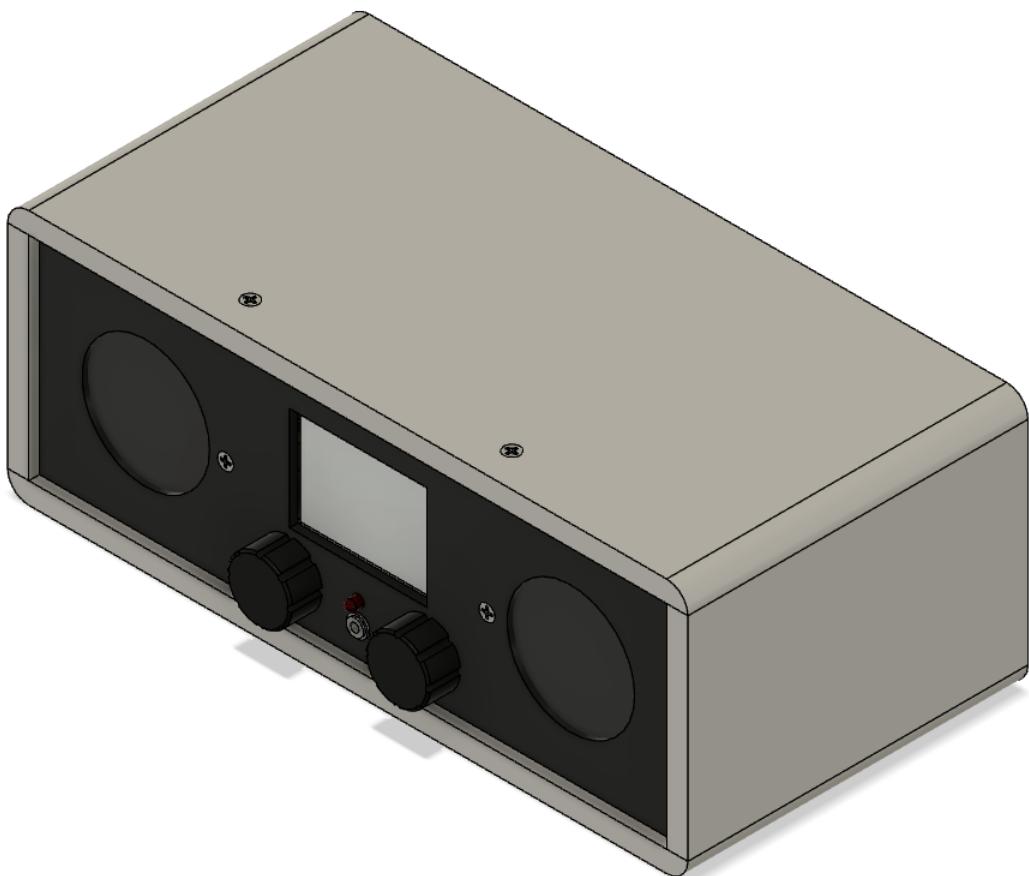


Gruppe E

# Internetradio

Diplomarbeit



Eingereicht

Kranz Gergő  
Leitinger Christoph  
Nestl Elias

im Schuljahr  
der Klasse  
bei  
am

2020/2021  
5BHEL  
DI. Karl Friedl  
21. April 2021



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung .....</b>	<b>1</b>
1.1	Kurzbeschreibung.....	1
1.2	Abstract .....	2
1.3	Ismertető.....	3
<b>2</b>	<b>Lastenheft .....</b>	<b>4</b>
2.1	Projektziel.....	4
2.2	Was kann ein Nutzer mit dem Radio tun? .....	4
2.3	Kunden, Anwender, Zielmarkt .....	4
2.4	Anforderungen .....	4
2.5	Randbedingungen .....	5
2.6	Chancen und Risiken .....	5
2.7	Aufwandsabschätzung (vor Projektstart).....	6
<b>3</b>	<b>Funktionsübersicht .....</b>	<b>7</b>
3.1	Audioquellen .....	7
3.1.1	Netzwerk.....	7
3.1.2	Bluetooth .....	7
3.1.3	USB .....	7
3.1.4	FM-Radio.....	7
3.1.5	Analoger Eingang .....	7
3.1.6	Warum kein DAB+? .....	8
3.2	Hardwareüberblick.....	9
3.2.1	Haupt- und Coprozessor .....	9
3.2.2	Audiopfad (Digital).....	9
3.2.3	Audiopfad (Analog) .....	10
3.2.4	Steuer- und Kommunikationssignale.....	10
<b>4</b>	<b>Gehäuse .....</b>	<b>11</b>
4.1	Vorstellung .....	11
4.2	Das endgültige Gehäuse.....	12
<b>5</b>	<b>Schaltung und Layout .....</b>	<b>20</b>
5.1	Komponenten.....	20
5.1.1	Verstärker .....	20



5.1.2	Audiodämpfungsglied.....	20
5.1.3	Multiplexer.....	21
5.1.4	Raspberry Pi Zero W .....	21
5.1.5	Display.....	21
5.1.6	Infrarotempfänger .....	21
5.1.7	Coprozessor .....	21
5.1.8	Spannungsversorgung.....	22
5.1.9	Zusätzliche Module und Anschlüsse .....	22
5.2	Planung und Fertigung .....	22
5.3	Probleme .....	22
<b>6</b>	<b>Stücklisten.....</b>	<b>28</b>
6.1	Elektronik .....	28
6.2	Gehäuse.....	29
<b>7</b>	<b>Firmware.....</b>	<b>30</b>
7.1	Kommunikation zwischen Co- und Hauptprozessor .....	30
7.1.1	Software am Coprozessor .....	30
7.2	Infrarot .....	33
<b>8</b>	<b>Software.....</b>	<b>34</b>
8.1	Raspberry Pi.....	34
8.2	Klassen.....	34
8.2.1	Klasse - Wifi.....	35
8.2.2	Klasse - Bluetooth .....	37
8.2.3	Klasse - StreamPlayer.....	40
8.2.4	Klasse - FileService .....	44
8.2.5	Klasse - Coprocessor .....	46
8.2.6	Klasse - HomeBar .....	49
8.2.7	Klasse - NotificationBar.....	50
8.2.8	Klasse - Window.....	52
8.2.9	Klasse - Player .....	53
8.2.10	Klasse - Settings .....	56
8.2.11	Klasse - Settings::SubSettings .....	57
8.2.12	Klasse - Settings::WifiSettings : SubSettings .....	59
8.2.13	Klasse - Settings::WifiSettings::Popup .....	61



8.2.14 Klasse - Settings::BluetoothSettings : SubSettings .....	62
8.2.15 Klasse - Settings::InputSettings : SubSettings.....	64
8.2.16 Klasse - Settings::DisplaySettings : SubSettings.....	65
8.2.17 Klasse - Settings::DisplaySettings::Popup .....	66
8.2.18 Klasse - Settings::SystemSettings : SubSettings.....	67
8.2.19 Klasse - Sources.....	68
8.2.20 Klasse - Sources::SubSources .....	70
8.2.21 Klasse - Sources::RadioSource : SubSources.....	71
8.2.22 Klasse - Sources::StreamSource : SubSources .....	72
8.2.23 Klasse - Sources::LocalSources : SubSources .....	73
8.3 D-Bus & XMLHttpRequest .....	74
8.4 Raspberry Pi einrichten .....	75
8.4.1 Installer Script .....	75
8.5 Kompilieren .....	76
8.6 Ausführen .....	76
8.7 ESP32 Fehlschläge .....	77
8.7.1 Probleme.....	77
8.7.2 Lösung .....	77
<b>9 Testbetrieb mit Prototyp am Steckbrett .....</b>	<b>78</b>
9.1 Steckbrettprototyp.....	78
9.1.1 Raspberry .....	78
9.1.2 ATmega328P .....	78
<b>10 Testbetrieb mit gefertigter Platine .....</b>	<b>79</b>
10.1 Brandschutz.....	79
10.2 Zusammenbau.....	79
10.3 Tests .....	79
10.3.1 Versorgung, Display, DAC und Raspberry Pi .....	79
10.3.2 Eingangsmultiplexer.....	80
10.3.3 Attenuator und Ausgangsmultiplexer.....	80
10.3.4 ATmega328P .....	80
10.3.5 Power on / off Transistor .....	80
10.3.6 Verstärker .....	81
<b>11 Bedienungsanleitung .....</b>	<b>81</b>



11.1	Stromversorgung.....	81
11.2	Aus- und Einschalten .....	81
11.3	Navigation im User Interface .....	81
11.4	Lautstärke ändern. ....	81
11.5	Musik vom Audio-Eingang abspielen. ....	81
11.6	Musik am Audio-Ausgang ausgeben. ....	81
11.7	Musik von einem USB-Speicher .....	82
<b>12</b>	<b>Quellenverzeichnis .....</b>	<b>82</b>
12.1	Hardware.....	82
12.2	Software .....	83
12.3	Sonstige .....	83
<b>13</b>	<b>Abbildungsverzeichnis .....</b>	<b>84</b>



# 1 Einleitung

## 1.1 Kurzbeschreibung

In dieser Diplomarbeit ist die Entwicklung und der Bau eines Radios beschrieben, das Musik aus möglichst vielen Quellen mit guter Qualität abspielen kann.

Es soll möglich sein, Musik über Bluetooth, Internet, FM und aus externen, analogen Audioquellen zu hören. Auch USB-Speicher können direkt an das Radio angeschlossen werden. Es soll nicht nur möglich sein, Musik über die internen Lautsprecher, sondern auch über Kopfhörer oder externe Lautsprecher abzuspielen. Die internen Lautsprecher sollen ausreichend Leistung haben, um einen ganzen Raum gut beschallen zu können. Die Steuerung des Radios soll entweder direkt durch einen eingebauten Touchscreen und Drehregler oder indirekt durch eine Fernbedienung erfolgen.

Jemand mit Programmierkenntnissen wird auch die Möglichkeit haben das Radio selbst zu erweitern und anzupassen, denn der modulare Hardwareaufbau und die Open Source Software bieten die ideale Plattform für eigene Ergänzungen.



## 1.2 Abstract

This thesis describes the development and construction of a radio is constructed to be able to play music with excellent quality from a wide variety of sources.

Listening to audio via Bluetooth, Internet, FM and external analogue sources should be supported. Audio can also be played from a USB device connected to the radio. Not only should it be possible to use the internal speakers, but also headphones or external speakers. The built-in speakers should have enough output power to fill a whole room with music at a decent volume. Either a touchscreen and rotary knobs or an infrared remote should be able to control the device.

Anyone with sufficient programming skills will have the ability to extend and customise the feature-set of the radio thanks to the modular hardware and the open-source software. These provide an ideal platform for personal modifications.



### 1.3 Ismertető

Ebben a diplomamunkában egy rádió fejlesztése és építése van leírva, ami a lehető legtöbb forrásból képes jó minőségű zenét lejátszani.

Annak is lehetségesnek kell lennie, hogy zenét Bluetooth-on, interneten FM-en és külső, analóg forrásból is lehessen hallgatni. Pendrive-ot is lehet a készülékkel használni. Zenét nemcsak a beépített hangszórókon keresztül lehet hallgatni, hanem fejhallgatón vagy külső hangszórón keresztül is. A beépített hangszóróknak annyi teljesítménnyel kell rendelkezniük, hogy egy egész szobát képesek legyenek jó minőségű hanggal megtölteni. A rádiót lehet közvetlenül a beépített érintőképernyőn keresztül és a rajta levő gombokkal, vagy pedig közvetve egy távirányítóval kezelni.

Aki programozási tudással rendelkezik, képes lesz a rádiót egyedül fejleszteni és testreszabni. A moduláris felépítés és a nyílt forráskód ideális felületet biztosítanak az egyedi fejlesztéseknek.



## 2 Lastenheft

### 2.1 Projektziel

Zum Abgabetermin ist ein Stereo-Radio im Holzgehäuse fertiggestellt, das gängige Empfangsmöglichkeiten wie z.B. UKW-Band bietet, jedoch auch Musik aus dem Internet oder dem lokalen Netzwerk abspielen kann (z.B. Podcasts und DLNA).

### 2.2 Was kann ein Nutzer mit dem Radio tun?

- FM-Radio hören
- Die weitaus größere Musikvielfalt gegenüber herkömmlichen Radios genießen
  - Radiosender im Internet
  - Streams
  - Musik von einem USB-Speichermedium
  - Netzwerkspeicher im lokalen Netzwerk
  - Musik vom Handy über Bluetooth
  - Über einen 3,5mm Audioeingang
- Ein technisch versierter Benutzer kann die Open Source-Software nach seinen eigenen Wünschen anpassen.

### 2.3 Kunden, Anwender, Zielmarkt

- Als potenzielle Kunden kommen alle diejenigen in Frage,
  - denen Musik- & Radioprogrammvielfalt gute Soundqualität und Funktionalität wichtig sind.
  - die gerne in den eigenen vier Wänden Radio hören.
  - die unter Umständen wissen wollen, wie das Radio im Inneren funktioniert.
- Da der verwendete Herstellungsprozess (händische Manufaktur) nicht freundlich für die Serienproduktion ist, lässt sich Serienfertigung und Verkauf ausschließen
  - dank der relativ wenigen elektronischen Komponenten, könnte das Radio aber als Bausatz für Elektronikbastler verkauft werden.

### 2.4 Anforderungen

- Das Gerät sollte Musik abspielen können, verschiedene Radiosender empfangen, einen klaren Klang haben und leicht zu bedienen sein.
- Es soll ein zeitloses Aussehen haben, das in jeden Haushalt passt.
- Zwischen Drücken des Einschalters und dem Hören des ersten Klangs soll nicht viel Zeit vergehen.
- Eine Fernbedienungsmöglichkeit, um das Radio von Überall im Raum bedienen zu können wäre nützlich, aber nicht unbedingt notwendig.



## 2.5 Randbedingungen

- Das Radio soll Audio aus dem Internet, von verschiedenen FM-Radiosendern und über Bluetooth empfangen können.
- Als Schnittstelle zum Benutzer sind ein Touchscreen, zwei Lautsprecher, sowie einige „herkömmliche“ Bedienelemente (z.B. der Lautstärkeregler) vorhanden.
- Audioverstärker, Touchdisplay, Lautsprecher und Tuner werden als fertiges Modul bzw. als IC gekauft. Als „Hirn“ soll ein Raspberry Pi Zero W eingesetzt werden. Die Leiterplatte wird selbst designet, aber mit der Fertigung wird eine externe Firma beauftragt.
- Das Radio wird von einem externen Netzteil mit Strom versorgt. Über 3,5 mm Klinkenstecker sind Audio-Ein- und Ausgänge gelöst.
- Das Gehäuse sollte möglichst kompakt sein, aber dennoch ausreichend groß, um gut zu klingen.
- Die Materialkosten sollen nicht mehr als € 150 betragen. Sie werden möglichst niedrig gehalten, solange dies die Qualität nicht beeinträchtigt. Der Preis, exkl. Lieferkosten, des endgültigen Produkts wird aus den tatsächlichen Herstellungskosten in Verbindung mit den Kosten für Research & Development berechnet.

## 2.6 Chancen und Risiken

- Unsere Zielgruppe sind alle, die sich ein eigenständiges Gerät zum Hören von Musik wünschen, das aber modernen Standards entspricht. Das sind Menschen aller Altersgruppen, von Jugendlichen, die bei einer Party weiterhin ihre Handysucht ausleben wollen, bis zu Senioren, die ein standalone Radio gewohnt sind.

Täglich hören 4,9 Millionen Personen ab 10 Jahren zumindest ein ORF-Radio<sup>1</sup>. Da unser Radio auch Wiedergabe aus dem Internet unterstützt, ist die Zielgruppe umso größer.

Man mag es nicht glauben, aber die Anzahl der Radiohörer in Österreich, und somit auch die Größe unsere Zielgruppe ist wieder im Anstieg.<sup>2</sup>

- Ein großes Risiko ist das Planen und Bauen der eingebauten Lautsprecher. Die Grenze zwischen einem klaren Klang mit angenehmem Bass und zerfetzten Tönen ist sehr dünn.
- Auch durch Interferenzen verursachte Störungen im Audiosignal könnten zum Problem werden.

---

<sup>1</sup> [https://www.radioszene.de/135220/radiotest-2019\\_2.html](https://www.radioszene.de/135220/radiotest-2019_2.html)

<sup>2</sup> <https://de.statista.com/statistik/daten/studie/315028/umfrage/taegliche-radionutzung-in-oesterreich/>



## 2.7 Aufwandsabschätzung (vor Projektstart)

Bezeichnung	Preis
ESP32	2,84€
Antenne	0,83€
I <sup>2</sup> S DAC	2,50€
Drehimpulsgeber (Rotary Encoder)	1,00€
FM Tuner	2,00€
Touch Display	20,00€
Verstärker	2,00€
Stereo Lautsprecher	17,00€
Leiterplatte	20,00€
Kleinpauschale (unsicherer Wert)	50,00€
Gehäuse	20,00€

- Es werden pro Mitarbeiter ca. 30 Mann-Tage benötigt.  
Es werden Software-, Hardware- und Gehäusespezialisten benötigt.



### 3 Funktionsübersicht

Im Rahmen dieser Arbeit wurde eine Hard- und Softwareplattform aufgebaut mit deren Komponenten ein funktionsreiches Internetradio aufgebaut werden kann.

#### 3.1 Audioquellen

##### 3.1.1 Netzwerk

Wie der Name Internetradio vermuten lässt, ist es möglich, dieses Radio mit einem Netzwerk, und damit auch mit dem Internet, zu verbinden. Dies geschieht primär über W-LAN, theoretisch könnte aber auch ein USB Netzwerkadapter mit der rückseitigen USB-Buchse verbunden werden.

Eine Softwarekomponente wurde entwickelt, die es ermöglicht, fast jeden möglichen HTTP Audiostream abzuspielen. Damit ist grundsätzlich das vollständige, öffentliche Webradioangebot abgedeckt, wenn man eine URL zu dem gewünschten Sender hat.

Mithilfe dieser Softwarekomponente könnten auch einfach andere Netzwerkmusikdienste implementiert werden, beispielsweise DLNA oder Integration mit einem lokalen Medienserver wie Jellyfin. Auch Podcasts könnten integriert werden.

##### 3.1.2 Bluetooth

Da im Smartphonezeitalter nun fast jeder ein äußerst fähiges „Internetradio“ in der Hosentasche hat, wäre es schade, dieses nicht auszunutzen, um dem stationären Radio mehr Möglichkeiten zu bieten. Ist ein Gerät mit A2DP verbunden, so wird jedes Geräusch über Bluetooth an das Radio geschickt. Dies ermöglicht es auch Dienste, die sonst aus Lizenzgründen nicht direkt über das Radio verwendet werden könnten (wie beispielsweise kommerzielle Musik-Streamingdienste), indirekt zu nutzen.

Zusätzliche Sicherheit gegenüber klassischen Bluetooth-Lautsprechern wird dadurch gewährleistet, dass das zu verbindende Gerät bewusst am Radio ausgewählt werden muss, anstatt dass, wie üblich, der Lautsprecher einen Pairing-Modus betritt, in dem jedes Gerät sich ohne Bestätigung verbinden kann.

##### 3.1.3 USB

Eine lokale Musiksammlung von einem USB-Stick, einer externen USB Festplatte oder einer SD-Karte (mit USB Adapter) kann ebenfalls abgespielt werden, wenn das jeweilige Speichermedium am USB-Anschluss auf der Rückseite angeschlossen wird.

##### 3.1.4 FM-Radio

Für den klassischen Radioempfang befindet sich im Internetradio ein FM-Tuner.

##### 3.1.5 Analoger Eingang

Über einen 3,5 mm Klinkenstecker ist es möglich, einfach jede analoge Audioquelle an das Radio anzuschließen, um es wie einen Aktivlautsprecher zu nutzen. Beispielsweise kann ein Smartphone verbunden werden, wenn eine verkabelte Verbindung bevorzugt wird.



### 3.1.6 Warum kein DAB+?

Es stellt sich womöglich die Frage, warum bei einem modernen Radio der digitale Radiostandard DAB+ fehlt. Der Grund dafür ist, dass die verwendeten Technologien teilweise noch immer unter Patentschutz stehen. Daher halten Hersteller von Empfangsbausteinen für DAB+ ihre Firmware und/oder Datenblätter unter Verschluss. Für jemanden, der keine Firma besitzt ist es nicht oder nur sehr schwer möglich an diese Daten zu kommen, und selbst wenn, unterliegen diese einem Geheimhaltungsvertrag.



### 3.2 Hardwareüberblick

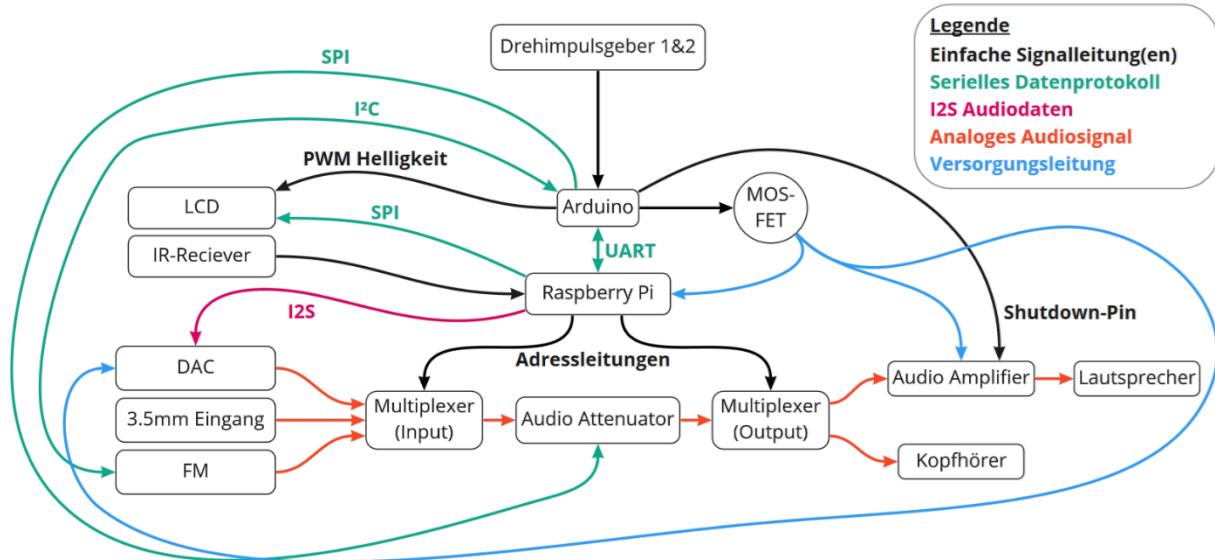


Abbildung 1: Hardwareüberblick

#### 3.2.1 Haupt- und Coprozessor

Das System verfügt über einen Raspberry Pi Zero W mit Linux als Hauptprozessor und einen ATmega328P mit Arduino Code als Coprozessor.

Der Raspberry bietet den Kern der Internetradiofunktionalität (W-LAN, Bluetooth, USB) während der ATmega einige Peripheriebausteine und einen MOSFET-Schalter ansteuert.

Der Grund für diesen Aufbau ist die Reduktion der Stromaufnahme im Standby und Entlastung des Raspberries.

Nach dem Aktivieren des Standby-Modus bleibt der Coprozessor weiterhin aktiv und wartet, bis einer der Drehimpulsgeber hineingedrückt wird. Ist das der Fall, wird der MOSFET-Schalter durchgeschaltet und dadurch der Pi, der Verstärker und der DAC mit Strom versorgt. Der Pi bootet von selbst, sobald eine Versorgungsspannung anliegt.

Wieder aktiviert wird der Standby-Modus auf der Benutzeroberfläche. Dabei wird der Pi heruntergefahren und anschließend der MOSFET-Schalter abgeschaltet.

Eine genauere Beschreibung aller Komponenten befindet sich in der [Sektion Komponenten](#).

#### 3.2.2 Audiopfad (Digital)

Alle digitalen Audioeingangsmöglichkeiten des Radios, also Netzwerk, Bluetooth und USB, werden vom Raspberry Pi abgedeckt und über das Linux-System mit Pulseaudio gesteuert. (für Details siehe die [Sektion Software](#)).

Vom Pi gehen diese dann über seine I2S Schnittstelle an den PCM5102 DAC und werden von ihm in ein analoges, Stereo-Audiosignal konvertiert.



### 3.2.3 Audiopfad (Analog)

Da das Radio neben den genannten digitalen Quellen auch FM und einen analogen Eingang bietet, muss zwischen diesen und dem DAC-Ausgang umgeschaltet werden können. Dazu dient der Eingangsmultiplexer (ein Ausgang, viele Eingänge; durch Adressleitungen kann zwischen ihnen gewechselt werden). Sein Ausgang geht durch den Attenuator LM1972, der es ermöglicht die Lautstärke des Audiosignals zu verringern, und dann an den Ausgangsmultiplexer. Er dient dazu, zwischen Lautsprecher- und Kopfhörerausgang umzuschalten und ist daher andersherum (ein Eingang, viele Ausgänge) eingebaut als der Eingangsmultiplexer.

Der Kopfhörerausgang des zweiten Multiplexers ist direkt an eine 3,5mm Klinke am Gehäuse verbunden. Der Lautsprecherausgang wird vom TPA3144D2 Verstärker verstärkt und geht dann an die beiden eingebauten Lautsprecher.

### 3.2.4 Steuer- und Kommunikationssignale

Die Schaltung ist so konzeptualisiert, dass der ATmega328P alle für einen Standby-Betrieb notwendigen Aufgaben erledigen kann.

Dazu kann er den Shutdown-Pin des Verstärkers kontrollieren, und auch die Stromversorgung des Pis und des Verstärkers über einen MOSFET ein- oder ausschalten. Auch der Pin zur Steuerung der Helligkeit des Displays ist mit ihm verbunden.

Da der Audio-Attenuator LM1972 ein SPI-kompatibles Interface bietet ([Quelle: 12.1.2 Datenblatt](#)), ist er direkt an das SPI-Interface des ATmega328P angeschlossen. Um ohne Umwege über den Pi direkt von der Drehbewegung des Drehimpulsgebers zu einer Lautstärkeänderung zu kommen und um den Pi nicht mit vielen Interrupts zu belasten, sind die Drehimpulsgeber ebenfalls mit dem ATmega verbunden.

Das FM-Modul Si4703 ist auch mit dem 328P verkabelt, da es für die Arduino Plattform geeignete Libraries für dieses Modul gibt ([Quelle: 12.2.14 Library](#)) und die Verwendung dieser Zeit spart.

Vom Raspberry direkt kontrolliert werden die beiden Audio Multiplexer TMUX1309. Auch der Infrarot Empfänger und das Display (abgesehen vom oben erwähnten Pin für die Helligkeit) sind direkt mit dem Pi verbunden.



## 4 Gehäuse

### 4.1 Vorstellung

Die erste Vorstellung war das Gehäuse aus Beton mit dem Gießverfahren zu fertigen. Es wurden entsprechende Gießformen entworfen und aus Sperrholz gefertigt. Es wurde Acetatfolie verwendet, um eine möglichst glatte Oberfläche zu erstellen. Leider war diese Idee nicht durchführbar.

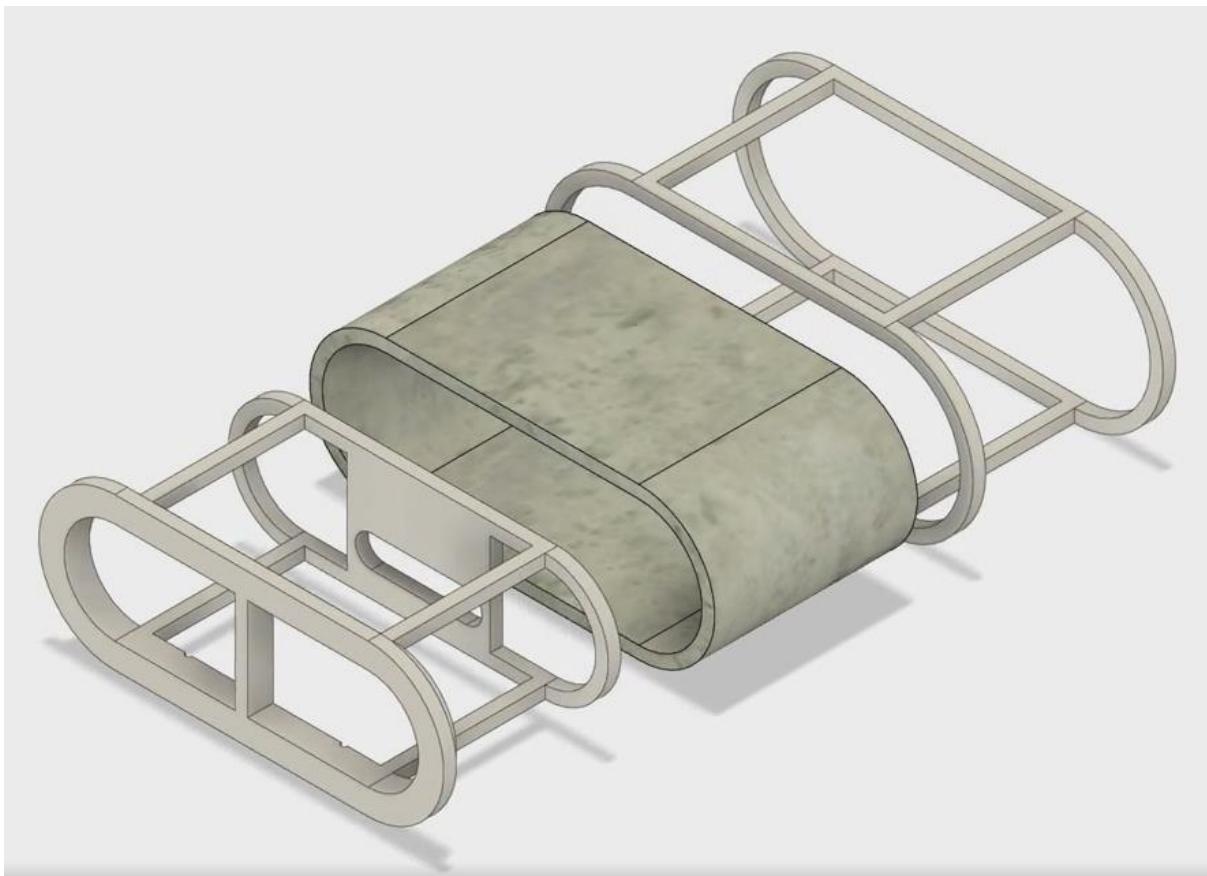


Abbildung 2: Gießform für Beton

Trotz mehreren Versuchen ist es nicht gelungen ein Gehäuse aus Beton zu gießen. Es wurde feinkörniger Beton benutzt, um eine homogene und schöne Form zu erreichen. Es stellte sich aber leider heraus, dass dieser Beton nicht die nötige Stabilität für diesen Verwendungszweck hatte. Er war brüchig und ist immer bei der Entnahme aus der Gießform zerfallen.

Da keine weiteren Ressourcen, Zeit und Geld, in eine nicht funktionierende Idee investiert werden sollten, wurde diese angepasst. Es wurde ein Plan für ein Gehäuse aus Holz erstellt.

Für das Gehäuse wurde auch ein 3D-gedrucktes inneres Gerüst entworfen. An diesem Gerüst sind die Lautsprecher, der Bildschirm und die vordere Abdeckung befestigt. Nach dem Misserfolg mit dem Beton musste dieses Gerüst leicht angepasst werden, konnte aber seine ursprüngliche Bestimmung erfüllen.



Abbildung 3: Beton Fehlschläge

## 4.2 Das endgültige Gehäuse

Das Gehäuse wird nach Plan aus MDF (Mitteldichte Holzfasern) gefertigt. Die Holzplatten werden zugeschnitten, gebohrt und zusammengeleimt. Im Anschluss werden sie lackiert. MDF-Platten eignen sich gut zum Bau von Lautsprecherboxen, weil sie gleichermaßen homogen sind und eine wesentlich geringere Festigkeit als Vollholz haben, weswegen sie leichter zu bearbeiten sind.

Das Gehäuse besteht aus drei großen Teilen. Es setzt sich aus dem MDF-Gehäuse, einem 3D-gedruckten inneren Gerüst und aus den lasergeschnittenen, vorderen und hinteren Abdeckplatten zusammen. Es wurden zwei Versionen des Gerüsts geplant, aber nur eine, die die in dem fertigen Produkt eingebaut wurde, gefertigt. Innen wurde das Gehäuse mit Dämmwatte ausgekleidet ([Quelle: 12.3.5](#)). Das MDF-Gehäuse und die Dämmwatte verbessern die akustischen Eigenschaften. An dem 3D-gedruckten inneren Gerüst werden alle Bauteile – die Lautsprecher, das Display und die Leiterplatte – befestigt. Dieses Gerüst wird mit den zwei Abdeckplatten abgedeckt, um das Radio ästhetisch ansprechender zu gestalten.

Alle Pläne für das Gehäuse wurden mit Fusion 360 ([Quelle: 12.3.1](#)) erstellt.

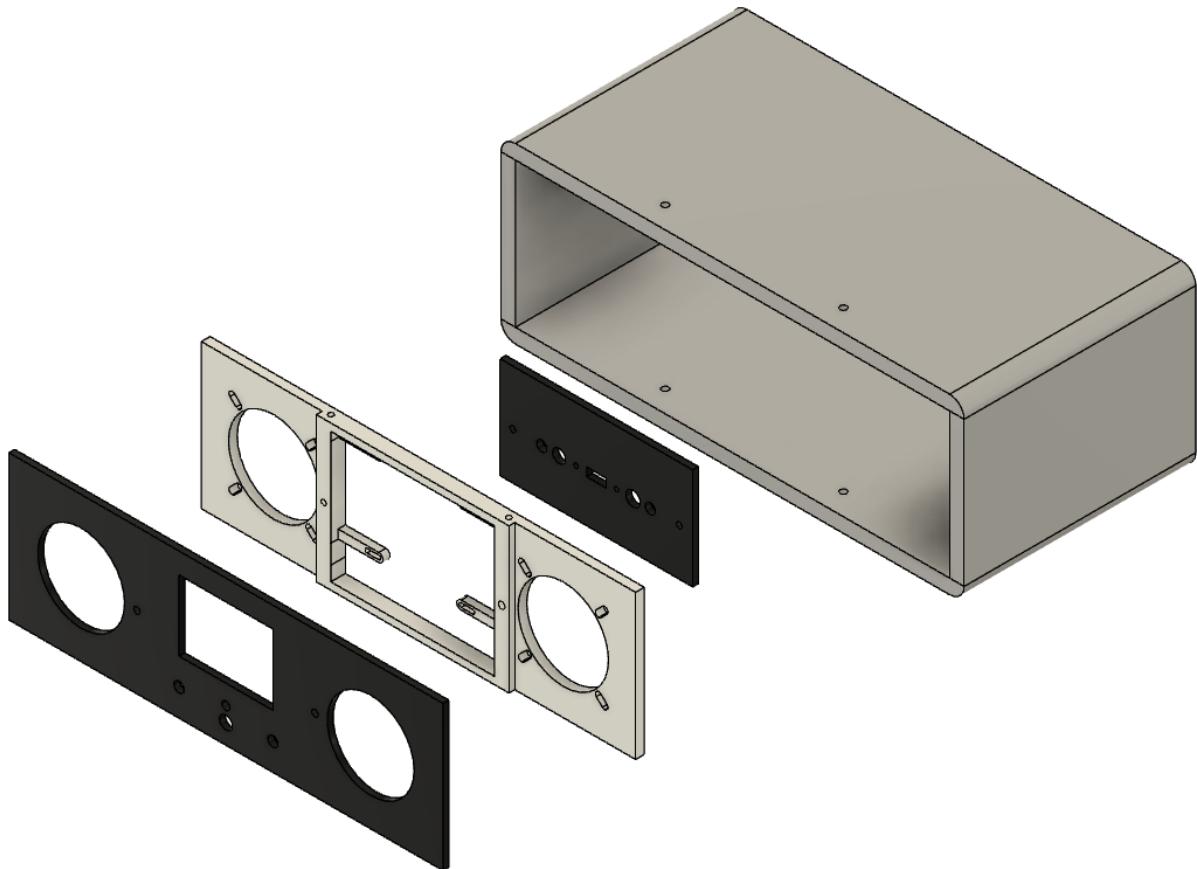


Abbildung 5: Das endgültige Gehäuse

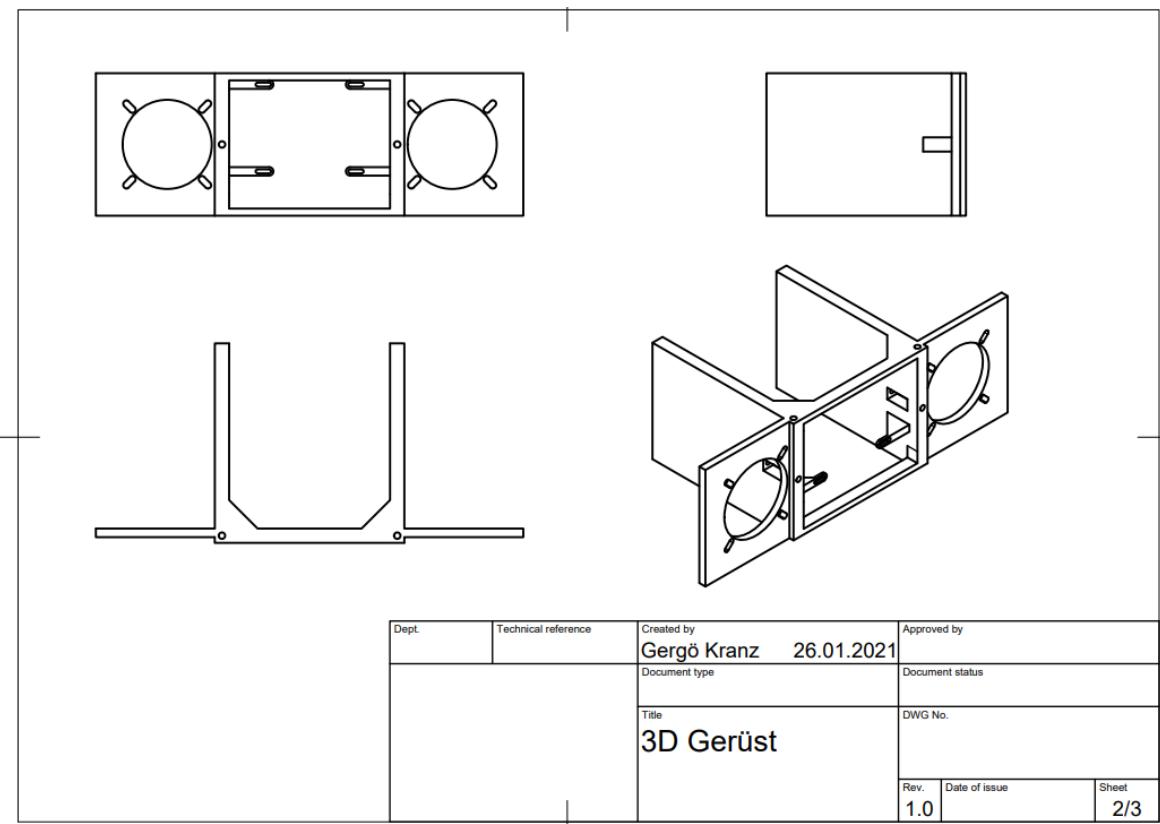


Abbildung 4: Alternatives 3D Gerüst, das nicht gefertigt wurde

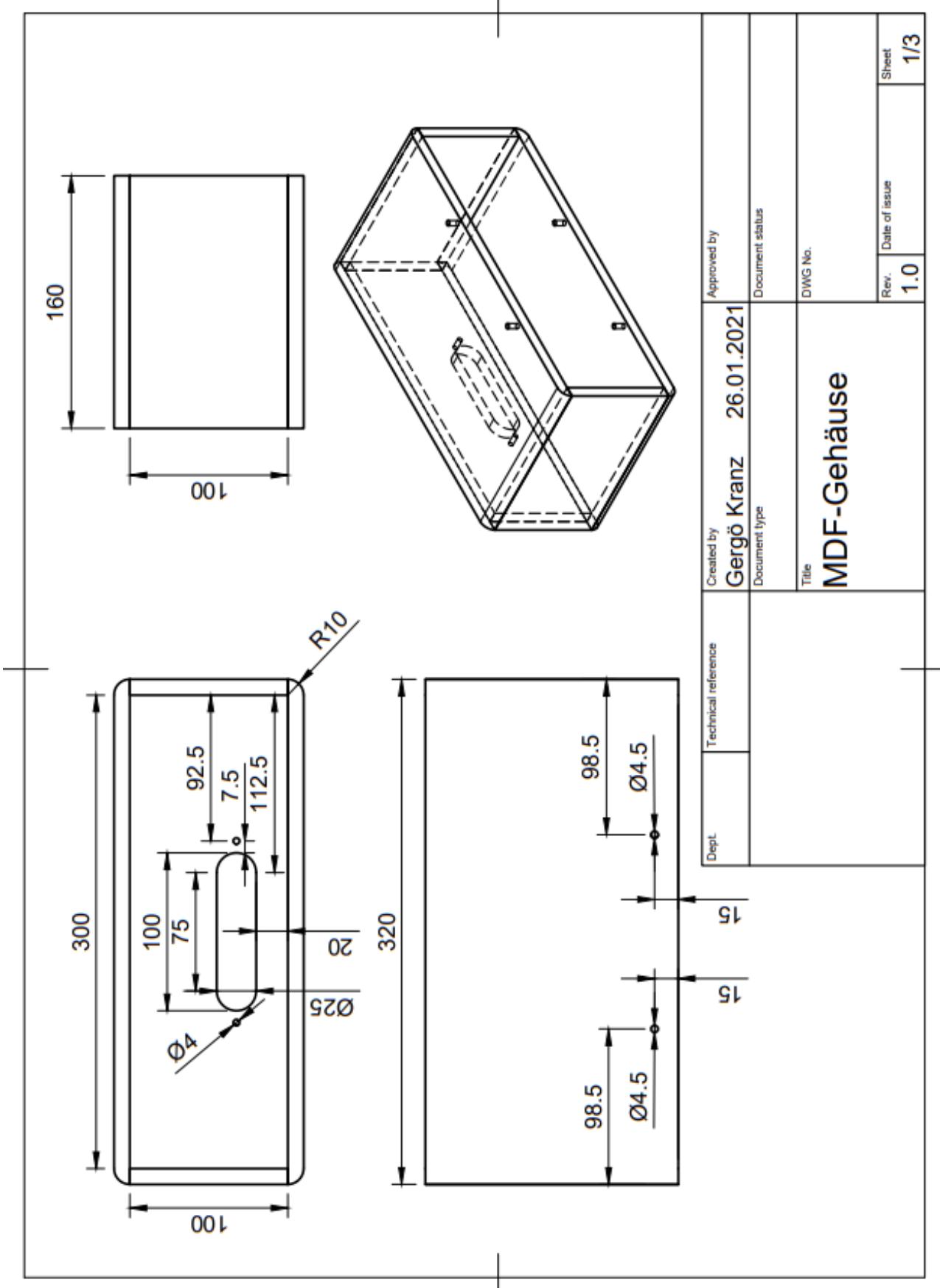


Abbildung 6: Gehäuseplan

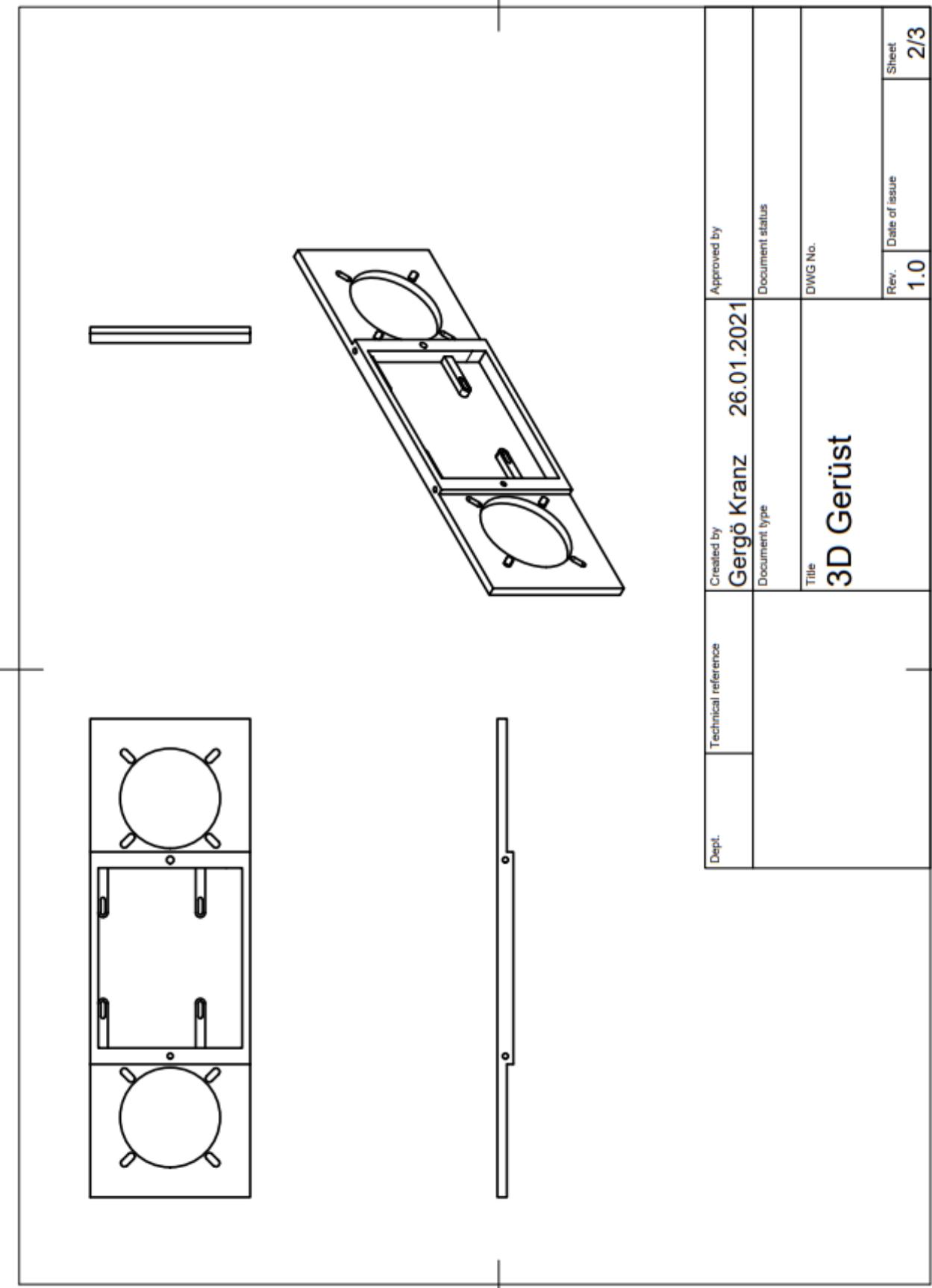


Abbildung 7: Gehäuseplan (2)

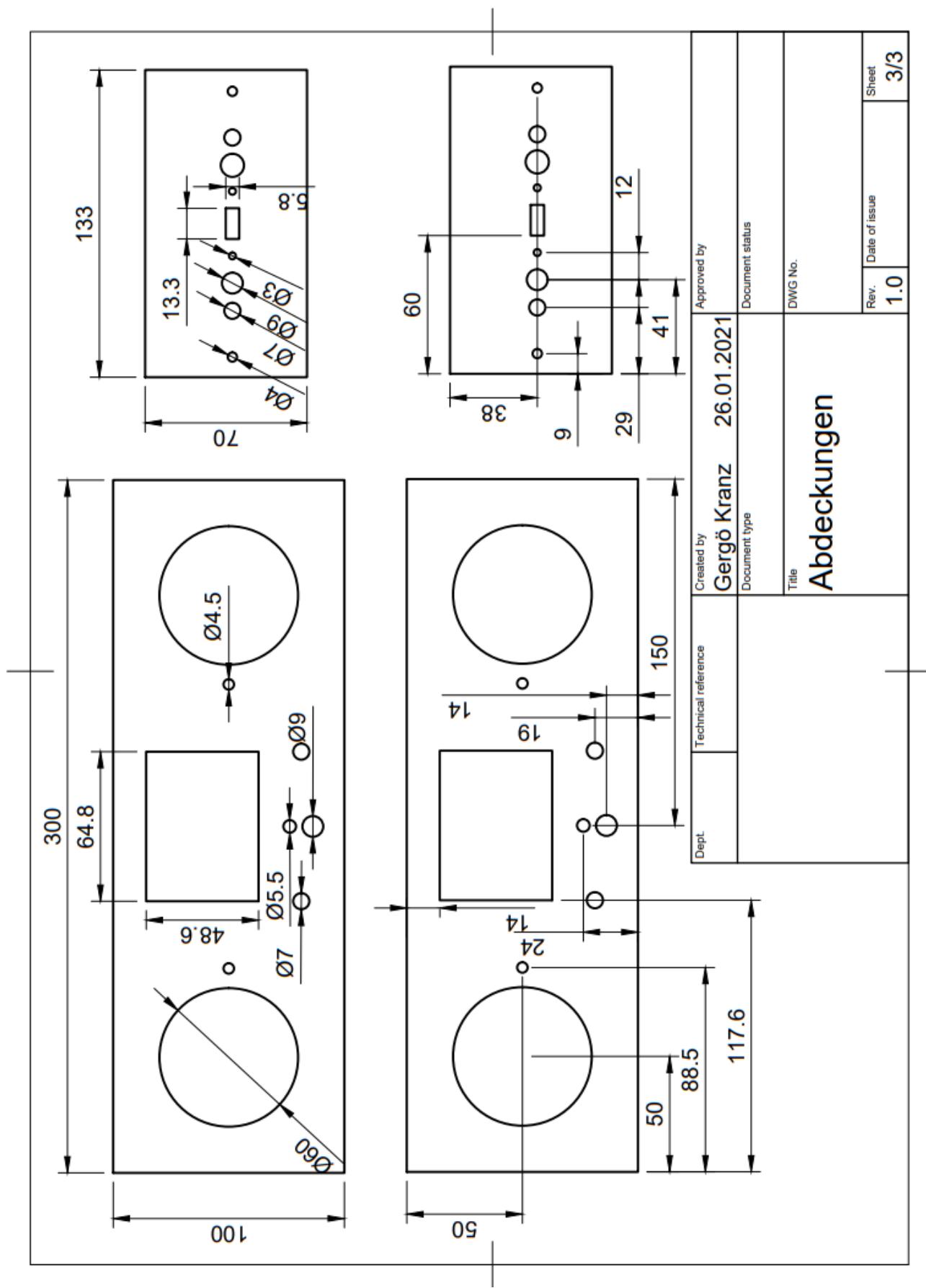


Abbildung 8: Gehäuseplan (3)



Abbildung 9: Zusammengeklebtes Gehäuse

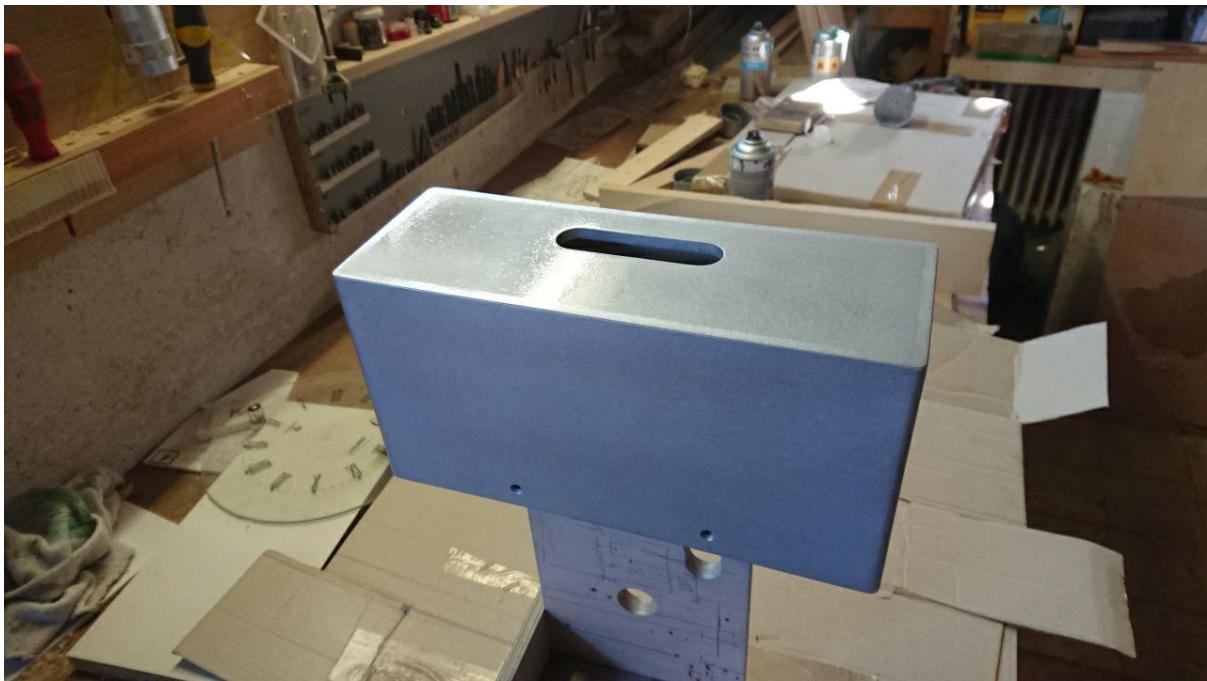


Abbildung 10: Lackieren des Gehäuses



Abbildung 11: Fertiges Gehäuse



Abbildung 12: Lackieren der Frontplatte



Abbildung 13: Fertig zusammengebauter Prototyp



## 5 Schaltung und Layout

### 5.1 Komponenten

Die Schaltung besteht aus einer doppelseitigen Leiterplatte und mehreren wichtigen Bauteilen, die direkt an der Platine montiert sind, und aus einigen Modulen, wie das FM-, und DAC-Modul.

#### 5.1.1 Verstärker

Der größte und komplizierteste Teil ist die Verstärkerschaltung. Der IC TPA3144D2 von Texas Instruments ist ein 6W Stereooverstärker mit einem breiten Versorgungsspannungsbereich von 4.5V bis 14.4V. Dieser IC ist mit einem Lautsprecherschutz und einem Pin (EAGLE: /shutdown) zum Ausschalten des ICs versehen, der ein popfreies Ein- und Ausschalten ermöglicht. Der IC wird immer ausgeschaltet, wenn keine Musik gespielt wird. Außerdem hat er vier auswählbare, fixierte Verstärkungseinstellungen, die mit einem einfachen Spannungsteiler eingestellt werden können. Die Verstärkung ist auf 32dB eingestellt. ([Quelle: 12.1.3 Datenblatt](#))

Die Verstärkerschaltung ist aus dem Datenblatt und besteht aus dem Verstärker-IC und aus Kondensatoren, die an den Ein- und Ausgängen platziert sind, um schnellen Spannungsänderungen entgegenzuwirken, was ein möglichst konstantes Signal an den Lautsprechern ermöglicht. Andere Kondensatoren, die an der Vorstufe und Endstufe angeschlossen sind, filtern den Gleichstromanteil aus, um das Übertragen des Signals ohne zusätzliche Störungen zu ermöglichen. Idealerweise sollten nur die eigentlichen Informationen weitergegeben werden. Es soll also nur die Wechselspannung übertragen werden, aber keine Gleichspannung. Diese Aufgabe übernehmen die Koppel-Kondensatoren, die die Gleichspannung sperren, die Wechselspannung aber durchlassen. Die Widerstände dienen als einfache Spannungsteiler, um die Verstärkerstufe und das Threshold-Limit einzustellen.

#### 5.1.2 Audiodämpfungsglied

Da der Verstärker eine feste, nicht während der Laufzeit änderbare Verstärkung hat, braucht man einen zusätzlichen IC, um die Lautstärke einstellen zu können. Zu diesem Zweck besitzt die Schaltung ein Audiodämpfungsglied, auch Attenuator genannt, mit der Bezeichnung LM1972. Dieses dämpft die eingehenden Audiosignale und wird über ein SPI kompatibles Interface angesteuert. Der IC besitzt bis zu 104dB Stummschaltung und Pop-und-Klick-freie Lautstärkeunterdrückung. ([Quelle: 12.1.2 Datenblatt](#))



### 5.1.3 Multiplexer

Die Schaltung beinhaltet außerdem noch zwei TMUX1309 Multiplexer. Sie haben einen breite Versorgungsspannungsbereich von 1.62V bis 5.5V. Jeder Baustein besitzt vier Stereoeingänge und einen Stereoausgang. Sie brauchen keine zusätzlichen Bauteile, weil sie nur einfache Schalter sind, die über einen zwei Bit Eingang, mit TTL Logik, angesteuert werden. Ihre Besonderheit ist, dass sie bidirektional sind. Sie können Signale zwischen vier Eingängen und einem Ausgang, oder einem Eingang und vier Ausgängen, hin- und herschalten. ([Quelle: 12.1.1 Datenblatt](#))

Sie sind vor und hinter dem Lautstärkeregler angeschlossen und können die ein- und ausgehenden Signale zwischen Quelle, wie Digital-Analog Converter oder FM-Modul, und Ziel, wie Lautsprecher oder Kopfhörerausgang, umschalten. Sie werden vom Raspberry Pi Zero W angesteuert.

### 5.1.4 Raspberry Pi Zero W

Der Pi Zero W ist ein Linux Einplatinencomputer mit einem 1 Ghz Single-Core-Prozessor, 512 MB RAM, USB 2.0 und eingebautem Bluetooth 4.1 und W-LAN b/g/n. ([Quelle: 12.1.9 Datenblatt](#))

Am Raspberry Pi Zero W sind auch der Infrarotempfänger und das Display angebunden.

### 5.1.5 Display

Als Display wurde das ER-TFTM032-3, wegen der ausreichenden, aber dennoch ressourcensparenden Auflösung von 240x320 Pixeln, dem geringen Stromverbrauch und dem kapazitiven Touchscreen gewählt. Es hat eine 3,2 Zoll Diagonale und wird über SPI mit Bilddaten versorgt. ([Quelle: 12.1.10 Datenblatt](#))

### 5.1.6 Infrarotempfänger

Als Infrarotempfänger wurde der TSOP48 benutzt. Dieser hat einen Data, Versorgungs und Ground Pin. Der Data Pin kann direkt an den Pi angeschlossen und ausgelesen werden. An der Versorgungsleitung sollte, laut Datenblatt ([Quelle 12.1.8](#)), ein Tiefpass angeschlossen sein, um hochfrequente Störungen herauszufiltern.

### 5.1.7 Coprozessor

Als Coprozessor dient ein Arduino (ATMEGA 328P-PU). Dieser liest die Stellung der Drehimpulsgeber ein, und schaltet das Radio in den Stand-by-Modus. Das erfolgt durch das Schalten des MOSFETs, dieser wird mit der Hilfe eines NPN-Transistors angesteuert, um die Stromversorgung vom Raspberrys und Verstärker zu trennen. Der Arduino selbst besitzt einen Kondensator, zum Schutz vor kurzen, hohen Strömen beim Einschalten, und einen 8MHz Quartz zur externen Takterzeugung. Die Beschaltung des Quarzes und Arduinos erfolgt wie im Datenblatt ([Quelle 12.1.11](#)) beschrieben. Die Kommunikation zwischen Pi und Arduino erfolgt mit UART.



### 5.1.8 Spannungsversorgung

Die Schaltung enthält Komponenten mit einer Versorgungsspannung von sowohl 5V als auch 3,3V. Der Spannungsregler, LD1117, ist für die Senkung der Spannung vom 5V Netzteileingang auf 3,3V verantwortlich. Dieser IC wurde in der Variante mit einer fixen Ausgangsspannung von 3,3V bestellt.

### 5.1.9 Zusätzliche Module und Anschlüsse

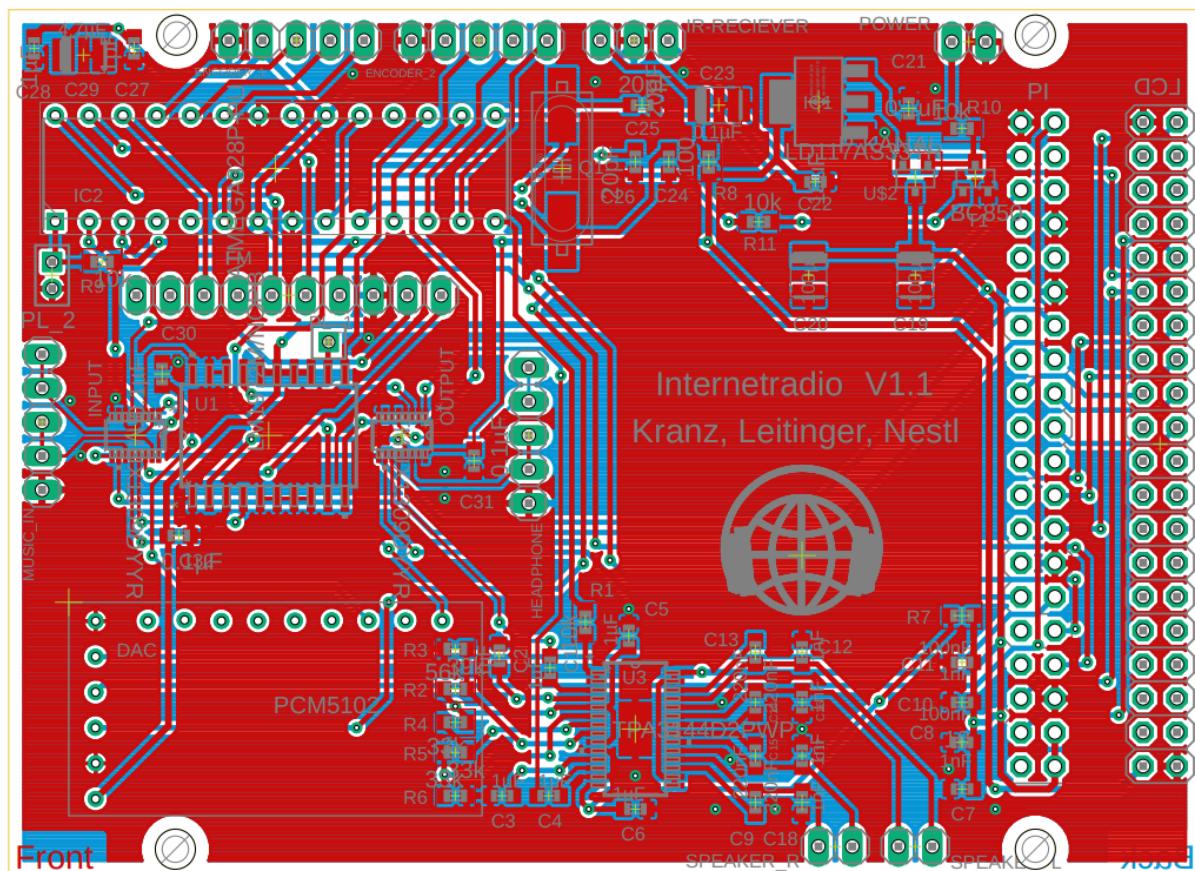
Das fertige FM- und DAC-Modul kann direkt auf die Leiterplatte gesteckt werden. Die Platine besitzt außerdem noch Verbindmöglichkeiten für die zwei eingebauten Lautsprecher, für zwei Drehregler und zwei Audiobuchsen.

## 5.2 Planung und Fertigung

Die Leiterplatte wurde in EAGLE geplant und von JLCPCB gefertigt. Die Bestellung erfolgt durch das Hochladen einer Gerber-Datei, die aus EAGLE exportiert wird. Nach der Bezahlung wird die Fertigung innerhalb von 24 Stunden begonnen.

## 5.3 Probleme

Leider wurden beim Bau kleine, unwesentliche Planungsfehler im Layout festgestellt. (Siehe: [Sektion Tests](#)) Aus diesem Grund wurde eine zweite Version des Layouts (V 1.1) erstellt, aber nie gefertigt.



*Abbildung 14: Layout V1.1 (Rückseite gleich wie V1.0)*

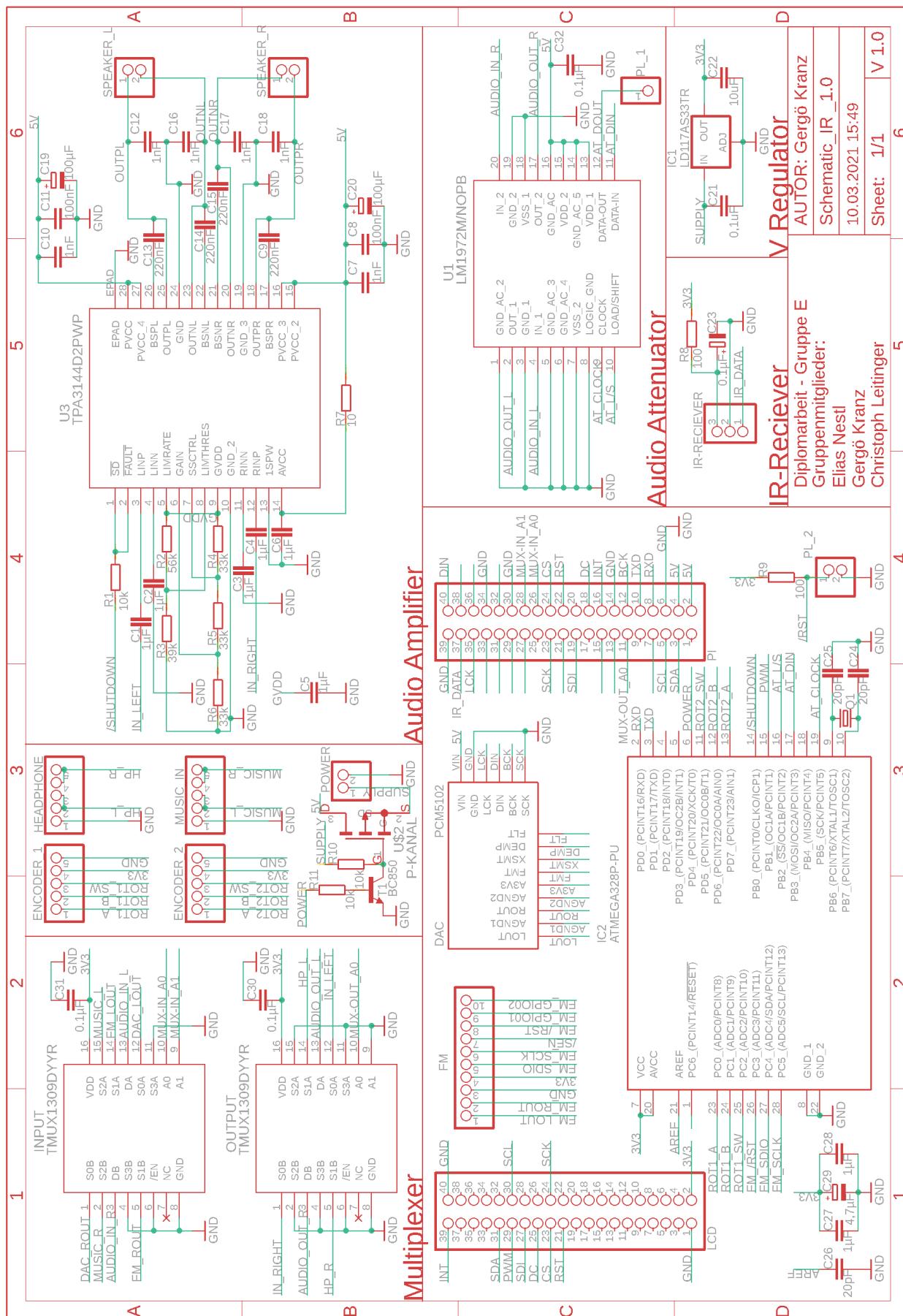


Abbildung 15: Schaltplan

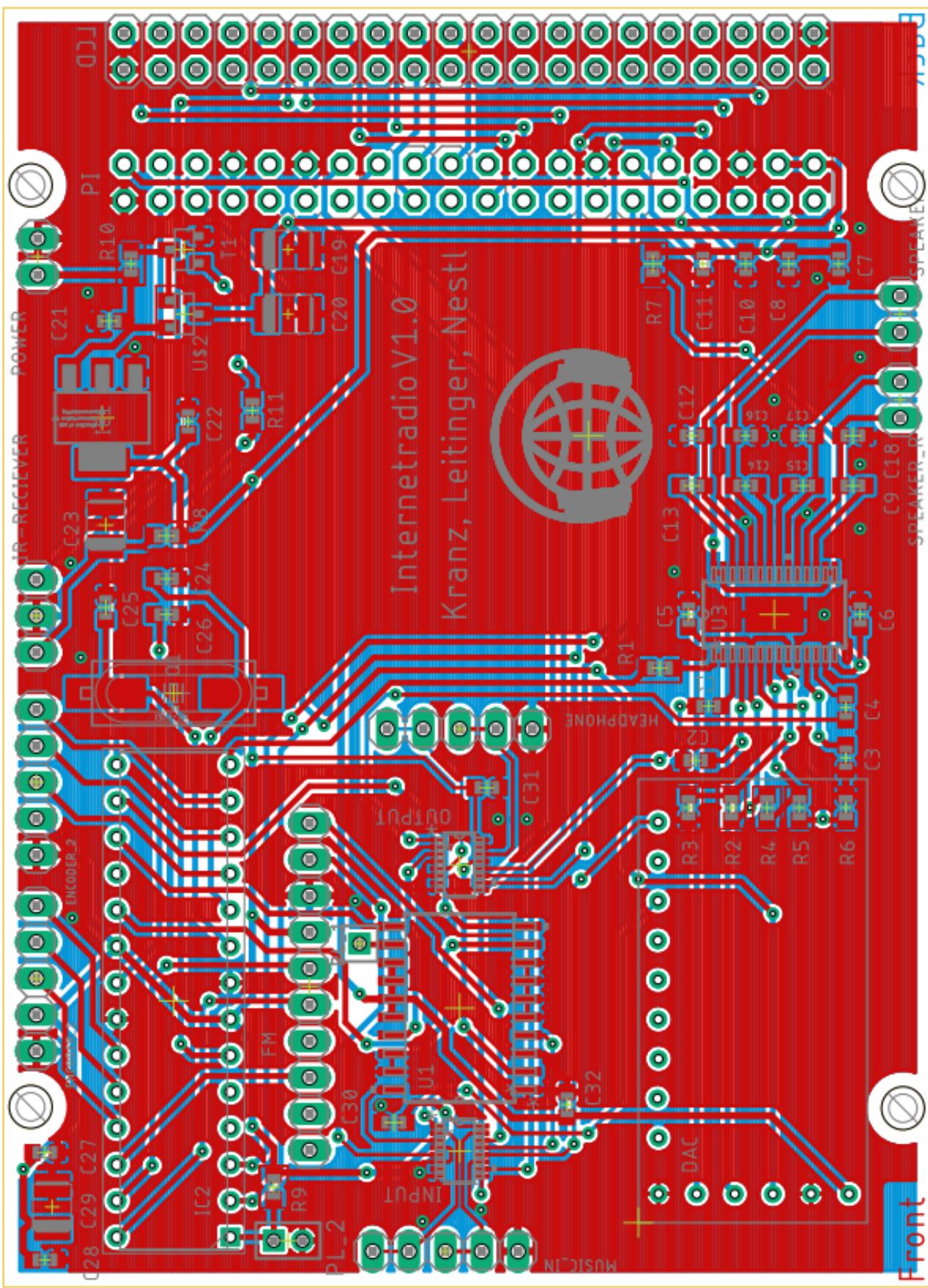


Abbildung 16: Layout V1.0 (Vorderseite)

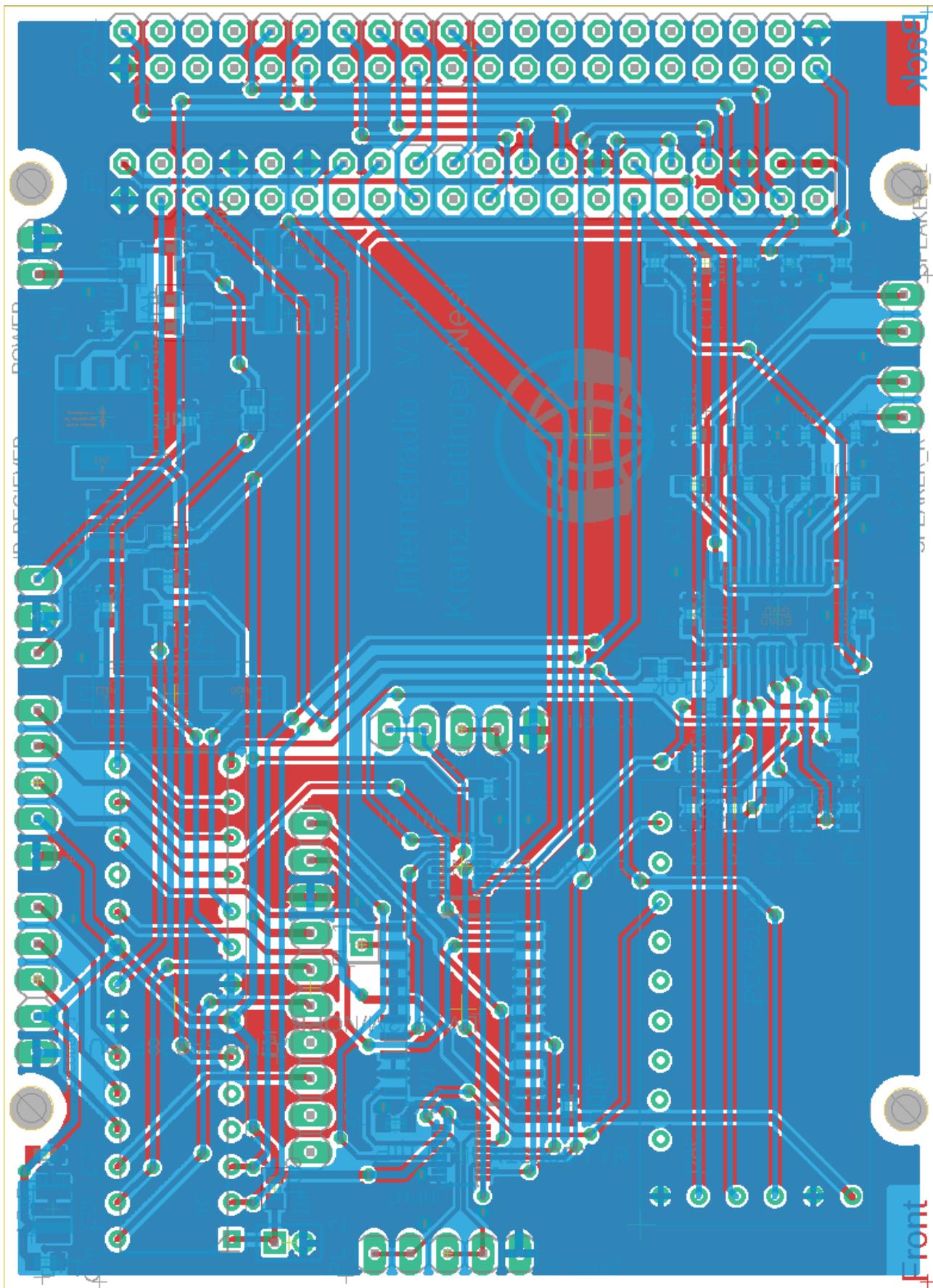


Abbildung 17: Layout V1.0 (Rückseite)

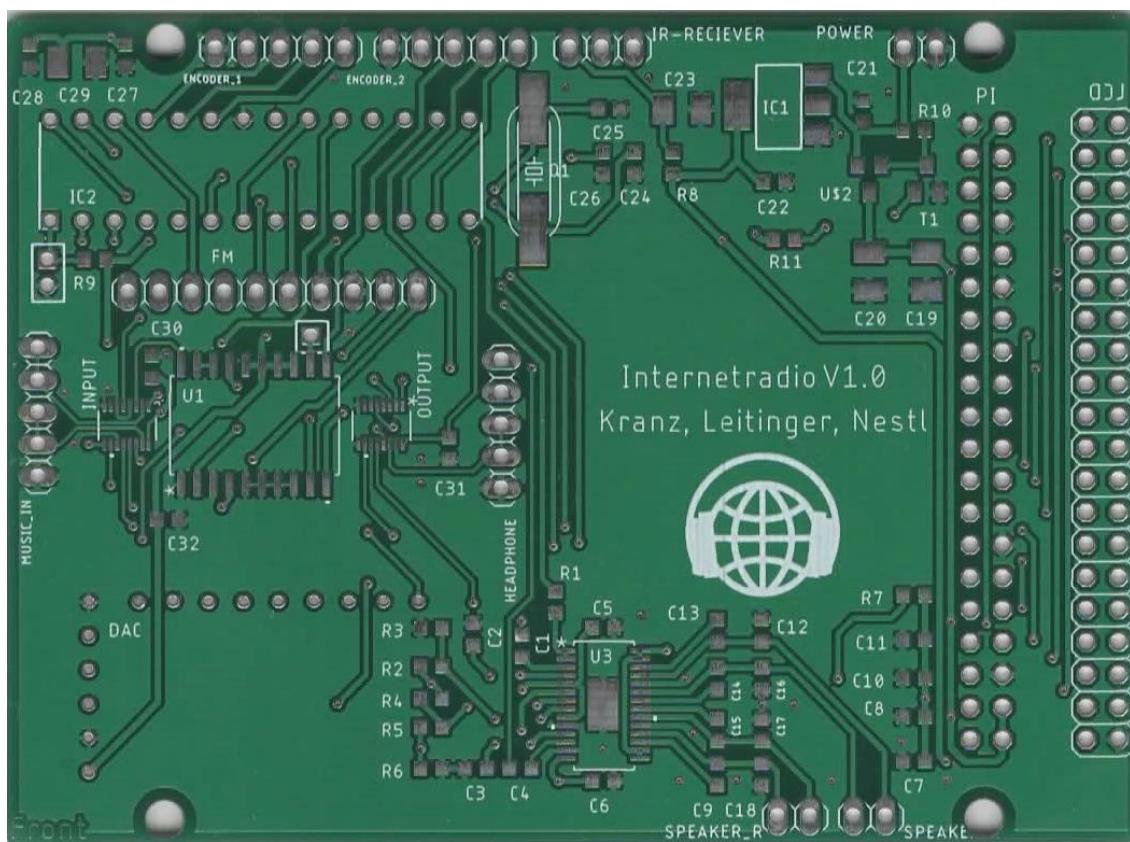


Abbildung 19: Vorderseite der Leiterplatte

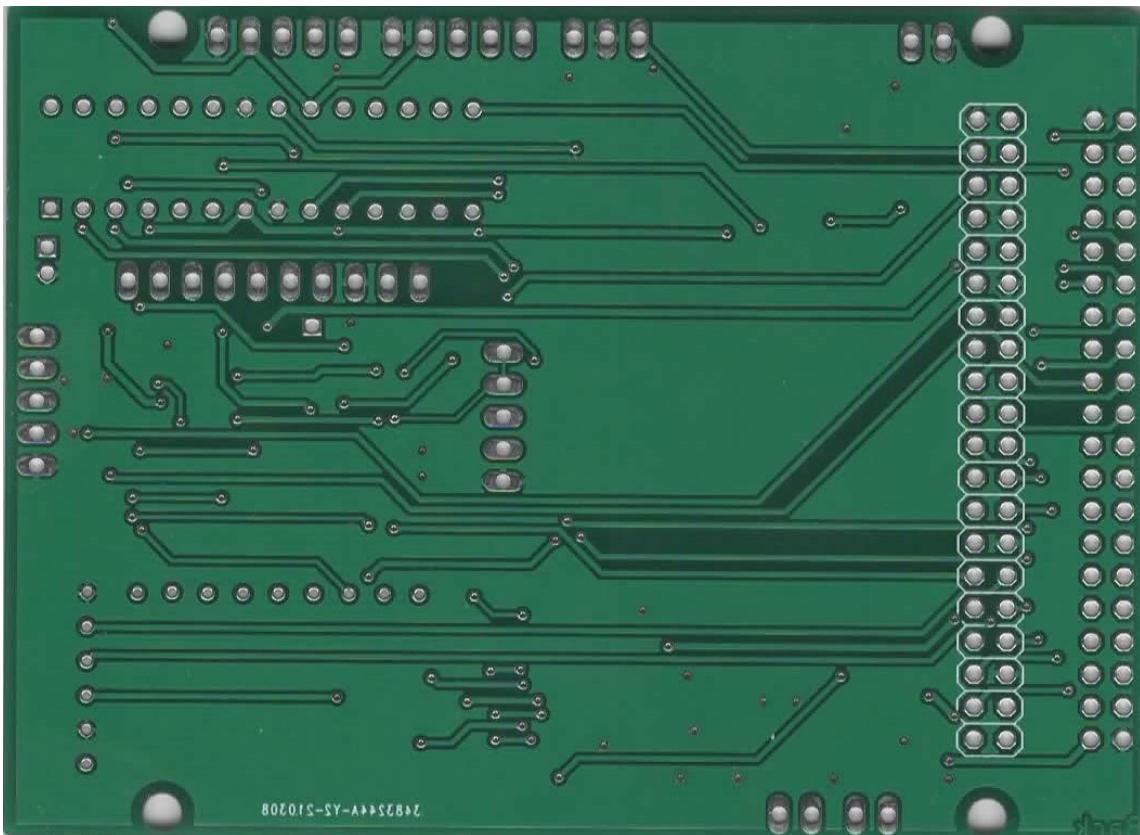


Abbildung 18: Rückseite der Leiterplatte



## 6 Stücklisten

### 6.1 Elektronik

Anzahl	Bezeichnung	Wert	Typ
8	Kondensator	1 µF	C0603K
6	Kondensator	1 nF	C0603K
2	Kondensator	100 nF	C0603K
4	Kondensator	220 nF	C0603K
2	Elko	100 µF	SMD
4	Kondensator	0,1 µF	C0603K
1	Kondensator	10 µF	C0603K
1	Elko	0,1 µF	SMD
3	Kondensator	20 pF	C0603K
1	Elko	4,7 µF	SMD
1	DAC		PCM5102
2	Rotary Encoder mit Schalter		PEC11R-4215F-S0024
1	FM Modul		Si4703 Basic
2	Klinkeneinbaubuchsen	3,5 mm	54-0008
1	LD1117AS33TR		SOT223
1	Arduino		ATMEGA328P-PU
2	TMUX1309DYYR		SOIC_309DYYR_TEX
1	IR-Empfänger		THT
1	LCD	320x240 Pixel	ER-TFTM032-3
3	Widerstand	10 kΩ	R0603
1	Widerstand	56 kΩ	R0603
1	Widerstand	39 kΩ	R0603
3	Widerstand	33 kΩ	R0603
1	Widerstand	10 Ω	R0603
2	Widerstand	100 Ω	R0603
1	Raspberry		Zero W
2	Lautsprecher	8 Ω	FRS 7 W 8 OHM
1	Transistor	BC850	SOT23
1	MOSFET	Si2323CDS	SOT23
1	LM1972M/NOPB		M20B_TEX
1	TPA3144D2PWP		PWP28_2P4X3.4_TEX
1	DC-Einbaubuchse		Stift 2,1 mm
1	Schaltnetzteil	5V / 3A	
1	Quartz	8 MHz	SMD
1	ATMEGA328P-PU		THT
10	Pinheader		
10	Pinstecker		
3m	Leitungen		
1	Mikro-USB zu USB-A Kabel		



## 6.2 Gehäuse

Anzahl	Bezeichnung	Größe
1	MDF-Platte	146x400 mm
1	3D gedruckte Gerüst	300x100 mm
1	Sperrholzplatte 4mm	297x420 mm
16	Schrauben	M4x20
4	Schrauben	M3x12
1	Dämmwatte	630x330 mm



## 7 Firmware

Firmware bezeichnet hier die Interaktion zwischen Software und Hardware.

### 7.1 Kommunikation zwischen Co- und Hauptprozessor

Die Kommunikation zwischen dem Hauptprozessor (Raspberry Pi Zero W) und dem Coprozessor (ATmega328P) geschieht über UART mit einer Baudrate von 9600. Eine höhere Geschwindigkeit kann nicht gewählt werden, da der Mikrocontroller nur mit 8 MHz taktet und eine höhere Datenrate außerhalb der Spezifikation für diese Geschwindigkeit läge. ([Quelle: 12.1.11 Datenblatt](#))

#### 7.1.1 Software am Coprozessor

Am ATmega328P läuft ein auf der Arduino Plattform basierendes Programm, dass Befehle vom Pi über die Kommunikationsschnittstelle erhält, diese verarbeitet und eine Antwort liefert.

Um Bandbreite zu sparen, sind die Befehle einem C enum definiert.

Eine Befehlsnachricht vom Hauptprozessor besteht aus der Zahl aus dem enum und, falls vorhanden, den Argumenten des Befehls. Getrennt sind die beiden durch ein Pluszeichen ("+") und das Ende der Nachricht wird mit einem Newline-Charakter ("\n") gekennzeichnet.

Die Antwort geschieht über das e\_cmd\_response Ereignis (siehe [Sektion Ereignisse](#)).

Beispielsweise sieht eine Nachricht, die die Attenuation auf 10 ändert, so aus:

3+10

#### 7.1.1.1 „Hintergrund“-Prozesse

Die Drehimpulsgeber werden über das AdaEncoder Library ([Quelle 12.2.20](#)) verarbeitet. Die Wahl fiel auf dieses Library, da es die PCINT (Pin-Change-Interrupt) Interrupts des ATmegas nutzen kann, die ein Interrupt liefern, wenn irgendein Pin auf dem jeweiligen Port den Zustand ändert. Denn er verfügt nur über zwei „reale“ Interrupt Pins ([Quelle: 12.1.11 Datenblatt](#)).

Wenn der erste Drehimpulsgeber gedreht oder gedrückt wird, dann erhält der Raspberry die Information und kann sie verarbeiten. Wenn der zweite gedreht oder gedrückt wird, dann ändert der Arduino die Attenuation selbstständig und liefert die neue Attenuation an den Pi.

#### 7.1.1.2 Ereignisse

Damit der Raspberry nicht ständig nachfragen muss, ob etwas im Hintergrund geschehen ist (auch Polling genannt), kann der Coprozessor, von sich aus, eine Nachricht an den Raspberry senden, um Ereignisse bekanntzugeben.

Das Nachrichtenformat für Ereignisse, sieht gleich aus wie das der Befehle, jedoch werden die Nachrichten vom 328P zum Pi geschickt anstatt in die andere Richtung.



### 7.1.1.3 Kommunikation mit Attenuator

Das Interface des Attenuator ist mit SPI kompatibel, da es wie ein einfaches Shiftregister funktioniert ([Quelle 12.1.2 Datenblatt](#)). Daher wird die SPI Peripherie im ATmega benutzt, um den Attenuator anzusprechen ([Quelle 12.2.21 japanischer Blogpost zum LM1972 am Arduino](#)).

### 7.1.1.4 Kommunikation mit dem FM-Modul

Die Kommunikation mit dem Si4703 Modul läuft über die I<sup>2</sup>C Peripheriehardware im ATmega328P.

Das Library „Radio“ vom GitHub Benutzer „mathertel“ ([Quelle 12.2.22](#)) wird genutzt, da es einen hohen Funktionsumfang und einen eingebauten RDS-Decoder bietet.



#### 7.1.1.5 Liste der Befehle

Name im enum	Beschreibung	Parameter	Antwort
c_change_att	Lautstärkedämpfung verändern	Dämpfungslevel als Zahl zwischen 0 und 127	Die übergebene Zahl
c_set_amp_power	Verstärker ein- oder ausschalten	0 für aus, 1 für ein	Der neue Zustand („on“ oder „off“)
c_set_lcd_pwm	Displayhelligkeit einstellen	Helligkeit als Zahl zwischen 0 und 255	Die übergebene Zahl
c_power_off	Standby-Modus aktivieren		
c_fm_init	FM-Modul initialisieren		
c_fm_deinit	FM-Modul deinitialisieren		
c_fm_scan_up	Frequenzbereich nach oben absuchen		
c_fm_scan_down	Frequenzbereich nach unten absuchen		
c_fm_set_freq	Frequenz setzen	Die Frequenz in Mhz mal 100 als Zahl	Die übergebene Zahl
c_fm_set_mono	Mono ein/aus	0 für aus, 1 für ein	
c_fm_set_vol	Interne Laustärke des FM-Moduls festlegen	Laustärke als Zahl zwischen 0 und 15	Die übergebene Zahl

#### 7.1.1.6 Liste der Ereignisse

Name im enum	Beschreibung	Parameter
e_cmd_response	Antwort auf einen Befehl	Die Antwort des Befehls
e_rot0_dir1	Rotation des 1. Drehimpulsgebers	
e_rot0_dir2	Rotation des 1. Drehimpulsgebers (andere Richtung)	
e_rot0_btn	Knopfdruck am 1. Drehimpulsgeber	
e_rot1_btn	Knopfdruck am 2. Drehimpulsgeber	
e_change_att	Veränderung der Lautstärkedämpfung (geschieht beim Drehen des 2. Drehimpulsgebers)	Neues Dämpfungslevel
e_fm_scan_complete	Rückgabe des FM-Scan-Ergebnisses	Die Frequenz in Mhz mal 100 als Zahl
e_fm_rds	Ausgabe der Empfangenen RDS Daten	Der Sendername



### 7.1.1.7 Arduino Bootloader auf den Coprozessor schreiben

Eine ESP32 Entwicklungsplatine wird genutzt, um den Arduino Bootloader auf den ATmega328P zu schreiben (auch flashen oder programmieren genannt), da sie, wie der Coprozessor des Internetradios, mit einem Logikpegel von 3,3V arbeitet.

(Quellen: [12.1.14](#), [12.1.15](#))

Der Prozess funktioniert folgendermaßen:

Als erstes muss die Programmiersoftware auf den ESP32 geladen werden, dazu öffnet man sie in der Arduino IDE über den Menüpunkt File -> Examples -> ArduinoISP.

Anschließend muss die richtige Platine über Tools -> Board gewählt werden (in diesem Fall „Lolin D32“). Dann wird unter Tools -> Port die passende serielle Schnittstelle (siehe Windows Geräte-Manager) gewählt.

Da der Code des ArduinoISP ursprünglich nicht für den ESP32 sondern für AVR basierte Controller gedacht wurde, muss Zeile 269, `analogWrite(LED_HB, hbval);` auskommentiert werden.

Außerdem muss in Zeile 73, der RESET Pin auf 22 geändert werden (zumindest für die unten beschriebene Beschaltung): `#define RESET 22`

Dann kann der Code mit dem „Upload“ Knopf auf den ESP32 geladen und die Pins verbunden werden:

(IO Pins bei ESP32, Pin Nummer in 328P Spalte)

ESP32	328P
GND	8 GND
3.3V	7 VCC
18 SCK	19 SCK
19 MISO	18 MISO
23 MOSI	17 MOSI
22 RESET	1 RESET

Bei Tools -> Board den „Arduino Pro or Pro Mini“ und den Prozessor „ATmega328P (3.3V, 8 MHz)“ wählen

Tools -> Burn Bootloader

Serial Adapter mit RX/TX von 328P verbinden

Versuchen einen Sketch normal mit Upload hochzuladen (Reset Taste drücken, wenn Text auf Uploading... geht)

## 7.2 Infrarot

Über das LIRC Softwarepaket verarbeitet der Raspberry die Codes, die die Infrarotfernbedienung sendet. (Quelle: [12.2.15](#))



## 8 Software

### 8.1 Raspberry Pi

Die gesamte Software ist in der Sprache C++ programmiert.

Für die Benutzeroberfläche wird die Bibliothek LVGL ([Quelle: 12.2.5](#)) verwendet. LVGL ist eine Open-Source Bibliothek mit einer großen Vielfalt an Grafik- und Steuerelementen. LVGL ist visuell ansprechend gestaltet und effizient bei der Speicherauslastung.

Das Projekt baut auf dem [LVGL Simulator Beispiel](#) auf, da es einen einfachen Einstieg bietet, um LVGL Code am PC auszuführen ([Quelle: 12.2.9](#)).

Da es SDL benutzt, wird es auch direkt am Framebuffer des Raspberries ausgegeben, wenn es auf ihm gestartet wird.

### 8.2 Klassen

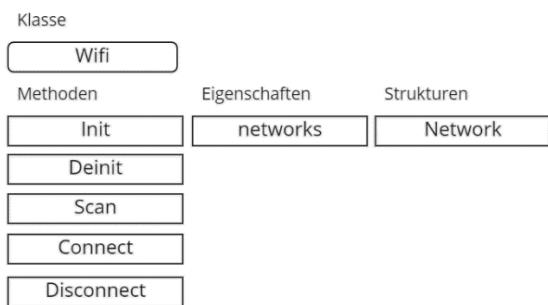
Das Programm ist objektorientiert gestaltet. Entsprechend dieser Strukturierung werden hier die öffentlichen (public) Eigenschaften und Funktionen beschrieben. Die Funktionen und Eigenschaften die nur von den Klassen selbst verwendet werden können sind nur für die Funktionalität der Klasse selbst von Bedeutung. Da diese privaten (private) Eigenschaften und Funktionen nicht zum Verständnis der Interaktion zwischen den Klassen beitragen werden sie hier nicht angeführt.



## 8.2.1 Klasse - Wifi

---

Ist für die Verbindung mit W-Lan Netzwerken verantwortlich. Dafür wird das Programm wpa\_supplicant ([Quelle: 12.2.10](#)) verwendet, das über dessen D-Bus Interface ([Quelle: 12.2.1](#)) angesteuert wird. Diese Klasse bildet dabei eine Abstraktionsschicht zur Ansteuerung dieses Programms. Sie ruft D-Bus Funktionen auf und stellt Informationen bereit.



### Eigenschaften

`vector<Network>`  
networks

Liste der verfügbaren Netzwerke.

## Struktur - Network

---

Beinhaltet alle Informationen über ein Netzwerk die an das Restliche Programm übergeben werden können. Des Weiteren dient ein Objekt von diesem Typ der Identifizierung eines Netzwerkes für Funktionen dieser Klasse.

### Eigenschaften

`string ssid`

Name des Netzwerks.

`string psk`

Passwort des Netzwerks.

`bool isConnected`

Gibt an, ob dieses Netzwerk verbunden ist.

## Funktion - Init

---

Initialisiert die Klasse:

- Stellt eine Verbindung mit dem System D-Bus her.
- Initialisiert Variablen, die für die Klasse benötigt werden.

### Funktionssignatur

`static void Init(void)`

### Argumente

`void`

### Rückgabetyp

`void`



## Funktion - Deinit

---

Deinitialisiert die Klasse:

- Trennt die Verbindung mit dem System D-Bus.
- Gibt Variablen frei.

### Funktionssignatur

**static void Deinit(void)**

Argumente                   **void**

Rückgabetyp                   **void**

## Funktion - Scan

---

Startet eine Suche nach verfügbaren W-Lan Netzwerken.

### Funktionssignatur

**static void Scan(void(\*CallBack)())**

Argumente                   **void(\*CallBack)()**

Functionpointer einer Funktion, die aufgerufen werden soll, wenn der Suchvorgang beendet ist.

Rückgabetyp                   **void**

## Funktion - Connect

---

Startet einen Versuch sich mit einem W-Lan Netzwerk zu verbinden.

### Funktionssignatur

**static void Connect(Network network)**

Argumente                   **Network network**

Netzwerk mit dem eine Verbindung hergestellt werden soll.

Rückgabetyp                   **void**

## Funktion - Disconnect

---

Trennt das Netzwerk mit dem aktuell eine Verbindung besteht.

### Funktionssignatur

**static void Disconnect(void)**

Argumente                   **void**

Rückgabetyp                   **void**

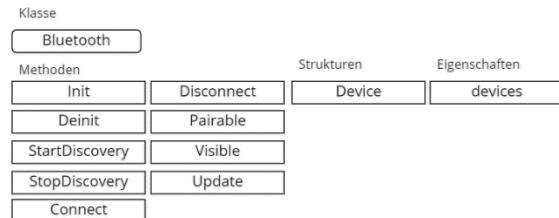
---



## 8.2.2 Klasse - Bluetooth

---

Ist für die Verbindung mit Bluetooth Geräten verantwortlich. Dafür werden die Programme **bluez** (Quelle: 12.2.11) und **mpрис** (Quelle: 12.2.12) verwendet, die über dessen D-Bus Interfaces (Quelle: 12.2.2 - 3) angesteuert. Alle Bereiche die mit der Verbindung von Geräten zu tun hat übernimmt hierbei bluez. Um die Wiedergabe von Musik über Bluetooth zu beeinflussen wird mpрис verwendet. Diese Klasse bildet dabei eine Abstraktionsschicht zur Ansteuerung dieser Programme.

**Eigenschaften**`vector<Device> devices`

Liste der verfügbaren Geräte.

## Struktur - Device

---

Beinhaltet alle Informationen über ein Gerät die an das Restliche Programm übergeben werden können. Des Weiteren dient ein Objekt von diesem Typ der Identifizierung eines Gerätes für die Funktionen dieser Klasse.

**Eigenschaften**`string name`

Name des Geräts.

**Rückgabetyp**`bool isConnected`

Gibt an, ob eine Verbindung mit diesem Gerät besteht.

## Funktion - Init

---

Initialisiert die Klasse:

- Stellt eine Verbindung mit dem System D-Bus her.
- Stellt eine Verbindung mit dem User D-Bus her.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

**Funktionssignatur**`static void Init(void)`**Argumente**`void`**Rückgabetyp**`void`

## Funktion - Deinit

---

Deinitialisiert die Klasse:

- Trennt die Verbindung mit dem System D-Bus.
- Trennt die Verbindung mit dem User D-Bus.
- Gibt Variablen frei.

**Funktionssignatur**`static void Deinit(void)`**Argumente**`void`**Rückgabetyp**`void`



## Funktion - StartDiscovery

---

Startet eine Suche nach verfügbaren Bluetooth Geräten.

**Funktionssignatur**

`static void StartDiscovery(void(*CallBack)())`

**Argumente**

`void(*CallBack)()`

Functionpointer einer Funktion, die aufgerufen werden soll, wenn ein neues Gerät verfügbar ist.

**Rückgabetyp**

`void`

## Funktion - StopDiscovery

---

Stoppt die Suche nach verfügbaren Bluetooth Geräten.

**Funktionssignatur**

`static void StopDiscovery(void)`

**Argumente**

`void`

**Rückgabetyp**

`void`

## Funktion - Connect

---

Startet einen Versuch sich mit dem angegebenen Bluetooth Gerät zu verbinden.

**Funktionssignatur**

`static void Connect(Device device)`

**Argumente**

`Device device`

Gerät mit dem eine Verbindung hergestellt werden soll.

**Rückgabetyp**

`void`

## Funktion - Disconnect

---

Trennt das Gerät mit dem aktuell eine Verbindung besteht.

**Funktionssignatur**

`static void Disconnect(void)`

**Argumente**

`void`

**Rückgabetyp**

`void`



## Funktion - Update

---

Funktion die von der [Klasse - Player](#) aufgerufen wird, um die aktuellen Informationen zur Wiedergabe zu erhalten.

Funktionssignatur

`static Player::State* Update(void)`

Argumente                    `void`

Rückgabetyp                `Player::State*`                    Siehe [Klasse - Player](#).

## Funktion - Pairable

---

Legt fest, ob eine Verbindung von außen erlaubt ist.

Funktionssignatur

`static void Pairable(bool pairable)`

Argumente                    `bool pairable`                    Verbindung erlauben/verbieten.

Rückgabetyp                `void`

## Funktion - Visible

---

Legt fest, ob das Gerät durch ein anderes gesehen werden kann.

Funktionssignatur

`static void Visible(bool visible)`

Argumente                    `bool visible`                    Sichtbar/Unsichtbar.

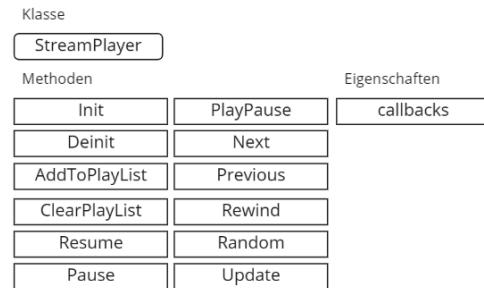
Rückgabetyp                `void`

---



### 8.2.3 Klasse - StreamPlayer

Ist für das Abspielen von Netzwerk-Streams und lokalen Dateien verantwortlich. Dafür wird das Programm [vlc](#) ([Quelle: 12.2.13](#)) verwendet, das über dessen Http-Interface ([Quelle: 12.2.3](#)) angesteuert wird. Diese Klasse bildet dabei eine Abstraktionsschicht zur Ansteuerung dieses Programms.



#### Eigenschaften

`Player::CallBacks callbacks`

Liste mit Functionpointer die an die [Klasse - Player](#) übergeben werden.

### Funktion - Init

Initialisiert die Klasse:

- Startet einen lokalen vlc-Server.
- Initialisiert Variablen die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

`static void Init(void)`

#### Argumente

`void`

#### Rückgabetyp

`void`

### Funktion - Deinit

Deinitialisiert die Klasse:

- Beendet den lokalen vlc-Server.
- Gibt Variablen frei.

#### Funktionssignatur

`static void Deinit(void)`

#### Argumente

`void`

#### Rückgabetyp

`void`

### Funktion - AddToPlayList

Fügt einen Stream oder Titel zur Playlist hinzu.

#### Funktionssignatur

`static void AddToPlayList(const char* mrl)`

#### Argumente

`const char* mrl`

Pfad zu einem Netzwerk-Stream oder einer lokalen Audiodatei.

#### Rückgabetyp

`void`



## Funktion - ClearPlayList

---

Leert die Wiedergabeliste und stoppt die Wiedergabe.

**Funktionssignatur**

**static void ClearPlayList(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

## Funktion - Resume

---

Setzt die Wiedergabe fort, wenn diese pausiert ist.

**Funktionssignatur**

**static void Resume(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

## Funktion - Pause

---

Pausiert die Wiedergabe, wenn sie nicht pausiert ist.

**Funktionssignatur**

**static void Pause(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

## Funktion - PlayPause

---

Setzt die Wiedergabe fort, wenn diese pausiert ist.

Pausiert die Wiedergabe, wenn sie nicht pausiert ist.

**Funktionssignatur**

**static void PlayPause(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**



## Funktion - Next

---

Überspringt den aktuellen Titel, und gibt den nächsten wieder.

**Funktionssignatur**

`static void Next(void)`

**Argumente**                    `void`

**Rückgabetyp**                    `void`

## Funktion - Previous

---

Überspringt den aktuellen Titel, und gibt den vorherigen wieder.

**Funktionssignatur**

`static void Previous(void)`

**Argumente**                    `void`

**Rückgabetyp**                    `void`

## Funktion - Rewind

---

Lässt den aktuellen Titel wiederholt abspielen.

**Funktionssignatur**

`static void Rewind(void)`

**Argumente**                    `void`

**Rückgabetyp**                    `void`

## Funktion - Random

---

Lässt den nächsten Titel zufällig auswählen.

**Funktionssignatur**

`static void Random(void)`

**Argumente**                    `void`

**Rückgabetyp**                    `void`



## Funktion - Update

---

Funktion die von der [Klasse - Player](#) aufgerufen wird, um die aktuellen Informationen zur Wiedergabe zu erhalten.

**Funktionssignatur**

**static Player::State\* Update(void)**

**Argumente**                   **void**

**Rückgabetyp**

**Player::State\***

Siehe [Klasse - Player](#).

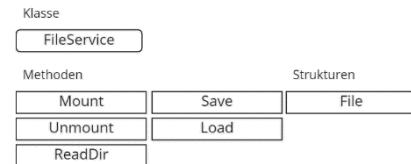
---



## 8.2.4 Klasse - FileService

---

Ist für das Speichern und Laden von Einstellungen und Dateien verantwortlich.  
Diese Klasse dient als Abstraktionsschicht zur Verwendung des Dateisystems.



### Struktur - File

---

Beinhaltet die wichtigsten Informationen zu einer Datei.

**Eigenschaften**            `string name`            Name der Datei.

`string path`            Absoluter Dateipfad.

`bool isFolder`            Ordner/Datei.

### Funktion - Mount

---

Mounted ein externes USB-Speichermedium, um auf dessen Inhalt zugreifen zu können.

**Funktionssignatur**

`static void Mount(void)`

**Argumente**            `void`

**Rückgabetyp**            `void`

### Funktion – Unmount

---

Unmounted ein externes USB-Speichermedium.

**Funktionssignatur**

`static void Unmount(void)`

**Argumente**            `void`

**Rückgabetyp**            `void`



## Funktion - ReadDir

---

Liest alle Dateien und Ordner an einem Dateipfad.

**Funktionssignatur**

`static vector<File> ReadDir(const char* path)`  
Argumente                    `const char* path`                    Dateipfad.

**Rückgabetyp**

`vector<File>`

Liste mit Dateien und Ordner.

## Funktion – Save

---

Speichert Informationen in eine Konfigurationsdatei.

**Funktionssignatur**

`template<typename Type>`

`static void Save(string key, Type data)`

Argumente

`string key`

String zur Identifizierung des Datensatzes.

`Type data`

Datensatz des Typs `Type`.

**Rückgabetyp**

`void`

## Funktion – Load

---

Lädt Informationen aus einer Konfigurationsdatei.

**Funktionssignatur**

`template <typename Type>`

`static Type Unmount(string key)`

Argumente

`String key`

String zur Identifizierung des Datensatzes.

**Rückgabetyp**

`Type`

Datensatz des Typs `Type`.

---



## 8.2.5 Klasse - Coprocessor

---

Ist für die Kommunikation mit dem Coprozessor zuständig.

Klasse	
Methoden	Enumerationen
Init	Input
Deinit	Output
SetInput	
SetOutput	
EnableAmplifier	
SetAttenuation	

### Enumeration - Input

---

Legt den Eingang für die Verstärkerschaltung fest.

Zustände                    **DAC**                    Systemeingang.

**AUDIO\_IN**                    Audioeingang.

### Enumeration - Output

---

Legt den Ausgang der Verstärkerschaltung fest.

Zustände                    **SPEAKER**                    Interne Lautsprecher.

**AUDIO\_OUT**                    Audioausgang.

### Funktion - Init

---

Initialisiert die Klasse:

- Stellt Kommunikation mit dem Arduino her.
- Initialisiert all Variablen die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

**static void Init(void)**

Argumente                    **void**

Rückgabetyp                    **void**



## Funktion – Deinit

---

Deinitialisiert die Klasse:

- Trennt die Verbindung mit dem Arduino.
- Gibt Variablen frei.

### Funktionssignatur

`static void Deinit(void)`

Argumente                    `void`

Rückgabetyp                    `void`

## Funktion – SetInput

---

Legt den Eingang für die Verstärkerschaltung fest.

### Funktionssignatur

`static void SetInput(Input input)`

Argumente                    `Input input`                    Eingang siehe [Enumeration - Input](#).

Rückgabetyp                    `void`

## Funktion – SetOutput

---

Legt den Ausgang für die Verstärkerschaltung fest.

### Funktionssignatur

`static void SetOutput(Output output)`

Argumente                    `Output output`                    Ausgang siehe [Enumeration - Output](#).

Rückgabetyp                    `void`

## Funktion – EnableAmplifier

---

Schaltet den Verstärker aus/ein.

### Funktionssignatur

`static void EnableAmplifier(bool enable)`

Argumente                    `bool enable`                    Ein/Aus.

Rückgabetyp                    `void`



## Funktion – SetAttenuation

---

Lädt Informationen aus einer Konfigurationsdatei.

**Funktionssignatur**

**static void Unmount(unsigned int attenuation)**  
**Argumente**                    **unsigned int attenuation**      Dämpungswert von 0 – 100%.

**Rückgabetyp**

**void**

---

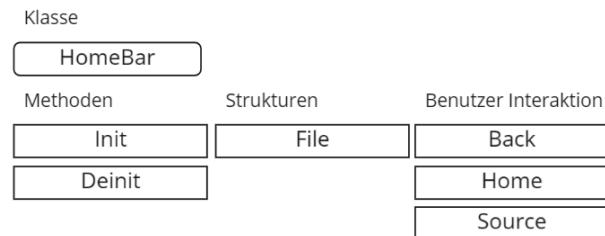


## 8.2.6 Klasse - HomeBar

Ist für einen Teil der Benutzeroberfläche verantwortlich, des es dem Benutzer erlaubt durch das Interface zu navigieren.



Abbildung 20: Benuteroberfläche - Navigationsleiste



<u>Benutzer Interaktion</u>	Back	Navigiert zur vorherigen Ansicht.
	Home	Navigiert auf die Startseite.
	Source	Navigiert zur Quellenauswahl.

### Funktion - Init

Initialisiert die Klasse:

- Erstellt das Steuerelement.

#### Funktionssignatur

`static void Init(void)`

<u>Argumente</u>	<code>void</code>
------------------	-------------------

<u>Rückgabetyp</u>	<code>void</code>
--------------------	-------------------

### Funktion – Deinit

Deinitialisiert die Klasse:

- Entfernt das Steuerelement.

#### Funktionssignatur

`static void Deinit(void)`

<u>Argumente</u>	<code>void</code>
------------------	-------------------

<u>Rückgabetyp</u>	<code>void</code>
--------------------	-------------------

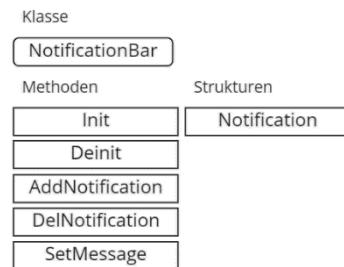


## 8.2.7 Klasse - NotificationBar

Ist für einen Teil der Benutzeroberfläche verantwortlich die Statusinformationen bereitstellt.



Abbildung 21: Benutzeroberfläche - Statusleiste



### Struktur - Notification

Beinhaltet alle Informationen über eine Statusmeldung. Des Weiteren dient ein Objekt von diesem Typ der Identifizierung eines Icons für die Funktionen dieser Klasse.

#### Eigenschaften

`const char* icon`

Typ des Icons nach LV\_SYMBOL\_[Type].

### Funktion - Init

Initialisiert die Klasse:

- Erstellt das Grafikelement.

#### Funktionssignatur

`static void Init(void)`

#### Argumente

`void`

#### Rückgabetyp

`void`

### Funktion - Deinit

Initialisiert die Klasse:

- Entfernt das Grafikelement.

#### Funktionssignatur

`static void Deinit(void)`

#### Argumente

`void`

#### Rückgabetyp

`void`



## Funktion – AddNotification

---

Fügt der Statusleiste eine Statusmeldung hinzu.

**Funktionssignatur**

**static void AddNotification(Notification\* notification)**

**Argumente**                    **Notification\*** **notification**    Statusmeldung.

**Rückgabetyp**

**void**

## Funktion - DelNotification

---

Entfernt eine Statusmeldung aus der Statusleiste.

**Funktionssignatur**

**static void DelNotification(Notification\* notification)**

**Argumente**                    **Notification\*** **notification**    Statusmeldung.

**Rückgabetyp**

**void**

## Funktion – SetMessage

---

Legt die in der Statusleiste anzuzeigende Nachricht fest.

**Funktionssignatur**

**static void SetMessage(const char\* message)**

**Argumente**                    **const char\*** **message**                 Statusmeldung.

**Rückgabetyp**

**void**

---



## 8.2.8 Klasse - Window

Ist für einen Teil der Benutzeroberfläche zur Navigation verantwortlich. Objekte dieser Klasse sind die Container, in denen alle anderen Steuer und Grafikelemente Platz finden. Sie erlauben es durch Fingergesten oder automatisiert zwischen Seiten (Window-Objekten) zu wechseln.



Benutzer Interaktion      **Swipe**

Zeigt das Window-Objekt das in der Richtung, von der die Bewegung der Geste ausgeht, mit diesem Objekt verknüpft ist. Hat keinen Effekt, wenn kein Objekt verknüpft ist.

Eigenschaften

**lv\_obj\_t\* container**

Lvgl-Container dieses Objekts. Besitzt einen Verweis auf dieses Objekt.

**lv\_obj\_t\* leftContainer**

Lvgl-Container des auf der linken Seite verknüpften Objektes. Besitzt einen Verweis auf dieses Objekt. Ist **nullptr** wenn nicht verknüpft.

**lv\_obj\_t\* rightContainer**

Lvgl-Container des auf der rechten Seite verknüpften Objektes. Besitzt einen Verweis auf dieses Objekt. Ist **nullptr** wenn nicht verknüpft.

## Enumeration - Direction

Gibt die Richtung der Animation an.

Zustände

**LEFT**

Animation nach links.

**RIGHT**

Animation nach rechts.

## Funktion - Show

Navigiert durch eine Animation auf eine andere Seite.

### Funktionssignatur

```
static void Show(Window* replace = nullptr, Direction direction = Direction::LEFT)
```

Argumente

**Window\* replace**

Zu ersetzendes Window-Objekt. Optional, wenn **direction** nicht verwendet wird (Default: Keine Animation).

Rückgabetyp

**Direction direction**

Richtung der Animation. Optional (Defaults: LEFT).



## 8.2.9 Klasse - Player

Ist für einen Teil der Benutzeroberfläche zur Steuerung der Musikwiedergabe zuständig. Diese Klasse dient als Abstraktionsschicht zur Steuerung aller Audioquellen.

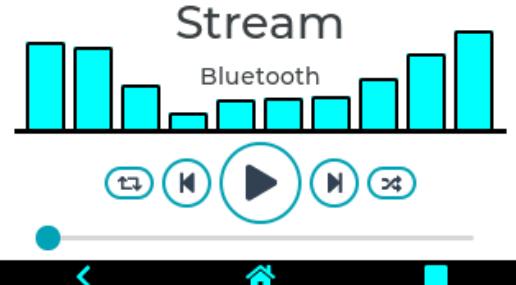
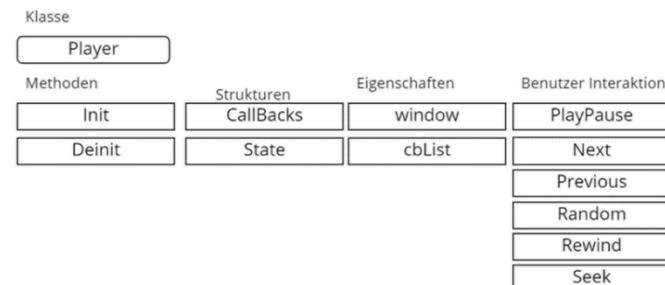


Abbildung 22: Benutzeroberfläche - Wiedergabe

Benutzer Interaktion      **PlayPause**

Pausiert die Wiedergabe, wenn sie nicht pausiert ist.  
Setzt die Wiedergabe fort, wenn sie pausiert ist.

**Next**

Überspringt den aktuellen Titel und gibt den nächsten wieder.

**Previous**

Überspringt den aktuellen Titel und gibt den vorherigen wieder.

**Random**

Lässt den nächsten Titel zufällig auswählen.

**Rewind**

Lässt den aktuellen Titel wiederholt Wiedergeben.

**Seek**

Springt an einen Zeitpunkt im Titel.

Eigenschaften

**Window\*** `window`

Window-Objekt.

**CallBacks** `cbList`

Liste mit Functionpointer mit Funktionen, welche aufgerufen werden, wenn eine der obigen Eingaben erfolgt. Die Funktionen werden von der Klasse festgelegt, die aktuell für die Wiedergabe zuständig ist.



## Struktur - CallBacks

---

Beinhaltet eine Liste aus Functionpointer welche aufgerufen werden wenn eine entsprechende Benutzereingabe vorgenommen wird..

<u>Eigenschaften</u>	<code>void(*playpause)()</code>	Functionpointer für eine <b>PlayPause</b> Benutzereingabe.
	<code>void(*next)()</code>	Functionpointer für eine <b>Next</b> Benutzereingabe.
	<code>void(*previous)()</code>	Functionpointer für eine <b>Previous</b> Benutzereingabe.
	<code>void(*rewind)()</code>	Functionpointer für eine <b>Rewind</b> Benutzereingabe.
	<code>void(*random)()</code>	Functionpointer für eine <b>Random</b> Benutzereingabe.
	<code>void(*seek)(int)</code>	Functionpointer für eine <b>Seek</b> Benutzereingabe.

## Struktur - State

---

Beinhaltet Einstellungen, die für das Erscheinungsbild des Players verantwortlich sind. Ein Objekt dieses Typs wird regelmäßig von der für die Wiedergabe verantwortlichen Klasse eingefordert.

<u>Eigenschaften</u>	<code>string title</code>	Name des aktuell gespielten Titels.
	<code>artist artist</code>	Künstler des aktuell gespielten Titels.
	<code>bool.isPlaying</code>	Gibt an ob die Wiedergabe pausiert ist.
	<code>bool.isRandom</code>	Gibt an ob der nächste Titel zufällig ausgewählt wird.
	<code>bool.isRewinding</code>	Gibt an ob der aktuell gespielte Titel wiederholt wird.
	<code>int position</code>	Gibt die position der Wiedergabe des Titels an.
	<code>bool.isStream</code>	Gibt an ob die oberfläche für einen Musikstream ohne die möglichkeit der benutzung einiger Eingaben dargestellt werden soll.



## Funktion - Init

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

**Funktionssignatur**

**static void Init(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

## Funktion - Deinit

---

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

**Funktionssignatur**

**static void Deinit(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

---



## 8.2.10 Klasse - Settings

Ist für einen Teil der Benutzeroberfläche zur Konfiguration des Radios verantwortlich.

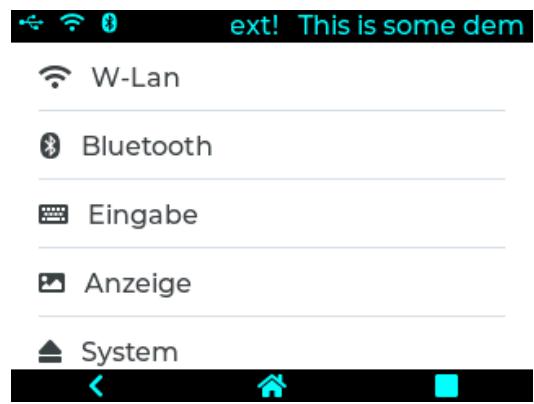
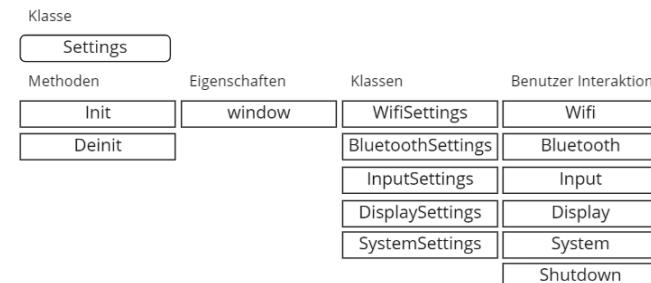


Abbildung 23: Benutzeroberfläche - Einstellungen

Benutzer Interaktion      `Wifi`

Zeigt die W-Lan-Einstellungen  
([Klasse - WifiSettings](#)).

`Bluetooth`

Zeigt die Bluetooth-Einstellungen  
([Klasse - BluetoothSettings](#)).

`Input`

Zeigt die Eingabe-Einstellungen  
([Klasse - InputSettings](#)).

`Display`

Zeigt die Anzeige-Einstellungen  
([Klasse - DisplaySettings](#)).

`System`

Zeigt die System-Einstellungen  
([Klasse - SystemSettings](#)).

`Shutdown`

Schaltet das Radio aus.

Eigenschaften      `Window* window`

Window-Objekt.

## Funktion - Init

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

### Funktionssignatur

`static void Init(void)`

Argumente      `void`

Rückgabetyp      `void`



## Funktion - Deinit

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

### Funktionssignatur

**static void Deinit(void)**

Argumente

**void**

Rückgabetyp

**void**

## 8.2.11 Klasse - Settings::SubSettings

Fasst die Initialisierung, Deinitialisierung und Eigenschaften aller SubSettings zusammen.

Klasse

Settings::SubSettings

Methoden

Init

Eigenschaften

window

Deinit

Eigenschaften

**Window\* window**

Window-Objekt.

## Funktion - Init

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

### Funktionssignatur

**static void Init(void)**

Argumente

**void**

Rückgabetyp

**void**



## Funktion - Deinit

---

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

### Funktionssignatur

**static void Deinit(void)**

#### Argumente

**void**

#### Rückgabetyp

**void**

---



## 8.2.12 Klasse - Settings::WifiSettings : SubSettings

Ist für einen Teil der Benutzeroberfläche zur Steuerung der Klasse - Wifi zuständig.

Klasse

Settings::WifiSettings : SubSettings		
Methoden	Klassen	Benutzer Interaktion
Init	Popup	Settings
Deinit		Scan
UpdateNetworks		Network



Abbildung 24: Benutzeroberfläche - W-Lan Einstellungen

Benutzer Interaktion      Settings

Zeigt die Einstellungen Klasse - Settings.

Scan

Startet Klasse - Wifi::Scan.

Network

Startet Klasse - Wifi::Connect oder

Startet Klasse - Wifi::Disconnect oder  
Zeigt Popup Klasse -  
Settings::WifiSettings::Popup.

## Funktion - Init

Initialisiert die Klasse:

- Ruft Init der Basisklasse SubSettings auf
- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur

`static void Init(void)`

Argumente                  `void`

Rückgabetyp                  `void`



## Funktion - Deinit

---

Initialisiert die Klasse:

- Ruft Dinit der Basisklasse auf
- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

**Funktionssignatur**

**static void Deinit(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

## Funktion - UpdateNetworkList

---

Funktion die aufgerufen wird, wenn **Klasse - Wifi::Scan** beendet ist.

**Funktionssignatur**

**static void UpdateNetworkList(void)**

**Argumente**                   **void**

**Rückgabetyp**                   **void**

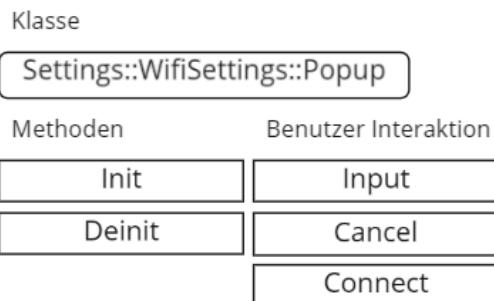


### 8.2.13 Klasse - Settings::WifiSettings::Popup

Ist für einen Teil der Benutzeroberfläche, der für das Einsammeln von W-Lan-Passwörtern verantwortlich ist.



Abbildung 25: Benutzeroberfläche - Passwort-Feld



#### Benutzer Interaktion

##### Input

Stellt eine Tastatur zur Eingabe bereit.

##### Cancel

Zeigt Klass - Settings::WifiSettings.

##### Connect

Startet Klass - Wifi::Connect oder

Startet Klass - Wifi::Disconnect oder  
Zeigt Popup Klass -  
Settings::WifiSettings::Popup.

### Funktion - Init

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.

#### Funktionssignatur

```
static void Init(void)
```

#### Argumente

void

#### Rückgabetyp

void

### Funktion - Deinit

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.

#### Funktionssignatur

```
static void Deinit(void)
```

#### Argumente

void

#### Rückgabetyp

void



### 8.2.14 Klasse - Settings::BluetoothSettings : SubSettings

---

Ist für einen Teil der Benutzeroberfläche zur Ansteuerung von **Klasse - Bluetooth** verantwortlich.

Klasse

Settings::BluetoothSettings : SubSettings

Methoden

Benutzer Interaktion

Init	Settings
Deinit	Discover
UpdateDevices	Device

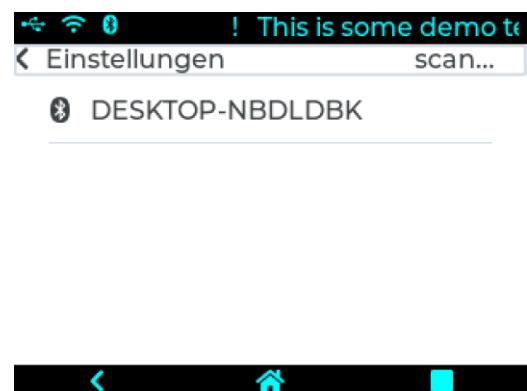


Abbildung 26: Benutzeroberfläche Bluetooth Einstellungen

**Benutzer Interaktion**

Settings

Zeigt die Einstellungen **Klasse - Settings**.

Discover

Startet **Klasse - Bluetooth::Discover**.

Device

Startet **Klasse - Bluetooth::Connect** oder  
Startet **Klasse - Bluetooth::Disconnect**.

### Funktion - Init

---

Initialisiert die Klasse:

- Ruft Init der Basisklasse SubSettings auf
- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

**Funktionssignatur**

**static void Init(void)**

**Argumente**

**void**

**Rückgabetyp**

**void**

### Funktion - Deinit

---

Initialisiert die Klasse:

- Ruft Dinit der Basisklasse auf
- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

**Funktionssignatur**

**static void Deinit(void)**



Argumente void

Rückgabetyp void

### Funktion - UpdateDeviceList

---

Funktion die aufgerufen wird, wenn Klasse - Bluetooth::Scan ein neues Gerät entdeckt hat.

Funktionssignatur

static void UpdateDeviceList(void)

Argumente void

Rückgabetyp void

---



## 8.2.15 Klasse - Settings::InputSettings : SubSettings

Ist für einen Teil der Benutzeroberfläche, die für Eingabe-Einstellungen verantwortlich ist.

Klasse

Settings::InputSettings : SubSettings

Methoden

Benutzer Interaktion

Init

Settings

Deinit

SwipeThreshold

ShowHomeBar

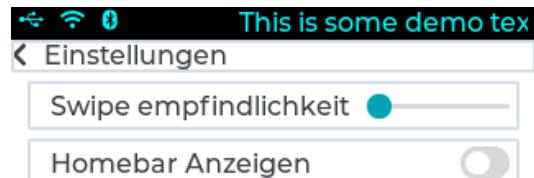


Abbildung 27: Benutzeroberfläche - Eingabeeinstellungen

Benutzer Interaktion      Settings

Zeigt die Einstellungen Klasse – Settings.

SwipeThreshold

Legt die Swipe-Empfindlichkeit zum Navigieren zwischen Seiten fest.

ShowHomeBar

Legt fest, ob das HomeBar-Element angezeigt werden soll.

### Funktion - Init

Initialisiert die Klasse:

- Ruft Init der Basisklasse SubSettings auf
- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

static void Init(void)

Argumente      void

Rückgabetyp      void

### Funktion - Deinit

Initialisiert die Klasse:

- Ruft Dinit der Basisklasse auf
- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

#### Funktionssignatur

static void Deinit(void)

Argumente      void

Rückgabetyp`void`

## 8.2.16 Klasse - Settings::DisplaySettings : SubSettings

Ist für einen Teil der Benutzeroberfläche, die für Anzeige-Einstellungen verantwortlich ist.

Klasse

`Settings::DisplaySettings : SubSettings`

Methoden

Benutzer Interaktion

`Init``Settings``Deinit``AccentColor``ShowNotificationBar`

Abbildung 28: Benutzeroberfläche - Anzeigeeinstellungen

Benutzer Interaktion`Settings`

Zeigt die Einstellungen Klasse – Settings.

`AccentColor`

Legt die Akzentfarbe fest.

`ShowNotificationBar`

Legt fest, ob das Statusleisten-Element angezeigt werden soll.

## Funktion - Init

Initialisiert die Klasse:

- Ruft Init der Basisklasse SubSettings auf
- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur`static void Init(void)`Argumente`void`Rückgabetyp`void`

## Funktion - Deinit

Initialisiert die Klasse:

- Ruft Dinit der Basisklasse auf
- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

Funktionssignatur`static void Deinit(void)`Argumente`void`



Rückgabetyp void

### 8.2.17 Klasse - Settings::DisplaySettings::Popup

---

Ist ein Teil der Benutzeroberfläche, der für das Sammeln von Farbeingaben verantwortlich ist.



Klasse

Settings::DisplaySettings::Popup

Methoden

Benutzer Interaktion

Init

Apply

Deinit

Abbildung 29:  
Benutzeroberfläche -  
Farbauswahl

Benutzer Interaktion Apply

Legt die Akzentfarbe fest.

#### Funktion - Init

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.

##### Funktionssignatur

static void Init(void)

Argumente void

Rückgabetyp void

#### Funktion - Deinit

---

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.

##### Funktionssignatur

static void Deinit(void)

Argumente void

Rückgabetyp void

---

---



## 8.2.18 Klasse - Settings::SystemSettings : SubSettings

Ist für einen Teil der Benutzeroberfläche, die für Anzeige-Einstellungen verantwortlich ist.

Klasse

Settings::SystemSettings : SubSettings

Methoden

Benutzer Interaktion

Init

Settings

Deinit

Language

Information

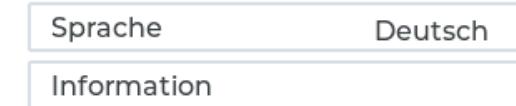


Abbildung 30: Benutzeroberfläche - Systemeinstellungen

Benutzer Interaktion

Settings

Zeigt die Einstellungen [Klasse – Settings](#).

Language

Legt die Systemsprache.

Inforamtion

Zeigt informationen über das Radio an.

### Funktion - Init

Initialisiert die Klasse:

- Ruft Init der Basisklasse SubSettings auf
- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

`static void Init(void)`

Argumente

`void`

Rückgabetyp

`void`

### Funktion - Deinit

Initialisiert die Klasse:

- Ruft Dinit der Basisklasse auf
- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

#### Funktionssignatur

`static void Deinit(void)`

Argumente

`void`

Rückgabetyp

`void`



### 8.2.19 Klasse - Sources

Ist für einen Teil der Benutzeroberfläche zur Auswahl der Musikquelle verantwortlich.

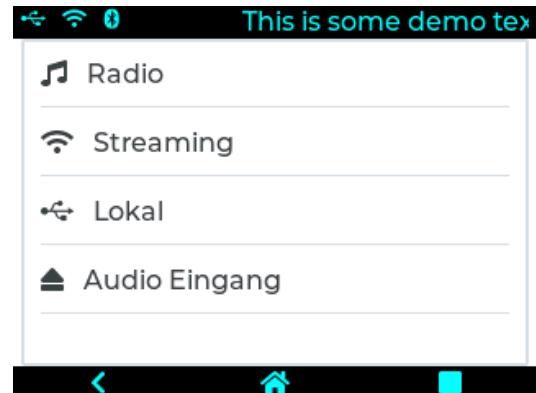
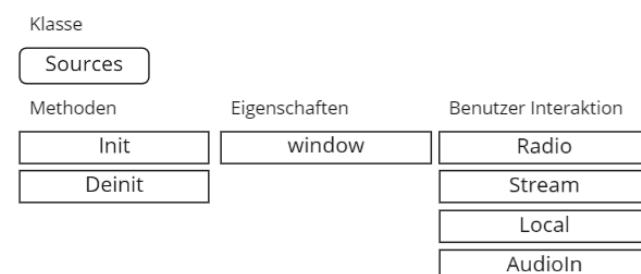


Abbildung 31: Benutzeroberfläche - Quellenauswahl

#### Benutzer Interaktion      Radio

Zeigt die W-Lan-Einstellungen  
([Klasse - RadioSource](#)).

#### Stream

Zeigt die W-Lan-Einstellungen  
([Klasse - StreamSource](#)).

#### Local

Zeigt die W-Lan-Einstellungen  
([Klasse - LocalSource](#)).

#### AudioIn

Die Audioeingangsbuchse wird als Quelle verwendet.

#### Eigenschaften      Window\* window

Window-Objekt.

### Funktion - Init

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

**static void Init(void)**

#### Argumente      void

#### Rückgabetyp      void



## Funktion - Deinit

---

Initialisiert die Klasse:

- Entfernt Steuer- und Grafikelemente.
- Gibt Variablen frei.

### Funktionssignatur

**static void Deinit(void)**

Argumente                   **void**

Rückgabetyp                   **void**



## 8.2.20 Klasse - Sources::SubSources

---

Fasst die Initialisierung, Deinitialisierung und Eigenschaften aller SubSources zusammen.

Klasse

Sources::SubSources

Methoden

Init

Eigenschaften

window

Deinit

Eigenschaften

Window\* window

Window Objekt.

### Funktion - Init

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur

static void Init(void)

Argumente

void

Rückgabetyp

void

### Funktion - Deinit

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur

static void Deinit(void)

Argumente

void

Rückgabetyp

void

---



## 8.2.21 Klasse - Sources::RadioSource : SubSources

---

Ist für einen Teil der Benutzeroberfläche zur Auswahl eines FM-Radiosenders als Quelle verantwortlich.

Klasse

Sources::RadioSource : SubSources

Methoden

Init

Deinit

### Benutzer Interaktion

### Funktion - Init

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

static void Init(void)

Argumente                    void

Rückgabetyp                    void

### Funktion - Deinit

---

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

static void Deinit(void)

Argumente                    void

Rückgabetyp                    void

---



## 8.2.22 Klasse - Sources::StreamSource : SubSources

Ist für einen Teil der Benutzeroberfläche in dem eine Quelle zum Streamen ausgewählt werden kann verantwortlich.

Klasse

Sources::StreamSource : SubSources

Methoden

Benutzer Interaktion

Init	Sources
Deinit	Stream

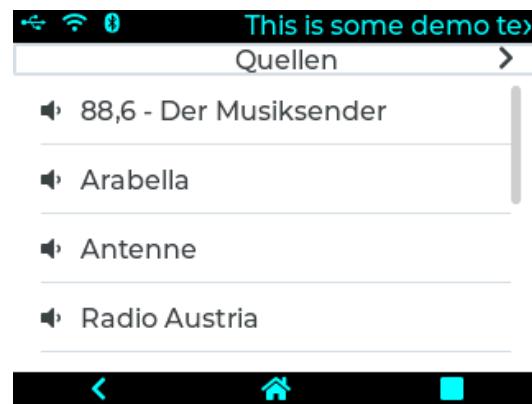


Abbildung 32: Benutzeroberfläche - RadioStreams

Benutzer Interaktion      Sources

Zeigt die Quellenauswahl (Klasse - Sources).

Stream

Spielt einen Stream aus der Liste ab.

### Funktion - Init

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur

static void Init(void)

Argumente      void

Rückgabetyp      void

### Funktion - Deinit

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

Funktionssignatur

static void Deinit(void)

Argumente      void

Rückgabetyp      void



## 8.2.23 Klasse - Sources::LocalSources : SubSources

Ist für einen Teil der Oberfläche zum Auswählen einer Quelle aus den Dateien eines USB-Speichermediums verantwortlich.

Klasse

Sources::LocalSource : SubSources

Methoden	Benutzer Interaktion
Init	Sources
Deinit	Back
	Queue
	Folder



Bang Bang.mp3

Club Rocker.mp3

Hinterland.mp3

Playlist

Abbildung 33: Benutzeroberfläche - LokaleDateien

Benutzer Interaktion

Sources

Zeigt die Quellenauswahl ([Klasse - StreamPlayer::AddToPlayList](#)).

Back

Navigiert in den übergeordneten Ordner.

Queue

Fügt eine Datei oder alle Dateien in einem Ordner der Playlist hinzu ([Klasse - StreamPlayer::AddToPlayList](#)) und Startet die Wiedergabe.

Folder

Navigiert in den ausgeählten Ordner.

### Funktion - Init

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

`static void Init(void)`

Argumente

`void`

Rückgabetyp

`void`

### Funktion - Deinit

Initialisiert die Klasse:

- Erstellt Steuer- und Grafikelemente.
- Initialisiert Variablen, die für die Funktionalität der Klasse benötigt werden.

#### Funktionssignatur

`static void Deinit(void)`

Argumente

`void`

Rückgabetyp

`void`



### 8.3 D-Bus & XMLHttpRequest

Die Kommunikation mit C++ über D-Bus wird durch die Bibliothek sd-bus ([Quelle: 12.2.6](#)) ermöglicht.

Die Kommunikation zwischen C++ und vlc wird mit curl und libxml ([Quelle: 12.2.7](#)) ermöglicht.



## 8.4 Raspberry Pi einrichten

1. Raspberry Pi OS Lite auf eine SD-Karte schreiben. ([Quelle: 12.2.16](#))
  - a. Wenn das „boot“ Laufwerk nicht am PC erscheint, dann den Kartenleser aus- und wieder einstecken.
2. Im Dateiexplorer in das „boot“ Laufwerk navigieren.
3. W-LAN Verbindung konfigurieren ([Quelle 12.2.17](#))
  - a. Dazu eine Textdatei mit dem Namen „wpa\_supplicant.conf“ im Hauptverzeichnis des Laufwerks erstellen.
  - b. Inhalt nach dieser Anleitung ([Quelle:](#)) mit dem Country-Code „AT“ und den W-Lan Daten füllen.
4. SSH konfigurieren: Leere Datei mit dem Namen „ssh“ (Ohne Dateierweiterung) im Hauptverzeichnis des Laufwerks erstellen.
5. Raspberry Pi mit der SD-Karte starten.
6. Am PC auf der Konsole mit „ssh pi@<IP-Adresse hier>“ eine SSH-Verbindung aufbauen (Beim Fingerprint mit "yes" bestätigen).
  - a. IP-Adresse kann über den Router oder mit NMAP herausgefunden werden.
7. Mit dem Passwort raspberrypi anmelden.
8. Über SSH am Pi mit sudo apt update && sudo apt upgrade alle Packages aktualisieren (mit y+Enter bestätigen)
9. Pi Konfigurieren, dazu sudo raspi-config ausführen
  - a. Interface Options: SPI und I2C aktiviere
  - b. Passwort ändern
10. esp-radio-pc repository clonen
  - a. git mit sudo apt install git installieren
  - b. git clone --recursive <https://github.com/dereisl/esp32-radio-pc.git>
11. Display & Touch Treiber installieren
  - a. In den esp-radio-pc Ordner gehen: cd esp32-radio-pc
  - b. In den raspi Order darin gehen: cd raspi
  - c. Installer Script ausführen: sudo ./rpi\_setup.sh
12. Pi ausschalten: sudo shutdown now
13. Pi vom Strom trennen
14. Pi an Schaltung anschließen
15. Versorgung anschließen

### 8.4.1 Installer Script

Das Installer Script kompiliert und installiert ein Device Tree Overlay (aktiviert Kerneltreiber) für den kapazitiven Touchscreen ([Quelle: 12.2.18](#)). Außerdem lädt es den Userspace-Treiber für den Bildschirm ([Quelle: 12.2.19](#)) herunter, kompiliert ihn und erstellt eine systemd unit Datei, damit er automatisch startet.

Auch werden vom Script in der /boot/config.txt die notwendigen Peripheriegeräte und das oben genannte Overlay aktiviert und die Bildschirmauflösung festgelegt.



## 8.5 Kompilieren

Mit dem LVGL-Simulator kann das UI auch am PC entwickelt werden, um Zeit zu sparen.

Zum Kompilieren in den Ordner esp32-radio-pc navigieren und folgenden Befehl ausführen:  
cmake . && make

## 8.6 Ausführen

Einfaches

make

ausführen:

run

Um ein USB-Speichermedium mounten zu können werden sudo rechte benötigt:  
sudo make run



## 8.7 ESP32 Fehlschläge

Der ursprüngliche Plan war es, statt einem Raspberry Pi Zero mit Linux, einen ESP32 mit dem ESP-ADF zu benutzen. Die Gründe dafür waren der niedrigere Stromverbrauch und die schnellere Startzeit. Eine Probesoftware wurde dafür auch geschrieben, dabei wurden jedoch verschiedene Probleme entdeckt. Die UI Teile dieser Software konnten aufgrund der plattformübergreifenden Natur von LVGL teilweise übernommen werden, der Rest musste jedoch von Grund auf neu geschrieben werden.

### 8.7.1 Probleme

Um Bluetooth zu aktivieren, war interner RAM notwendig, also wurde der Framebuffer für das Display auf den externen RAM verschoben, danach kam es jedoch zu starkem Stottern, sobald sich am Display etwas verändert hat. Die Vermutung ist, dass die Bandbreite am SPI Bus einfach nicht ausreichte.

Außerdem gab es im ESP-ADF einige Bugs in nicht quelloffenen Teilen des Frameworks, die unter anderem zu Abstürzen führten, wenn zu schnell vor- und zurückgespult wurde.

### 8.7.2 Lösung

Da ein Internetradio so nicht realisiert werden kann, musste auf einen Raspberry Pi als „nächstbessere“ Lösung umgestiegen werden.

Wegen der notwendigen Neuprogrammierung und den anderen nun zu lösenden Problemen, die durch den Umstieg auf den Raspberry Pi entstanden sind (z.B. der Coprozessor, um den Standby-Stromverbrauch zu senken), musste der bisherige Zeitplan verworfen werden.



## 9 Testbetrieb mit Prototyp am Steckbrett

## 9.1 Steckbrettprototyp

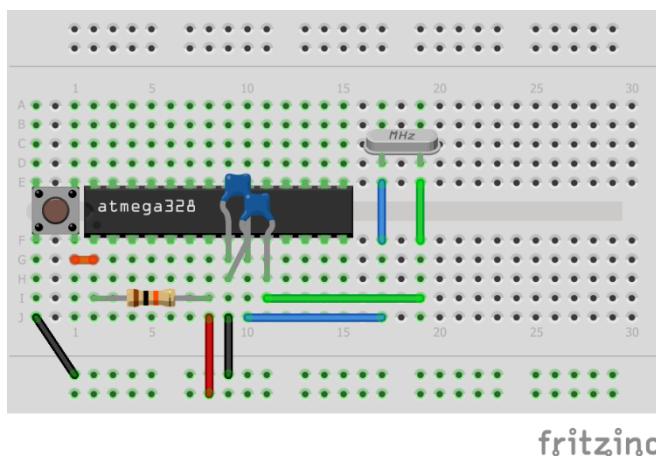
### 9.1.1 Raspberry

Bevor die Fertigung der Platine eingeleitet wurde, wurden alle Funktionen, die ohne die Platine funktionieren getestet. Dazu wurden ein Steckbrett und Jumper Kabel eingesetzt und der Pi wurde mit dem Display, dem DAC, dem Infrarotempfänger und dem Coprozessor (ebenfalls auf einem Steckbrett, siehe unten) zusammen getestet. Die Pinbelegung dabei ist die aus dem finalen Schaltplan. Dieser Aufbau wurde auch während der Entwicklung vom Großteil der Software benutzt.

Die ICs, von denen es keine Entwicklungsplatine gibt (Multiplexer, Attenuator und Verstärker), konnten erst auf der gefertigten Platine getestet werden, siehe dazu die Sektion [Testbetrieb mit gefertigter Platine](#).

## 9.1.2 ATmega328P

Ein ATmega328P IC wird auf dem Steckbrett getestet, der Schaltplan orientiert sich an dem finalen. Auch die Rotary Encoder und das FM Modul sind an diesem Aufbau getestet worden (nicht abgebildet, um das Bild übersichtlicher zu gestalten).



*Abbildung 34: Steckbrett Prototyp*

Dieses Steckbrettabbild wurde erstellt mit dem Programm Fritzing ([Quelle 12.3.6](#)).



## 10 Testbetrieb mit gefertigter Platine

### 10.1 Brandschutz

Um sicherzustellen, dass alle Teile der Schaltung funktionieren, die Funktionalität nicht an einer unbekannten Ursache scheitert und keine Teile der Schaltung durch andere beschädigt werden, wurden alle funktionellen Gruppen nacheinander eingebaut und getestet.

### 10.2 Zusammenbau

Die Platine wurde von JLCPCB gefertigt (siehe [Sektion Planung und Fertigung](#)).

Die kleinen ICs wurden mit einer Technik namens Schleplöten („drag soldering“ auf Englisch) eingelötet ([Quelle: 12.1.16 Video](#)).

### 10.3 Tests

#### 10.3.1 Versorgung, Display, DAC und Raspberry Pi

Der erste, realisierte Schaltungsteil ist die Spannungsversorgung. Dafür wurden der Spannungsregler und die Transistoren eingelötet. Mit einem Multimeter wurde die Versorgung auf ihre Werte überprüft.

Beim Aufbau der Spannungsversorgung ist aufgefallen, dass die eingeplante Grundfläche für zwei Elektrolytkondensatoren zu klein ist.

Da noch kein ATmega Chip eingebaut wurde, der die Steuerung der 5V Versorgung kontrolliert, mussten die Transistoren überbrückt werden, um im nächsten Zug die Funktionalität des DAC-Moduls zu gewährleisten.

Beim Einbau der Buchsenleisten für das DAC-Modul ist aufgefallen, dass die Löcher, in die dieses montiert werden soll, falsche Abstände haben.



### 10.3.2 Eingangsmultiplexer

Als nächstes wurden die Multiplexer getestet. Mit einem zu Testzwecken eingebauten Audiostecker am Ausgang und dem Signal des DACs am Eingang wird überprüft ob der Multiplexer richtig funktioniert. Zum Umschalten werden folgende Befehle am Raspberry Pi ausgeführt:

```
1. echo 1 > /sys/class/gpio/export
2. echo 7 > /sys/class/gpio/export
3. echo 17 > /sys/class/gpio/export
4. sleep 2
5. echo out > /sys/class/gpio/gpio1/direction
6. echo out > /sys/class/gpio/gpio7/direction
7. echo out > /sys/class/gpio/gpio17/direction
8.
9. echo 0 > /sys/class/gpio/gpio1/value
10. echo 0 > /sys/class/gpio/gpio7/value
11. echo 0 > /sys/class/gpio/gpio17/value
```

### 10.3.3 Attenuator und Ausgangsmultiplexer

Um den Attenuator zu überprüfen ist bereits der ATmega-Chip notwendig. Da dieser noch nicht eingebaut werden sollte, hat er seine Aufgabe mit Hilfe von Jumper-Kabeln von einem Steckbrett aus erledigt.

Damit die Dämpfung des Signals überprüft werden kann, wird das Signal durch den Ausgangsmultiplexer an einen Kopfhöreranschluss weitergeleitet und während des Betriebs mit unterschiedlicher Stärke gedämpft.

### 10.3.4 ATmega328P

Beim fehlgeschlagenen Versuch Firmware auf den ATmega, der sich bereits auf dem Board befand, zu flashen ist aufgefallen, dass dieser verdreht eingelötet wurde.

Der Chip wurde ausgetauscht und hat glücklicherweise keinen Schaden an der restlichen Schaltung verursacht.

### 10.3.5 Power on / off Transistor

Mit dem einsatzbereiten ATmega-Chip am Board kann nun versucht werden die Versorgung aus- und einzuschalten. Wenn die Schaltung wie vorgesehen versorgt wird, bricht diese in sehr kurzen Abständen zusammen.

Dieses Problem konnte durch einen außerplan eingebauten Pull-Up Widerstand an der Steuerleitung des Transistors gelöst werden.



### 10.3.6 Verstärker

Beim ersten Versuch die Verstärkerschaltung zu testen war der „Not-Shutdown“ Pin durch den ATmega auf LOW geschalten, weshalb dieser auch gescheitert ist.

Dieses Problem wurde beim zweiten Versuch durch eine Neuprogrammierung behoben. Mit provisorisch verbundenen Lautsprechern konnte die Funktionalität des letzten Schaltungsteils bestätigt und ein großer Fehler entdeckt werden.

Beim Abspielen tritt ein Knacken, und sobald die Schaltung versorgt wird ein Piepsen auf. Als Ursache dieses, bis zum Ende des Projekts nicht behobenen, Fehlers wird eine Störung des analogen Audiosignals durch den ATmega, das Display oder den Raspberry über die Versorgung angenommen.

## 11 Bedienungsanleitung

### 11.1 Stromversorgung

Das Radio wird mit einem 5V Netzteil betrieben, das über die Stromversorgungsbuche an der Rückseite angeschlossen wird.

### 11.2 Aus- und Einschalten

Nach einem Anschluss an die Stromversorgung, startet das Radio automatisch.

Das Radio kann in der Software unter Einstellungen > Ausschalten ausgeschalten werden. Das Ausschalten durch eine Trennung der Versorgung sollte vermieden werden.

Das Radio kann durch das Drücken einer der Drehknöpfe wieder eingeschaltet werden.

### 11.3 Navigation im User Interface

Durch die Eingabe von Gesten kann in der Benutzeroberfläche navigiert werden. Die drei Schaltflächen am unteren Rand des Interfaces haben die Funktionen zurück, zur Startseite und zur Quellenauswahl, in dieser Reihenfolge von links nach rechts.

Wahlweise ist dies auch durch das Drehen des rechten Drehknopfs möglich. Durch Hineindrücken wird eine Auswahl bestätigt.

### 11.4 Lautstärke ändern.

Durch Drehen des linken Drehknopfs können Änderungen an der Lautstärke vorgenommen werden.

### 11.5 Musik vom Audio-Eingang abspielen.

Zusätzlich zum Verbinden einer Audioquelle muss der Audioeingang unter Quellen > Audioeingang als Quelle ausgewählt werden.

### 11.6 Musik am Audio-Ausgang ausgeben.

Zusätzlich zum Verbinden von Lautsprechern muss der Audioausgang unter Einstellungen > System > Ausgabe als Ausgabegerät ausgewählt werden.



## 11.7 Musik von einem USB-Speicher

An der Rückseite kann ein USB-Speichermedium angeschlossen werden. Um Musik von diesem abzuspielen muss unter Quellen > Lokal eine Musikdatei oder ein Ordner ausgewählt werden.

## 12 Quellenverzeichnis

Der gesamte Quellcode dieses Projekts und alle für die Fertigung notwendigen Dateien sind auf GitHub in einer Organisation veröffentlicht: <https://github.com/bulme-internetradio>

### 12.1 Hardware

	Bezeichnung	Quelle
12.1.1	TMUX1309DYYR	<a href="https://www.ti.com/lit/ds/symlink/tmux1309.pdf">https://www.ti.com/lit/ds/symlink/tmux1309.pdf</a>
12.1.2	LM1972MX/NOPB	<a href="https://www.ti.com/lit/ds/symlink/lm1972.pdf">https://www.ti.com/lit/ds/symlink/lm1972.pdf</a>
12.1.3	TPA3144D2PWP	<a href="https://www.ti.com/lit/ds/symlink/tpa3144d2.pdf">https://www.ti.com/lit/ds/symlink/tpa3144d2.pdf</a>
12.1.4	SI2323CDS-T1-GE3	<a href="https://www.vishay.com/docs/65700/si2323cds.pdf">https://www.vishay.com/docs/65700/si2323cds.pdf</a>
12.1.5	BC850BLT1G	<a href="https://www.onsemi.com/pdf/datasheet/bc846alt1-d.pdf">https://www.onsemi.com/pdf/datasheet/bc846alt1-d.pdf</a>
12.1.6	LD1117AS33TR	<a href="https://www.st.com/content/ccc/resource/technical/document/datasheet/a5/c3/c9/2b/15/40/49/CD00002116.pdf/files/CD00002116.pdf/jcr:content/translations/en.CD00002116.pdf">https://www.st.com/content/ccc/resource/technical/document/datasheet/a5/c3/c9/2b/15/40/49/CD00002116.pdf/files/CD00002116.pdf/jcr:content/translations/en.CD00002116.pdf</a>
12.1.7	PEC11R-4215F-S0024	<a href="https://www.bourns.com/docs/Product-Datasheets/PEC11R.pdf">https://www.bourns.com/docs/Product-Datasheets/PEC11R.pdf</a>
12.1.8	TSOP4438	<a href="https://www.vishay.com/docs/82459/tsop48.pdf">https://www.vishay.com/docs/82459/tsop48.pdf</a>
12.1.9	Raspberry Pi	<a href="https://www.raspberrypi.org/products/raspberry-pi-zero-w/">https://www.raspberrypi.org/products/raspberry-pi-zero-w/</a>
12.1.10	ER-TFTM032-3	<a href="https://www.buydisplay.com/download/manual/ER-TFTM032-3_Datasheet.pdf">https://www.buydisplay.com/download/manual/ER-TFTM032-3_Datasheet.pdf</a>
12.1.11	ATMEGA328P-PU	<a href="https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf">https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf</a>
12.1.12	SI4703 Basic	<a href="https://www.sparkfun.com/datasheets/BreakoutBoards/Si4702-03-C19-1.pdf">https://www.sparkfun.com/datasheets/BreakoutBoards/Si4702-03-C19-1.pdf</a>
12.1.13	FRS 7 W 8 OHM	<a href="http://www.visatonshop.at/Chassis-Zubehoer/Breitband-Systeme/FRS-7-W-8-OHM.html">http://www.visatonshop.at/Chassis-Zubehoer/Breitband-Systeme/FRS-7-W-8-OHM.html</a>
12.1.14	Arduino Nano CH340G 3.3 Volt conversion	<a href="https://forum.arduino.cc/index.php?topic=409415.0">https://forum.arduino.cc/index.php?topic=409415.0</a>
12.1.15	ESP32 ArduinoISP	<a href="https://www.youtube.com/watch?v=06Kj1ALprGw">https://www.youtube.com/watch?v=06Kj1ALprGw</a>
12.1.16	Schleplöten	<a href="https://youtu.be/wUyetZ5RtPs">https://youtu.be/wUyetZ5RtPs</a>



## 12.2 Software

	Bezeichnung	Quelle
12.2.1	wpa_supplicant D-Bus-Interface Dokumentation	<a href="https://w1.fi/wpa_supplicant-devel/dbus.html">https://w1.fi/wpa_supplicant-devel/dbus.html</a>
12.2.2	bluez D-Bus-Interface Dokumentation	<a href="https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc">https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc</a>
12.2.3	mpris D-Bus-Interface Dokumentation	<a href="https://specifications.freedesktop.org/mpris-spec/latest/">https://specifications.freedesktop.org/mpris-spec/latest/</a>
12.2.4	vlc Http-Interface Dokumentation	<a href="https://wiki.videolan.org/VLC_HTTP_requests/">https://wiki.videolan.org/VLC_HTTP_requests/</a>
12.2.5	Lvgl Dokumentation	<a href="https://docs.lvgl.io/latest/en/html/index.html">https://docs.lvgl.io/latest/en/html/index.html</a>
12.2.6	sd-bus Dokumentation	<a href="https://www.freedesktop.org/software/systemd/man/sd-bus.html">https://www.freedesktop.org/software/systemd/man/sd-bus.html</a>
12.2.7	Libxml Dokumentation	<a href="https://libxmlplusplus.github.io/libxmlplusplus/">https://libxmlplusplus.github.io/libxmlplusplus/</a>
12.2.8	Softwars Internetradio -GitHub Repository	<a href="https://github.com/dereisl/esp32-radio">https://github.com/dereisl/esp32-radio</a>
12.2.9	LVGL-Beispiel	<a href="https://github.com/lvgl/lv_sim_eclipse_sdl">https://github.com/lvgl/lv_sim_eclipse_sdl</a>
12.2.10	wpa_supplicant	<a href="https://wiki.archlinux.org/index.php/wpa_supplicant">https://wiki.archlinux.org/index.php/wpa_supplicant</a>
12.2.11	bluez	<a href="https://wiki.archlinux.org/index.php/bluetooth">https://wiki.archlinux.org/index.php/bluetooth</a>
12.2.12	mpris	<a href="https://wiki.archlinux.org/index.php/MPRIS">https://wiki.archlinux.org/index.php/MPRIS</a>
12.2.13	vlc	<a href="https://wiki.archlinux.org/index.php/VLC_media_player">https://wiki.archlinux.org/index.php/VLC_media_player</a>
12.2.14	Si4702 Arduino Library	<a href="https://github.com/mathertel/Radio">https://github.com/mathertel/Radio</a>
12.2.15	LIRC Tutorial	<a href="https://tutorials-raspberrypi.de/raspberry-pi-ir-remote-control/">https://tutorials-raspberrypi.de/raspberry-pi-ir-remote-control/</a>
12.2.16	Raspberry Pi Software	<a href="https://www.raspberrypi.org/software/">https://www.raspberrypi.org/software/</a>
12.2.17	Setting up a Raspberry Pi headless	<a href="https://www.raspberrypi.org/documentation/configuration/wireless/headless.md">https://www.raspberrypi.org/documentation/configuration/wireless/headless.md</a>
12.2.18	Raspberry Touchscreen Treiber	<a href="https://cdn-learn.adafruit.com/downloads/pdf/adafruit-2-8-pitft-capacitive-touch.pdf">https://cdn-learn.adafruit.com/downloads/pdf/adafruit-2-8-pitft-capacitive-touch.pdf</a> & <a href="https://github.com/pimoroni/hyperpixel4/blob/pi3/install.sh">https://github.com/pimoroni/hyperpixel4/blob/pi3/install.sh</a>
12.2.19	Raspberry Bildschirmtreiber	<a href="https://github.com/juj/fbcp ili9341">https://github.com/juj/fbcp ili9341</a>
12.2.20	AdaEncoder Arduino Library	<a href="https://github.com/GreyGnome/AdaEncoder">https://github.com/GreyGnome/AdaEncoder</a> <a href="https://github.com/GreyGnome/ooPinChangeInt">https://github.com/GreyGnome/ooPinChangeInt</a> <a href="https://github.com/ScottCProjects/Arduino/blob/master/Libraries/cbiface.h">https://github.com/ScottCProjects/Arduino/blob/master/Libraries/cbiface.h</a>
12.2.21	LM1972 am Arduino (japanisch)	<a href="https://lowreal.net/2018/02/20/1">https://lowreal.net/2018/02/20/1</a>
12.2.22	Radio Arduino Library	<a href="https://github.com/mathertel/Radio">https://github.com/mathertel/Radio</a>

## 12.3 Sonstige

	Bezeichnung	Quelle
12.3.1	Autodesk Fusion 360	<a href="https://www.autodesk.de/campaigns/education/fusion-360">https://www.autodesk.de/campaigns/education/fusion-360</a>
12.3.2	EAGLE	<a href="https://www.autodesk.com/products/eagle/free-download">https://www.autodesk.com/products/eagle/free-download</a>
12.3.3	JLCPCB	<a href="https://jlcpcb.com/">https://jlcpcb.com/</a>
12.3.4	Miro (für Diagramme)	<a href="https://miro.com/">https://miro.com/</a>
12.3.5	Lautsprecherbauanleitung	<a href="https://www.lautsprecherbau.de/grundlagen/theorie/gehaeusematerial/10101.de">https://www.lautsprecherbau.de/grundlagen/theorie/gehaeusematerial/10101.de</a>
12.3.6	Fritzing	<a href="https://fritzing.org/">https://fritzing.org/</a>



## 13 Abbildungsverzeichnis

Abbildung 1: Hardwareüberblick .....	9
Abbildung 2: Gießform für Beton .....	11
Abbildung 3: Beton Fehlschläge.....	12
Abbildung 4: Alternatives 3D Gerüst, das nicht gefertigt wurde.....	13
Abbildung 5: Das endgültige Gehäuse .....	13
Abbildung 6: Gehäuseplan .....	14
Abbildung 7: Gehäuseplan (2).....	15
Abbildung 8: Gehäuseplan (3).....	16
Abbildung 9: Zusammengeklebtes Gehäuse .....	17
Abbildung 10: Lackieren des Gehäuses .....	17
Abbildung 11: Fertiges Gehäuse .....	18
Abbildung 12: Lackieren der Frontplatte .....	18
Abbildung 13: Fertig zusammengebauter Prototyp .....	19
Abbildung 14: Layout V1.1 (Rückseite gleich wie V1.0).....	23
Abbildung 15: Schaltplan .....	24
Abbildung 16: Layout V1.0 (Vorderseite) .....	25
Abbildung 17: Layout V1.0 (Rückseite) .....	26
Abbildung 18: Rückseite der Leiterplatte .....	27
Abbildung 19: Vorderseite der Leiterplatte .....	27
Abbildung 20: Benutzeroberfläche - Navigationsleiste.....	49
Abbildung 21: Benutzeroberfläche - Statusleiste .....	50
Abbildung 22: Benutzeroberfläche - Wiedergabe .....	53
Abbildung 23: Benutzeroberfläche - Einstellungen .....	56
Abbildung 24: Benutzeroberfläche - W-Lan Einstellungen.....	59
Abbildung 25: Benutzeroberfläche - Passwort-Feld .....	61
Abbildung 26: Benutzeroberfläche Bluetooth Einstellungen .....	62
Abbildung 27: Benutzeroberfläche - Eingabeeinstellungen .....	64
Abbildung 28: Benutzeroberfläche - Anzeigeeinstellungen .....	65
Abbildung 29: Benutzeroberfläche - Farbauswahl .....	66
Abbildung 30: Benutzeroberfläche - Systemeinstellungen .....	67
Abbildung 31: Benutzeroberfläche - Quellenauswahl.....	68
Abbildung 32: Benutzeroberfläche - RadioStreams .....	72
Abbildung 33: Benutzeroberfläche - LokaleDateien.....	73
Abbildung 34: Steckbrett Prototyp .....	78