

Entwicklung eines Aufnahme- und Wiedergabegerätes für DMX-Daten

Bachelorabschlussarbeit

von

Felix Bauer

Im Studiengang B.Eng Elektrotechnik/Informationstechnik
mit Schwerpunkt Mess- und Automatisierungstechnik

an der HAWK Hochschule für angewandte Wissenschaft und Kunst
Hildesheim/Holzminden/Göttingen
Fakultät Ingenieurwissenschaften und Gesundheit in Göttingen



Erstprüfer: Prof. Dr. rer. nat. Roman Grothausmann
Zweitprüfer: Dipl.-Ing. Tobias Bürmann

2. September 2021

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbstständig, ohne fremde Hilfe und nur unter Verwendung der angegebenen Literatur angefertigt habe. Alle fremden, öffentlichen Quellen sind als solche kenntlich gemacht. Mir ist bekannt, dass ich für die Quellen Dritter in dieser Arbeit die Nutzungsrechte zur Verwendung in dieser Arbeit benötige. Weiterhin versichere ich, dass diese Abschlussarbeit noch keiner anderen Prüfungskommission vorgelegen hat.

Göttingen, den Unterschrift
.....

Erklärung zu Nutzungsrechten und Verwertungsrechten^①

Ich bin hiermit einverstanden,
dass von meiner Abschlussarbeit (ggf. nach Ablauf der Sperre) 1 Vervielfältigungsstück erstellt werden kann, um es der Bibliothek der HAWK zur Verfügung zu stellen und Dritten öffentlich zugänglich zu machen.

Ich erkläre, dass Rechte Dritter der Veröffentlichung nicht entgegenstehen.

Göttingen, den Unterschrift
.....

Sperrvermerk der Abschlussarbeit

[] NEIN*

JA / Dauer der Sperre: [] 3 Jahre* [] 5 Jahre*

Göttingen, den Unterschrift
.....

* Zutreffendes bitte ankreuzen

① Dadurch räumen Sie der HAWK ein einfaches, zeitlich unbeschränktes, unentgeltliches Nutzungsrecht nach §§ 15 Abs. 2 Nr. 2, 16, 17, 19a, 31 Abs. 2 UrhG ein.

Inhaltsverzeichnis

1 Einleitung	2
2 Ziel der Arbeit	3
3 Hardware	4
3.1 Schaltungsprinzip	4
3.2 STMicroelectronics STM32F446ZE	5
3.3 Maxim Integrated MAX14945EWE+ RS485 Treiberchip	8
3.4 Encoder	9
3.5 Liquid Crystal Display (LCD)	10
3.6 LED	11
3.7 Spannungsversorgung	13
3.8 Autodesk EAGLE	14
3.9 Platinendesign	15
3.10 Gehäusedesign	17
4 Software	20
4.1 Hardwareabstraktionsschicht HAL (Hardware Abstraction Layer)	20
4.2 STMicroelectronics CubeMX	20
4.3 STMicroelectronics STM32CubeIDE	21
4.4 Taster	21
4.5 Encoder	24
4.6 Aufnahmefunktionen	25
4.6.1 Festlegen der Aufnahmezeit	26
4.6.2 Festlegen des Dateinamens	26
4.6.3 Aufnahme eines DMX-Datenpaketes	27
4.6.4 Speicherung der Aufnahmedaten	29
4.7 Wiedergabefunktion	30
4.7.1 Auswahl der Aufnahmedatei	30
4.7.2 Wiedergabe einer kontinuierlichen Aufnahme	31
4.7.3 Ausgabe eines DMX-Datenpaketes	31
4.8 LCD	33
4.9 SD-Karte (SDIO)	34
4.10 Menü	35
4.11 Einstellungen	37
4.11.1 LCD-Kontrast & LED-Helligkeit	37
4.11.2 NPC ändern	38
4.11.3 Trigger	39
4.11.4 Aufnahme löschen	39
5 Zusammenfassung	40
A Flussdiagramme	44
A.1 Aufnahmefunktion	44
A.2 Wiedergabefunktion	45
A.3 Standard Wiedergabe	46
B CD-Anhang	47

1 Einleitung

2 Ziel der Arbeit

In dieser Arbeit werden die in der vorangegangenen Praxisprojektarbeit erarbeiteten Funktionsprinzipien für die Aufnahme und Wiedergabe von DMX-Daten mit einem Mikrocontroller weiterentwickelt und in ein für den Endbenutzer ausgerichtetes Gerät implementiert. Ziel dabei ist die Herstellung eines Prototyps eines DMX-Aufnahme und -Wiedergabegerätes, welches eine gewisse marktreife darstellt. Zusätzlich zu den Inhalten der Praxisprojektarbeit wird eine umfassende Benutzerschnittstelle entwickelt, mit der dem Benutzer ohne spezielle Vorkenntnisse die Bedienung des Geräts ermöglicht wird. Konkret werden Taster und Drehgeber für die Eingaben des Benutzer, ein Display und LEDs für Rückmeldungen an den Benutzer verwendet. Durch die Zunahme an Funktionität und Anzahl der Komponenten wird außerdem die elektrische Schaltung der Praxisprojektarbeit dementsprechend erweitert und an bereits vorhandenen Stellen optimiert. Für die erweiterte Schaltung wird eine Platine entwickelt, hergestellt und bestückt. Zudem wird ein passendes, kompaktes Gehäuse für die Platine gestaltet und Hergestellt. Damit ist bereits die optische Sichtweise auf eine gewisse Marktreife sichtbar.

Für die neu hinzugekommene Benutzerschnittstelle sind nicht nur die Komponenten wichtig, sondern auch die Software die sie steuert. Dafür wird entsprechender Programmcode entwickelt der dem Benutzer sinnvolle und auswertbare Rückmeldung gibt und ihm eine intuitive Bedienung des Gerätes ermöglichen. Dem Benutzer sollen außerdem Möglichkeiten gegeben sein, gewisse Einstellungen vorzunehmen um die Funktion des Gerätes auf die eigenen Bedürfnisse anzupassen. Ein besonderes Augenmerkt liegt auf der Optimierung der Aufnahme, Wiedergabe und Speicherung der DMX-Daten. Auch hier sollen dem Benutzer möglichst viele Freiheiten eingeräumt werden. Die in der Praxisprojektarbeit erarbeiteten Grundlagen werden dafür verwendet, erweitert und optimiert. Zudem werden die Optimierungsvorschläge der Praxisprojektarbeit berücksichtigt und zum Teil umgesetzt. Für die Entwicklung des Gerätes sind folgende Haupt- und Nebenanforderungen definiert.

Hauptanforderungen:

- Möglichkeit mehrere Aufnahmen anzulegen
- Einstellbare Aufnahmezeit
- Aufnahme eines DMX-Stroms und einzelner DMX-Datenpakete
- Benutzerschnittstelle mit Display und Menüführung
- Maßgeschneidertes Platinendesign
- Gehäuse
- Handliche Größe

Nebenanforderungen:

- Intuitive Benutzerschnittstelle
- Optisch ansprechendes Design des Gehäuses
- Optimierung des Speicherbedarfs auf der SD-Karte
- Möglichkeit die Wiedergabegeschwindigkeit während der aktiven Wiedergabe zu ändern

3 Hardware

In den folgenden Kapiteln wird auf die in dieser Arbeit verwendete Hardware eingegangen. Zunächst wird das grundlegende Prinzip der Schaltung erläutert. Daraufhin werden die Bestandteile bzw. Module im Einzelnen näher erläutert und dessen Funktionsweise und Funktion in der Gesamtschaltung erklärt. Abschließend wird erläutert wie die abstrakte Schaltung in ein Platinendesign überführt wird und wie diese in einem Gehäuse plaziert wird.

3.1 Schaltungsprinzip

In diesem Kapitel wird auf das grundlegende Prinzip der gesamten Schaltung eingegangen. Auf die einzelnen Komponenten und deren benötigte Beschaltung wird in den folgenden Kapiteln eingegangen. Generell wird bei der Entwicklung der Schaltung darauf geachtet, dass möglichst keine fertigen Modulbausteine verwendet werden. Dadurch ergibt sich in der Regel ein Preisvorteil und vereinfacht, in Hinsicht auf eine Serienproduktion, die industrielle Bestückung der Platine. Abbildung 1 zeigt das grundlegende Prinzip der Schaltung. Auf der linken Seite befindet der Benutzer des Gerätes. Er bedient das Gerät durch seine Eingaben und erhält entsprechende Rückmeldungen zurück. Die grüne Fläche spiegelt das Gerät wieder, welches mit der DMX-Quelle, bzw. der DMX-fähigen Lichttechnik, auf der rechten Seite, kommuniziert. Im Folgenden wird auf die Bestandteile des Gerätes eingegangen.

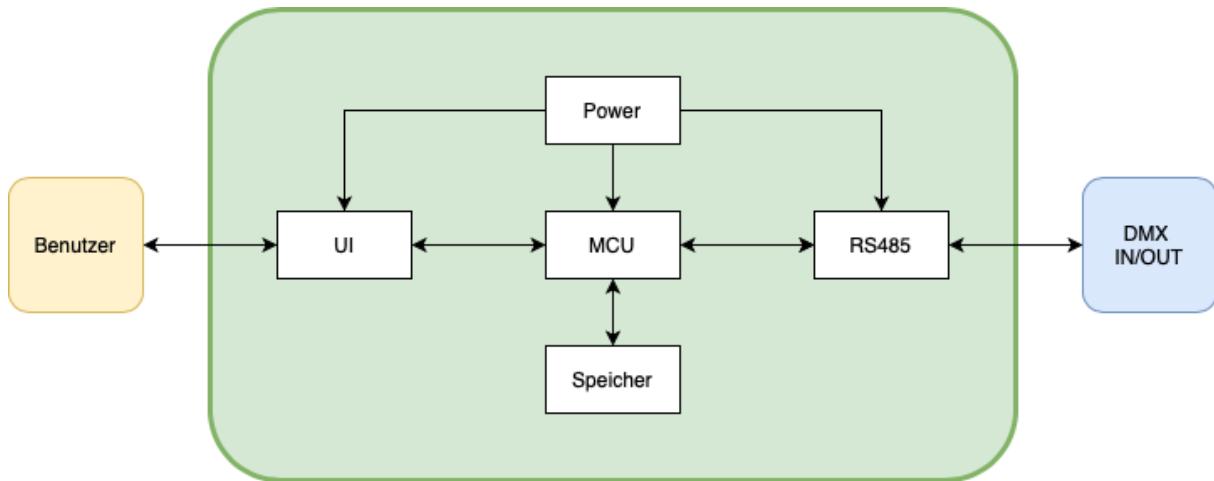


Abbildung 1: Schaltungsprinzip

Mikrocontroller (MCU)

Der Mikrocontroller, im Folgenden MCU genannt, bildet das Mittelpunkt der Schaltung. Die auf ihm ausgeführte Software steuert die Benutzerschnittstelle (UI), speichert und ruft Daten vom Speicher ab und sendet und empfängt die DMX-Daten. Aufgrund der hohen Anforderungen an den MCU wird bei der Entwicklung ein besonderes Augenmerk auf ihn und seine Beschaltung gelegt. Die Auswahl des MCUs richtet sich an dem in der Praxisprojektarbeit verwendeten Mikrocontroller, da dort bereits die Funktionsfähigkeit aufgezeigt ist.

Spannungsversorgung (Power)

Alle Komponenten werden von einer USB-Schnittstelle mit Spannung versorgt. Da das Gerät eigenständig verwendet werden soll, werden über die Schnittstelle keine Daten übertragen. Die Schnittstelle liefert 5 V Spannung [14, s. 44], welche in die benötigten Spannungen transofmiert werden. Besonders kritisch ist die Spannungsversorgung des MCUs, denn unter normalen Betriebsbedingungen darf die Versorgungsspannung nur soweit einbrechen, dass der MCU weiterhin den darauf befindlichen Programmcode fehlerfrei ausführen kann.

Benutzerschnittstelle (UI)

Damit der Benutzer das Gerät bedienen kann, ist ein Informationsfluss in zwei Richtungen erforderlich. Zum einen müssen Eingaben durch den Benutzer erfasst werden, zum anderen muss das Gerät dem Benutzer eine Rückmeldung über Fehler oder den aktuellen Status geben können. Für die Benutzereingaben werden Taster und ein Drehgeber (Encoder) verwendet. Durch den Einsatz von Tastern können präzise und zeitgenaue Eingaben vom Benutzer vorgenommen werden, an Stellen an denen es nötig ist, zum Beispiel beim Starten einer Wiedergabe. Der Encoder hingegen bietet die Möglichkeit viele Eingaben in kurzer Zeit zu tätigen, zum Beispiel wenn die Aufnahmezeit eingestellt wird.

RS485

Die RS485-Schnittstelle ist die Verbindung zwischen DMX-Quelle/Senke und dem MCU, welche notwendig ist, da der MCU keine RS485 konformen Signalpegel liefern, oder auswerten kann. Zudem werden durch die Schnittstelle die DMX-Leitungen galvanisch von der restlichen Schaltung getrennt. Spannungs- und Stromspitzen ausgehend von den DMX-Leitungen können so die Schaltung nicht beeinträchtigen oder beschädigen.

Speicher (SD-Karte)

Damit ein sinnvoller Einsatz des Gerätes möglich ist, ist es notwendig die aufgenommenen Daten auf einem Speichermedium zu sichern und diese nach einem Neustart des Gerätes nach wie vor verfügbar sind. In der vorangegangen Praxisprojektabreit wurde bereits gezeigt, dass der Einsatz einer SD-Karte für diesen Zweck geeignet ist. Die Verwendung einer SD-Karte ermöglicht dem Benutzer außerdem eine gewisse Sicherheit in Bezug auf den Schutz der Daten. Eine Sicherungskopie der Daten kann einfach auf einem PC erstellt werden. Zudem können für verschiedene Einsatzgebiete verschiedene SD-Karten verwendet werden.

3.2 STMicroelectronics STM32F446ZE

Aufgrund der aktuellen Corona-Pandemie und einer damit einhergehenden Siliziumchip-Knappheit wird für die Entwicklung ein anderer Mikrocontroller als in der Praxisarbeit verwendet. Die Eckdaten beider Mikrocontroller sind sich trotzdem sehr ähnlich. Abbildung 2 zeigt die Pinbelegung des Chips. Insgesamt stehen 114 interruptfähige Ein- und Ausgangsports, von denen 112 5 V Tolerant sind und bis zu 20 Schnittstellen zur Verfügung. Der interne 512 Kbytes große Flash-Speicher und 128 Kbyte SRAM bieten genügend Kapazität für einen großen Programmcode, sowie genügend Platz für Heap und Stack. Der STM32F446ZE arbeitet mit einer maximalen Taktfrequenz von 180MHz und basiert auf einem *Arm® 32-bit Cortex®-M4* Prozessor [2, s. 1].

Spannungsversorgung

Die Spannungsversorgung des MCUs ist besonders kritisch, da eine einbrechende Spannung zu Fehlern bei der Ausführung des Programmcodes oder sogar zu einem kurzzeitigen Ausfall und Neustart des MCUs führen kann. Bei steigenden Schaltfrequenzen des MCUs wirken die Leiterbahnen auf der Platine induktiv, wodurch möglicherweise nicht schnell genug genügend Strom geliefert werden kann, was zu einem Einbruch der Spannung führen kann. Um dem entgegenzuwirken werden sogenannte Stützkondensatoren möglichst nah an den Versorgungsspannungs-Pins (V_{DD} und V_{SS}) des MCUs plaziert. Benötigt der MCU kurzzeitig einen großen Strom, so entlädt sich der Kondensator. Durch die kurze Distanz zum MCU muss der Strom über eine kürzere Strecke der Leiterbahn fließen, was eine niedrige Induktivität zur Folge hat. Im Datenblatt [3] des MCUs kann die Dimensionierung der Stützkondensatoren entnommen werden. Insgesamt werden elf Pin-Paare, bestehen aus V_{DD} (3,3 V) und V_{SS} (0 V) mit $0,1 \mu\text{F}$ Kondensatoren parallel verbunden.

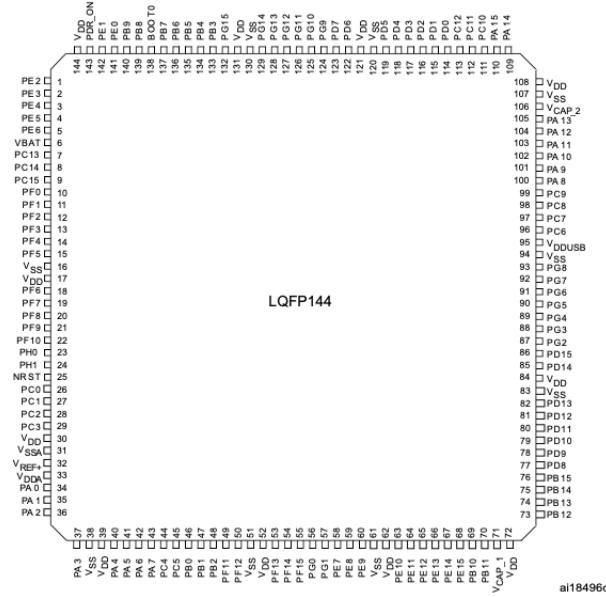


Abbildung 2: STMicroelectronics STM32F446ZE Mikrocontroller Pinout [2, S. 41]

Taktgeber (Clock)

Der MCU verfügt über einen internen hochgeschwindigkeits-Oszillator *High-Speed-Internal-Clock* (HSI) und einen niedergeschwindigkeits-Oszillator *Low-Speed-Internal-Clock* (LSI). Die HSI schwingt mit einer Frequenz von 16 MHz und ist werksseitig auf eine Abweichung der Schwingfrequenz von einem Prozent kalibriert. Um die maximale Taktfrequenz des Prozessors zu erreichen wird eine interne Phasenregelschleife (PLL) eingesetzt. Durch die PLL kann ein Vielfaches der Oszilatorfrequenz erzeugt werden, wobei die Stabilität der Schwingung nicht eingeschränkt wird [4, S. 270]. Eine weitere Möglichkeit die Taktfrequenz zu generieren, ist die Verwendung eines externen Oszillators (HSE). Der in dieser Arbeit verwendete externe Quartz schwingt mit einer Frequenz von 25 MHz und hat dabei eine Genaigkeit von 30 ppm. Das entspricht einer maximalen Abweichung von ca. $30 * 10^{-6}$ %, also einer deutlich geringeren Abweichung im Vergleich zum internen Oszilator. Da für die Aufnahme und Wiedergabe der DMX-Daten möglichst genaue Zeitabstände benötigt werden, wird der externe Quartz verwendet.

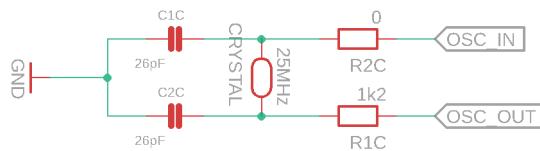


Abbildung 3: Beschaltung Quartz

Abbildung 3 zeigt die Beschaltung des externen Quarzes. Er wird parallel zu dem Eingang *OSC_IN* und dem Ausgang *OSC_OUT* des MCUs geschaltet. Zwischen dem Ein- und Ausgang des MCUs befindet sich ein integriertes *Nicht-Gatter* (Inverter) [5, S. 11], welcher bei einem Schaltvorgang hohe Ströme ausgeben kann. Damit kein Schaden am Quartz und MCU entsteht, wird der maximal fließende Strom durch den Widerstand *R1C* am *OSC_OUT*-Pin begrenzt. Aus Symmetriegründen befindet sich ein 0Ω -Widerstand am *OSC_IN*-Pin, der dafür sorgt, dass auf beiden Seiten des Quarzes die gleichen Übergangskapazitäten der Lötstellen vorhanden sind. Der Quartz wird mithilfe von zwei Lastkapazitäten mit dem 0 V Potential

verbunden. Der Wert der Kapazitäten C_{L1} und C_{L2} lässt sich mit folgender Formel berechnen.

$$C_L = \frac{C_{L1} * C_{L2}}{C_{L1} + C_{L2}} + C_s [5, S.12] \quad (1)$$

C_L wird vom Hersteller des Quarzes vorgegeben und beträgt in diesem Fall 18 pF. C_S ist die Kapazität der Lötstellen und Leiterbahnen zwischen dem Quartz und dem MCU, welche mit 5 pF angenommen wird. Um möglichst wenig verschiedene Bauteile verwenden zu müssen, erhalten in der Regel die Kapazitäten C_{L1} und C_{L2} die gleiche Wertigkeit, wodurch die Formel wie folgt vereinfacht werden kann.

$$C_{L1} = C_{L2} = 2(C_L - C_S) \quad (2)$$

Laut der Formel müssen C_{L1} und C_{L2} eine Kapazität von 26 pF aufweisen. Da Kondensatoren mit dieser Kapazität nur schwer erhältlich sind wird eine marktüblichere Kapazität von 27 pF verwendet.

Serial Wire Debug-Schnittstelle (SWD)

Im Auslieferungszustand des MCUs befindet sich keine Software auf diesem. Mit einem Entwicklungsboard, wie es in der Praxisarbeit verwendet wurde, kann eine Software einfach über eine USB-Schnittstelle auf den MCU übertragen werden, da auf dem Entwicklungsboard ein sogenannter *programmer* verbaut ist. Das in der Praxisarbeit verwendete Entwicklungsboard bietet die Möglichkeit einen Teil der Platine abzutrennen und als eigenständigen *programmer* zu verwenden mit dem die Software mithilfe der *Serial Wire Debug*-Schnittstelle (SWD-Schnittstelle) auf den MCU übertragen werden kann. Die SWD-Schnittstelle ermöglicht das Übertragen der Software und das Debuggen¹ des Codes während der Laufzeit. Für die Datenübertragung zwischen MCU und PC werden nur zwei Verbindungen (SWD und SWCK) benötigt. Über die Leitung SWD werden die Daten gesendet wohingegen die Leitung SWCK den Takt des Datenstroms angibt. Soll ein neues Programm auf den MCU übertragen werden, so muss er zunächst in einen speziellen Modus versetzt werden, um den Programmspeicher beschreiben zu können. Dazu muss der MCU neugestartet werden und beim Start des MCU am *BOOT0*-Pin ein High-Pegel anliegen. Sobald die Übertragung beendet ist, muss der MCU ein weiteres Mal neugestartet werden um den Programmcode auszuführen, jedoch muss beim Start nun ein Low-Pegel am *BOOT0*-Pin anliegen. Das Anlegen der entsprechenden Pegel am *BOOT0*-Pin sowie der Neustart kann per Hand durchgeführt werden, jedoch kann diese Aufgabe auch von dem *programmer* übernommen werden. Dafür müssen zwei weitere Verbindungen für den *BOOT0*-Pin und *NRST*-Pin vorhanden sein. Zuletzt ist außerdem eine Verbindung des 0 V-Potentials notwendig da es sich bei allen Signalen um spannungsbezogene Signale handelt. Optional kann der MCU mit einer weiteren Verbindung vom *programmer* mit 3,3 V Spannung versorgt werden.

NRST- und *BOOT0*-Pin

Unter den 144 Pins des MCUs befinden sich unter anderem, wie bereits im vorigen Abschnitt erwähnt, der *NRST*- und *BOOT0*-Pin. Beide Pins sind mit externen Tastern beschaltet um Fehler bei den Verbindungen zwischen MCU und *programmer* zu kompensieren. Außerdem kann das Programm mit dem Betätigen eines Tasters neugestartet werden, was bei der Programmierung von Vorteil ist. Für den verkaufsfertigen Zustand des Gerätes werden die Taster allerdings nicht mehr benötigt und können einfach weggelassen werden.

¹Analyse und Fehlerbehebung von Programmcode

3.3 Maxim Integrated MAX14945EWE+ RS485 Treiberchip

Für das Senden und Empfangen von DMX-Daten wird eine galvanisch getrennte RS485-Schnittstelle benötigt [1, s.6-8]. Der Einsatz von einzelnen Optokopplern und einem RS-485-Treiberchip, wie in der vorangegangenen Praxisarbeit, benötigt viel Platz auf der Platine und bietet ein hohes Fehlerpotential, da insgesamt vier ICs² auf die Platine aufgebracht werden müssen. Potentielle Fehlerquellen sind eine hohe Anzahl an Lötstellen der ICs und dessen Beschaltung, sowie die Leiterbahnen zwischen den einzelnen Komponenten. Jedes Stück Leiterbahn kann Störungen in den Rest der Schaltung streuen, was zu Fehlern in kritischen Teilen der Schaltung führen kann, beispielsweise der SD-Karte. Die Lösung dieses Problems ist der intern galvanisch getrennte Treiberchip *MAX14945EWE+* der Firma *Maxim Integrated*. Abbildung 4 zeigt den schematischen Aufbau des Treiberchips inklusive dessen typischer Beschaltung. Im

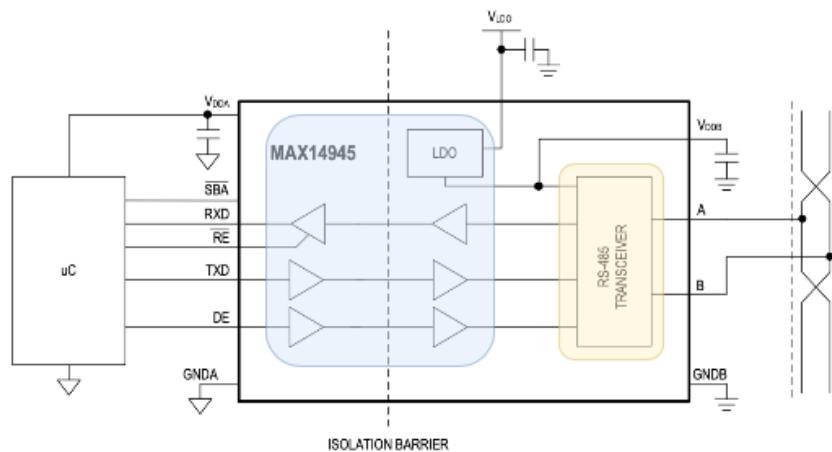


Abbildung 4: MAX14945EWE+ Typical Application Circuit [6, s.19]

blau markierten Bereich befindet sich die galvanische Trennung des Chips. Die gestrichelte Linie in der Mitte markiert die Trennung der Seiten *A* und *B*. Die Funktionen der Pins können Tabelle 1 entnommen werden. Die linke Seite (*A* Seite) ist dem Mikrocontroller zugewendet und wird mit dem 5 V Potential der USB-Schnittstelle versorgt und mit einem $0,1 \mu\text{F}$ und $1 \mu\text{F}$ Kondensator in Parallelschaltung gestützt. Die Kondensatoren sollen laut Datenblatt möglichst nah am Treiberchip platziert werden. Pin *TXD* wird mit dem Ausgang, Pin *RXD* mit dem Eingang der UART-Schnittstelle des MCUs verbunden. Zum aktivieren des Senders oder Empfängers des Treibers muss ein entsprechender Logikpegel an den Pins *DE* und \overline{RE} anliegen. Da die Logik des Pins zum aktivieren und deaktivieren des Empfängers invertiert ist, können beide Pins kurzgeschlossen werden und mit einem Ausgang des MCUs verbunden werden. Ein interner pulldown-Widerstand an beiden Pins ermöglicht die direkte Verbindung zum MCU ohne weitere Beschaltung. Die rechte Seite (*B* Seite) des Treiberchips muss, um die galvanische Trennung zu gewährleisten, mit einer entsprechend getrennten Spannungsquelle versorgt werden. Diese wird mithilfe eines DC/DC Konverters, der die von der USB-Schnittstelle eingehenden 5 V isoliert und am Ausgang wieder ausgibt [7]. Um eine möglichst stabile Spannungsversorgung zu gewährleisten, wird der im Treiberchip interne low – dropout-Spannungsregler verwendet, wodurch Pin *V_{DDB}* als Ausgang der stabilisierten Spannung fundiert. Pin *V_{DDB}* und *V_{LDO}* werden mit jeweils einem $0,1 \mu\text{F}$ und paralegenschaltetem $1 \mu\text{F}$ Kondensator mit *GND_B* verbunden.

²enlg. *integrated circuit*: Integrierter Schaltkreis

Pin	Funktion
<i>TXD</i>	serieller Dateneingang
<i>RXD</i>	serieller Datenausgang
<i>DE</i>	1: Sender aktivieren, 0: Sender deaktivieren
<i>RE</i>	1: Empfänger deaktivieren, 0: Empfänger aktivieren
<i>SBA</i>	1: <i>B</i> Seite inaktiv, 0: <i>B</i> Seite funktionsfähig
<i>A & B</i>	symmetrischer Treiberausgang
<i>V_{DDA} & GNDA</i>	Spannungsversorgung <i>A</i> Seite
<i>V_{DBB} & GNDB</i>	Spannungsversorgung <i>B</i> Seite
<i>V_{LDO}</i>	Spannungsregler Eingang <i>B</i> Seite

Tabelle 1: Pinbeschreibung MAX14945EWE+ [6, s.12-13]

3.4 Encoder

Für die Benutzereingaben sind Taster und vor allem ein Encoder wichtig. Taster ermöglichen eine zeitlich präzise ein Eingabe, wohingegen ein Encoder viele Eingaben in kurzer Zeit ermöglicht. Möchte ein Benutzer eine Aufnahmezeit von 55 Sekunden einstellen, so müsste er 55 mal einen Taster betätigen, mit einem Encoder reichen wenige Umdrehungen dafür aus. In diesem Kapitel wird auf das Funktionsprinzip und die Beschaltung des verwendeten Encoders eingegangen.

Ein Encoder besteht im einfachsten Sinne aus zwei Schaltern, wie Abbildung 5 zeigt. Die Schalter sind

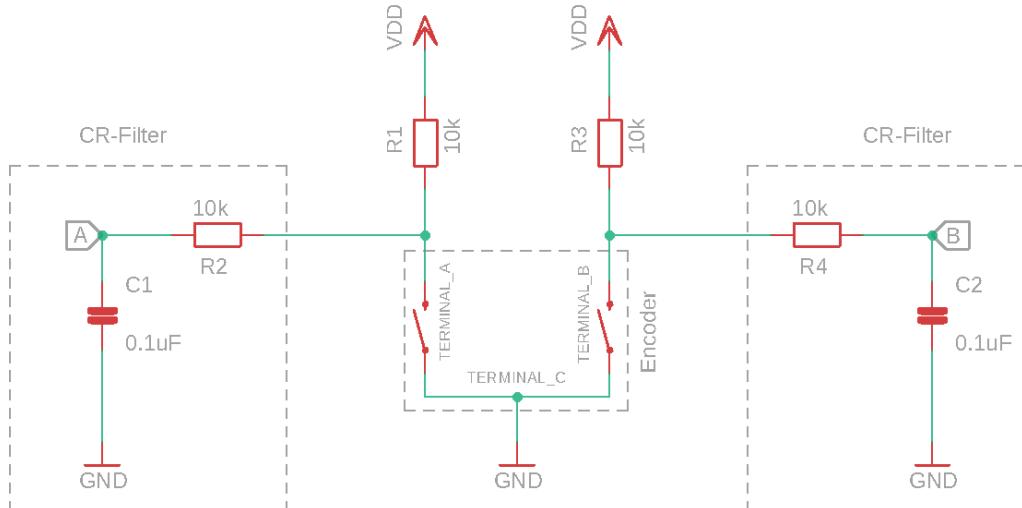


Abbildung 5: Encoder Schaltung [8]

auf der einen Seite intern miteinander verbunden (*Terminal C*) und werden auf das 0 V Potential geschaltet. Die andere Seite der Schalter wird direkt über *Terminal A* und *Terminal B* nach außen geführt. Beider Terminals werden mit jeweils einem 10 kΩ Pull-Up-Widerstand auf die Betriebsspannung von 5 V gezogen. Gleichzeitig wird der Strom, der fließt wenn ein Schalter geschlossen ist, begrenzt. Der Hersteller des Encoders empfiehlt das Besetzen des Encoders mit zwei CR-Tiefpassfiltern um hochfrequente Schaltvorgänge zu filtern. Bei einem Schaltvorgang können die Kontakte des Schalters mechanisch schwingen und dadurch mehrfach hintereinander öffnen und schließen [9, s. 67], wodurch mehrfache Eingaben getätigten werden, obwohl der Encoder nur um einen Schritt gedreht wurde. Die Signale des Encoder wer-

den an Punkt A und B in Abbildung 5 entnommen und über zwei Leiterbahnen zum MCU geführt. Ist ein Schalter geöffnet so wird die Kapazität des CR-Filters geladen und die an ihm abfallende Spannung erhöht sich bis zur Betriebsspannung von 5 V. Wird der Schalter nun geschlossen, so entlädt sich der Kondensator in Richtung des entsprechenden Terminals. Die Spannung an der Kapazität sinkt bis zum 0 V-Potential. Fängt ein Schalter an zu schwingen, reicht die Zeit zum laden bzw. entladen der Kapazität nicht aus um eine Pegeländerung zu erzielen. Wenn der Encoder gedreht wird schließen und öffnen die

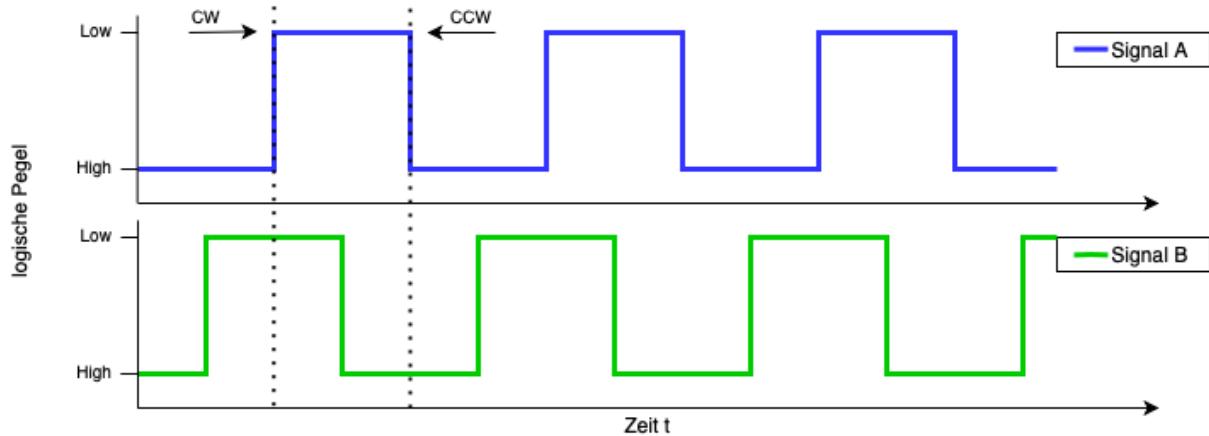


Abbildung 6: Encoder-Signal bei Drehung im Uhrzeigersinn

internalen Schalter versetzt zueinander. Abbildung 6 zeigt die ausgehenden Signale A und B im zeitlichen Verlauf bei einer gleichmäßigen Rotationsgeschwindigkeit. Durch den versetzten Rhythmus der beiden Schalter kann die Drehrichtung bestimmt werden, indem eines der beiden Signale betrachtet wird, in diesem Beispiel Signal A. Ändert sich der Zustand des Signals von *high* auf *low* kann zunächst bestimmt werden, dass eine Drehbewegung stattgefunden hat. Ist nun der Pegel des anderen Signals *high*, so handelt es sich um eine gegen den Uhrzeigersinn drehende Bewegung. Ist das Signal *low*, so handelt es sich um eine mit dem Uhrzeigersinn drehende Bewegung. Die einzelnen Schritte der Schalter können haptisch vom Benutzer wahrgenommen werden. Jeweils an den Stellen an denen sich die Pegel beider *Terminals* im *high*-Zustand befinden rastet der Drehregler leicht ein. Dadurch wird sichergestellt, dass sich die Signale des Encoders während der Nichtbenutzung in einem stabilen Zustand befinden. Zudem ergibt sich daraus eine zusätzliche Rückmeldung an den Benutzer während der Eingabe.

3.5 Liquid Crystal Display (LCD)

Das LCD-Display ist der wichtigste Bestandteil der Benutzerschnittstelle. Auf ihm können wichtige Informationen über den aktuellen Status des Geräts und Fehlermeldungen angezeigt werden. Eine Menüführung in Verbindung mit den Tasten und dem Encoder gibt dem Benutzer die Möglichkeit alle Funktionen intuitiv zu bedienen. Bei der Auswahl des Display ist vor allem die Geschwindigkeit der Datenübertragung zum Display wichtig, da das Beschreiben des Displays auf keinen Fall die Aufnahme oder Wiedergabe der DMX-Daten stören darf. Aus diesem Grund wird ein einfaches Liquid-Crystal-Display (LCD) für die Anzeige von Zeichen verwendet. Insgesamt stehen 80 Punkt-Matrizen, angeordnet in 20 Spalten und 4 Zeilen, zur Anzeige von Zeichen zu Verfügung, wobei jede Matrize aus 40 Punkten (5 Spalten x 8 Zeilen) besteht. Die Zeichen werden in weiß auf blauem Hintergrund dargestellt und eine LED-Hintergrundbeleuchtung sorgt dafür, dass die angezeigten Inhalte auch in dunklen Umgebungen einfach abgelesen werden können. Im Auslieferungszustand (Abbildung 7 und 8) besteht das Modul aus einer großen grünen Platine mit dem fest verbauten LCD-Display darauf und einem an die 16 Pins gelöteten I^2C -Modul³ (kleine schwarze

³Bussystem zur seriellen Übertragung von Daten [10, s.61]

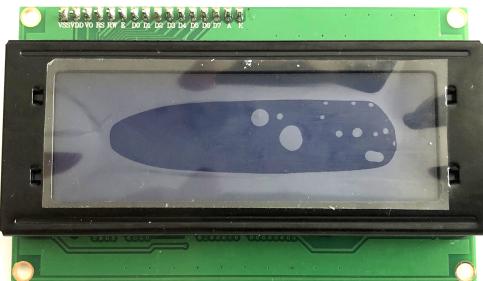


Abbildung 7: LCD-Modul Vorderseite



Abbildung 8: LCD-Modul Rückseite

Platine) mit der das Beschreiben des Displays mittels serieller Übertragung möglich ist. Der Vorteil der seriellen Datenübertragung mittels I^2C ist, dass nur eine Datenleitung, eine Takteleitung und das 0 V-Potential mit dem MCU verbunden werden müssen. Ein Nachteil stellt die Übertragungsgeschwindigkeit dar, denn alle Bits müssen nacheinander übertragen werden. Geht man davon aus, dass die gleiche Taktrate bei einer parallelen Datenübertragung mit 8 Bits erreicht wird, so ist die Übertragungsrate der seriellen Übertragung mindestens um den Faktor 8 kleiner, da zu den 8 zu übertragenden Bits eine Start-Bedingung des I^2C -Protokolls hinzukommt [10, s. 62]. Da die Geschwindigkeit der Übertragung von besonderer Bedeutung ist, wird das I^2C -Modul von dem Modul entfernt und stattdessen die parallele 8-Bit Datenübertragung verwendet. Insgesamt werden elf Leiterbahnen vom LCD-Modul direkt zum MCU geführt. Die einzelnen Funktionen der Pins des LCD-Moduls können Tabelle 2 entnommen werden. Die Pin-Paare VDD/VSS und A/K werden jeweils mit dem 5 V und 0 V Potential der USB-Schnittstelle beschaltet. Pin R/W wird außerdem mit dem 0 V Potential verbunden, da das LCD-Display ausschließlich vom MCU beschrieben und nicht gelesen werden soll. Die übrigen Pins werden direkt mit dem MCU verbunden. Der Kontrast des Display wird mit einer analogen Spannung an Pin VO eingestellt, welche als Pulsweltenmoduliertes Signal vom MCU ausgeht.

Pin	Funktion
VDD & VSS	generelle Spannungsversorgung
A & K	Anode und Kathode der LED-Hintergrundbeleuchtung
D0-D7	Datenbus
RS	H: Display Data, L: Display Instruction
E	Enable
R/W	Read/Write-Modus
VO	LCD-Kontrast

Tabelle 2: LCD-Pinbelegung [11, s. 7]

Die Darstellung der Zeichen in den Punkt-Matrizen wird mithilfe von insgesamt drei Treiberchips realisiert. Die Treiberchips übersetzen die eingehenden Signale der 8 Daten und 4 Steuerleitungen in entsprechende Funktionen, wie die Ausgabe eines bestimmten Zeichens oder das Leeren des gesamten Displayinhaltes. Eine Tabelle im Datenblatt des LCD-Moduls zeigt die Konstellationen von High- und Low-Pegeln der 8-Datenleitungen und welche Zeichen daraus resultieren. Das Datenblatt befindet sich im Anhang B.

3.6 LED

Licht emittierende Dioden (LED) sind ein essentieller Bestandteil der Benutzerschnittstelle für die Lokalisierung von Fehlern und der Überwachung während einer Aufnahme- oder Wiedergabe aus kurzer und

großer Distanz. Insgesamt werden fünf LEDs in vier verschiedenen Farben (blau, orange, grün und rot) verbaut um dem Benutzer eine Unterscheidung auch auf große Distanz zum Gerät möglichst einfach zu machen. Ziel ist eine möglichst hardwarenahe Rückmeldung an den Benutzer zu geben. Tabelle 3 fasst die Funktionen der einzelnen LEDs zusammen. Die erste blaue LED wird direkt an die 5 V Versorgungs-

LED-Farbe	Funktion
blau	Eingeschaltet bei aktiver Spannungsversorgung des Geräts
rot	Zustandsänderung bei eingehendem DMX-Datenpaket
blau	Zustandsänderung bei ausgehendem DMX-Datenpaket
orange	Eingeschaltet bei Schreib- oder Lesevorgängen der SD-Karte
grün	Allgemeine Zustandsanzeige

Tabelle 3: LED Funktionen

spannung, ausgehend von der USB-Buchse, über einen Widerstand zur Strombegrenzung, angeschlossen. Dadurch kann sehr einfach festgestellt werden ob das Gerät mit Spannung versorgt wird, unabhängig von der Funktion der restlichen Schaltung. Die zweite blaue und die rote LED müssen zwangsläufig per Software geschaltet werden, da der XLR Ein- und Ausgang parallel verbunden sind. Eine Unterscheidung von ein- bzw. ausgehenden Datenpaket ist so nicht möglich. Zudem haben Tests gezeigt, dass das direkte Verbinden einer LED mit einer Datenleitung die LED nur kaum bis nicht sichtbar zum leuchten bringt. Gleiches gilt für die orange LED. In einer Weiterentwicklung des Gerätes kann z.B. mithilfe einer monostabilen Kippstufe, wie einem NE555 IC [12], welche den eingeschalteten Zustand einer LED verlängern kann, eine hardwarenähtere Lösung realisiert werden. Abbildung 9 zeigt die Verschaltung der

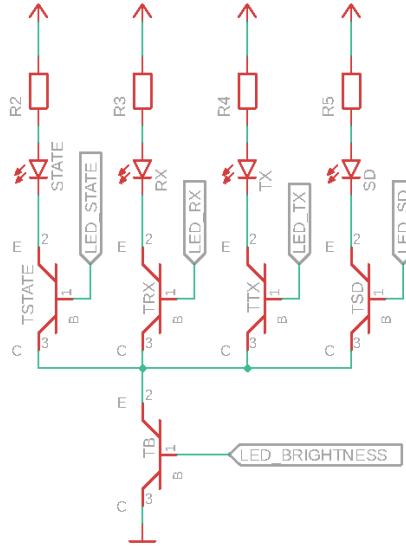


Abbildung 9: LED Schaltung

von dem MCU gesteuerten LEDs. Jede LED wird mit 5 V Spannung versorgt. Jeweils ein Widerstand begrenzt den Strom um ein Durchbrennen der LED zu verhindern. Da für die vorhandenen LEDs keine Datenblätter verfügbar sind, werden alle Vorwiderstände mit 120 Ohm dimensioniert. Das eigentliche Ein- und Ausschalten wird über jeweils einen Transistor realisiert. Die Basen des Transistoren werden mit jeweils einem Ausgang des MCUs verbunden. Der Benutzer soll die Möglichkeit besitzen die Helligkeit der LEDs je nach Anforderung zu verändern. Dazu werden die Kollektoren aller LED-Schaltransistoren auf den Emitter eines weiteren Transistors TB geschaltet. Die Basis wird mit einem Ausgangaspin des

MCUs verbunden, der mittels eines Timers ein Pulsweitenmoduliertes-Signal (PWM-Signal) ausgeben kann. Mithilfe einer Anpassung der Pulsweite kann so die Helligkeit aller LEDs reguliert werden. Je kürzer die Pulsweite ist, also je kürzer die LED eingeschaltet ist, so weniger hell erscheint sie. Die Frequenz des Pulsweitenmodulierten Signals muss dabei so hoch sein, dass für das mensliche Auge nicht mehr ersichtlich ist wann die LED leuchtet oder nicht.

3.7 Spannungsversorgung

Eine stabile und zuverlässige Spannungsversorgung ist für jede Schaltung besonders wichtig, denn die beste Schaltung funktioniert nicht ohne sie. Um eine größtmögliche Kompatibilität für den Benutzer zu gewährleisten, wird das Gerät über eine USB-Schnittstelle mit der nötigen Spannung versorgt. Dadurch kann das Gerät von mobilen Akkus oder USB-Netzteilen betrieben werden. Die nachstehende Tabelle zeigt den benötigten Versorgungsspannungs- und den Strombedarf der einzelnen Komponenten. Die USB-Schnittstelle liefert laut der USB2.0-Spezifikation zwischen 4,75 V und 5,5 V [13, s. 283] und

Bauteil	Versorgungsspannung	Strombedarf
MCU	1,7 V - 3,6 V	max. 240 mA
RS485 Treiberchip VDDA	1,71 V - 5,5 V	max. 6,6 mA
RS485 Treiberchip VDBB	4,5 V - 5,5 V (isoliert)	max. 12,5 mA
DC/DC Konverter	4,5 V - 5,5 V	28 mA
LCD-Modul	3 V - 10 V	100 mA
LED	5 V	ca. 20 mA
SD-Kartensteckplatz	3,3 V	-
Encoder	5 V	-
Taster	-	-
Gesamt	-	max. ca. 481 mA

Tabelle 4: Spannungs- und Strombedarf

standardmäßig einen Strom von 100 mA. Um mehr Strom von einem USB-Host⁴ anzufordern ist eine Kommunikation über die USB-Datenleitungen mithilfe eines Mikrocontrollers oder einem speziell für diesen Zweck vorgesehenen Chip notwendig. Allerdings kann das Gerät durch das Verbinden der beiden USB-Datenleitungen $D+$ und $D-$ mit einem Widerstand kleiner 200Ω als *dedicated charging port* (DCP) vom USB-Host registriert werden [14, s. 41]. Durch die Einstufung als DCP kann der USB-Host ohne jegliche Kommunikation bis zu 1,5 A zur Verfügung stellen [14, s. 45]. Als Stromversorgung können dann einfache USB-Netzteile, mobile -Batterien, -Anschlüsse an Laptops und PCs, sowie HUBs mit einer externen Stromversorgung genutzt werden. Voraussetzung für die Lieferung des Stroms ist eine ausreichend dimensionierte Stromversorgung des USB-Host selbst. Abbildung 10 zeigt die Schaltung der Spannungsversorgung des Gerätes. Der Widerstand $R1$ sorgt dafür, dass das Gerät, wie bereits erwähnt, als DCP erkannt wird. Der Elektrolyt-Kondensator $C1$ fungiert als kleiner Puffer, falls die Schaltung für einen kurzen Moment mehr Strom benötigt als die USB-Schnittstelle liefern kann. Der damit einhergehende Spannungsabfall wird zudem kurzzeitig ausgeglichen. Um eventuell zurückfließenden Strom in den USB-Host zu verhindern wird die Diode $D1$ parallel zum 5 V Potential (VCC) und Ground (GND) geschaltet. Auf der rechten Seite der Abbildung 10 befindet sich ein LDO-Spannungsregler. Er transformiert die

⁴In der Hierarchie übergeordnetes USB-Gerät

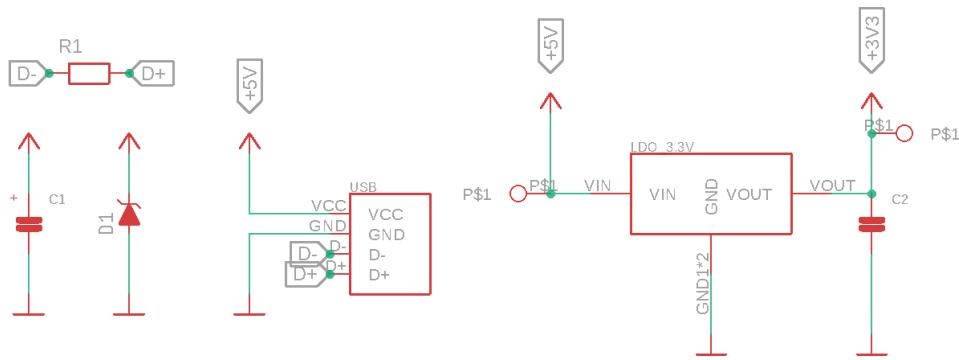


Abbildung 10: Schaltung Spannungsversorgung

eingehenden 5 V vom USB-Host auf 3,3 V herunter und stabilisiert diese gleichzeitig. Laut Tabelle 4 wird ein Strom von maximal ca. 240 mA aus dem Spannungsregler benötigt. Laut Datenblatt fällt an ihm bei 25 °C Umgebungstemperatur und einem Strom von 250 mA eine Spannung von ca. 100 mV ab. Der MCU und der SD-Kartensteckplatz können somit mit ausreichend Spannung versorgt werden. Alle anderen Komponenten werden direkt mit den 5 V der USB-Schnittstelle versorgt um den LDO-Spannungsregler so wenig wie möglich zu belasten.

3.8 Autodesk EAGLE

Die Schaltpläne und das Design der Platine wird in der *Electronic Design Automation* (EDA) Software *EAGLE* der Firma *Autodesk* entwickelt. Innerhalb der Software können Schaltungspläne erstellt werden und anhand dieser Schaltung ein entsprechendes Layout der zu fertigenden Platine erstellt werden. Zuletzt können Produktionsdaten exportiert werden, mit denen die Platine industriell hergestellt werden kann [15].

Für die Erstellung des Schaltplans und Platinenlayouts wird eine Bauteilbibliothek benötigt. *EAGLE* beinhaltet bereits initial eine große Bibliothek mit verschiedenen Bauteilen vieler Hersteller, jedoch umfasst die nicht alle am Markt verfügbaren Bauteile. Viele der in dieser Arbeit verwendeten Bauteile befinden sich nicht darin, weswegen eine eigene Bauteilbibliothek erstellt und explizit für die Entwicklung verwendet wird. Zudem soll dadurch das Fehlerrisiko in Bezug auf Bauteilabmessungen und Pinbelegungen reduziert werden. Ein Bauteil der Bauteilbibliothek besteht aus Icht der Software aus drei Teilen. Dem *symbol* (Abbildung 11), dem *footprint* (Abbildung 12) und einem 3D-Modell (Abbildung 13). Das



Abbildung 11: EAGLE Symbol

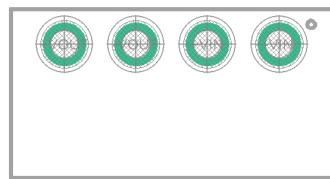


Abbildung 12: EAGLE Footprint

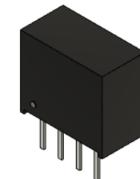


Abbildung 13: EAGLE 3D-Modell

Symbol wird für die Entwicklung des Schaltplans verwendet und besitzt keine elektrischen Eigenschaften oder Dimensionen. Eine Simulation der Schaltung ist dementsprechend in *EAGLE* nicht möglich. Die Anschlüsse der Symbole können im Schaltplan mit Leitungen verbunden werden. Um den Schaltplan auf die Hardwareebene zu projizieren, wird ein zugehöriger *footprint* benötigt. Dieser enthält die Dimensionen und Anordnung der entsprechenden Lötstellen des Bauteils, welche in der Regel vom Hersteller des Bauteils zur Verfügung gestellt werden. Im Bibliotheks-Manager in *EAGLE* werden dann die Anschlüsse des Sym-

bols den entsprechenden Lötstellen des *footprints* zugeordnet. Die Verbindung zwischen Schaltplan und Platine ist nun hergestellt. Um einen noch realistischeren Entwurf der Platine einsehen zu können, kann ein 3D-Modell des Bauteils eingefügt werden. Standard Bauteile wie SMD-Kondensatoren, -Widerstände oder gängige Chip-Gehäuse können mit einem in *EAGLE* integrierten Generator generiert werden. Mit den vom Hersteller des Bauteils angegebenen Dimensionen und Toleranzen erzeugt der Generator einen *footprint* mit einem dazugehörigen 3D-Modell des Bauteils. Handelt es sich bei dem Bauteil um kein Standard-Bauteil, so muss der *footprint* in *EAGLE* und das 3D-Modell in einer externen CAD-Software konstruiert werden. Im Bibliotheks-Manager kann dann das 3D-Modell bezogen auf den *footprint* dreidimensional plaziert werden. Mithilfe der 3D-Modelle kann *EAGLE* in Verbindung mit der Software *Fusion 360* der Firma Autodesk ein 3D-Modell der gesamten Platine inklusive aller Bauteile, für die entsprechende 3D-Modelle in der Bauteilbibliothek existieren, exportieren. Dieses kann dazu verwendet werden Kollisionen zwischen Bauteilen oder einem Gehäuse vor der Fertigung der Platine zu erkennen oder ein passendes Gehäuse für die Platine zu entwerfen.

Mithilfe der in *EAGLE* integrierten sogenannten *Forward-Back-Annotation* werden Änderungen im Schaltplan unmittelbar in das Platinenlayout übertragen. Wird Beispielsweise nur der Wert eines Widerstandes im Schaltplan geändert, so ändert sich auch die Beschriftung des Bauteils auf der Platine. Außerdem werden in der Schaltung gesetzte Verbindungen im Platinenlayout dargestellt und damit das verlegen von Leiterbahnen vereinfacht und kann fehlerfrei durchgeführt werden. Im Schaltplan nicht bestehende Verbindungen können im Platinenlayout nicht hergestellt werden. Um die Schaltung möglichst

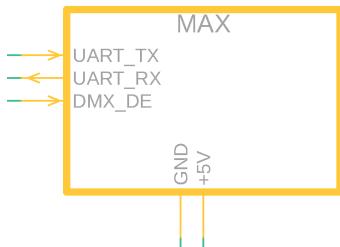
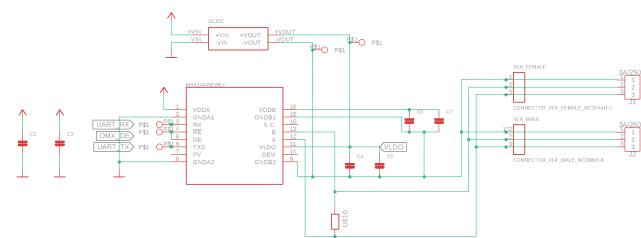


Abbildung 14: Modul "MAX"



und in der Lage dazu sein die DMX- und das USB-Kabel in die entsprechenden Buchsen zu stecken.



Abbildung 16: Platinenvorderseite

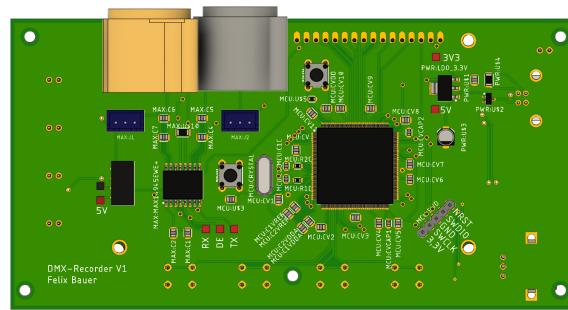


Abbildung 17: Platinenrückseite

Abbildung 16 und 17 zeigen das Layout der Platine als virtuelles dreidimensionales Modell. Auf der Platine werden zwei Arten von Komponenten verbaut. Zum Einen sogenannte *surface-mount-technology*-Komponenten (SMT-Komponenten), zum Anderen *through-hole technology*-Komponenten (THT- Komponenten). SMT-Komponenten werden direkt auf die Oberfläche der Platine gelötet, was keine Notwendigkeit von Löchern in der Platine erfordert. Durch die Oberflächenmontage befindet sich das Bauteil auf der selben Seite wie die dazugehörige Lötstelle. Für THT-Komponenten werden Löcher in der Platine benötigt, da die Pins der Komponenten im 90 Grad Winkel zur Oberfläche der Platine gerichtet sind. Diese können unter Umständen nur auf der gegenüberliegenden Seite der Platine verlötet werden. Eine LED zum Beispiel, die bündig mit der Platine verlötet werden soll, kann nur auf der gegenüberliegenden Seite der Platine verlötet werden, da die LED selbst ihre eigenen Pins verdeckt. Damit alle Komponenten frei auf der Platine platziert werden können, wird eine zweiseitige Platine entworfen. Diese besitzt auf der Vorder- und Rückseite eine Kupferschicht, welche mithilfe von sogenannten *Vias*⁶ verbunden werden können. Abbildung 18 zeigt die generelle Aufteilung der Platine. Diese Aufteilung vereinfacht die mögliche

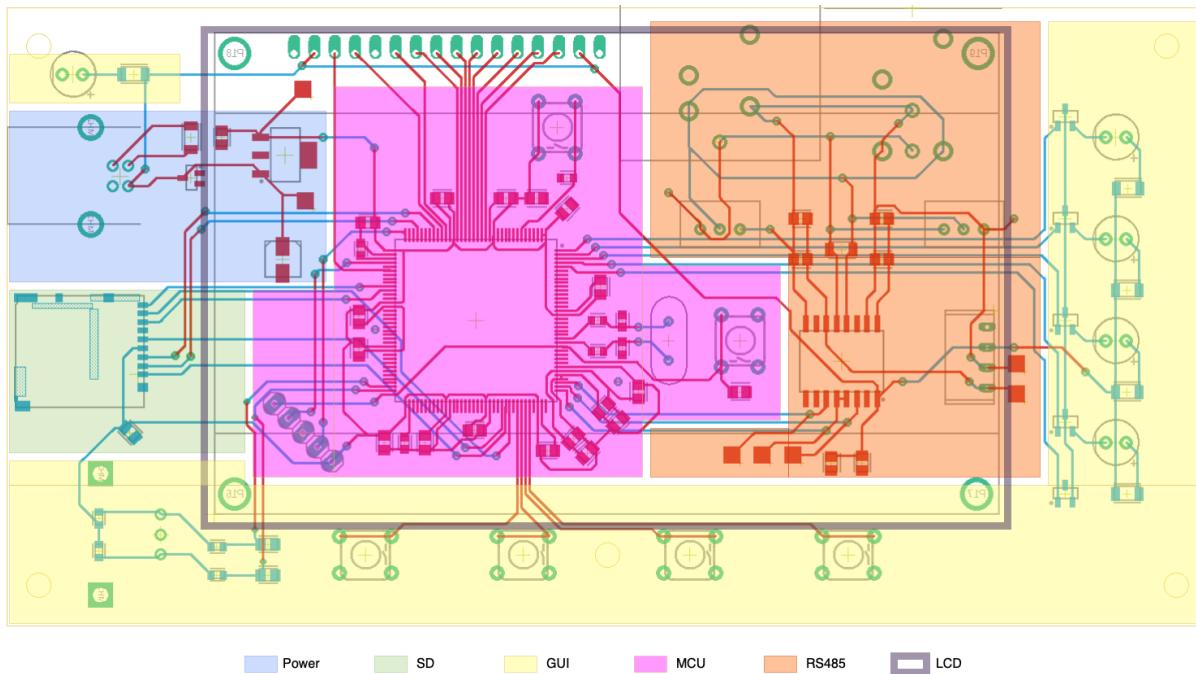


Abbildung 18: Platinenlayout-Aufteilung

Fehlersuche und trägt dazu bei die Leiterbahnlängen zu verkürzen. Alle blau gefärbten *Pads*⁷ und Leiter-

⁶Durchkontakte Löcher in der Platine

⁷In der Regel rechteckige Lötstellen (freigelegte Kupferflächen) zum verlöten von SMT-Komponenten

bahnen befinden sich auf der Vorderseite, alle rot eingefärbten auf der Rückseite der Platine. Die freien Flächen zwischen den Leiterbahnen auf der Ober- und Unterseite sind mit dem 0 V Potential verbunden. Die grünen Bohrungen sind *Vias* und durchkontakteert Bohrungen für die THT-Komponenten. Der Formfaktor der Platine wird im wesentlichen von dem LCD-Modul vorgegeben, da es bei weitem das größte Bauteil darstellt. Neben dem LCD-Modul muss außerdem Platz für den Encoder, die Taster und LEDs vorhanden und für den Benutzer sichtbar und verwendbar sein. Aus diesem Grund werden das LCD-Modul, die Taster, der Encoder und die LEDs auf der Vorderseite der Platine platziert. Im Zentrum der Platine befindet sich das LCD-Modul. Unterhalb des Moduls befinden sich die Taster, rechts davon die LEDs. Die LED, die den Zustand der Spannungsversorgung anzeigt befindet sich oben links um eine Verwechslungen mit der zweiten blauen LED zu verhindern. Der SD-Kartensteckplatz, grün markiert, findet am linken Rand unterhalb der USB-Buchse und Schaltung der Spannungsversorgung, blau markiert, der Platine Platz. Der MCU bildet den größten Knotenpunkt der Schaltung und befindet sich deswegen in der Mitte der Platine. Laut Hersteller sollen die Stützkondensatoren möglichst nah am MCU plaziert werden, damit die Induktivität der Leiterbahnen den hohen Stromfluss aus den Kondensatoren heraus nicht beeinträchtigt, wie bereits in Kapitel 3.2 erläutert. Kritische an den MCU angeschlossene Komponenten sind unter anderem der SD-Kartenslot, der Quartz und der RS485-Treiberchip. Um möglichst wenig Störungen auf die Leiterbahnen einwirken zu lassen sind diese möglichst kurz. Die XLR-Buchsen für die ein- und ausgehenden DMX-Daten befinden sich am oberen Rand der Rückseite der Platine. Sie und die Schaltung des Treiberchips, rot markiert, liegen nah beieinander um auch hier das Störungspotential durch lange Leiterbahnen möglichst gering zu halten.

3.10 Gehäusedesign

Mit dem Gehäuse wird aus der eher abstrakt wirkenden Platine ein optisch ansprechendes Endprodukt. Wichtig ist nicht nur die Optik sondern auch das Design in funktioneller Sichtweise. Das beschriebene Platinendesign in Kapitel 3.9 und 3D-Modell bilden die Basis für die Entwicklung des Gehäuses und kann durch die Verbindung von *EAGLE* mit der 3D-CAD-Software *Fusion360* direkt virtuell in die Konstruktion eingefügt werden. Damit kann sichergestellt werden, dass z.B. Bohrungen im Gehäuse mit den in der Platine befindlichen übereinstimmen und das die Bauteile auf der Platine nicht mit dem Gehäuse kollidieren. Die Konstruktion ist auf die Fertigung im 3D-Druckverfahren ausgerichtet, da damit in kurzer Zeit Design-Iterationen des Designs kostengünstig hergestellt werden können. Bei dem verwendeten Material handelt es sich um den biologisch abbauaren Kunststoff *Poly – lactid – acid* (PLA). Er ist einfach zu drucken, nachzubearbeiten, kostengünstig in der Anschaffung und besitzt eine ausreichende Stabilität.

Abbildung 19 und 20 zeigen das 3D-Modell des Gehäuses, dessen grundsätzliches Design simpel und kompakt ist (15,7 cm breit, 8,14 cm tief und 5,25 cm hoch). Die Breite und Tiefe des Gehäuses werden nahezu komplett von der Platine ausgefüllt. Die Höhe des Gehäuses wird von den XLR-Buchsen vorgegeben und ist daher besonders im vorderen Teil nicht voll ausgenutzt. Bei einer Weiterentwicklung könnte dieser freie Platz für einen integrierten Akku genutzt werden. Der Boden des Gehäuses verläuft parallel zur Platine damit die eingesteckten Kabel rechtwinklig zur Tischoberfläche verlaufen und sie somit das Gerät durch ihr Eigengewicht nicht zum kippen bringen. Auf der Oberseite befindet sich ein Rechteckiger Ausschnitt für das LCD-Modul, runde Ausschnitte für die LEDs, Taster und den Encoder. Über den LEDs befinden sich weiße Diffusionsscheiben, die das Licht der darunterliegenden LEDs brechen und somit das Leuchten aus größerer Distanz und einem größeren Blickwinkel sichtbar machen. Auf der Rückseite befinden sich runde Ausschnitte für die XLR-Buchsen und eine Vielzahl von Schlitten, die einen Luftaustausch ermöglichen und einem Hitzestau vorbeugen. Auf der linken Seite befinden sich Ausschnitte für den SD-Kartensteckplatz und USB-Anschluss. Die Platine muss ausreichend im Gehäuse befestigt sein damit sie sich beim Betätigen der Taster nicht verbiegt und somit das Betätigen erschwert



Abbildung 19: Gehäuse 3D-Modell Frontseite



Abbildung 20: Gehäuse 3D-Modell Rückseite

bzw. unmöglich macht oder die Platine unter Umständen beschädigt wird. Abbildung 21 und 22 zeigen Schnittanalysen des 3D-Modells. Für einen größtmöglichen Halt der Platine im Bereich der Taster wird die Platine in einen Schlitz im Gehäuse eingeführt (rot markiert). Durch den Schlitz wird die Auflagefläche der Platine im Gehäuse erhöht und die durch einen Tastendruck einwirkende Kraft auf sie verteilt. Auf der gegenüberliegenden Seite wird die Platine mithilfe von Schrauben mit M3-Gewinde befestigt (blau markiert). Dazu befinden sich zylinderförmige Extrusionen an der Oberseite, die mittig eine Bohrung mit einem M3 Gewinde besitzen. Mit der grün markierten Bohrung wird das LCD-Modul mit den bereits vorhanden Löchern verschraubt. Abstandshalter zwischen LCD-Modul und Platine ermöglichen die Verschraubung von der Unterseite der Platine. Damit die Platine im Gehäuse plaziert und befestigt werden

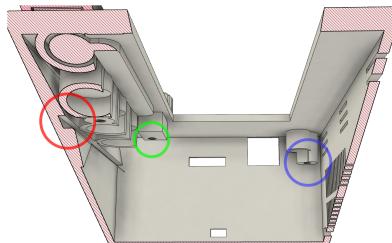


Abbildung 21: Gehäuse Schnitt

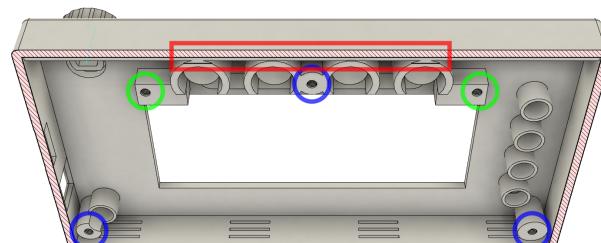


Abbildung 22: Gehäuse Schnitt Draufsicht

kann, muss das Gehäuse geöffnet werden können. Um bei der Entwicklung möglichst unkompliziert die Platine erreichen zu können, befindet sich ein Clip-Mechanismus an der Unterseite des Gerätes, mit dem der Boden einfach entfernt werden kann. Abbildung 24 zeigt die Seitenansicht des Mechanismus. In der Oberseite des Gehäuses befinden sich Rechteckige Aussparungen in die die an der Unterseite befindlichen Clips einhaken können. Durch das verwendete 3D-Druckverfahren sind die Clips eine Schwachstelle der Konstruktion, da PLA verhältnismäßig spröde ist und die Clips beim einhaken in die Aussparungen leicht gebogen werden müssen. Um das Material beim verbiegen weniger zu belasten sind die Clips bis in den Boden hinein verlängert. Dadurch wird das Material auf eine längere Strecke hinweg gebogen und das Biegemoment somit verteilt.

Ein wichtiger Bestandteil der Bedienbarkeit stellen die Taster dar. Abbildung 25 und 26 zeigen den Mechanismus zum Betätigen des Tasters als Schnittansicht des 3D-Modells. Die Körper mit der Kennzeichnung 1 sind das Gehäuse an sich, die Taster sind mit 4 gekennzeichnet. Unter den Tastern befindet sich die Platine. Im Gehäuse befinden sich vier nach unten gerichtete Führungen für einen beweglichen Zylinder, welcher mit 3 gekennzeichnet ist. Alle vier Zylinder sind mit miteinander verbunden um die Stabilität zu erhöhen und ein verkanten dieser mit der Führung zu verhindern. Für die Zylinder wird ein gummiartiges Material, namens *TPU* für den 3D-Druck verwendet, wodurch das Klicken des Tasters gedämpft wird. Die für den Benutzer sichtbaren beweglichen Taster-Flächen, gekennzeichnet mit 2, drücken den Zylinder bei

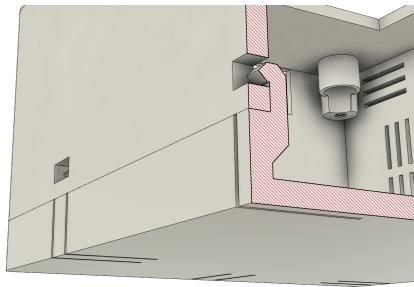


Abbildung 23: Gehäuse Clip-Mechanismus
Schnittansicht frontal

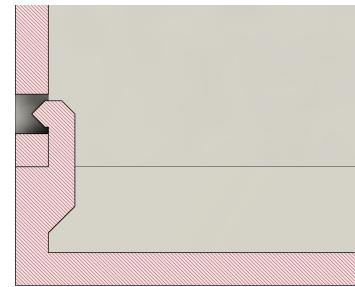


Abbildung 24: Gehäuse Clip-Mechanismus
Schnittansicht

einer Betätigung nach unten und diese wiederrum betätigen den Taster. Auf den Tasterflächen befinden sich Extrusionen mit der Kennzeichnung der entsprechenden Funktion des jeweiligen Tasters. Dadurch kann der Taster optisch und haptisch wahrgenommen werden, was besonders an Orten mit schlechten Belichtungsverhältnissen wichtig ist.

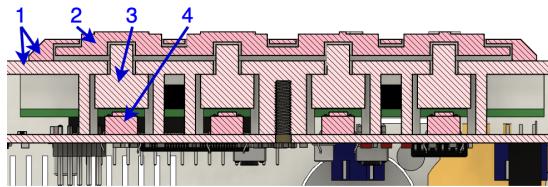


Abbildung 25: Gehäuse Clip-Mechanismus

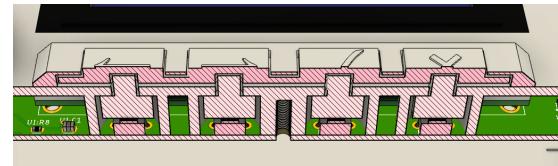


Abbildung 26: Gehäuse Clip-Mechanismus
Schnittansicht

4 Software

4.1 Hardwareabstraktionsschicht HAL (Hardware Abstraction Layer)

Bei der Programmierung eines Mikrocontrollers muss selbstverständlich eine für den Mikrocontroller verständliche Sprache gesprochen werden. Möchte man beispielsweise ein Byte mithilfe der UART-Schnittstelle senden, so muss der intern verbundene entsprechende Pin das Signal ausgeben. Wird nun der selbe Programmcode für einen anderen Mikrokontroller verwendet, ist der Ausgangspin des UARTs eventuell ein anderer und der Programmcode wird nicht funktionieren. Die Lösung des Problems ist entweder das Ändern des Programmcodes oder die Verwendung der Hardwareabstraktionsschicht (HAL). Die HAL trennt die Anwendungsschicht von der Hardwareschicht, sodass der Programmcode unabhängig von der Hardware geschrieben werden kann. Bei der Programmierung werden HAL-Funktionen aufgerufen, welche dann die entsprechenden Hardware-Operationen durchführen. Die HAL ist außerdem in einzelne Komponenten aufgeteilt, wie z.B. die UART-Schnittstelle oder Timer [16, s.77 ff.]. Als Beispiel soll ein Byte über die UART-Schnittstelle gesendet werden. Im Hauptprogramm wird dafür die Funktion *HAL_UART_Transmit(...)* aus der Datei *stm32f4xx_hal_uart.c* aufgerufen. In dieser Funktion wird die UART-Schnittstelle für das Senden vorbereitet, das Byte in das entsprechende Datenregister geschrieben und anschließend der Sendevorgang gestartet. Dabei werden in der Funktion nur bestehende Konstanten der Registeradressen verwendet. Diese Konstanten sind in der Datei *stm32f446xx.h* definiert. Soll der Programmcode nun auf einem STM32F401CE Mikrocontroller laufen, so muss lediglich die Datei *stm32f446xx.h* ersetzt werden. Voraussetzung dafür ist, dass alle Schnittstellen, die im Programmcode verwendet werden, auch in dem neuen Mikrocontroller existieren. Die HAL-Treiber und Definitionen der Mikrocontroller müssen in der Regel nicht händisch programmiert werden, sondern werden vom Hersteller des Mikrocontrollers zur Verfügung gestellt. STMicroelectronics bietet darüber hinaus kostenlose Software mit der der Umgang mit den hochkomplexen Mikrocontrollern vereinfacht wird.

4.2 STMicroelectronics CubeMX

Die Firma STMicroelectronics[®] bietet für Programmierung der eigenen Mikrocontroller und -prozessoren eine Reihe von kostenloser Software zum Download an. Darunter befindet sich unter anderem die Software CubeMX, die für jeden Mikrocontroller und jedes Entwicklungsboard von STMicroelectronics[®] verwendet werden kann. Mit der Software können Schnittstellen, Ein und Ausgangspins, Einstellungen der Taktgebung und vieles mehr in einer grafischen Benutzeroberfläche vor der eigentlichen Programmierung konfiguriert werden, was das Durchsuchen des Handbuchs nach den entsprechenden Registern überflüssig macht. CubeMX generiert .c und .h Dateien mit diversen Funktionen für die konfigurierten Schnittstellen und der Hardware. Abbildung 27 zeigt die Benutzeroberfläche zur Konfiguration der Taktgebung in CubeMX. Die Oberfläche ist nach der Flussrichtung der Taktsignale durch die Parameter und Umschalter angeordnet. Auf der linken Seite finden sich die Oszillatoren, dessen Frequenz durch die in der Mitte befindlichen Parameter und Umschalter geteilt oder multipliziert werden kann. Auf der rechten Seite befinden sich die ausgehenden Taktfrequenzen für die einzelnen Schnittstellen. Um den externen Hochfrequenzoszillator zu verwenden, wird im grün markierten Bereich der *HSE* Umschalter aktiviert und die Frequenz des Oszillators links neben dem grün markierten Bereich eingegeben. Damit der Mikrocontroller mit der maximalen Taktfrequenz von 180 MHz arbeitet wird das rot markierte Feld mit 180 beschrieben. CubeMX berechnet alle Parameter und Umschalter um die gewählte Taktfrequenz des Mikrocontrollers und der verwendeten Schnittstellen zu erreichen. Anhand der großen Anzahl an Parametern ist das richtige Einstellen der Taktfrequenzen per Hand nur mit sehr großem Aufwand zu bewerkstelligen. CubeMX unterstützt den Prozess des Programmierens auch in der Konfiguration von Schnittstellen. Der in dieser Arbeit verwendete STM32F446ZE Mikrocontroller besitzt 20 Schnittstellen welche ohne CubeMX einzeln durch das Beschreiben von Registern konfiguriert werden müssten. Mit CubeMX können diese Schnitt-

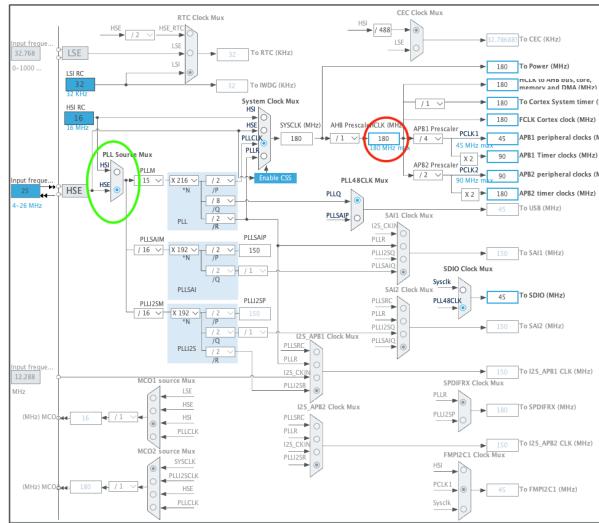


Abbildung 27: CubeMX Clock Configuration

stellen mithilfe einer grafischen Oberfläche eingestellt werden. Zudem schlägt die Software verwendbare Pins zu den jeweiligen Schnittstellen vor, welche durch die Zuweisung an einem virtuellen Mikrocontroller der Schnittstelle zugeordnet werden können. Die Pins des Mikrocontrollers besitzen unter Umständen mehrere Funktionen gleichzeitig, können jedoch nur von einer Schnittstelle parallel verwendet werden. Solche Überschneidungen werden von CubeMX erkannt und können manuell durch die Zuweisung eines anderen noch verfügbaren Pins behoben werden.

4.3 STMicroelectronics STM32CubeIDE

Nachdem der Grundcode von CubeMX generiert ist, wird der Programmcode mithilfe der Entwicklungs-Umgebung STM32CubeIDE von STMicroelectronics® erstellt. Sie ist speziell für die Entwicklung von Programmcode für STM32 Mikrocontroller und -prozessoren ausgelegt und beinhaltet neben der Kompilierung von Code auch Debugging-Funktionalitäten. Durch die Einbindung von CubeMX in der Entwicklungsumgebung wird keine weitere Software zum Programmieren eines STM32 Mikrocontrollers oder -prozessors benötigt. Jederzeit kann die Konfiguration in CubeMX geändert und ein neuer Grundcode generiert werden [17]. Code der vom Benutzer geschrieben wurde, wird bei diesem Vorgang nicht verändert. Voraussetzung dafür ist die Position des Benutzer-Programmcodes innerhalb der *USER CODE*-Blöcke (1).

Codeausschnitt 1: main.c: USER CODE Block

```
106 /* USER CODE BEGIN 0 */
107
108 /* USER CODE END 0 */
```

Code der zwischen der *BEGIN* und *END*-Zeile geschrieben steht, wird bei einer Neugenerierung des Grundcodes durch CubeMX beibehalten. Diese Blöcke finden sich an verschiedenen Stellen und Dateien wieder.

4.4 Taster

Um eine einwandfreie Funktion der Taster zu garantieren ist nicht nur deren Schaltung, sondern auch ein entsprechender Programmcode wichtig. Im folgenden Kapitel wird auf den Programmcode, der zum auslesen eines Tasterdrucks verwendet wird, eingegangen. Ein Tasterdruck kann zu jedem Zeitpunkt des

Programmablaufs geschehen und sollte möglichst vom Programmcode nicht unbeachtet bleiben. Um diese Funktionalität zu gewährleisten, sind die Taster mit interruptfähigen Eingängen des MCUs verbunden. Die Beschaltung des Tasters mit einem im MCU internen Pull-Up-Widerstand sorgt dafür, dass an den Tastern im nicht betätigten Zustand ein High-Pegel und während der Betätigung ein Low-Pegel anliegt. Der Interrupt muss also auslösen, wenn sich der Logikpegel eines Tasters von einem High- auf einen Low-Pegel ändert, also eine fallende Flanke des Taster-Signals erkannt wird. In CubeMX werden die Taster-Pins so konfiguriert, dass eine Interrupt-Routine beim Auftreten einer fallenden Flanke aufgerufen wird. Diese Routine ruft eine sogenannte Callback-Funktion auf, in der die Betätigung des Tasters bearbeitet wird. Code-Ausschnitt 2 zeigt einen Ausschnitt aus der Datei button.h. Jedem Taster wird ein Zahlenwert der Basis 2 zugewiesen. Diese Spiegeln die Bits 1-4 eines Bytes wieder.

Codeausschnitt 2: button.h: Definitionen und Funktionsprototypen

```

5 #define BACK      0x1
6 #define ENTER    0x2
7 #define UP       0x4
8 #define DOWN    0x8
9
10 uint8_t Button_pressed(uint8_t button);
11 void Button_reset();
12 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

Codeausschnitt 3 zeigt einen Teil der Callback-Funktion. Zunächst wird in Zeile 30 geprüft ob in den letzten 200 ms (*DELAYTIME*) bereits ein Aufruf der Funktion erfolgt ist. Diese Abfrage soll mehrfache Eingaben, augelöst durch ein mechanisches Schwingen des Tasters verhindern. Falls sie bereits aufgerufen wurde, so wird die Funktion beendet, da der Aufruf wahrscheinlich durch das mechanische Schwingen des Taster aufgerufen wurde. Falls nicht wird die Variable *last_updated* mit der aktuellen Zeit aktualisiert. Daraufhin wird mithilfe einer *switch*-Anweisung festgestellt welcher Taster den Interrupt ausgelöst hat. Dafür wird die Variable *GPIO – Pin*, welche von der Interrupt-Funktion an die Callback-Funktion übergeben wird, mit den einzelnen Pins der Taster verglichen.

Codeausschnitt 3: button.c: Codeausschnitt Taster-Interruptfunktion

```

28 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
29 {
30     if ((last_updated + DELAYTIME) > HAL_GetTick())
31         return;
32     last_updated = HAL_GetTick();
33     switch(GPIO_Pin)
```

Ist der auslösende Pin festgestellt, wird der Variable *presses* mithilfe einer bitweisen *oder*-Operation der entsprechende Zahlenwert des Tasters, der in der Datei *button.h* (Codeausschnitt 2 definiert ist, wie folgt zugewiesen.

Codeausschnitt 4: button.c: Bitweise Zuordnung eines betätigten Tasters

```

37     case BTN_BACK_Pin:
38     {
39         presses |= BACK;
40         return;
41     }
```

Der Vorteil der bitweisen *oder*-Operation ist, dass nur das entsprechende Bit gesetzt wird ohne alle anderen zu verändern. Ergänzende Abfragen ob das Bit bereits gesetzt ist entfallen. Im Hauptprogramm

werden ausschließlich die Funktionen *Button_pressed(uint8_t button)* für die Abfrage ob ein bestimmter Taster betätigt wurde und *Button_reset()* um die Variable *presses* zurückzusetzen verwendet. Bei der Abfrage eines Tasters wird das entsprechende Bit wieder zurückgesetzt. Wurden zwei verschiedene Taster innerhalb kurzer Zeit betätigt, können beide nacheinander oder gleichzeitig mit der Funktion *Button_pressed(...)* abgefragt werden. Codeausschnitt 5 zeigt den Inhalt dieser Funktion. Zunächst folgt eine *if*-Abfrage wann die letzte Tastereingabe registriert wurde. Ist die Eingabe nicht älter als der Wert von *TIMEOUT*, so wird zunächst die temporäre Variable *temp* mit dem Wert 0 beschrieben. Mithilfe einer bitweisen *und*-Operation der durch die Callback-Funktion beschriebenen Variable *presses* und dem Wert des abgefragten Tasters wird die Variable *temp* mit dem entsprechenden Wert beschrieben. Zuletzt werden die abgefragten Bits aus der Variable *presses* mit einer 0 überschrieben. Der Rückgabewert der Funktion ist die Variable *temp*. Im Hauptprogramm kann anhand eines Rückgabewertes ungleich 0 erkannt werden, dass der oder einer der abgefragten Taster betätigt wurde.

Codeausschnitt 5: button.c: Abfrage eines Tasters

```

10 uint8_t Button_pressed (uint8_t button)
11 {
12     if (!button || ((last_updated + TIMEOUT) < HAL_GetTick()))
13     {
14         Button_reset ();
15         return 0;
16     }
17     uint8_t temp = 0;
18     temp = presses & button;
19     presses &= ~button;
20     return temp;
21 }
```

Zur Veranschaulichung des Ablaufs der Funktion soll im Folgenden als Beispiel abgefragt werden, ob der Zurück-Taster betätigt wurde. Im Hauptprogramm wird die Funktion *Button_pressed(BACK)* dazu aufgerufen. Die Konstante *BACK* besitzt den Wert 1, also Binär 0000 0001. Kurz bevor die Funktion aufgerufen wurde, wurde der Zurück- und Bestätigung-Taster betätigt. Der Inhalt der Variable *presses* ist dadurch Binär 0000 0011. Die bitweise *und*-Operation der beiden Variablen ergibt Binär 0000 0001. Der Rückgabewert der Funktion ist somit 1 und damit kann im Hauptprogramm von einer Betätigung des Zurück-Taster ausgegangen werden. Wird nun allerdings die gleiche Funktion ein zweites Mal aufgerufen, ohne das der Zurück-Taster betätigt wurde, so ist der Inhalt der Variable *presses* Binär 0000 0010. Das Bit des Zurück-Taster wurde mithilfe der Operation in Zeile 19 zurückgesetzt. Die *und*-Operation hat dann das Ergebnis 0000 0000. Der Rückgabewert der Funktion ist null, der abgefragte Taster ist nicht betätigt worden.

Optimierungsmöglichkeiten

Die Konstante *TIMEOUT* besitzt keine Zuordnung zu einem einzelnen Taster. Wurde vor längerer Zeit ein Taster betätigt und kurz vor der Abfrage ein anderer Taster betätigt, so scheint die Betätigung des ersten Tasters auch innerhalb der Timeout-Zeit geschehen zu sein. Dieses Problem kann gelöst werden, indem eine Variable pro Taster angelegt wird, in der die Zeit der letzten Betätigung des jeweiligen Tasters gespeichert wird. Diese ersetzt die globale letzte Aktualisierungszeit *last_updated*. Der Nachteil dieser Methode ist eine deutlich aufwendigere Abfrage in der Funktion *Button_pressed(...)* und Zuweisung in der Callback-Funktion. Eine zweite Möglichkeit das Problem zu lösen ist der Einsatz eines Timers. Dieser wird gestartet wenn ein Taster betätigt wird. Läuft der Timer über, wird in der Interrupt-Funktion des Timers der Betätigungsstatus in der Variable *presses* für alle Taster zurückgesetzt. Soll nur der Zustand eines Tasters zurückgesetzt werden, so muss je ein Timer pro Taster zur Verfügung stehen, da in

der Interrupt-Funktion des Timers nicht unterschieden werden kann welcher Taster den Timer gestartet hat.

4.5 Encoder

Für den Programcode des Encoders kommt ein ähnliches Prinzip wie für die Taster im vorherigen Kapitel zum Einsatz. Der Aufruf der entsprechenden Funktion erfolgt über einen Interrupt. Wie bereits in Kapitel 3.4 beschrieben, muss der Zustand beider Terminals des Encoders ausgelesen werden um eine Drehrichtung bestimmen zu können. Terminal *A* ist in CubeMX als externer Interruptheingang mit Erkennung einer fallenden Flanke, Terminal *B* als Standard-Eingang konfiguriert. Wird durch eine fallende Flanke des Signals des Terminals *A* ein Interrupt ausgelöst, so entscheidet der Zustand des Terminals *B* über die Drehrichtung des Encoders. In der Interrupt-Funktion muss also lediglich der Zustand von Terminal *B* geprüft werden.

Codeausschnitt 6: Encoder.c: Interrupt-Funktion

```

45 void enc_Interrupt ()
46 {
47     if ((last_update + delaytime) > HAL_GetTick ())
48         return;
49     last_update = HAL_GetTick ();
50     if (HAL_GPIO_ReadPin(ENC_B_GPIO_Port, ENC_B_Pin))
51     {
52         if (enc_position > 0)
53             enc_position--;
54     }
55     else
56     {
57         enc_position++;
58     }
59     return;
60 }
```

Codeausschnitt 6 zeigt den Inhalt der Interruptfunktion. Zunächst wird in Zeile 47-49, wie auch in der Interrupt-Funktion der Taster, eine softwareseitige Entprellung durchgeführt. Die Verzögerungszeit fällt für den Encoder mit 50 ms deutlich kürzer aus als für die Taster, um auch schnelle Drehbewegungen registrieren zu können. Eine längere Verzögerungszeit hat einen Verlust von getätigten Eingaben, besonders bei höheren Drehgeschwindigkeiten, zur Folge. 50 ms haben sich in Tests als angenehm zu bedienen erwiesen und einen guten Kompromiss zwischen maximaler Eingabegeschwindigkeit und Zuverlässigkeit der Eingaben hergestellt. Nach der Entprellung folgt die Abfrage des Zustands des Terminals *B* mithilfe der Funktion *HAL.GPIO.ReadPin(...)*. Ist der Zustand *High*, so wird die global verfügbare Variable *enc_position* um 1 dekrementiert, wenn sie größer als 0 ist. Ist der Zustand *Low*, so wird die Variable um 1 inkrementiert. Bei der Variable *enc_position* handelt es sich um eine 16-Bit Variable ohne Vorzeichen, welche $2^{16} = 65536$ Werte umfasst und damit genug Platz für Eingaben bietet.

Optimierungsmöglichkeiten

Bei der Entwicklung des Schaltplans ist ein Fehler im Bereich der CR-Filter zum Entprellen des Encoders nach der Fertigung der Platine aufgefallen. Durch diesen Fehler wird die Filter-Beschaltung des Encoders unwirksam im Bezug auf dessen Entprellung. Mit der Behebung könnte die softwareseitige Entprellung reduziert werden oder sogar vollständig entfallen. Dadurch kann die Responsivität des Encoders verbessert

werden. Außerdem wäre eine Erkennung der Drehgeschwindigkeit sinnvoll um bei einer größeren Geschwindigkeit die Inkrement- und Dekrementgröße zu variieren, wodurch schnellere und präzise Eingaben gleichermaßen möglich sind.

4.6 Aufnahmefunktionen

Für die Aufnahme von DMX-Daten stehen insgesamt vier verschiedene Aufnahmemodi zur Verfügung, welche jeweils eine Aufnahmefunktion bilden. Der erste Aufnahmemodus ist der Standard-Modus, bei dem vorab die Dauer der Aufnahme festgelegt wird. Nach Ablauf der Aufnahmedauer stoppt die Aufnahme und kehrt zum Menü zurück. Der zweite Modus ist der sogenannte *Trigger*-Modus, bei dem zuvor in den Einstellungen ein *Trigger*-Kanal und *Trigger*-Wert festgelegt wird. Ist die Aufnahme gestartet und in den eingehenden DMX-Datenpaketen wird der *Trigger*-Wert des ausgewählten *Trigger*-Kanals überschritten, so startet ab diesem Zeitpunkt die Speicherung der DMX-Daten. Die Aufnahme wird gestoppt, wenn der *Trigger*-Wert wieder unterschritten wird. Dieser Modus ermöglicht das Starten und Stoppen der Aufnahme zu sehr präzisen Zeitpunkten. Vor allem für sich wiederholende Bewegungsabläufe von Lichttechnischen Geräten ist diese Funktion äußerst sinnvoll. Ist die Position eines Lichtkegels beispielsweise am Anfang und Ende der Aufnahme identisch, so kann die Aufnahme endlos nacheinander wiedergegeben werden, ohne dass ein Anfang oder Ende der aufgenommenen Sequenz sichtbar ist. Der letzte kontinuierliche Aufnahme-Typ ist die endlose Aufnahme, bei der die Aufnahme ohne die Wahl einer Aufnahmedauer gestartet wird und durch betätigen des Zurück-Tasters gestoppt wird. Dieser Modus richtet sich an Situationen, in denen die Endzeit der Aufnahme beim Aufnahmebeginn nicht festgelegt werden kann. Ein Beispiel für einen Anwendungsfall ist ein Konzert. Zu Beginn des Konzerts kann die Aufnahme gestartet werden und nachdem die letzte Zugabe gespielt ist, wieder gestoppt werden. Bei dem letzten Aufnahme-Typ handelt es sich um eine nicht-kontinuierliche Aufnahme der DMX-Daten, die sogenannte *Step*-Aufnahme. Bei diesem Aufnahme-Typ können einzelne Datenpakete des eingehenden DMX-Datenstroms aufgenommen werden. Jedes Datenpaket entspricht einem *Step* und enthält, im Gegensatz zu den drei anderen Aufnahmemodi, keine Zeitinformationen. Die einzelnen Steps werden bei der Wiedergabe in einem regelmäßigen Zeitabstand, der erst bei der Wiedergabe vom Benutzer festgelegt wird, wiedergegeben. Dieser Modus gibt dem Benutzer die Möglichkeit die DMX-Datenpakete in einer bestimmten Taktung auszugeben und diese beispielweise an den Takt eines Liedes anzupassen. Zudem wird durch die Speicherung einzelner Datenpakete, im Vergleich zu den anderen Aufnahmemodi, deutlich weniger Speicherplatz auf der SD-Karte benötigt. Im Folgenden wird auf das Grundprinzip der Aufnahmefunktionen eingegangen. Ein Flussdiagramm des Standard-Aufnahmemodus befindet sich im Anhang A.1.

Die drei Aufnahmemodi der kontinuierlichen Aufnahme, also Standard-, Endlose- und *Trigger*-Aufnahme unterscheiden sich im wesentlichen nur in den Start- bzw. Stoppbedingungen. Im Standardmodus muss zusätzlich eine Aufnahmezeit festgelegt werden.

Modus	Startbedingung	Stoppbedingung
Standard	Bestätigungstaster	Ablauf der Aufnahmezeit
Endlos	Bestätigungstaster	Zurück-Taster
<i>Trigger</i>	<i>Trigger</i> -Wert auf <i>Trigger</i> -Kanal überschreiten	<i>Trigger</i> -Wert auf <i>Trigger</i> -Kanal unterschreiten

4.6.1 Festlegen der Aufnahmezeit

Abbildung 28 zeigt die Ausgabe auf dem LCD-Display für die Festlegung der Aufnahmezeit. Die Auswahl der Zeit ist in vier Blöcke für das Einstellen der Stunden, Minuten, Sekunden und Millisekunden aufgeteilt. Mit den Links- und Rechts-Tasten kann der ausgewählte Block, markiert mit dem Zeichen in der untersten Zeile des Displays, verändert werden. Der Wert des Blocks wird mit einer Drehung des Enco-



Abbildung 28: LCD-Ausgabe Aufnahmzeit

ders im Uhrzeigersinn erhöht und gegen den Uhrzeigersinn verringert. Der Millisekunden-Block erhöht, bzw. verringert sich immer um 10 Millisekunden. Eine feinere Auflösung im Millisekundenbereich ist nicht sinnvoll, da die DMX-Datenpakete eine Zeitspanne von 22 ms zur vollständigen Übertragung benötigen. Wird eine Drehbewegung des Encoders registriert, so wird zunächst überprüft ob der vorherige Wert weiter erhöht oder verringert werden kann. Damit wird ausgeschlossen, dass eine Zeit kleiner null gewählt werden kann oder beispielsweise die Anzahl der Minuten größer 59 ist. Ist die Auswahl der Zeit beendet so muss sie mit dem Bestätigung-Taster bestätigt werden. Die eingegebene Zeit wird dann in Millisekunden umgerechnet und in die entsprechende Variable der Aufnahmzeit geschrieben.

Optimierungsmöglichkeiten

Um die Festlegung der Aufnahmzeit noch Benutzerfreundlicher zu gestalten, ist es sinnvoll die Begrenzungen der einstellbaren Zeit aufzuheben. Sind zum Beispiel 59 Sekunden eingestellt und der Wert wird mit einer Drehung des Encoder weiter erhöht, so wäre es sinnvoll die Anzahl der Minuten zu inkrementieren und die Anzahl der Sekunden auf null zurückzusetzen. Das gleiche Prinzip könnte auch in die andere Richtung funktionieren. Außerdem ist bei der Festlegung der Zeit die maximal mögliche Aufnahmedauer nicht berücksichtigt. Der Benutzer hätte also die Möglichkeit eine Aufnahmzeit festzulegen, die technisch aufgrund der maximalen Dateigröße auf der SD-Karte nicht erreicht werden kann. Dieser Fall sollte durch eine entsprechende Abfrage abgefangen werden.

4.6.2 Festlegen des Dateinamens

Jede Aufnahme erfordert die Festlegung eines Dateinamens um eine Auswahl bei der Wiedergabe zu ermöglichen. Der vor der Aufnahme festgelegte Name entspricht dem Dateinamen der Datei auf der SD-Karte und ermöglicht dem Benutzer ausgewählte Aufnahmen auf andere SD-Karten zu übertragen oder Sicherheitskopien anzulegen. Dem Benutzer stehen bei der Festlegung des Namens insgesamt acht Zeichen zur Verfügung. Abbildung 29 zeigt die Ausgabe auf dem LCD-Display. Wie bereits im vorherigen Kapitel, wird mithilfe der Links- und Rechts-Taster eine Auswahl getroffen, in diesem Fall ein Zeichen des Dateinamens. Durch drehen des Encoders können die Zeichen geändert werden. Verfügbare Zeichen sind alle Buchstaben von A-Z in kleiner und großer Schreibweise, Zahlen von 0-9, sowie ein Leerzeichen und ein Unterstrich. Umgesetzt wird dieser Programmteil mithilfe eines Zeichen-Arrays, dessen Anzahl



Abbildung 29: LCD-Ausgabe Dateinamen einstellen

an Zellen der Länge des Dateinamens entspricht. Die letzten vier Zellen des Arrays werden mit der

Dateiendung (*.dmx*) beschrieben. Die Endung ist bei der Auswahl des Namens nicht veränderbar. Die restlichen Zellen des Arrays werden initial mit einem Unterstrich beschrieben und sollen damit dem Benutzer ein veränderbares Zeichen signalisieren. Ein Zeichen besteht aus einem Byte und kann somit 256 Werte annehmen, wobei jeder Wert einem Zeichen entspricht. Wird zum Beispiel der Wert der einem *a* entspricht um 1 inkrementiert, so wird ein *b* dargestellt. Aufgrund dieses Systems kann der Wert der Position des Encoder unmittelbar als Zeichen dargestellt werden und in die entsprechende Zelle des Arrays geschrieben werden. Eine Veränderung der Encoderposition bewirkt eine Änderung des gerade ausgewählten Zeichens.

4.6.3 Aufnahme eines DMX-Datenpaketes

Die Aufnahme eines DMX-Datenpaketes zählt zu den kritischsten Zeilen Programmcode dieser Arbeit. Wird das erste Datenbyte nicht registriert, hat das bei der Wiedergabe, aufgrund der seriellen Übertragung des DMX-Protokolls, Auswirkungen auf alle angeschlossenen Geräte. Das Prinzip der Aufnahme von DMX-Datenpaketen ist bereits in der Praxisprojektarbeit aufgezeigt und wird in dieser Arbeit optimiert. Die Aufnahme eines Datenpaketes basiert auf den Interrupt-Events der UART-Schnittstelle. Bei jedem empfangenen oder gesendetem Datenbyte, erkannten Fehler der Übertragung oder der Schnittstelle wird ein globaler Interrupt ausgelöst. In der aufgerufenen Funktion wird identifiziert was den Interrupt ausgelöst hat und es wird ein entsprechender Code ausgeführt. Für die Aufnahme sind die Events des *Framing Errors* und des erfolgreichen Empfangs eines Datenbytes von Bedeutung. Wie in der Praxisprojektarbeit wird mithilfe des *Framing Errors* das Ende eines DMX-Datenpaketes erkannt. Codeausschnitt 7, 8, 10, 11 und 9 zeigen Ausschnitte aus der globalen UART-Interruptfunktion, die wesentlich für den Empfang von DMX-Datenpaketen sind. Die folgende Funktion wird aufgerufen wenn ein Datenbyte ohne einen Fehler vollständig empfangen und bereit zum Lesen ist.

Codeausschnitt 7: stm32f4xx_it.c: UART Save_Byte_Rx()

```

442 static void Save_Byte_Rx()
443 {
444     *huart4.pRxBuffPtr++ = huart4.Instance->DR; //DR in Buffer speichern
445     if(--huart4.RxXferCount == 0U)
446     {
447         //save to SD
448         HAL_UART_RxCpltCallback(&huart4);
449         Reset_Rx();
450     }
451 }
```

Zunächst wird in Zeile 444 das eingegangene Datenbyte aus dem *DR*-Register der UART-Schnittstelle dem Zeiger auf das Puffer-Array (*huart4.pRxBuffPtr*) zugewiesen und dieser anschließend um einer Stelle inkrementiert. Darauf folgt eine *if*-Abfrage, die Identifiziert, ob alle erwarteten Datenbytes empfangen sind. Bei einem Aufruf der Empfangsfunktion *HAL_UART_Receive_IT(...)* zum Starten des Empfangsvorgangs der UART-Schnittstelle wird die Anzahl der zu empfangenen Datenbytes der Variable *RxXferCount* zugewiesen und wird bei jedem empfangenen Datenbyte um eine Stelle dekrementiert. Enthält die Variable nach der Dekrementierung den Wert 0, so sind alle Datenbytes empfangen und ein entsprechender *Callback* wird aufgerufen. Dieser befindet sich in der Datei *DMX.c* und ist in Codeausschnitt ?? zu sehen.

Codeausschnitt 8: DMX.c: Callback-Funktion eingehender UART-Daten

```

103 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) //Aufruf wenn DMX Paket vollsta
104 {
```

```

105     if (Univers.recording == 1)
106     {
107         Univers.RxComplete = 1;
108         Univers.received_packets++;
109         HAL_GPIO_TogglePin(LED_RX_GPIO_Port, LED_RX_Pin);
110         return;
111     }
112     HAL_GPIO_TogglePin(LED_STATE_GPIO_Port, LED_STATE_Pin);
113 }
```

Wenn die Callback Funktion aufgerufen wird, ist ein DMX-Datenpaket vollständig empfangen. Dem Hauptprogramm wird das mithilfe der Zuweisung der Variable *RxComplete* mit dem Wert 1 signalisiert. Die Variable der Anzahl der empfangenen Datenpakete *received_packets* wird außerdem inkrementiert. Das Ein- bzw. Ausschalten der Empfangs-LED (*LED_RX*) gibt dem Benutzer eine Rückmeldung über den erfolgreichen Empfang eines DMX-Datenpaketes. Ist durch das Hauptprogramm die Variable *recording* nicht mit dem Wert 1 beschrieben, also findet keine aktive Speicherung der Daten statt, wird die Status-LED ein- bzw. ausgeschaltet. Der Benutzer bekommt somit eine Rückmeldung über den Datenfluss bevor eine Aufnahme aktiv gestartet wird. Somit kann sichergagengen werden, dass DMX-Daten fehlerfrei eingehen und bei einem Start der Aufnahme keine unerwarteten Fehler im Bezug auf den Datenfluss auftreten.

Nach der Bearbeitung des Callbacks wird als letzter Schritt die Funktion *Reset_Rx()* (Codeausschnitt 9) aufgerufen. In ihr werden alle Variablen der UART-Schnittstelle auf die für den Empfang eines neuen DMX-Datenpaketes benötigten Initialwerte zurückgesetzt.

Codeausschnitt 9: stm32f4xx_it.c: UART Reset_Rx()

```

434     static void Reset_Rx() //Rx complete or Error ->dmx-brake
435     {
436         huart4.RxXferCount = 513;
437         huart4.RxXferSize = 513;
438         huart4.pRxBuffPtr = Univers.RxBuffer;
439         (void)*huart4.pRxBuffPtr--;
440     }
```

Das zweite wichtige Event der UART-Schnittstelle ist der *Framing – Error (FE)* der das Ende des DMX-Datenpaketes signalisiert. Codausschnitt 10 zeigt den Auschnitt der Behandlung eines *FE* in der globalen Interrupt-Funktion der UART-Schnittstelle. Wird ein *FE* identifiziert, so muss zunächst das Bit, welches den Error anzeigt zurückgesetzt werden damit die Funktion nicht unmittelbar nach dem Beenden erneut aufgerufen wird.

Codeausschnitt 10: stm32f4xx_it.c: UART Framing Error

```

349 /* UART frame error interrupt occurred -----*/
350 if (((isrflags & USART_SR_FE) != RESET) && ((cr3its & USART_CR3_EIE) != RESET))
351 {
352     Clear_Rx_Error();
353     huart4.ErrorCode |= HAL_UART_ERROR_FE;
354 }
```

Mithilfe der Funktion *Clear_Rx_Error()*, dessen Inhalt Codesausschnitt 11 zeigt, wird durch einen Lesvorgang des entsprechenden Status-Registers *SR* und des Daten-Registers *DR* alle gesetzten Bits auf den Initialwert zurückgesetzt. Anschließend wird die Funktion *Reset_Rx()* aufgerufen. Zuletzt wird der *ErrorCode* der UART-Schnittstelle mit dem entsprechenden *FE* Error-Code beschrieben.

Codeausschnitt 11: stm32f4xx_it.c: UART Clear_Rx_Error()

```

426 static void Clear_Rx_Error()
427 {
428     uint16_t tmp = huart4.Instance->SR;
429     tmp = huart4.Instance->DR;
430     (void) tmp;
431     Reset_Rx();
432 }
```

Dieser Ablauf findet außerdem bei allen anderen auftretenden Fehlern statt um fehlerhafte Datensätze zu verhindern. Nur wenn ein vollständiges Datenpaket empfangen ist oder das Datenpaket durch einen *FE* als beendet erklärt ist, wird dieses für die Speicherung dem Hauptprogramm freigegeben:

Optimierungsmöglichkeiten

Aus programmatischer Sicht ist die Verteilung des Codes über mehrere Dateien nicht besonders elegant und erschwert die Implementierung in zukünftigen Projekten. Besser wäre es den Code vollständig in den Dateien DMX.c und DMX.h unterzubringen. Das erleichtert die Übersichtlichkeit des Programms und beugt damit Fehlern vor. Zudem ist es sinnvoll jeden der bei der Übertragung auftretenden Fehler anders zu behandeln um z.B. eine Fehlerkorrektur durchzuführen und unnötig großen Datenverlust zu verhindern. Anstatt ein komplettes Datenpaket zu ignorieren sobald ein Byte fehlerhaft ist, könnte der Wert des fehlerhaften Bytes aus dem vorherigen Datenpaket übernommen werden. Damit reduziert sich der Datenverlust von einem Datenpaket auf ein eventuell falsches Daten-Byte. Entsprechende Informationen über den Anteil von fehlerhaften Datenpaketen ober -Bytes könnten für den Benutzer ersichtlich dargestellt werden und eine Rückmeldung über die generelle Übertragungsqualität geben.

4.6.4 Speicherung der Aufnahmedaten

Abbildung 30 zeigt die Reihenfolge in der die einzelnen Bytes in der Aufnahme-Datei mit der Endung *.dmx* gespeichert werden. Alle aufgenommenen Datenpakete werden hintereinander in der Datei gespeichert. Begonnen wird mit dem Zeitpunkt an dem das Datenpaket vollständig empfangen wurde. Alle vier Bytes der 32-Bit Variable werden hintereinander gespeichert. Darauf folgen die eigentlichen DMX-Informationen der einzelnen DMX-Kanäle. Auch diese werden nacheinander gespeichert, beginnend mit dem DMX-Startbyte (Kanal 0). Nach dem letzten Datenbyte des DMX-Datenpaketes befindet sich das Schlusszeichen (NPC) des Datenpaketes. Direkt nach dem Schlusszeichen beginnt das nächste Datenpaket. Geht man

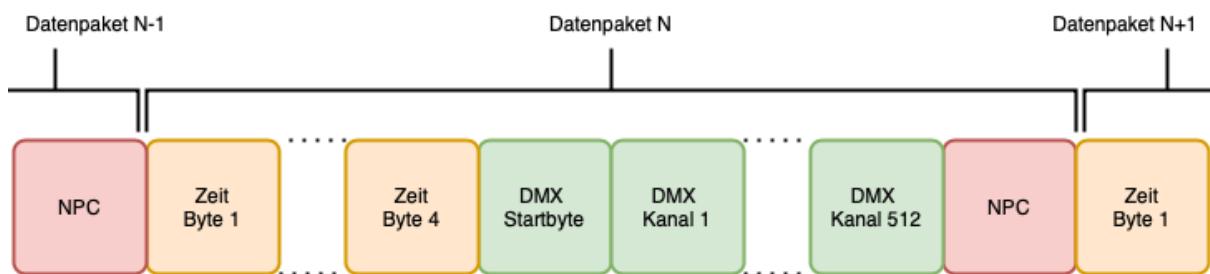


Abbildung 30: DMX-Aufnahmedaten Bytereihenfolge

davon aus, dass die DMX-Datenpakete mit der maximal möglichen Frequenz von ca. 44,1 Hz eingehen, so wird pro Minute ein Speicherplatz von:

$$44,1 \text{ Hz} * 518 \text{ Byte} * 60 \text{ s} = 1370628 \text{ Byte} = 1,37 \text{ MByte} \quad (3)$$

auf der SD-Karte benötigt. Eine SD-Karte mit einer Kapazität von 32 GByte bietet somit Speicherplatz für ca. 23346 Minuten Aufnahme. Das entspricht einer Aufnahmedauer von etwas über 16 Tagen. Begrenzender Faktor der maximalen Dauer einer Aufnahme stellt das *FAT*-Dateisystem dar. Es stellt nur 4-Bytes für die Angabe der Dateigröße zur Verfügung und begrenzt damit die maximale Dateigröße auf 4 GB [18, s. 128]. Eine Datei umfasst demnach maximal 2^{32} Byte. Das entspricht einer maximalen kontinuierlichen Aufnahmedauer von:

$$t_{Aufnahme_{max}} = \frac{2^{32} \text{ Byte}}{518 \text{ Byte} * 44,1 \text{ Hz} * 60 \frac{\text{s}}{\text{min}} * 60 \frac{\text{min}}{\text{h}}} = 55,227 \text{ h} \quad (4)$$

Diese Dauer sollte in den meisten Fällen ausreichend sein und könnte nur erweitert werden indem ein anderes Dateisystem verwendet wird, oder beim Erreichen der maximalen Dateigröße die Speicherung der Daten in einer zweiten Datei fortgeführt wird. Zusätzlich zu den DMX-Daten, werden generelle Informationen über die Aufnahme in einer separaten Datei mit der Endung *.info* gespeichert. Alle Informationen werden unabhängig vom Aufnahmemodus gespeichert. Darunter befinden sich die Aufnahmedauer, die Anzahl der gespeicherten Datenpakete und das NPC-Byte.

Optimierungsmöglichkeiten

Die Datenpakete werden unabhängig vom Informationsgrad gespeichert, was unnötig viel Speicherplatz erfordert. Denkbar ist ein Vergleich des aktuell zu speichernden DMX-Datenpaket mit dem zuvor gespeicherten DMX-Datenpaket. Beinhaltet das zuvor gespeicherte DMX-Datenpaket die selben Informationen wie das aktuelle, so könnte die Speicherung des aktuellen Paketes übersprungen werden, da es keine neuen Informationen enthält. Dieser Vorgang benötigt allerdings Ressourcen um den Vergleich der Datenpakte vorzunehmen. Durch den Vergleich und das eventuelle wegfallen von Speicherungsvorgängen könnten wiederum Ressourcen eingespart werden.

4.7 Wiedergabefunktion

Für die Wiedergabe der aufgenommenen DMX-Daten werden zwei Wiedergabemodi benötigt. Die Aufnahmedaten der Standard, Trigger und Endlosaufnahme werden für die Wiedergabe gleich behandelt. Die Aufnahmedaten der *Step*-Aufnahme wiederum müssen gesondert behandelt werden, da diese keine Zeitinformation enthalten. Für beide Aufnahmemodi muss zunächst die entsprechende Aufnahmedatei ausgewählt werden.

4.7.1 Auswahl der Aufnahmedatei

Um dem Benutzer eine Auswahl der Aufnahmedatei zu ermöglichen muss zunächst die SD-Karte nach vorhandenen Dateien durchsucht werden. Zudem müssen die Namen abgerufen und auf dem LCD-Display ausgegeben werden. Mithilfe der Funktion *f_findfirst(..)* kann die SD-Karte an einem bestimmten Dateipfad nach Dateien mit einem bestimmten Muster im Dateinamen durchsucht werden. Alle Aufnahmedateien werden mit der Endung *.dmx* gespeichert. Das Suchmuster wird mit **.dmx* festgelegt und bedeutet, dass jede Datei dessen letzten vier Zeichen *.dmx* beinhalten gefunden werden. Der Funktion *f_findfirst(...)* wird zudem ein *FILINFO* Struktur namens *info* übergeben. Wird eine Datei passend zu dem festgelegten Suchmuster gefunden, so wird unter anderem der vollständige Dateiname in dem Zeichen-Array *info.fname* gespeichert. Ist das Array leer, wurde keine Datei gefunden. Nachdem die erste Datei gefunden ist, müssen die restlichen auf der SD-Karte befindlichen Dateien gefunden werden. Dafür wird die Funktion *f_findnext(...)* verwendet. Diese Funktion wird so lange aufgerufen, bis keine weitere Datei gefunden wird. Nach jedem erfolgreichen Aufruf wird der Dateiname in ein multidimensionales Zeichen-Array gespeichert, welches 20 Dateinamen beinhalten kann. Der Inhalt dieses Arrays wird dazu benutzt die Dateinamen auf dem LCD-Display darzustellen. Durch drehen der Encoders wird die

Auswahl verändert. Ist eine Auswahl getroffen wird sie mit dem Bestätigen-Taster bestätigt. Die Auswahl der Aufnahmedatei kann jederzeit durch betätigen des Zurück-Taster abgebrochen werden.

Optimierungsmöglichkeiten

Die Auswahlfunktion ist durch die Größe des multidimensionalen Zeichen-Arrays auf die Anzeige von maximal 20 Dateinamen begrenzt. Für die Weiterentwicklung der Software ist es sinnvoll eine Lösung zu finden bei der die Anzahl der maximal anzuzeigenden Dateien nicht begrenzt ist.

4.7.2 Wiedergabe einer kontinuierlichen Aufnahme

Nachdem eine Aufnahmedatei erfolgreich ausgewählt ist, wird die zugehörige Info-Datei geöffnet und die Daten gelesen. Ist die darin befindliche Aufnahmedauer größer als null, handelt es sich um eine kontinuierliche Aufnahme. Für die Wiedergabe der Standardmodus verwendet. Ein Flussdiagramm der Funktion befindet sich im Anhang A.3. Als erstes wird ein Millisekunden Timer gestartet, woraufhin vier Bytes aus der Aufnahmedatei gelesen werden. Der Inhalt der ersten vier Bytes ist der bei der Aufnahme registrierte Zeitpunkt an dem das DMX-Datenpaket vollständig empfangen ist, wie in Kapitel 4.6.4 erläutert ist. Bei der Wiedergabe fungiert diese Zeit als Startzeitpunkt des jeweiligen Datenpaketes. Anschließend werden einzelne Bytes aus der Datei gelesen, bis entweder das Ende der Datei erreicht, die maximale Anzahl DMX-Kanälen eines Datenpaketes erreicht ist oder das NPC-Zeichen erkannt wird. Ist das Ende der Datei erreicht wird mithilfe der Funktion `f_lseek(...)` zum Beginn der Datei zurückgekehrt und der Zähler des gestarteten Timers auf 0 zurückgesetzt. Ist das NPC-Zeichen erkannt oder die maximale Anzahl an Bytes empfangen, so wird die am Anfang gelesene Startzeit des Datenpaketes mit dem Zähler des Timers verglichen, bis der Zähler den gleichen oder einen höheren Wert als die Startzeit besitzt. Erst danach wird das DMX-Datenpaket gesendet und die Funktion startet erneut mit dem Lesen der Startzeit des nächsten Datenpaketes. Die Wiedergabe kann durch die Betätigung des Zurück-Tasters beendet werden. Die Abfrage der Betätigung erfolgt jeweils vor dem Lesen der Startzeit.

4.7.3 Ausgabe eines DMX-Datenpaketes

Die Funktion zum Senden von DMX-Datenpakten ist im Vergleich zu der Praxisarbeit optimiert und vereinfacht. Der Vollständige Sendevorgang wird mit nur zwei Funktionen ausgeführt. Codeausschnitt 12 zeigt die Funktion zum Senden eines DMX-Datenpaketes, welche als einzige aufgerufen werden muss um ein Datenpaket zu senden.

Codeausschnitt 12: DMX.c: Transmit-Funktion

```

66 void DMX_Transmit(DMX_TypeDef* hdmx, uint16_t size)
67 {
68     HAL_GPIO_WritePin(DMX_DE_GPIO_Port, DMX_DE_Pin, GPIO_PIN_SET); // Treiber
69     DMX_set_TX_Pin_manual();
70     HAL_GPIO_WritePin(DMX_TX_GPIO_Port, DMX_TX_Pin, GPIO_PIN_SET); // Ausgang
71     htim11.Instance->CNT = 0;
72     CLEAR_BIT(htim11.Instance->SR, TIM_SR UIF);
73     SET_BIT(htim11.Instance->CR1, TIM_CR1_CEN);
74     while (!READ_BIT(htim11.Instance->SR, TIM_SR UIF));
75     CLEAR_BIT(htim11.Instance->SR, TIM_SR UIF);
76     HAL_GPIO_WritePin(DMX_TX_GPIO_Port, DMX_TX_Pin, GPIO_PIN_RESET);
77     DMX_set_TX_Pin_auto();
78     HAL_UART_Transmit_IT(Univers.uart, Univers.TxBuffer, size);
79 }
```

Die Übergabeparameter der Funktion sind eine DMX-Struktur (*struct*⁸) *DMXTypeDef* * *hdmx* und die Anzahl der zu übertragenden Datenbytes *uint16_t size*. In der DMX-Struktur befinden sich Informationen über die zu verwendende UART-Schnittstelle, sowie das Array *TxBuffer* mit den zu übertragenen Datenbytes. Zu Beginn der Funktion wird in Zeile 68 der Treiberchip über das Beschreiben des Pins *DMX_DE_Pin* mit einem High-Pegel aktiviert. Daraufhin wird der Ausgangspin der UART-Schnittstelle mithilfe der Funktion *DMX_set_TX_Pin_manual()* manuell beschreibbar gemacht und mit einem High-Pegel beschrieben. Das Zählerregister (*CNT*) des Timers 11 (*htim11*) wird mit dem Wert 0 überschrieben und das *UIF*-Bit *Update Interrupt Flag* im Statusregister des Timers zurückgesetzt. Dieses Bit wird hardwareseitig gesetzt wenn der Timer übergelaufen ist. Mit der Funktion *SET_BIT(...)* wird das *CEN* Bit gesetzt, wodurch der Timer startet. Der Timer ist zuvor in CubeMX als *one-pulse*-Timer konfiguriert. Sobald der Timer übergelaufen ist, wird er gestoppt und zählt den Zähler *CNT* nicht weiter hoch. Mit der *while*-Abfrage in Zeile 74 wird auf den Überlauf des Timers gewartet. Ist er übergelaufen, so wird das *UIF*-Bit wieder zurückgesetzt und der Ausgangspin in Zeile 76 mit einem Low-Pegel beschrieben. Zu diesem Zeitpunkt ist das DMX-Startsignal vollständig ausgegeben und die Datenbytes können gesendet werden. Um der UART-Schnittstelle wieder Zugriff auf den Ausgangspin zu gewähren wird mit der Funktion *DMX_set_TX_Pin_auto()* die manuelle Beschreibbarkeit des Pins deaktiviert. Anschließend wird mit dem Funktionsaufruf in Zeile 78 die Übertragung der Daten gestartet. Bei dieser Funktion handelt es sich um eine nicht-blockierende Funktion. Die Daten werden Interruptbasiert gesendet, während das Programm weiterläuft. Sind alle Daten vollständig gesendet, so wird durch einen Interrupt der UART-Schnittstelle die Funktion in Codeausschnitt 13 aufgerufen.

Codeausschnitt 13: DMX.c: Interrupt-Funktion ausgehender UART-Daten

```

120 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
121 {
122     if (Univers.sending == 1)
123     {
124         DMX_set_TX_Pin_manual();
125         HAL_GPIO_WritePin(DMX_TX_GPIO_Port, DMX_TX_Pin, GPIO_PIN_RESET); // Ausga
126         HAL_GPIO_TogglePin(LED_TX_GPIO_Port, LED_TX_Pin);
127     }
128 }
```

Der Inhalt der Funktion soll nur ausgeführt werden, wenn ein aktiver Sendevorgang ausgeführt wird, also wenn die Variable *Univers.sending* den Wert 1 enthält. Ist dies der Fall, so wird der UART-Ausgangspin ein weiteres Mal mit der Funktion *DMX_set_TX_Pin_manual* manuell beschreibbar gemacht und der Pin mit einem Low-Pegel beschrieben um das *Brake*-Signal des DMX-Protokolls zu senden. Anschließend wird der Zustand der entsprechenden LED invertiert um den Benutzer eine Rückmeldung über einen erfolgreichen Sendevorgang eines Datenpaketes zu geben.

Optimierungsmöglichkeiten

Wie auch in der Praxisarbeit wird das Signal ohne *Interbytedelay*, also Verzögerungszeit zwischen den einzelnen Datenbytes, gesendet. Dadurch wird das eingehende Signal nicht originalgetreu wiedergegeben, jedoch entsteht daraus kein Datenverlust oder eine Veränderung der Wiedergabefrequenz der Datenpakete. Außerdem wird in der aktuellen Version des Programmcodes das *Interbytedelay* während der Aufnahme nicht gemessen oder registriert. Um diese Funktionalität bei der Wiedergabe zu implementieren muss zunächst eine entsprechende Messung des *Interbytedelays* während der Aufnahme erfolgen.

⁸text

4.8 LCD

Für die Programmierung des LCD-Displays wird ein HAL-Treiber (HD44780-Stm32HAL) von Olivier Van den Eede verwendet und erweitert. Die Treiberdateien befinden sich im Anhang B. Der Treiber ist für die Steuerung von HD44780-Punktmatrixtreibern für LCD-Displays und ursprünglich für die Verwendung mit den Mikrokontrollern der STM32 F1-Serie entwickelt. Auf dem in dieser Arbeit verwendeten LCD-Modul befindet sich ein SPLC780-Treiberchip zum steuern des Displays und ist kompatibel mit den Anweisungen für HD44780-Treiberchips. Der Treiber kann LCD-Displays der Größe 16xN und 20xN Zeichen steuern. Standardmäßig ist die Größe 16x2 aktiviert, kann jedoch durch auskommentieren der entsprechenden Konstanten-Definition verändert werden.

Codeausschnitt 14: lcd.h: Einstellung Displaygröße

```
16 #define LCD20xN           // For 20xN LCDs
17 // #define LCD16xN        // For 16xN LCDs
```

Damit der Treiber mit den Mikrokontrollern der STM32 F4-Serie kompatibel ist, muss zudem die inkludierte Datei *stm32f1xx_hal.h* mit *stm32f4xx_hal.h* in der Header-Datei ersetzt werden.

Um das LCD-Display verwenden zu können benötigt der Treiber Informationen über die Verbindungen der Pins des LCD-Moduls mit den Pins des MCUs. Die Verbindungsinformationen werden für die Datenleitungen *D0* bis *D7* in zwei Arrays (*ports* und *pins*), jeweils eins für Pin und Port, global gespeichert. Der Zellenindex (0-7) der Arrays entspricht der jeweiligen Datenleitung (*D0-D7*). Zusätzlich zu den Informationen zu den Datenleitungen werden die Informationen für den *RS* und *EN*-Pin benötigt. Diese werden zusammen mit den Zeigern auf die Arrays *pins* und *ports* der Funktion *lcd_create(...)* (Codeausschnitt 15) übergeben. Außerdem wird der 8-Bit Modus des Treibers durch die Übergabe des Parameters *LCD_8_BIT_MODE* aktiviert um schnellere Schreibraten zu erreichen, wie bereits in Kapitel 3.5 erläutert.

Codeausschnitt 15: main.c: Funktionsaufruf lcd_create(...)

```
302 lcd = Lcd_create( ports , pins , LCD_RS_GPIO_Port , LCD_RS_Pin , LCD_E_GPIO_Port
, LCD_E_Pin , LCD_8_BIT_MODE ) ;
```

Soll ein Zeichen an einer bestimmten Stelle des LCD-Displays gezeigt werden, so muss zunächst der sogenannte *cursor* an der entsprechenden Stelle platziert werden. Für diesen Zweck ist die Funktion *Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col)* im Treiber zu finden. Mit dem Parameter *row* wird die Zeile und mit dem Parameter *col* die Spalte der Position des Cursors festgelegt, wobei der Wert 0 der ersten Zeile oder Spalte entspricht. Der Cursor verschiebt sich um eine Stelle nach rechts nachdem ein neues Zeichen dargestellt ist. Ist das Ende einer Zeile erreicht, wird der Cursor an den Anfang der nächsten Zeile verschoben. Diese Funktionsweise ermöglicht es einzelne bereits dargestellte Zeichen zu überschreiben, ohne den Inhalt des gesamten LCD-Displays erneut beschreiben zu müssen. Bei dem Beschreiben des Display sind des öfteren Fehler aufgetreten, bei dem zusätzliche, willkürliche Zeichen nach dem Ende der auszugebenden Zeichenkette ausgegeben werden. Der Grund für diesen Fehler ist die Unfähigkeit der Ausgabefunktion das Ende der Zeichenkette zu erkennen. Aus diesem Grund wird für die Ausgabe die zusätzliche Funktion *void Lcd_string_length(...)* in Codeausschnitt 16 dem Treiber hinzugefügt, bei dessen Aufruf die Anzahl der auszugebenden Zeichen als Parameter übergeben wird. Damit kann sichergestellt werden, dass nur eine bestimmte Anzahl an Zeichen ausgegeben werden.

Codeausschnitt 16: lcd.c: Funktion Lcd_string_length(...)

```
95 void Lcd_string_length (Lcd_HandleTypeDef * lcd , char * string , uint8_t length )
96 {
97     for (uint8_t i = 0; i < length; i++)
```

```

98     {
99         lcd_write_data(lcd, string[i]);
100    }
101 }
```

Umgesetzt wird diese Funktion mit einer einfachen *for*-Schleife, dessen Anzahl an Wiederholungen durch den Übergabeparameter *uint8_t length* bestimmt werden. Mithilfe der Funktion *lcd_write_data(..)* werden einzelne Bytes in das Datenregister des SPLC780-Chips geschrieben und auf dem LCD-Display ausgegeben. Zum Treiber ist außerdem eine Funktion hinzugekommen, mit der eine gesamte Zeile des LCD-Displays zurückgesetzt werden kann. Die entsprechende Zeile wird der Funktion übergeben und mithilfe einer *for*-Schleife jedes Zeichen der Zeile mit einem Leerzeichen überschrieben.

Optimierungsmöglichkeiten

Grundsätzlich bietet der Treiber eine stabile Grundlage für einfache Ausgaben auf dem Zeichen-LCD-Display, jedoch nur wenn die Funktion *lcd_string_length(...)* mit der entsprechenden Anzahl an Zeichen der auszugebenden Zeichenkette aufgerufen wird. Dafür muss während der Programmierung die Anzahl der Zeichen in der Zeichenkette händisch bestimmt und der Funktion übergeben werden und bietet damit ein großes Fehlerpotential. Besser wäre eine zuverlässige und automatische Erkennung der Zeichenkettenlänge in der Funktion *lcd_string(...)*.

4.9 SD-Karte (SDIO)

Die von CubeMX generierte Initialisierungs-Funktion der SDIO-Schnittstelle führt beim Start des Gerätes oft zu Fehlern, wodurch die SD-Karte weder gelesen noch beschrieben werden kann. In der Regel fällt das Programm dann in eine endlose Fehler-Funktion und ein Neustart des Geräts erwirkt keine Änderung des Zustands. Um dieses Problem zu lösen, wird nach den von CubeMX generierten Initialisierungsfunktionen im Hauptprogramm eine weitere Funktion zur Initialisierung der Verbindung mit der SD-Karte aufgerufen, dessen wesentlicher Inhalt in Codeausschnitt 17 zu sehen ist.

Codeausschnitt 17: main.c: SD Initialisierungs-Funktion

```

284     while(f_mount(&Univers.filesystem, Univers.path, 1) != FR_OK)
285     {
286         DSTATUS sd_state = disk_status(0);
287         if(sd_state == STA_NODISK)
288             Lcd_string_length(&lcd, "NO_SD", 5);
289         else if(sd_state == STA_NOINIT)
290         {
291             f_mount(0, Univers.path, 1);
292             memset(&Univers.filesystem, 0, sizeof(FATFS));
293             sd_state = SD_initialize(Univers.path);
294         }
295     }
```

Um eine SD-Karte verwenden zu können, muss diese als erstes mithilfe der Funktion *f_mount(...)* montiert werden. Dabei wird ein Dateisystem *filesystem* und ein Pfad *path* mit der Hardware verknüpft. Bei diesem Schritt treten in der Regel Fehler auf. Ist der Rückgabewert der Funktion *f_mount(...)* nicht *FR_OK* werden die Zeilen 285 bis 294 so lange wiederholt bis die SD-Karte erfolgreich montiert ist, denn ohne ein Speichermedium ist das Gerät in keinerlei Hinsicht sinnvoll verwendbar. Der aktuelle Status der SD-Karte wird als erstes der Variable *sd_state* zugewiesen. Befindet sich keine SD-Karte im SD-Kartenslot (*STA_NODISK*) wird eine entsprechende Meldung über das LCD-Display an den Benutzer

gegeben. Ist der Status der SD-Karte *STA_NOINIT*, so wird das Montieren der SD-Karte zunächst rückgängig gemacht, indem erneut die Funktion *f_mount(...)* aufgerufen wird, jedoch wird der Wert 0 als Dateisystem-Übergabeparameter der Funktion übergeben. daraufhin wird die Variable des Dateisystem mithilfe der *memset(...)* Funktion mit 0 initialisiert und eine erneute SD-Initialisierung gestartet. Zuletzt wird der Pfad mithilfe der Funktion *SD_initialize(...)* erneut initialisiert. Die Schleife startet erneut mit einem weiteren Versuch der Montierung.

Codeausschnitt 18 zeigt einen Beispielhaften Ablauf eines Lesevorgangs von der SD-Karte. Der Vorgang startet mit dem Öffnen einer Datei (*File*), zu sehen in Zeile 1009. Mit dem Parameter *FA_READ* wird der Funktion mitgeteilt, dass die Datei gelesen werden soll. Der Parameter *FA_OPEN_ALWAYS* sorgt dafür, dass die Datei geöffnet wird wenn sie existiert, ansonsten wird eine Datei mit dem entsprechenden Dateinamen (*hdmx->DMXInfoFile_name*) erstellt. In dem gezeigten Beispiel wird von der Existenz der Datei ausgegangen. Mit dem Parameter *FA_OPEN_EXISTING* wird eine bereits existierende Datei geöffnet. Falls sie nicht existiert gibt die Funktion einen entsprechenden Fehlercode zurück. Soll eine Datei beschrieben werden, muss der Parameter *FA_WRITE* übergeben werden.

Codeausschnitt 18: DMX.c: Beispiel Lesevorgang der SD-karte

```

1009     if ( f_open(&File , hdmx->DMXInfoFile_name , FA_OPEN_ALWAYS | FA_READ)
1010         == FR_OK)
1010     {
1011         f_read(&File , &hdmx->rec_time , 4 , &bytesread );
1012         f_read(&File , &hdmx->received_packets , 4 , &bytesread );
1013         f_read(&File , &hdmx->newpacketcharacter , 1 , &bytesread );
1014         f_close(&File );
1015         return 1;
1016     }

```

Nachdem die Datei geöffnet ist, werden Daten mit der Funktion *f_read(...)* aus der Datei gelesen. Der erste Übergabeparameter (*File*) ist das Objekt der Datei aus der gelesen werden soll. Darauf folgt die Adresse der Zielvariable in der die gelesenen Daten gespeichert werden sollen. An dritter Stelle steht die Anzahl der zu lesenden Bytes. Der letzte Übergabeparameter ist die Adresse einer Variable in der die Anzahl der tatsächlich gelesenen Bytes gespeichert wird. Durch einen nachfolgenden Vergleich der Soll und Ist Anzahl der gelesenen Bytes können Fehler, die beim Lesevorgang aufgetreten sind, identifiziert werden [19]. Nachdem alle Lesevorgänge beendet sind, muss die Datei mit der Funktion *f_close(...)* wieder geschlossen werden, da die Anzahl der gleichzeitig geöffneten Dateien begrenzt ist um Speicherprobleme auszuschließen.

Optimierungsmöglichkeiten

In der aktuellen Version des Programms wird der 1-Bit-Modus der SDIO-Schnittstelle verwendet. Von den vier vorhandenen Datenleitungen wird nur eine in diesem Modus verwendet. Auf der Platine sind zusätzliche Leiterbahnen für die Verwendung eines 4-Bit-Modus vorhanden. Scheinbar befinden sich Fehler in der SDIO-Bibliothek in CubeMX, wodurch der montierungs-Vorgang nicht erfolgreich durchgeführt werden kann. Durch die Verwendung des 4-Bit-Modus könnte die Lese- und Schreibgeschwindigkeit der SD-Karte im bestmöglichen Fall vervierfacht werden und trägt damit zur Stabilität während der Aufnahme und Wiedergabe bei.

4.10 Menü

Die Menüführung ist ein essentieller Bestandteil für die intuitive Bedienbarkeit des Gerätes. Sie muss leicht für den Benutzer zu durchschauen sein und mit den begrenzten Möglichkeiten der C-Porgrammiersprache

programmierbar sein. Für die Menüführung wird ein einfaches Modell aus mehreren Ebenen verwendet. Nach dem das Gerät gestartet und alle Schnittstellen initialisiert sind wird das Hauptmenü auf dem LCD-Display angezeigt. Der Inhalt des Display ist in Abbildung 31 zu sehen. Das Hauptmenü stellt Ebene 0, also die unterste Ebene des Menüs dar. Zu Beginn ist der erste Menüeintrag gewählt und wird mit einem Pfeil in der ersten Spalte des LCD-Displays markiert. Durch drehen des Encoders kann der Pfeil und damit der ausgewählte Eintrag verändert werden. Eine Drehung im Uhrzeigersinn verschiebt die Auswahl nach unten, eine Drehung gegen den Uhrzeigersinn nach oben. Ist die auszuführende Funktion mithilfe des Pfeils ausgewählt, so kann mit der Betätigung des Bestätigungs-Tasters die nächste Ebene des Menüs erreicht werden. Mithilfe des Zurück-Tasters kann zu der vorherigen Ebene zurückgekehrt werden. Je nach Auswahl werden die Inhalte aus Abbildung 33, 32 oder 34 angezeigt. Da auf dem LCD-Display lediglich vier Zeilen zur Anzeige stehen sind Abbildung 32 und 34 um die nicht sichtbaren Zeilen gestreckt. In Ebene 3 des Menüs befinden sich die tatsächlichen Funktionen zur Aufnahme und

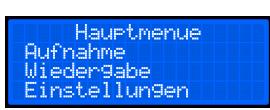


Abbildung 31: LCD-Hauptmenü



Abbildung 32: LCD-Aufnahme



Abbildung 33: LCD-Wiedergabe

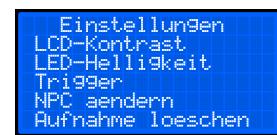


Abbildung 34: LCD-Einstellungen

Wiedergabe der DMX-Daten und den Einstellungen. Im Hauptprogramm wird lediglich die Funktion `main_menu(...)` aufgerufen, um das Hauptmenü anzuzeigen. Die Zeichenketten, welche über das LCD-Display ausgegeben werden, sind in mehrdimensionalen `char`-Arrays definiert (Codeausschnitt 19). Die erste Dimension des Arrays entspricht der Anzahl der anzulegenden Menüelemente, die zweite die Anzahl der zu beschreibenden Spalten des LCD-Displays. Es werden bewusst nur 19 Spalten der verfügbaren 20 mit den Menuelementen beschrieben, da vor jedem Element Platz freigehalten wird für den Pfeil, der die aktuelle Auswahl anzeigt. Um willkürliche Zeichen, die sich eventuell noch in den nicht beschriebenen Speicherzellen der Arrays befinden, werden die nicht benötigten Zeichen mit Leerzeichen überschrieben.

Codeausschnitt 19: menu.c: Definition Zeichenkettenarray für LCD-Display-Ausgaben

Durch die Verwendung der Zeichenkettenarrays können die Menüeinträge einfach verändert oder erweitert werden. Abbildung 35 zeigt den Ablauf der Funktion `main_menu(...)`. Zunächst werden alle Inhalte vom Display entfernt und die Menüüberschrift ausgegeben. Die globale Variable der Position des Encoders wird auf den Initialwert 0 zurückgesetzt. Daraufhin werden die ersten drei Menüeinträge aus dem Zeichenkettenarray unterhalb der Menüüberschrift ausgegeben (Abbildung 31). Der Wert der Encoderposition entspricht der aktuellen Auswahl des Benutzers. Initial gilt der erste Eintrag als ausgewählt. Um nicht bei jedem Durchlauf der Funktion den gesamten Displayinhalt erneut ausgeben zu müssen erfolgt eine Abfrage ob sich die Encoderposition geändert hat. Nur wenn er sich verändert hat, wird der Auswahlpfeil an der entsprechend resultierenden Stelle angezeigt. Ist der Enter-Taster betätigt, wird die nächste entsprechende Menüebene aufgerufen. Welche Menüfunktion aufgerufen wird, wird mithilfe einer `switch - case`-Anweisung der Encoderposition entschieden.

Um zu einer vorherigen Menüebene zurückzukehren müssen die Menüfunktionen durch einen *return*-Befehl oder durch Erreichen des Endes der Funktion beendet werden. Werden die vorherigen Menüebenen

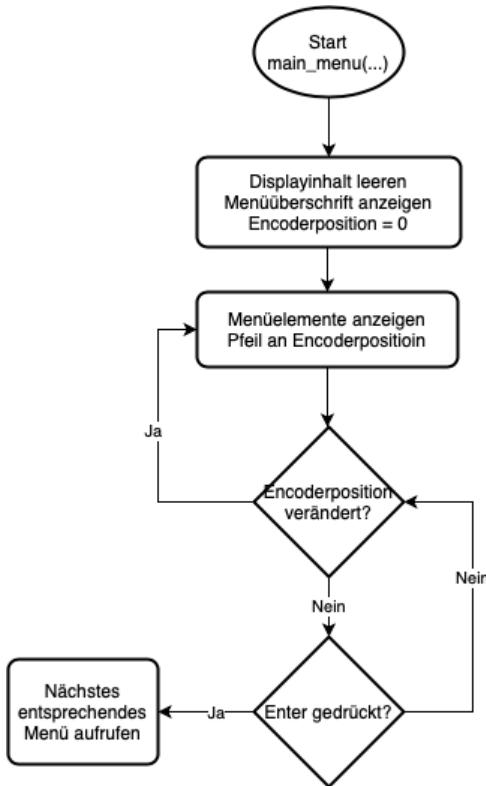


Abbildung 35: Flussdiagramm Hauptmenü

in der jeweiligen Funktion durch einen Funktionsaufruf aufgerufen kommt es zu einem rekursiven Aufruf von Funktionen. Zwangsläufig entstehen daraus Speicherprobleme, da reservierter Speicherplatz durch den Aufruf der Funktionen nicht wieder freigegeben wird. Ist der Speicher voll, so kommt es bei einem nächsten Funktionsaufruf zum Überlauf des Speichers und dadurch zu Fehlfunktionen oder vollständigen Abbruch des Programms.

4.11 Einstellungen

Dem Benutzer soll die Möglichkeit gegeben werden bestimmte Anpassungen am Gerät vorzunehmen, um die Funktionsweise und Eigenschaften des Gerätes auf die eigenen Bedürfnisse anzupassen. Im den folgenden Kapiteln wird auf die Einstellungsfunktionen eingegangen und dessen Funktion und Funktionsweise erläutert.

4.11.1 LCD-Kontrast & LED-Helligkeit

Das Gerät kann durch die zahlreichen Einsatzgebiete auch an vielen verschiedenen Orten eingesetzt werden. Um zu garantieren, dass die auf dem LCD-Display angezeigten Inhalte jederzeit erkennbar sind, soll der Benutzer die Möglichkeit haben den Kontrastwert des Displays einzustellen. Gleicher gilt für die Helligkeit der LEDs. Soll das Gerät an einem unauffälligen Ort platziert werden, könnten aufblinkende LEDs stören. Aus diesem Grund sind die Einstellungsfunktionen zum Einstellen des LCD-Kontrastes und LED-Helligkeit implementiert. In diesem Kapitel wird auf die Grundfunktionen beider Funktionen eingegangen. Beide Einstellungen basieren auf dem gleichen Prinzip, der PWM-Steuerung.

LCD-Kontrast

In der Regel wird mithilfe eines einstellbaren Spannungsteilers (Potentiometer) eine Spannung an den V0-Pin des LCD-Moduls angelegt und damit der Kontrast geregelt. Diese verschiedenen Spannungswerte

können auch mithilfe einer Pulsweltenmodulierten, festen Spannung erzeugt werden. Abbildung 36, 37 und 38 zeigen die Auswirkungen der Änderung der Pulswelte auf den Kontrast des LCD-Displays. Der

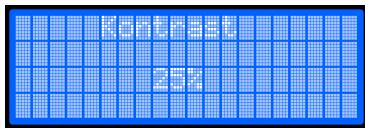


Abbildung 36: LCD 25% Kontrast

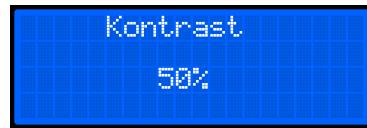


Abbildung 37: LCD 50% Kontrast



Abbildung 38: LCD 75% Kontrast

Kontrastwert wird mithilfe des Encoders verändert und wird entsprechend auf dem Display angezeigt. Der Wert wird erhöht bei einer Drehbewegung in Uhrezigersinn und verringert bei einer Drehbewegung gegen den Uhrzeigersinn.

Für die Erzeugung des Pulsweltenmodulierten Signals wird Timer 1 des MCUs verwendet, welcher mit einer Frequenz von 10 kHz Pulse erzeugt an einem Ausgangspin erzeugt, der mit dem V0-Pin des LCD-Moduls verbunden ist. Die Pulswelte wird durch das Beschreiben des *CCR*-Registers⁹ mit Werten von 0-100 eingestellt. In der Einstellungsfunktion entspricht der Wert des *CCR*-Registers dem Kontrastwert in %. Das Signal des Ausgangspins des Timer wird in Abbildung 39 mit einer Pulswelte von 25%, in Abbildung 40 mit einer Pulswelte von 75% gezeigt.

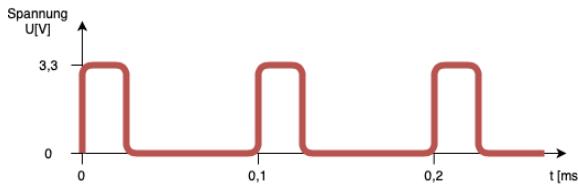


Abbildung 39: 10 kHz PWM-Signal mit 25% Pulswelte

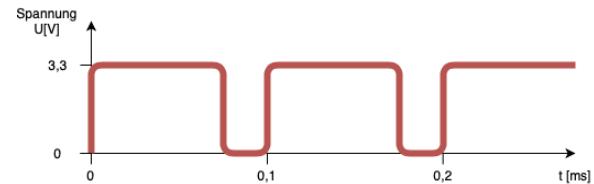


Abbildung 40: 10 kHz PWM-Signal mit 75% Pulswelte

LED-Helligkeit

Wie auch der Kontrast des LCD-Displays, wird die Helligkeit der LEDs mithilfe eines Pulsweltenmodulierten Signals eingestellt. Für die LEDs wird Timer 9 des MCU mit einer Pulsfrequenz von 12,5 kHz verwendet. Bei dieser Frequenz ist bei niedrigen Helligkeiten kein Flackern der LEDs zu erkennen. Um den Effekt der Einstellung sichtbar zu machen wird eine LED während der Einstellungsfunktion eingeschaltet. Der Unterschied zu der LCD-Kontrast-Einstellungsfunktion ist lediglich das zu beschreibende Register, da ein anderer Timer verwendet wird und die angezeigte Überschrift auf dem LCD-Display.

4.11.2 NPC ändern

Das Zeichen, welches in der Aufnahmedatei zum Signalisieren des Datenpaketendes genutzt wird (NPC-Zeichen), ist initial 1. Die eingehenden DMX-Datenpakete werden während der Aufnahme nach diesem Zeichen durchsucht und durch ein anderes Zeichen (0) ersetzt. In speziellen Fällen wird das NPC-Zeichen explizit für die Steuerung von DMX-Geräten benötigt. Um eine Aufnahme der Daten und eine vollständig kompatible Wiedergabe der Daten trotzdem zu ermöglichen, hat der Benutzer die Möglichkeit das NPC-Zeichen und das entsprechende Ersatzzeichen frei zu wählen. Abbildung 41 zeigt den Inhalt des LCD-Displays der NPC-Einstellungsfunktion. Das Zeichen in der untersten Zeile des Displays signalisiert die aktuelle Auswahl und kann mit den Links- und Rechts-Tasten verändert werden. Der ausgewählte Wert wird mit der Drehung des Encoders eingestellt. Ist eine Auswahl getroffen muss sie mit dem Bestätigen-Taster bestätigt werden. Daraufhin wird überprüft, ob beide Werte unterschiedlich sind. Sind beide

⁹Capture Compare Register



Abbildung 41: LCD-Ausgabe NPC Einstellung

Werte identisch wird eine Fehlermeldung über das LCD-Display ausgegeben. Andernfalls werden die Werte gespeichert und die Funktion kehrt zum Menü zurück. Die Funktion kann zudem jederzeit durch die Betätigung des Zurück-Tasters beendet werden.

4.11.3 Trigger

4.11.4 Aufnahme löschen

In dieser Einstellungsfunktion hat der Benutzer die Möglichkeit Aufnahmen zu löschen. Die Funktion durchsucht zunächst die SD-Karte nach Dateien mit der Endung `.dmx` und zeigt deren Namen auf dem LCD-Display an. Durch drehen des Encoders die zu löschen Datei ausgewählt werden, welche mit einem vor dem Dateinamen befindlichen Pfeil angezeigt wird. Für die Auswahl der Datei wird die selbe Funktion verwendet, die in der Wiedergabefunktion Anwendung findet (Kapitel ??). Wird die Auswahl mit dem Bestätigen-Taster bestätigt, wird der Benutzer durch eine entsprechende Anzeige auf dem LCD.-Display zum Bestätigen des Löschtorgangs aufgefordert. Wird auch diese Bestätigt wird die entsprechende Aufnahme- und zugehörige Info-Datei gelöscht, wird der Zurück-Taster betätigt, kehrt die Funktion zum Menü zurück. Der Löschtorgang wird mit der Funktion `f_unlink()` der FATFS-Bibliothek durchgeführt. Die Daten werden dabei nicht gelöscht, sondern lediglich die entsprechenden Speicherplätze wieder freigegeben, also aus der Dateizuordnungsliste des Dateisystems entfernt.

5 Zusammenfassung

Literatur

- [1] Felix Bauer. *Prototypentwicklung eines Aufnahme- und Wiedergabegerätes für DMX-Daten*. PhD thesis, Hochschule für angewandte Wissenschaft und Kunst Hildesheim, Holzminden, Göttingen, 2021.
- [2] STMicroelectronics. *STM32F446xC/E - Datasheet - production data*, 2021.
- [3] STMicroelectronics. *RM0390 Reference manual*, 2018.
- [4] Dietmar Ehrhardt. *PLL (Phase Locked Loop)*, pages 270–278. Vieweg+Teubner Verlag, Wiesbaden, 1992.
- [5] STMicroelectronics. *AN2867 Application note*, 2020.
- [6] Maxim Integrated. *MAX14945*, 2017.
- [7] Traco Electronic AG. *DC/DC Converter TEA 1 Series, 1 Watt*, April 2020. Rev. April 27, 2020.
- [8] ALPS Electric co., ltd. *Specifications No. 12E2006-3023*, November 2006.
- [9] Günter Kemnitz. *Technische Informatik*, volume Band. Springer, entwurf digitaler schaltungen edition, 2011.
- [10] M. Mitescu I. Susnea. *Microcontrollers in Practice*. Springer, 2005.
- [11] Ltd Shenzhen Eone Electronics co. *Specification For LCD Module 2004A*.
- [12] conrad.de. NE555 Schaltungen - Aufbau & Funktionsweise des Timers erklärt. abgerufen: 29.08.2021.
- [13] Inc. USB Implementers Forum. Universal Serial Bus Power Delivery Specification, January 2017.
- [14] Inc. USB Implementers Forum. Battery Charging Specification, March 2012.
- [15] Autodesk.com. What is EAGLE. abgerufen: 26.07.2021.
- [16] Felix Hüning. *Embedded Systems für IoT*. Springer Vieweg, 2019.
- [17] STMicroelectronics. STM32CubeIDE - Integrated Development Environment for STM32, 2021. abgerufen: 25.08.2021.
- [18] Christian Baun. *Betriebssysteme kompakt*. Springer Vieweg, 2 edition, 2020.
- [19] Elm Chan. FatFs - Generic FAT Filesystem Module. Webseite, April 2021. abgerufen: 19.08.2021.

Abbildungen

1	Schaltungsprinzip	4
2	STMicroelectronics STM32F446ZE Mikrocontroller Pinout [2, S. 41]	6
3	Beschaltung Quartz	6
4	MAX14945EWE+ Typical Application Circuit [6, s.19]	8
5	Encoder Schaltung [8]	9
6	Encoder-Signal bei Drehung im Uhrzeigersinn	10
7	LCD-Modul Vorderseite	11
8	LCD-Modul Rückseite	11
9	LED Schaltung	12
10	Schaltung Spannungsversorgung	14
11	EAGLE Symbol	14

12	EAGLE Footprint	14
13	EAGLE 3D-Modell	14
14	Modul "MAX"	15
15	Inhalt des Moduls "MAX"	15
16	Platinenvorderseite	16
17	Platinenrückseite	16
18	Platinenlayout-Aufteilung	16
19	Gehäuse 3D-Modell Frontseite	18
20	Gehäuse 3D-Modell Rückseite	18
21	Gehäuse Schnitt	18
22	Gehäuse Schnitt Draufsicht	18
23	Gehäuse Clip-Mechanismus Schnittansicht frontal	19
24	Gehäuse Clip-Mechanismus Schnittansicht	19
25	Gehäuse Clip-Mechanismus	19
26	Gehäuse Clip-Mechanismus Schnittansicht	19
27	CubeMX Clock Configuration	21
28	LCD-Ausgabe Aufnahmezeit	26
29	LCD-Ausgabe Dateinamen einstellen	26
30	DMX-Aufnahmedaten Bytereihenfolge	29
31	LCD-Hauptmenü	36
32	LCD-Aufnahme	36
33	LCD-Wiedergabe	36
34	LCD-Einstellungen	36
35	Flussdiagramm Hauptmenü	37
36	LCD 25% Kontrast	38
37	LCD 50% Kontrast	38
38	LCD 75% Kontrast	38
39	10 kHz PWM-Signal mit 25% Pulsweite	38
40	10 kHz PWM-Signal mit 75% Pulsweite	38
41	LCD-Ausgabe NPC Einstellung	39

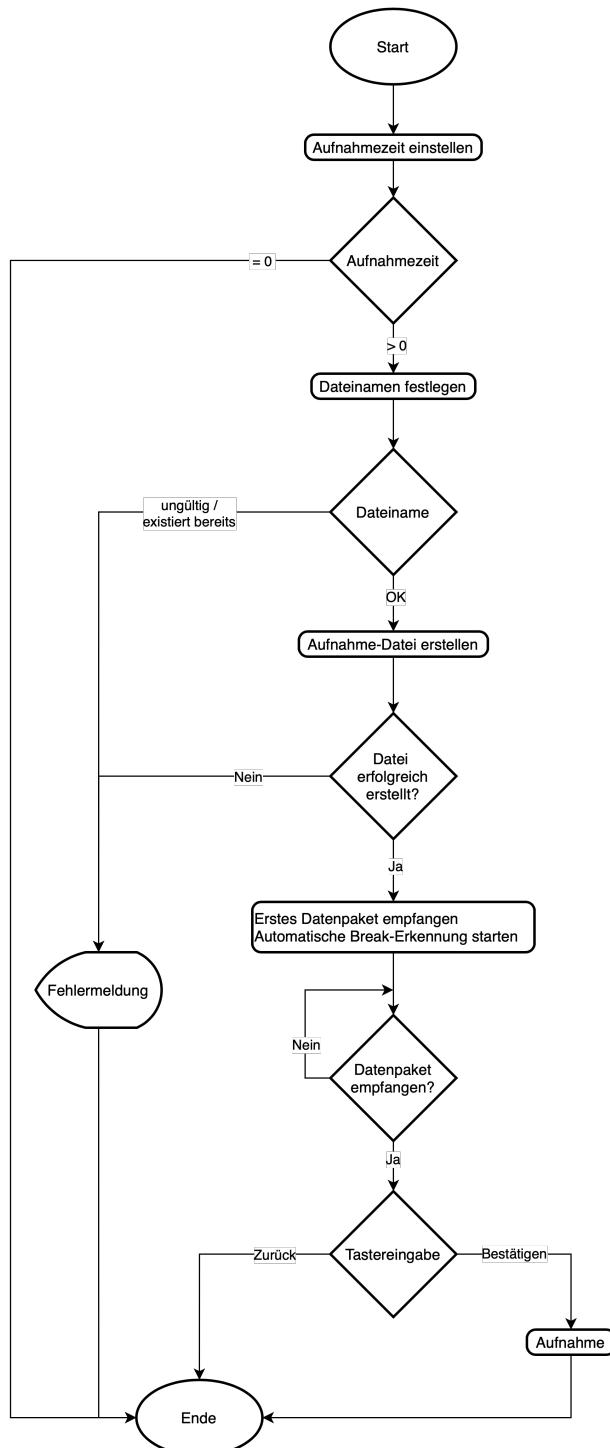
Codeausschnitte

1	main.c: USER CODE Block	21
2	button.h: Definitionen und Funktionsprototypen	22
3	button.c: Codeausschnitt Taster-Interruptfunktion	22
4	button.c: Bitweise Zuordnung eines betätigten Tasters	22
5	button.c: Abfrage eines Tasters	23
6	Encoder.c: Interrupt-Funktion	24
7	stm32f4xx_it.c: UART Save_Byte_Rx()	27
8	DMX.c: Callback-Funktion eingehender UART-Daten	27
9	stm32f4xx_it.c: UART Reset_Rx()	28
10	stm32f4xx_it.c: UART Framing Error	28
11	stm32f4xx_it.c: UART Clear_Rx_Error()	29
12	DMX.c: Transmit-Funktion	31
13	DMX.c: Interrupt-Funktion ausgehender UART-Daten	32
14	lcd.h: Einstellung Displaygröße	33

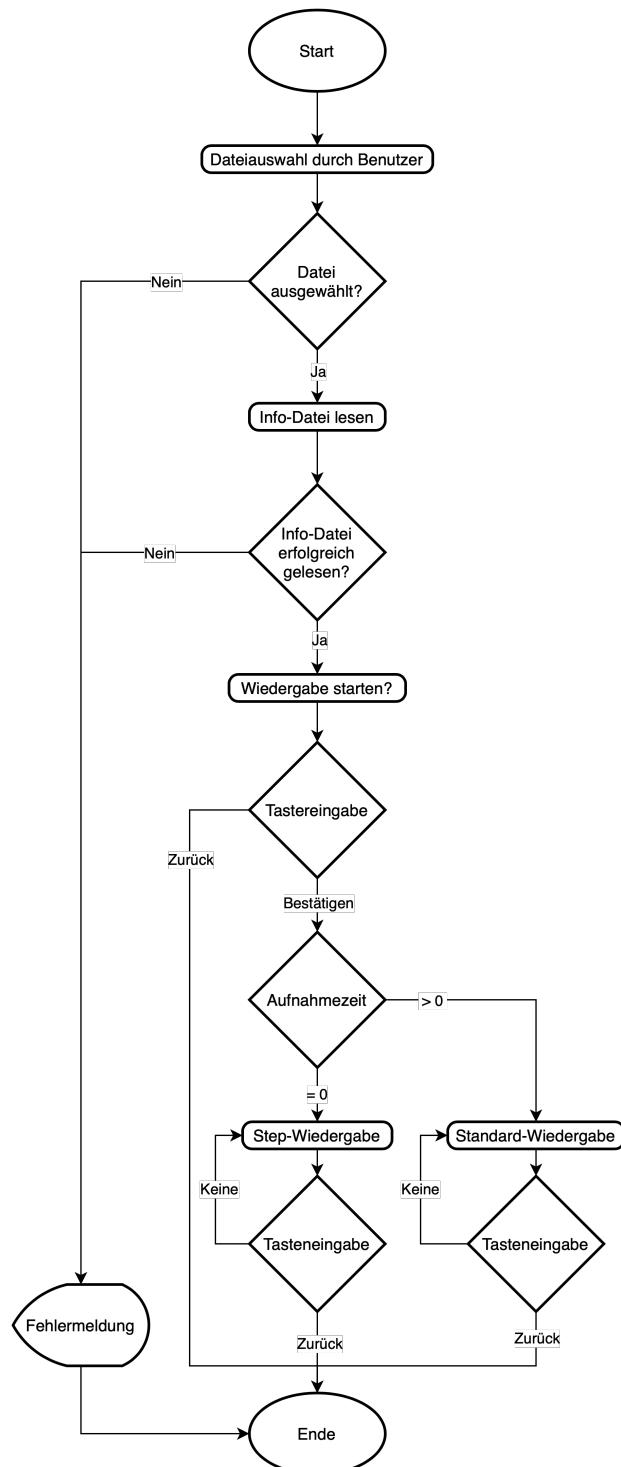
15	main.c: Funktionsaufruf lcd.create(...)	33
16	lcd.c: Funktion Lcd_string_length(...)	33
17	main.c: SD Initialisierungs-Funktion	34
18	DMX.c: Beispiel Lesevorgang der SD-karte	35
19	menu.c: Definition Zeichenkettenarray für LCD-Display-Ausgaben	36

A Flussdiagramme

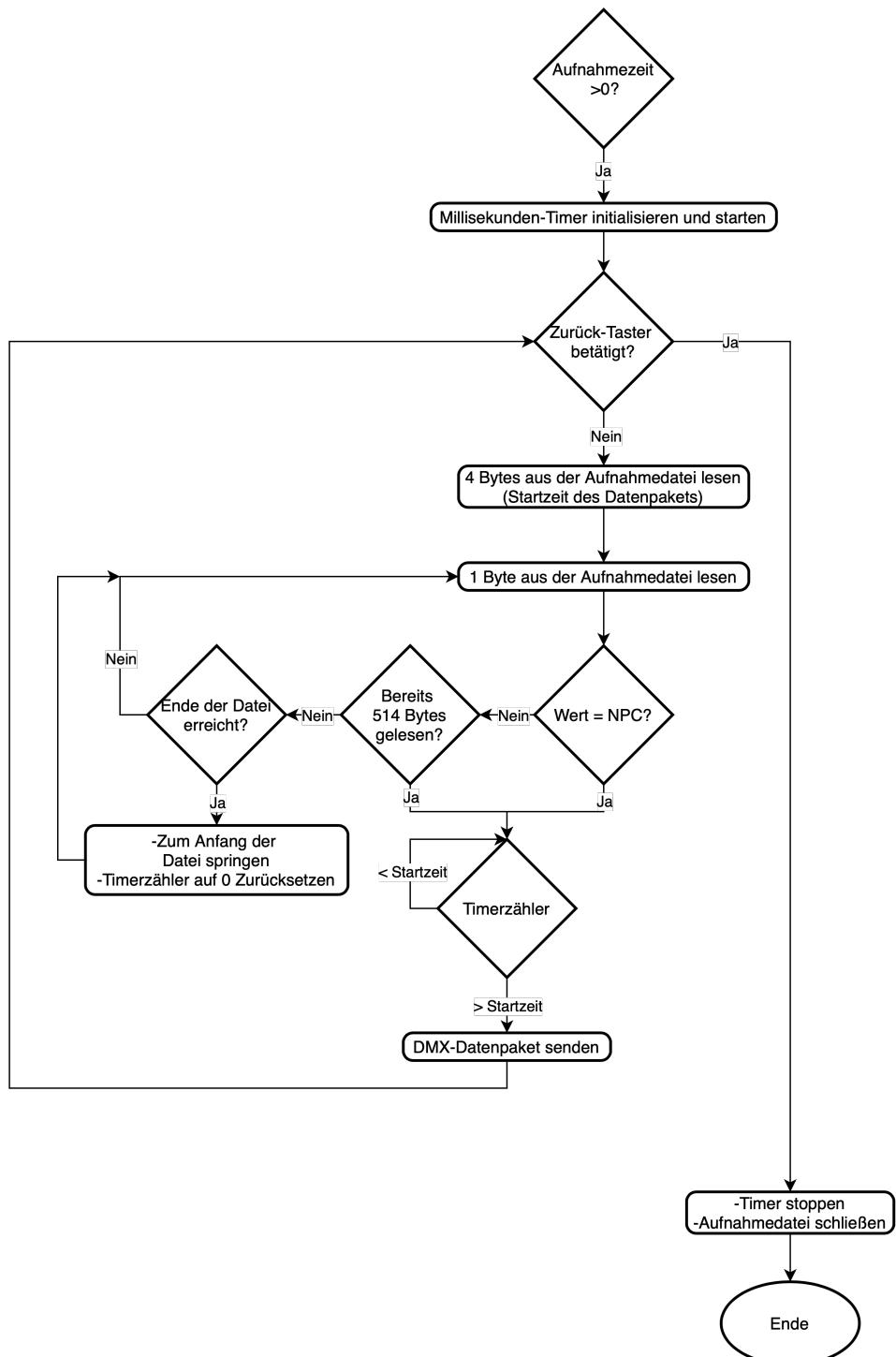
A.1 Aufnahmefunktion



A.2 Wiedergabefunktion



A.3 Standard Wiedergabe



B CD-Anhang

- lcdlib
- Partsbib
- Projekt
- 3D-Modelle aus Fusion
- Bilder der Schaltung & und Platinenlayout