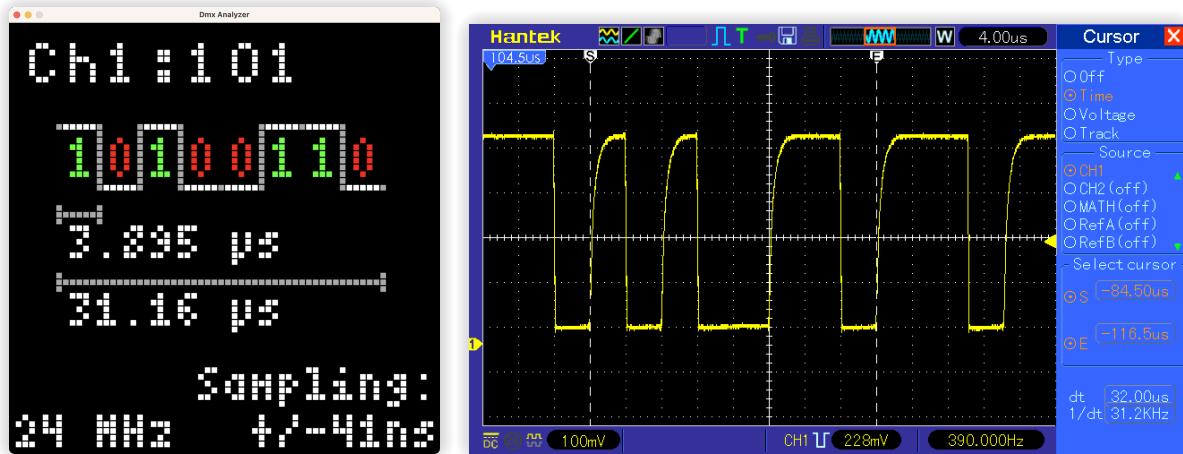


FIGURE 5.7: Vergleich Projekt und Oszilloskope: Reset Sequenz

Die analysierten Zeitabstände zeigen einen Unterschied zwischen dem Logikanalysator und dem Oszilloskope. Die Messunsicherheit des Logikanalysators beträgt 41ns. Die mit dem Oszilloskope gemessene Zeit hat jedoch eine Abweichung größer als die Messunsicherheit.



Bezeichnung:	Logikanalysator	Oszilloskope	ΔZ Zeit
Bit	3.895 μs	4 μs (abgelesen)	0.105 μs
Byte	31.16 μs	32.00 μs (abgelesen)	0.84 μs

FIGURE 5.8: Vergleich Projekt und Oszilloskope: Kanal 1

Auf dem linken Bildschirm, wird aufgrund der kleine Auflösung die Start- und Stopbits nicht dargestellt. Der erste und letzte Bit vom ersten Kanal ist beim Oszilloskope mit dem Mauszeiger markiert. Die dargestellte Bytereihenfolge ist identisch mit der Reihenfolge des Oszilloskope.

Die Zeitmessunsicherheit bei einer Abtastrate von 24 Mhz beträgt 41ns. Die Abweichungen sind oft größer. Beim *Break* Signal ist die Abweichung 1.51μs. Die Anstiegszeit des DMX Signals ist leider auch nicht perfekt.

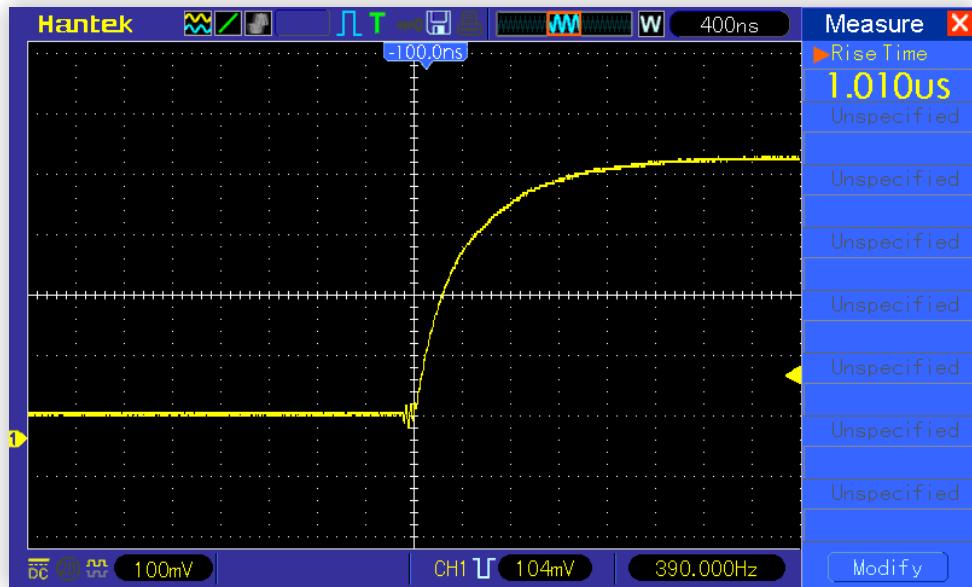


FIGURE 5.9: Rise Time Signal

Die Anstiegszeit beträgt $1.010 \mu\text{s}$. Es ist möglich, dass der Logikanalysator erst bei einem anderen Logikpegelschwellenwert ausschlägt als das Oszilloskop. Der untersuchte Zeitunterschied könnte dadurch erklärt werden.

6. Fazit

Das Projekt wurde exemplarisch einigen Studierenden vorgestellt. Einige der Studierende hatten bereits Vorkenntnisse in der Lichttechnik oder auch in der Elektrotechnik. Die Studierenden mit Vorkenntnissen hatten bereits ein grobes Verständnis, wie ein Datenprotokoll aufgebaut ist und wie die Daten decodiert werden. Der Großteil der Studierenden hatten jedoch keine Vorkenntnisse in der Lichttechnik oder von Steuerprotokollen.

Die Studierenden haben einen guten Eindruck vom Protokoll erhalten, und konnten auch schnell die Limitierungen des Protokolls erkennen. So wurde den meisten schnell bewusst, wie die DMX Kanäle für die Steuerung von Leuchten verwendet werden können. Auch war den Studierenden schnell erkenntlich, warum ein Kanal einen Wertebereich von 0 bis 255 besitzt.

Auch die Studierenden, welche bereits Vorkenntnisse hatten, haben sich für die genauen Zeitabstände innerhalb des Protokolls interessiert. Die analysierten Zeitabstände haben kleine Messunsicherheiten, für dieses Projekt stellt das jedoch kein Problem da.

Der Formfaktor des Projektes ist ebenfalls gut, da er leicht zu transportieren ist, aber auch groß und robust genug ist, um es anderen Studierenden zu demonstrieren.

Es können zwischen verschiedenen Bildschirmansichten gewechselt werden, um verschiedene Aspekte des Protokolls genauer zu demonstrieren. Dadurch ist es noch einfacher das Protokoll zu verstehen.

Das gesamte Projekt ist öffentlich [23] einsehbar, und gut dokumentiert, damit andere Personen die Möglichkeit haben weiter auf diesem Projekt aufzubauen.

Abbildungsverzeichnis

2.1 Beispiele für Lichtszenen [3, S.6]	3
2.2 Verdrahtungsdiagramm [3, S.64]	5
2.3 DMX gesteuerte Lichtszene in einer Berliner Bar	6
2.4 Verdrahtungsdiagramm [5, S.64]	8
2.5 MIDI Licht Integration [9, S. 1]	11
2.6 ZigBee Frequenzüberlappung [10, S.23]	13
3.1 DMX Paketdiagramm [13, S.107]	17
3.2 DMX Paket [6, S.17]	18
4.1 Gehäuse	22
4.2 Raspberry Pi LED Matrix Erweiterungsplatine	22
4.3 Zeitdiagramm [6, S.22]	23
4.4 Schaltplan: DMX Platine	24
4.5 Aufbau: DMX Platine	25
4.6 Aufbau mit Komponenten: DMX Platine	26
4.7 Logikanalysator: MCU123 Saleae Logic Klone	26
4.8 DMX Analysator Gehäuse	30
4.9 Verkabelung	31
4.10 Gehäuse	32
5.1 Simulierte LED Matrix [22]	38
5.2 DMX Zustandsautomat	41
5.3 Reset Sequenz	42
5.4 Kanal 1	43
5.5 Farbe gesteuert von Kanal 1-3	44
5.6 Bildschirmansichten auf der Hardware	46
5.7 Vergleich Projekt und Oszilloskope: Reset Sequenz	47
5.8 Vergleich Projekt und Oszilloskope: Kanal 1	48

5.9 Rise Time Signal	49
--------------------------------	----

Tabellenverzeichnis

2.1 Lichtsteuer Protokolle im Vergleich	14
3.1 Zeitdiagramm [6, S.18]	19

Literaturverzeichnis

- [1] DALI Alliance. Changes to iec 62386 enablingdali-2 certification, Nov. 2017. URL https://www.dali-alliance.org/data/downloadables/3/3/2/diia-pwgtw021b_changes-to-iec-62386-enabling-dali-2.pdf. Aufgerufen am 21-03-2023.
- [2] Heidelberg H. Döhling. Dali-lichtmanagement. Jan. 2002. URL https://www.elektropraktiker.de/ep-2002-01-38-41.pdf?eID=tx_mspdamlinks&dlid=57984&hash=bfc5adb4e67a730481ac07eb8a412063. Aufgerufen am 21-03-2023.
- [3] Tridonic. Changes to iec 62386 enablingdali-2 certification. Mar. 2020. URL https://www.tridonic.com/com/de/download/technical/DALI-Handbuch_de.pdf. Aufgerufen am 21-03-2023.
- [4] Elation Professional. Dmx 101: A dmx 512 handbook. May. 2008. URL https://sitelec.org/download.php?filename=themes/dmx/dmx101_handbook.pdf. Aufgerufen am 21-03-2023.
- [5] eldoLED. How to wire dmx/rdm lighting systems. 2020. URL https://www.soliled.com/wp-content/uploads/2020/06/Learning-Center_Application-note_How-to-wire-DMX-lighting-systems.pdf. Aufgerufen am 21-03-2023.

- [6] ESTA: Entertainment Services and Technology Association. Usitt dmx512-a asynchronous serial digital data transmission standard for controlling lighting equipment and accessories. 2008. URL https://tsp.estas.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2018.pdf. Aufgerufen am 21-03-2023.
- [7] The MIDI Manufacturers Association. The complete midi 1.0 detailed specification. Apr. 2006. URL <http://www.shclemen.com/download/The%20Complete%20MIDI1.0%20Detailed%20Spec.pdf>. Aufgerufen am 21-03-2023.
- [8] The MIDI Manufacturers Association. Midi 1.0 detailed specification. Feb. 1996. URL <https://www.midi.org/specifications/m1-v4-2-1-midi-1-0-detailed-specification-96-1-4/download>. Aufgerufen am 21-03-2023.
- [9] The MIDI Manufacturers Association. Midi visual control specification. Feb. 2011. URL <https://www.midi.org/specifications/midi-visual-control/download>. Aufgerufen am 21-03-2023.
- [10] Daintree Networks. Getting started with zigbee and ieee 802.15.4. Feb. 2008. URL <https://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/Zigbee%20GettingStarted.pdf>. Aufgerufen am 21-03-2023.
- [11] Phillips. Universal asynchronous receiver/transmitter (uart). Aug. 2006. URL <https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf>. Aufgerufen am 18-04-2023.

- [12] Flo Edelmann. Using a raspberry pi as a pc-dmx interface. Nov. 2017. URL <https://github.com/FloEdelmann/bachelor/raw/master/edel17.pdf>. Aufgerufen am 01-04-2023.
- [13] Chang-Hoon Lee Nguyen Manh Hung. Design of bi-directional rdm-dmx512 converter for led lighting control. Jun. 2013. URL https://www.researchgate.net/profile/Chang-Hoon-Lee-2/publication/263633914_Design_of_Bi-directional_RDM-DMX512_Converter_for_LED_Lighting_Control/links/5bcadd0c92851cae21b49813/Design-of-Bi-directional-RDM-DMX512-Converter-for-LED-Lighting-Control.pdf. Aufgerufen am 02-04-2023.
- [14] Joy It. Matrixmodul und erweiterungsplatine. Jun. 2021. URL https://joy-it.net/files/files/Produkte/RB-MatrixCtrl/RB-MatrixCtrl_Anleitung_2021-08-05.pdf. Aufgerufen am 30-03-2023.
- [15] Raspberry Pi Ltd. Raspberry pi pico c/c++ sdk. Mar. 2023. URL <https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf>. Aufgerufen am 18-04-2023.
- [16] Dmx timing. Mar. 2013. URL https://www.thedmxwiki.com/dmx_basics/dmx_timing. Aufgerufen am 22-04-2023.
- [17] adafruit. Cad drawings led matrix. URL https://cdn-shop.adafruit.com/product-files/4732/4732_p3_192X192mm-64x64+pixel.pdf. Aufgerufen am 01-04-2023.
- [18] hzeller. Rpi rgb led matrix. <https://github.com/hzeller/rpi-rgb-led-matrix>. Aufgerufen am 23-04-2023.

- [19] embedded graphics. embedded graphics. <https://github.com/embedded-graphics/embedded-graphics>, . Aufgerufen am 23-04-2023.
- [20] sigrokproject. Libsigrok. <https://github.com/sigrokproject/libsigrok>, . Aufgerufen am 23-04-2023.
- [21] sigrokproject. Libsigrokdecode. <https://github.com/sigrokproject/libsigrokdecode>, . Aufgerufen am 23-04-2023.
- [22] embedded graphics. Embedded graphics simulator. <https://github.com/embedded-graphics/simulator>, . Aufgerufen am 23-04-2023.
- [23] Conrad Klaus. Dmx analyzer. <https://github.com/EagleEyeElite/dmx-analyzer>. Aufgerufen am 23-04-2023.