

I M A

Intelligent Magic Audio

Jonas Lux, Szymon Banasiak, Leon Braungardt

*Ein Projekt aus dem Modul ESP an der TH Köln im
Sommersemester 2024*



Inhaltsverzeichnis

1 Einleitung	1
2 Detaillierte Projektbeschreibung	2
2.1 Modulare Eurorack Synthesizer	2
2.2 Das Konzept	3
3 Projektanforderungen	4
3.1 Funktionale Anforderungen	4
3.1.1 LF01: Steuerung des Cursors und Auswahl eines Samples	4
3.1.2 LF02: Verhalten der Fader und des Displays	5
3.1.3 LF03: Klassifizierung von Audiodateien des Benutzers durch ein neuronales Netz .	6
3.1.4 LF04: Aufnahme von Audioquelle über REC IN	7
3.1.5 LF05: Wiedergabe von Audiodaten über OUT	7
3.1.6 LF06: Dynamische Änderung der Tonhöhe/Abspielgeschwindigkeit	7
3.2 Nicht-Funktionale Anforderungen	8
3.2.1 Performance	8
3.2.2 Benutzerfreundlichkeit	8
3.2.3 Effizienz	8
4 Spezifikation	9
4.1 Architektur	9
4.1.1 Systemkontext	9
4.1.2 Zustandsautomat	10
4.2 Arbeitspakete	11
4.3 Schnittstellenbeschreibung und Integration der Komponenten	12
4.3.1 Interaktion Downsampling	12
4.3.2 Interaktion Klassifizierung	12
4.3.3 Interaktion Interfacing	13
4.4 Technische Spezifikation	14
4.4.1 Hardware	14
4.4.2 Pinout Interface Komponente	16
4.4.3 Pinout Audio Komponente	18
4.4.4 Schaltpläne	20
5 Durchführung	22
5.1 Repository Struktur	22
5.1.1 Code	22
5.1.2 Dokumentation	22
5.1.3 Hardwaredesign	23
5.1.4 Anderes	23
5.2 Audio Engine des Samplers	23
5.2.1 Signalfluss der Audioengine	23

5.2.2	Latenzen	23
5.2.3	PCM5102a und I2S	25
5.2.4	Audiostream von SD-Karte	25
5.2.5	Playback und Pitched Playback	26
5.2.6	Downsampling eines Audiochunks für die Verarbeitung von Audio durch das Neuronalen Netzes	27
5.3	Klassifizierung von Audiodateien durch ein neuronales Netz	29
5.3.1	Ansatz für die Klassifizierung von Audiodaten	29
5.3.2	Generieren der Eingabedaten	31
5.3.3	Datenvorverarbeitung	31
5.3.4	Aufteilen der Eingabedaten in Trainings- Test und Validierungsdaten	33
5.3.5	Training des neuronalen Netzes	33
5.3.6	Betrieb des neuronalen Netzes auf dem STM32 Microcontroller	36
5.4	Interface	39
5.4.1	Encoder	39
5.4.2	Schiebenpotentiometer, ADC und DMA	41
5.4.3	Display, SD1306 Treiber und I2C	44
5.4.4	Dateimangement	46
5.5	PCB-Design	48
5.6	Integration der einzelnen Komponenten in das Gesamtsystem	49
6	Test und Validierung	50
6.1	Testprotokoll: /LF01/	50
6.1.1	Auswertung der Drehung des Encoders	50
6.1.2	Debouncing des Push-Buttons	50
6.1.3	Menünavigation	51
6.2	Testprotokoll: /LF02/	52
6.2.1	ADC-Konfiguration überprüfen	52
6.2.2	DMA-Konfiguration überprüfen	52
6.2.3	Daten analysieren	53
6.2.4	Wiedergabe der Fader-Werte und Filterung auf dem Display	54
6.3	Testprotokoll: LF01 LF02 SD-Karten SPI-Schnittstelle	55
6.3.1	Testzielsetzung	55
6.3.2	Testdurchführung	55
6.3.3	Testergebnisse	56
6.4	Testprotokoll zu /LF03/	57
6.4.1	Validierung der Sinnhaftigkeit der Merkmalsklassen	57
6.4.2	Validierung des erfolgreichen Training des neuronalen Netzes	57
6.4.3	Validierung der korrekten Funktion der Klassifikation auf dem STM32 Microcontroller	59
6.5	Testprotokoll zu /LF05/ Wiedergabe über Audio Codec	61
6.6	Testprotokoll zu /LF06/ Pitchgenauigkeit des Wiedergabealgorithmus	63
6.7	Testprotokoll zu /LQ01/ Latenzmessung	65

Abbildungsverzeichnis

1	Gepatchtes Eurorack 3U Rack (einreihig)	2
2	Panel Design Mockup	3
3	Kontextdiagramm des Gesamtsystems	9
4	Zustandsautomat des Systems	10
5	GANTT Diagramm	11
6	Komponentendiagramm	12
7	Rotary-Encoder	14
8	LCD-Display	14
9	Potentiometer	15
10	PCM5102a Breakout Board	17
11	Waveshare Micro SD-Karten Modul	17
12	STM32 NUCLEO-F401RE Board	19
13	STM32 Nucleo-F722ZE Board	19
14	Digitaler Audio Signalfluss	23
15	Double Buffering mit DMA und DSP	24
16	CubeMX FATFS SDIO Einbindung	26
17	Anti-Aliasing FIR Lowpass Filter	27
18	Aufbau des künstlichen Neurons. Quelle: Thieling, Lothar: "Neuronale Netze (Vorlesungsskript ML)", Kapitel F, Seite 6.	29
19	Darstellung eines PCM Audiosignals [19]	30
20	Darstellung eines Audiosignals als Spektrogramm [19]	30
21	Python-script zum manuellen Auswählen Labeln der Trainingsdaten	31
22	Repräsentation der verschiedenen Merkmalsklassen in den Entwicklungsdaten	32
23	Darstellung der Unterteilung der Audiosamples in Audiosubsamples und Frames	32
24	Darstellung des Sliding Window Prinzips	33
25	Aufteilung der Eingabedaten in Trainings-, Validierungs- und Testdaten (erstellt mit sankeydiagram.net)	34
26	Darstellung des Loss-Werts über den Verlauf der Epochen hinweg	35
27	Ablauf der Datenvorverarbeitung und Klassifizierung eines Audiosamples im C-Code	36
28	Konfiguration von STM32Cube.AI in CubeMX	37
29	Phasenverschiebung A und B	39
30	ADC Conversion	41
31	DMA ADC FADER Bus System	42
32	I2C Kommunikation	44
33	Display SD1306 I2C Kommunikation	45
34	Confusion-Matrix" des trainierten neuronalen Netzes	58
35	Screenshot vom CubeMX Debugger. In den Elementen des "aiOutData"-Arrays kann das Klassifikationsergebnis abgelesen werden	60
37	Frequenzmessung bei Pitchfaktor 0.5	64
38	Frequenzmessung bei Pitchfaktor 1.0	64
39	Frequenzmessung bei Pitchfaktor 2.0	64

-
- 40 Oszilloskopansicht der Latenzmessung von Trigger (Blau) bis Audioausgang (Gelb) 65

1 Einleitung

Bei dem Projekt "Intelligent Magic Audio" (IMA) handelt es sich um ein Projekt aus dem Modul "Embedded Systems Praktikum" (ESP) im Sommersemester 2024 an der TH Köln.

Ziel des Projekts ist die Entwicklung eines Audiosamplers im Eurorack-Standard, der auf einem STM32 Mikrocontroller basiert. Die Besonderheit des Samplers darin, dass auf eine SD-Karte geladenene Audiosamples über ein lokal laufendes Neuronales Netz (Edge AI) in fünf Klassen klassifiziert werden. Mit Hilfe von fünf Fadern und einer Suchfunktion kann der Nutzer sich passende Samples aus seiner Audio-bibliothek vorschlagen und bei Bedarf wiedergeben lassen.

Das Projekthema wurde aus Eigeninitiative vorgeschlagen, da es zentrale Aspekte der eingebetteten Systeme wie die Signal- bzw. Audioverarbeitung und den Umgang mit Peripheriegeräten vereint. Dies ermöglicht den Teammitgliedern, ihre Kenntnisse aus dem Modul "Embedded Systems" (ES) zu nutzen, um ein eigens entworfenes Produkt zu entwickeln und gleichzeitig ihre Fähigkeiten in ihren persönlichen Interessensgebieten zu vertiefen. Die Integration der Edge AI-Komponente bietet zudem die Möglichkeit, Erfahrungen im Betrieb neuronaler Netze auf Mikrocontrollern zu sammeln.

Der folgende Bericht bietet eine detaillierte Übersicht über den Projektverlauf, die verwendeten Technologien, die Herausforderungen und die Ergebnisse.

2 Detaillierte Projektbeschreibung

2.1 Modulare Eurorack Synthesizer

In der modernen elektronischen Musikproduktion erfreut sich die Modularität von Synthesizern wieder großer Beliebtheit. Die Methoden, verschiedene Module von unterschiedlichen Herstellern miteinander zu kombinieren, erleben ein Revival. [1] Wo einst die Reise der elektronischen Musik begann, werden diese Ansätze erneut aufgegriffen.

In den 1950er Jahren erzeugten die Pioniere der elektronischen Musik mit Labor- und Testequipment neue Klänge. Signalgeneratoren, Filter und anderes technisches Gerät aus den Ingenieurlabors wurden funktional zu Musikinstrumenten umgewandelt – die elektronische Musik war geboren. [2]

In den 1990er Jahren, durch die bessere Zugänglichkeit und günstigeren Kosten von Chips und Halbleitern, fand dieses Mindset den Weg zu den Consumer-Musikproduzenten - Dieter Doepfer entwickelte den Eurorack-Standard für modulare Synthesizer. [3]

Die Module sind in 3U-Höhe (etwa 133,4 mm) und variabler Breite in HP (horizontal pitch) erhältlich, wobei 1 HP etwa 5,08 mm entspricht. Diese Maße basieren auf dem 19"-Rack-Standard, der in Computertechnik und Audio Equipment weit verbreitet ist. Die Module werden in Racks oder Cases montiert, die mit Schienen ausgestattet sind, um eine flexible und modulare Konfiguration zu ermöglichen. [4]

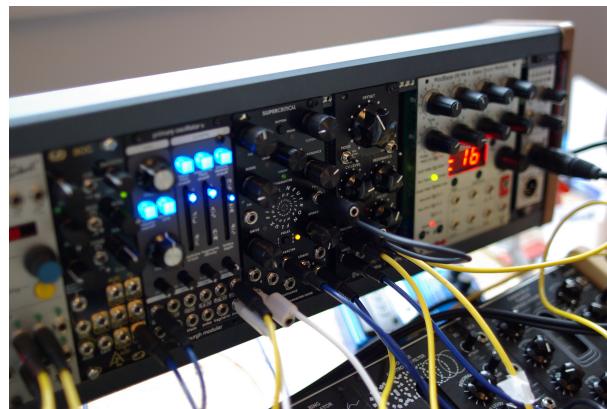


Abbildung 1: Gepatchtes Eurorack 3U Rack (einreihig)

Elektrisch werden Eurorack-Module über eine gemeinsame Stromversorgung betrieben, die typischerweise +12V, -12V und manchmal +5V bereitstellt. Module werden über 10-polige oder 16-polige Flachbandkabel an eine Stromverteilungsplatine (Bus Board) angeschlossen. Die wichtigsten elektrischen Verbindungen umfassen +12V, -12V und GND, wobei der 16-polige Anschluss zusätzlich +5V und weitere Signale unterstützen kann. [3]

Für die Signalübertragung nutzen Eurorack-Module 3,5 mm Monoklinkenstecker, um die Module miteinander zu „verpatchen“. Über diese Kabel werden Audiosignale (typischerweise $\pm 5V$), Steuersignale (CV) und Triggersignale (0V bis +5V) zwischen den Modulen übertragen.

Audiosignale, Steuersignale und Triggersignale werden also mit denselben Buchsen und Kabeln verbunden. Dies eröffnet enorme Möglichkeiten, die Klangpalette zu erweitern. So können beispielsweise Audiosignale zur Steuerung der Parameter eines Moduls verwendet werden, was kreative und innovative Patches ermöglicht.

Mit der Eurorack Revolution haben sich modulare Synthesizer auch technisch weiterentwickelt. Es wird vermehrt auf digitale Module gesetzt. Die Zeit der Analog-Puristen ist vorbei; alle Möglichkeiten der digitalen Signalverarbeitung werden genutzt, um innovative Module zu entwerfen, die neue Methoden und Ideen in das eigene Rack bringen.

Eine Unterkategorie der digitalen Module bilden die Audiosampler. Diese Module nehmen digitale Aufnahmen (Samples) von Klängen oder Musikstücken auf, speichern und spielen sie ab. Die Samples können durch verschiedene Trigger aktiviert und in unterschiedlichen Tonhöhen und Geschwindigkeiten abgespielt werden. Audiosampler ermöglichen es Musikern, realistische Klänge von Instrumenten, Stimmen oder Umweltgeräuschen in ihre Musik zu integrieren und kreativ zu manipulieren.

2.2 Das Konzept

Es gibt durchaus Audiosampler mit ergonomischer Steuerung, großen hochauflösenden Displays und zahlreichen Knöpfen, um diese komplexen Audiomaschinen angenehm zu steuern. Beispiele hierfür sind die legendäre MPC von AKAI [5] oder der Octatrack von Elektron [6].

Die Hardware dieser Geräte sind jedoch aufgrund ihrer Bedienelemente und der damit einhergehenden Ergonomie so groß, dass sie unmöglich in ein Eurorack unterzubringen sind.

Daher sind Samplermodule im Eurorack-Standard häufig mit wenigen Features ausgestattet oder haben aufgrund der minimalen Bedienelemente eine komplizierte, verschachtelte Bedienung mit vielen Untermenüs und Ebenen.

“**I M A**“ (Intelligent Magic Audio) macht es Musikern und Sounddesignern hier deutlich einfacher! Der Schlüsselpunkt ist das gezielte Finden von Samples aus dem vorhandenen persistenten Speicher. Nur Samples des gewünschten Typs zu filtern, setzt normalerweise ein hohes Maß an manueller Sortierung und penibler Ordnerstrukturierung voraus.

Durch den neuronal gesteuerten Suchfilter von I M A kann eine Vorauswahl anhand musikalischer Parameter getroffen werden, ohne den kreativen Fluss des Künstlers zu unterbrechen.

Das Neuronale Netz klassifiziert jedes neu hinzugefügte Audiosample nach 5 Klassen, welche speziell musikalischen Parametern entsprechen. Das ermöglicht einen, für den Musiker, organischen Suchvorgang. Die Klassifizierung wird nach der Aufnahme durch den **REC IN**, oder durch das manuelle Kopieren auf die SD Karte angestoßen, also immer wenn eine neue Audiodatei in den persistenten Speicher des Samplers hinzukommt.

Wie in **Abbildung 2** ersichtlich, gibt es 5 Schieberegler (von nun an Fader genannt), sowie 2 Encoder. Der Suchfilter lässt sich über die 5 “**TASTE**“ Fader einstellen. Jeder Fader entspricht einer Klasse der Klassifizierung des Neuronalen Netzes. Je nach eingestelltem Suchfilter, werden Samples zum Abspielen vorgeschlagen, die den Suchfiltereinstellungen entsprechen.

Ist man beispielsweise auf der Such nach einem bassigen und flächigen Sample, stellt man die Fader entsprechend seiner Wünsche ein (viel *Bassy* und *Sustained*). Ein Filteralgorithmus zeigt nun den Suchfilter entsprechenden Samples auf dem Display an. Nun kann ein Sample mit dem Encoder ausgewählt werden, und per Druck auf den Encoder, oder per Impuls über den **TRIG IN** abgespielt werden.



Abbildung 2: Panel Design Mockup

3 Projektanforderungen

Basierend auf dem zuvor entwickelten Produktkonzept wurden funktionale und nicht-funktionale Anforderungen abgeleitet. Sie bilden das Lastenheft des Projekts und werden in den folgenden zwei Abschnitten näher beschrieben.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen definieren die Funktionen, die dem Benutzer zur Verfügung gestellt werden sollen. Jede dieser Anforderungen wird mit dem Kürzel „LF“ für Lastenfall, gefolgt von einer fortlaufenden Nummer, gekennzeichnet.

3.1.1 LF01: Steuerung des Cursors und Auswahl eines Samples

Priorität	Muss
Akteur	Benutzer
Beschreibung	<p>1. Cursor Bewegung:</p> <p>Aktion: Der Benutzer dreht den Encoder.</p> <p>Reaktion: Der Cursor auf dem LCD-Display bewegt sich nach oben oder unten entsprechend den Stepperschritten des Encoders.</p> <p>Details:</p> <ul style="list-style-type: none"> - Der Cursor bewegt sich um eine Position in der Liste pro Schritt des Encoders. - Der Cursor kann den Beginn oder das Ende der Liste erreichen, ohne die Liste über die Grenzen hinaus zu bewegen. Er springt von daher am Anfang oder Ende der Liste. <p>2. Sample Auswahl:</p> <p>Aktion: Der Benutzer drückt den Switch des Encoders.</p> <p>Reaktion: Das aktuell ausgewählte Sample wird hervorgehoben und sein Name wird unter der Liste auf dem LCD-Display angezeigt.</p> <p>Details:</p> <ul style="list-style-type: none"> - Der Name des ausgewählten Samples wird deutlich unter der Liste angezeigt. - Der Name sollte in einem Format angezeigt werden, das gut lesbar ist und keine Abkürzungen oder Unklarheiten aufweist.

3.1.2 LF02: Verhalten der Fader und des Displays

Priorität	Muss
Akteur	Benutzer
Beschreibung	<p>1. Anzeige der Fader-Werte:</p> <p>Aktion: Der Benutzer bewegt die Schiebepotentiometer.</p> <p>Reaktion: Die prozentuale Angabe der Spannung, in der sich der Potentiometer befindet, wird angezeigt.</p> <p>Details:</p> <ul style="list-style-type: none"> - Die prozentualen Ausgaben spiegeln die Klassen wider, wie Rhythmic, Sustained, usw., sowie die Einstellung des Thresholds - Die Prozentsätze sollen klar und deutlich angezeigt werden, ohne Verzögerung. <p>2. Filtern der Samples:</p> <p>Aktion: Der Benutzer bewegt die Schiebepotentiometer.</p> <p>Reaktion: In der Liste auf dem Display werden die Samples angezeigt dessen Audio Klassen nach der Klassifizierung, die den Prozentsatz der Potentiometereinstellung nicht mehr als einen bestimmten Schwellenwert überschreiten.</p> <p>Details:</p> <ul style="list-style-type: none"> - Die Anzeige der Samples soll dynamisch aktualisiert werden, basierend auf den aktuellen Fader-Werten. - Der Schwellenwert kann angepasst werden, um eine präzise Filterung der Samples zu ermöglichen.

3.1.3 LF03: Klassifizierung von Audiodateien des Benutzers durch ein neuronales Netz

LF03

Priorität	Muss
Akteur	Benutzer
Beschreibung	<p>Ein neuronales Netz soll beim Einschalten des Samplers die unklassifizierten Audiosamples in für die Musikproduktion relevante Kategorien klassifizieren, wie beispielsweise Genre, Instrumententyp oder Klangfarbe.</p> <p>Dieser Lastenfall enthält drei Arbeitspakete:</p> <ol style="list-style-type: none"> 1. /LF03/AP01 Auswahl geeigneter Merkmalsklassen für Audiosamples: Diese Kategorien sollten zum einen sinnvoll bei der Musikproduktion sein, zum anderen auch mit der genutzten Hardware klassifizierbar und damit realistisch sein. 2. /LF03/AP02 Entwicklung und Training des neuronalen Netzes: Das neuronale Netz muss in der Lage sein, annährend zutreffende Klassifizierungsergebnisse auszugeben. Ziel ist ein Accuracy-Wert von über 50%. Zum Vergleich läge der Accuracy-Wert bei $\frac{1}{5} = 20\%$, wenn das neuronale Netz rein zufällige Ergebnisse ausgeben würde. Gleichzeitig ist dieses Ziel von 50% realistisch erreichbar. Um dieses Ziel zu erreichen, muss das neuronale Netz ausreichend mit relevanten Daten trainiert werden. Weiterhin sollte die Confusion-Matrix eine klare Tendenz in der korrekten Klassifizierung der Kategorien aufzeigen. Darauf hinaus sollten auch manuell eingegebene Samples, wie beispielsweise ein Rock-Song, zuverlässig den entsprechenden Kategorien zugeordnet werden können. 3. /LF03/AP03 Betrieb des neuronalen Netzes auf dem Nucleo-F7 Board: Das neuronale Netz, einschließlich aller vor- und nachgelagerten Datenvorverarbeitungsschritte und der Generierung der Spektrogramme, muss auf einem Nucleo-F7 Board laufen können.

3.1.4 LF04: Aufnahme von Audioquelle über REC IN

Priorität	Muss
Akteur	Benutzer
Beschreibung	Das System muss eine Audioquelle mit Line-Pegel, welche in der Eingangsbuchse eingesteckt ist aufnehmen können, und diese Aufnahme als korrekt kodierte PCM .wav Datei auf der SD-Karte abspeichern. Nach der Aufnahme muss eine Indexierung und Klassifizierung durch das Neuronale Netz geschehen (Wie in LF03 beschrieben)

3.1.5 LF05: Wiedergabe von Audiodaten über OUT

Priorität	Muss
Akteur	Benutzer
Beschreibung	Das System muss Stereo PCM .wav Audiodaten, welche auf der SD-Karte gespeichert sind, über den Audiocodec abspielen können. Ziel ist eine zuverlässige, saubere Audiowiedergabe, die keine Störgeräusche wie Knackser und andere Artifakte produziert.

3.1.6 LF06: Dynamische Änderung der Tonhöhe/Abspielgeschwindigkeit

Priorität	Muss
Akteur	Benutzer
Beschreibung	Das System muss die Audiodaten (wie in LF05 beschrieben) in verschiedenen Abspielgeschwindigkeiten wiedergeben können. So kann die Tonhöhe angepasst werden. Hierbei ist ein Algorithmus zu implementieren, der eine auditiv qualitative Tonhöhenanpassung umsetzt. Artifakte sollen vermieden werden.

3.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen spezifizieren Qualitätsmerkmale, die das System erfüllen muss, aber nicht direkt mit den spezifischen Funktionen verbunden sind. Diese Anforderungen werden mit dem Kürzel „LQ“, gefolgt von einer fortlaufenden Nummer, gekennzeichnet.

3.2.1 Performance

/LQ1/	Latenz
Kategorie	Performance
Beschreibung	Die Latenzzeit zwischen dem Triggern des Abspielmechanismus (Play-Button oder TRIG IN) und der Audiowiedergabe soll minimal sein (unter 20 ms), um die Funktionalität als Musikinstrument für die Live-Performance zu gewährleisten.

/LQ02/	Performante Sampleklassifizierung
Kategorie	Performance
Beschreibung	Das System muss in der Lage sein, Audio-Daten schnell zu Klassifizieren, damit der Benutzer nicht in seinem kreativen Drang behindert wird.

/LQ3/	Datenübertragungsrate
Kategorie	Performance
Beschreibung	Die Middleware muss hohe Datenübertragungsraten unterstützen, um eine kontinuierliche und verlustfreie Audioübertragung zu gewährleisten.

3.2.2 Benutzerfreundlichkeit

/LQ4/	Intuitive Benutzeroberfläche
Kategorie	Benutzerfreundlichkeit
Beschreibung	Die Benutzeroberfläche muss intuitiv und leicht bedienbar sein, damit Nutzer direkt damit arbeiten können, ohne eine Dokumentation lesen zu müssen.

3.2.3 Effizienz

/LQ5/	Optimierung der Audioverarbeitung
Kategorie	Effizienz
Beschreibung	Algorithmen für die Audioverarbeitung und Klassifizierung sollten so optimiert sein, dass sie minimale Rechenressourcen benötigen, ohne die Genauigkeit zu beeinträchtigen.

4 Spezifikation

Basierend auf dem entwickelten Konzept und den definierten Anforderungen wird die detaillierte Spezifikation des Systems ausgearbeitet. Diese umfasst die Architektur des gesamten Systems, welche die Aufteilung in einzelne Komponenten und deren Interaktion untereinander beschreibt. Ebenso wurden die Schnittstellen zwischen den Komponenten definiert. Weiterhin wird erläutert, wie das System mit seiner Umgebung interagiert. Die technische Spezifikation umfasst die verwendete Hardware sowie die eingesetzten Protokolle und Standards, die für Umsetzung der einzelnen Komponenten benötigt werden.

Zur besseren Veranschaulichung des Gesamtsystems, Beziehungen zwischen den Komponenten und der Abläufe wurden, wo sinnvoll, Diagramme erstellt. Diese visualisieren bestimmte Zusammenhänge. Bei der Erstellung dieser Diagramme wurde das SA/RT-Entwurfskonzept als Orientierungshilfe genutzt. Aufgrund der des hohen Umfangs bzw. der Komplexität des Projekts wurden die strengen Vorgaben des SA/RT-Konzepts jedoch flexibel gehandhabt und nicht strikt befolgt.

4.1 Architektur

Die Architektur des Projekts bietet eine Gesamtdarstellung des Systems und zeigt, wie die verschiedenen Komponenten miteinander interagieren. Das System gliedert sich in vier Hauptbereiche: Audioverarbeitung, Benutzeroberfläche (User Interface) und das neuronale Netz und das Dateimanagement.

4.1.1 Systemkontext

IMA interagiert als komplexes System mit zahlreichen Peripherien und externen Libraries.

In Abbildung 3 wird diese Interaktion durch ein Kontextdiagramm deutlich gemacht.

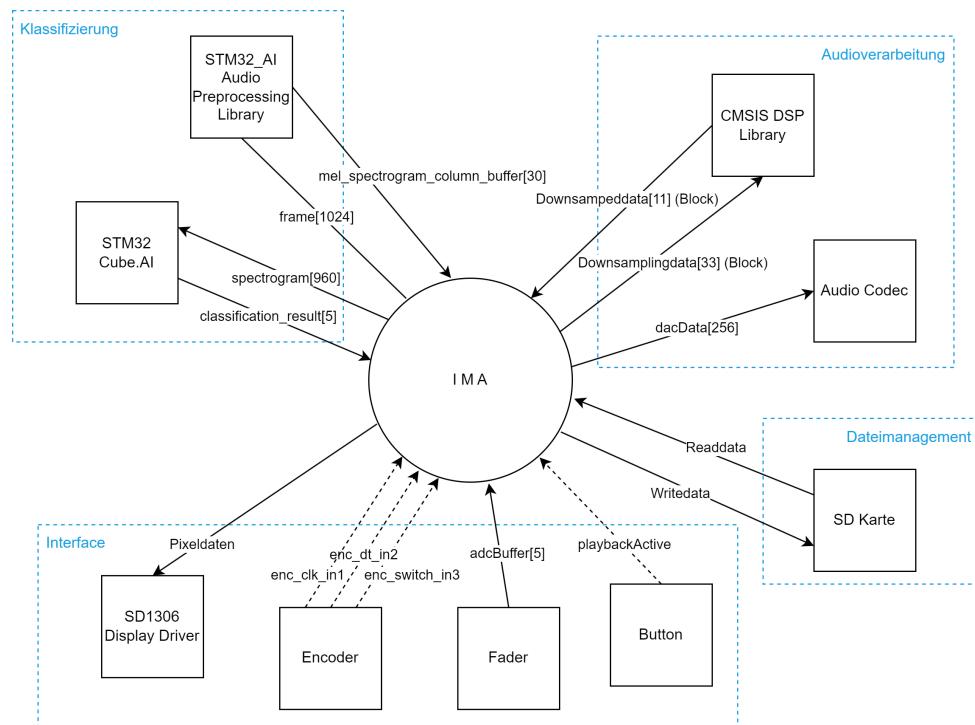


Abbildung 3: Kontextdiagramm des Gesamtsystems

Klassifizierung Die Klassifizierung durch das Neuronale Netz setzt das proprietäre **STM32Cube.AI** Software Pack voraus (näheres dazu im Abschnitt 5.3.6). Das neuronale Netz erhält als Eingabewert ein Spektrogramm bestehend aus 960 Elementen (32x30 Pixel), und liefert daraufhin ein Klassifikationsergebnis mit den Ähnlichkeitswerten zu den fünf Merkmalsklassen zurück.

Da das Neuronale Netz die Klassifizierung über ein Spektrogramm und nicht mit den rohen Audiodaten vollzieht, müssen die Audiodaten zunächst in ein solches umgewandelt werden. Das geschieht spaltenweise, wie in Abschnitt 5.3.1 beschrieben. Die Bibliothek für diese Umwandlung ist die ebenfalls proprietäre

STM32_AI_Audio_Preprocessing_Libray. Der Input der Umwandlungsfunktion ist ein Buffer von 1024 Samples(`frame[1024]`). Der Rückgabewert ist eine Spalte des Spektrogramms `mel_spectrogram_column_buffer[30]`.

Audioverarbeitung Um sicherzustellen, dass die **STM32_AI_Audio_Preprocessing_Libray** die zu klassifizierenden Audiodaten im richtigen Format erhält, müssen diese zuerst umgewandelt werden. Wie in Abschnitt 5.2.6 beschrieben, sind dafür Downsampling von 44.1kHz auf 16kHz Samplerate und die Umwandlung von Stereo auf Mono erforderlich. Für diese Schritte wird die Open-Source **CMSIS DSP Library** verwendet.

Bei der Audiomeldung erhält der Audio Codec pro Iteration den gesamten Audiobuffer `dacData[256]`.

Interface Um das Menü anzuzeigen, erhält der Treiber des Displays Strings, welche er intern in Pixeldaten umwandelt, sowie Linieninformationen in Form von Pixeldaten.

Es sind 3 GPIO Inputs für die Nutzung des Encoders erforderlich. `enc_clk_in1` und `enc_dt_in2` sind für die Auswertung des Drehimpulsgebers nötig. `enc_switch_in3` dient für die Push-Button Funktion des Encoders.

Die 5 Fader Spannungen werden in das Array `adcBuffer[5]` vom ADC über die DMA befüllt.

Der Play-Button schaltet das GPIO Signal `playbackActive`, womit die Audio Wiedergabe gestartet wird.

Dateimanagement Das Dateimanagement der Applikation wird über ein die C-Struktur **struct FileManager** realisiert, das die Dateizugriffe auf die Audiodateien organisiert und die persistente Speicherung der Klassifizierungsdaten durchführt.

Um die Klassifizierungsdaten zu speichern, wird eine neue Datei erstellt oder eine bestehende Datei geöffnet. Die Daten werden dann im ASCII- oder Binärformat in die Datei geschrieben. Der Filemanager-Struct verwaltet dabei die Datei-Handles und stellt sicher, dass die Datei nach dem Schreibvorgang ordnungsgemäß geschlossen wird.

4.1.2 Zustandsautomat

Zur detaillierten Darstellung des Funktionsablaufs und zur präzisen Planung des Zusammenspiels der Einzelkomponenten wurde ein Zustandsautomat entwickelt, der sich am SA/RT-Entwurfsmodell orientiert.

Die Struktur und die Übergänge dieses Zustandsautomaten sind in Abbildung 4 veranschaulicht. Diese Abbildung bietet eine umfassende Visualisierung der verschiedenen Zustände und deren Wechselwirkungen innerhalb des Systems.

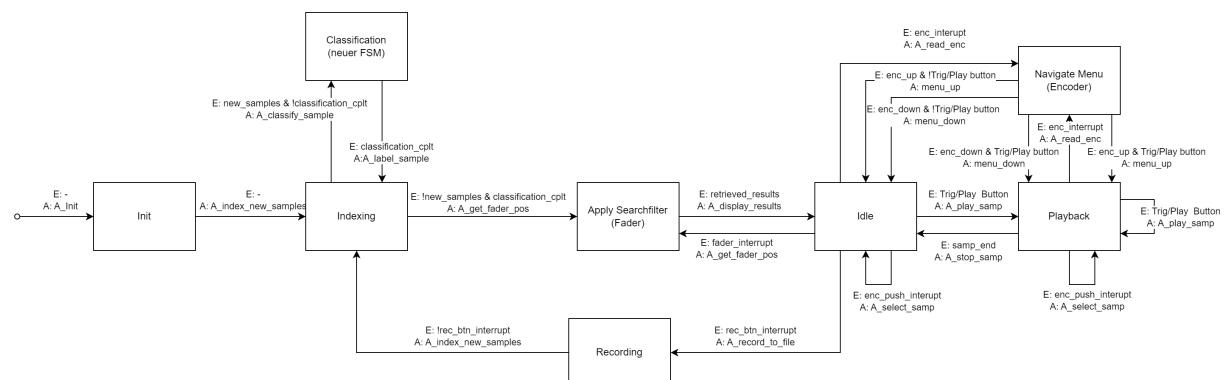


Abbildung 4: Zustandsautomat des Systems

Nach dem Initialisierungszustand **Init** beginnt der Automat im **Indexing**-Zustand. In diesem Zustand wird überprüft, ob noch unklassifizierte Samples vorhanden sind. Falls ja, wechselt der Automat in den **Classification**-Zustand, um das unklassifizierte Sample zu klassifizieren.

Nach der Klassifikation kehrt der Automat wieder in den **Indexing-Zustand** zurück, um zu überprüfen, ob noch weitere unklassifizierte Samples vorhanden sind. Dieser Prozess wiederholt sich, bis alle Samples

klassifiziert sind.

Sobald alle Samples klassifiziert sind, wechselt der Automat in den **Idle-Zustand**. Hier wird der Suchfilter angewendet, der mit fünf Fadern eingestellt ist, und die entsprechenden Samples werden auf dem Display angezeigt. Im **Idle-Zustand** können die Samples entweder abgespielt (im **Playback-Zustand**) oder über den **Navigate Menu-Zustand** durch das Menü navigiert werden.

Wird ein neues Sample (**Recording-Zustand**) aufgenommen, erfolgt die Klassifikation wie nach Initialisierung des Systems über den **Indexing-Zustand**.

Nach Änderung der Faderstellung, wird der Suchfilter wieder angewandt (**Apply-Searchfilter-Zustand**).

4.2 Arbeitspakete

Das in Abbildung 5 dargestellte GANTT-Diagramm zeigt den zu Projektbeginn festgelegten Arbeitsablauf sowie den geplanten Aufwand für jeden Arbeitsschritt.

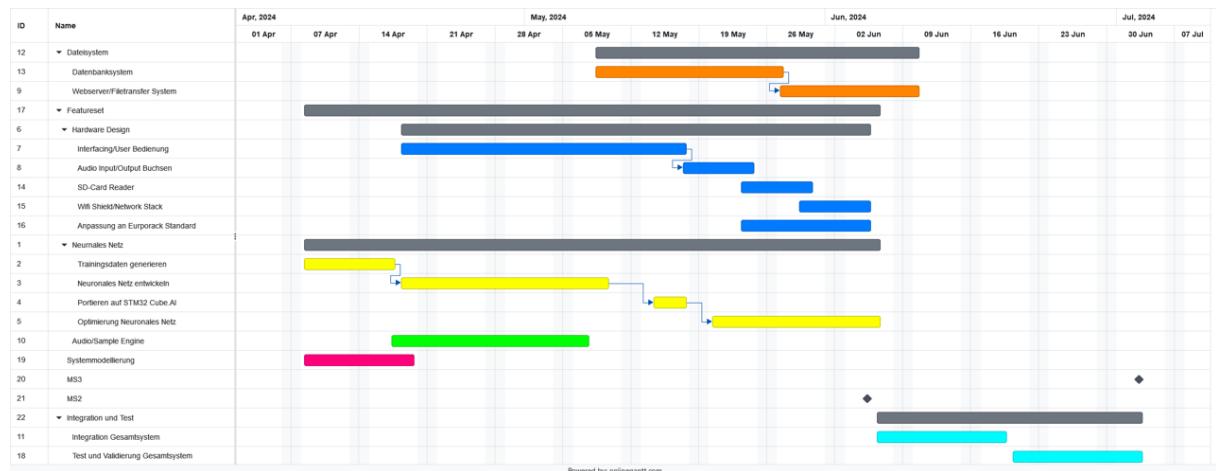


Abbildung 5: GANTT Diagramm

Arbeitspakete Aufteilung

Zur Aufteilung der Arbeitslast wurden Arbeitspakete definiert. Diese Pakete wurden dann unter Berücksichtigung der individuellen Interessen der Teammitglieder verteilt. Eine detaillierte Übersicht der Arbeitspaketaufteilung ist in der nachfolgenden Tabelle zu finden.

Leon Braungardt	Jonas Lux	Szymon Banasiak
<ul style="list-style-type: none"> • Neuronales Netz Topologie entwickeln • NN Trainieren und Anpassen • Portierung für STM32CubeAI • Algorithmus - Spektrogramm aus Audio Datei erstellen 	<ul style="list-style-type: none"> • Sample Record • Sample Playback • LearnData Generation (Aufbereitung der Lerndaten) • Hardware Design • Resampling • Audiodaten – SD-Card Read/Write 	<ul style="list-style-type: none"> • Fader • Encoder • LCD-Display • ADC • Dateisystem für Nutzdaten – SD-Card Read/Write

4.3 Schnittstellenbeschreibung und Integration der Komponenten

Damit eine Gesamtintegration geplant werden kann, wurden beim Entwurf des Systems Schnittstellen auf C-Ebene zwischen den Komponenten festgelegt.

Folgendes Diagramm zeigt die Relation und Interaktion der 4 Komponenten miteinander:

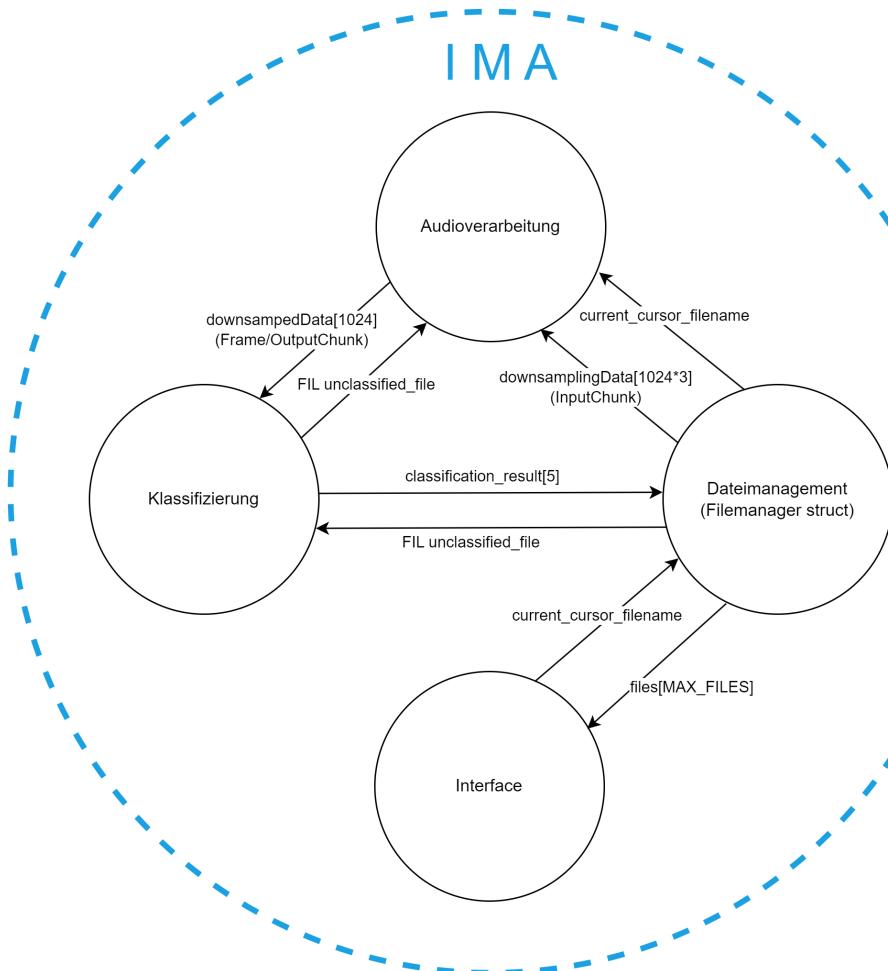


Abbildung 6: Komponentendiagramm

Das **Dateimanagement** ist der zentrale Knotenpunkt des Gesamtsystems. Es dient zunächst als Zugriffspunkt auf die Samples anhand ihrer Namen.

4.3.1 Interaktion Downsampling

Bevor die Klassifizierung durchgeführt wird, müssen die zu klassifizierenden Audiodaten von der **Audioverarbeitungs**-Komponente in das richtige Format gebracht werden. Die **Klassifizierung** übergibt den FATFS-Filehandle des herunterzusmplenden Files **FIL unclassified_file**. Mit dem Filehandle greift Die **Audioverarbeitung** auf die Datei zu und liest sich Chunkweise die `downsamplingData[1024*3]`. Die heruntergesampelten Daten `downsampledData[1024]` werden dann an die **Klassifizierung** zurück gegeben.

4.3.2 Interaktion Klassifizierung

Die für das Spektrogramm benötigten heruntergesampelten Daten werden zur **Klassifizierung** weitergeleitet. Basierend auf dem Spektrogramm werden die Dateien anhand ihrer Klangmerkmale klassifiziert. Die Ergebnisse dieser Klassifikation werden in `classification_result[5]` gespeichert. Dieses Array wird anschließend an das Dateimanager übergeben und wird dem Sample zugeordnet.

4.3.3 Interaktion Interfacing

Das Interface ruft alle Dateien aus dem **FileManager** Struct in `files[MAX_FILES]` ab. Es filtert die Dateien basierend auf den festgelegten Filtereinstellungen und speichert die übereinstimmenden Dateien in `shownFiles[MAX_FILES]`.

Beim Auswählen eines Samples wird dessen Name in `current_cursor_filename` gespeichert welcher ein Part den **FileManager** ist. Die **Audioverarbeitung** greift auf diesen Namen zu, um das Sample zu laden und abzuspielen.

4.4 Technische Spezifikation

Dieser Abschnitt behandelt die technischen Anforderungen des Projekts, die für die Umsetzung der verschiedenen Lastenfälle erforderlich sind. Es wird beschrieben, welche Hardware für jeden Lastenfall benötigt wird und aus welchen Gründen diese Auswahl getroffen wurde. Zudem wird die Wahl bestimmter Standards und Protokolle erläutert, die im Projekt verwendet werden.

4.4.1 Hardware

Encoder

/LF01/

Model RIC11-22S16D5M-TH

Betriebsspannung 3.3V

Mechanisch:

- 20 Impulse pro Umdrehung (20 PPR)
- 20 Positionen (20 DET)
- Schalter (SW)



Abbildung 7: Rotary-Encoder

Zur Auswahl der Samples und zum Navigieren durch das Menü wird ein Rotary-Encoder mit einem Switch Button eingesetzt. Das Empfangen der **A** und **B** Signale des Encoders erfolgt über die Pins **PA0** und **PA1**. Das Drücken des Switch über den Pin **PA4**. Durch das Drücken des Switch-Buttons wird der Pin **PA4** auf high gesetzt.

- **Präzise Steuerung:** Der Encoder ermöglicht eine präzise Steuerung des Cursors auf dem Display.
- **Benutzerfreundlichkeit:** Der Benutzer kann durch die Liste navigieren und ein Sample auswählen. Die Kombination aus Drehbewegung und Druckknopf-Funktionalität macht den Encoder intuitiv.

LCD-Display

/LF01/

Model: GME128128-01-ii2

Treiber: SH1107

Mode Monochrom (1Bit)

Betriebsspannung 5.0V



Abbildung 8: LCD-Display

Zur Visualisierung der Samples haben wir ein monochromes LCD-Display verwendet. Dieses Display wird über das **I2C**-Protokoll angesteuert. Der Datenpin **PB7** ist mit dem SDA-Pin des Displays verbunden, während der Taktpin **PB6** an den SCL-Pin des Displays angeschlossen ist.

Im Zusammenspiel mit dem Encoder ermöglicht das Display eine präzise Navigation durch den gewünschten Samplepool. Das Display wird mit einer Aktualisierungsrate von 20 FPS betrieben. Die Aktualisierungen werden mithilfe des Timers **TIM5** durchgeführt, der regelmäßig ausgelöst wird, um die Anzeige auf dem LCD zu aktualisieren.

- **Klare Visualisierung:** LCD-Displays bieten eine klare und gut lesbare Darstellung von Text und Grafiken.
- **Anpassbarkeit:** Sie können einfach an verschiedene Layouts und Designs angepasst werden.

Schiebe-Potentiometer

/LF02/

Model Bourns PTL45-15R0-103B2

Widerstand 10 kΩ

Weg 45mm

Betriebsspannung 3.3V



Abbildung 9: Potentiometer

Für die Filterfunktion benötigen wir 5 Potentiometer. Es wird zyklisch die Ausgangsspannung des Schleifers abgegriffen die die Teilspannung zwischen den VCC und dem GND darstellt. Dies erfolgt mit Hilfe vom ADC und dem DMA. Die Auswertung der Spannung erfolgt über die Pins **PA6**, **PA7**, **PB0**, **PB1**, **PC0**. Die Pins **PA6**, **PA7**, **PB0**, **PB1** sind für die Klassen zuständig **PC0** für den Schwellenwert an erlaubter Abweichung.

- **Präzise Steuerung und feine Abstimmung:** Ein Potentiometer ermöglicht eine stufenlose und präzise Einstellung. Durch das Schieben des Potentiometers kann der Benutzer den Cursor in kleinen, genauen Schritten bewegen.
- **Einfache Bedienung und intuitive Nutzung:** Potentiometer sind einfach und intuitiv zu bedienen.
- **Direkte visuelle Rückmeldung:** Durch die sofortige visuelle Rückmeldung auf dem LCD-Display kann der Benutzer sofort sehen, wie sich die Bewegung des Potentiometers auf die Position des Cursors auswirkt.

(LCD-Display)

Der LCD-Display ist der gleiche wie im /LF01/ beschrieben. Dieser dient zur Darstellung der Fader Einstellung in prozentualer Form.

4.4.2 Pinout Interface Komponente

Interface

Komponente	PIN	Signal-On-PIN	GPIO-Mode	GPIO-Pull-Up/Pull-Down	User-Label
Encoder Menü	PA0	n/a	EIMRETD	PULL UP	enc_a_clk_in1
	PA1	n/a	INPUT	PULL UP	enc_a_dt_in2
	PA4	n/a	EIMRETD	PULL UP	enc_a_switch_in3
ADC	PA6	ADC1_IN6	ANALOG	NPU NPD	FADER1
	PA7	ADC1_IN7	ANALOG	NPU NPD	FADER2
	PB0	ADC1_IN8	ANALOG	NPU NPD	FADER3
	PB1	ADC1_IN9	ANALOG	NPU NPD	FADER4
	PC0	ADC1_IN10	ANALOG	NPU NPD	FADER5
I2C	PB6	I2C1_SCL	AFOD	PULL UP	n/a
	PB7	I2C1_SDA	AFOD	PULL UP	n/a
SDIO	PC8	SDIO_D0	AFPP	PULL UP	n/a
	PC9	SDIO_D1	AFPP	PULL UP	n/a
	PC10	SDIO_D2	AFPP	PULL UP	n/a
	PC11	SDIO_D3	AFPP	PULL UP	n/a
	PC12	SDIO_CK	AFPP	NPU NPD	n/a
	PD2	SDIO_CMD	AFPP	PULL UP	n/a

EIMRETD = External Interrupt Mode and Rising Edge Trigger Detection

AFOD = Alternate Function Open Drain

NPU NPD = No Pull Up No Pull Down

AFPP = Alternate Function Push Pull

PCM5102a Breakout Board

/LF04/, /LF05/, /LF06/

Der PCM5102a Audio Codec wurde für die DAC Wandlung der Audiodaten verwendet. Für die Implementation und die Entwicklung des Prototyps wurde ein Breakout Board gekauft, um ohne die Notwendigkeit eines PCB-Design die Firmware für **I M A** zu schreiben. Komfortablerweise ist direkt eine 3.5 mm Stereo-Klinkenbuchse auf dem Board verbaut. Der Pegel beträgt Line-Level ($\pm 1.7 \text{ V}_{\text{rms}}$), und ist somit grob kompatibel zum Eurorack-Standard.

Das Breakout Board wird mit einer Versorgungsspannung von 3.3 V betrieben.

Als Übertragungsprotokoll dient **I₂S**. Die detaillierte Inbetriebnahme findet sich in Abschnitt 5.2.3.



Abbildung 10: PCM5102a Breakout Board

Waveshare Micro SD-Card Module

/LF01/, /LF02/, /LF03/, /LF04/, /LF05/, /LF06/

Das Waveshare Micro SD-Card Modul wird für die Speicherung und den Zugriff auf Audiodaten in diesem Projekt verwendet. Dieses Modul ermöglicht die einfache Integration von Micro SD-Karten in das System, um persistente Daten zu lesen und zu schreiben, ohne aufwändige PCB-Designs oder zusätzliche Hardwarekomponenten implementieren zu müssen.

Das Modul unterstützt den Standard **SDIO**-Bus zur Datenübertragung, was eine einfache Anbindung an das Mikrocontroller-System ermöglicht.

Die Inbetriebnahme und die spezifischen Konfigurationen für die Verwendung dieses Moduls werden in Abschnitt 5.2.4 detailliert beschrieben.

Das Modul wird mit einer Versorgungsspannung von 3.3 V betrieben.



Abbildung 11: Waveshare Micro SD-Karten Modul

4.4.3 Pinout Audio Komponente

Audio

Komponente	PIN	Signal-On-PIN	GPIO-Mode	GPIO-Pull-Up/Pull-Down	User-Label
SDIO	PC8	SDIO_D0	AFPP	PULL UP	n/a
	PC9	SDIO_D1	AFPP	PULL UP	n/a
	PC10	SDIO_D2	AFPP	PULL UP	n/a
	PC11	SDIO_D3	AFPP	PULL UP	n/a
	PC12	SDIO_CK	AFPP	NPU NPD	n/a
	PD2	SDIO_CMD	AFPP	PULL UP	n/a
I2S	PB10	IS2_CK	AFPP	NPU NPD	n/a
	PB12	IS2_WS	AFPP	NPU NPD	n/a
	PC3	IS2_SD	AFPP	NPU NPD	n/a
	PC6	IS2_MCK	AFPP	NPU NPD	n/a

NPU NPD = No Pull Up No Pull Down

AFPP = Alternate Function Push Pull

STM32 NUCLEO-F401RE

/LF01/, /LF02/, /LF04/, /LF05/, /LF06/

Da nicht genug NUCLEO-F722 Boards zur Verfügung standen, um jede Komponente damit zu entwickeln, wurden die Audio- und Interface-Komponenten mit dem, bereits aus ES bekannten, NUCLEO-F401RE Board entwickelt. Eine Migration und anschließende Integration auf das F7 Board folgt nach der Entwicklung der Komponenten.

Die F4-CPU ist im Vergleich zur F7 Zielhardware zwar schwächer, sollte jedoch für die minimalen DSP-Operationen und Peripherien des Interface ausreichen.

Es verfügt über einen ARM Cortex-M4 Prozessor mit einer Taktfrequenz von 84 MHz, 512 kB Flash-Speicher und 96 kB SRAM.

Das Board wird mit einer Versorgungsspannung von 5V betrieben. [7]



Abbildung 12: STM32 NUCLEO-F401RE Board

STM32 Nucleo-F722ZE

/LF03/

Das in **Abbildung 13** gezeigte STM32 Nucleo-F722ZE Board integriert einen Microcontroller und wird in erster Linie für den Betrieb des neuronalen Netzes benötigt. Auf diesem Board soll jedoch das gesamte Projekt umgesetzt werden. Nachdem noch keine Erfahrungen mit dem Ressourcenverbrauch von neuronalen Netzen auf Microcontrollern gesammelt wurden, wurde dieses Board ausgewählt, da es über mehr Ressourcen verfügt als die Nucleo-F7401RE Boards. Dadurch soll sicher gestellt werden, dass bei der Integration aller Komponenten nicht zu Ressourcenknappheiten, insbesondere beim RAM, kommt.

Es verfügt im Vergleich zum STM32 Nucleo-F401RE Board über mehr SRAM (256 Kbytes vs. 96 Kbytes) und über eine leistungsstärkere CPU (Cortex M7 CPU mit 462 DMIPS/2.14 DMIPS vs. Cortex M4 CPU mit 105 DMIPS/1.25 DMIPS) [8].

Das Board wird mit einer Versorgungsspannung von 5V betrieben.

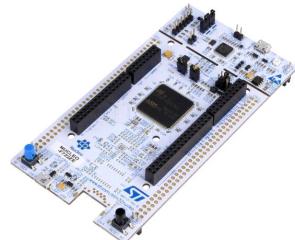
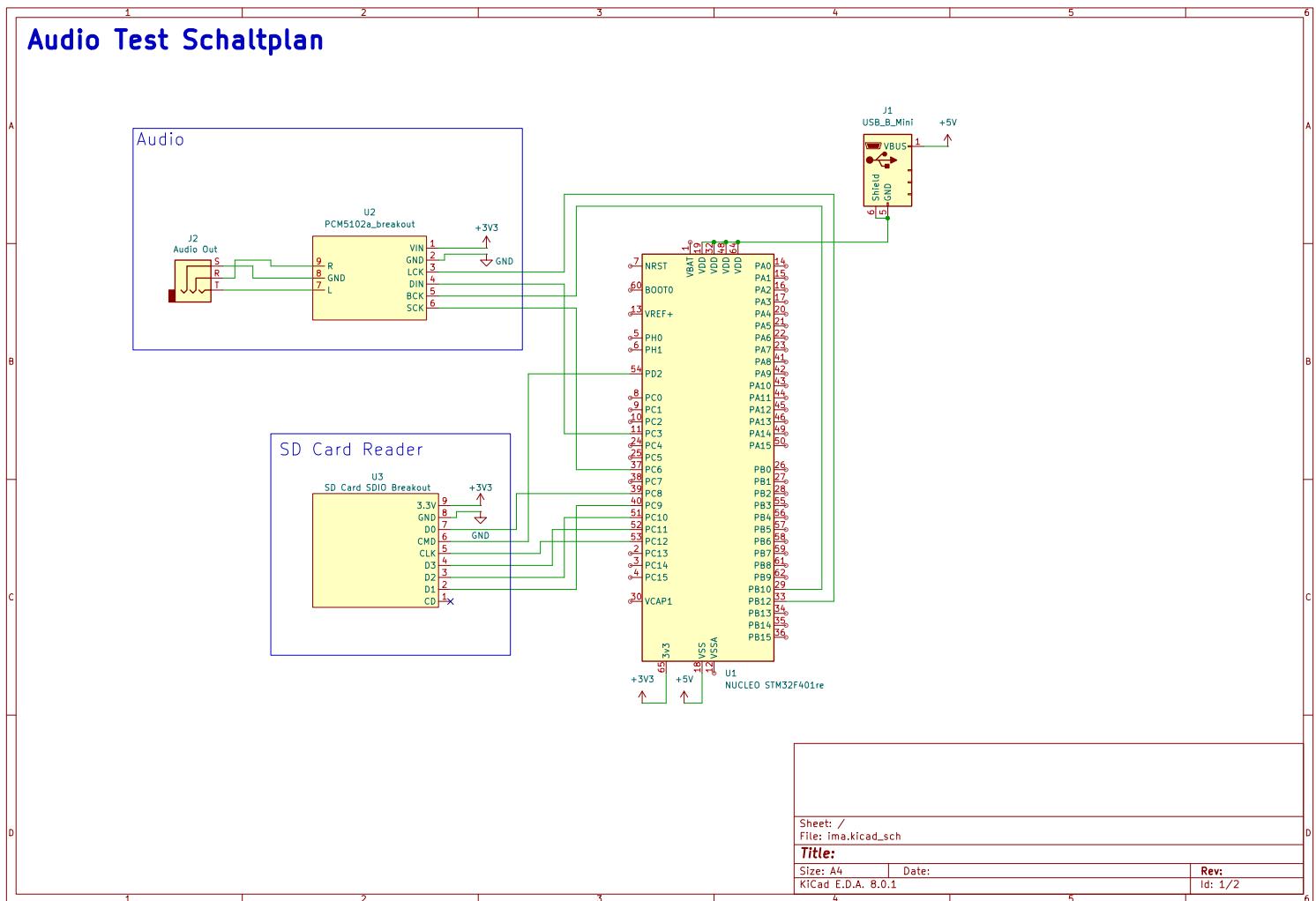
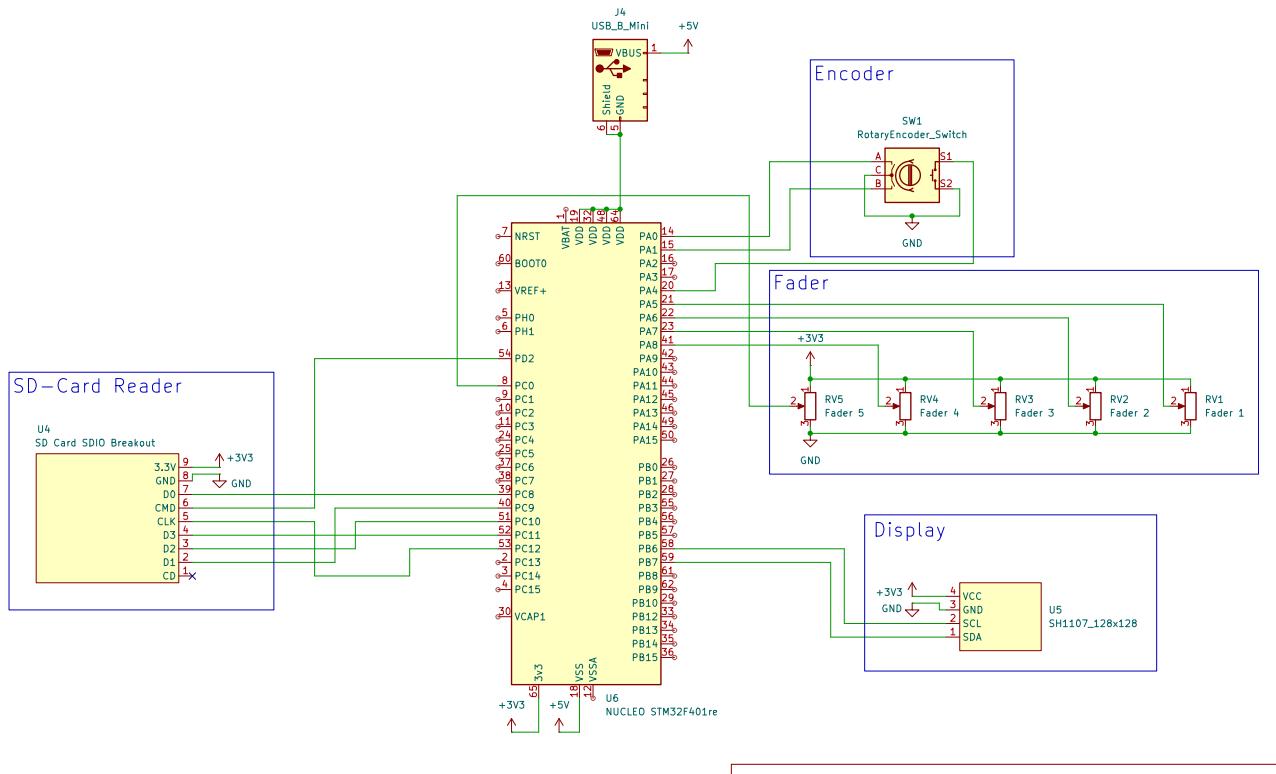


Abbildung 13: STM32 Nucleo-F722ZE Board

4.4.4 Schaltpläne



Interface Test Schaltplan



Sheet: /Interface/
File: interface.kicad_sch

Title:

KiCad EDA 8.0.1

4

Rev:
Id: 2/2

5 Durchführung

Im Anschluss an die Spezifikation des Systems folgt die Beschreibung der Umsetzung der einzelnen Komponenten. In diesem Abschnitt werden die angewandten Ansätze und Methodiken erörtert, die zur Realisierung der einzelnen Komponenten genutzt wurden. Weiterhin wird die Herangehensweise beschrieben, sowie wichtige Erkenntnisse und aufgetretene Probleme während der Implementierungsphase dokumentiert. Abschließend wird die Thematik der Gesamtintegration behandelt, einschließlich der Herausforderungen und der Gründe, warum eine vollständige Integration nicht erreicht wurde.

Da nur ein NUCLEO F7 Board (siehe Abschnitt 4.4.3) zur Verfügung stand, wurden die Komponenten (siehe Abschnitt 4.3) Audioverarbeitung und Interface im Zusammenspiel mit dem Dateimanagement auf dem, aus ES bereits bekannten, NUCLEO F401RE (siehe Abschnitt 4.4.3) Board entwickelt.

Lediglich die Komponente Klassifizierung wurde direkt auf dem F7 Board entwickelt, um direkt korrekte Einschätzungen über mögliche Performanceprobleme zu erlangen.

5.1 Repository Struktur

Das gesamte Projekt wurde in einem Git Repository auf dem Gitlab der TH Köln <https://gitlab.nt.fh-koeln.de/gitlab/jolux/ima> gespeichert und versioniert.

5.1.1 Code

Die Implementation der Komponenten erfolgt in abgekapselten Testprojekten. Planmäßig sollen diese nach Fertigstellung in ein Gesamtprojekt integriert werden (siehe Abschnitt 5.6).

/f401_sd_card_audio_codec_test/ Enthält die Implementation der Audio Engine auf dem F401 Board, sowie die DSP-Operationen welche für die Audiovorverarbeitung der Klassifizierung notwendig sind. Eine detaillierte Beschreibung gibt es in Abschnitt 5.2.

/f401_display_encoder_fader_test/ Enthält die Implementation des Interfaces auf dem F401 Board, sowie das Dateimanagement. Eine detaillierte Beschreibung gibt es in Abschnitt 5.4.

/f401_display_encoder_fader_audio_integration/ Enthält die Integration der Interface-, Dateimanagement- und Audioverarbeitungskomponente auf dem F401 Board. Die Integration ist bislang nicht gelungen, mehr dazu in Abschnitt 5.6.

/neuronal_network/ Enthält die Jupyter Notebooks für die Entwicklung des Neuronalen Netzes, sowie die Implementation der eingebetteten Klassifizierung (**/neuronal_network/esp_cubai_test**) und die Teilintegration der Klassifizierungskomponente (**/neuronal_network/esp_cubai_for_integration**). Mehr dazu im **Abschnitt 5.3.5** und im **Abschnitt 5.3.6**.

/neuronal_training/ Enthält das Python Script für die Lerndatengenerierung, mehr dazu in Abschnitt 5.3.2.

5.1.2 Dokumentation

/doc/ und alle Unterordner enthält die gesamte Projektdokumentation.

/doc/tex/ enthält die LaTeX Dokumente dieses PDFs.

/doc/schematics/ enthält Schaltbilder für die Teilprojekte: Audioverarbeitung, sowie Interface.

/doc/doxygen/ enthält Generierte Doxygen Code-Dokumentation.

/doc/datasheets/ enthält Alle verfügbaren Datenblätter der Hardwarekomponenten.

/doc/notes/ enthält Notizen die während des Projektablaufs entstanden sind.

5.1.3 Hardwaredesign

/hardware/ enthält Enthält das KiCad Projekt für die Schaltpläne.

5.1.4 Anderes

/praesi/ enthält die Powerpoint Projekte für die Milestone-Präsentationen.

/util/ enthält Nützliche Binaries (Zurzeit nur `riffpad.exe`, ein Tool zum Auslesen von .wav Headern).

5.2 Audio Engine des Samplers

Die Hauptfunktionalität eines Audiosamplers ist das Aufnehmen und Abspielen von Audiosamples.

Der gewählte PCM5102a Audio Codec verfügt lediglich über einen Ausgabestream, was eine Neubestellung eines Codecs mit In- und Output Stream erfordern würde.

Angesichts des ohnehin schon ambitionierten Featureumfangs und der damit verbundenen Zeitknappheit, wurde die Aufnahmefunktion **LF04** gestrichen, sodass der Prototyp zu einer reinen “Sample-Playback“ Maschine wird.

Das folgende Kapitel befasst sich mit der Implementierung und Ansteuerung der Audiowiedergabe über den Audio Codec und aller verbundenen DSP-Operationen.

Zunächst folgt eine Erläuterung des grundlegenden Signalfluss der Audioengine:

5.2.1 Signalfluss der Audioengine

Alle Audiosamples werden auf einer angeschlossenen **SD Karte** persistent gespeichert. Diese werden dann immer stückweise mit dem **FATFS** Dateisystem, welches für eingebettete Systeme optimiert ist, in die Applikationslogik und somit den Audiopufferspeicher geladen.

Von hier aus reicht die **DMA** den Buffer per **I2S** Protokoll an den **Audio Codec** weiter.

Der Audio Codec wandelt die digitalen PCM Signale in eine analoges Signal mit Line-Level ($\pm 1.7 \text{ V}_{\text{rms}}$).

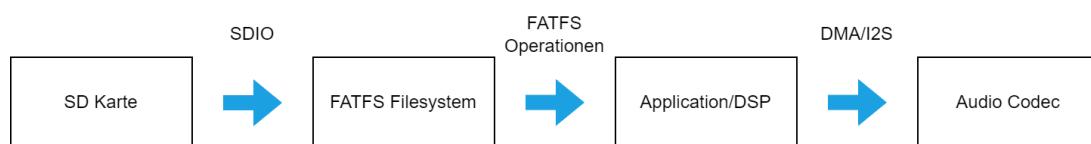


Abbildung 14: Digitaler Audio Signalfluss

5.2.2 Latenzen

Echtzeitfähigkeit ist ein kritischer Aspekt bei elektronischen Audioinstrumenten. Eine niedrige Latenz ist entscheidend, um rhythmisch präzise und “tight“ zu spielen, besonders wenn mehrere Instrumente miteinander synchronisiert werden müssen.

Die maximale Latenz, die ein menschlicher Profimusiker noch akzeptieren kann, liegt bei etwa 20ms. Diese Richtwert basiert auf der Wahrnehmungsgrenze, bei der Musiker und Live-Performern keine signifikante zeitliche Verzögerung zwischen dem Triggern eines Samples und dessen tatsächlichem Output am Ausgang spüren. [9]

Um die Latenz auf ein Minimum zu reduzieren und sicherzustellen, dass elektronische Audioinstrumente reaktionsschnell und synchronisiert sind, können folgende Methoden angewendet werden:

Dimensionierung der Buffersize

Die erste Einstellungsmöglichkeit ist die *Buffersize*. Das ist die Größe (in Bytes) des Audiobuffers. Je kleiner der Audiobuffer, desto öfter pro Sekunde gibt die DMA dessen Inhalt an den Audio Codec weiter.

Das bringt jedoch auch Performanceeinbußen mit sich: Bei kleinen Audiobuffern hat die CPU, je nach Komplexität der durchzuführenden DSP-Operationen, möglicherweise nicht genug Zeit, um den gesamten Buffer zu verarbeiten. Ein nur teils verarbeiteter Buffer kann hörbare Knackser und Störgeräusche am Ausgangssignal verursachen.

Hier gilt es also, die kleinstmögliche Buffersize zu ermitteln, ohne dass Störgeräusche auftreten. Nach Experimentieren hat sich ein Wert von **128 Bytes** als adäquat herausgestellt.

Verwendung von Double Buffering

Double Buffering ermöglicht es, Daten in einem Pufferspeicher zu verarbeiten, während gleichzeitig ein anderer Puffer für die Eingabe oder Ausgabe verwendet wird. Dies reduziert Verzögerungen und ermöglicht eine nahtlose Datenverarbeitung, da der Prozessor nicht auf das Ende einer Übertragung oder Berechnung warten muss, bevor er fortfahren kann. [10]

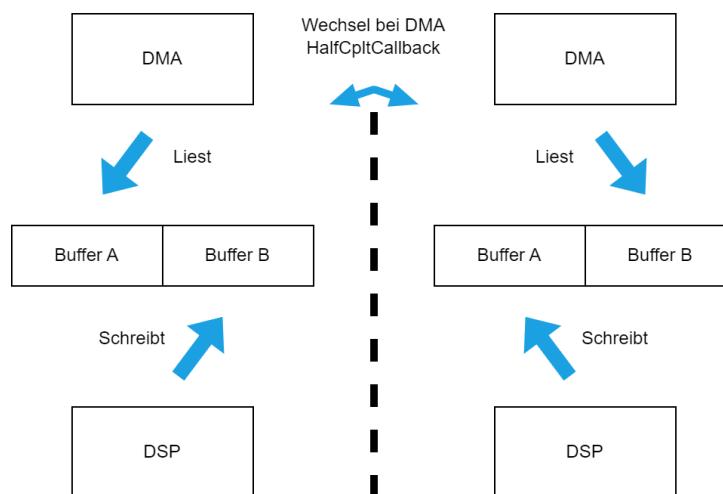


Abbildung 15: Double Buffering mit DMA und DSP

Bei I M A wird der Audiobuffer in zwei Hälften unterteilt. Durch den Pointer `int16_t *outBufPtr`, welcher jeweils auf die erste oder zweite Hälfte zeigt, wird mit der DSP die Hälfte bearbeitet, die gerade nicht von der DMA übertragen wird.

In den Callback-Funktionen `HAL_I2S_TxHalfCpltCallback()` und `HAL_I2S_TxCpltCallback()` wird der Pointer auf die jeweilige Hälfte des Buffers, also auf (wie in Abbildung 15 beschrieben: **Buffer A** und **Buffer B**) gesetzt.

```

24 void initSineTable() {
25     for (int i = 0; i < SINE_TABLE_SIZE; ++i) {
26         sineTable[i] = (int16_t)(32767 * sin(2.0 * M_PI * i / SINE_TABLE_SIZE));
27     }
28 }

64 void generateSineWave(double frequency) {
65     static uint16_t phaseIndex = 0;                                /*< Retains the index between
66     → function calls */
67     double phaseIncrement = frequency * SINE_TABLE_SIZE / I2S_AUDIOFREQ_44K;
68     HAL_I2S_Transmit_DMA(&hi2s2, (void *)dacData, BUFFER_SIZE);

69     while(1){
70         if(dma_dataReady){
71             for (uint8_t n = 0; n < (HALF_BUFFER_SIZE) - 1; n += 2) {
72                 // Lookup sine value from table
73                 outBufPtr[n] = sineTable[phaseIndex];
    
```

```

74
75      // Increment phase index
76      phaseIndex += phaseIncrement;
77
78      // Wrap phase index if it exceeds table size
79      if (phaseIndex >= SINE_TABLE_SIZE) {
80          phaseIndex -= SINE_TABLE_SIZE;
81      }
82      dma_dataReady = false;
83  }
84 }
85 }
86 }
```

Diese Callbacks, werden systembedingt aufgerufen, sobald die DMA die erste Hälfte oder den gesamten Buffer übertragen hat. Sie eignen sich daher sehr gut um den Pointer zu setzen.

Sobald das Flag `dma_dataReady == true` gesetzt ist, wird die nächste Bufferhälfte von der SD-Karte gelesen.

Latenzmessung Die Latenzmessung im Abschnitt 6.7 zeigt, dass die gemessene Latenz von Trigger Input bis Audioausgang mit 6.7 ms sehr zufriedenstellend ist, was ein erfolgreiches Zusammenspiel mit anderen elektronischen Instrumenten ermöglicht.

5.2.3 PCM5102a und I2S

Ein sehr weit verbreitetes Protokoll zur digitalen Audioübertragung ist I2S. Viele Codecs, so auch der PCM5102a, unterstützen dieses Protokoll.

I2S (Inter-IC Sound) ist ein Standard für die digitale Übertragung von Audiodaten zwischen integrierten Schaltkreisen (ICs). Es wurde entwickelt, um die Kommunikation von digitalen Audiodaten zwischen verschiedenen Komponenten wie Mikrofonen, DACs (Digital-Analog-Wandler) und ADCs (Analog-Digital-Wandler) zu ermöglichen. [11]

Der PCM5102a erkennt komfortablerweise die eingehende Samplerate anhand der Bitclock. Anders als bei vielen Audio Codecs ist keine zusätzliche Konfiguration des Chips über ein Kommunikationsprotokoll, wie I2C notwendig.

Somit fallen auch keine Treiber für diesen Chip an, was die Einbindung sehr vereinfacht.

Die I2S Signale des PCM5102a sind:

- **SCK (System Clock):** Dieser Takt wird für den Betrieb des internen Digitalfilters und des DAC-Kerns verwendet. Er kann entweder extern bereitgestellt oder intern vom PCM5102a erzeugt werden.
- **BCK (Bit Clock):** Dieser Takt signalisiert den Beginn jedes Bits im Datenwort. Der BCK wird verwendet, um die Daten auf der Datenleitung (DIN) zu takten.
- **LCK (Left/Right Clock):** Auch als Word Select oder Frame Sync bekannt. Dieser Takt signalisiert den Beginn eines neuen Audioframes und zeigt an, ob die aktuellen Daten den linken oder rechten Kanal darstellen.
- **DIN (Digital Input):** Dies ist die Datenleitung, über die die PCM-Audiodaten übertragen werden.

Der Einfachheit halber wird eine festgelegte Samplerate und nur Stereo-Samples verwendet.

5.2.4 Audiostream von SD-Karte

Durch den sehr begrenzter RAM-Speicher des NUCLEO F401re Boards , ist es notwendig die Daten in Echtzeit von der SD-Karte zu streamen. Das setzt bei Stereo PCM Dateien, mit den gängigen Parametern, eine bestimmte Datenrate R voraus:

- Abtastrate (Sampling Rate): 44.100 Hz (44,1 kHz)
- Bit-Tiefe: 16 Bit
- Kanäle: 2 (Stereo)

Die Datenrate R berechnet sich aus:

$$R = \text{Sampling Rate} \times \text{Bit-Tiefe} \times \text{Anzahl der Kanäle}$$

$$R = 44,100 \text{ Hz} \times 16 \text{ bit} \times 2$$

$$R = 1,411,200 \text{ bit/s} \approx 0.168 \text{ MB/s}$$

Theoretisch hätte eine Implementierung des FATFS-Dateisystems über SPI ausreichen müssen, um diese Datenraten zu bewältigen.

Um die SDIO-Schnittstelle in das FATFS-Dateisystem zu integrieren, mussten zunächst Treiber entwickelt werden. Diese Treiber mappen die üblichen Dateioperationen wie `f_open()`, `f_read()` usw. über die SPI-Schnittstelle.

In der Praxis stellte sich jedoch heraus, dass der Audiobuffer nicht schnell genug gefüllt worden ist, was starke Knackser und Störgeräusche mit sich gebracht hat.

Dies erwies sich als äußerst vorteilhaft, da die SDIO-Implementierung zusammen mit FATFS direkt im Konfigurationstool von STM, **CubeMX**, ausgewählt werden kann. Die benötigten Treiber werden von CubeMX automatisch generiert und integrieren sich problemlos in das Dateisystem.

5.2.5 Playback und Pitched Playback

Der Audioplayer unterstützt das Abspielen von WAV-Dateien mit Tonhöhenanpassung. Dies wird durch die Verwendung von DMA zur Übertragung über I2S erreicht. Die Audiomeldung erfolgt mit dynamischer Anpassung der Abspielgeschwindigkeit, was eine Echtzeit-Tonhöhenänderung ermöglicht.

Die Funktion `wavPlayPitched(WavPlayer *player)` liest Audiodaten von der SD-Karte:

```
1 bytesRead = fillHalfBufferFromSD(player, true);
```

wobei die Funktion `fillHalfBufferFromSD()`, die Anzahl der benötigten Samples für den Buffer, dynamisch anhand der Abspielgeschwindigkeit errechnet.

Pitchfaktor Ein Pitchfaktor größer als 1.0 bedeutet, dass die Wiedergabegeschwindigkeit erhöht wird. Um diese höhere Geschwindigkeit zu kompensieren, müssen mehr Samples in den Puffer geladen werden. Das liegt daran, dass mehr Daten pro Zeiteinheit benötigt werden, um die schnellere Wiedergabe zu unterstützen.

Ein Pitchfaktor kleiner als 1.0 reduziert die Wiedergabegeschwindigkeit. In diesem Fall werden weniger Samples benötigt, weil sich die Daten im Puffer wiederholen. Das bedeutet, dass der Puffer nicht so schnell gefüllt werden muss, da die Audio-Daten langsamer abgespielt werden.

PCM-Stereocodierung Beim Füllen des Audiobuffers mit einem Pitchfaktor $\neq 0$ muss darauf geachtet werden, dass die PCM Stereo-Codierung korrekt beachtet wird. Im PCM Stereo-Format repräsentieren alle geraden Samples im Buffer den linken Kanal, während die ungeraden Samples den rechten Kanal darstellen.

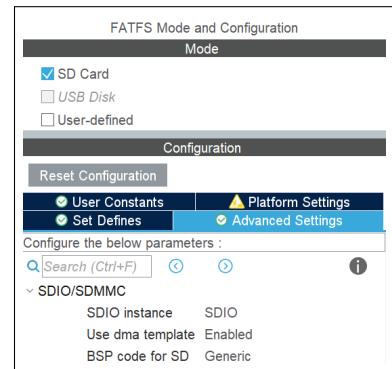


Abbildung 16: CubeMX FATFS SDIO Einbindung

Wenn der Pitchfaktor verändert wird, können Samples entweder übersprungen oder doppelt abgespielt werden. Dabei ist es entscheidend, dass die richtigen Samples für den linken und rechten Kanal ausgewählt werden. Der folgende Code sorgt dafür, dass diese Auswahl korrekt erfolgt:

```
1  leftReadIndex = (leftReadIndex + 1) & ~1;
2  rightReadIndex = leftReadIndex + 1;
```

Hier ist eine detaillierte Erklärung dieser Operationen:

1. Sicherstellung eines geraden Indexes:

Der Code:

```
leftReadIndex = (leftReadIndex + 1) & ~1;
```

sorgt dafür, dass `leftReadIndex` immer eine gerade Zahl ist. Zunächst wird 1 zu `leftReadIndex` addiert, um eine gerade Zahl zu erhalten.

Die bitweise UND-Operation mit `~1` (der negierten Zahl 1) entfernt das niedrigstwertige Bit von `leftReadIndex + 1`, wodurch der Wert zu einer geraden Zahl wird.

Zum Beispiel wird aus 5 (ungerade) 6 (gerade), und aus 7 (ungerade) wird ebenfalls 6 (gerade).

2. Berechnung des `rightReadIndex`:

Der Index für das rechte Sample wird durch die folgende Zeile berechnet:

```
rightReadIndex = leftReadIndex + 1;
```

Nachdem `leftReadIndex` als gerade Zahl festgelegt wurde, ist `rightReadIndex` automatisch der nächste Index, der ungerade ist. Dies entspricht dem Format von PCM Stereo-Samples, bei dem gerade Indizes für den linken Kanal und ungerade Indizes für den rechten Kanal verwendet werden.

Diese Schritte stellen sicher, dass die berechneten Indizes korrekt auf die im Stereo-PCM-Format erwarteten Positionen verweisen, indem sie sicherstellen, dass der Puffer die richtigen Daten für den linken und rechten Kanal enthält.

5.2.6 Downsampling eines Audiochunks für die Verarbeitung von Audio durch das Neuronalen Netzes

Damit die proprietäre “STM32_AI_AudioPreprocessing_Library“ (Abschnitt 5.3.6) die Audiodaten in Spektrogramme umwandeln kann, müssen die Daten zunächst ins korrekte Format gebracht werden. Für die Umwandlung erwartet die Library 16 kHz Samplerate, Mono Dateien.

In der Datei `/f401_sd_card_audio_codec_test/Core/Src/audioPreprocessor.c` finden sich alle Funktionen, die das Audiopreprocessing der Audiodatei für die Klassifizierung betreffen.

Die Funktion

```
1  uint32_t downsample_to_1024_samples(FIL *file, int16_t
→   outChunk[NUM_SAMPLES_CHUNK_OUT])
```

liest einen Audiochunk aus einer Datei von der SD Karte, konvertiert die Stereo-Samples in Mono und führt anschließend eine Abtastratenreduktion (Downsampling) durch. Die verarbeiteten Samples werden im `outChunk`-Buffer gespeichert. Die Funktion arbeitet blockweise und gibt die Anzahl der heruntergesampelten Samples zurück.

Zunächst werden die Eingabedaten in den Puffer `inputChunk` eingelesen. Die Stereo-Samples werden durch Mittelung der linken und rechten Kanäle in Mono umgewandelt und in den Puffer `currentIn-BlockMono` geschrieben. Anschließend wird die Funktion `downsample_Block` aufgerufen, die die eigentliche Abtastratenreduktion und FIR-Filterung durchführt.

Die Funktion `downsample_Block` konvertiert die 16-Bit-Integer-Audiodaten in ein Fließkommaformat, führt FIR-Filterung und Dezimierung durch und konvertiert die gefilterten Samples zurück in das 16-Bit-Integer-Format. Die verarbeiteten Daten werden dann in den Zielpuffer geschrieben.

Um Aliasing beim heruntergesampleten Audiomaterial zu vermeiden, werden die benötigten Samples vor dem Downsampling mit einem FIR Highcut geglättet. Die Koeffizienten können mit Filterdesign-Softwares, wie <http://t-filter.engineerjs.com> generiert werden. Es wurde ein Filter mit 57 Koeffizienten gewählt, der eine gute Balance zwischen Performance und Ripple bietet. Wie in Abbildung 17 zu sehen, bietet der Filter ab 16 kHz eine recht steile Flanke bis –45 dB Abschwächung.

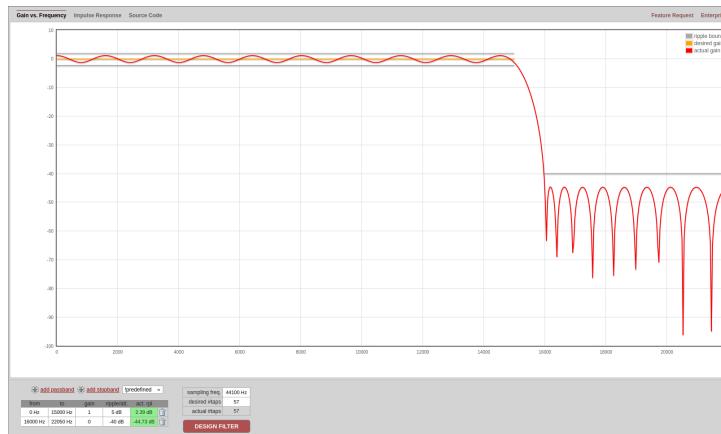


Abbildung 17: Anti-Aliasing FIR Lowpass Filter

Die Open Source CMSIS DSP Library [12], für ARM Architekturen, bietet direkt eine optimierte Funktion, welche die FIR-Filterung, sowie das eigentliche Downsampling kombiniert: `arm_fir_decimate_f32`. Diese Funktion akzeptiert jedoch nur ganze Downsampling-Faktoren.

Der benötigte Downsampling-Faktor ist:

$$\frac{44,100 \text{ Hz}}{16,000 \text{ Hz}} = 2,75625$$

Daher wird auf 3.0 aufgerundet, um einen ganzzahligen Faktor zu erreichen. Die resultierende neue Samplerate des heruntergesampleten Samples beträgt:

$$\frac{44,100 \text{ Hz}}{3.0} = 14,700 \text{ Hz}$$

5.3 Klassifizierung von Audiodateien durch ein neuronales Netz

Durch das neuronale Netz sollen digitale Audiodateien (Audiosamples) mit variabler Länge in fünf Merkmalsklassen klassifiziert werden. Dabei kann ein Datenpunkt, also ein Audiosample, mehreren Merkmalsklassen zugehörig sein. Dies wird als “Multi-Label-Klassifizierung“ bezeichnet [13]. Wird eine Audiodatei (Eingabevektor) zur Klassifizierung in das neuronale Netz eingegeben, erhält man als Ausgabe einen Ausgabevektor von fünf Werten zwischen 0.00 und 1.00, wobei jeder der fünf Werte die Ähnlichkeit mit einer der fünf Merkmalsklassen repräsentiert (0.00 \equiv keine Ähnlichkeit, 1.00 \equiv Übereinstimmung). Die Merkmalsklassen werden wie folgt definiert:

- **bass:** Das Audiosample enthält Töne aus dem tiefen Frequenzspektrum
- **pitched:** Das Audiosample enthält Töne aus dem hohen Frequenzspektrum
- **melodic:** Das Audiosample enthält melodische Elemente
- **rhythmic:** Das Audiosample enthält rhythmische Elemente
- **sustained:** Das Audiosample enthält “flächige“ Elemente, also “langgezogene“ Töne

Dieses neuronale Netz wird mit selbst generierten Trainingsdaten trainiert und anschließend als fertiges Modell mittels dem STM32 proprietären Tool “STM32Cube.AI“ in C-Code umgewandelt. Damit kann das Neuronale Netz im eigenen Code als normale C-Funktion verwendet werden, mit dem Eingabevektor (Audiodaten) als Funktionsparameter und dem Ausgabevektor (Klassifikationsergebnis) als Rückgabewert [14].

Die Umwandlung in C-Code ist möglich, da ein künstliches Neuron, wie in **Abbildung 18** dargestellt, aus aus mehreren Gewichten (W_x) in Form eines Vektors besteht, der mit dem Eingabevektor (X_n) des Neurons multipliziert wird. Beide Vektoren sind Fließkommazahlen. Anschließend werden diese Produkte aufaddiert und als Eingabewert einer mathematischen Funktion, der sog. “Aktivierungsfunktion“ (f) verwendet. Der Ausgabewert dieser Funktion ist die Ausgabe des Neurons. [15]

Bei einem neuronalen Netz werden die Neuronen in Schichten hintereinander angeordnet, die Neuronen der benachbarten Schichten werden vereinfacht gesagt miteinander verbunden, also die Eingabe eines Neurons bildet eine der Eingaben des nachfolgenden Neurons. [15]

Sowohl die Multiplikation von Vektoren, als auch die Berechnung von Funktionswerten ist in der Programmiersprache C problemlos möglich.

Der ressourcenintensive Teil des Umgangs mit neuronalen Netzen ist das Training, also der Anpassung der Gewichtswerte bis das Neuronale Netz akzeptable Ausgabewerte liefert [15]. Bei diesem Prozess müssen vergleichsweise sehr viele Berechnungen durchgeführt werden. Aus diesem Grund wird das Neuronale Netz zuerst trainiert und erst dann als fertiges Modell in C-Code umgewandelt und auf dem STM32 Microcontroller betrieben. Dass das Neuronale Netz fortlaufend durch Benutzerinteraktion weiter trainiert wird, ist nicht vorgesehen.

5.3.1 Ansatz für die Klassifizierung von Audiodaten

Die Audiosamples liegen als WAVE-Dateien (Dateiendung .wav) vor. Diese enthalten in der Regel pulsweitenmodulierte (PCM) Audiodaten [16]. Da die Daten nicht komprimiert sind, gehören sie in der Audio- und Musikindustrie zu den gängigsten Dateiformaten [17]. Eine typische Samplerate für WAVE-Dateien beträgt 44,1 kHz, also 44.100 Samples pro Sekunde, wobei ein Sample in der WAVE-Datei ein quantierter Amplitudenwert zu einem bestimmten Zeitpunkt ist [18]. Plottet man dies als Graph, könnte ein Audiosignal wie in **Abbildung 19** aussehen.

Diese Daten direkt durch ein neuronales Netz klassifizieren zu lassen, ist aus verschiedenen Gründen wenig praktikabel. Die zwei Hauptgründe sind die Merkmalsextraktion und die Größe des Eingabevektors.

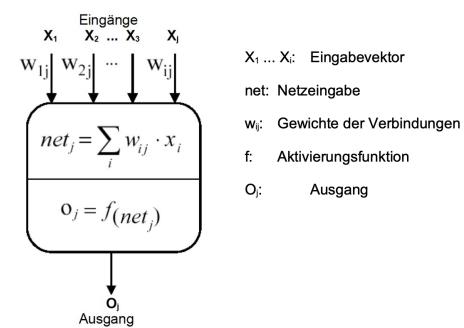


Abbildung 18: Aufbau des künstlichen Neurons. Quelle: Thieling, Lothar: “Neuronale Netze (Vorlesungsskript ML)”, Kapitel F, Seite 6.

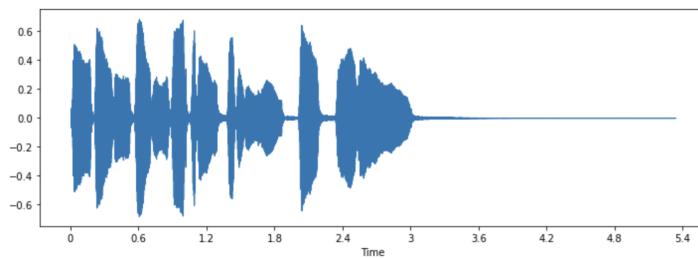


Abbildung 19: Darstellung eines PCM Audiosignals [19]

Letzterer Punkt würde dazu führen, dass selbst bei einer geringeren Samplerate von 16 kHz der Eingabevektor 16.000 Werte umfasst. Ein Downsampling auf eine deutlich geringere Zahl, z.B. 1000, ist aufgrund des Shannon-Nyquist-Theorems nicht praktikabel, da dieses besagt, dass die Samplerate mindestens doppelt so hoch sein muss wie die höchste Frequenz [20]. Damit läge der klassifizierbare Frequenzbereich nur im Bereich von 0 Hz bis maximal $1000 / 2 = 500 \text{ Hz}$.

Deutlich geeigneter ist es, die Daten als Spektrogramm (Amplitude der verschiedenen Frequenzen eines Signals über die Zeit) wie in **Abbildung 20** darzustellen und mit einem Convolutional Neural Network (CNN) zu klassifizieren. CNNs werden in erster Linie zur Klassifizierung von Bildern eingesetzt. Die wesentliche Idee ist, dass das neuronale Netz dann nicht nur klassifiziert, sondern auch die Bildvorverarbeitung und die Merkmalsextraktion übernimmt [21]. Da die Audiodateien für das menschliche Gehör klassifiziert werden, sind Mel-Spektrogramme besonders geeignet. Sie basieren auf der Mel-Skala, die das menschliche Gehör nachahmt. Die Mel-Skala ist eine nichtlineare Skala der Frequenzen, die mehr Gewicht auf tiefere Frequenzen legt [22].

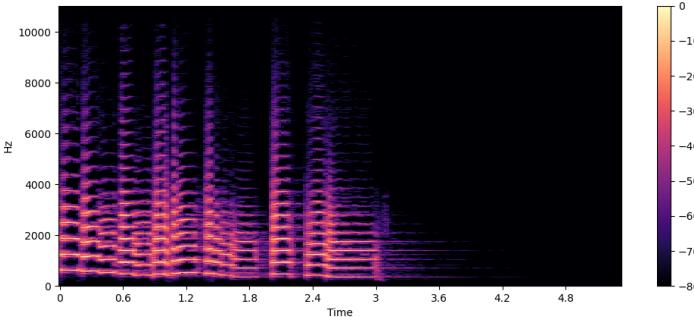


Abbildung 20: Darstellung eines Audiosignals als Spektrogramm [19]

Der limitierende Faktor beim Einsatz eines CNN auf einem Microcontroller sind die Hardwareressourcen, also in erster Linie Flash-Speicher und RAM.

Aufgrund der fehlenden Erfahrungswerte der Teammitglieder mit der möglichen Komplexität von neuronalen Netzen, die mit den Hardwareressourcen eines Microcontrollers betrieben werden können, wird sich auf ein Beispielprojekt von STMicroelectronics gestützt. Dieses heißt "Acoustic Scene Classification" [23][24]. Kern ist die Klassifizierung von 30x32 px großen Mel-Spektrogrammen mit einem zwei Schichten CNN, die einen Eingabevektor von $32 \times 30 = 960$ Werten ergeben.

Sowohl die Dimensionierung der Spektrogramme, als auch die Topologie des Neuronalen Netzes, die Anzahl der Neuronen je Schicht und Elemente der Datenvorverarbeitung wurden aus diesem Projekt übernommen. Damit ist sicher gestellt, dass das Neuronale Netz am Ende nicht die Ressourcen des Microcontrollers übersteigt.

Um die in **Abschnitt 5.3** beschriebenen Merkmalsklassen zu definieren, wurden zunächst Kategorien überlegt, die für die Musikproduktion relevant sind. Anschließend wurden Spektrogramme dieser Klassen in der vorgesehenen Auflösung erstellt und analysiert. Ziel war es zu prüfen, ob das menschliche Auge bzw. Gehirn in der Lage ist, die Merkmalsklassen anhand der Spektrogramme zu differenzieren. Diese Überprüfung diente dazu, die grundsätzliche Machbarkeit der Klassifikation in die Merkmalsklassen durch das neuronale Netz zu bestätigen.

5.3.2 Generieren der Eingabedaten

Für das Training des neuronalen Netzes sind Trainingsdaten erforderlich. Diese bestehen aus Eingabedaten, die bereits mit den korrekten Klassifikationsergebnissen, also Labels, versehen sind. Um das Training des neuronalen Netzes während des Trainings einschätzen und nach Abschluss des Trainings validieren zu können, werden die Eingabedaten in drei Segmente aufgeteilt.

Da online keine kostenlosen und bereits mit den passenden Labels versehene Datensätze gefunden werden konnten, wurden eigene Daten generiert. Grundlage hierfür bildet eine private Audiosample-Bibliothek. Ein eigens entwickeltes Python-Skript ermöglicht es, Audiosamples auszuwählen, abzuspielen und zeiteffizient zu labeln. Ein Screenshot des Labeling-Prozesses mit dem Script ist in **Abbildung 21** abgebildet.

```

Sweet Bass - 6A.wav | Tags: | Status:
Your Bass - 9A.wav | Tags: | Status:
Null Bass - 10B.wav | Tags: | Status:
Times Bass - 7A.wav | Tags: pitched, sustained, rhythmic | Status: Added
Aura Bass - 9B.wav | Tags: | Status:
Will They Hi Bass - 5B.wav | Tags: | Status:
Shoes Bass - 5A.wav | Tags: | Status:
Calm Bass - 8A.wav | Tags: | Status:
Wheel Up Sub.wav | Tags: | Status:
Logik Bass - 9A.wav | Tags: | Status:
Drumz Reese.wav | Tags: | Status:
Pressin Sub - 4A.wav | Tags: | Status:
Bouncer Bass - 2B.wav | Tags: | Status:
Illicit Bass - 9B.wav | Tags: | Status:
Beta Bass - 11A.wav | Tags: | Status:
> Chain Bass 2 - 9A.wav | Tags: | Status:
Dubplate Bass - 5A.wav | Tags: | Status:
Will They Bass.wav | Tags: | Status:
Night Bass - 1A.wav | Tags: | Status:
Meltdown Bass - 9A.wav | Tags: melodic | Status: Added
Who Run Bass - 8A.wav | Tags: | Status:
Ganga Bass - 1A.wav | Tags: | Status:
Town 808 Bass - 1A.wav | Tags: rhythmic | Status: Added
Drumz Sub - 3B.wav | Tags: | Status:
Type Bass - 4B.wav | Tags: rhythmic | Status: Added
Gang Sub.wav | Tags: | Status:
Terrorist Reese - 3A.wav | Tags: bass, pitched | Status: Added
Moomin 808 - 5A.wav | Tags: | Status:

```

Abbildung 21: Python-script zum manuellen Auswählen Labeln der Trainingsdaten

Für jedes manuell gelabelte Audiosample wird eine eindeutige Identifikationsnummer (UID) generiert. Die zugehörigen Labels werden in einer CSV-Datei gespeichert. Ein Beispiel für einen solchen Datensatz zeigt **Tabelle 6**. Außerdem wird eine Kopie des Audiosamples unter dem Namen der UID im Ausgabeordner des Skripts abgelegt. Diese Dateien werden später vom Jupyter-Notebook für das Training verwendet.

UID	File	bass	pitched	sustained	rhythmic	melodic
ecfad96b740844c3 9c96127279f22cf6	BD 606 Long MPC60.wav	1	0	0	1	0

Tabelle 6: Beispielhafter Datensatz eines Audiosamples

Bei der Auswahl der Audiosamples wurde darauf geachtet, dass die Daten annähernd gleich verteilt sind, um eine ausgewogene Trainingsbasis zu schaffen. Ein Ungleichgewicht in den Daten kann sich negativ auf das Training und die Klassifikationsergebnisse auswirken. Überrepräsentierte Merkmale könnten die Klassifikationsschwellenwerte beeinflussen, sodass die häufiger vorkommenden Klassen später bevorzugt erkannt werden [25]. Wie in **Abbildung 22** zu sehen, ist die Gleichverteilung jedoch zugegebenermaßen nur begrenzt gelungen, mit deutlich überrepräsentierten Eingabedaten mit dem Merkmal "melodic".

Zudem wurde versucht sicherzustellen, dass der Variationsbereich jeder Klasse möglichst umfassend abgedeckt ist. Dies umfasst sowohl reine Formen jeder Klasse – beispielsweise bei „pitched“ Audiosamples ausschließlich mit Tönen aus dem hohen Frequenzbereich – als auch Mischformen, die Merkmale mehrerer Klassen kombinieren.

Insgesamt wurden 209 Audiosamples unterschiedlicher Länge mit Labels versehen.

5.3.3 Datenvorverarbeitung

Für die Datenvorverarbeitung und das Training des neuronalen Netzes wird ein Jupyter-Notebook verwendet. Dieses ermöglicht eine Kombination aus formatiertem Text und Code, in diesem Fall Python, was die Lesbarkeit und Nachvollziehbarkeit des Codes erleichtert.

Damit die gelabelten Daten für das Training des neuronalen Netzes verwendet werden können, müssen

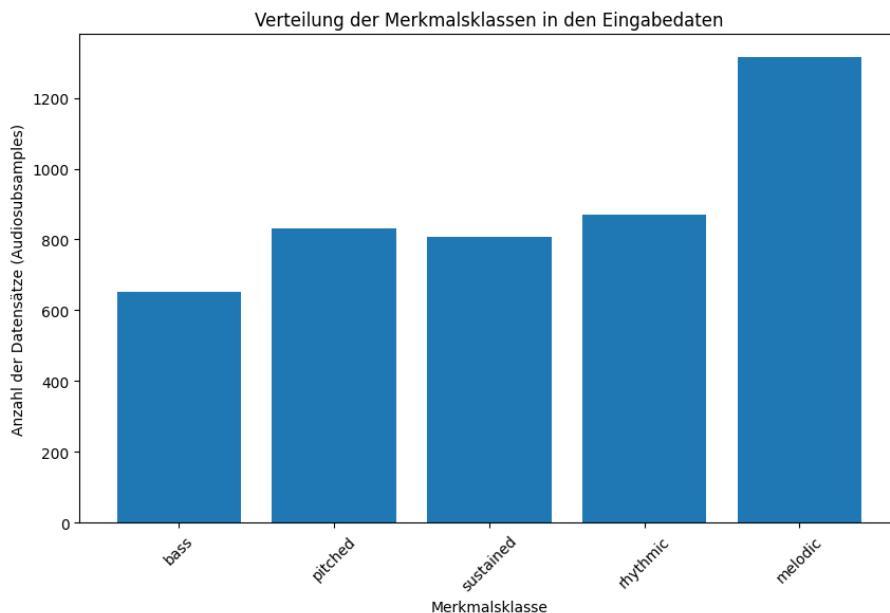


Abbildung 22: Repräsentation der verschiedenen Merkmalsklassen in den Entwicklungsdaten

sie zunächst aufbereitet bzw. vorverarbeitet werden. Um die zu verarbeitenden Datenmengen sinnvoll zu verringern, wird zunächst die Samplerate aller Audiosamples auf 16 kHz reduziert.

Wie im **Abschnitt 5.3.2** erwähnt, liegen die Audiosamples als WAVE-Dateien unterschiedlicher Längen vor. Die in **Abschnitt 5.3.1** beschriebenen Dimensionen der durch das neuronale Netz klassifizierbaren Mel-Spektrogramme betragen 32x30 Pixel. Um bei unterschiedlich langen Audiosamples die zeitliche Abhängigkeit zu bewahren und auch bei längeren Audiosamples wichtige Informationen im Spektrogramm abzubilden, müssen die Audiosamples in Sektionen gleicher Länge unterteilt werden, wobei später aus jeder Sektion ein Spektrogramm erstellt wird. Diese Sektionen werden im Folgenden als „Audiosubsamples“ bezeichnet.

Aus den in **Abschnitt 5.3.2** erwähnten 209 gelabelten Audiosamples wurden durch deren Aufteilung insgesamt 2.837 Audiosubsamples gewonnen.

Ein Audiosubsample, also ein Mel-Spektrogramm, entspricht dabei 16.896 Samples, was bei einer Samplerate von 16 kHz etwas mehr als einer Sekunde entspricht. Analog zum Beispielprojekt „Acoustic Scene Classification“ [23][24] werden die Audiosubsamples anschließend mittels eines „Sliding Window“ in 1024 Samples lange, um 512 Samples überlappende, Frames unterteilt. Diese Überlappung stellt sicher, dass Informationen zu den Anfangs- und Endzeitpunkten der Frames nicht durch das „Abschneiden“ verloren gehen. Aus den 16.896 Samples eines Audiosubsamples lassen sich dadurch 32 Frames gewinnen. Die Aufteilung der Audiosamples in Audiosubsamples und Frames wird in **Abbildung 23** visualisiert.

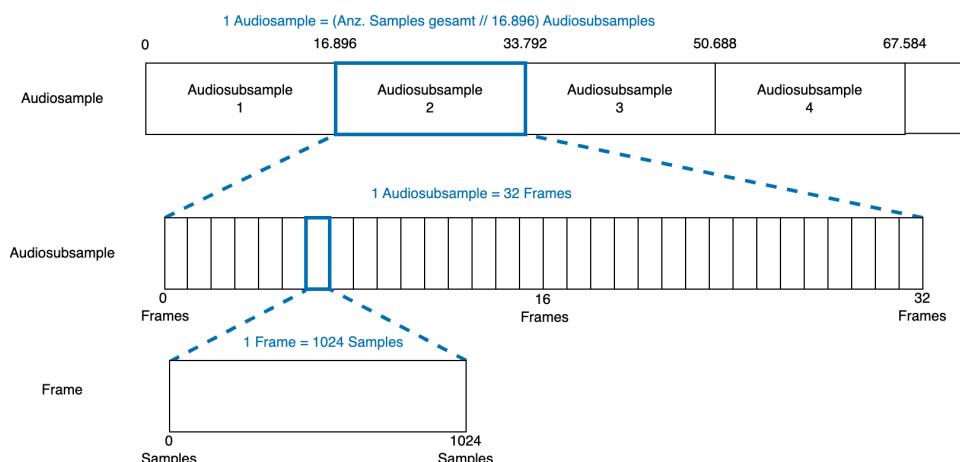


Abbildung 23: Darstellung der Unterteilung der Audiosamples in Audiosubsamples und Frames

Das Sliding Window Prinzip, mit dem Audiosubsamples in überlappende Frames unterteilt werden, ist in **Abbildung 24** dargestellt.

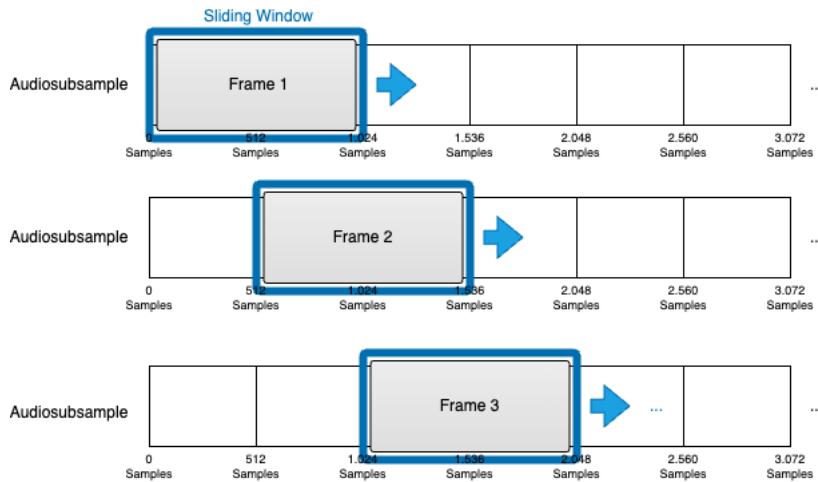


Abbildung 24: Darstellung des Sliding Window Prinzips

Nun können die Mel-Spektrogramme erstellt werden. Eine Spalte des Spektrogramms wird dabei aus einem Frame berechnet. Setzt man die 32 in einem Audiosubsample enthaltenen Frames, also Spektrogramm-Spalten, zusammen, erhält man das 32x30 Pixel große Spektrogramm für ein Subsample.

Abschließend müssen die erhaltenen Daten standardisiert, also mittelwertfrei gemacht werden. Dies ist bei maschinellen Lernen üblich und trägt dazu bei, dass das Modell schneller und effizienter konvergiert. Durch die Reduzierung der Varianz in den Eingabedaten und die Sicherstellung einer gleichmäßigen Verteilung der Werte verbessert sich die Modellleistung [26]. Um diese Standardisierung auch später beim Einsatz des neuronalen Netzes auf dem Mikrocontroller nachzuahmen, werden die Werte des Scalers, der zur Berechnung der standardisierten Werte verwendet wird, exportiert. Bei den Werten des Scalers handelt es sich um Fließkommazahlen, jeweils eine für jeden Eingabevektorwert. Diese Werte werden dann später mit den Eingabevektorwerten multipliziert, um den standardisierten Wert zu simulieren.

5.3.4 Aufteilen der Eingabedaten in Trainings- Test und Validierungsdaten

Wie in **Abschnitt 5.3.2** beschrieben, wurden 209 Audiosamples mit Labeln versehen. Diese in 2.837 Audiosubsamples aufgeteilt. Für das Training des neuronalen Netzes werden die Daten wie üblich in Trainingsdaten, Validierungsdaten und Testdaten aufgeteilt [27].

Trainingsdaten (1.701 Datensätze) werden direkt für das Training des neuronalen Netzes genutzt.

Validierungsdaten (568 Datensätze) dienen dazu, den Fortschritt und Erfolg des Trainings über die verschiedenen Durchläufe (Epochen) hinweg zu überwachen.

Testdaten (568 Datensätze) werden nach dem Abschluss des Trainingsvorgangs eingesetzt, um die Genauigkeit und Zuverlässigkeit der Klassifikation zu bewerten.

Eine Visualisierung der Datenaufteilung ist in **Abbildung 25** dargestellt.

Hierbei muss festgehalten werden, dass die Datenaufteilung einen methodischen Fehler enthält, der erst zu spätem Zeitpunkt aufgefallen ist und nicht mit geringem Zeitaufwand behoben werden konnte. Die einzelnen Audiosamples werden wie zuvor beschrieben erst in Audiosubsamples aufgeteilt und dann randomisiert auf die drei Kategorien aufgeteilt. Der Gedanke hierbei war, dass so auch bei stark unterschiedlicher Länge der Audiosamples gewährleistet ist, dass jede Kategorie immer eine bestimmte Anzahl an Audiosubsamples enthält. Ein Nebeneffekt davon ist allerdings, dass dadurch Audiosubsamples der gleichen Datei in allen drei Kategorien verteilt sein können. Da es nahe liegt, dass sich Audiosubsamples der gleichen Datei immer ähneln, wird dadurch die Genauigkeit der Klassifizierung ggf. verfälscht dargestellt.

5.3.5 Training des neuronalen Netzes

Nach der Datenvorverarbeitung beginnt das eigentliche Training des neuronalen Netzes. Typischerweise bestehen Convolutional Neural Networks (CNNs) aus mehreren Faltungsschichten (Convolutional Layers), die der Vorverarbeitung der Eingabedaten und der Merkmalsextraktion dienen. Die Faltungsmasken, die

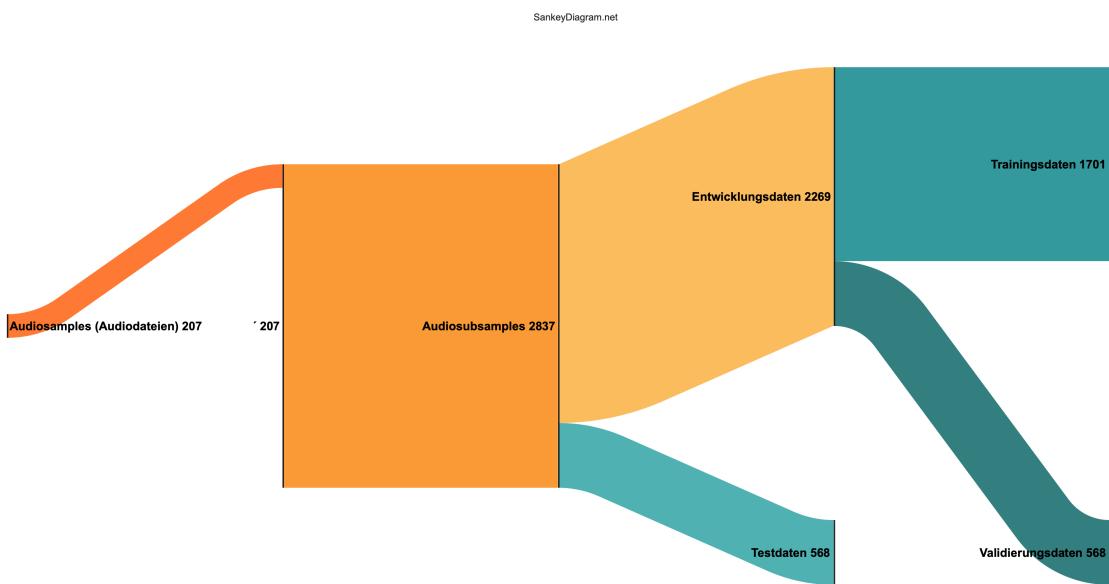


Abbildung 25: Aufteilung der Eingabedaten in Trainings-, Validierungs- und Testdaten (erstellt mit sankeydiagram.net)

über das Eingabebild gefaltet werden, stellen die Neuronen dar, die trainiert werden. Jede Faltungsschicht wird in der Regel von einer Pooling-Schicht (Pooling Layer) gefolgt, welche die Datenmenge reduziert und die Dimensionalität des Ausgabebildes sinnvoll verkleinert. [21]

“Normale“ neuronale Netze, oft als “Dense“ oder “Fully Connected Networks“ bezeichnet, können nur Vektoren verarbeiten. Eine sogenannte “Flattening“-Schicht wandelt die zweidimensionale Ausgabematrix des CNNs in einen eindimensionalen Vektor um. Nach dieser Transformation übernehmen zwei “Dense-Layers“ die eigentliche Klassifikation. An der Ausgabe des zweiten Dense-Layers, die einen Vektor mit fünf Werten zurückgibt, kann das Klassifikationsergebnis abgelesen werden. [21]

Im folgenden Codeausschnitt des Jupyter-Notebooks wird die Struktur des neuronale Netzes definiert. Die Parameter der `add`-Funktion beziehen sich auf die Art der Schicht, der Anzahl bzw. Dimensionierung der Ausgabematrix und die Aktivierungsfunktion.

```

1 model = models.Sequential()
2 model.add(layers.Conv2D(16, (3, 3), activation='relu',
3     input_shape=(30, 32, 1), data_format='channels_last'))
4 model.add(layers.MaxPooling2D((2, 2)))
5 model.add(layers.Conv2D(16, (3, 3), activation='relu'))
6 model.add(layers.MaxPooling2D((2, 2)))
7 model.add(layers.Flatten())
8 model.add(layers.Dense(25, activation='relu'))
9 model.add(layers.Dense(5, activation='sigmoid'))
```

Bei der Aktivierungsfunktion ist zu beachten, dass bei allen Schichten des CNNs sich die ReLu Funktion und deren Abwandlungen (z.B. Leaky-ReLu) bewährt haben, da sie die Performance des Modells aus unterschiedlichen Gründen verbessern. In den Dense-Layern, insbesondere in der Ausgabeschicht, ist die sigmoidale Aktivierungsfunktion besonders geeignet, da sie die Ergebnisse der einzelnen Ausgabeneuronen als Ähnlichkeiten interpretiert. Diese Ähnlichkeiten liegen im Bereich zwischen 0 und 1 und sind ideal für mehrklassige Klassifikationsaufgaben, bei denen jedes Ausgabeneuron eine Merkmalsklasse repräsentiert. [28]

In der folgenden Zeile Code geschieht das Training des zuvor definierten neuronalen Netzes.

```

1 history = model.fit(x_train_r, y_train, validation_data=(x_val_r, y_val),
2     batch_size=1, epochs=100, verbose=2)
```

x_train_r: Eingabedaten, die für das Training des Modells verwendet werden.

y_train: Soll-Ausgabedaten (Labels), die die tatsächlichen Klassen der Trainingsdaten repräsentieren.

validation_data: Validierungsdaten (**x_val_r**) und zugehörige Labels (**y_val**). Sie dienen dazu, die Klassifizierung des Modells zu überprüfen und zu verhindern, dass das Modell einfach nur die Trainingsdaten "auswendig lernt" (Overfitting).

batch_size: Anzahl der Datensätze, die durch das neuronale Netz verarbeitet werden, bevor eine Gewichtsanpassung der Neuronen erfolgt. Eine Batch-Size von 1 bedeutet, dass nach jedem Datensatz die Gewichte aktualisiert werden, was dem sogenannten Stochastic Gradient Descent entspricht.

epochs: Gibt an, wie oft der gesamte Satz der Trainingsdaten durch das Modell verarbeitet wird. Hier werden die Trainingsdaten insgesamt 100 Mal durchlaufen.

Abbildung 26 veranschaulicht, wie sich der "Verlust-Wert" (Loss) während des Trainingsprozesses des neuronalen Netzes über die Epochen hinweg verändert. Der rote Graph repräsentiert die Trainingsdaten und der grüne Graph die Validierungsdaten. Der Loss-Wert gibt an, wie stark die Vorhersagen des Netzwerks von den tatsächlichen Daten abweichen [15]. Eine hohe Abweichung bedeutet, dass das Netzwerk nicht gut vorhersagt, während eine niedrige Abweichung auf bessere Vorhersagen hinweist.

Wie sich erkennen lässt, "lernt" das neuronale Netz im Laufe der Epochen, die Eingabedaten besser zu klassifizieren, was sich in einem sinkenden Loss-Wert niederschlägt. Da der Loss-Wert der Validierungsdaten ebenfalls sinkt, ist gewährleistet, dass das neuronale Netz nicht einfach nur die Trainingsdaten auswendig lernt, sondern tatsächlich generalisiert.

Abschließend kann das trainierte Modell als "Tensorflow-Lite"-Dateiformat (Dateiendung ".tflite") exportiert werden. Die "model.tflite"-Datei enthält sowohl die Architektur, als auch die Gewichte der einzelnen Neuronen [29].

Im Anhang sind zwei Jupyter-Notebooks vorzufinden:

- **ESP_IMA_dev.ipynb:** In diesem Notebook wird die gesamte Datenvorverarbeitung und das Training des neuronalen Netzes durchgeführt.
- **ESP_IMA_validation.ipynb:** Dieses Notebook wurde für die Testfälle in **Abschnitt 6.4** verwendet. Es nimmt das bereits trainierte neuronale Netz und den Scaler und ermöglicht es, eigene WAVE-Dateien klassifizieren zu lassen.

Möchte man diese Jupyter-Notebooks bei sich selbst ausführen, ist der leichteste Weg der folgende:

1. Im Anhang ist im Unterordner "neuronal_network" die Datei "jupyter_notebooks_with_dir_structure.zip" zu finden. Diese muss zunächst entpackt werden.
2. Anschließend muss der Ordner in das Root-Verzeichnis des eigenen Google Drive Accounts hochgeladen werden.

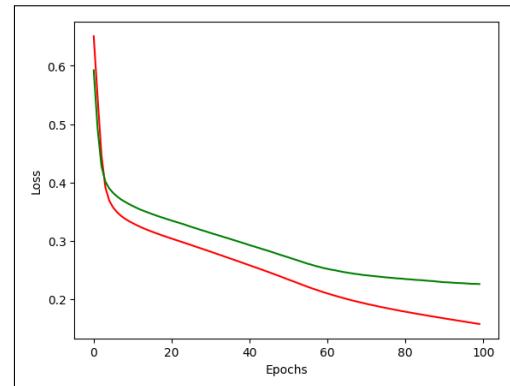


Abbildung 26: Darstellung des Loss-Werts über den Verlauf der Epochen hinweg

5.3.6 Betrieb des neuronalen Netzes auf dem STM32 Microcontroller

Für den Betrieb des Neuronalen Netzes wird "STM32Cube.AI" verwendet, um das in der "model.tflite" Datei definierte Neuronale Netz in C-Code nachzubilden. Um jedoch Audiodateien klassifizieren zu können, müssen zunächst sämtliche Schritte der Datenvorverarbeitung, die im Jupyter-Notebook gemacht wurden, wiederholt werden.

Einlesen der Datei und Datenvorverarbeitung

Zum Einlesen der Datei wird die in **Abschnitt 5.2.6** beschriebene Funktion `downsample_1024_samples()` verwendet. Diese reduziert die Datenmenge um den Faktor 3, was bei einer Eingabe-Samplerate von 44.1 kHz zu einer Ausgabe-Samplerate von 14.700 kHz (statt 16.000 kHz) führt. Die resultierende Abweichung von $\approx 8.8\%$ wird jedoch als vernachlässigbar für das Klassifikationsergebnis eingeschätzt.

Für die Berechnung der Mel-Spektrogramme wird die Bibliothek "STM32_AI_AudioPreprocessing_Library" verwendet. Diese ist Teil des "FP-AI-SENSING1" Function Packs [30]. Mit dieser lassen sich einzelne Spektrogramm Spalten berechnen, deren Amplitudenwerte müssen jedoch im Anschluss noch in Dezibel (dB) umgerechnet werden.

Ein Ablaufdiagramm, das den geplanten Prozess der Datenvorverarbeitung und der Klassifizierung der Audiosamples im Zusammenspiel mit den anderen Komponenten veranschaulicht, ist in **Abbildung 27** dargestellt.

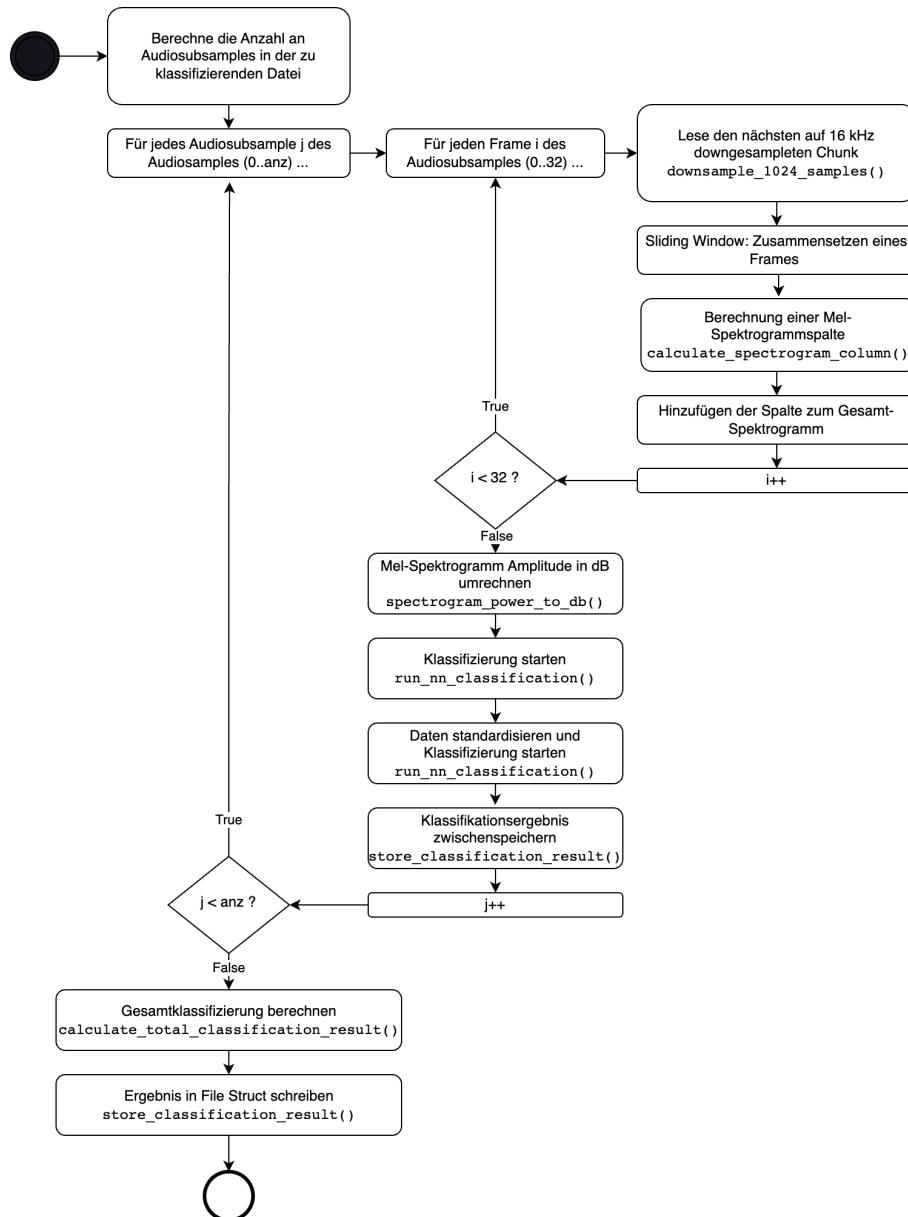


Abbildung 27: Ablauf der Datenvorverarbeitung und Klassifizierung eines Audiosamples im C-Code

Übersetzen des neuronalen Netzes mit STM32Cube.AI

Für die Nutzung von STM32Cube.AI muss zunächst in der CubeMX-Datei das Software Package “X-CUBE-AI“ in Version 9.0.0 heruntergeladen werden. Im Reiter “Configuration“ lässt sich ein Name für das Modell festlegen, welcher später im Code zur Referenzierung verwendet wird. Hier wird ebenfalls die Modell-Datei “model.tflite“ eingebunden. Ein Ausschnitt des Konfigurationsfensters ist in **Abbildung 28** dargestellt. Durch einen Klick auf “Analyze“ wird das Modell in C-Code übersetzt und ein Bericht zur Ressourcennutzung erstellt. Dieser bestätigt, dass die Ressourcenlimits des verwendeten Nucleo-Boards eingehalten werden.

Der folgende Auszug gibt einen Überblick über die genutzten Ressourcen:

```

1 Total Flash      82.050 B (80.13 KiB)
2 Total Ram       21.844 B (21.37 KiB)
```

Nutzung von STM32Cube.AI

Die Dokumentation zu STM32Cube.AI beschreibt detailliert die Nutzung anhand eines Beispiels [14]. Obwohl das Beispiel auf einer älteren Version basiert, hat sich der grundlegende Umgang mit der Software nicht verändert. Da die Nutzung der Funktionen und Zeck der einzelnen Variablen dort im Detail beschrieben sind, wird dies im folgenden Abschnitt nur kurz zusammengefasst.

Zur Nutzung von STM32Cube.AI ist ein Handle vom Typ `ai_handle` erforderlich. Dieses erhält man durch den Aufruf der Initialisierungsfunktion `ai_network_1_create_and_init()`. Anschließend werden Input- und Outputbuffer vom Typ `ai_buffer` initialisiert, die zur Eingabe von Daten in das neuronale Netz und zur Rückgabe des Klassifikationsergebnisses verwendet werden. Diese Initialisierungsschritte werden im Code des Projekts in der Funktion `init_nn()` durchgeführt:

```

1 int init_nn() {
2     ai_error err;
3     const ai_handle act_addr[] = { activations };
4
5     err = ai_network_1_create_and_init(&network, act_addr, NULL);
6     if (err.type != AI_ERROR_NONE) { return -1; }
7
8     ai_input = ai_network_1_inputs_get(network, NULL);
9     ai_output = ai_network_1_outputs_get(network, NULL);
10
11    return 0;
12 }
```

Hat man den Handle, kann man Daten durch das neuronale Netz klassifizieren lassen. Dazu müssen zunächst Pointer auf die Eingabe- bzw. Ausgabedaten in das `data` Feld der Input- und Outputbuffer gesetzt werden. Die Klassifizierung wird durch den Aufruf der Funktion `ai_network_1_run()` initiiert. Das Ergebnis der Klassifizierung wird anschließend in der Variable verfügbar, die durch den zuvor übergebenen Output-Pointer referenziert wurde. Dieser Prozess ist im Code in der Funktion `run_nn_classification` implementiert. Da vor der Klassifizierung die Standardisierung mit den exportierten Scaler-Werten erforderlich ist, wird diese in der Funktion vor dem Klassifizierungsvorgang umgesetzt.

```

1 int run_nn_classification(ai_float* pSpectrogram, ai_float* classification_result)
2   {
3     ai_i32 batch;
4     ai_error err;
5
6     ai_input[0].data = AI_HANDLE_PTR(pSpectrogram);
7     ai_output[0].data = AI_HANDLE_PTR(classification_result);
```

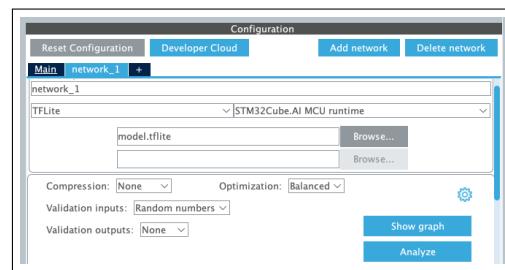


Abbildung 28: Konfiguration von STM32Cube.AI in CubeMX

```

8     if (network == AI_HANDLE_NULL) { return -1; }
9
10    for (int i = 0; i < AI_NETWORK_1_IN_1_SIZE; i++) {
11        pSpectrogram[i] = (pSpectrogram[i] - feature_scaler_mean[i]) /
12            feature_scaler_std[i];
13    }
14
15    batch = ai_network_1_run(network, ai_input, ai_output);
16    if (batch != 1) { return -1; }
17
18    return 0;
}

```

Erfahrungswerte bei der Nutzung von STM32Cube.AI

Insgesamt konnte für die Klassifikation eines Spektrogramms, also eines Audiosubsamples, eine Inferenz von 17 ms gemessen werden. Die Inferenz gibt an, wie viel Zeit die Klassifizierung eines Datensatzes benötigt. Damit ist auch die nicht-funktionale Anforderung **LQ02** validiert. Ausgehend von einem 30 Sekunden langen Audiosample ergibt sich bei einer Samplerate von 14,7 kHz eine reine Klassifikationszeit wie folgt:

$$\text{Anzahl der Audiosubsamples} = \frac{441000 \text{ Samples}}{16896 \text{ Samples/Subsample}} \\ \approx 26 \text{ Audiosubsamples}$$

Die reine Klassifikationszeit ergibt:

$$\text{Klassifikationszeit} = 26 \text{ Audiosubsamples} \times 17 \text{ ms/Subsample} \\ = 442 \text{ ms}$$

Während der Nutzung von STM32Cube.AI trat im Projektverlauf ein “HardFault“-Fehler bei der Ausführung der Funktion `ai_network_1_run()` auf. Dies konnte darauf zurückgeführt werden, dass eine fehlerhafte Deklaration einer Variable im Header der aktuellen Datei vorlag. Sie war nicht als `extern` gekennzeichnet. Obwohl die aufgerufene externe STM32Cube.AI-Bibliothek nicht direkt auf diese Variable zugriff, führte das Fehlen der `extern`-Deklaration vermutlich zu einer inkonsistenten Referenz beim Linken der Module.

Leider konnte die Klassifizierung durch das neuronale Netz aus zeitlichen Gründen nicht in das Gesamtprojekt integriert werden. Daher sind für die Komponente “Neuronales Netz“ im Anhang zwei CubeMX-Projekte enthalten:

- **esp_cubeai_test:** Dieses Projekt diente der Überprüfung und Validierung der Funktionen von STM32Cube.AI, der Spektrogramm-Berechnungsbibliothek sowie der selbst entwickelten Funktionen für das Aufteilen in Frames. Als Datenbasis dient ein Audiosubsample, das in einer C-Datei als 16.896 `short`-Werte gespeichert ist. Dadurch konnte der Ablauf der Klassifikation unabhängig von anderen Komponenten getestet werden. Weitere Details zur Validierung und zu den Tests sind im **Abschnitt 6** zu finden.
- **esp_cubeai_for_integration:** Dieses Projekt enthält Code, der für die Integration in das Gesamtprojekt vorbereitet und strukturiert wurde. Der Code wurde jedoch noch nicht getestet und könnte Fehler enthalten. Zusätzlich wurde das Auffüllen von Audiosamples (“zero-padding”), die weniger als 16.896 Samples enthalten, noch nicht implementiert.

5.4 Interface

Ein **User Interface (UI)** in eingebetteten Systemen ermöglicht eine Interaktion zwischen Benutzer und Gerät.

Das **Interface** besteht aus **Encodern**, **Schiebepotentiometern**, einem **LCD-Display** und **Buttons**. Dessen Zusammenspiel ermöglicht dem Benutzer die nötige Kontrolle über den Sampler.

Im folgenden Kapitel wird Ihnen die Funktionalität, Implementierung, Ansteuerung, sowie das Zusammenspiel der Komponenten untereinander nähergebracht:

5.4.1 Encoder

/LF01/
Encoder-TS
Grundfunktionalität:

Der **Encoder** dient der Navigation im Benutzer-Menü des Samplers. Er ermöglicht die Auswahl von Samples und das Scrollen durch die Sample-Liste, die auf dem LCD-Display angezeigt wird. Diese Liste zeigt die Namen der Samples an, die zuvor durch die Fader-Einstellungen bestimmt wurden. Der Cursor an der Seite zeigt die Aktuelle Position des Cursors an. Der Sample gilt als ausgewählt wenn dessen Name unter der Liste erscheint.

Umsetzung der Funktionalität:

Die Auswahl von Samples sowie das Inkrementieren und Dekrementieren des Cursors, welcher sich im Struct des Filemanager befindet werden durch Interrupts unterstützt. Wenn der Encoder bewegt oder gedrückt wird, sendet er Signale an die MCU, die Interrupts auslösen.

Die Auswertung dieser Signale erfolgt dann in der Callback `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`.

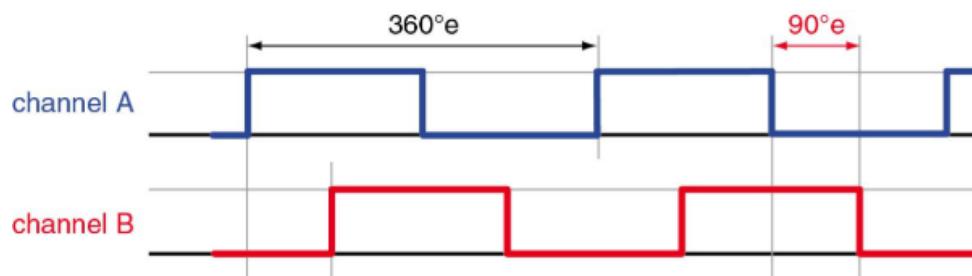


Abbildung 29: Phasenverschiebung A und B

Wenn A und B beide High sind[31], wird die Cursor-Position welche in auf dem Display durch den Methodenaufruf `cursorUp(FileManager *fm)` erhöht. Wenn Abbildung 29 A High und B Low ist, wird die Cursor-Position auf dem Display durch `cursorDown(FileManager *fm)` verringert. Wenn der Schalter gedrückt wird, wird die Datei durch `selectFile(FileManager *fm)` der Index des Files gespeichert.

```

68 void cursorUp(FileManager *fm){
69     if (fm->cursor_index > 0) {
70         fm->cursor_index--;
71     } else {
72         fm->cursor_index = fm->num_matched_files - 1 ;
73     }
74 }

84 void cursorDown(FileManager *fm){
85     if (fm->cursor_index < fm->num_matched_files - 1) {
86         fm->cursor_index++;
87     } else {
88         fm->cursor_index = 0;
89     }
90 }

```

Anhand des Index des Files kann der Names des Files ermittelt werden aus dem Filemanager.

```

159 void selectFile(FileManager *fm) {
160     fm->current_file_index = fm->cursor_index;
161 }

```

Problematik:

Beim Drücken des Knopfes kann es in Systemen zu Mehrfachauslösungen kommen. Dies war auch beim Schalter des Encoders der Fall.

Lösung:

Um Mehrfachauslösungen zu verhindern, wurde ein Debouncing-Mechanismus implementiert. Dieser Mechanismus verwendet ein Flag `debounce` und einen Timer `tim3`, um wiederholte Auslösungen zu verhindern. Beim Drücken des Knopfes wird ein Interrupt ausgelöst, der eine Callback-Funktion `HAL_GPIO_EXTI_Callback(uint GPIO_Pin)` aufruft. In dieser Callback-Funktion wird ein Flag gesetzt und ein Timer gestartet. Der Timer sorgt dafür, dass weitere Auslösungen innerhalb einer definierten Zeitspanne ignoriert werden, wodurch Mehrfachauslösungen effektiv verhindert werden.

Erst nach dem Ablauf des Timers ist es wieder möglich den Knopf zu betätigen.

5.4.2 Schiebenpotentiometer, ADC und DMA

/LF02/

Grundfunktionalität:

ADC

Die an den Schleifer des **Schiebepotentiometers** erfassten analogen Spannungen, werden vom **Analog-Digital-Wandler ADC** erfasst. Abbildung 30.

ADC[32] (Analog Digital Converter) wird benutzt, um die Ausgangsspannung an den Schleifern zu digitalisieren und in Zahlen zu fassen. Er nimmt in regelmäßigen Intervallen mit Hilfe des DMA Proben des analogen Signals. Die Auflösung in 12 Bits 15 ADC Clock Cycles hat sich als am effizientesten herausgestellt. Dies erfolgt über die Pins **PA6**, **PA7**, **PB0**, **PB1**, **PC0**, die als Schnittstellen parallel der Reihenfolge der Channels **6**, **7**, **8**, **9**, **10** dienen. Jedes Mal, wenn an der Ausgangsspannung eines der Schleifer eine Veränderung wahrgenommen wird, startet eine Interrupt Service Routine.

Configuration

Resolution: 12Bit divided 15 ADC ClockCycles (Wertebereich: 0-4096)

Mode: Scan Mode und Continues Convervion Mode

DMA: DMA Continuous Requests Enable

Conversions: 5

Sampling Time: 3 Cycles

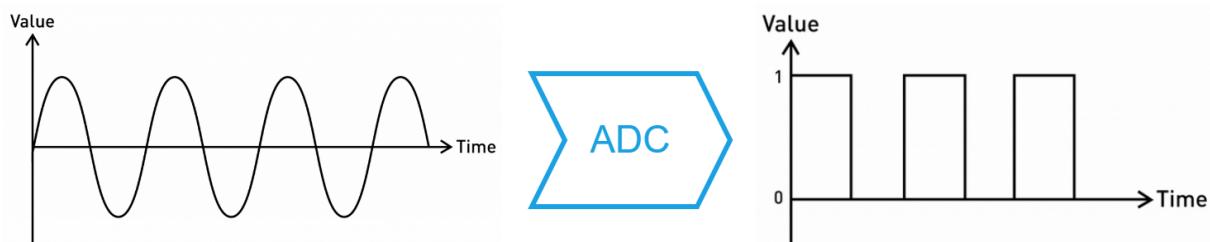


Abbildung 30: ADC Conversion

DMA

Der **Direct Memory Access (DMA)-Controller** übernimmt anschließend die direkte Übertragung dieser digitalen Werte in den Speicher, konkret in `adcBuffer[NUM_CHANNELS]`.

DMA (Direct Memory Access) wird gestartet, um die Werte, die an den Pins des ADC liegen, zyklisch zu pollen. Er wird zeitbasiert mit einem **Timer tim5** gestartet und in der Callback nach der Glättung der kumulierten ADC-Werte gestoppt.

Configuration

Timer: Timer5, Internal Clock, One Pulse Mode

Data Width: Word Übertragung von 32-Bit-Datenblöcken

Mode: Circular

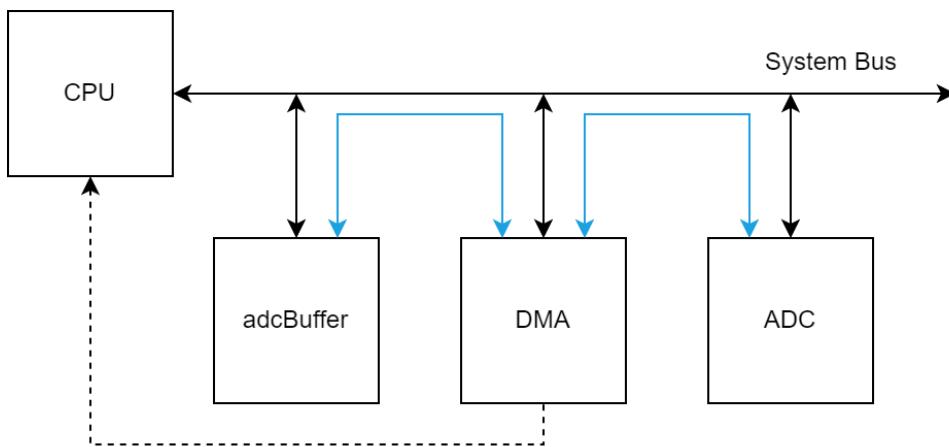


Abbildung 31: DMA ADC FADER Bus System

Der Direct Memory Access **DMA**-Controller ermöglicht die direkte Datenübertragung zwischen Speicher und Peripheriegeräten, wodurch die Prozessorbelastung reduziert wird. Analoge-Digital-Wandler (ADCs) sorgen für die schnelle Digitalisierung analoger Signale, was eine präzise Verarbeitung der Messdaten ermöglicht. Potentiometer bieten die Möglichkeit zur benutzerfreundlichen Anpassung von Spannungswerten. Das Bussystem sorgt für eine effiziente Kommunikation zwischen den verschiedenen Komponenten und gewährleistet eine konsistente und zuverlässige Datenverarbeitung. **Abbildung 31**[33]

Problematik:

No.1

Bei der Implementierung der DMA traten Probleme auf, die dazu führten, dass der Bildschirm einfrohr und das Bedienen der Interfaces unmöglich wurde. Das Problem entstand, weil die DMA fortlaufend Daten übertrug und die CPU dadurch vollständig beansprucht wurde, um die Daten, die von der DMA übertragen wurden, zu verarbeiten. Dies führte zu einer Überlastung der CPU, da sie gezwungen war, sich auf die Verarbeitung der von der DMA bereitgestellten Daten zu konzentrieren, anstatt andere Aufgaben zu bewältigen. Infolgedessen konnten keine weiteren Systeminteraktionen durchgeführt werden, da die CPU keine Ressourcen mehr für andere Aufgaben zur Verfügung hatte.

No.2

Die Auswertung und Verarbeitung der Signale erfolgt in der Funktion

¹ HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)

Im Laufe der Implementation des ADC könnte man Schwankungen an den Ausgangssignalen des ADCs feststellen, was die Genaugigkeit der Potentiometer einstellung beeinflusste.

Lösung No.1:

Um das Problem zu beheben, wurde ein **Timer** **tim5** implementiert, der die DMA-Übertragung steuert. Der Timer sorgt dafür, dass die DMA in regelmäßigen Intervallen gestartet und gestoppt wird. Diese Maßnahme ermöglicht es der CPU, sich ausreichend um andere Aufgaben zu kümmern, indem sie nicht kontinuierlich mit der Datenverarbeitung durch die DMA beschäftigt ist. Der Einsatz des Timers verhindert, dass die DMA die CPU überlastet und gewährleistet eine ausgewogene Nutzung der Systemressourcen.

Lösung No.2:

Diese Schwankungen sind durch Glättung der Werte behoben.

Zunächst werden des `adcBuffer[i]` in `smoothValue[i]` aufeinander addiert. Dann wird deren Durchschnittswerte berechnet. Die geglätteten Werte werden für alle Kanäle des ADC berechnet.

```

142     if (hadc->Instance == ADC1){
143         cnt++;
144         for(int i = 0; i < NUM_CHANNELS; i++){
145             smoothValue[i] += adcBuffer[i];
146         }
147         // Calculation of the average values for the display and the compare
148         // operations
149         if(cnt >= SMOOTHING_HEIGHT){
150             HAL_ADC_Stop_DMA(&hadc1);
151             for(int j = 0; j < NUM_CHANNELS; j++){
152                 //Shown values
153                 currentClassPercentADC[j]=(smoothValue[j]/SMOOTHING_HEIGHT)/41;
154                 //Compare values for the Sortalgorythm
155                 fm.fader_Class[j] = ((smoothValue[j]/SMOOTHING_HEIGHT)/4096.0f);
156                 smoothValue[j]=0;
157             }

```

Diese Glättung sorgt dafür, dass die Messwerte stabiler und weniger anfällig für zufällige Schwankungen sind. Anschließend werden die Durchschnittswerte ermittelt. Diese Durchschnittswerte dienen zwei Zwecken: Einerseits werden sie zur Anzeige auf dem Bildschirm verwendet `currentClassPercentADC[]`, andererseits sind sie für Vergleichsoperationen innerhalb des Sortieralgorithmus von Bedeutung `fm.fader_Class[]`. Schließlich wird ein Zeichenarray `faderProzent[0]` initialisiert, das später auf dem Display angezeigt wird.

Eine andere Möglichkeit des Spannungs Inkonsistenz könnte mit Kondensatoren umgesetzt werden was jedoch nicht umgesetzt würde aus zeitlichen Gründen.

5.4.3 Display, SD1306 Treiber und I2C

/LF01/
/LF02/

I2C

I2C[34] (Inter-Integrated Circuit) ist ein Kommunikationsprotokoll, das über zwei Leitungen arbeitet: **SDA** (Serial Data Line) und **SCL** (Serial Clock Line). Es verwendet eine Master-Slave-Architektur, bei der der Master die Kommunikation steuert und die Slaves Daten empfangen oder senden. Die Datenübertragung erfolgt in Byte-Paketen, wobei der Master eine Start-Bedingung sendet, gefolgt von der Adresse des Slaves und den Daten. Nach dem Empfang eines Datenbytes sendet der Slave ein Bestätigungsbit (ACK). Die Kommunikation endet mit einer Stop-Bedingung **Abbildung 32**.

- **SDA (Serial Data Line)**: Überträgt die Daten zwischen Master und Slave.
- **SCL (Serial Clock Line)**: Taktet die Datenübertragung, gesteuert vom Master.

Vorteile

- **Einfache Verkabelung**: Es werden nur zwei Leitungen für die Kommunikation benötigt, was den Verkabelungsaufwand erheblich reduziert.
- **Geringe Hardware-Kosten**: Die Implementierung von I2C ist kostengünstig und benötigt wenig zusätzliche Hardware, da die Logik für das Protokoll minimal ist.
- **Fehlererkennung**: I2C bietet einfache Methoden zur Fehlererkennung, wie das Bestätigungsbit (ACK), um sicherzustellen, dass Daten erfolgreich übertragen wurden.
- **Unterstützung von Standard-Bibliotheken**: Viele Mikrocontroller und Entwicklungsplattformen bieten Standard-Bibliotheken für **I2C**, was die Implementierung und Fehlerbehebung erleichtert.

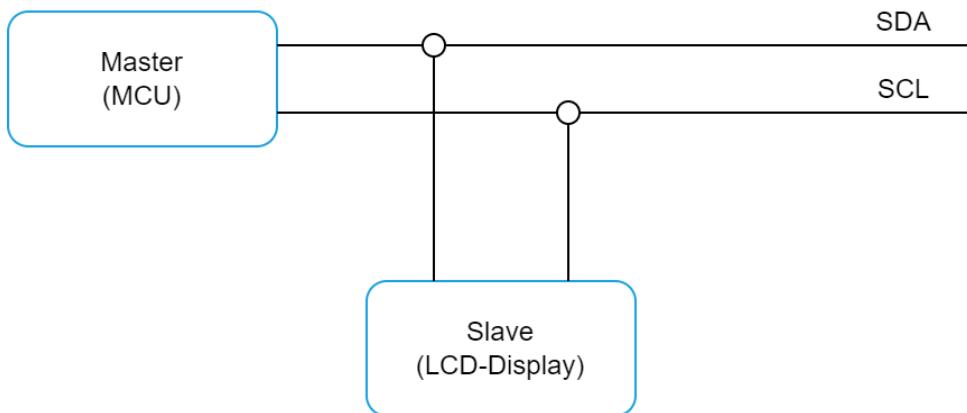


Abbildung 32: I2C Kommunikation

Display

Das Display ist die zentrale Komponente des Interfaces, da es visuelles Feedback auf die Benutzerinteraktionen bietet. Die Kommunikation zwischen dem Nucleo F401RE-Mikrocontroller und dem Display erfolgt über das **I2C**-Protokoll. Der Mikrocontroller übernimmt dabei die Rolle des Masters, der den Bus steuert, während das Display als Slave agiert.

Die Ansteuerung des Displays erfolgt über den SD1306-Treiber [35], der als Schnittstelle zwischen dem Mikrocontroller Nucleo F401RE und dem LCD-Display fungiert. Der Treiber nutzt das I2C-Protokoll, um Befehle und Daten effizient zu übermitteln. Er sendet die erforderlichen Befehle an das Display und übernimmt die Kommunikation gemäß den I2C-Spezifikationen. Bei Bedarf kann die Initialisierungssequenz des Treibers angepasst werden, um auch SH1103-kompatible Bildschirme zu unterstützen **Abbildung 33**.

Zusammenfassend ermöglicht der SD1306-Treiber die vollständige Integration des Displays in das System, indem er die I2C-Kommunikation zwischen Mikrocontroller und Display umsetzt.

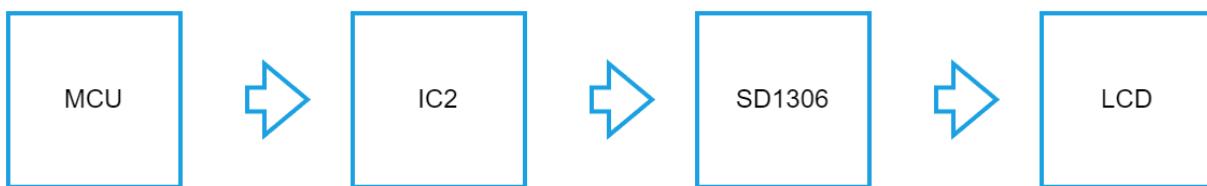


Abbildung 33: Display SD1306 I2C Kommunikation

Die beiden Hauptfunktionen zur Darstellung auf dem OLED-Bildschirm sind:

```

void displayStrings(I2C_HandleTypeDef *hi2c1, char** strings, uint8_t numStrings, uint8_t cursor_index)

void renderSelectedFile(I2C_HandleTypeDef *hi2c1, const char *filename) bzw.

drawFaderProzent ähnliche funktionsweise wie renderSelectedFile.
  
```

In der Funktion `displayStrings` werden die ‘ssd1306 Treiber’-Funktionen verwendet, um die Strings auf dem LCD-Display darzustellen. Die Funktion `ssd1306_SetCursor` legt die Position für die Anzeige fest, und `ssd1306_WriteString` wird verwendet, um die Strings auf dem Display auszugeben. Bei der Anzeige des ausgewählten Strings wird zusätzlich ein Cursor-Symbol integriert.

In der Funktion `renderSelectedFile` wie bei `displayStrings` ‘ssd1306’-Funktionen verwendet, um den Namen der ausgewählten Datei auf dem OLED-Display darzustellen. Die Funktion `ssd1306_DrawPixel` wird verwendet, um den Bereich für die Dateianzeige zu löschen. Anschließend legt `ssd1306_SetCursor` die Position für die Textausgabe fest, und `ssd1306_WriteString` gibt den Namen der ausgewählten Datei auf dem Display aus.

Problematik:

Der ursprünglich verwendete SD1306-Treiber war nicht mit dem Display kompatibel. Der Fehler wurde durch die ungeeignete Initialisierungssequenz für das Display verursacht, was zu einer fehlerhaften Darstellung der Daten auf dem Bildschirm führte.

Lösung:

Die Anpassung der Definitionskonstanten, die als Befehle über I2C an das Display gesendet werden, sowie die Änderung der Schleifenbedingung in der Methode `ssd1306_UpdateScreen(I2C_HandleTypeDef *hi2c)` von einer Länge von 8 auf 16 haben das Problem behoben.

5.4.4 Dateimanagement

Das Dateimanagement fungiert als zentrale Schnittstelle zwischen der Audio-Wiedergabe, dem neuronalen Netzwerk zur Klassifizierung und der Benutzeroberfläche. Es organisiert und verwaltet die Audiodateien, die vom neuronalen Netzwerk analysiert und klassifiziert werden. Nach der Klassifizierung speichert das Dateisystem die Ergebnisse und gewährleistet, dass nur relevante und korrekt kategorisierte Dateien im Interface angezeigt werden. Das Dateisystem besteht aus zwei Hauptstrukturen: File und FileManager welche Ihnen im folgenden näher gebracht werden.

File

Die **File**-Struktur repräsentiert einzelne Audiodateien im System. Sie enthält den Dateinamen, der sowohl für das Abspielen der Audiodatei als auch für die Anzeige in einer Liste von Samples verwendet wird. Zusätzlich speichert die Struktur Klassifizierungsdaten, die vom neuronalen Netzwerk bereitgestellt werden.

```
60  typedef struct {
61      char filename[MAX_FILENAME_LENGTH];           ///< The name of the file.
62      float classes[MAX_CLASSES];                 ///< The classification classes for the
63      → file.
64 } File;
```

FileManager

Die **FileManager**-Struktur repräsentiert das Dateisystem selbst. Sie ist verantwortlich für die Verwaltung und Speicherung von Dateien und umfasst wesentliche Elemente, die für die Visualisierung der Dateien auf dem LCD-Display erforderlich sind.

```
72  typedef struct {
73      File files[MAX_FILES];                      ///< Array to store all files that have been
74      → classified.
75      File shownFiles[MAX_FILES];                ///< Array to store files that match the
76      → fader settings.
77      char current_cursor_filename[25];          ///< The filename of the selected by the
78      → user.
79      int num_files;                            ///< The total number of files.
80      int num_matched_files;                     ///< The number of files that match the fader
81      → settings.
82      int cursor_index;                          ///< The index of the current cursor
83      → position.
84      int current_file_index;                    ///< The index position of the selected file.
85      float fader_Class[MAX_CLASSES];            ///< The settings from fader.
86 } FileManager;
```

Problematik:

Problematik:

Das Schreiben und Lesen des **FileManager**-Struktur auf der SD-Karte war zunächst nicht möglich und konnte aus zeitlichen Gründen nicht behoben werden.

Teillösung:

Das Lesen der Audiodateinamen und die Übertragung in den **FileManager** (im RAM der MCU) funktionierten und ermöglichten somit eine Visualisierung auf dem Display.

Das Lesen der Namen wird über **FILINFO fno** in der Funktion **listFiles(const char *path)** realisiert, indem man als Pfad den im Root-Verzeichnis befindlichen Ordner **/SamplePool**” übergeben hat.

```

373     // Check whether it a file
374     if (!(fno.fattrib & AM_DIR))
375     {
376         // It is a file
377         char *fileName = fno.fname;
378
379         // Check whether the file have a extension .wav
380         if (strstr(fileName, ".wav") != NULL)
381         {
382             // File with extension .wav get Safed
383             strncpy(fileNamesSDCard[fileCount + 1], fileName,
384                     MAX_FILENAME_LENGTH - 1);
385             fileNamesSDCard[fileCount][MAX_FILENAME_LENGTH - 1] = '\0'; // ← Null-terminieren
386             fileCount++;
387         }
388     }

```

Das Array **fileNamesSDCard** wird bei der Initialisierung des Displays weiter verwendet.

5.5 PCB-Design

Der ursprüngliche Plan, ein voll funktionales Eurorack-Modul mit eigens entwickeltem PCB, wie in **LF05** ist leider aufgrund von zeitlicher Knappheit und einem zu ambitionierten Funktionsumfang aus der Projektabgabe gestrichen worden.

Hier haben schlachtweg die Vorerfahrung im PCB-Design gefehlt, um noch nebenbei ein funktionierendes Board zu entwickeln.

Jedoch ist im Nachgang begonnen worden ein solches PCB zu entwickeln, die bisherigen Schaltpläne befinden sich im Anhang 7.

Die Levelshifting Schaltungen für die Umwandlung von Eurorack Leveln $\pm 5\text{ V}$ zu den Pegeln des Audio Codecs $\pm 0.7\text{ V}_{\text{rms}}$ für die Ein-/Ausgangsstufen wurden bereits entwickelt.

Die Schaltung basiert auf einem Operationsverstärker mit negativer Rückkopplung, wodurch die gewünschte Abschwächung am Eingang und Verstärkung am Ausgang erzielt wird.

Das Projekt, und somit ein eigenes PCB wird in jedem Fall Privat weiterentwickelt.

5.6 Integration der einzelnen Komponenten in das Gesamtsystem

Leider konnte aufgrund von Zeitmangel die Integration der einzelnen Komponenten in ein Gesamtsystem nicht realisiert werden. Verschiedene Ursachen trugen zu diesen zeitlichen Verzögerungen bei.

Wie bereits in den vorherigen Abschnitten erwähnt, führten Probleme bei der Implementierung zu erheblichen Zeitverlusten. Insbesondere verursachte ein “Hard-Fault“-Fehler auf dem STM Board in der Klassifizierungskomponente lange Verzögerungen, da die Ursache schwer zu identifizieren war.

Anfänglich gab es auch Schwierigkeiten beim Auslesen der SD-Karte, da der zunächst verwendete Treiber[36] nicht für die Nutzung von SD-Karten der Version 3 ausgelegt war. Der Einsatz von älteren SD-Karten-Versionen erwies sich als erfolgreich. Allerdings war die Datenübertragung über SPI aufgrund des verwendeten Treibers nicht effizient genug, was letztlich zum Umstieg auf die SDIO-Schnittstelle führte.

Ein weiteres Hindernis war die Verfügbarkeit von nur einem Nucleo-F7-Board für das gesamte Team, welches bis zuletzt durch die Implementierung des neuronalen Netzes in Anspruch genommen wurde. Die Entwicklung der übrigen Komponenten erfolgte daher auf Nucleo-F4-Boards. Die Portierung dieser Projekte auf das F7-Board gestaltete sich dann wesentlich aufwendiger als angenommen. Das DMA-Verhalten auf dem F7-Board entsprach nicht den Erwartungen, und aufgrund der begrenzten Zeit konnte dieses Problem nicht gelöst werden. Dies erschwerte die Integration der Teilprojekte erheblich.

Der Versuch, die Teilprojekte Audio und Interface auf das F4-Board zu portieren, scheiterte ebenfalls aufgrund von Pointer-Problemen bei der Auswahl der abzuspielenden Datei. Eine Lösung des Problems stand in Aussicht, musste allerdings wieder aufgrund von Zeitmangel eingestellt werden.

Obwohl einzelne Komponenten schließlich für die Integration in das Gesamtsystem vorbereitet wurden, kam es zu keiner tatsächlichen Zusammenführung in einem einheitlichen Projekt.

Die größten Schwierigkeiten bei der Integration waren die Nutzung uns zuvor unbekannter Technologien wie CUBE.AI, der Mangel an Ressourcen in Bezug auf die Verfügbarkeit des F7-Boards und vor allem die Zeit als kritischer Faktor, der die verspätete Portierung von Projekten verursachte.

6 Test und Validierung

Zur Überprüfung der Anforderungserfüllung jedes Lastenfalls (LF) wurden zunächst alle Komponenten einzeln entwickelt und getestet. Wie im **Abschnitt 5.6** erläutert, verhinderten spezifische Herausforderungen eine Gesamtintegration der Systemkomponenten. Daher wurden keine Integrationstests entwickelt oder durchgeführt.

In den nachfolgenden Abschnitten werden die Testfälle und deren Ergebnisse für jeden Lastenfall beschrieben. Diese dienen dazu, die korrekte Funktionalität und die erfolgreiche Umsetzung der jeweiligen Anforderungen zu validieren.

6.1 Testprotokoll: /LF01/

LF01

6.1.1 Auswertung der Drehung des Encoders

- **Testfall:** Prüfung des Drehverhaltens
- **Schritte:**
 1. Der Encoder wird schnell und langsam in beide Richtungen gedreht.
 2. Dabei wird die Zählvariable `rotary_enc_count` beobachtet.
- **Testergebnisse:**
 - Die Zählvariable zählt wie erwartet hoch und runter.
 - Nur beim Betreten der `HAL_GPIO_EXTI_Callback` soll die Variable hochgezählt werden.

Gemessene Ergebnisse:

- Während der Tests wurde beobachtet, dass die Zählvariable korrekt hoch- und runtergezählt hat. Bei jeder Interaktion mit dem Encoder hat sich der Wert der Zählvariable entsprechend erhöht oder verringert.
- Die Zählvariable wurde ausschließlich beim Betreten der `HAL_GPIO_EXTI_Callback` Funktion verändert. In allen anderen Fällen blieb die Zählvariable unverändert.
- Es wurden keine Mehrfachauslösungen festgestellt. Jede Drehung des Encoders, sowohl schnell als auch langsam, führte zu einer einzigen Zähleränderung, wie erwartet.

Beschreibung des Vorgehens: Um das Drehverhalten des Encoders zu testen, wurde der Encoder an das System angeschlossen und die Funktionalität überprüft. Anschließend haben wir den Encoder in beiden Richtungen, sowohl schnell als auch langsam, gedreht und die Zählvariable `rotary_enc_count` stand dabei unter Beobachtung, um sicherzustellen, dass sie ausschließlich durch die `HAL_GPIO_EXTI_Callback` beeinflusst wird.

6.1.2 Debouncing des Push-Buttons

- **Testfall:** Prüfung der Funktionalität des Push-Buttons.
- **Schritte:**
 1. Erstellung einer Zähl Variable `cnt`
 2. Drücken des Push-Button.
 3. Prüfen ob der Interrupt korrekt ausgelöst wird.
 4. Das korrekte Verhalten des Entprellen prüfen.
- **Erwartete Werte:**
 - Der Interrupt wird bei jedem Drücken sauber und eindeutig ausgelöst.
 - Es treten kein Prellen oder Mehrfachauslösungen auf.

- **Testergebnisse:**

- Der Interrupt wurde bei jedem Drücken zuverlässig und eindeutig ausgelöst.
- Es traten keine Prellen oder Mehrfachauslösungen auf. Die `cnt` hat korrekt hochgezählt.

Beschreibung des Vorgehens: Der Push-Button würde mehrmals gedrückt und dabei beobachtet, ob der Interrupt korrekt ausgelöst wurde. Während des Tests wird darauf geachtet, ob der Button prellte oder Mehrfachauslösungen verursachte in dem ein Zähl Variable `cnt` im Debugger beobachtet würde. Die Ergebnisse bestätigten, dass der Interrupt zuverlässig und ohne Prellen funktionierte.

6.1.3 Menünavigation

- **Testfall:** Testen der Navigation im Menü mithilfe des Encoders.

- **Schritte:**

1. Den Encoder wird in beide Richtungen gedreht und die Cursor-Bewegung beobachtet. Die cursor Variable `cursor_index` des Filemanagers im Debugger prüfen auf deren Zählverhalten.
2. Der Push-Button wird gedrückt, um eine Auswahl zu treffen. Der Flag `switch_push_button` prüfen um die Validierung des korrekten Verhaltens beim drücken zu bestätigen.

- **Erwartete Werte:**

- Der Cursor bewegt sich entsprechend der Drehrichtung des Encoders.
- Die Auswahl wird korrekt angezeigt, wenn der Push-Button gedrückt wird.

- **Testergebnisse:**

- Der Cursor bewegte sich korrekt und präzise entsprechend der Encoder-Drehung.
- Die Auswahl wurde zuverlässig angezeigt, nachdem der Push-Button gedrückt wurde.

Beschreibung des Vorgehens: Der Encoder wird gedreht, um sicherzustellen, dass der Cursor im Menü korrekt bewegt wird. Die Cursor Variable `cursor_index` im FileManager Struct `fm` wird korrekt hoch und runter gezählt bei entsprechender Bewegung. Anschließend habe ich den Push-Button betätigt, um zu prüfen, ob die Auswahl entsprechend angezeigt wird. Der Flag `switch_push_button` würde ebenfalls korrekt gesetzt.

6.2 Testprotokoll: /LF02/

LF02

6.2.1 ADC-Konfiguration überprüfen

- **Testfall:** Überprüfung der ADC-Konfiguration.
- **Schritte:**
 1. Die Auflösung, Sample-Rate und Referenzspannung des ADCs wurden überprüft.
 2. Es wird sichergestellt, dass der ADC korrekt konfiguriert ist.
 3. Die Referenzspannung wird mit einem Multimeter gemessen, um ihre Übereinstimmung mit den Werten im Debugger zu bestätigen.
 4. Die Sample-Rate wird durch Analyse der ADC-Konfigurationsregister überprüft und mit den erwarteten Werten verglichen. **ADC1_CFGR Configuration Register**.
- **Erwartete Werte:**
 - Auflösung: 12-bit (0-4096 Werte).
 - Sample-Rate entspricht den Spezifikationen.
 - Referenzspannung entspricht der spezifizierten Spannung.
- **Testergebnisse:**
 - Die Auflösung des ADCs wurde korrekt auf 12-bit (0-4096 Werte) eingestellt.
 - Die Referenzspannung wurde mit einem Multimeter gemessen und entsprach der spezifizierten Spannung.
 - Die Sample-Rate wurde durch Überprüfung der ADC-Konfigurationsregister bestätigt und entsprach den angegebenen Spezifikationen.
 - Die ADC-Konfiguration war ordnungsgemäß und entsprechend den Anforderungen eingerichtet.

Beschreibung des Vorgehens: Die Konfiguration des ADCs wurde durch Überprüfung der Auflösung, Sample-Rate und Referenzspannung sichergestellt. Die Referenzspannung wurde direkt mit einem Multimeter gemessen, um ihre Übereinstimmung mit den Spezifikationen zu bestätigen. Die Sample-Rate wurde durch die Analyse der ADC-Konfigurationsregister verifiziert, indem die tatsächliche Rate mit den erwarteten Werten verglichen wurde. Die Validierung erfolgte durch den Einsatz eines Debugging-Tools, um zu gewährleisten, dass der ADC gemäß den Anforderungen konfiguriert ist.

6.2.2 DMA-Konfiguration überprüfen

- **Testfall:** Überprüfung der DMA-Konfiguration.
- **Schritte:**
 1. Es wird sichergestellt, dass der DMA die ADC-Daten in den Puffer `currentValues` überträgt.
 2. Die Übertragungsart und die Übertragungsrate werden überprüft.
 3. Das Datenformat wird auf WORD (16-Bit) konfiguriert und geprüft.
- **Erwartete Werte:**
 - Korrekte Konfiguration des DMA, Übertragungsmodus auf Circular.
 - Datenformat korrekt auf WORD (16-Bit) eingestellt.
- **Testergebnisse:**
 - Der DMA überträgt die ADC-Daten wie erwartet in den Puffer `currentValues`.
 - Die Übertragungsart ist korrekt auf Circular eingestellt.

- Das Datenformat wurde erfolgreich auf WORD (16-Bit) konfiguriert. Die Überprüfung wurde durch Einsicht in die DMA-Register bestätigt. Insbesondere wurden die Registerwerte für ‘PSIZE’ und ‘MSIZE’ auf 16-Bit überprüft, um die korrekte Einstellung des Datenformats zu bestätigen.

Beschreibung des Vorgehens: Die DMA-Konfiguration wurde überprüft, indem zuerst sichergestellt wurde, dass der DMA die ADC-Daten korrekt in den Puffer `currentValues` überträgt(Debugging). Danach wurde die Übertragungsart auf Circular gesetzt und die Übertragungsrate validiert. Schließlich wurde das Datenformat auf WORD (16-Bit) konfiguriert und durch Einsicht in die entsprechenden DMA-Register überprüft, insbesondere durch Überprüfung der Registerwerte für ‘PSIZE’ und ‘MSIZE’.

6.2.3 Daten analysieren

- **Testfall:** Analyse der aufgezeichneten ADC-Werte und Prüfung der Glättung.
- **Schritte:**
 1. Die ADC-Werte `fm.fader_Class[]`, `adcBuffer[]`, `smoothValue[]` und `currentClassPercentADC[]` werden mit einem Debugger analysiert.
 2. Es wird geprüft, ob eine lineare Zunahme der Werte entsprechend der Stellung des Schiebe-Potentiometers vorliegt.
 3. Der `smoothValue[]` wird berechnet, und es wird auf Schwankungen und plötzliche Änderungen geprüft. Glättung erfolgt mit 100, 1000, 10000, 30000, 100000 aufeinander addierten Werten.
- **Erwartete Werte:**
 - ADC-Werte entsprechen den Positionen des Potentiometers.
 - Die geglätteten Werte zeigen eine stabile Wertentwicklung.
 - Keine unerwarteten Sprünge oder signifikanten Schwankungen; Grundrauschen um wenige Volt ist normal.
- **Testergebnisse:**
 - Die ADC-Werte `fm.fader_Class[]`, `adcBuffer[]`, `smoothValue[]` und `currentClassPercentADC[]` wurden erfolgreich mit dem Debugger analysiert.
 - Die Werte zeigten eine erwartungsgemäße lineare Zunahme in Abhängigkeit von der Potentiometer-Position.
 - Der `smoothValue[]` zeigte eine stabile Wertentwicklung ohne signifikante Schwankungen oder plötzliche Änderungen.
 - Es traten keine unerwarteten Sprünge auf. Ein sehr geringes Grundrauschen wurde erstmal als normal eingestuft. **30000** aufeinander addierte Werte dessen Mittelwert berechnet würde stellten sich jedoch am als Effizientesten raus.

Beschreibung des Vorgehens: Die ADC-Werte wurden mit einem Debugger analysiert, um sicherzustellen, dass sie der Stellung des Potentiometers entsprechen. Es wurde überprüft, ob die Werte linear ansteigen und ob der geglättete Wert `smoothValue[]` stabil bleibt. Schwankungen und plötzliche Änderungen wurden geprüft, um die Effektivität der Glättung zu bewerten. Das Grundrauschen wurde als normal eingestuft.

Lösung um das Grundrauschen zu minimieren: Das Grundrauschen könnte zukünftig durch den Einsatz von Kondensatoren, z.B. 100 nF, minimiert oder beseitigt werden.

6.2.4 Wiedergabe der Fader-Werte und Filterung auf dem Display

- **Testfall:** Überprüfung der korrekten Anzeige der Fader-Werte und der Filterung der Samples auf dem Display.

- **Schritte:**

1. Fader-Werte anzeigen:
 - (a) Bewege die Schiebepotentiometer (Fader) zyklisch.
 - (b) Überprüfe, ob die prozentualen Werte der Potentiometer korrekt und in Echtzeit auf dem Display angezeigt werden.
 - (c) Vergewissere dich, dass die angezeigten Werte die richtige Spannung und die korrekte Zuordnung zu den Klassen (wie Rhythmic, Sustained, usw.) widerspiegeln.
2. Filterung der Samples:
 - (a) Bewege die Schiebepotentiometer und stelle den Schwellenwert für die Filterung ein.
 - (b) Überprüfe, ob die Liste der Samples auf dem Display dynamisch aktualisiert wird und nur die Samples anzeigt, deren Audio-Klassen nach der Klassifizierung den Schwellenwert nicht überschreiten.
 - (c) Teste verschiedene Schwellenwerte und überprüfe, ob die angezeigten Samples korrekt gefiltert werden.

- **Erwartete Ergebnisse:**

- Die prozentualen Werte der Fader werden korrekt und in Echtzeit auf dem Display angezeigt.
- Die Klassen der Potentiometer-Einstellungen werden korrekt auf dem Display angezeigt.
- Die Liste der Samples wird dynamisch und korrekt basierend auf den Potentiometer-Werten und dem Schwellenwert aktualisiert.
- Keine Anzeigeprobleme oder Verzögerungen bei der Aktualisierung des Displays.

- **Testergebnisse:**

- Die prozentualen Fader-Werte wurden korrekt auf dem Display angezeigt und spiegelten die realen Werte der Potentiometer wider.
- Die Filterung der Samples funktionierte wie erwartet; nur die Samples, die den eingestellten Schwellenwert erfüllten, wurden angezeigt.
- Die Anzeige auf dem Display war frei von Verzögerungen oder Fehlern.

6.3 Testprotokoll: LF01 LF02 SD-Karten SPI-Schnittstelle

LF01
LF02

6.3.1 Testzielsetzung

Dieser Test überprüft die Funktionalität der Lese- und Schreibfunktionen für die SD-Karte über die SPI-Schnittstelle.

6.3.2 Testdurchführung

- **Mounten des Dateisystems:**
 - Die Funktion `f_mount` wurde aufgerufen, um das Dateisystem der SD-Karte zu mounten.
 - Der Rückgabewert (`fres`) wurde überprüft. Im Fehlerfall wurde eine Meldung ausgegeben und der Test abgebrochen.
- **SD-Karten-Statistiken:**
 - Die Funktion `f_getfree` wurde aufgerufen, um die freien Cluster, freien Sektoren und Gesamtsektoren der SD-Karte zu ermitteln.
 - Im Fehlerfall wurde eine Meldung ausgegeben und der Test abgebrochen.
 - Die Gesamt- und freien Speicherplatzwerte wurden berechnet und über `myprintf` ausgegeben.
- **Lesen einer Datei:**
 - Die Datei `test.txt` wurde mit der Funktion `f_open` im Lesemodus geöffnet.
 - Der Rückgabewert (`fres`) wurde überprüft. Im Fehlerfall wurde eine Meldung ausgegeben und der Test abgebrochen.
 - Es wurde versucht, 30 Bytes aus der Datei zu lesen (`f_gets`).
 - Gelingt das Lesen, wurde der Inhalt der gelesenen Daten über `myprintf` ausgegeben.
 - Im Fehlerfall wurde eine Meldung ausgegeben.
 - Die Datei wurde mit `f_close` geschlossen.
- **Schreiben einer Datei:**
 - Die Datei `write.txt` wurde mit der Funktion `f_open` im Schreibmodus und mit Flags zum Anlegen der Datei geöffnet.
 - Der Rückgabewert (`fres`) wurde überprüft. Im Fehlerfall wurde eine Meldung ausgegeben.
 - Ein String (''a new file is made; '') wurde in den Puffer `readBuf` kopiert.
 - Die Daten aus `readBuf` wurden mit `f_write` in die Datei geschrieben.
 - Die Anzahl der geschriebenen Bytes wurde über `myprintf` ausgegeben.
 - Im Fehlerfall wurde eine Meldung ausgegeben.
 - Die Datei wurde mit `f_close` geschlossen.
- **Dismounten des Dateisystems:**
 - Die Funktion `f_mount` wurde mit `NULL` aufgerufen, um das Dateisystem der SD-Karte zu dismounten.

6.3.3 Testergebnisse

Der Test verlief erfolgreich:

- Das Mounten und Dismounten des Dateisystems wurden erfolgreich durchgeführt.
- Die SD-Karten-Statistiken stimmten mit den Erwartungen überein.
- Das Lesen und Schreiben von .txt Dateien funktionierte einwandfrei.

Es wurde festgestellt, dass das Schreiben eines ‘struct‘ nicht möglich war. Eine zukünftige Lösung wird angestrebt.

6.4 Testprotokoll zu /LF03/

LF03

6.4.1 Validierung der Sinnhaftigkeit der Merkmalsklassen

Die Sinnhaftigkeit und Relevanz der ausgewählten Merkmalsklassen wurden durch eine musikaffine Person, die gleichzeitig ein Gruppenmitglied ist, bestätigt. Da die Relevanz der ausgewählten Merkmalsklassen für die Musikproduktion jedoch nicht sachlich sinnvoll in einem Testfall validiert werden kann, wird stattdessen die Fähigkeit zur Klassifizierung auf der Grundlage der auf einem Mikrocontroller verarbeitbaren Spektrogrammauflösung in die ausgewählten Merkmalsklassen betrachtet. Dabei wird zunächst ausschließlich die theoretische Umsetzbarkeit vor dem Training des neuronalen Netzes geprüft.

Testfall: Testen der Umsetzbarkeit der Merkmalsklassen vor dem Training

Schritte:

1. Für jede Merkmalsklasse (bass, pitched, rhythmic, sustained, melodic) werden 1-3 passende Audiosamples ermittelt.
2. Diese Audiosamples werden in Audiosubsamples zerteilt und Spektrogramme in der festgelegten Auflösung (32x30 Pixel) generiert. Dies geschieht mit dem “ESP_IMA_validation” Jupyter-Notebook.
3. Anschließend wird manuell geprüft, ob mit dem menschlichen Auge bestimmte Muster sichtbar sind, anhand derer man als Mensch die Spektrogramme einer Merkmalsklasse zuordnen könnte.

Erwartetes Ergebnis: Für jede Merkmalsklasse sind bestimmte Merkmale in den generierten Spektrogrammen sichtbar, die diese Merkmalsklasse deutlich von anderen Merkmalsklassen unterscheidet. Dadurch ist die theoretische Umsetzbarkeit durch ein neuronales Netz gewährleistet.

Tatsächliches Ergebnis: Für jede Merkmalsklasse lassen sich in den Spektrogrammen die folgenden Merkmale erkennen:

- **bass:** Überwiegend hell gefärbt (hohe Dezibel Zahl) in den tiefen Frequenzbereichen.
- **pitched:** Überwiegend hell gefärbt (hohe Dezibel Zahl) in den hohen Frequenzbereichen.
- **rhythmic:** Im zeitlichen Verlauf immer wieder vertikale “Trennlinien“.
- **sustained:** Im zeitlichen Verlauf häufig durchgehende Linien für eine bestimmte Frequenz.
- **melodic:** Oft sind überlagerte, hell gefärbte und wellenförmige Segmente zu sehen.

Damit ist die theoretische Umsetzbarkeit der Klassifizierung dieser Merkmalsklassen auf Basis von Spektrogrammen mit der Auflösung 32x30 Pixel gewährleistet.

6.4.2 Validierung des erfolgreichen Training des neuronalen Netzes

Testfall: Überprüfen der Ausgabeparameter des Trainings

Schritte:

1. Das neuronale Netz wird im Jupyter-Notebook “ESP_IMA_dev” trainiert.
2. Die Ausgabeparameter werden betrachtet: Accuracy, Confusion-Matrix.

Erwartetes Ergebnis:

- Der Accuracy-Wert liegt bei 50% oder höher.
- In der Confusion-Matrix lässt sich auf der Diagonalen (links oben - rechts unten) eine Tendenz bei der Klassifikation ableiten, also dass die Merkmalsklassen tendenziell richtig erkannt werden.

Tatsächliches Ergebnis: Der Accuracy-Wert liegt bei den Testdaten bei 71% und damit höher als angestrebte. Dabei muss jedoch der in **Abschnitt 5.3.4** beschriebene methodische Fehler bei der Datenaufteilung im Hinterkopf behalten werden.

In der Confusion-Matrix, die in **Abbildung 34** dargestellt ist, lässt sich insgesamt auf der beschriebenen Diagonalen eine Tendenz für die korrekte Klassifikation jeder Merkmalsklasse ableiten. Auf der Ordinate stehen Merkmalsklassen, die in das System eingegeben wurden (IST-Wert), auf der Abszisse lässt sich dann das tatsächliche Klassifikationsergebnis ablesen. Wie hier zu sehen ist, werden sowohl die Klassen "melodic" als auch "rhythmic" fast immer korrekt erkannt (Wert nahe 1.0). Tendenziell richtig werden "bass" und "pitched" erkannt. Die Klasse "sustained" wird dagegen nicht gut erkannt und wird oft fälschlicherweise als "melodic" erkannt.

Die Performance des neuronalen Netzes könnte mit weiteren Trainingsdaten für die Merkmalsklassen "bass", "pitched" und "sustained" noch einmal verbessert werden, indem diese Trainingsdaten die jeweilige Klasse möglichst in Reinform enthalten.

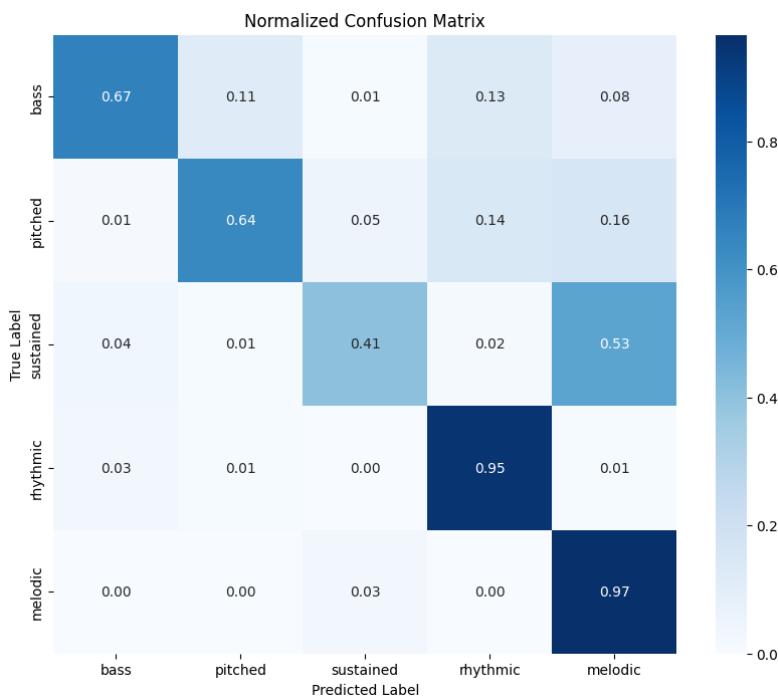


Abbildung 34: Confusion-Matrix" des trainierten neuronalen Netzes

Testfall: Überprüfen der Sinnhaftigkeit des Klassifikationsergebnisses

Schritte:

1. Es werden vier Audiosamples ermittelt, die nicht in den Trainingsdaten enthalten waren.
2. Die Audiosamples werden wiedergegeben und notiert, welchen Merkmalsklassen sie jeweils zugeordnet sein sollten.
3. Die fünf Audiosamples werden nacheinander mit dem Jupyter-Notebook "ESP_IMA_validation" klassifiziert.
4. Die Klassifikationsergebnisse werden betrachtet und mit den notierten SOLL-Merkmalsklassen verglichen.

Erwartetes Ergebnis: Die tatsächlichen Klassifikationsergebnisse (SOLL-Werte) decken sich mit den notierten Klassifikationsergebnissen (IST-Werte). Dabei wird ein Output-Wert je Merkmalsklasse, der $\geq 0,5$ ist, als "erkannt" gewertet.

Tatsächliches Ergebnis:

1. Rock-Song

	bass	pitched	rhythmic	sustained	melodic
SOLL-Ausgabe	0	0	1	0	1
IST-Ausgabe	0.09	0.28	0.48	0.31	0.76

2. Klaviermusik

	bass	pitched	rhythmic	sustained	melodic
SOLL-Ausgabe	0	0	0	1	1
IST-Ausgabe	0.14	0.14	0.01	0.85	0.92

3. Schlagzeug-Solo

	bass	pitched	rhythmic	sustained	melodic
SOLL-Ausgabe	0	0	1	0	0
IST-Ausgabe	0.16	0.24	0.91	0.00	0.10

4. Action-Song

	bass	pitched	rhythmic	sustained	melodic
SOLL-Ausgabe	1	0	1	0	0
IST-Ausgabe	0.41	0.23	0.97	0.03	0.36

Wie in den Beispielen erkennbar ist, werden Merkmalsklassen nicht immer gut erkannt, jedoch lässt sich auch hier eine Tendenz ableiten. Beim Rock-Song beispielsweise liegt der Ähnlichkeitswert für die Klasse "rhythmic" mit 0,48 nur knapp unterhalb der Schwelle von 0,5. Beim Action-Song wird das Merkmal "bass" mit einer Ähnlichkeit von 0,41 erkannt, was auch nicht allzu weit weg ist vom Schwellenwert. Dennoch bleibt wie zuvor erwähnt noch Optimierungspotenzial mit mehr Trainingsdaten für bestimmte Klassen, um das Merkmalspektrum besser abzubilden.

6.4.3 Validierung der korrekten Funktion der Klassifikation auf dem STM32 Microcontroller

Testfall: Überprüfen der berechneten Spektrogramm-Werte

Schritte:

1. Ein Audiosample wird in das Jupyter-Notebook "ESP_IMA_validation" geladen und auf 16 kHz heruntergesampled.
2. Die ersten 16.896 Samples (das erste Audiosubsample) des Audiosamples wird in Form von C-Code in Datei `test_audio_data.c` im CubeMX Projekt "esp-cubeai-test" geladen.
3. Ein Breakpoint wird vor der Standardisierung der Eingabewerte gesetzt.
4. Der Code wird via Debugger auf das Nucleo-Board geflashed.
5. Es wird zu dem gesetzten Breakpoint gegangen und in CubemMX eine Expression für die Variable `spectrogram` gesetzt.

6. Die Werte des Arrays werden mit den Spektrogramm-Werten des ersten Audiosubsamples im Jupyter-Notebook verglichen.

Erwartetes Ergebnis: Es gibt keine bzw. kaum Abweichungen zwischen den Spektrogramm-Werten im Debugger und den Werten aus dem Jupyter-Notebook.

Tatsächliches Ergebnis: Von den Werten 0 bis 700 wurden keine Abweichungen festgestellt. Nach diesem Bereich treten jedoch leichte Abweichungen auf. Diese können darauf zurückgeführt werden, dass die höheren Indizes des `spectrogram`-Arrays die Dezibel-Werte der höheren Frequenzbereiche enthalten. Es lässt sich daraus schließen, dass die verwendete Bibliothek zur Spektrogramm-Berechnung die höheren Frequenz-Werte anders skaliert. Insgesamt kann der Test mit einem positiven Ergebnis bewertet werden, wobei darauf hingewiesen werden muss, dass die leichten Abweichungen in der Spektrogramm-Berechnung zu geringfügigen Abweichungen in den Klassifikationsergebnissen führen könnten.

Testfall: Überprüfen des Klassifikationsergebnisses

Schritte:

1. Ein Audiosample wird in das Jupyter-Notebook “ESP_IMA_validation“ geladen und auf 16 kHz heruntergesampled.
2. Die ersten 16.896 Samples (das erste Audiosubsample) des Audiosamples wird in Form von C-Code in Datei `test_audio_data.c` im CubeMX Projekt “esp-cubeai-test“ geladen.
3. Ein Breakpoint wird nach der Zeile des Funktionsaufrufs von `ai_network_1_run()` gesetzt.
4. Der Code wird via Debugger auf das Nucleo-Board geflashed.
5. Es wird zu dem gesetzten Breakpoint gegangen und im CubemMX Debugger eine Expression für die Variable `aiOutData` gesetzt.
6. Die Werte des Arrays werden mit dem Klassifikationsergebnis des ersten Audiosubsamples im Jupyter-Notebook verglichen.

Erwartetes Ergebnis: Es gibt kaum Abweichungen zwischen den Werten im Debugger und den Werten aus dem Jupyter-Notebook. Eine geringe Abweichung ist zu erwarten, da wie im Ergebnis des vorherigen Testfalls beschrieben die berechneten Spektrogramm-Werte nicht identisch sind.

Tatsächliches Ergebnis: Folgende IST-Werte wurden aus dem Debugger ausgelesen. Ein Screenshot Tabs “Expressions“, in dem die Variableninhalte zur Laufzeit gezeigt werden, ist in **Abbildung 35** zu sehen.

	bass	pitched	rhythmic	sustained	melodic
SOLL-Ausgabe	0.12	0.28	0.06	0.93	0.37
IST-Ausgabe	0.13	0.33	0.048	0.95	0.31

Wie erwartet gibt es leichte Abweichungen im Klassifikationsergebnis. Dies ist wie zuvor erwähnt auf die abweichenden Spektrogramm-Werte zurückzuführen. Das Testergebnis ist als positiv zu beurteilen.

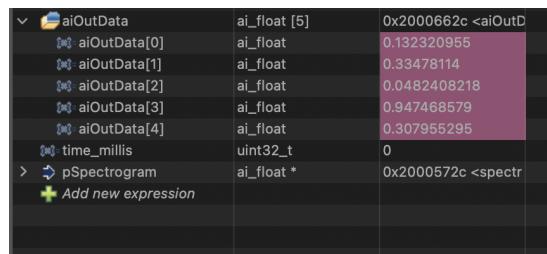


Abbildung 35: Screenshot vom CubeMX Debugger. In den Elementen des “aiOutData“-Arrays kann das Klassifikationsergebnis abgelesen werden

6.5 Testprotokoll zu /LF05/ Wiedergabe über Audio Codec

LF05

Ziel: Überprüfung der Grundfunktionalität und Ansteuerung des Audio Codecs.

Durchführung: Um Grundfunktionalität des Audio Codecs sicherstellen, wird die Audiowiedergabe ohne die Verwendung von Dateizugriffen von der SD-Karte durchgeführt. Somit wird eine mögliche Fehlerquelle eliminiert, was die Fehlersuche deutlich vereinfacht. Die Audiodaten, in Form einer Sinuswelle, werden in einen Lookup-Table generiert.

```

15 volatile int16_t *outBufPtr = &dacData[0];
16 volatile bool pitchChanged = false;
17
18 #define SINE_TABLE_SIZE 256 // Adjust the table size as needed for your
→   accuracy/performance tradeoff
19

```

Dieser Lookup-Table wird dann iterativ ausgelesen und mit folgender Funktion in den Audiobuffer geschrieben.

```

55 * @brief Populates half of the DAC buffer with a sine wave using a lookup table.
56 *
57 * The function generates a sine wave based on the provided frequency. It uses a
58 * lookup table to fill the DAC buffer with sine values and manages the phase
→   index
59 * for the waveform. It waits for the DMA to be ready before updating
60 * the buffer.
61 *
62 * @param frequency Desired frequency of the sine wave.
63 */
64 void generateSineWave(double frequency) {
65     static uint16_t phaseIndex = 0;           /**< Retains the index between
→   function calls */
66     double phaseIncrement = frequency * SINE_TABLE_SIZE / I2S_AUDIOFREQ_44K;
67     HAL_I2S_Transmit_DMA(&hi2s2, (void *)dacData, BUFFER_SIZE);
68
69     while(1){
70         if(dma_dataReady){
71             for (uint8_t n = 0; n < (HALF_BUFFER_SIZE) - 1; n += 2) {
72                 // Lookup sine value from table
73                 outBufPtr[n] = sineTable[phaseIndex];
74
75                 // Increment phase index
76                 phaseIndex += phaseIncrement;
77

```

Schritte:

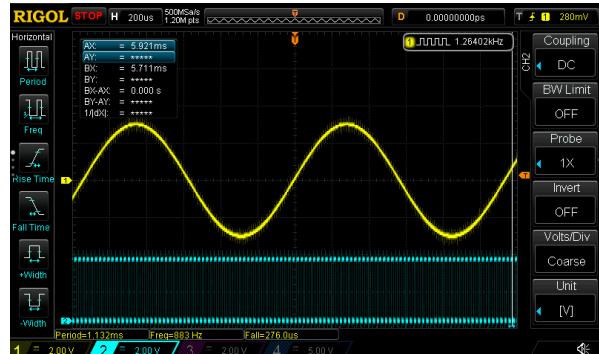
1. System gemäß dem Schaltplan verbinden, wobei der Audio Codec und die Lautsprecher verbunden sind.
2. Sinuswelle per Lookup Table Generieren.
3. Wiedergabe der Sinuswelle über den Audio Codec.
4. Anschluss von Oszilloskop-Sonde mit dem Audioausgang.
5. Visuelle Analyse der Wellenform über Oszilloskop
6. Auditiv Analyse der Sinuswelle über den Lautsprecher.

Erwartete Werte:

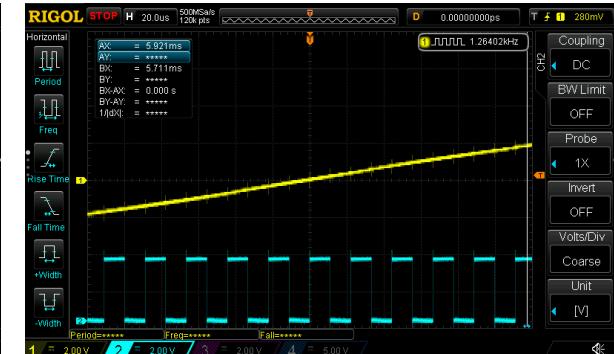
- Die Sinuswelle sollte im Oszilloskop klar und sauber dargestellt werden.
- Die Audioausgabe sollte frei von Störgeräuschen wie Knacken oder Verzerrungen sein.
- Die Sinuswelle sollte auditiv klar und konsistent über die Lautsprecher wiedergegeben werden.

Testergebnisse:

- Oszilloskop zeigt eine saubere Sinuswelle ohne Verzerrungen.
- Die Audioausgabe ist frei von Störgeräuschen.
- Die Sinuswelle klingt sauber und klar über die Lautsprecher.



(a) Oszilloskopansicht der Sinuswellenwiedergabe



(b) Nahe Oszilloskopansicht der Sinuswellenwiedergabe

Die Tests des Pitchfaktors zeigen, dass die Sinuswellenwiedergabe bei allen getesteten Werten sauber und klar ist. Kleinere Abweichungen von den erwarteten Frequenzen sind vorhanden, beeinflussen jedoch nicht die Audioqualität. Die Wiedergabe bleibt störungsfrei und konsistent über die getesteten Pitchfaktoren hinweg.

6.6 Testprotokoll zu /LF06/ Pitchgenauigkeit des Wiedergabealgorithmus

LF06

Ziel: Überprüfung der Pitchgenauigkeit des Wiedergabealgorithmus.

Durchführung: Um die Pitchgenauigkeit des Wiedergabealgorithmus zu überprüfen, wird ein 440 Hz Audiofile mit verschiedenen Tonhöhen über den Wiedergabealgorithmus abgespielt.

Der Pitchfaktor, also der Faktor um den die Wiedergabe verschnellert/verlangsamt wird, soll nicht über eine externen Potentiometer oder ähnlicher Hardware gesteuert werden, um unpräzise Werte zu vermeiden. Er im folgender Funktion mit `player->pitchFactor = 2.0f;` präzise festgelegt (hardcoded):

```

27 void initPlayer(WavPlayer *player, FIL *file, wav_header_t *wavHeader) {
28     player->file = file;
29     player->wavHeader = wavHeader;
30     player->restartPlayback = false;
31     player->playbackActive = false;
32     player->headerSize = 0;
33     player->pitchFactor = 1.0f;
34     player->pitchChanged = false;
35 }
```

Schritte:

1. System gemäß dem Schaltplan verbinden, wobei der Audio Codec und die Lautsprecher verbunden sind.
2. Anschluss von Oszilloskop-Sonde mit dem Audioausgang.
3. Pitchfactor auf 1.0f setzen.
4. Wiedergabe des 440 Hz Testfile über den Audio Codec.
5. Visuelle Analyse der Wellenform über Oszilloskop (Messung der Frequenz).
6. Auditiv Analyse der Sinuswelle über den Lautsprecher.
7. Pitchfactor auf 0.5f setzen.
8. Wiedergabe des 440 Hz Testfile über den Audio Codec.
9. Visuelle Analyse der Wellenform über Oszilloskop (Messung der Frequenz).
10. Auditiv Analyse der Sinuswelle über den Lautsprecher.
11. Pitchfactor auf 2.0f setzen.
12. Wiedergabe des 440 Hz Testfile über den Audio Codec.
13. Visuelle Analyse der Wellenform über Oszilloskop (Messung der Frequenz).
14. Auditiv Analyse der Sinuswelle über den Lautsprecher.

Erwartete Werte:

- Die Sinuswelle sollte im Oszilloskop mit jedem Pitchfaktor klar und sauber dargestellt werden.
- Die Audioausgabe sollte frei von Störgeräuschen wie Knacken oder Verzerrungen sein.
- Die Sinuswelle sollte auditiv klar und konsistent über die Lautsprecher wiedergegeben werden.
- Die erwartete Frequenzen sind:
 - Pitchfaktor = 1.0 → 440 Hz
 - Pitchfaktor = 0.5 → 220 Hz
 - Pitchfaktor = 2.0 → 880 Hz

Testergebnisse:

- Bei einem Pitchfaktor von 1.0 beträgt die gemessene Frequenz 450 Hz anstelle der erwarteten 440 Hz. Abgesehen von dieser Abweichung zeigt die Sinuswelle eine klare und saubere Darstellung im Oszilloskop, und die Audioausgabe ist frei von Störgeräuschen. Die Sinuswelle wird auditiv klar und konsistent über die Lautsprecher wiedergegeben.
- Bei einem Pitchfaktor von 0.5 beträgt die gemessene Frequenz 225 Hz anstelle der erwarteten 220 Hz. Die Frequenz ist wie erwartet, halb so groß wie bei einem Pitchfaktor von 1.0, allerdings zeigen sich kleine Spitzen in der Wellenform auf dem Oszilloskop, die jedoch nicht hörbar sind.
- Bei einem Pitchfaktor von 2.0 beträgt die gemessene Frequenz 890 Hz anstelle der erwarteten 880 Hz. Die Frequenz ist wie erwartet, doppelt so groß wie bei einem Pitchfaktor von 1.0. Abgesehen von dieser Abweichung zeigt die Sinuswelle eine klare und saubere Darstellung im Oszilloskop, und die Audioausgabe ist frei von Störgeräuschen. Die Sinuswelle klingt sauber und klar über die Lautsprecher.



Abbildung 37: Frequenzmessung bei Pitchfaktor 0.5

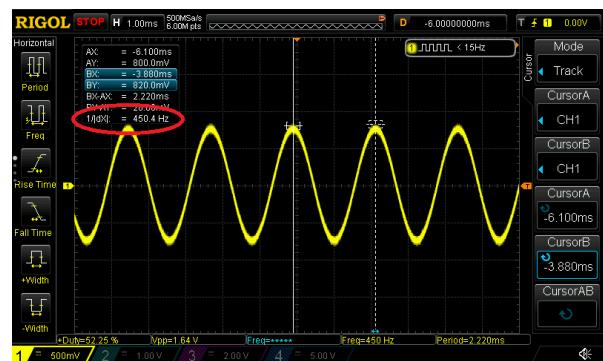


Abbildung 38: Frequenzmessung bei Pitchfaktor 1.0

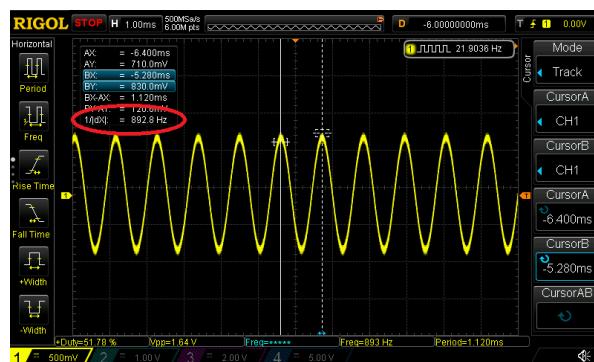


Abbildung 39: Frequenzmessung bei Pitchfaktor 2.0

Die Grundfunktionalität und Ansteuerung des Audio Codecs ist sichergestellt. Die Sinuswellenwiedergabe erfolgt sauber und klar, sowohl visuell als auch auditiv, was auf eine erfolgreiche Integration und Funktionsweise des Audio Codecs hinweist.

6.7 Testprotokoll zu /LQ01/ Latenzmessung

LQ01

Messung der Latenz vom Zeitpunkt des Triggerinputs bis zur Audioausgabe über den Audio-Codec.

Durchführung mit dem STM32-Audioprojekt aus dem Repository ([/f401_sd_card_audio_codec_test/](#)). Der Audio-Codec, sowie der SD-Kartenleser müssen wie im Schaltplan 4.4.4 verbunden werden.

Schritte:

1. Anschluss zweier Oszilloskop-Sonden: an Trigger-Input/Play-Button-Pin und Audio-Ausgang.
2. Laden einer Test PCM .wav-Datei mit einer Samplerate von 44.1 kHz, 16 Bit und Stereo/2 Kanälen auf die SD-Karte.
3. Umschalten des Oszilloskops in den Single-Shot-Modus und Konfiguration zur Auslösung durch den Audio-Kanal.
4. Abspielen der Testdatei durch Auslösen des Play-Buttons.
5. Positionierung zweier Mess-Cursor: einer auf die erste Flanke des Trigger-Input-Signals und der andere auf den Beginn des Audio-Ausgangssignals.

Erwartete Werte Die erwartete Latenz errechnet sich wie folgt:

Gegeben:

$$\text{Samplerate} = 44.1 \text{ kHz} = 44,100 \text{ Samples pro Sekunde}$$

$$\text{Puffergröße} = 256 \text{ Samples}$$

Berechnung der Latenz:

$$\text{Zeit pro Sample} = \frac{1 \text{ Sekunde}}{44,100 \text{ Samples}} \approx 22.68 \mu\text{s}$$

$$\text{Zeit für einen Puffer} = 256 \text{ Samples} \times 22.68 \mu\text{s} \approx 5.8 \text{ ms}$$

Die Latenz für das System mit den angegebenen Parametern beträgt also im Idealfall 5.8 ms.

Testergebnisse

- Gemessene Latenz von 6.4 ms (**BX-AX** in Abbildung 40).
- Die Abweichung von der erwarteten Latenz ist wahrscheinlich auf die Verarbeitungszeiten des Audio-Codecs zurückzuführen.
- Sehr praktikabler Latenzwert für Audioinstrument.

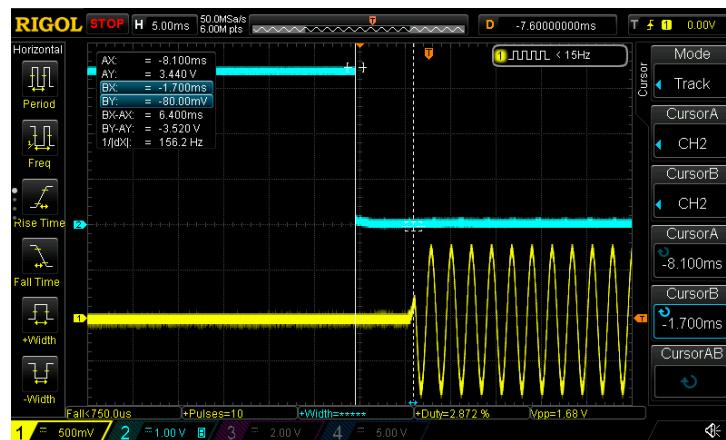


Abbildung 40: Oszilloskopansicht der Latenzmessung von Trigger (Blau) bis Audioausgang (Gelb)

7 Fazit

Im Rahmen des Projekts konnten die in der Vorlesung “Embedded Systems“ (ES) erlernten Kenntnisse, vor allem die Arbeit mit dem STM32 Ecosystem, weiter vertieft und um einige Themenbereiche, Technologien und Protokolle erweitert werden. Die praktische Anwendung dieser Kenntnisse ermöglichte es den Gruppenmitgliedern, spezialisiertes Wissen und wertvolle Erfahrungen zu sammeln, die sowohl technische als auch organisatorische Fähigkeiten umfassten. Die eigenständige Entwicklung eines Systems von der Idee bis zur Implementierung förderte nicht nur das tiefgreifende Verständnis für die verwendeten Technologien, sondern auch die Teamfähigkeit und Problemlösungskompetenz der Teilnehmer.

Die Herausforderungen bei der Umsetzung komplexer Funktionalitäten und die damit verbundenen Probleme führten jedoch zu Zeitverzögerungen, die eine vollständige Fertigstellung des Projekts innerhalb des vorgegebenen Zeitrahmens verhinderten. Trotz dieser Schwierigkeiten war das Projekt eine bereichernde Erfahrung, insbesondere im Hinblick auf die Organisation und Durchführung von zukünftigen Projekten.

Es lässt sich festhalten, dass das Projekt trotz der Hindernisse eine wertvolle Lerngelegenheit bot, durch die die Teammitglieder nicht nur fachliche, sondern auch überfachliche Kompetenzen erweitern konnten. Diese Erfahrungen stellen eine wichtige Grundlage für die erfolgreiche Durchführung zukünftiger Projekte dar.

Da das Projekt sehr von privatem Interesse vorangetrieben wird, ist davon auszugehen, dass es in naher Zukunft zu einer Weiterentwicklung und im Besten Falle Fertigstellung dieses Eurorack-Samplers kommen wird.

Die Integration klassischer Bereiche der eingebetteten Systeme, wie analoge Steuerung, Nutzerinteraktion und PCB-Design, mit neuartiger EdgeAI-Technologie bietet weiterhin ein besonders reizvolles Potenzial.

Literaturverzeichnis

- [1] T. Ferguson. "Interview with Olivier Gillet (Mutable Instruments)." Zugegriffen am: 01.08.2024. (2015), Adresse: <https://www.keithmcmillan.com/interview-with-olivier-gillet-mutable-instruments>.
- [2] T. B. Holmes, *Electronic and Experimental Music: Pioneers in Technology and Composition*, 2nd. London: Routledge Music/Songbooks, 2002, (cloth) (pbk), ISBN: 0-415-93643-8.
- [3] W. Groves. "Intro to Eurorack Part I: Doepfer's Beginnings and Power Supply Basics." Zugegriffen am: 02.08.2024. (2020), Adresse: <https://reverb.com/news/beginners-guide-to-eurorack-case-basics-oscillators-filters>.
- [4] *Mechanical structures for electronic equipment - Dimensions of mechanical structures of the 482.6 mm (19 in) series - Part 3-108: Dimensions of R-type subracks and plug-in units*, 2014.
- [5] A. Aciman. "Meet the unassuming drum machine that changed music forever." Zugegriffen am: 02.08.2024. (2018), Adresse: <https://www.vox.com/culture/2018/4/16/16615352/akai-mpc-music-history-impact>.
- [6] elektron. "Octatrack MKII official website." Zugegriffen am: 02.08.2024. (2024), Adresse: <https://www.elektron.se/us/octatrack-mkii-explorer>.
- [7] STMicroelectronics. "NUCLEO F401RE Datasheet." Zugegriffen am: 06.08.2024. (2023), Adresse: https://www.st.com/resource/en/data_brief/nucleo-f401re.pdf.
- [8] STMicroelectronics. "cubeide-sd-card." Zugegriffen am: 06.08.2024. (2023), Adresse: <https://www.st.com/en/microcontrollers-microprocessors/stm32f722ze.html>.
- [9] H. Fidelity. "How Much Latency Can Live Musicians Tolerate?" Zugegriffen am: 01.08.2024. (2013), Adresse: <https://www.hifidelity.com/backlog/how-much-latency-can-live-musicians-tolerate-da8e2ebe587a>.
- [10] D. Katz, R. Gentile und T. Lukasiak. "Fundamentals of embedded audio, part 3." Zugegriffen am: 01.08.2024. (2007), Adresse: <https://www.eetimes.com/fundamentals-of-embedded-audio-part-3/>.
- [11] PHILIPS. "I2S bus specification." Zugegriffen am: 01.08.2024. (1996), Adresse: https://web.archive.org/web/20070102004400/http://www.nxp.com/acrobat_download/variou/I2SBUS.pdf.
- [12] CMSIS-DSP. "CMSIS-DSP documentation." Zugegriffen am: 06.08.2024. (2023), Adresse: <https://arm-software.github.io/CMSIS-DSP/latest/>.
- [13] M. Gupta. "Multi-label classification for beginners with codes." Zugegriffen am: 30.07.2024. (2023), Adresse: <https://medium.com/data-science-in-your-pocket/multi-label-classification-for-beginners-with-codes-6b098cc76f99>.
- [14] STMicroelectronics. "How to perform motion sensing on STM32L4 IoTnode." Zugegriffen am: 30.07.2024. (2022), Adresse: https://wiki.st.com/stm32mcu/wiki/AI:How_to_perform_motion_sensing_on_STM32L4_IoTnode#Create_an_STM32Cube-AI_application_using_X-CUBE-AI.
- [15] O. A. Montesinos-López und J. Crossa. "Multivariate Statistical Machine Learning Methods for Genomic Prediction." Zugegriffen am: 30.07.2024. (2022), Adresse: <https://www.ncbi.nlm.nih.gov/books/NBK583971/>.
- [16] P. Kebal. "Audio File Format Specifications." Zugegriffen am: 30.07.2024. (2022), Adresse: <https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>.
- [17] G. Berry und L. Boutillette. "The best audio format types for audiophiles." Zugegriffen am: 30.07.2024. (2024), Adresse: <https://www.adobe.com/creativecloud/video/discover/best-audio-format.html>.
- [18] V. K. Garg. "An Overview of Digital Communication and Transmission." Zugegriffen am: 30.07.2024. (2007), Adresse: <https://www.sciencedirect.com/topics/engineering/pulse-code-modulation>.
- [19] HuggingFace. "Audio Course, Kapitel "Introduction to audio data"" Zugegriffen am: 03.08.2024. (), Adresse: https://huggingface.co/learn/audio-course/chapter1/audio_data.
- [20] H. Nyquist. "Certain Topics in Telegraph Transmission Theory." Zugegriffen am: 30.07.2024. (1928), Adresse: <https://bayes.wustl.edu/Manual/CertainTopicsInTelegraphTransmissionTheory.pdf>.
- [21] MathWorks. "Die Funktionsweise von CNNs." Zugegriffen am: 30.07.2024. (), Adresse: <https://de.mathworks.com/discovery/convolutional-neural-network.html>.

- [22] L. Roberts. "Understanding the Mel Spectrogram." Zugegriffen am: 30.07.2024. (2020), Adresse: <https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53>.
- [23] STMicroelectronics. "Acoustic scene classification." Zugegriffen am: 30.07.2024. (), Adresse: https://www.st.com/content/st_com/en/st-edge-ai-suite/case-studies/acoustic-scene-classification.html.
- [24] STMicroelectronics. "Audio scene classification using machine learning on STM32." Zugegriffen am: 30.07.2024. (), Adresse: <https://www.st.com/en/solutions-reference-designs/sl-auid011501v1.html>.
- [25] Z. ao Huang, Y. Sang, Y. Sun und J. Lv. "A neural network learning algorithm for highly imbalanced data classification." Zugegriffen am: 30.07.2024. (2022), Adresse: <https://www.sciencedirect.com/science/article/pii/S0020025522009847>.
- [26] A. Tirawi. "The Crucial Role of Normalization and Standardization in Enhancing Machine Learning Models." Zugegriffen am: 03.08.2024. (2023), Adresse: <https://medium.com/@aryantiwari7777/the-crucial-role-of-normalization-and-standardization-in-enhancing-machine-learning-models-71afa24a79a>.
- [27] A. S. Gillis. "Definition: data splitting." Zugegriffen am: 03.08.2024. (2023), Adresse: <https://www.techtarget.com/searchenterpriseai/definition/data-splitting#:~:text=In%20machine%20learning%2C%20data%20splitting,into%20three%20or%20four%20sets>.
- [28] A. Thakur. "ReLU vs. Sigmoid Function in Deep Neural Networks." Zugegriffen am: 03.08.2024. (2022), Adresse: <https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--Vm1ldzoyMDk0MzI>.
- [29] EITCA. "What are the different formats of the model file in TensorFlow Lite and what information do they contain?" Zugegriffen am: 03.08.2024. (2023), Adresse: [https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/programming-tensorflow/introducing-tensorflow-lite/examination-review-introducing-tensorflow-lite/what-are-the-different-formats-of-the-model-file-in-tensorflow-lite-and-what-information-do-they-contain/#:~:text=tflite\)%2C%20TensorFlow%20Lite%20model%20format, on%20mobile%20and%20embedded%20devices..](https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/programming-tensorflow/introducing-tensorflow-lite/examination-review-introducing-tensorflow-lite/what-are-the-different-formats-of-the-model-file-in-tensorflow-lite-and-what-information-do-they-contain/#:~:text=tflite)%2C%20TensorFlow%20Lite%20model%20format, on%20mobile%20and%20embedded%20devices..)
- [30] STMicroelectronics. "STM32Cube function pack for ultra-low power IoT node with artificial intelligence (AI) application based on audio and motion sensing." Zugegriffen am: 03.08.2024. (), Adresse: <https://www.st.com/en/embedded-software/fp-ai-sensing1.html>.
- [31] Omron. "Technical Explanation for Rotary Encoders." Zugegriffen am: 06.08.2024. (n/a), Adresse: https://www.ia.omron.com/data_pdf/guide/34/rotary_tg_e_7_2.pdf.
- [32] R. Y. Jonathan Valvano. "Chapter 14: Analog to Digital Conversion, Data Acquisition and Control Modified to be compatible with EE319K Lab 8." Zugegriffen am: 06.08.2024. (2014), Adresse: https://users.ece.utexas.edu/~valvano/Volume1/E-Book/C14_ADCdataAcquisition.htm.
- [33] M. Hasan. "STM32 ADC tutorial using DMA with HAL Code Example." Zugegriffen am: 06.08.2024. (2024), Adresse: <https://embeddedthere.com/stm32-adc-tutorial-using-dma-with-hal-code-example/>.
- [34] C. Nelson. "Working with I2C Devices." Zugegriffen am: 06.08.2024. (2024), Adresse: <https://learn.adafruit.com/working-with-i2c-devices?view=all>.
- [35] A. Alekseev. "stm32-ssd1306." Zugegriffen am: 06.08.2024. (2024), Adresse: <https://github.com/afiskon/stm32-ssd1306>.
- [36] H. Pearce. "cubeide-sd-card." Zugegriffen am: 06.08.2024. (2023), Adresse: <https://github.com/kiwihi/cubeide-sd-card>.

A

A

B

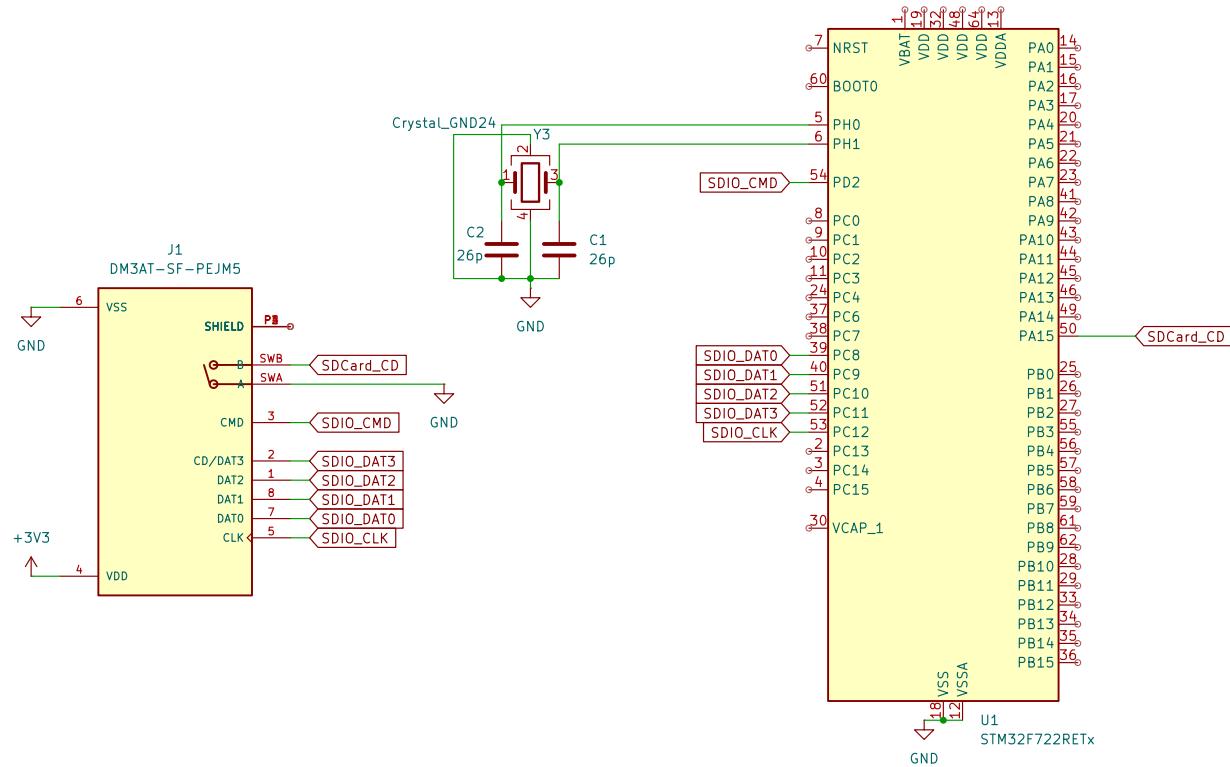
B

C

C

D

D



Audio Codec

File: audio_codec.kicad_sch

Sheet: /
File: ima_new_codec.kicad_sch

Title:

Size: A4 | Date:
KiCad E.D.A. 8.0.4

Rev:
Id: 1/3

A

A

B

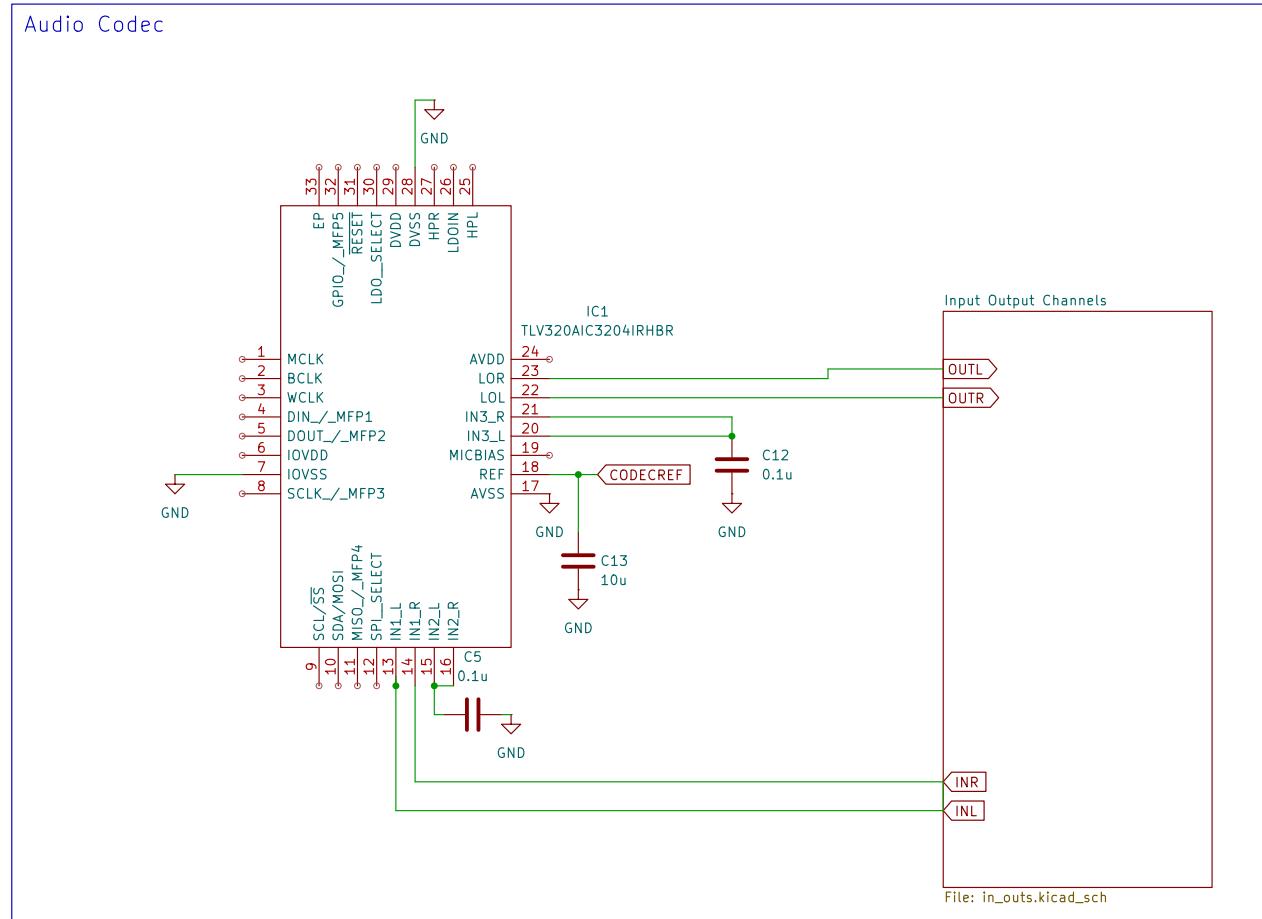
B

C

C

D

D



Sheet: /Audio Codec/
File: audio_codec.kicad_sch

Title:

Size: A4 | Date:
KiCad E.D.A. 8.0.4

Rev:
Id: 2/3

1 2 3 4 5 6

Calculation of input AC-coupling Capacitor

$$Req = Rin \times (1 + 2g) / (1 + g)$$

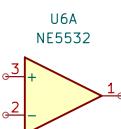
bei $g = 1$ und $Rin = 10k$

$$Req = 30k$$

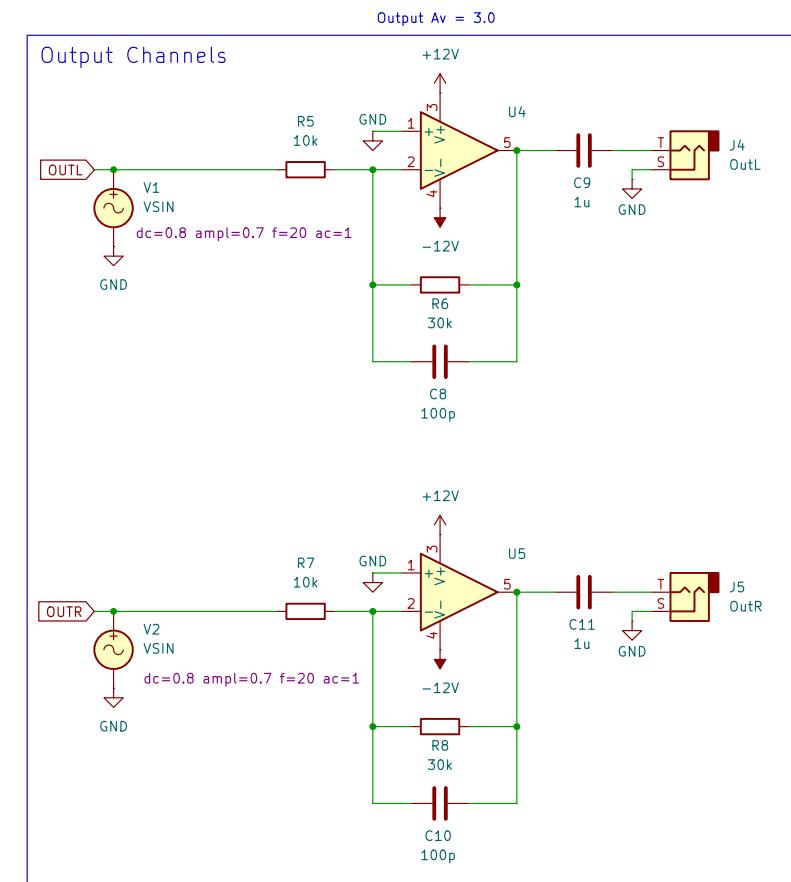
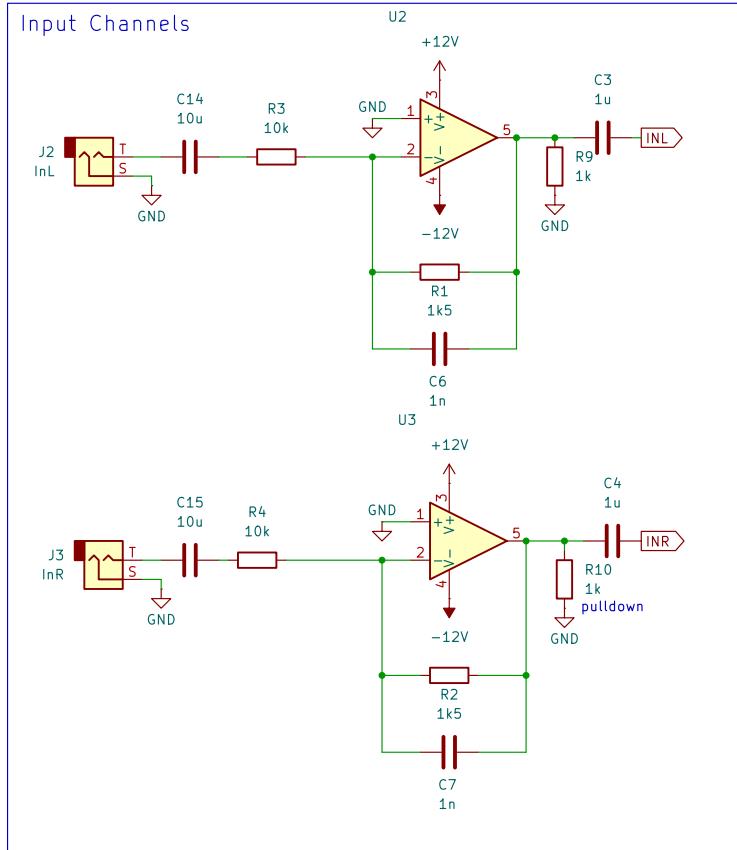
$$Fc = 1 / (2 \times \pi \times Req \times Cc)$$

bei $Cc = 1u$

$$Fc = 5Hz$$



Input Av = 0.15



Sheet: /Audio Codec/Input Output Channels/
File: in_outs.kicad_sch

Title:

Size: A4 | Date:

KiCad E.D.A. 8.0.4

Rev:

Id: 3/3

1 2 3 4 5 6