Multi-Room Sound Adapter

Nico Lang, Philipp Immler

Februar 2025

1 Projekt

1.1 Projektteam

Nico Lang
Wirtschaftsingenieure/Betriebsinformatik
Grießau
6651 Häselgehr AT
Nico.Lang@hak-reutte.ac.at

Philipp Immler
Wirtschaftsingenieure/Betriebsinformatik
Hoheneggweg 21a
6682 Vils AT
Philipp.Immler@hak-reutte.ac.at

1.2 Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benützt habe. Die Stellen, die anderen Werken (gilt ebenso für Werke aus elektronischen Datenbanken oder aus dem Internet) wörtlich oder sinngemäß entnommen sind, habe ich unter Angabe der Quelle und Einhaltung der Regeln wissenschaftlichen Zitierens kenntlich gemacht. Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen. Für die Erstellung der Arbeit habe ich auch folgende Hilfsmittel generativer KI-Tools (ChatGPT 3.5) zu folgendem Zweck verwendet: Inspiration und allgemeine Information. Auch Übersetzer (DeepL) wurden zur Hilfe genommen. Die verwendeten Hilfsmittel wurden vollständig und wahrheitsgetreu inkl. Produktversion und Prompt ausgewiesen.

.....	.....
Ort, Datum	Unterschrift Schüler/in
	.....
	Unterschrift Schüler/in

1.3 Abstract Deutsch

1.4 Abstract English

1.5 Danksagung

Inhaltsverzeichnis

1 Projekt **2**

1.1 Projektteam 2

1.2 Eidesstattliche Erklärung 3

1.3 Abstract Deutsch 4

1.4 Abstract English 4

1.5 Danksagung 4

2 Einleitung **6**

2.1 Einleitung Hardware 6

2.2 Einleitung Software 7

3 Planung **7**

3.1 Festlegung Funktionsweise 7

3.1.1 Was soll das System können? 7

3.1.2 Was muss es nicht können? 9

3.1.3 Wie könnte man es erweitern? 9

3.2 Auswahl Hardwarekomponenten 9

3.2.1 Auswahl externe Hardware 9

3.2.2 Auswahl interne Hardware 10

3.3 Auswahl Technologien 11

3.3.1 Protokolle 11
3.4 Auswahl Softwaretools 13
3.4.1 Einleitung 13
3.4.2 Bibliotheken Microcontroller 15
3.4.3 Softwaretools Smartphoneapp 16

4 Entwicklung **17**

4.1 Entwicklung Software Adapter 17
4.1.1 Ziele 17
4.1.2 Programmablauf 17
4.1.3 Klassen 18
4.1.4 Herausforderungen 19
4.2 Entwicklung Smartphone-App 19
4.2.1 Struktur 19
4.2.2 Zielsetzung 20
4.2.3 UI Design 20
4.2.4 Funktionen 20
4.2.5 Aufbau 21
4.2.6 Komponenten 22
4.2.7 Seiten 24
4.3 Design Adaptergehäuse 25
4.3.1 Grundsätzlicher Aufbau 25
4.3.2 Virtuelles 3D-Design 25
4.3.3 Wärmeableitung 27
4.4 Fertigung Adaptergehäuse 27
4.4.1 Drucken des Prototyps 28

4.5 Zusammensetzen des Prototypen 28
4.5.1 Schaltplan 28
4.5.2 Verdrahten 29
4.5.3 Kleben 29

5 Testen und Fehlerbehebung 30

5.1 Testen des Gesamtsystems 30
5.2 auftretende Fehler beheben 30
5.3 Test auf Cybersecurity 30
5.4 Auftretende Sicherheitslücken schließen 30

6 Einzelnachweise 30

6.1 Literaturverzeichnis 30
6.2 Abbildungsverzeichnis 30
6.3 Anhang 30

2 Einleitung

Hier befindet sich die allgemeine Einleitung der Diplomarbeit.

2.1 Einleitung Hardware

Das Ziel des Hardware-Teils für folgende Diplomarbeit war, einen sinnvollen internen Aufbau des Geräts zu erzielen, die am besten geeigneten Hardware-Komponenten zu finden, das System bzw. die einzelnen Komponenten zusammenzusetzen und zu testen. Dieser Teil der Diplomarbeit wird von Nico Lang übernommen. Zudem beschäftigt sich dieser Teil mit dem Gehäuse des Geräts und bestimmt die technischen Anforderungen (Schnittstellen), die der Adapter letztendlich haben soll. Bei der Planung soll darauf geachtet werden, möglichst viele Kosten einzusparen, ohne dabei die Faktoren der Sicherheit und Qualität zu vergessen.

2.2 Einleitung Software

Das Ziel des Software-Teils für folgende Diplomarbeit war, einerseits die Software des Adapters, andererseits die Software der Smartphoneapp zu entwickeln. Dieser Teil der Diplomarbeit wird von Philipp Immler übernommen. Die Software des Adapters wird mit der Programmiersprache C++ codiert. Die Software der Smartphoneapp wird mit JavaScript codiert. Um eine bestmögliche Leistung und Effizienz zu garantieren, werden bei der Programmierung zahlreiche Bibliotheken und Frameworks verwendet. Bei der Entwicklung der Software wird ein großes Augenmerk auf Sicherheit und Effizienz gelegt.

3 Planung

3.1 Festlegung Funktionsweise

Beim Festlegen einer grundlegenden Funktionsweise des Adapters stellen sich vor allem folgende Fragen:

3.1.1 Was soll das System können?

Das Hauptziel ist, dass das System in verschiedenen, voneinander getrennten Räumlichkeiten bestimmte Audiosignale auf einen Line-Ausgang abspielen kann.

Line-Ausgang
Ein Line-Ausgang (Line-Out) ist eine Ausgangs-Schnittstelle für analoge Audiosignale, deren Ausgangsspannung immer grob dem Line-Pegel entspricht. Dieser „Line-Pegel beträgt etwa 0,5 Volt bis 1 Volt“.
Diese geringe Spannung reicht jedoch nicht, um das Audio-Signal auszugeben. Es muss zuerst noch durch einen Verstärker verstärkt werden. Solche Verstärker kommen einzeln also extern der Lautsprecher vor (passive Lautsprecher), sind jedoch häufig in einem Gehäuse mit den Lautsprechern inkludiert (aktive Lautsprecher).

(vgl. `https://www.monacor.de/magazin/audio-pegel`)

Aktive vs. Passive Lautsprecher
Ein klassisches Beispiel für passive Lautsprecher sind herkömmliche Hi-Fi Stereoanlagen. Diese bestehen meistens aus einem oder mehreren Playern, einem Verstärker und zwei oder mehreren (Surround Sound also Raumklang) passiven Lautsprechern. Der Player liest das Signal (beispielsweise einer CD oder

einer Schallplatte) und gibt den Line-Pegel über ein Kabel (im Hi-Fi Bereich meist Cinch oder 3,5mm Klinke) an den Verstärker weiter. Dieser Line-Pegel kann aber auch direkt aus einem TV-Gerät oder wie in unserem Fall aus einem Multi-Room Sound Adapter kommen. Der Verstärker verstärkt das Audiosignal nun von der geringen Spannung des Line-Pegels auf die für die Lautsprecher passende Spannung. Mit dem Lautstärkeregler am Verstärker kann man sich die Spannung (also Lautstärke) letztendlich noch auf persönliche Präferenzen anpassen.
Ein klassisches Beispiel für aktive Lautsprecher sind Bluetooth-Lautsprecher, deren Hauptziel es ist, möglichst kompakt und leicht transportierbar zu sein. Solche Bluetooth-Lautsprecher enthalten im Normalfall einen Akku, um auch unterwegs, ohne aktive Stromquelle, Musik hören zu können. Somit enthält das Gehäuse den Verstärker, die Lautsprecher, den Akku und sonstige Elektronik wie unter anderem ein Bluetooth-Modul. Hier fungiert meist ein herkömmliches Smartphone als Signalgeber, ob über Bluetooth oder 3,5mm Klinke bleibt dem/der Benutzer/-in überlassen.
Lenovo beschreibt Line-Ausgänge zum Beispiel folgendermaßen: „Der Line-Ausgang unterscheidet sich von anderen Audioausgängen wie z. B. Kopfhörerbuchsen, da er ein Signal mit festem Pegel liefert, das nicht von der Lautstärkeregelung Ihres Geräts beeinflusst wird. Er ist für den Anschluss an Geräte gedacht, die das Audiosignal verstärken oder weiterverarbeiten können."
Man kann also daraus schließen, dass man das Line-Out Signal des Multi-Room Sound Adapter vor dem Lautsprecher selbst noch verstärken muss. Wie genau, ist dem/der Endverbraucher/-in überlassen.

(vgl. `https://www.lenovo.com/at/de/glossary/line-out`)

Audioqualität
Zudem ist es wichtig, dass das System den Ton zuverlässig und möglichst flüssig überträgt und ausgibt.
blabla

Audio-Quellen
Grundsätzlich kann jeder beliebige Audio-Stream aus dem Internet verwendet werden. Das können beispielsweise Radiosender sein. Ein Beispiel für einen solchen Audio-Stream wäre der, des österreichischen Radio-Senders „OE3":
`https://orf-live.ors-shoutcast.at/oe3-q2a`

Benutzerfreundlichkeit
Es wird zudem viel Wert auf Benutzerfreundlichkeit gelegt. Das bedeutet, dass sich der Adapter zum einen leicht einrichten lässt, aber auch, dass er sich (mithilfe der Smartphone-App) einfach bedienen lässt.

(vgl. (Buch) `https://books.google.de/books?id=UI2INugaKwIC&pg=PA219#v=onepage`)

3.1.2 Was muss es nicht können?

Dieser Multi-Room Sound Adapter soll als Hi-Fi Produkt für den klassischen Durchschnittsbürger und/oder Musik-Liebhaber durchgehen. Aufgrund dessen wird die Bedienung sehr einfach und benutzerfreundlich, jedoch eindeutig nicht so präzise oder vielfältig einstellbar wie es bei professionellem Audio-Equipment der Fall ist. Während der Laie das Produkt einfach anstecken und benutzen möchte, hätte ein Audio-Nerd gerne noch einen eingebauten Acht-Band Equalizer und vieles mehr. Das ist jedoch nicht das Ziel dieser Diplomarbeit. Es geht eher darum, die Hauptfunktion, also Ton kabellos in Räume zu übertragen, und Einstellungsmöglichkeiten per App ohne großes Kopfzerbrechen zu ermöglichen.

3.1.3 Wie könnte man es erweitern?

Unsere Variante des Multi-Room Sound Systems zeichnet sich vor allem durch die beliebige Erweiterbarkeit aus. In der Theorie soll es ein einzelnes Modell, also den Adapter selbst geben. Mit jedem weiteren Adapter kann dementsprechend ein weiterer Lautsprecher oder ein Raum zugefügt werden. In Zukunft wäre es auch vorstellbar, dass man aus mehreren Adaptern Gruppen bilden kann, in denen die Adapter synchronisiert ist und somit der gleiche Audio-Stream auf mehreren Adaptern synchron läuft. Dies ist aber technisch sehr aufwendig, da die Latenz von WiFi ziemlich hoch ist.

3.2 Auswahl Hardwarekomponenten

Zur Auswahl der Hardwarekomponenten des Adapters wurde zu aller erst die externe Ausstattung des Adapters überlegt. Das bedeutet praktisch alles, mit dem ein Endverbraucher letztendlich zu tun hat. Dann kann der interne Teil, also die Technik dahinter, individuell auf die Anforderungen des externen Teils designet und entwickelt werden.

3.2.1 Auswahl externe Hardware

Der Adapter sollte ein möglichst kompakt konstruiertes und stabiles Gehäuse bekommen. An diesem wird ein einfacher Taster zur Interaktion angebracht. Mit dem Taster sollen einige Funktionen des Adapters ermöglicht werden. Beispielsweise per Klick, Doppelklick oder kurzem Halten. Da sich die Aufgaben des Tasters selbst gering halten werden (Verbindungsvorgang, Ein- und Ausschalten, ...) wird nur ein einziger Taster verwendet, um die Komplexität des Gesamtsystems zu senken. Die weitaus komplizierteren Funktionen sollen alle samt in der Smartphone-Applikation ermöglicht werden. Zusätzlich werden eine

oder mehrere Leuchtdioden zur Statusanzeige verbaut, um beispielsweise den aktuellen Verbindungsstatus zum Mobilgerät und zum Internet anzuzeigen.

Gehäuse
Das Gehäuse soll alle Komponenten auf möglichst kleinem Raum zusammenhalten, schützen und kühlen. Da sich Komponenten und möglicherweise auch das Design selbst laufend ändern, wird dieses deshalb erst gegen Ende des Projekts finalisiert werden können.

Taster
Für den Taster wird ein herkömmlicher Knopf in das Gehäuse eingelassen.

LED (Light-Emitting Diode)
here comes the led dadudada

3.2.2 Auswahl interne Hardware

Mikroprozessor
Als Herz des Systems wird ein ESP32 Mikroprozessor mit angebauter Platine verwendet. Der ESP32 ist ein weit verbreiteter Mikroprozessor. Man kann die Arduino IDE mit C++ als Programmiersprache zum programmieren verwenden. Zudem verfügt er schon Onboard über einen Hybrid WIFI- und Bluetooth-Chip, wodurch externe Module vermieden, und somit Platz eingespart werden kann. Espressif selbst beschreibt den ESP32 als optimal für IoT-Anwendungen; auch wegen der hohen Energieeffizienz. (vgl. Espressif Website)

Digital-/Analogwandler
Eine DAC-Decoderplatine wird verwendet, um das Audiosignal vom Arduino über eine angebaute Klinkenbuchse an die jeweilige Lautsprecherbox auszugeben.

Akku
Das Endprodukt soll mithilfe eines Akkus auch ohne Strom auskommen, dafür wird ein Lithium-Ionen-Akku mit 3,7 Volt verwendet. Akkus dieser Art zeichnen sich durch ihre hohe Energiedichte und, unter guten Umständen, hohe Lebensdauer aus. Man verwendet Li-Ionen-Akkus meist für (tragbare) Geräte in denen andere Akkus zu schwer oder zu groß wären. (vgl. `https://www.chemie.de/lexikon/Lithium-Ionen-Akkumulator.html`)

Natürlich haben Li-Ionen-Akkus auch gewisse Nachteile und bergen wie jeder andere Akku Gefahren. Beispiele dafür sind elektrische Überlastung, mechanische Beschädigung und thermische Überlastung:

Eine elektrische Überlast kann etliche Gründe haben, darunter:

- Verwendung eines falschen Ladegerätes
- Tiefenentladung
- Falsche Lagerbedingungen (z.B.: zu hohe Temperaturen)
  Zitat: „Hier kommt es zur Zersetzung der Elektrolytflüssigkeit und infolgedessen zur Bildung leicht brennbarer Gase. Wird anschließend versucht, die tiefentladenen Lithium-Ionen-Zellen wieder aufzuladen, kann die zugeführte Energie durch das Fehlen von Elektrolytflüssigkeit nicht mehr korrekt umgesetzt werden. Es kann zum Kurzschluss beziehungsweise zum Brand kommen.“ (vgl. `https://www.denios.de/services/denios-magazin/gefahren-im-umgang-mit-lithium-ionen-akkus`)

Eine mechanische Beschädigung jeglicher Art kann zu Kurzschlüssen im inneren der Zelle führen. Da unser Gerät nicht dafür gemacht ist, ständig in Bewegung zu sein, spielt dies keine zu große Rolle, es muss jedoch trotzdem ausreichend Schutz (durch das Gehäuse) vorhanden sein.

Wie oben schon kurz erwähnt, muss viel Wert auf die richtige Lagerung/Kühlung des Akkus gelegt werden. Wird dieser zu heiß (etwa durch den Mikroprozessor oder sonstige Bauteile) oder durch äußere Einflüsse beschädigt kann es zum Brand kommen.

Man kann daraus schließen, dass jeder kleinste Fehler beispielsweise zu einem Brand oder sogar einer Explosion des Akkus führen kann. Es ist daher wichtig, den Akku mit absoluter Vorsicht zu handhaben. Ausreichend Tests (Betriebstemperatur, etc.), richtige Konfiguration des Ladereglers und die Auswahl des Akkus sind ausschlaggebend für die Sicherheit des Endverbrauchers und dessen Umfeld.

(vgl. `https://www.denios.de/services/denios-magazin/gefahren-im-umgang-mit-lithium-ionen-akkus`)

Laderegler
Für einen optimalen Ladeprozess und Schutz des Akkus wird ein Laderegler verwendet. Dieser regelt den Ladevorgang des Akkus und hört auf zu laden, sobald er voll ist.

3.3 Auswahl Technologien

3.3.1 Protokolle

In diesem Kapitel geht es um die Recherche und Auswahl von Protokollen, die für den Austausch von Daten verwendet werden.

HTTP
Das Hyper Text Transfer Protocol ist ein weitverbreitetes Protokoll im Web und wird größtenteils für die Kommunikation zwischen Browsern und Webservern eingesetzt. Dabei basiert das Protokoll auf sogenannten „Requests“ (auf Deutsch: Anfragen). Es gibt zahlreiche Anwendungen für HTTP. Wir nutzen es einerseits als REST-API und andererseits für das Audio-Streaming. (vgl. URLPI02)

REST-API
Der Begriff „REST-API“ setzt sich aus den Abkürzungen „REST“ und „API“ zusammen. Dabei steht „REST“ für „Representational State Transfer“ (auf Deutsch: „gegenständliche Zustandsübertragung“) und „API“ für „Application Programming Interface“ (auf Deutsch: „Anwendungsprogrammierschnittstelle“). Eine REST-API zeichnet sich dadurch aus, dass sie eine einheitliche Schnittstelle zwischen Server und Client bietet. Dies wird durch die „Routes“ (auf Deutsch: „Routen“) geschaffen. Wenn der Client Requests an diese Routen sendet werden Aktionen auf dem Server ausgeführt. Es ist auch möglich, dass anschließend der Client Daten vom Server erhält. (vgl. URLPI04)

In unserer Diplomarbeit stellt der Microcontroller als Webserver eine REST-API zur Verfügung um so einheitlich Daten mit dem Client (Smartphone) auszutauschen. Folgende Routen sind dabei auf dem Webserver aufrufbar:

Route	**Anfragen-Typ**	**Funktion**
/getInfo	GET	Client bekommt Infos vom Microcontroller
/getAvailableNetworks	GET	Client bekommt eine Liste im JSON-Format, gefüllt mit SSID und RSSI (Stärke) von verfügbaren Netzwerken in der Nähe des Microcontrollers
/setWiFiCredentials	POST	Client sendet SSID und Passwort des gewünschten Netzwerks an den Microcontroller
/setStreamUrl	POST	Client sendet die URL des gewünschten Audio-Streams an den Microcontroller

Anwendung fürs Audio-Streaming
HTTP wird oft zum Streamen von Daten eingesetzt. Dies können Audio- oder auch Videodaten sein.

Beim Streaming wird grundsätzlich zwischen Live-Streaming und On-Demand-Streaming unterschieden. Beim Livestreaming werden die Daten gleich nach dem Erstellen an den Client gesendet. Ein Beispiel dafür ist das Streaming eines Live-Events. Das Video, welches von der Kamera eingefangen wird, wird direkt an

die Clients gesendet. Eine Alternative zum Livestreaming ist das On-Demand-Streaming. Dabei werden fertige Daten (z.B. Filme, Musik) auf einem Server gespeichert. Auf Anfrage eines Clients, werden diese in Teilstücke zerlegt und Stück für Stück an den Client gesendet. Dabei fügt der Client diese Stücke wieder zusammen und kann sie somit als Ganzes wiedergeben. Dies hat den Vorteil, dass die Daten nicht (oder nur kurz) auf dem Client gespeichert werden und es somit ressourcenschonend ist. Der Nachteil dabei ist, dass gerade bei größeren Datenmengen eine hohe Bandbreite benötigt wird. Dabei werden die Daten auch meist nicht (bzw. nur kurz) auf dem Client gespeichert. (vgl. URLPI05)

In unserer Diplomarbeit muss der Microcontroller fähig sein, Audiodaten von Livestreams und auch von On-Demand-Streams zu erhalten.Livestreams könnten dabei von Radiosendern stammen und On-Demand-Streams von Musikanbietern.

I2S
Das Inter IC Sound Protocol wird verwendet, um Stereo-Audio-Daten zwischen ICs auszutauschen. Es benötigt für die Datenübertragung folgende Leitungen:

- Taktleitung
- Wortauswahl
- mindestens eine Datenleitung

Die Datenübertragung erfolgt seriell und synchron. Seriell bedeutet, dass die Daten nur durch eine Leitung (die Datenleitung) übertragen werden. Synchron bedeutet, dass die Daten in einem bestimmten Takt übertragen werden. Dieser Takt wird von der Taktleitung vorgegeben. Die Leitung für die Wortauswahl wählt den Stereokanal aus (links oder rechts). (vgl. https://www.mikrocontroller.net/articles/I2S)
In unserem Projekt wird das I2S Protokoll verwendet, um die digitalen Stereo-Audio-Daten vom Microcontroller an den Digital-Analag-Wandler zu übertragen. Dabei werden die digitalen Buffer, die der Microcontroller vom Audio-Stream erhält, mittels I2S an den Digital-Analog-Wandler gesendet, welcher die digitalen Daten in analoge Daten umwandelt, so dass diese dann anschließend auf der Lautsprecherbox ausgegeben werden können.

3.4 Auswahl Softwaretools

3.4.1 Einleitung

In diesem Kapitel geht es um die Recherche und Auswahl von geeigneten Softwaretools, welche für die App-Entwicklung, als auch für die Entwicklung der Software des Microcontrollers verwendet werden. Zusätzlich werden auch die

Softwaretools zum Schreiben dieser Diplomarbeit kurz beschrieben.

LaTeX
"LaTeX ist ein hochwertiges Schriftsatzsystem, das Funktionen für die Erstellung technischer und wissenschaftlicher Dokumentationen enthält. LaTeX ist der Defacto-Standard für die Kommunikation und Veröffentlichung von wissenschaftlichen Dokumenten."(Übersetzung des englischen Originals von: https://www.latex-project.org/)
Wir haben uns für das Schreiben unserer Diplomarbeit in LaTeX entschieden, weil es sich sehr gut für wissenschaftliche Arbeiten eignet und wir somit schon damit vertraut sind, wenn wir es in der Zukunft brauchen.

draw.io "draw.io ist eine kostenlose Online-Diagrammsoftware zur Erstellung von Flussdiagrammen, Prozessdiagrammen, Organigrammen, UML, ER und Netzwerkdiagrammen."(Übersetzung des englischen Originals von: https://app.diagrams.net/)
Wir haben alle Diagramme unserer Diplomarbeit in draw.io erstellt, weil es einfach zu handhaben ist und es eine große Auwahl an Diagrammtypen und Formen gibt.

Visual Studio Code
"Visual Studio Code ist ein leichtgewichtiger, aber leistungsstarker Quellcode-Editor, der auf Ihrem Desktop läuft und für Windows, macOS und Linux verfügbar ist. Er bietet integrierte Unterstützung für JavaScript, TypeScript und Node.js und verfügt über ein umfangreiches Ökosystem von Erweiterungen für andere Sprachen und Laufzeiten (wie C++, C#, Java, Python, PHP, Go, .NET)."(Übersetzung des englischen Originals von: https://code.visualstudio.com/docs)
Wir haben Visual Studio Code als IDE für unsere Diplomarbeit gewählt, weil durch die unzähligen Erweiterungen viele verschieden Programmiersprachen und Bibliotheken unterstützt und wir auch schon etwas Erfahrung damit haben.

GitHub
"GitHub ist eine webbasierte Schnittstelle, die Git verwendet, die Open-Source-Software zur Versionskontrolle, mit der mehrere Personen gleichzeitig separate Änderungen an Webseiten vornehmen können. Wie Carpenter anmerkt, fördert GitHub die Zusammenarbeit von Teams bei der Erstellung und Bearbeitung von Website-Inhalten, da es eine Zusammenarbeit in Echtzeit ermöglicht."(Übersetzung des englischen Originals von: https://digital.gov/resources/an-introduction-github/)
Wir verwenden GitHub für die Verwaltung unseres Codes und unserer Dokumente. Der Vorteil dabei ist, dass jedes Projektmitglied auf seinem lokalen PC an den Dokumenten arbeiten kann und die Änderungen dann per GitHub synchronisiert werden können.

DeepL Wir verwenden DeepL um englische Texte, welche für unsere Diplomarbeit relevant sind, ins Deutsche zu übersetzen. Wir haben uns für DeepL entschieden weil dieser einer der genauesten Übersetzer ist und man dießen au-

ßerdem kostenlos nutzen kann.

3.4.2 Bibliotheken Microcontroller

Im folgenden werden die verwendeten Bibliotheken im Code des Microcontrollers aufgezählt und kurz beschrieben:

Arduino
Die Arduino-Bibliothek wird verwendet um den ESP32 ähnlich wie einen Arduino programmieren zu können. Es erleichtert dabei die Programmierung enorm, vorallem dann, wenn man schon Vorerfahrung mit der Programmierung von Arduinos hat.

WiFi
`https://github.com/arduino-libraries/WiFi`
Die WiFi-Bibliothek wird verwendet, um die Funktionen der eingebauten WiFi-Antenne des ESP32 zu verwenden. Der ESP32 kann dabei entweder als Access Point oder als Client fungieren. Wenn er als Access Point fungiert, stellt er ein eigenes WiFi-Netzwerk bereit, mit dem sich andere Geräte verbinden können und der ESP32 somit einen Host darstellt. Als Client kann er sich mit anderen WiFi-Netzwerken bzw. Access Points verbinden. In unserem Projekt fungiert der ESP32 sowohl als Access Point, als auch als Client.

ArduinoJson
Die ArduinoJson-Bibliothek wird verwendet, um Daten in das JSON vormat zu kodieren. Der Vorteil dabei ist, dass JSON ein weit verbreitetes Format in der Informatik ist und deshalb mit vielen Schnittstellen kompatibel ist. In unserem Projekt wird die ArduinoJson-Bibliothek für den einheitlichen Datenaustausch zwischen Webserver (ESP32) und Client (Smartphone) verwendet.

WebServer
Die WebServer-Bibliothek wird verwendet, um einen Webserver auf dem ESP32 bereitzustellen. Dieser ist wichtig für den Datenaustausch mittels HTTP, zwischen ESP32 und Smartphone. Mithilfe des Webserver wird einerseits das WiFi-Passwort des gewählten WLANs, als auch die URL des Audiostreams an den ESP32 geleitet. In die andere Richtung, werden grundlegende WiFi-Informationen und verfügbare WiFi-Netzwerke vom ESP32 bereitgestellt.

Audio
Die Audio-Bibliothek wird verwendet, um die MP3-kodierten Audio-Daten von einem HTTP-Stream zu empfangen, diese in PCM-Daten umzuwandeln und diese dann an den Digital-Analog-Wandler per I2S zu senden. Der ESP32 verfügt bereits standartmäßig über Funktionen, mit deren Hilfe man Audiodaten mittels I2S übertragen kann. Allerdings bereitet dies einen viel höheren Aufwand

für Konfiguration usw. Daher verwenden wir in unserem Projekt die Audio-Bibliothek.

3.4.3 Softwaretools Smartphoneapp

Die Smartphoneapp wurde mit "React Nativeëntwickelt. Mithilfe von React Native ist es möglich eine zentrale Applikation zu entwickeln und diese dann auf mehreren Plattformen wie IOS, Android und auch im Web zu verwenden. React Native basiert auf React, welches ein Framework für die Frontend-Entwicklung von Web-Applikationen ist. Außerdem wird die Radio-Browser-API für die Bereitstellung diverser Internetradios verwendet.

React Native
React Native ist ein Framework, welches die plattformübergreifende Entwicklung von Apps ermöglicht. Das heißt, man schreibt einen Code und kann diesen dann für IOS, Anrdoid und fürs Web verwenden. Der Code wird in JavaScript geschrieben. React Native wurde erstmals 2015 von Meta (damals noch Facebook) als Open-Source-Projekt veröffentlicht. Seither wird es weiterhin von Meta instandgehalten und hat eine riesige Community, welche ständig neue Bibliotheken für das Framewok veröffentlicht. React Native basiert auf React, welches man bereits aus der Webentwicklung kennt. Der Vorteil von React im Gegensatz zur normalen Webentwicklung ist, dass man wiederverwendbare Komponenten bauen kann. Dies ist auch mit React Native möglich. In React Native stehen dabei einige Standardkomponenten zur Verfügung, welche dann jeweils in native Komponenten, passend für das jeweilige Betriebssystem, gerendert werden. Wir haben React Native für unsere Smartphoneapp verwendet, weil wir nicht für jede Plattform eigenen Code schreiben wollten. Außerdem haben wir auch schon etwas Erfahrung mit JavaScript und React, was uns den Einstieg erleichterte. (vgl. URLPI07)

Expo
Um das Entwickeln der Smartphoneapp noch einfacher bzw. effizienter zu gestalten, wurde das Expo Framework verwendet. Dieses Framework stellt wichtige Funktionen und eine Projektstruktur standardmäßig zur Verfügung. Dies hat den Vorteil, dass man dann nicht mehr von 0 anfangen muss, sondern schon einen guten Grundaufbau hat. Ein weiterer Vorteil von Expo ist das File-basierte Routing. Mithilfe diesem, ist es möglich die Navigation in der App an die Dateien im Projekt anzupassen. (vgl. URLPI08)

4 Entwicklung

4.1 Entwicklung Software Adapter

In diesem Kapitel wird der Übergang der Planung in die Entwicklung der Software des Adapters beschrieben. Zur Entwicklung der Software des Microcontrollers wurde die IDE Visual Studio Code in Verbindung mit dem Framwork PlatformIO verwendet. Um die Entwicklung in C++ zu ermöglichen und vorgefertigte Bibliotheken zu verwenden, wurde das Arduino-Framework verwendet.

4.1.1 Ziele

Der Adapter soll grundsätzlich folgende Funktionen bereitstellen:

- als AP fungieren, um Verbindung mit Smartphone herzustellen
- mit WLAN verbinden
- Audio-Stream aus Internet empfangen
- Audio-Stream dekodieren
- dekodierten Audio-Stream an Lautsprecherbox übertragen

4.1.2 Programmablauf

Der Ablauf des Programmes wird mit folgendem UML-Ablaufdiagramm veranschaulicht:

Start
Der Microcontroller wird durch einmaliges Drücken auf den Knopf aus dem Standby (Deep Sleep) Modus erweckt.

Lesen der WiFi-Zugangsdaten
Es wird im EEPROM des Microcontrollers nach einer gespeicherten SSID und nach einem gespeicherten Passwort gesucht. Wenn die Zugansdaten nicht vorhanden sind wird die Status-LED auf rot geschalten. Anschließend werden keine weiteren Prozesse ausgeführt, bis der/die Benutzer/in mit einem weiteren Druck auf den Knopf in den Konfigurationsmodus schaltet. Wenn die Zugangsdaten allerdings gespeichert sind versucht der Microcontroller sich als Client mit dem WLAN zu verbinden. Wenn dies erfolgreich ist, wird in den Standardmodus gewechselt. Wenn die Verbindung allerdings fehlschlägt, wird in den Konfigurationsmodus gewechselt.

Standardmodus
Sobald der Standardmodus aktiviert wird, wechselt die Farbe der Status-LED auf grün. Anschließend wird aus dem EEPROM ausgelesen, ob bereits eine Stream-URL gespeichert ist. Wenn dies der Fall ist, fungiert der Microcontroller als HTTP-Client und empfängt den Stream. Anschließend wird dieser mittels I2S-Protokoll an den Digital-Analog-Wandler übertragen. Wenn allerdings noch keine Stream-URL gespeichert ist, wartet der Microcontroller bis eine Stream-URL vom Client gesendet wird.

Konfigurationsmodus
Wird der Konfigurationsmodus mit Druck auf den Taster aktiviert, wechselt die Status-LED ihre Farbe auf blau. Anschließend wird der WiFi-Chip in den AP-Modus gestellt. Somit bietet der Microcontroller einen Access Point, auf den sich andere Geräte als Clients verbinden können. Dies wird benötigt, damit die Clients in weiterer Folge WiFi-Zugangsdaten an den Microcontroller senden können. Um die Daten zu empfangen wird ein WebServer auf dem Microcontroller gestartet. Dieser erhält dann die WiFi-Zugangsdaten mittels HTTP-Protokoll.

Wechsel in normalen Modus
Sobald der Microcontroller die WiFi-Zugangsdaten vom Client erhalten hat, schreibt er diese in den EEPROM. Anschließend wird der Microcontroller neu gestartet und der Programmablauf fängt wieder von vorne an.

4.1.3 Klassen

Bei der Entwicklung der Microcontroller-Software wurde aufgrund der Übersichtlichkeit und um die Design-Patterns der Softwareentwicklung einzuhalten, auf objektorientierte Programmierung gesetzt. Das folgende UML-Klassendiagramm veranschaulicht die Beziehung der verschiedenen Klassen zueinander: hier kommt das UML-Klassendiagramm her Im folgenden Teil werden die Klassen und deren Funktionen noch näher beschrieben:

StatusLED
Mithilfe der Klasse StatusLED wird die RGB-LED, welche am ESP32 angeschlossen ist, gesteuert. Mit ihr wird z.B. angezeigt ob der Controller sich im Konfigurationsmodus befindet oder mit dem WLAN verbunden ist.

NetworkManager
Die Klasse NetworkManager kümmert sich um alle Funktionen, die mit dem WiFi des ESP32 zu tun haben. Dazu gehören: Funktion als Access Point, Funktion als Client.

AudioManager

Die Klasse AudioManager ist für das Empfangen des Internet-Audio-Streams und in weiterer Folge für das Senden der empfangenen Audiodaten an den Digital-Analog-Wandler zuständig.

Logger
Die Klasse Logger ist zuständig, für das Speichern von Logs, welche von anderen Klassen geschrieben werden.

Server Die Klasse Server ist für das managen des Webservers, welcher auf dem Microcontroller läuft, zustängig. In ihr werden unter anderem die verfügbaren Routen der REST-API definiert.

MemoryManager Die Klasse MemoryManager ist für das Schreiben in und das Lesen vom EEPROMs des Microcontrollers zuständig.

4.1.4 Herausforderungen

Bei der Entwicklung der Software für den Adapter, sind wir auf einige Herausforderungen gestoßen. Ursprünglich war geplant, eine Funktion zu implementieren, die es ermöglicht mehrere Adapter zu Gruppen zu verbinden. In diesen Gruppen sollte dann jeweils der gleiche Stream synchron laufen. Die Synchronisierung der Adapter hat sich allerdings als sehr schwierig erwiesen, da die Ressourcen des ESP32 nicht dafür ausreichten, die Latenz gering zu halten.

4.2 Entwicklung Smartphone-App

In diesem Kapitel wird der Übergang der Planung in die Entwicklung der Smartphone-App beschrieben. Die verwendeten Tools wurden schon genauer im Kapitel ... beschrieben.

4.2.1 Struktur

Die App ist in drei Haupt-Screens gegliedert. Im folgenden werden die einzelnen Screens genauer beschrieben.

Adapter
In diesem Screen werden alle Adapter, welche schon hinzugefügt wurden, angezeigt. Es ist hier auch möglich neue Adapter hinzuzufügen. Wenn man auf einen Adapter in der Liste drückt, kann man diesen entweder konfigurieren oder auswählen welchen Audiostream der Adapter erhalten soll.

Musik

In diesem Screen ist es möglich, nach weltweiten Streams von Internet-Radios mithilfe der RadioBrowser-API zu suchen. Die Streams können in weiterer Folge zu einer Favoritenliste hinzugefügt werden, so dass sie für den Benutzer schnell abrufbar sind.

Einstellungen
In diesem Screen kann der Benutzer wichtige Einstellungen für die App treffen. Dabei gibt es folgende Einstellungen: ...

4.2.2 Zielsetzung

Es wurde festgelegt, dass die Smartphoneapp folgende Anforderungen erfüllen soll:

Konfiguration der Adapter
Mit der App soll es einerseits möglich sein, neue Adapter zu konfiguieren, andererseits, bereits hinzugefügte Adapter zu verwalten. Dabei können von den Adaptern verschiedene Daten, wie z.B. Akkustand eingesehen werden.

Verwaltung der Adapter
Mehrere Adapter abspeichern und bestimmen, welcher Adapter welchen Stream erhalten soll.

Suche nach Audio-Streams
Suche nach URLs von Audio-Streams von Internet-Radios mittels der RadioBrowser-API.

4.2.3 UI Design

Bei der Smartphonapp wurde großer Wert auf Benutzerfreundlichkeit gelegt. Dies wurde mit einem sehr übersichtlichen, simplen User Interface erreicht. In der App werden nur folgende drei Farben verwendet: ...

4.2.4 Funktionen

In der Datei Utilities, wurden Funktionen definiert, welche dabei helfen Aktionen, welche für die App notwendig sind durchzuführen. Dazu gehören Abfragen der RadioBrowser-API und Abfragen an den jeweiligen Adapter. Im folgenden werden die Funktionen genauer beschrieben:

getCountries

Die Funktion getCountries fragt alle verfügbaren Ländernamen der RadioBrowser-API ab. Der GET-Request wird dabei mit Axios ausgeführt. Als Rückgabe liefert die Funktion einen Promise, welcher entweder ein String-Array oder null ist. Das String-Array ist gefüllt mit allen verfügbaren Ländernamen. Null wird dann zurückgelifert. wenn bei der Abfrage ein Fehler auftritt.

getLanguages
Die Funktion getLanguages ist gleich aufgebaut wie die Funktion getCountries. Der einzige Unterschied liegt in der URL. Hier werden nähmlich alle verfügbaren Sprachen der RadioBrowser-API abgefragt. Da die Sprachen, welche zurückgeliefert werden, alle kleingeschrieben sind, werden diese anschließend noch umgewandelt, in groß geschriebene Wörter. Dies hat designtechnische Gründe.

getStations
Die Funktion getStations frägt alle Stations, welche die übergebene Sprache bzw. das übergebene Land haben von der RadioBrowser-API ab und gibt diese dann als String-Array zurück. Wenn bei der Abfrage ein Fehler entsteht, wird null zurückgeliefert.

getFavouriteStations
...

addFavouriteStations
...

removeFavouriteStation
...

clearFavouriteStationList
...

Allgemeines
Für die API-Abfragen wurden asynchrone Funktionen verwendet. Asynchrone Funktionen müssen immer dann verwendet werden, wenn eine Aktion länger dauert, bzw. man auf Daten warten muss. Dies ist bei der Abfrage von APIs der Fall. Asynchrone Funktionen werden in Java Script mit äsync function"definiert.

4.2.5 Aufbau

In diesem Kapitel wird die Strukturierung bzw. Aufteilung der App beschrieben. Zur grundlegenden Navigation wurde ein Tab-Navigator verwendet. Es gibt dabei die Tabs Verbindungen, Adapter und Musik. Neben dem Tab-Navigator wurde auch noch ein Stack-Navigator verwendet. Dieser ermöglicht das Navigieren zwischen Seiten mittels einem Stack. Die Navigation mittels Stack funktioniert

nach dem Last In First Out - Prinzip. Das heißt, wenn eine neue Seite aufgerufen wird, wird diese dem Stack hinzugefügt, also sozusagen auf den Stapel oben drauf gelegt. Wenn man wieder auf die vorherige Seite will, wird die aktuelle Seite einfach vom Stack entfernt, also vom Stapel oben herabgenommen.

4.2.6 Komponenten

In diesem Kapitel werden die selbst erstellen React-Komponenten genauer beschrieben.

AdapterItem
Die Komponente AdapterItem wird dazu verwendet, die Daten eines einzelnen Adapters abzufragen und diese anzuzeigen. Dabei muss man dieser Komponente ein Objekt der Klasse Adapter als Argument mitgeben. Beim rendern wird direkt versucht eine Verbindung zum Webserver mit der IP-Adresse des übergebenen Adapters herzustellen. Ist dies erfolgreich, so wird ein GET-Request auf die /getInfo - Route des Adapters mittels Axios durchgeführt. Dabei werden die erhaltenen Informationen grafisch dargestellt. Zu diesen Daten zählt Lautstärke und Akkustand des Adapters. Ist der Adapter allerdings nicht erreichbar, werden die Daten nicht angezeigt. Gleichzeitig ändert sich die Hintergrundfarbe der Komponente zu einem helleren Grauton und es erscheint rechts eine durchgestrichene Wolke. Dies soll signalisieren, dass der Adapter nicht erreichbar ist. Diese Informations-Abfrage wird in einem Intervall von 5 Sekunden ausgeführt, um Änderungen in den Daten des Adapters bzw. der Verbindung des Adapters schnellstmöglich zu signalisieren. Dieses Intervall wird mit der JavaScript-Methode setInterval verwirklicht. Am Ende des renderns, wird das Interval noch geschlossen, um zu vermeiden, dass es mehrere Instanzen davon gibt.

AdapterList
Die Komponente AdapterList stellt eine Liste dar, in der alle bisher hinzugefügten Adaptern aufgelistet sind. Die Adapter werden dabei mithilfe der AdapterItem-Komponente dargestellt. Die Adapter werden mithilfe der Methode getAdapter() aus dem Speicher abgefragt. Wenn noch keine hinzugefügten Adapter vorhanden sind bzw. die Methode getAdapter() null zurückgibt, wird die Komponente ErrorScreen dargestellt.

AddToListButton
Die Komponente AddToListButton wurde verwendet, um ein drückbares Icon darzustellen, welches symbolisieren soll, ein weiteres Element zu einer Liste hinzuzufügen. Für die Umsetzung der Drück-Funktion wurde die Pressable-Komponente von React Native verwendet. Das Icon wurde von der Entypo-Bibliothek importiert.

BatteryIndicator

Die Komponente BatteryIndicator zeigt die Batterieladung in Prozent mit dazugehörigem Icon an. Dabei erwartet sie als Argument die Batterieladung in Prozent. Je nachdem, in welchem Bereich diese Batterialadung liegt, wird ein entsprechendes Icon gerendert. Wenn für die Batterieladung ein negativer Wert (-1) übergeben wird, bedeutet dies, dass der Akku aktuell geladen wird. In diesem Fall, wird ein Lade-Icon dargestellt.

ConnectionItem
...

DeleteButton
Die Komponente DeleteButton stellt ein drückbares Icon dar, welches signalisieren soll, ein Element zu löschen. Dabei wurde die Drück-Funktion mit der Pressable-Komponente realisiert. Das Icon wurde von der FontAwesome-Bibliothek importiert.

ErrorScreen
Die Komponente ErrorScreen stellt eine Ansicht bereit, die signalisieren soll, dass ein Fehler aufgetreten ist. Dabei wird die Fehlermeldung als Text angezeigt und man hat die Möglichkeit auf einen Knopf zu drücken, welcher dann eine Aktion ausführt. Als Parameter werden der zu anzeigende Text, der Text des Knopfs und die Funktion, welche beim druck auf den Knopf ausgeführt wird, übergeben. Der Fehlertext wird durch die Text-Komponente angezeigt und für den Knopf wurde eine Button-Komponnte verwendet.

FavouriteStationList
Die Komponente FavouriteStationList stellt eine Liste aus mehreren StationItem-Komponenten dar. Die Daten dafür, werden beim Rendern der Komponente mithilfe der Funktion getFavouriteStations() aus dem Speicher ausgelesen. Wenn noch keine Favoriten im Speicher sind, wird die ErrorScreen-Komponente gerendert. Dabei ist es möglich, bei längerem Drücken auf ein FavouriteStationItem dieses zu selektieren und in weiterer Folge aus der Liste zu löschen. Die Selektierung wird mit einem kurzen Klick auf ein FavouriteStationItem wieder aufgehoben. Beim Druck auf die AddToListButton-Komponente, wird zum Screen SStationsearch"navigiert. Beim druck auf die DeleteButton-Komponente, wird die ausgewählte Komponente aus dem Speicher gelöscht.

LoadingScreen
Die Komponente LoadingScreen soll signalisieren, dass ein Vorgang durchgeführt wird und deshalb der Screen bzw. die Komponenten noch nicht angezeigt werden können. Dies kann zum Beispiel das Laden von Daten sein. Als Argument wird der Text übergeben, welcher in der Komponente angezeigt wird. Der Ladevorgang wird mit der ActivityIndicator-Komponente siganlisiert.

NetworkItem
Die Komponente NetworkItem stellt Daten von einem Netzwerk dar. Dabei wer-

den als Argumente die SSID und RSSI des Netzwerks übergeben. Die SSID wird mit der Text-Komponente angezeigt. Abhängig von dem Wert der RSSI werden verschiedene Icons angezeigt. Diese Icons stellen die Stärke des Netzwerks dar. Das Argument selected gibt an, ob die Komponente ausgewählt wurde. Wenn dies der Fall ist, verändert sich die Hintergrundfarbe der Komponente.

StationItem
Die Komponente StationItem stellt die Name und Icon einer Radiostation dar. Die Radiostation wird dabei als Argument, als Objekt der Klasse Station übergeben. Das Argument selected gibt an, ob die Station ausgewählt ist. Wenn dies der Fall ist, verändert sich die Hintergrundfarbe der Komponente.

TextInputWindow
Die Komponente TextInputWindow ermöglicht es Text in einem Fenster, welches vor dem anderen Content gerendert wird, einzugeben. Dabei ist unter dem Text ein Knopt "Bestätigenünd ein Knopt Äbbrechen"verfügbar. Der Text, welcher ganz oben angezeigt wird, wird als Argument übergeben. Außerdem wird mit isPassword festgelegt, ob die Eingabe in das Textfeld sichtbar sein soll oder nicht und die übergebenen Funktionen onEnter bzw. onCancel bestimmen, was passiert, wenn man den Äbbrechen oder "Bestätigen"Knopf drückt.

VolumeIndicator
Die Komponente VolumeIndicator zeigt eine Lautstärke in Prozent an, mit einem Lautstärke-Icon davor. Das Lautstärke-Icon wird von der Feather-Bibliothek importiert. Die Lautstärke in Prozent wird mit einer Text-Komponente angezeigt. Als Argument wird die Lautstärke in Prozent übergeben. Wenn diese kleiner als 0 ist, wird ein Icon angezeigt, welches symbolisiert, dass die Lautstärke stumm ist.

4.2.7 Seiten

In diesem Kapitel werden die einzelnen Seiten genauer beschrieben. **Favoriten** Auf dieser Seite wird eine Liste aus Radio-Stationen, welche als Favoriten festgelegt wurden, dargestellt. Dafür wird die Komponente FavouriteList verwendet. **Station-Suche** Auf dieser Seite ist es möglich aus einer Liste von Ländern und Sprachen auszuwählen, welche Radio-Stationen man suchen will. **Favoriten-Auswahl** Auf dieser Seite wird eine Liste, mit allen Stationen, welche dem ausgewählten Land und der ausgewählten Sprache entsprechen, angezeigt. Hier kann man Radio-Stationen markieren und mit klick auf den Haken diese zu seiner Favoritenliste hinzufügen. **Adapter** Auf dieser Seite wird eine Liste von allen bisher hinzugefügten Adaptern angezeigt.

4.3 Design Adaptergehäuse

Das Adaptergehäuse trägt einen wesentlichen Teil zur Sicherheit des Endverbrauchers sowie zur optimalen Funktionalität der Komponenten bei. Zudem soll es möglichst kompakt sein.

4.3.1 Grundsätzlicher Aufbau

Die Grundlage für den Prototyp bildet ein Kasten mit Deckel.
Das Gehäuse wurde mit frei schwebenden, jedoch an den Wänden befestigten Stützen ausgestattet, um den Mikrocontroller fest montieren zu können. Der Prototyp wurde zudem mit kleinen Zylindern auf den Stützen ausgestattet, um den Mikrocontroller mit seinen bereits Vorhandenen Aussparungen darauf platzieren zu können. Der Digital-/Analogwandler und der Akku-Laderegler finden auch auf solchen Stützen ihren Platz. Der Gedanke dahinter war, den Akku unter den Bauteilen zu platzieren. Mehr dazu im Teil "Wärmeableitung". Zudem wurden in einer Wand des Gehäuses Aussparungen für die Buchsen platziert. Die Aussparungen für die RGB-LED und den Taster wurden im Deckel platziert. Eine Art Abdeckung für den Taster selbst wird auf diesen geklebt um ein gleichbleibendes Erscheinungsbild des Gehäuses zu behalten. Der Deckel, der von oben auf das Gehäuse gedrückt wird, schließt dieses. Im Deckel ist zusätzlich ein Belüftungsgitter eingelassen. Mehr dazu auch im Teil "Wärmeableitung".

4.3.2 Virtuelles 3D-Design

Um ein 3D-Modell des Prototypen zu zeichnen, wurde AUTODESK Fusion verwendet. Der Anbieter beschreibt seine Software folgendermaßen: Äutodesk Fusion verbindet Ihren gesamten Fertigungsprozess durch die Integration von CAD, CAM, CAE und PCB in einer einzigen Lösung, mit der Sie Ihre Ideen verwirklichen und praktisch alles fertigen können."
Wenn man schon früh beachtet, dass die Prototypen mit einem 3D-Drucker gefertigt werden, kann man schon das 3D-Design für eine gute Druckbarkeit auf 3D-Drucker anpassen. Damit ist hauptsächlich gemeint, überhängende Drucke, komplizierte Stützstrukturen oder ähnliches zu vermeiden.

Basis
Die Basis des Adapters bildet ein 71x58x27mm großer Kasten mit 2mm Wanddicke. Aus Erfahrung kann man sagen, dass diese Dicke bei 3D-Drucken stabil ist, während sie jedoch nicht zu klobig wirkt.
Die Stützen des Mikrocontroller beginnen auf 11mm Höhe und sind auf einer Längenseite 6x6x5mm und auf der anderen 6,9x6x5mm groß. Sie sind jeweils mit einer oder zwei Seiten an der Wand der Basis und somit überhängend. Mehr dazu im Teil "Drucken des Prototyps/Stützstruktur".

Auf den Stützen befinden sich jeweils Zylinder, die genau auf die Aussparungen des Mikrocontroller vermessen wurden. Die Zylinder haben einen Durchmesser von exakt 3mm. Der Abstand der Mittelpunkte dieser Zylinder war bei unserem Modell 47,10mm in der Länge und 23,10mm in der Breite. Auf einer Seite befindet sich zwischen Wand und Zylinder etwas mehr Platz, da der USB-C Port des Mikrocontrollers etwas über diesen herausragt. Deswegen auch der zuletzt erwähnte Versatz der Stütze von 0,9mm.
Die Breite der Basis ist also genau auf die Länge von dem von uns benutzten Mikrocontroller zugeschnitten.
In den gegenüberliegenden Ecken der Basis befinden sich der Digital-/Analogwandler und der Laderegler. Die Maße des Digital-/Analogwandler sind 31,8x17,2mm. Die Maße des Laderegler sind 28x17,45mm. Beide liegen, wie der Mikrocontroller auch, auf überhängenden, 5mm hohen Stützen auf. Aufgrund der USB-C Buchse wird der Halt des Laderegler noch von einer 3,5mm breiten 2mm-Erhöhung am Ende der Stütze verstärkt. Die USB-C Buchse wurde passend zum Laderegler und der Norm entsprechend (8,34x2,56mm Größe) eingelassen. Der Digital-/Analogwandler wird aufgrund der 3,5mm Klinkenbuchse von einer herabstehenden Wand gestützt, die im Teil "Deckel"genauer beschrieben wird. Die Aussparung der AUX-Buchse wurde auch passend für den Digital-/Analogwandler platziert und hat einen Durchmesser von 6mm.

BILD: BASIS DRAUFSICHT FUSION
BILD: BASIS MIT INHALT

Deckel
Der Deckel des Adapters ist grundsätzlich, wie die Wände der Basis, 71x58x2mm groß. Dieser hat jedoch eine zentrierte 67x54x1mm große Stufe. Mit dieser Stufe lässt sich der Deckel kleberlos auf den Adapter setzen und hält aufgrund der Eigenschaften des 3D-Drucks auch, zumindest für den Prototypen, fest genug. Wie schon erwähnt, wird der DAW durch eine herabstehende Wand zusätzlich gestützt. Die Maße dieser Wand sind 32x2x10mm. Es würde keinen Sinn machen, die Wand wie beim Laderegler von der Außenwand aus überhängend zu machen, da der DAW länger als der Laderegler ist und die Buchse sich eher mittig in der entsprechenden Außenwand der Basis befindet.
An der freien Seite der Stützwand befindet sich ein runder Schacht für die RGB-LED mit 5mm Durchmesser und eine Aussparung für den Aufsatz des Tasters mit 10,10mm Durchmesser. Der Taster mit den Grundmaßen 6x6mm wird durch eine Art U-Form aus Wänden im Gehäuse gehalten. Eine Wand davon bildet die gerade eben beschriebene Stützwand. Die untere Wand, auf der der Taster aufliegt, hat zudem zwei auf den Taster angepasste, 6mm große Aussparungen für die zwei Pole des Tasters.

Tasteraufsatz
Den Tasteraufsatz bildet ein Zylinder mit 9,5mm Durchmesser, auf dem sich ein weiterer Zylinder mit 5,5mm Durchmesser und zentrierter 3,5mm Aussparung für den Taster befindet.

vgl. https://www.autodesk.com/de/products/fusion-360/overview
vgl. https://www.elektronik-kompendium.de/sites/com/2009021.htm

4.3.3 Wärmeableitung

Wärmeableitung ist wichtig, da Mikroprozessoren, wie alle anderen Prozessoren auch, bei intensivem Betrieb Hitze entwickeln. Laut einer eigens durchgeführten Messung wird der in diesem Fall eingesetzte ESP32 meistens nur rund 38°C warm. Bei hoher Rechenleistung sind jedoch auch höhere Temperaturen möglich. Der ESP32 hat laut Hersteller eine mögliche Betriebstemperatur von –40°C bis +125°C. Damit die entstandene Wärme gut ableiten kann und keine der Komponenten beeinflusst (vor allem den Akku, da dieser bei hohen Temperaturen Explosionsgefahr ausgesetzt ist), wird der Mikrocontroller im Gehäuse oben, also auf Stützen angebracht. Die aufsteigende Wärme kann somit nach oben durch das dafür vorgesehene Gitter entweichen und staut sich somit nicht im Inneren des Gehäuses. Der Akku liegt dementsprechend unter dem Mikrocontroller und allen anderen Komponenten und wird durch die abstrahlende Hitze dementsprechend nur etwas wärmer als Raumtemperatur.

vgl. https://www.espressif.com/en/products/socs/esp32

4.4 Fertigung Adaptergehäuse

Das Material unseres Gehäuses wurde auf Kunststoff begrenzt. Für die Fertigung von Kunststoffgehäusen gibt es hauptsächlich diese Möglichkeiten, welche für uns in Frage kommen:

Spritzgießen
„Beim Spritzgießen wird der Kunststoff aus einem Plastifiziergerät (erwärmt den Kunststoff auf Schmelztemperatur) in einen Hohlraum (Formwerkzeug) gespritzt, in welchem er erst verdichtet wird und dann erkaltet."
Ein Vorteil für dieses Verfahren ist, dass auch komplizierte Formteile voll automatisiert sehr schnell in hohen Stückzahlen produziert werden können. Der große Nachteil sind jedoch die hohen Stückkosten für die Formwerkzeuge.

vgl. https://www.chemie.de/lexikon/Kunststoffverarbeitung.html

3D-Druck
Die zwei gängigsten 3D-Druck-Methoden sind Filament und Resin Harz. Aufgrund des hohen Aufwands, den ein Resin-Drucker mit sich bringt, wurde für dieses Projekt die Methode mit Filament gewählt.
Beim 3D-Drucken durch Fused Deposition Modeling (Schichtschmelzverfahren) wird Kunststoff in Drahtform (Filament) (die häufigsten Dicken sind 2,85mm

(allgemein als 3mm bezeichnet) und 1,75mm wobei die 1,75mm Version weltweit am häufigsten verbreitet ist) durch beheizte Düsen geleitet und somit geschmolzen. Das nun weiche Filament wird in Schichten auf die Druckplatte aufgetragen und erhärtet kurz darauf. Durch dieses Schichten lassen sich präzise Körper aus Kunststoff bauen.

vgl. `https://www.printer-care.de/de/drucker-ratgeber/wie-funktioniert-ein-3d-drucker`
vgl. `https://help.prusa3d.com/de/glossary/175-mm_134816`

4.4.1 Drucken des Prototyps

Die Gehäuse-Prototypen wurden mit einem „PRUSA MK4S“ 3D-Drucker gefertigt. Alle FFF-Drucker von Prusa sind für 1,75-Filament konfiguriert.

Druckeinstellungen
Die Temperatur der Build Plate lag bei uns Standartmäßig auf 60°C. Die Drucktemperatur, also die der Nozzle (Düse) wurde beim ersten Layer auf 200°C eingestellt (praktisch 190°C), die finale Temperatur lag bei 185°C.
Das erste Layer wurde 0,15mm Dick gedruckt, die restlichen 0,2mm dick.
Als Infill-Pattern wurde „Grid“ mit 20% Dichte und 4mm Line-Distance gewählt.
Das Drucken verlief mit den von uns gewählten Einstellungen reibungslos, jedoch an manchen Stellen etwas unsauber. Beispielsweise war das Ergebnis der Aussparung für den Taster im Deckel des Adapters so ungenau, dass der Durchmesser des Aufsatzes für den Taster um 0,5mm verkleinert werden musste. Sonstige Ungenauheiten stellten, zumindest abgesehen von der Optik, kein Problem dar.

vgl. `https://help.prusa3d.com/de/glossary/175-mm_134816`

4.5 Zusammensetzen des Prototypen

Die Komponenten werden jeweils einzeln direkt mit dem Mikrocontroller verbunden. Bei diesem Prototyp lässt sich der Mikrocontroller nämlich vorerst als Platine betrachten, welcher zusätzliche Komponenten zugefügt werden.

4.5.1 Schaltplan

Die Pinbelegung zwischen den Komponenten und dem Mikrocontroller ist folgendermaßen gelöst:

Digital-/Analogwandler
BCK (Wandler) an D26 (Mikrocontroller)
LCK (Wandler) an D25 (Mikrocontroller)
DIN (Wandler) an D22 (Mikrocontroller)

Taster
1 (Taster) an 3V3 (Mikrocontroller)
2 (Taster) an D12 (Mikrocontroller)

RGB-LED
R (LED) an D15 (Mikrocontroller)
G (LED) an D2 (Mikrocontroller)
B (LED) an D4 (Mikrocontroller)
GND (LED) an GND (Mikrocontroller)

4.5.2 Verdrahten

Bei unserem Prototyp wurden alle Komponenten, der Pinbelegung entsprechend, mit dem Mikrocontroller verbunden. Dabei kamen Drahtkabel, Litzenkabel und Steckkabel (Litze) zur Verwendung. Dies hatte hauptsächlich den Grund, dass zum Beispiel nur mit Drähten nicht alle Verbindungen optimal möglich gewesen wären, was hauptsächlich der Löthaftung an einigen Komponenten zu verschulden war. Um keine Wackelkontakte, oder gar unterbrochene Verbindungen zu riskieren, wurde für jede Verbindung einzeln entschieden, welches der genannten Verfahren sich am besten eignet. So entstanden zum Beispiel Steckverbindungen, gelötete Drahtverbindungen oder Mischungen aus gelöteten Litze- und Steckverbindungen (jeweils am anderen Ende des Kabels).

4.5.3 Kleben

Damit alle Komponenten und Teile sicher im Gehäuse sitzen und nichts klappert oder sich gar bewegt, müssen gewisse Komponenten angeklebt werden. Vorallem bei den Buchsen ist eine feste Verbindung wichtig, da diese bei jedem Ein- und Ausstecken großer Belastung ausgesetzt sind. Somit wurden der Laderegler und der Digital-/Analogwandler an den dafür gedruckten Stützen angeklebt. Zudem wurde der gedruckte Aufsatz in Gehäuseoptik für den Taster auf diesem befestigt. Für alle Klebverbindungen im Adapter wurden entweder herkömmliches Cyanacrylat (Superkleber) oder Schmelzklebstoff (Heißkleber) verwendet.

5 Testen und Fehlerbehebung

5.1 Testen des Gesamtsystems

5.2 auftretende Fehler beheben

5.3 Test auf Cybersecurity

In diesem Kapitel wird der gesamte Code auf Sicherheitslücken getestet.

5.4 Auftretende Sicherheitslücken schließen

In diesem Kapitel werden die bei den Tests aufgetretenen Sicherheitslücken geschlossen.

6 Einzelnachweise

6.1 Literaturverzeichnis

6.2 Abbildungsverzeichnis

6.3 Anhang