Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Oleksandr Müller

Entwicklung eines Treibers für eine neuentwickelte Audio-Aufsteckplatine für den Raspberry Pi

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Oleksandr Müller

Entwicklung eines Treibers für eine neuentwickelte Audio-Aufsteckplatine für den Raspberry Pi

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Robert Heß
Zweitgutachter : Prof. Dr. Thomas Lehmann

Abgegeben am 9. Januar 2018

Oleksandr Müller

Thema der Bachelorthesis

Entwicklung eines Treibers für eine neuentwickelte Audio-Aufsteckplatine für den Raspberry Pi

Stichworte

Linux, Treiber, Gerätetreiber, Audio Platine, ALSA, ASoC, Device Tree, Device Tree Overlay, Raspberry Pi, Raspbian

Kurzzusammenfassung

Diese Arbeit umfasst eine Gerätetreiberentwicklung für eine Audioplatine für einen Raspberry Pi im Linux-Betriebssystem. Anschließend wird das Gerät mit dem entwickelten Gerätetreiber Hardwaretests unterzogen.

Oleksandr Müller

Title of the paper

Development of a driver for newly developed audio soundcard for the Raspberry Pi

Keywords

linux, drivers, device drivers, soundcard, ALSA, ASoC, device tree, device tree overlay, Raspberry Pi, Raspbian

Abstract

In this report the development of a device driver for a soundcard is presented. Afterwards the device with the developed device driver is subjected to a hardware test.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADU	Analog-Digital-Umsetzer
ALSA	Advanced-Linux-Audio-Architecture
API	Applications-Programming-Interface
ARM	Acorn-RISC-Machines
ASOC	Advanced-Linux-Audio-Architecture-System-on-Chip
DAI	Digital-Audio-Interface
DAPM	Dynamic-Audio-Power-Management
DAU	Digital-Analog-Umsetzer
DMA	Direct-Memory-Access
DT	Device-Tree
DTB	Device-Tree-Blob
DTBO	Device-Tree-Blob-Overlay
DTC	Device-Tree-Compiler
DTO	Device-Tree-Overlay
DTS	Device-Tree-Source
DTSI	Device-Tree-Source-Include
EEPROM	Electrically-Erasable-Programmable-Read-Only Memory
Ftrace	Function-Tracer
GPIO	General-Purpose-Input-Output
HAT	Hardware-Attached-on-Top
I2C	Inter-Integrated-Circuit
I2S	Inter-Integrated-Sound

ISA Industrial-Standard-Architecture

LKM Loadable-Kernel-Module

MIDI Musical-Instrument-Digital-Interface

mmap Memmory-Mapping

MUX Multiplexer

OSI Open-Systems-Interconnection

OSS Open-Sound-System

PCI Peripheral-Component-Interconnect

PCM Pulse-Code-Modulation

PLL Phase-Locked-Loop

PM Power-Management

regmap Register-Map

RPI Raspberry Pi

SMD Surface-Mounted-Device

SNR Signal-to-Noise-Ratio

SoC System-on-Chip

UML Unified-Modeling-Language

1. Einleitung

Seit Anbeginn des Fortschritts im Bereich der Computersysteme werden diverse Geräte, zum Beispiel Massenspeichergeräte, Drucker, usw., auf dem Markt verkauft. Die stetige Entwicklung von neuer Hardware fordert eine entsprechend ausgearbeitete Software um dem Nutzer die gewonnenen Eigenschaften zur Verfügung zu stellen. Die Funktion eines Treibers ist die Übersetzung einheitlicher Betriebssystemzugriffe in Anweisungen auf der untersten Bit Ebene, woraus eine Unterstützung für sämtliche Anwendungen resultiert. Ziel dieser Arbeit ist die Entwicklung eines Gerätetreibers für eine Soundkarte in ein Linux-Betriebssystem.

Die Entwicklung von Geräten zur Verbesserung des Klanges beginnt mit den ersten Soundkarten ab dem Jahr 1988. Bis dahin haben interne Lautsprecher existiert, welche nicht mit einem Lautstärkeregler ausgestattet sind und nur die Möglichkeit besitzen einen Ton zur Zeit wiederzugeben. Ausschlaggebend in der Weiterentwicklung ist die Musikproduktion sowie die Videospielindustrie. Die Musiker benötigen eine Schnittstelle zum Computer für die Aufnahme von Instrumenten und die Videospiele zum Wiedergeben von Hintergrundmusik.

Heutzutage wird für das Erreichen eines optimalen Klangerlebnisses sowohl im professionellen als auch im privaten Bereich häufig zusätzliche Hardware benötigt. Oftmals bietet die Audiolösung auf den Mainboards eines Desktop-Computers oder Notebooks nicht die geforderten Eigenschaften hinsichtlich der Abstastfrequenz, Bittiefe und Anzahl der anzusteuernden Kanäle. Dem Nutzer steht ein Markt mit einer Vielzahl an möglichen Lösungen zur Verfügung. Die Unterschiede der einzelnen Geräte liegen in der eingesetzten Schnittstelle zur Verbindung, der Anzahl von Ein- und Ausgängen auf der Platine, dem Format der Buchsen beziehungsweise Stecker Form, der verbauten Komponenten für die Signalverarbeitung sowie dem Preis der jeweiligen Lösung.

Aus Kostengründen kann ein schlechter Frequenzgang hingenommen werden und anschließend über eine Software Lösung nach gegeben Spezifikationen aufgearbeitet werden. Eine Option bietet der Einplatinencomputer Raspberry Pi, welcher im Gegensatz zu einem konventionellen Computer durch seine kompakte Größe besticht und in vielen Projekten eingesetzt werden kann. Im Vergleich zu einen Mikrocontroller existieren durch die stark wachsenden Nutzerschaft des Raspberry Pis eine große Anzahl an Benutzerbibliotheken, was wiederum zu einer komfortableren als auch schnelleren Entwicklung führt. Die Audio-Schnittstellen des Raspberry Pis belaufen sich auf einen Audio Out sowie einen HDMI Ausgang womit

die Auswahl der zu verbindenden Geräte gering ausfällt. Die Raspberry Foundation ermutigt Entwickler eigene Aufsteckplatinen gemäß elektrischer und mechanischer Vorgaben zu entwickeln . Die Aufsteckplatine innerhalb dieser Arbeit wurde von Sebastian Albers in einer vorangegangen Bachelorarbeit geplant, entwickelt sowie auf eine einwandfreie Funktion erprobt.

Zielsetzung dieser Arbeit ist die Entwicklung eines Gerätetreibers für die Aufsteckplatine von Sebastian Albers im Linux-Betriebssystem Raspbian. Der Gerätetreiber übernimmt die Weitergabe der Informationen an das Betriebssystem sowie für die Funktionalität der einzelnen Eigenschaften. Basierend auf diesem Treiber soll es möglich sein, Software-Lösungen zu entwickeln, welche zur Signalverarbeitung verwendet werden. Wird eine Abfrage über die zur Verfügung stehenden Wiedergabe- und Aufnahmegeräte ausgeführt, muss die Soundkarte angezeigt werden. Erwartungsgemäß muss die Aufnahme- sowie Wiedergabefunktion, über die dafür vorgesehenen Klinkenbuchsen, einwandfrei möglich sein. Darüber hinaus sollte eine Möglichkeit bestehen relevante Parameter einzustellen. Außerdem soll eine Lösung der automatischen Treibererkennung ausgearbeitet werden um eine benutzerfreundliche Inbetriebnahme zu gewährleisten.

2. Grundlagen

Dieses Kapitel zeigt relevante Themen zur Entwicklung des Gerätetreibers für die HiFi-Hardware-Attached-on-Top (HAT) Platine auf.

2.1. Raspberry Pi

Der Raspberry Pi (RPI), ist ein kostengünstiger Kleinstcomputer, welcher sich mit verschieden Peripheriegeräten wie einem Monitor als auch Maus und Tastatur verbinden sowie nutzen lässt. Für Bildugszwecke entwickelte die Raspberry Foundation den RPI, um allen Menschen eine Möglichkeit zu geben Computer zu erforschen, als auch um Programmiersprachen zu erlernen. Der RPI steht im Funktionsumfang einem Desktop-Computer in nichts nach. Mit ihm ist es möglich, im Internet Seiten zu besuchen, hochauflösende Videos anzusehen, Texte zu bearbeiten und Spiele zu spielen.

Im Jahr 2012 wird erstmals der RPI in den zwei Varianten A und B zum Verkauf angeboten. Es folgen über die Jahre noch weitere zehn Modelle. Sie unterscheiden sich in ihren Abmessungen, der Prozessorleistung sowie der Ein- und Ausgänge. Außerdem werden zusätzlich zum Kreditkarten-Formfaktor drei Compute-Modelle als SO-DIMM Arbeitsspeicherriegel entworfen.

Die aktuelle Version, der RPI 3 B, besitzt einen ARM Cortex-A53 Vierkernprozessor mit einer Taktrate von 1,2 GHz sowie einen Arbeisspeicher von 1 Giga Byte. Darüber hinaus wird der RPI zum ersten Mal um integrierten WLAN-Chip sowie Bluetooth-Low-Energy erweitert. Da der Kleinstcomputer mit einem Mikro USB Anschluss betrieben wird, wird bei der Verwendung von vielen Peripheriegeräten ein leistungsstarkes Netzteil mit 5 V sowie 2,5 A empfohlen. Der Bootvorgang wird mittels einer Mikro SD Karte initiiert. Es stehen diverse Linux Distributionen wie Raspbian, Arch und Fedora, eine spezielle Windows 10 Internet of Things Version, zur Verfügung. Darüber hinaus besteht die Möglichkeit den RPI ohne ein Betriebssystem direkt in Assembler oder C zu programmieren. Zu den weiteren Vorteilen zählen die verfügbaren Schnittstellen [vgl. 10, S. 25-26].

"Hier sind unter anderem folgende Schnittstellen verfügbar: 2x I2C, 1x I2S und mehrere frei programmierbare General-Purpose Input/Output (GPIO) Pins. Außerdem liegen hier die Betriebsspannungen 5 V und 3,3 V an. Die Schnittstellen und GPIOs der Stiftleiste arbeiten mit

3,3 V und sind nicht 5 V-tolerant. Über weitere Schnittstellen kann eine Kamera (CSI) und ein Display (DSI) angeschlossen werden. Außerdem existieren übliche Computer-Schnittstellen wie ein vierfach USB-Hub, ein 100 MBit/s Ethernet-Anschluss sowie ein HDMI- und ein einfacher analoger Audioausgang."[10, S.26]

Wenn eine Schnittstelle nicht abgedeckt ist, gibt es eine Möglichkeit, diese über eine HAT nachzurüsten. Im Jahre 2014 ist eine Spezifikation für Erweiterungsplatinen veröffentlicht worden. Diese besagt, dass eine Erweiterungsplatine erst als HAT-kompatibel bezeichnet werden darf, sofern alle diese Voraussetzungen erfüllt. Die Voraussetzungen setzen sich aus mechanischen und elektrischen Gegebenheiten zusammen. Die Erweiterungsplatine muss eine Größe von 65x55mm, Löcher zum Befestigen, sowie Schlitze für Flachbandkabel aufweisen. Da die Erweiterungsplatine nur Signale von der 2x20 Stiftleiste nutzen kann, benötigt sie einen Electrically-Erasable-Programmable-Read-Only Memory (EEPROM) Baustein. Ist dieser mit entsprechendem Inhalt vorbereitet, kann mit Hilfe der Inter-Integrated-Circuit (I2C)-Identifizierungssignale eine Erkennung des aufgesteckten Moduls stattfinden. Der Inhalt auf dem EEPROM setzt sich aus Herstellerinformationen, Gerätetyp und der GPIO-Konfiguration im Device Tree Format zusammen [vgl. 5, 119-120].

2.2. HiFi-Hardware-Attached-on-Top

Bei dem HiFi-HAT aus der Bachelorarbeit „Entwicklung und Realisierung einer Raspberry Pi-Aufsteckplatine zur Bereitstellung von hochwertigen analogen Audioschnittstellen“ von Sebastian Albers handelt es sich um eine Erweiterung der Audioanschlüsse des RPIs. In Abbildung 2.1 ist die Aufsteck-Platine mit den großen Folienkondensatoren auf einem RPI aufgesteckt sowie die Variante mit Surface-Mounted-Device (SMD) Kondensatoren rechts dargestellt.

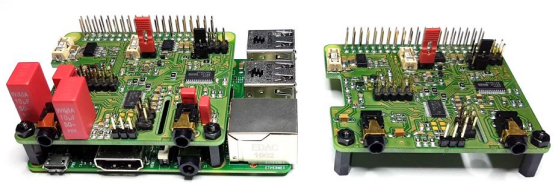

Abbildung 2.1.: HiFi-Hardware Attached on Top [vgl. 10, S. 65 Abb. 5.12]

Bezogen auf dem Signal-to-Noise-Ratio (SNR) ergänzt das HiFi-HAT Erweiterungsmodul den RPI um hochwertigere Audioanschlüsse. Darüber hinaus wird die Anzahl der Ein- und Ausgänge erweitert. Der RPI eignet sich nur eingeschränkt zur Audioverarbeitung was an folgendem liegt:

"Der Raspberry Pi bietet selbst eine analoge Audioschnittstelle zur Tonausgabe, einen Eingang stellt er jedoch nicht zur Verfügung. Die Ausgabe basiert auf einer Pulsweitenmodulation (PWM). Dieses Signal wird über ein RC-Tiefpass geführt und der so erzeugte Mittelwert ausgegeben. Der vorhandene Ausgang ist insbesondere bei älteren Versionen schlecht, da der zur Ausgabe verwendete GPIO Pin direkt von der stark verrauschten allgemeinen Betriebsspannung versorgt wird. In neueren Versionen (Modelle A+, B+ 2B und 3B) ist ein separat versorgter Puffer zwischengeschaltet und die Ausgabe damit etwas verbessert. Außerdem existieren Änderungen an der Software, sodass unter gestiegener Rechenlast per Überabtastung Werte knapp über 90 dB erreicht werden."[10, S. 33-34]

2.2.1. Schnittstellen

Zu den Schnittstellen des HiFi-HAT gehört der Kontakt mit dem RPI. Außerdem besitzt das Erweiterungsmodul zwei Stereo-Klinkenbuchsen für den unsymmetrischen Audioeingang beziehungsweise Audioausgang. Weiterhin existieren Stiftleisten für vier weitere analoge Audioeingänge, einen digitalen Pulsdichtemodulation Eingang für maximal zwei Mikrofone und einen Ausgang der Inter-Integrated-Sound (I2S) Daten für die Fehlersuche, Inbetriebnahme und eventuell Anschluss weiterer Komponenten. Der Schaltplan zur Platine kann im Anhang in Abbildung A.1, A.2 und A.3 entnommen werden.

2.2.2. Electrically-Erasable-Programmable-Read-Only-Memory Baustein

Der EEPROM zur HAT-Identifikation wird mit dem I2C-0 Bus verbunden, welcher ausschließlich für die Identifizierung von Aufsteckplatinen vorgesehen ist. Es wird ein 32kBit EEPROM Baustein von On Semiconductor verwendet mit der Adresse 0x50.

2.2.3. Analog-Digital- und Digital-Analog-Umsetzer

Die verbauten Umsetzer *pcm1863* sowie *pcm5142* eignen sich besonders gut, da es sich um höher integrierte Bauteile handelt. Sie benötigen jeweils pro Kanal einen analogen Filter, werden über 3,3 Volt versorgt und lassen sich über den I2C-Bus konfigurieren. Für die Signalverarbeitung wird der I2S-Bus verwendet. Zusätzlich werden die Komponenten über zwei eigene Taktbausteine mit einem jitterfreien Takt, welche einen Vielfaches der 44,1kHz sowie 48kHz Grundfrequenz beträgt, versorgt. Die Umschaltung zwischen beider Taktgenerator geschieht über ein General-Purpose-Input-Output (GPIO) Pin des Digital-Analog-Umsetzer (DAU). Infolgedessen muss am Analog-Digital-Umsetzer (ADU) der entsprechende Taktteiler-Wert eingestellt werden. Ist der ADU im Master-Modus, generiert er dem im

Slave-Modus befindenden DAU den Worttakt sowie den Links-Rechts-Takt. Die Adresse des ADUs ist 0x4a sowie des DAUs 0x4c.

2.3. Linux-Gerätetreiber

Gerätetreiber stellen eine Erweiterung für den Kernel dar und werden für die Unterstützung von Hardware benötigt. Im ersten Schritt wird dem Betriebssystem das neue Gerät vorgestellt sowie ordnungsgemäß eingebunden. Es folgt die Nutzung in Applikationen über Standardanweisungen. Abschließend muss dem Betriebssystem mitgeteilt werden, dass die Hardware nicht mehr verfügbar ist sowie, dass reservierter Speicher freigegeben werden kann. Im RPI Raspbian Betriebssystem befinden sich die Gerätetreiber in dem Pfad */lib/-modules/'uname -r'/kernel/drivers*. Abbildung A.4 zeigt den Aufbau dieses Ordners, sowie die einzelnen Treibersubsysteme, wie beispielsweise USB, PCI, I2C, usw. Weiterhin ist in der Abbildung A.4 kein *sound* Ordner dargestellt. Dieser ist eine Ebene früher getrennt und beinhaltet unter anderem die Audio-Treiber.

Im Folgenden werden einzelne Kapitel der Gerätetreiber Entwicklung näher betrachtet, da sie von hoher Relevanz für die Implementierung eines Audio-Treibers sind. Das grundlegende Vorgehen einer Treiberentwicklung unterscheidet sich wenig, dennoch existieren speziell angepasste Bibliotheken mit Funktionen für einzelne Subsysteme. Es findet eine Unterteilung von Geräten in verschiedene Klassen statt:

- zeichenorientierte Geräte
- blockorientierte Geräte
- socketorientierte Geräte

2.3.1. Loadable-Kernel-Module

Ein Loadable-Kernel-Module (LKM) ist eine Möglichkeit während des Betriebes Code dem Linux-Kernel hinzuzufügen oder zu entfernen. Dies hat den Vorteil, dass der Kernel mit dem Gerät kommunizieren kann ohne zu wissen wie es funktioniert. Die Alternative ist den Code für jeden Treiber im Linux-Kernel zu implementieren. Ohne diese modulare Eigenschaft ist die Größe des Kernels riesig, da er jeden veröffentlichten Treiber unterstützen muss. Darüber hinaus wird der komplette Kernel neu gebaut, wenn ein neuer Treiber hinzugefügt oder ein bereits bestehender Treiber erneuert wird. Die Nutzung von LKMs birgt den Nachteil, dass für jedes Gerät ein Treiber gepflegt werden muss [vgl. 3, S. 5].

2.3.2. Unterscheidung zwischen User-Space und Kernel-Space

Es findet eine grundlegende Unterscheidung statt, da LKMs im *kernel space* und Programme im *user space* ausgeführt werden. Beide besitzen einen getrennten Speicherbereich. Somit wird gewährleistet, dass die laufenden Programme immer die Geräte ungeachtet der Plattform-Eigenschaften gleich betrachten.

Die Kernel-Dienste werden für das *user space* über Systemaufrufe erreichbar gemacht. Zusätzlich verhindert der Kernel Konflikte, wenn mehrere Programme auf beschränkte Ressourcen zugreifen möchten. Dies ist unter anderem durch verschiedene Zugriffsrechte zum Beispiel über reguläre oder *superuser* Rechte realisiert.

2.3.3. Lebenszyklus eines Treibers

Abbildung 2.2 zeigt den Lebenszyklus eines Treibers. Zu Beginn muss der Treiber eine Initialisierung vornehmen. In dieser wird Speicher reserviert sowie der Treiber mit dem jeweiligen Gerät beim Betriebssystem registriert und ein Zeiger auf die File Operations, welche auch als Treibereinspringpunkte bekannt, übergeben. Weiterhin wird überprüft, ob eine Instanz vom Treiber bereits geöffnet ist. Zusätzlich kann die Hardware in diesem Schritt mit Anfangswerten vorbereitet werden. Wenn die Initialisierung erfolgreich durchlaufen ist, wird für das Gerät ein Eintrag im proc Verzeichnis angelegt.

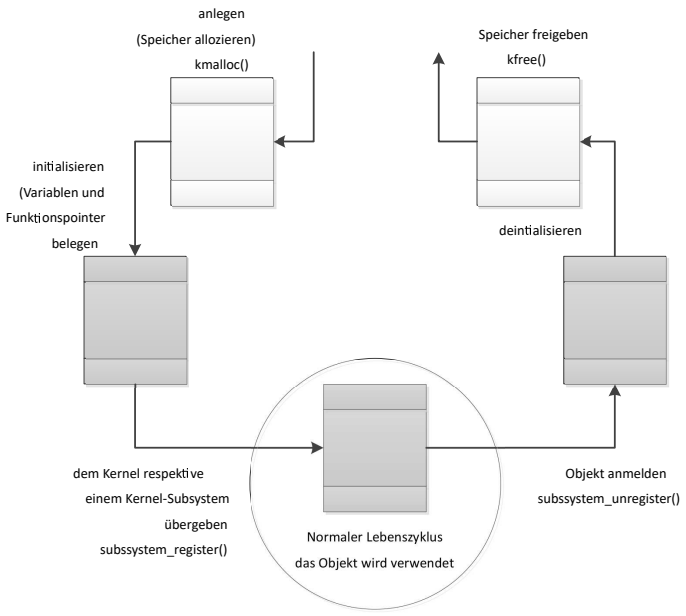

Abbildung 2.2.: Lebenszyklus eines Treibers [vgl. 1, S.61 Abb. 3-5]

Zum Schluss muss eine Deintialisierung vorgenommen werden, welche das Gerät vom Treiber entregistriert und allen reservierten Speicher wieder freigibt.

2.3.4. Aus Sicht einer Applikation

Für die Nutzung von Geräten mit Standardanweisungen muss die Hardware wie eine Datei aussehen, was über eine Gerätedatei gelöst wird wie in Abbildung 2.3.

Wenn eine Applikation Daten auf ein Gerät schreiben beziehungsweise von einem Gerät lesen möchte, muss es beim Kernel über die *open* Anweisung angemeldet werden. Über einen zweiten Parameter wird die Art des Zugriffs festgelegt. Es folgt eine Überprüfung, ob entsprechende Rechte für die Applikation vergeben sind und, ob ein Zugriff in diesem Moment möglich ist. Wenn eine von den beiden Voraussetzungen nicht erfüllt ist, wird der Zugriff verweigert. Analog zur Anmeldung über die *open* Anweisung muss ein *close*, zum Abmelden und Ressourcen freigeben, existieren.

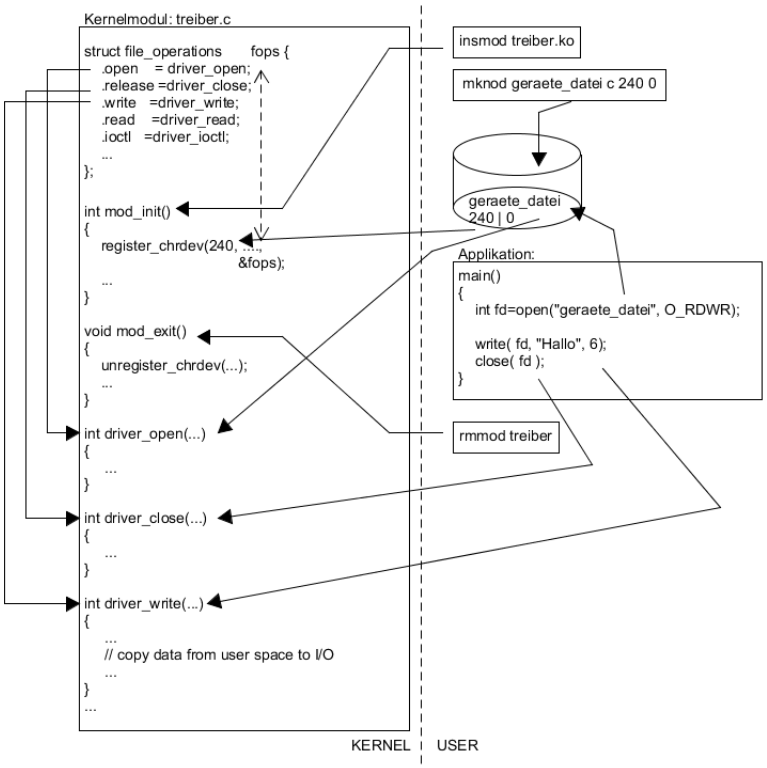

Abbildung 2.3.: Einbindung des Gerätetreibers in das System [vgl. 1, S.95 Abb. 5-2]

Für die weitere Nutzung sind die Anweisungen *read* und *write* von besonderer Bedeutung. Die *read* Funktion bekommt zusätzlich zum Dateideskriptor die Adresse sowie Größe des Speicherbereichs übergeben. In diesem Bereich werden die auszuwertenden Daten hineingeschrieben. Der Rückgabewert liefert die Anzahl der gelesenen Bytes. In Folge vom Dateiende wird eine Null zurückgegeben. Analog zur *read* Funktion wird beim *write* die Adresse sowie die Größe des zu schreibenden Speicherbereichs übergeben. Sie gibt die Anzahl der geschriebenen Bytes zurück.

Zweifellos lassen sich hiermit noch nicht alle Funktionalitäten abbilden und es gibt weitere Anweisungen zum Beispiel *ioctl*, *lseek*, *select* und *fcntl* [vgl. 1, S.71-78].

2.3.5. Zeichengeräte

Als Zeichengeräte werden Geräte klassifiziert, die wie eine Datei behandelt werden können. Unter diese Kategorie fallen Geräte, welche einen Strom an Daten erzeugen beziehungsweise einlesen unter anderem. Im Vergleich zu einer Datei besteht nur die Möglichkeit die aktuellen erzeugten Werte einzusehen. Hiermit entfällt die Option vorangegangene Werte zu betrachten, wenn diese nicht in einer Datei gespeichert werden.

```
pi@raspberrypi:~ $ ls -l /dev/
brw-rw---- 1 root disk      1,   7 Dec 16 15:47 ram7
brw-rw---- 1 root disk      1,   8 Dec 16 15:47 ram8
brw-rw---- 1 root disk      1,   9 Dec 16 15:47 ram9
crw-rw-rw- 1 root root      1,   8 Dec 16 15:47 random
drwxr-xr-x 2 root root          60 Jan  1  1970 raw
crw-rw-r-- 1 root root     10,  58 Dec 16 15:47 rfkill
lrwxrwxrwx 1 root root           7 Dec 16 15:47 serial1 -> ttyAMA0
drwxrwxrwt 2 root root          60 Dec 16 15:47 shm
drwxr-xr-x 3 root root         160 Dec 16 15:47 snd
lrwxrwxrwx 1 root root          15 Jan  1  1970 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root          15 Jan  1  1970 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root          15 Jan  1  1970 stdout -> /proc/self/fd/1
crw-rw-rw- 1 root tty       5,   0 Dec 16 15:47 tty
crw--w---- 1 root tty       4,   0 Dec 16 15:47 tty0
crw------- 1 pi   tty       4,   1 Dec 16 15:47 tty1
```

Abbildung 2.4.: Ausgabe der Geräte auf dem Raspberry Pi

Über den Kommandozeilen Befehl *ls -l* wird die Auflistung von Dateien im aktuellen Ordner in Langform dargestellt. Die erste Spalte beschreibt die Geräteklasse ein c steht für *character* und ein b für ein *block* Gerät. In Abbildung 2.4 wird beispielhaft das **TTY!** (**TTY!**) Gerät als Zeichengerät dargestellt. Gerätetreibern werden *major* und *minor* Zahlen zugeordnet. Die *major* Zahl wird für die Identifizierung des korrekten Treibers vom Kernel benötigt wenn auf das Gerät zugegriffen wird. Die *minor* Zahl steht für das jeweilige Gerät welches vom Treiber verwaltet wird. In Abbildung 2.5 wird der Zugriff auf ein Zeichengerät dargestellt.

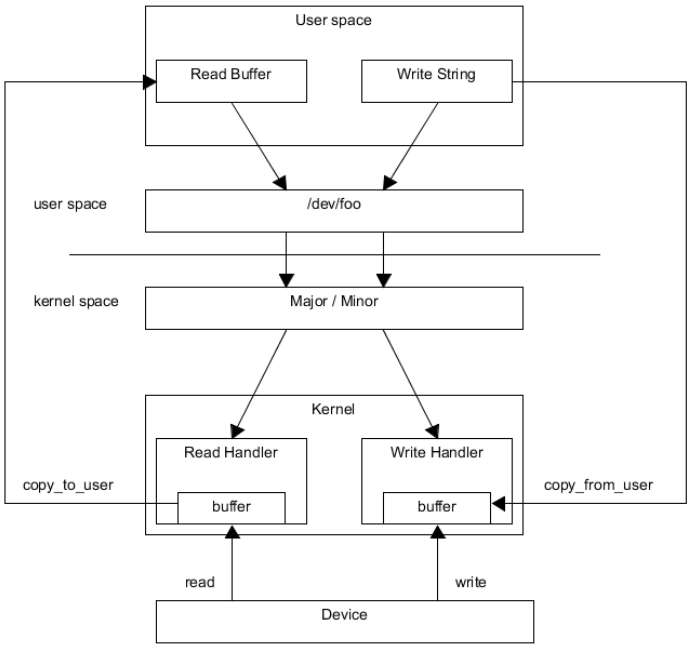

Abbildung 2.5.: Zugriff auf Zeichenorientierte Geräte [vgl. 22, S. 209]

Im Anschluss muss eine *file operations* Struktur deklariert. Sie verbindet Treiberfunktionen mit den Gerätenummern. Das Gerät wird als Objekt betrachtet und die Verarbeitungsfunktionen unter anderem *open*, *read*, *write*, *close*, usw. als Methoden. Jedes Feld in dieser Struktur zeigt auf eine bereits implementierte Funktion. Wenn dies nicht der Fall ist, wird sie nicht unterstützt [vgl. 3, S.42-52].

2.4. Device-Tree

Ein Device-Tree (DT) wird zur Beschreibung von Hardware benutzt. Beim Bootvorgang wird der DT in den Arbeisspeicher geladen sowie die Adresse an den Kernel übergeben. Hierfür wird ein spezieller Bootloader benötigt, welcher seit Kernel-Version 3.7 zum Einsatz kommt [vgl. 18, S. 1].

2.4.1. Motivation

Der Linux-Kernel beinhaltet Informationen über sämtliche Hardware auf allen unterstützten Plattformen. Dazu gehören unter anderem Speicheradressen, Interrupts, Peripherien auf ei-

nem Chip, fest ein programmiert. Beim Bootvorgang wird ein Parameter ausgelesen, welcher an den Kernel übergeben als auch der Typ der Plattform bestimmt. Im Folgenden wird die Hardware auf Basis dieses Wertes initialisiert.

Der Ansatz wird nur von wenigen Plattformen, wie bei der standardisierten X86 und X86_64 PC-Architektur unterstützt. Infolge des Lizenzmodells von Acorn-RISC-Machines (ARM) Baupläne anstatt Chips zu verkaufen entsteht eine schwer überschaubare Anzahl an neuen Plattformen. Die Größe des Kernelcodes steigt, da jede ARM Plattform eigene Treiber, Module sowie eine Ansteuerungslogik benötigt. Die Vielseitigkeit von Linux schwindet, da ein aus dem Quellcode kompilierter Kernel nur noch auf spezifischen Plattformen lauffähig ist. Im Jahr 2010 fordert Linus Torvalds eine Lösung dieses Problems und setzt die Entwickler damit unter Druck solange keine neue Hardware aufzunehmen. Es wird die DT Spezifikation erarbeitet, welche auf dem Prinzip der Open Firmware für den PowerPC basiert sowie Code stellenweise wiederverwendet. Zum Zeitpunkt des Bootvorgangs werden Informationen über die Hardware übergeben, womit kein festes einkompilieren oder lange Bootparameter benötigt werden [S. 1 18, vgl.].

2.4.2. Device-Tree-Source

Die Device-Tree-Source (DTS) Datei beschreibt strukturiert sämtliche Hardware im Klartextformat. Knoten werden mit Schlüsselwerten, welche besondere Eigenschaften repräsentieren, sowie darunter liegende Knoten erzeugt. Der Inhalt eines Knotens wird in geschweiften Klammern eingeschlossen. Wertzuweisungen werden mit einem Gleichheitszeichen ausgeführt. Falls dieses nicht vorhanden ist, genügt bereits der Name dieser Eigenschaft.

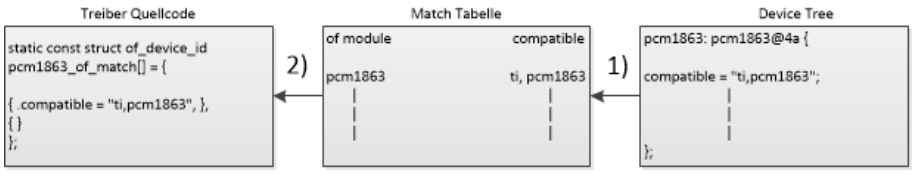

Abbildung 2.6.: Visualisierung des Matching über die *compatible* Eigenschaft [vgl. 1, S. 143 Abb. 5-12]

Um einen Zugriff aus dem *user space* auf die Hardware zu erzeugen, wird die *compatible* Eigenschaft benötigt. Eine automatische Verbindung vom Knoten zum Gerätetreiber erfolgt über eine Match Tabelle. Der Vorgang wird in Abbildung 2.6 verdeutlicht. Der Treiber muss einen identischen String wie unter dem *compatible* Attribut aufweisen [vgl. 1, S. 139-144].

2.4.3. Device-Tree-Overlay

Beim Device-Tree-Overlay (DTO) wird modular ein Teil zu einem DT definiert. Dies ist besonders bei einem RPI mit einem HAT Aufsteckmodul wichtig. So wird für die Plattformen RPI 2B, 3B und Zero mit verschiedenen System-on-Chip (SoC)s jeweils ein DT benötigt. Dieses Problem wird mit Device-Tree-Source-Include (DTSI) Dateien umgangen, welche grobe Eigenschaften abdecken. Der DT Code definiert die Peripherie und DTOs erweitern die Funktionalität einzelner Knoten. Über die Eigenschaft *target* wird der Knoten ausgewählt sowie dem Schlüsselwort *__overlay__* aufgezeigt, dass eine Erweiterung folgt. Im Kern Knoten eines DTO wird die Verbindung zum Basis DT des jeweiligen SoC hergestellt [vgl. 19].

2.4.4. Device-Tree-Compiler

Der Device-Tree-Compiler (DTC) kompiliert den DTS Klartext in eine binäre Device-Tree-Blob (DTB) Datei sowie umgekehrt. Wird ein bereits vorhandener DTB in ein Klartextformat rekompiliert, unterscheidet sich die DTC, da bereits die Platzhalter mit Informationen aus den Bibliotheken ersetzt sind. Der Code des Compilers ist in den Kernelquellen unter scripts/dts/ zu finden.

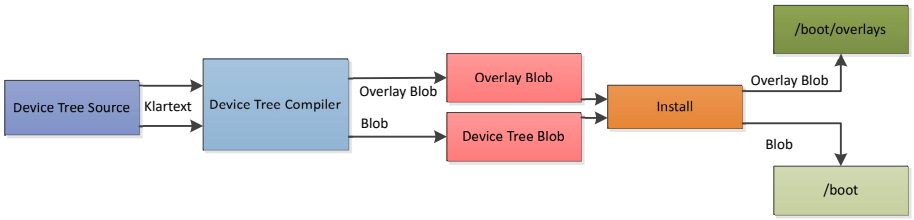

Abbildung 2.7.: Der Prozess einer Klartext Device-Tree-Source in eine Binäre-Blob-Datei [vgl. 7, S. 77 Abb. 1]

2.4.5. Installation

Die Abbildung 2.7 zeigt das Kompilieren als auch einen Teil der Installation. Die binäre DTB Datei muss im Ordner */boot/* bzw. */boot/overlays* abgelegt werden. Außerdem wird die */boot/-config.txt* um den Parameter *dtoverlay* angepasst.

Abbildung 2.8 zeigt den Bau des neuen DT nach einem Neustart und unterstützt mit sofortiger Wirkung die beschriebene Hardware.

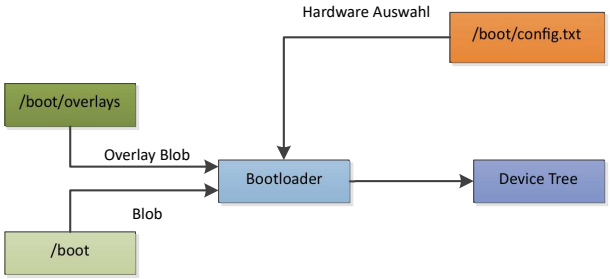

Abbildung 2.8.: Bau des neuen Device-Tree [vgl. 7, S. 77 Abb. 2]

2.5. Advanced-Linux-Sound-Architecture

Das Advanced-Linux-Audio-Architecture (ALSA) wird entworfen, um Einschränkungen von bereits existierenden Audio-Treibern zu bewältigen. Um darüber hinaus High-End Soundkarten zu unterstützen, wird das ALSA unter folgenden Gesichtspunkten entwickelt:

- Unterschiedliche Codes im *kernel-* sowie *userspace*
- Gemeinsame Bibliothek
- Bessere Verwaltung von mehreren Karten und Geräten
- Multi-Thread gesichertes Design
- Kompatibilität zum Vorgänger Open-Sound-System (OSS)

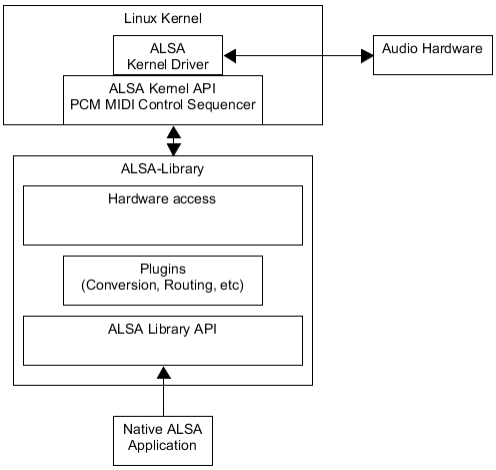

Abbildung 2.9.: Advanced-Linux-Sound-Architecture-System Aufbau [vgl. 8, S. 6 Abb. 1]

In Abbildung 2.9 wird der grundlegende Aufbau vom ALSA System und dem dazugehörigen Datenfluss gezeigt. Das ALSA System besteht aus ALSA Treibern sowie der ALSA Bibliothek. Anders als im OSS sollen die ALSA eigenen Applikationen nur über die ALSA Bibliothek und nicht direkt mit den Kernel-Treibern kommunizieren. Der ALSA Kernel-Treiber bietet den Zugriff auf jede Gerätekomponente wie beispielsweise dem Pulse-Code-Modulation (PCM) und Musical-Instrument-Digital-Interface (MIDI). Es werden so viel wie möglich aller Eigenschaften des Gerätes abgebildet. Währenddessen ergänzt die ALSA Bibliothek fehlende Funktionalitäten von Soundkarten und stellt eine gemeinsame Applications-Programming-Interface (API) für Applikationen zur Verfügung. Somit wird die Kompatibilität gewährleistet, wenn Änderungen an der Kernel API vorgenommen werden, da die Bibliothek die Änderungen kompensieren kann sowie nach außen hin eine konsistente API darstellt [vgl. 8, S. 5-6].

2.5.1. Komponenten

Es werden folgende Komponenten von ALSA unterstützt.

PCM

Die PCM wird Vollduplex betrieben, solange das Gerät dies unterstützt. Die ALSA PCM Komponente besteht aus mehreren Schichten. Jede Soundkarte kann mehrere PCM Geräte besitzen und jedes PCM Gerät hat zwei Datenströme für die Wiedergabe als auch Aufnahme. Darüber hinaus kann jeder PCM Datenstrom mehr als einen Substream besitzen. Das ALSA unterstützt zudem eine große Auswahl an Formaten hierzu gehören lineare Formate von acht bis 32 Bit, 32/64 Bit Gleitkomma und viele andere. Die Multi Kanal Abtastwerte können im geschachtelten sowie nicht geschachtelten Format angeordnet werden. Diest ist von den Anforderungen des Gerätes abhängig. Die digitale Ein- und Ausgabe Schnittstelle wird unterschiedlich und auf das Gerät passend implementiert. Üblicherweise wird ein eigenständiges PCM Gerät für den Ausgang zur Verfügung gestellt. Unter Umständen wird der Ausgabe Typ über einen Einstellungsschalter festgelegt. Diese verschiedenen Implementierungen werden vom ALSA Bibliotheks Konfigurator im *userspace* abgefangen. Es gibt ein virtuelles *snd-dummy* Modul, welches PCM Geräte emuliert. Somit können Programme PCM Daten zum virtuellen PCM Gerät schreiben ohne das es je abgespielt wird [vgl. 8, S. 6].

Controls

Die Einstellungen einer Soundkarte werden an der universellen Einstellungsschnittstelle implementiert. Hierzu gehören Mixer und kartenspezifische Laufzeit-Konfigurationen. Die Ein-

stellungsmerkmale werde in einem einzelnen Array verwaltet. Jedes Element wird anhand eines Stringnamen und einer Index-Nummer identifiziert. Es können verschiedene Datentypen wie Boolean oder Integer verarbeitet werden. Die Darstellung von Mixern hängt vom jeweiligen Gerät ab. Bei vielen Mixer-Elementen werden Integer-Werte für die Lautstärke und Boolean-Werte für einen Mute-Switch verwendet. Wenn es beim Gerät vorgesehen ist, können mehrere Eingangsquellen verwendet und zusammen gemischt werden in diesem Fall stellt ALSA eine Möglichkeit zur Auswahl der Quellen zur Verfügung. Besitzt die Hardware hingegen einen Multiplexer (MUX), welcher einen exklusiven Schalter darstellt, so stellt ALSA eine Liste mit Quellen zur Verfügung. Applikationen haben Zugriff auf das Status Bit und können somit auf das Verhalten der digitalen Ein- und Ausgänge Einfluss nehmen [vgl. 8, S. 6-7].

Zu den weiteren Komponenten gehört das **MIDI**, der **Sequencer**, **Timer** und das **Hardware-Dependent Device**. Diese Themen sind nicht von hoher Relevanz für die Implementierung und werden nicht weiter ausgeführt. Sie können hier nachgelesen werden [vgl. 8, S. 7].

2.5.2. Treiber

Der ALSA Kernel-Treiber besteht aus drei Schichten. Die niedrigste Schicht kommuniziert mit Anweisungen, welche auf die Hardware zugreifen und in Callback-Funktionen implementiert sind. Die mittlere Schicht ist das Kernstück des ALSA Treibers, welche einheitliche Routinen für jede andere Komponente besitzt. Die oberste Schicht ist der Einstiegspunkt für jede Soundkarte, die ein Geräteeinträge erzeugt mit der Callback-Tabelle mit den entsprechenden low-level Funktionen. Durch diese Trennung konzentriert sich die Entwicklung auf den top- sowie low level Zugriff und ignoriert den restlichen Datenfluss.

Jeder Soundkarteneintrag sowie die dazugehörigen Hardware-Komponenten werden in der gleichen Struktur *snd_device_t* zusammengehalten. Die Komponenten werden in einer Liste für jede Karte verwaltet und aus der obersten Schicht einsehbar. Dieser Mechanismus ist für die konsistente Zerstörung beispielsweise bei der Trennung des Gerätes über *hotplug* sowie Kontrolle notwendig. Bei der Entfernung eines Gerätes ändert ALSA die Dateideskriptor Tabelle für das Gerät, um weiteren Zugriff zu verhindern sowie der Aufruf zur Trennung von allen zugewiesenen Komponenten Callbacks. Hiernach wird der Desktruktor aufgerufen, welcher in einer Warteschlange wartet bis die Anweisungen für alle Komponenten beendet sind. Der Aufbau der ALSA Kernel API basiert auf dem Unix-Design. Auf die Geräte wird mit *open*, *close*, *read*, *write* und *ioctl* über Systemaufrufe zugegriffen. Zusätzlich gibt es *readv* und *writev* Befehle bei nicht geschachtelten PCM Abtastwerten für den effizienten Zugriff in Vektorform. Darüber hinaus sind die *poll* Systemaufrufe in allen Komponenten implementiert [vgl. 8, S. 8].

Memory-Mapping

Das Memmory-Mapping (mmap) ist die effizienteste Methode Daten zwischen dem *user-space* und dem *kernel-space* zu übertragen. Sie wird jedoch nicht von jedem Gerät unterstützt und die Größe des Zwischenspeichers wird durch die Hardware eingeschränkt. Somit variiert die Größe bei jedem Chip. Die beste Leistung entfaltet mmap bei Echtzeitanwendungen mit kleinen Zwischenspeichern, welche stark auf Verzögerungen reagieren. Der ALSA Treiber bildet den Direct-Memory-Access (DMA) Zwischenspeicher zum *user-space* ab. Somit kann ein Kopiervorgang gespart werden und die Daten direkt auf den Buffer geschrieben werden. Zusätzlich zu diesem Buffer wird die Einstellung- und die Statusaufnahme in das *user-space* abgebildet. Die aktuelle Abtastwertposition, welchen die Anwendung verarbeitet, ist im ALSA als Anwendungszeiger bekannt und wird in der Einstellungsaufnahme gespeichert. Die Statusaufnahme besteht aus der Abtastwertposition die der DMA verarbeitet, welcher in diesem Fall der Gerätezeiger ist, und den aktuellen Status. Es wird in diesem Fall kein Wechsel zwischen dem *user modes* sowie *kernel mode* benötigt, da die Anwendung direkt auf den Zustand des Treibers schreiben und lesen kann. Wenn die Anwendung nicht den Systemaufruf *poll* benötigen würde für die Synchronisation, werden keine weiteren Systemaufrufe benötigt und der *user mode* wird nicht verlassen. Der PCM Aufnahmezwischenspeicher wird nicht im schreibgeschützten sondern im Lese- sowie Schreibzugriff Modus abgebildet. Diese Bedingung wird benötigt, da Anwendungen Daten auf den Zwischenspeicher schreiben, um die Position festzuhalten. Aus dieser Anforderung müssen die Datenströme der Wiedergabe und der Aufnahme in unterschiedliche Dateien aufgeteilt werden [vgl. 8, S. 8].

PCM-Konfiguration

Die Einstellung der PCM Komponente gehört zum aufwendigsten Teil im ALSA Treiber. Jedes Gerät hat seine eigenen Grenzen und die Anwendung muss die sinnvollste Konfiguration, bestehend aus dem Dateiformat, der Abtastrate, der Zwischenspeichergröße und der Anzahl von erlaubten Perioden, auswählen. Das ALSA besitzt zwei Konfigurationstypen *hardware parameters* und *software parameters*. Im ersten Konfigurationstyp werden grundlegende Eigenschaften für den PCM Datenstrom eingestellt. Letzterer sind optionale Einstellungen zur präzisen Kontrolle. Hier kann beispielsweise festgelegt werden, wie sich das Verhalten bei einem Unter- sowie Überlauf oder ab welchem Timing der DMA beginnt, ändert. Die Schwierigkeit der Konfiguration liegt in den unterschiedlichen Einschränkungen aller Geräte. Diese Bedingungen werden als Schranken im Treiber definiert, welche zu verschiedenen Zeitpunkten eine andere Charakteristik aufweisen. Somit sinkt die Anzahl an möglichen Parameterwerten. Nachdem alle Bedingungen einer Anwendungen vorliegen, wird die beste Konfiguration auf Basis des verbleibenden Parameterraums ausgewählt [vgl. 8, S. 8-9].

Verwaltung der Zwischenspeicher

Das ALSA stellt ein unabhängiges Modul *snd-page-alloc*, welches im frühen Bootvorgang geladen werden kann, zur Verfügung. Der DMA Zwischenspeicher soll kontinuierlich sein, dies ist jedoch durch Einschränkungen der Speicher nicht möglich. Zum Beispiel bei Industrial-Standard-Architecture (ISA) Karten beträgt der Speicher für den Adressbereich weniger als 16 Mega Byte und einige Peripheral-Component-Interconnect (PCI) Chips haben ein 28 oder 30 Bit Limit. Somit wird eine einheitliche Reservierung für Seiten des DMA Zwischenspeichers erschwert. Während des Bootvorganges prüft das Modul nach PCI Geräteeinträgen, wird ein Gerät in der Match Tabelle gefunden wird so versucht das Modul die Zwischenspeicher im voraus zu reservieren. Es verwaltet zudem die Zwischenspeicher, welche von ALSA Soundkarten Modulen reserviert werden. Die Speicher bleiben reserviert selbst dann, wenn das Modul ausgeladen wird um erneut benutzt zu werden beim nächsten Einbinden in das Betriebssystem [vgl. 8, S. 9].

2.5.3. Bibliothek

Die ALSA Bibliothek liegt zwischen den ALSA Treibern sowie den Anwendungen und fungiert als Eingang in das ALSA System. Es stellt eine ALSA Bibliothek API zur Kontrolle von Geräten zur Verfügung.

Plug-Ins

Es wird eine Vielzahl an *plugins* in der ALSA Bibliothek zur Verfügung gestellt. Einige davon sind für die Konvertierung von Daten verantwortlich unter anderem für das Datenformat, die Abtastrate, die Anzahl der Kanäle, geschachtelte oder nicht geschachtelte Formate, usw. Fordert eine Anwendung eine nicht unterstützte Konfiguration vom Gerät, dann lädt ALSA die erforderlichen *plugins* und konvertiert sie zur Laufzeit. Die *plugins* fungieren als *user-space* Treiber, so können beispielsweise Anwendungen über die ALSA Bibliothek API auf andere Systeme zugreifen [vgl. 8, S. 10].

API

Die ALSA Bibliothek API wird entworfen, um die Hardware Abstraktion einfacher zu verdeutlichen. Sie fügt keine neuen Eigenschaften hinzu, sondern fungiert als eine Verpackung für die Systemaufrufe beim direkten Geräte Zugriff. Zur Zeit wird die ALSA Bibliothek API in C zur Verfügung gestellt [vgl. 8, S. 10-11].

2.5.4. Linux-Audio

Das ALSA stellt eine Schnittstelle für alle Soundkarten sowie eine Bibliothek mit einer API zur Ansprache von Gerätetreibern, zur Verfügung.

Die Geräte werden im Folgenden als Soundkarte / Gerät / Subgerät dargestellt. Jede Karte kann aus mehreren Geräten bestehen. Bei Ein- und Ausgangsgeräten zählen hierzu: Analoge Ein- sowie Ausgänge, HDMI, Mic-In, usw. Jedes Gerät hat mindestens ein Subgerät, welches ein ADU beziehungsweise DAU sein muss. Gibt es mehr als ein Subgerät bedeutet dies, dass das Gerät Hardware Mixing betreiben kann. Das Anzeigen von Geräten ist über *aplay -l* für Wiedergabegeräte sowie *arecord -l* für Aufnahmegeräte möglich.

Viele Soundkarten besitzen nur einen ADU/DAU und sind nicht in der Lage Hardware Mixing zu betreiben. In diesem Fall kann eine Soundkarte nur von einer Applikation benutzt werden. Außerdem werden festgelegte Dateiformate und Abtastraten von der Soundkarte unterstützt. ALSA nutzt Plugins für die Konvertierung des Dateiformates, der Abtastrate sowie der Kanäle. Die Plugins werden miteinander verkettet. Eine Standardabspielkette sieht folgendermaßen aus:plug->dmix->hw. In diesem Beispiel ist plug das Konvertierungsplugin, welches den Audio Stream in ein von der Hardware unterstütztes Format sowie Abtastrate konvertiert. Dies wird weitergegeben an dmix, welches Audio Daten mehrerer Applikationen in einen einzelnen Stream formt und an hw weiterreicht. Das Plugin hw sendet die Daten direkt an den Treiber, welcher sie auf dem Gerät in der untersten Bit-Ebene übersetzt.

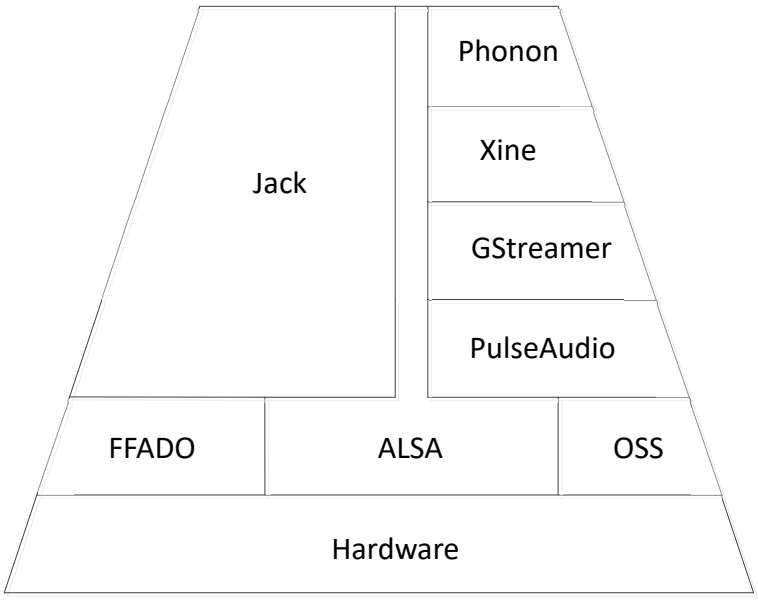

Abbildung 2.10.: Linux-Audio [vgl. 13]

Somit entsteht ein quasi Open-Systems-Interconnection (OSI) Modell wie in Abbildung 2.10 für den Audiobereich . Weitere Informationen gibt es unter: [13]

2.6. ALSA System on Chip

Das Advanced-Linux-Audio-Architecture-System-on-Chip (ASOC) ist ein Kernel Subsystem zur Verbesserung der ALSA Unterstützung auf SoCs und portablen Audio-Codecs. Ziel ist es die Wiederverwendung von Codec-Treibern unabhängig von der Architektur zu steigern sowie eine API zur Integration der Audio-Schnittstelle des SoC.

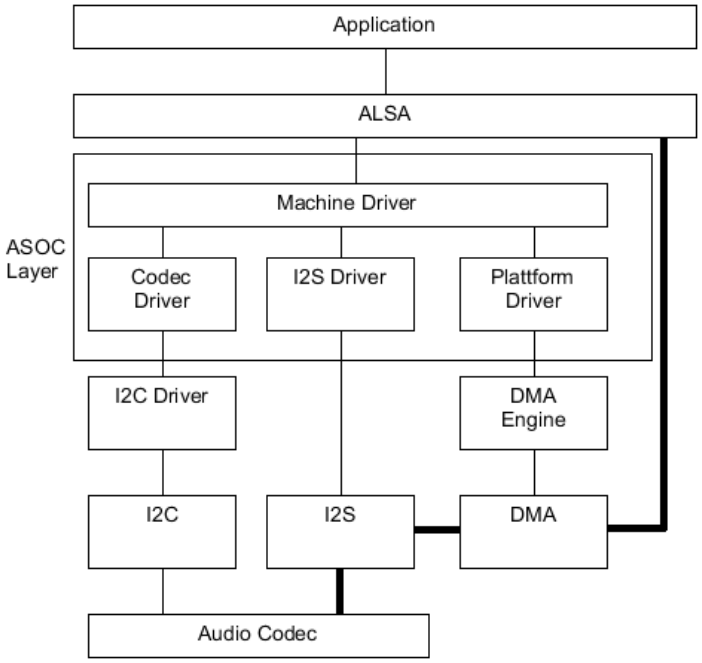

Abbildung 2.11.: Das ASOC-Layer, in dünn dargestellt Steuerungs- und in dick Datenverbindungen [vgl. 9, S. 5 Abb. 4]

Abbildung 2.11 zeigt den Aufbau der ASOC Schicht. Wenn eine Applikation die Wiedergabe von Audio Signalen startet, werden entsprechende Funktionen aus der ALSA Bibliothek aufgerufen. Im darauffolgenden Schritt werden die Initialsierungsfunktionen der Peripherie, welche im Maschinentreiber aufgelistet sind, ausgeführt. Der Codec-Treiber spielt die zentrale Rolle für die Steuerungsanweisungen über I2C, den I2S Treiber zur Einstellung der digitalen Audio-Schnittstelle und den Plattform-Treiber zum Anweisen des DMA *Engine* Treibers. Der DMA übernimmt die Verantwortung für die Übertragung im Voll-Duplex-Betrieb

von Audio-Daten aus dem Hauptspeicher zum I2S-Gerät. Das I2S-Gerät leitet die Daten über die I2S-Schnittstelle zum Audio-Codec weiter. Um die Wiedergabe des Audio-Codecs zu starten, muss der Codec-Treiber entsprechende Kommandos, über das I2C Subsystem, senden. Darüber hinaus wird der Codec-Treiber zum Übertragen von zusätzlichen Codec-Einstellungen beispielsweise der Lautstärke verwendet [vgl. 9, S. 4].

2.6.1. Codec-Klassentreiber

Der Codec-Klassen-Treiber definiert die Funktionalitäten eines Codecs. Dazu gehören die Schnittstellen, Einstellungsmöglichkeiten und analoge sowie digitale Ein- und Ausgänge.

Ein Codec kann Teil eines SoCs sein und muss nicht zwingend extern auf der Platine sein.

2.6.2. Maschinentreiber

Der Maschinentreiber ist speziell an die Platine angepasst. Er dient als Bindeglied zwischen dem Codec- und Plattform-Klassentreiber, da die Informationen wie beide untereinander verbunden sind in diesem definiert werden. Wenn die Hardware für eine Neuentwicklung auf Basis vorhandener Codec-Treiber gewählt wird, muss nur dieser implementiert werden.

2.6.3. Inbetriebnahme

Der Maschinentreiber registriert eine Struktur, welche die Soundkarte repräsentiert. Danach wird eine Verbindung zwischen dem Codec- und Plattform-Klassentreiber hergestellt. Hiernach folgt eine Definition der Ein- sowie Ausgänge über Routen.

3. Anforderungen

Die Anforderungen lassen sich primär in die Kategorien Funktion und Benutzerfreundlichkeit einteilen.

Zur Funktion gehört die Sichtbarkeit der Soundkarte im Betriebssystem als ein Aufnahme- sowie Wiedergabegerät. Sie soll auf Kommandozeilenebene mit ALSA Werkzeugen oder von Applikationen benutzt werden. Beispielhaft werden Audacity zur Aufnahme und der VLC Player zur Wiedergabe genannt. Außerdem kann die Soundkarte unter verschiedenen fest vorgegebenen Parametern betrieben werden. Einstellbar ist die Abtastrate sowie die Bitzahl.

Unter der Kategorie Benutzerfreundlichkeit gehört eine automatische Treibererkennung. Infolge der Einhaltung der HAT-Spezifikation besitzt die Platine einen EEPROM. Wenn in den Speicherbaustein ein auf die Soundkarte angepasstes DTO abgelegt wird, kann das Betriebssystem die Platine selbst erkennen. Zusätzlich wird ein Installationsskript entworfen, welches zur Einbindung des Codec als auch der Vorbereitung zum Einlesen des DT dient. Die Module werden compiliert, in die dafür vorgesehenen Ordner verschoben und die Modulabhängigkeiten werden aktualisiert.

4. Design

Bei einer Treiberentwicklung ist es vorteilhaft auf dem gegenwärtigen Stand aufzubauen. Wie im Grundlagenabschnitt bereits ausgeführt, unterliegt die Methodik sowie das Betriebssystem fortlaufenden Änderungen. Dieser Fortschritt vereinfacht die Entwicklung. Darüber hinaus führen Weiterentwicklungen von Betriebssystemen, wie beispielsweise die Einführung des DT, zu Änderungen in der Treiberentwicklung. Daraus folgt, dass ältere Anleitungen schwieriger umgesetzt werden können und im Zweifelsfall nicht funktionsfähig sind.

Es wird die Entscheidung getroffen im ALSA Subsystem ASOC für den TI-1863 ADU einen Codec-Treiber zu entwickeln. Da dieser bisher nicht unterstützt wird, wird zusätzlich eine Erweiterung des Codec-Treiber benötigt, um über die I2C Schnittstelle Anweisungen zu erhalten. Zusätzlich wird der simple-card Maschinentreiber zusammen mit einem DTO verwendet. Diese bilden die Verbindungen von der Hardware zum Betriebssystem. Für den TI-5142 DAU existiert bereits ein Codec sowie eine Codec-Treibererweiterung. Somit muss ein DTO mit zwei Audio-Verbindungen entwickelt werden, welches beide Funktionalitäten abdeckt.

Die Abbildung 4.1 zeigt den Aufbau des Gerätetreibers. Der Gerätetreiber setzt sich aus dem DTO, dem Codec-Treiber sowie dem Maschinentreiber zusammen.

Beim Hochfahren des Betriebssystems wird der Inhalt des EEPROM ausgelesen. Dieser erweitert den DT des RPIs und aktiviert die I2C sowie I2S Schnittstelle. Darüber hinaus wird der ADU sowie der DAU über die Bus-Adresse mit dem entsprechenden Codec-Treiber verbunden. Dieser bildet sämtliche Funktionen und einstellbare Parameter des Bausteins ab.

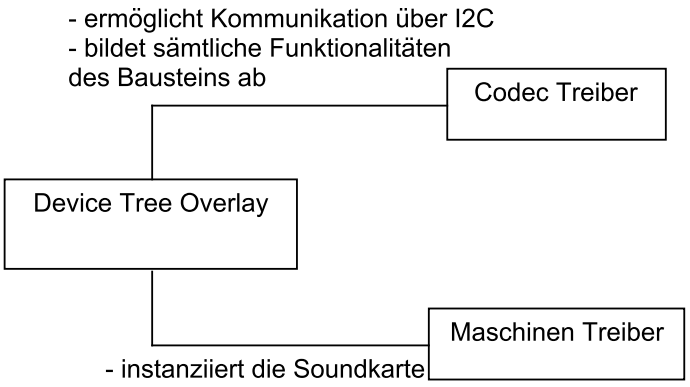

Abbildung 4.1.: Grundlegender Aufbau

Im Folgenden wird die Kommunikation mit dem Maschinentreiber eingeleitet. Dieser erhält Informationen, unter anderem den Namen des Gerätes, das Format für den Stream, über die Soundkarte. Besonders hervorzuheben sind die Definitionen der Audio-Links, welche auf die kürzlich verbundene Hardware mit dem Codec-Treiber zeigen. Mithilfe der aufgeführten Informationen kann der Maschinentreiber die Soundkarte im Betriebssystem instanziieren. Nachfolgend sollte das Gerät zur Wiedergabe, als auch der Aufnahme dem Betriebssystem zur Verfügung stehen.

5. Implementierung

In diesem Kapitel wird die Implementierung des Programmes erläutert. Die Implementierung basiert auf den in Kapitel 4 herausgearbeiteten Entwurf und Modulen.

5.1. Codec-Treiber

Der Codec-Treiber setzt sich aus der *snd_soc_codec_driver* sowie der *snd_soc_dai_driver* Struktur zusammen. Die Struktur des *snd_soc_codec_driver* bildet die Ein- und Ausgänge, die Verdrahtung des Bausteins sowie Einstellungsmöglichkeiten, wie beispielsweise die zusätzliche Nutzung eines Hochpassfilters, ab. Die Struktur des *snd_soc_dai_driver* hingegen definiert die Digital-Audio-Interface (DAI) und PCM Fähigkeiten sowie zur Verfügung stehenden Funktionen.

Im Folgenden werden die implementierten Strukturen mit den zugehörigen Variablen und Parametern näher betrachtet. Zur Visualisierung werden Unified-Modeling-Language (UML) Diagramme verwendet. Wenn möglich, sind die Variablen mit ihrem im Code vorkommenden Wert initialisiert.

5.1.1. snd_soc_codec_driver

Die *snd_soc_codec_driver* Struktur besteht aus einer *snd_soc_component_driver* Struktur, welche die zuvor aufgeführten Komponenten definiert. Sie besteht wiederum aus der *snd_soc_dapm_widget* sowie der *snd_soc_dapm_routes* Struktur. Abbildung 5.1 zeigt den Aufbau der Struktur *snd_soc_codec_driver*.

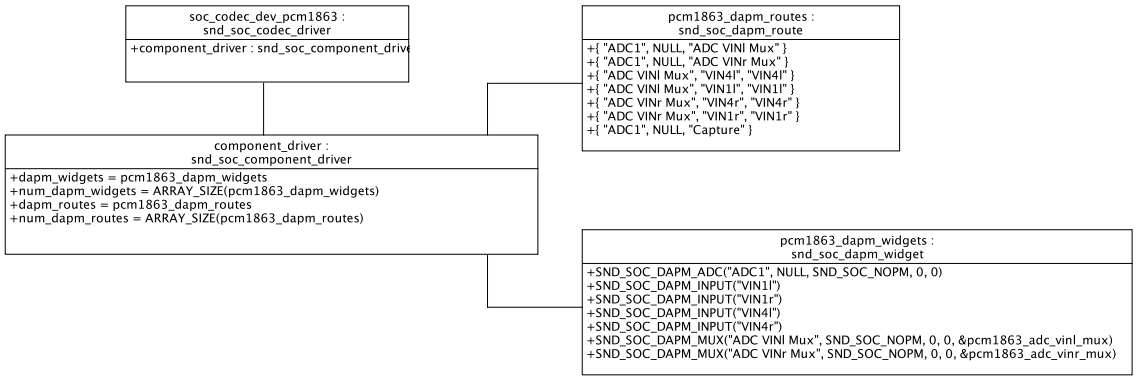

Abbildung 5.1.: Aufbau der Struktur *snd_soc_codec_driver*

Der Dynamic-Audio-Power-Management (DAPM) Bereich beschäftigt sich mit aktiven Komponenten. In der *soc_dapm.h* Bibliothek werden Strukturen für alle definierbaren Bauteile abgebildet.

Der ADU der *dapm_widget* Struktur wird beginnend mit *snd_soc_dapm_adc* definiert. Da das Ziel eine möglichst schnelle Implementierung ist, sowie die Soundkarte über einen an einem Netzteil angeschlossenen RPI betrieben wird, wird der ADU ohne Energiesparoptionen definiert. Es müssen der Name des Gerätes, der Stream-Name sowie für das Power-Management (PM) die Registeradresse, Anzahl der Shifts und der Inversion, angegeben werden. Aus Gründen der Übersichtlichkeit wird an dieser Stelle nicht der Stream-Name gesetzt, sondern erst zu einem späteren Zeitpunkt in der Verdrahtung definiert. Zusätzlich benötigt der ADU Eingänge, um Audio Signale über beide Kanäle aufzunehmen, welche über die *snd_soc_dapm_input* Struktur definiert werden.

Die Beschaltung der HiFi-HAT sieht als Eingang *Vin4* vom ADU für die Klinkenbuchse vor. Daraus folgt die Notwendigkeit eines MUXs für die Eingangsquelle des ADUs. Die Definition geschieht mittels *snd_soc_dapm_mux* mit folgenden Parametern: Name, die nächsten drei Parameter sind für PM relevant und als letztes ein Verweis auf die Einstellungsmöglichkeiten. Im Folgenden muss die Steuerung für den MUX implementiert werden. Im Vorfeld wird ein String-Array definiert, welches die Auswahlnamen beinhaltet. Zusätzlich wird ein Werte-Array angelegt, da der zu schreibende Wert inkrementiert wird. Das Register des ADU sieht hingegen vor, dass eine eins geschoben wird, womit sich die Wertefolge eins, zwei, vier und acht ergeben. An das *SOC_VALUE_ENUM_SINGLE_DECL* Makro wird ein Name, das Eingangsauswahl Register, die Anzahl der Shifts, die Bitmaske, das Auswahlnamen und das Wertearray übergeben. Im letzten Schritt muss eine neue Audio-Operation vom Typ *snd_kcontrol_new* erzeugt werden sowie der Aufruf des *SOC_DAPM_ENUM* Makros. Es wird der Name sowie die zuvor erzeugte nummerierte Liste übergeben. Dieser Vorgang

muss wiederholt werden, da die Eingangsquelle des ADUs für den linken sowie rechten Kanal ein eigenes Register besitzt. Anschließend existiert die Möglichkeit über das ALSA Bordmittel *alsamixer*, wie in Abbildung 5.2, die Eingangsquelle auf *Vin4* zu schalten.

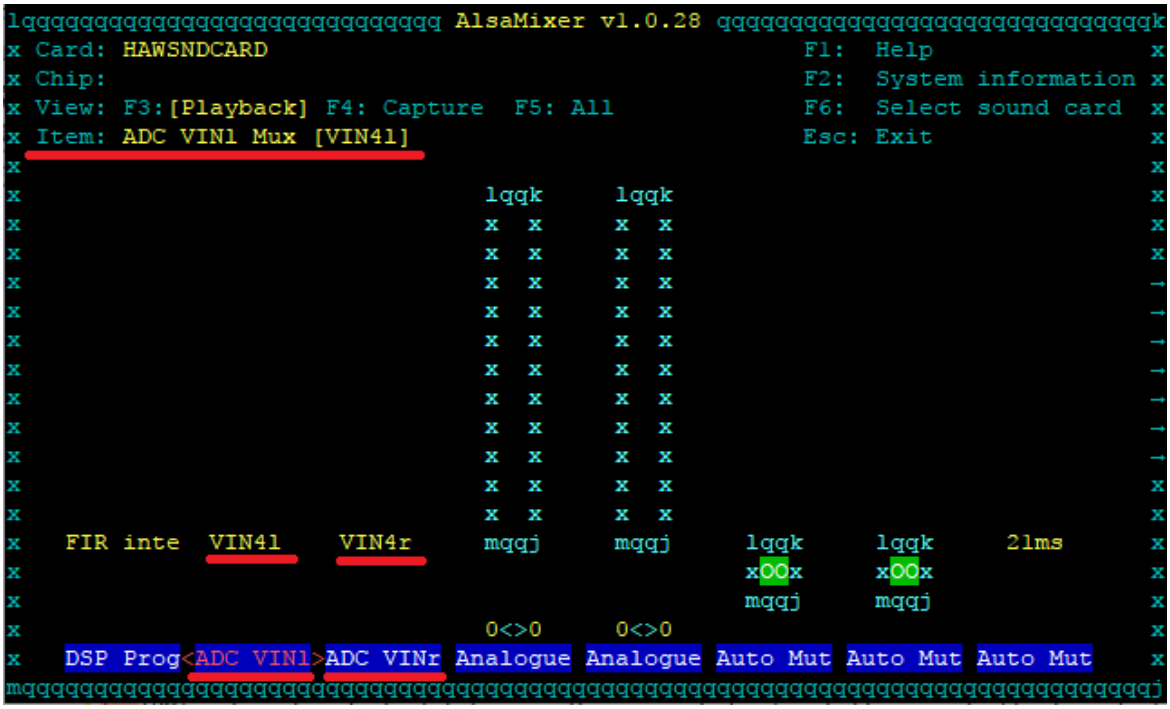

Abbildung 5.2.: Benutzeroberfläche des *alsamixer* in rot gekennzeichnet beide implementierten Multiplexer

Die *dapm_routes* Struktur definiert die Verdrahtung zwischen den Komponenten. Es müssen der ADU, die Eingänge, der MUX und der Stream entsprechend zusammengeführt werden. Bei der Definition ist die Anordung der Befehlsequenz essentiell. Zu Beginn muss eine Senke, gefolgt von einem Befehl beziehungsweise *NULL* bei einer direkten Verdrahtung und abschließend die Quelle angegebenen werden. Somit wird als Quelle der MUX und der Stream für den ADU als Senke definiert. Die Route für den MUX ist abhängig von dem Befehl, welcher den zweiten Parameter in der Definition darstellt. Wird die Eingangsquelle geändert, muss der dazugehörige Eingang als Quelle ausgewählt werden.

5.1.2. snd_soc_dai_driver

Abbildung 5.3 zeigt die Struktur *snd_soc_dai_driver*, welche aus dem Namen, den Einstellungsmöglichkeiten des Datenstroms sowie den DAI Operationen besteht. Die Parameter

der *snd_soc_dai_driver* Struktur sind von dem verbauten Baustein abhängig. Hier wird die Anzahl der Kanäle festgelegt, die möglichen Abtastraten und zu verarbeitende Formate eingestellt. Die *snd_soc_dai_ops* Struktur besteht im gegenwärtigen Stand aus der *hw_params* sowie der *set_fmt* Funktion.

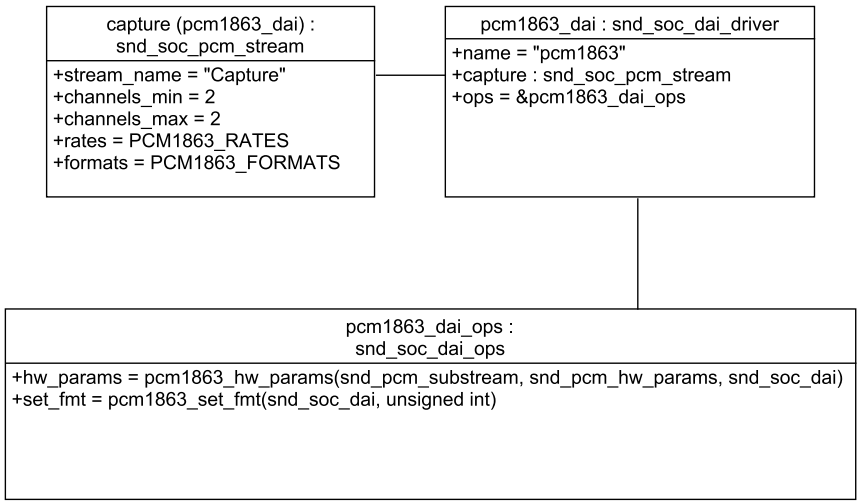

Abbildung 5.3.: Aufbau der Struktur *snd_soc_dai_driver*

Die *hw_params* Funktion ist das Kernstück zur Vorbereitung des ADU Bausteins vor der Nutzung. In ihr wird das Format, die Abtastrate und das Rollenverhalten definiert. Zur Einstellung der Parameter muss ein Zeiger auf den Substream, die Parameter sowie die DAI übergeben werden. Zu Beginn werden zwei Zeiger erzeugt. Der erste zeigt auf den Codec-Zeiger des DAI. Im zweiten Zeiger wird auf die Datenstruktur der privaten Variablen des ADU Codecs gezeigt. Dieser Aufruf ist besonders wichtig, da in diesem die *regmap* enthalten ist, welche innerhalb der nächsten Schritte beschrieben wird. Auf die *regmap* wird im Kapitel 5.1.4 näher eingegangen.

Der Aufbau gliedert sich in drei *switch case* Anweisungen. Mit dem Aufruf der *params_width(params)* Funktion wird die Anzahl der einzustellenden Bits zurückgegeben. Um das Rollenverhältnis einzustellen wird die private Format-Variable mit der *SND_SOC_DAIFMT_MASTER_MASK* verglichen. Für die Abtastrate wird die Funktion *params_rate(params)* benötigt, welche die geforderte Abtastrate zurückgibt. Anhand der entsprechenden Werte wird über den *regmap_update_bits* Aufruf in das entsprechende Register mit der benötigten Maske der einzustellende Wert geschrieben. Für Debugging und Fehlermeldungen wird der erste erzeugte Zeiger verwendet. Im Falle eines Fehlers bekommt der Zeiger vom Codec eine Nachricht mit den störungsverursachenden Parame-

tern. Die *set_fmt* Funktion wird benötigt, um das aktuelle Format zu setzen, welches in der *hw_params* für die Rollenverteilung benutzt wird.

5.1.3. Initialisierungs- und Deinitialisierungsfunktion

Die *probe* Funktion erstellt die private Variablen-Struktur. In dieser Struktur können Informationen über die *regmap*, Takte, Formate, Phase-Locked-Loop (PLL) und vieles mehr gespeichert werden. Mithilfe dieser Variablen können Funktionalitäten, welche von Informationen der Variablen abhängig sind, implementiert werden. Zunächst erfolgt eine Reservierung von Speicher für diese Struktur über den *devm_kzalloc* Befehl. Im nächsten Schritt wird die übergebene *regmap* in die private Variablen-Struktur geschrieben, sowie die Struktur in die Treiber Daten Struktur des Gerätes geschrieben. Anschließend wird die Funktion *snd_soc_register_codec* aufgerufen, um mit dem Gerät den Codec und die DAI zusammen zu registrieren. Hieraus entsteht in Abbildung 5.4 das folgende Aktivitätsdiagramm.

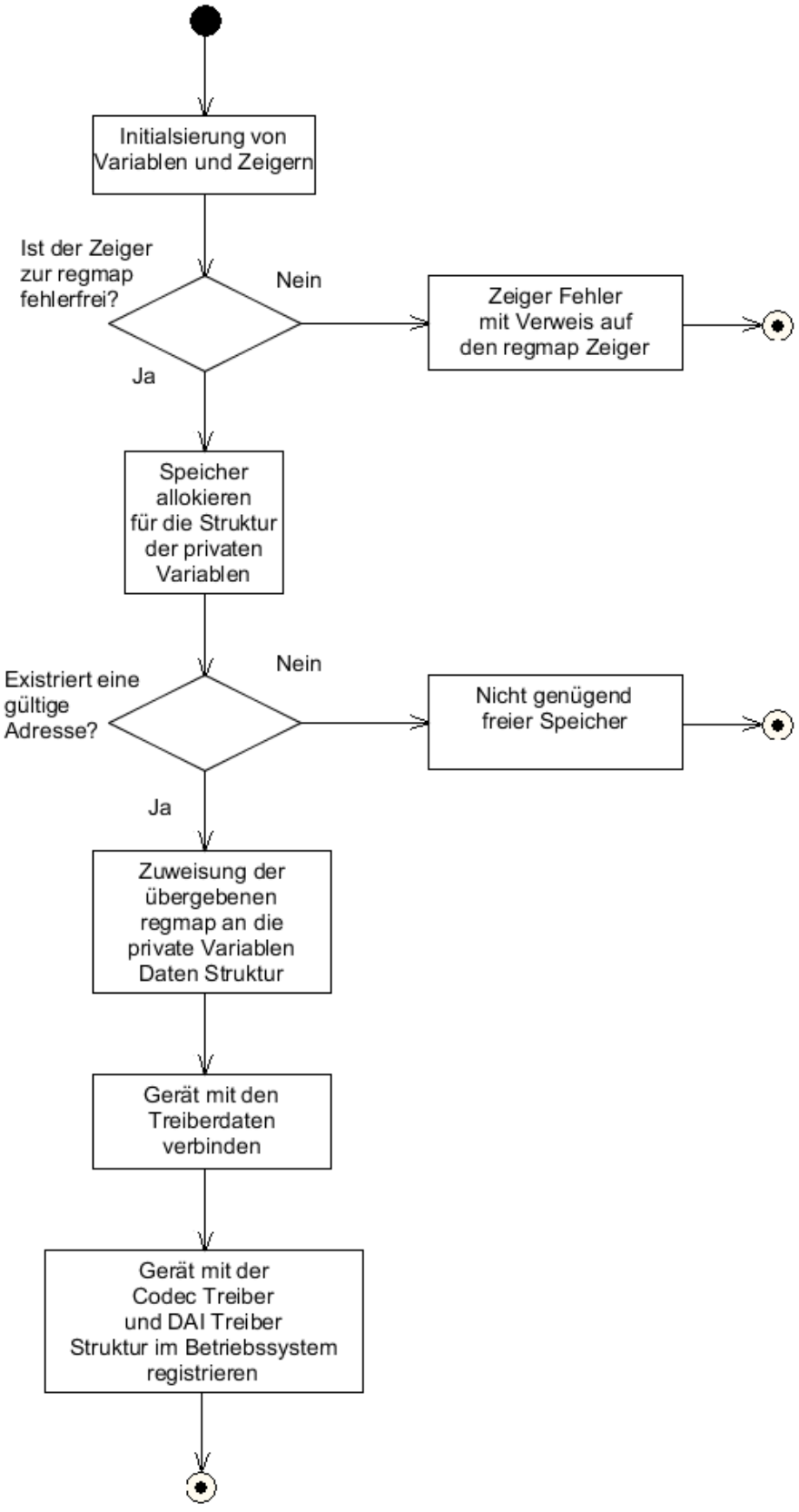

Abbildung 5.4.: Aktivitätsdiagramm der Initialisierungsfunktion

Zum Deinitialisieren wird die entgegengesetzte *snd_soc_unregister_codec* Funktion verwendet, in welcher das Gerät übergeben sowie vom Codec-Treiber getrennt werden muss.

Hervorzuheben ist das Ende beider Funktionen sowie der *regmap_config* Konfiguration. Das *EXPORT_SYMBOL_GPL* Makro exportiert die jeweilige Funktion als Symbol an den Kernel. Somit können sie außerhalb des Codes aufgerufen werden, da der Kernel die Adresse übergeben bekommt.

5.1.4. Konfiguration über I2C

Um den ADU über die I2C-Schnittstelle konfigurieren zu können, wird eine Erweiterung des Codec-Treibers benötigt. Die Erweiterung des Treibers trifft die notwendigen Vorbereitungen für die Nutzung des Gerätes über eine Schnittstelle. Die Schnittstellen sind in Subsysteme unterteilt und besitzen speziell angepasste Funktionen. Zu den wesentlichen Bestandteilen der I2C -Treiber Struktur gehört eine *probe* Funktion, eine *remove* Funktion, eine ID-Tabelle sowie eine Matching Tabelle. Die ID-Tabelle entscheidet, welche I2C-Geräte zugelassen werden und die *of_device_id* Struktur wird mit dem zum *compatible* Attribut Bauteil String ausgestattet. Somit entsteht folgende in Abbildung 5.5 dargestellte Struktur.

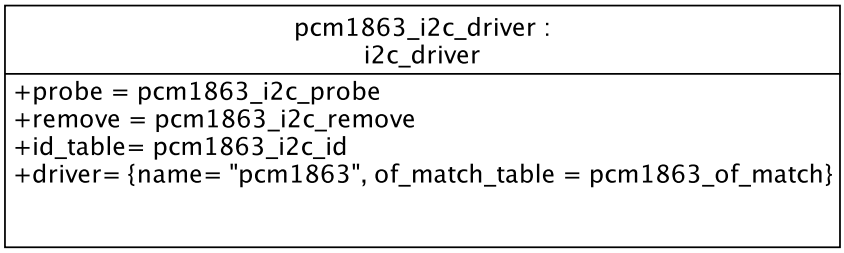

Abbildung 5.5.: Aufbau der Struktur *i2c_driver*

In der *probe* Funktion wird eine Register-Map (regmap) initialisiert. In diesem Schritt werden vom I2C Gerät sämtliche Register in eine Struktur geladen sowie ein Zeiger erzeugt, welcher auf diese Struktur zeigt. Anschließend wird die Probe Funktion des Codec-Treibers mit den Parametern des Gerätes sowie der allozierten regmap aufgerufen. Die Funktion dec Codec-Treiber kann direkt aufgerufen werden, da sie als Modul exportiert wurde und dem Betriebssystem zur Verfügung steht. Weiterhin muss der Treiber mit einer *remove* Funktion die Verbindung zum Gerät entfernen können.

5.1.5. Bauen von Loadable-Kernel-Modules

Zuallererst müssen für die aktuelle Betriebssystemversion *linux-header* installiert werden. Die Header-Dateien beinhalten die zugehörigen Deklarationen für verschiedenste Datenstrukturen und Funktionen des Kernels. Die aktuelle Version des Betriebssystems kann über den Kommandozeilen Befehl *uname -r* unter Raspbian in Erfahrung gebracht werden. Hierbei ist darauf zu achten, dass der numerische Wert der linux-header mit der Kernel-Version und den Werten der Konfiguration übereinstimmt. Dies bedeutet, wenn ein *custom* Kernel zum Einsatz kommt, müssen in der *.config* Datei der *header* dieselben *flags* gesetzt sein.

Im Anschluss wird ein *kbuild Makefile* benötigt, welches speziell für Kernel-Module genutzt wird. In der ersten Zeile wird mit dem Parameter *obj-m* eine Zieldatei definiert. Mit dem *-C* Flag wird in den Kernel Ordner gewechselt, bevor ein *make* Befehl ausgeführt wird. Das *M=$(PWD)* übergibt dem *make* Aufruf den Ort, an dem sich die Projekt Dateien befinden.

Abschließend wird das Modul erzeugt und befindet sich im gleichen Ordner wie die Projekt Dateien. Im nächsten Schritt folgt die Installation.

5.1.6. Installation und Nutzung von Loadable-Kernel-Modules

Die gebauten LKMs müssen unter */lib/modules/'uname -r'/kernel/sound/soc/codecs/* abgespeichert werden. Anschließend müssen die Abhängigkeiten mit dem Befehl *depmod* festgelegt werden. Hinterher wird in der Datei *Modules.dep* festgehalten, welches Modul voneinander abhängt. Die Einbindung in das Betriebssystem erfolgt über die *modprobe* Anweisung. Da ab diesem Zeitpunkt das Betriebssystem die LKMs und ihre Abhängigkeiten kennt, wird ein passendes DTO benötigt, um die Soundkarte vom Betriebssystem erkennen zu lassen.

5.2. Device-Tree-Overlay

In dieser Aufgabe eignet sich ein DTO besonders gut, da die Schnittstellen des RPI erweitert werden müssen. Somit wird eine zusätzliche Erweiterung für die Unterstützung der Soundkarte benötigt, welche am besten durch ein DTO realisieren lässt.

5.2.1. Single-Audio-Link Implementierung

Im Vorfeld wird ein DTO mit nur einem Audio-Link entwickelt. Da für den TI-5142 DAU ein Codec-Treiber bereits existiert und der simple-card Maschinentreiber verwendet wird, ist für

die Ermöglichung der Wiedergabe ein funktionierendes DTO notwendig. Für die Implementierung werden mehrere Fragmente benötigt.

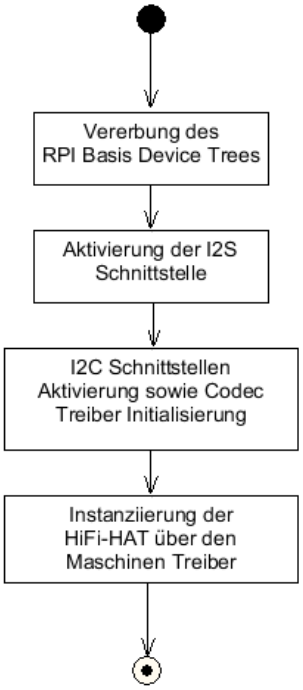

Abbildung 5.6.: Aktivitätsdiagramm des Device-Tree

Bei einem DTO ist die erste Anweisung obligatorisch und stellt die Kompatibilität mit dem DT des RPI her. Durch Setzen des Attributs *status* auf *okay* wird daraufhin die I2S-Schnittstelle aktiviert. Innerhalb des nächsten Knotens wird über die I2C-Schnittstelle der Codec-Treiber mit einem Gerät verbunden, welches sich an der Stelle der I2C Slave Adresse befindet. Darüber hinaus wird zu Beginn der I2C-Schnittstellen Initialisierung die Größe der Register Werte und Adressen eingestellt. Hierzu werden die Attribute *address-cells* sowie *size-cells* benötigt. Die Register der Codecs werden als *uint32* eingestellt und sind nur über eine Adresse erreichbar, wodurch ein Wert von null resultiert. Anschließend muss eine Verbindung zum Maschinentreiber hergestellt werden. Dieser erhält Informationen über die Soundkarte. Hierzu gehören der Name, das Format der Datenströme sowie Informationen über Clock- und Bitmaster. Nachfolgend wird im Label des *cpu_dai* die Adresse der Schnittstelle, die Kanalzahl sowie die Bittiefe an den Maschinentreiber übergeben. Zum Schluss muss der vorher definierte Codec angegeben werden. Mit dem Grundgerüst aus Abbildung 5.6 wird der erste Compilier Vorgang gestartet.

```
pi@raspberrypi:~/source/haw-snd-card/02_device-tree-overlay $ ls -la
total 16
drwxr-xr-x 2 pi pi 4096 Dec  4 18:45 .
drwxr-xr-x 8 pi pi 4096 Dec  4 18:42 ..
-rw-r--r-- 1 pi pi    0 Dec  4 18:42 .gitkeep
-rw-r--r-- 1 pi pi 1068 Dec  4 18:42 hawsndcard.dts
-rw-r--r-- 1 pi pi  947 Dec  4 18:42 hawsndcard-pcm512x-playback.dts
pi@raspberrypi:~/source/haw-snd-card/02_device-tree-overlay $ dtc -@ -H epapr -O
 dtb -o hawsndcard.dtbo -Wno-unit_address_vs_reg hawsndcard.dts
pi@raspberrypi:~/source/haw-snd-card/02_device-tree-overlay $ ls -la
total 20
drwxr-xr-x 2 pi pi 4096 Dec  4 18:45 .
drwxr-xr-x 8 pi pi 4096 Dec  4 18:42 ..
-rw-r--r-- 1 pi pi    0 Dec  4 18:42 .gitkeep
-rw-r--r-- 1 pi pi 2207 Dec  4 18:45 hawsndcard.dtbo
-rw-r--r-- 1 pi pi 1068 Dec  4 18:42 hawsndcard.dts
-rw-r--r-- 1 pi pi  947 Dec  4 18:42 hawsndcard-pcm512x-playback.dts
```

Abbildung 5.7.: Compiliervorgang

Wird dieses DTO mit folgendem in Abbildung 5.7 dargestellten Befehl durch den Compiler übersetzt, entsteht die binäre und für den RPI lesbare Device-Tree-Blob-Overlay (DTBO) Datei. Darüber hinaus kann der DT dynamisch über einen Terminal Aufruf erweitert sowie die DTBO getestet werden.

Erklärung der Flags des Compiler Befehls:

- -@ unterlässt das Werfen von Fehlern, wenn Referenzen fehlen, z.B. das i2s Label
- -H epapr es gelten phandle Eigenschaften
- -I Eingabeformat: dts, dtb, fs
- -i Eingabedatei
- -O Ausgabeformat: dtb, dtbo, asm
- -o Zieldatei
- -Wno-unit_address_vs_reg Vermeidung zur Anzeige von Warnungen

Da nur eine Verbindung hergestellt wird, lässt sich mit diesem DTO nur eine Funktion abbilden. Um jeweils zwei Geräte mit verschiedenen Codec-Treibern zu verbinden, muss eine Erweiterung erarbeitet werden sowie dem Maschinentreiber zwei Audio Verbindungen mitgeteilt werden.

5.2.2. Dual-Audio-Link Implementierung

Für die Abbildung beider Funktionalitäten wird ein Dual-Audio-Link DTO entwickelt. Hierfür kann Quellcode aus dem Single-Audio-Link DTO wiederverwendet werden. Die Definitionen

der Knoten und Initialisierungen der Schnittstellen bleiben bestehen. In der I2C Definition des Codecs wird eine zusätzliche Codec Deklaration ergänzt. Die größten Veränderungen erhält der *sound* Knoten. In diesem wird nur noch die Verbindung zum Maschinentreiber hergestellt sowie der Name übergeben. Weitere Deklarationen erfolgen unter dem Attribut *dai-link*. Es wird das Format, die CPU-seitig Adresse der Schnittstelle und die Referenz des jeweiligen Codecs angegeben.

5.2.3. EEPROM-Flashvorgang

Um den EEPROM flashen zu können wird das RPI HAT Git Repo gecloned und der DTC benötigt. Die hierfür notwendigen Dateien befinden ich in dem Ordner *eepromutils*. Zunächst wird die *eeprom_settings.txt* Datei angepasst. Hier werden Herstellerinformationen, der Gerätetyp sowie eventuell bentögite GPIO-Pins hinterlegt. Abschließend erfolgt über den Befehl *eepmake* in Abbildung 5.8 eine Kompilierung der Klartext Datei in ein binäres Format.

```
pi@raspberrypi:~/tmp/hat/hats/eepromutils $ ./eepmake eeprom_settings.txt haw-dt
l.eep hawsndcard.dtbo
Opening file eeprom_settings.txt for read
UUID=e8156b26-ca92-4719-b2e0-5bc12686ee07
Done reading
Opening DT file hawsndcard.dtbo for read
Adding 2207 bytes of DT data
Writing out...
Writing out DT...
Done.
```

Abbildung 5.8.: *eepmake* Befehl mit Ausgabe

Nachdem *eepmake* besteht die Möglichkeit des Flashvorganges auf den EEPROM. Wenn die Spezifikation eingehalten wird, dann besitzt der Baustein einen 4 kilo Bytes großen Speicher. Dieser ist größer, als die Binärdatei. Falls der EEPROM zuvor bereits mit einem anderen Code bespielt worden ist, empfiehlt es sich vor dem Beschreiben den Speicher des Bausteins zu säubern, da ansonsten Teile des Codes vom vorhergegangen Flashvorgang erhalten bleiben könnte.

Mit dem *dd* Befehl wird eine 4 Kilo Bytes große Datei mit Nullen erzeugt. Abbildung 5.9 zeigt, wie diese Datei mit dem *hexdump* Befehl eingesehen werden kann.

```
pi@raspberrypi:~/tmp/hat/hats/eepromutils $ dd if=/dev/zero ibs=1k count=4 of=bl
ank.eep
4+0 records in
4+0 records out
4096 bytes (4.1 kB) copied, 0.00109864 s, 3.7 MB/s
pi@raspberrypi:~/tmp/hat/hats/eepromutils $ hexdump blank.eep
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0001000
```

Abbildung 5.9.: Erzeugen der *blank.eep* Datei und Anzeige des Inhalts

Abschließend kann der Baustein im ersten Schritt mit der zuvor erzeugten Datei gesäubert und anschließend mit den gewünschten Daten beschrieben werden. Um den Schreibschutz des EEPROMs aufzuheben, muss vor der Ausführung des Befehls sichergestellt werden, dass der Pin auf Massepotential gezogen wird.

```
pi@raspberrypi:~/tmp/hat/hats/eepromutils $ sudo ./eepflash.sh -w -f=haw-dt.eep
-t=24c32
This will attempt to talk to an eeprom at i2c address 0x50. Make sure there is a
n eeprom at this address.
This script comes with ABSOLUTELY no warranty. Continue only if you know what yo
u are doing.
Do you wish to continue? (yes/no): yes
Writing...
4+1 records in
4+1 records out
2323 bytes (2.3 kB) copied, 9.51394 s, 0.2 kB/s
Done.
```

Abbildung 5.10.: *eepflash* Befehl mit Ausgabe

Da der Speicher auf 4096 Bytes beschränkt ist, kann ein DT oder DTO nur mit einer Größe, welche kleiner oder gleich des Bausteins verwendet werden. Diese Größe darf in keinem Fall überschritten werden.

5.2.4. Verifizierung

Wenn ein DT erfolgreich eingebunden ist, werden entsprechende Einträge unter */proc/device-tree* und */sys/firmware/devicetree/base* angelegt. Innerhalb dieser Verzeichnisse sind Informationen wie der Name des Gerätes, der Gerätetyp oder die Betriebsparameter der entsprechenden Geräte enthalten.

```
003349.279: Device tree loaded to 0x2effbb00 (size 0x44de)
```

Abbildung 5.11.: Ausgabe aus dem Videocore Debug Log

Existiert kein Eintrag nach dem Aufruf des *dtoverlay* Befehls muss es Fehler gegeben haben, die ein Einbinden unterbunden haben. Auf Maßnahmen zur Fehlersuche bei der Entwicklung von DT wird in Kapitel 5.4 näher eingegangen.

```
pi@raspberrypi:~ $ cat /proc/device-tree/hat/
name         product      product_id   product_ver  uuid         vendor
pi@raspberrypi:~ $ cat /proc/device-tree/hat/name
hatpi@raspberrypi:~ $ cat /proc/device-tree/hat/product
Audio Boardpi@raspberrypi:~ $ cat /proc/device-tree/hat/vendor
HAW Hamburgpi@raspberrypi:~ $ cat /proc/device-tree/hat/product_id
0x0001pi@raspberrypi:~ $ cat /proc/device-tree/hat/product_ver
0x0002pi@raspberrypi:~ $ cat /proc/device-tree/hat/uuid
8185cd6e-4525-420d-9a76-ccf7bb4c18f8pi@raspberrypi:~ $
```

Abbildung 5.12.: Informationen des Overlays im proc Verzeichnis

5.3. Anleitung zur Inbetriebnahme der HiFi-HAT

Das Folgende Kapitel beschreibt Vorkehrungen, die getroffen werden müssen zur Inbetriebnahme der Soundkarte. Anschließend wird geprüft, ob das Betriebssystem sie erkennt.

5.3.1. Vorbereitungen

Im Folgenden wird aufgezählt welche Schritte notwendig sind, um die Soundkarte im Raspbian Betriebssystem zu installieren.

- Herunterladen der *linux-header* Dateien
- Kompilieren der .ko LKM Codec-Treiber
- Codec-Treiber in den entsprechenden Pfad kopieren
- Mit *depmod* Abhängigkeiten der Module erneuern
- Über den Befehl *modprobe* die Module ins Betriebssystem laden
- In der */boot/config.txt* I2C- und I2S- Schnittstellen des RPIs beim Booten aktivieren
- Neustart

Der EEPROM beinhaltet bereits das DTO, somit muss dieses nicht erneut compiliert werden. Es müssen lediglich die Codec-Treiber erzeugt werden sowie zwei Zeilen in der */boot/-config.txt* Datei aukommentiert werden, um die Schnittstellen ab dem Bootvorgang zur Verfügung zu stellen. Dieser Schritte werden vom *installation_script.sh* Shell-Skript übernommen, wenn es mit *super user* Rechten ausgeführt wird. Es hat sich herausgestellt, dass die Kernel-Version 4.4.50 Probleme beim Laden der Module hat. Diese Probleme sind bei Version 4.9.60 nicht mehr aufgetreten. Das Skript prüft, ob die Kernel-Version mindestens der Kernel-Version 4.9.60 entspricht und wenn dies nicht der Fall ist, wird der Kernel auf diese Version über das Internet aktualisiert.

5.3.2. Instanziierung der Soundkarte

Innerhalb der Initialisierungsfunktion des Maschinentreibers werden im Vorfeld Zeiger erzeugt sowie initialisiert. Hiernach wird aus dem DT die Anzahl der DAI Links ausgewertet. Wird mehr als nur ein DAI-Link gefunden, muss dies bei der Größe der Strukturen berücksichtigt werden. Im Folgenden wird für die private Variablen Daten-, DAI-Eigenschaften- und DAI-Link-Struktur Speicher reserviert. Weiterhin findet die Initialisierung der *snd_soc_card* Struktur statt. Es werden Informationen, beispielsweise die private Variablen-Datenstruktur, das Gerät, die DAI-Links, die Anzahl der DAI-Links und der Besitzer, an die Struktur übergeben. Darüber hinaus wird im nächsten Schritt überprüft, ob der Knoten Zeiger auf ein erreichbares Gerät zeigt. Im erfolgreichen Fall werden über die *asoc_simple_card_parse_of* Funktion weitere relevante Parameter ausgelesen. In einem Fehlerfall beim Auslesevorgang wird zusätzlich eine *asoc_simple_card_clean_reference* Funktion aufgerufen. Wenn zum Zeitpunkt der Abfrage kein Gerät am Knoten Zeiger verfügbar ist, wird die Initialisierung der Soundkarte über einen Zeiger der *asoc_simple_card_info* probiert. Dem Zeiger wird der *platform_data* Zeiger des Gerätes zugewiesen. Es muss überprüft werden, ob an dieser Stelle Informationen stehen, sowie eine mehr als ausreichende Anzahl dieser gegeben ist. Sind beide Bedingungen erfüllt, werden die Informationen in die *snd_soc_card* übertragen. Im Anschluss muss die Soundkartenstruktur mit der privaten Variablen-Datenstruktur verbunden werden. Abschließend wird das Gerät zusammen mit der Soundkarten-Struktur beim Betriebssystem angemeldet. Hieraus ergibt sich das in Abbildung 5.13 dargestellte Aktivitätsdiagramm:

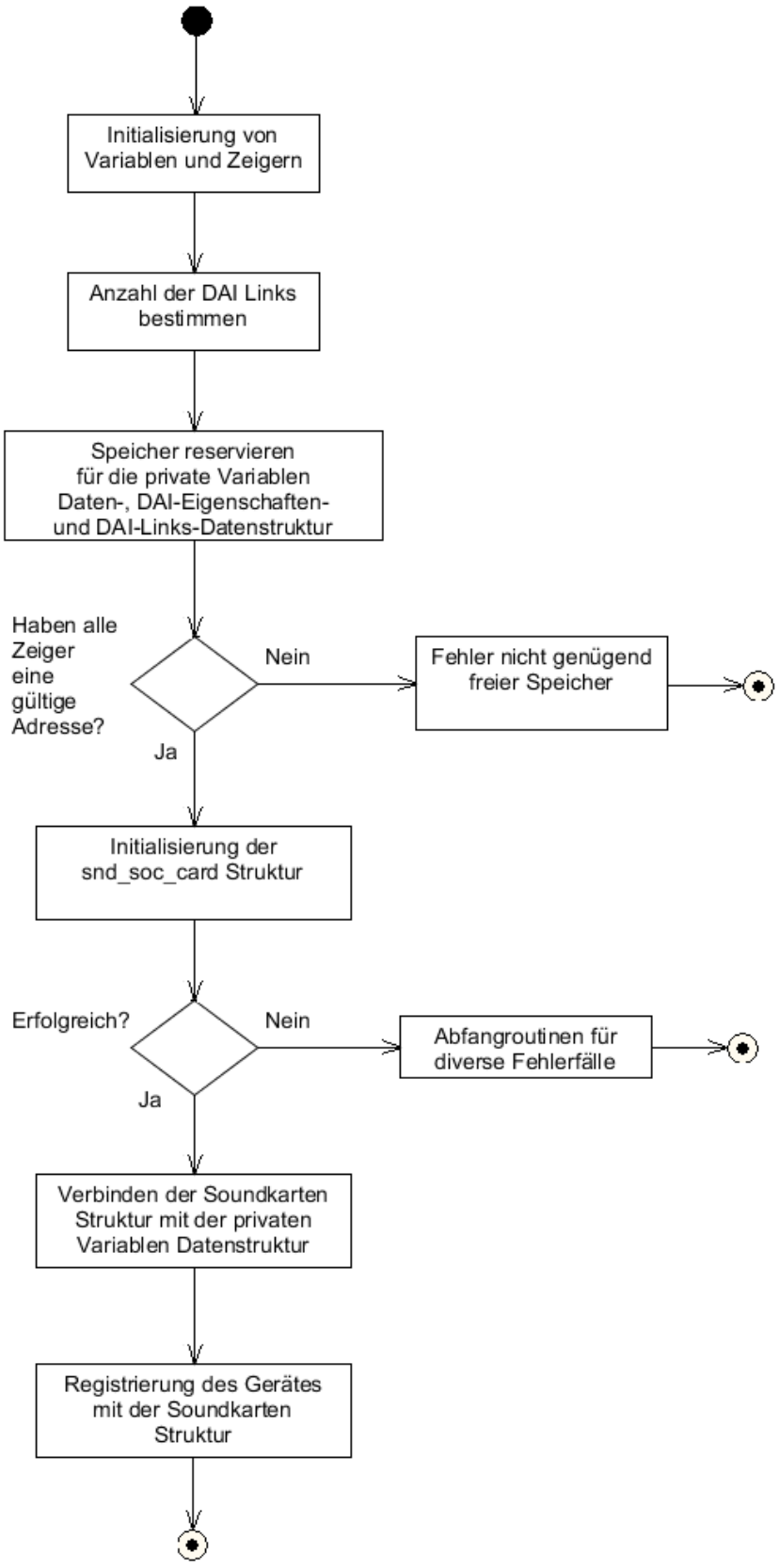

Abbildung 5.13.: Aktivitätsdiagramm des Maschinentreibers

5.3.3. Verifizierung

Wenn das Betriebssystem vorbereitet ist, kann über ALSA Boardmittel die HiFi-HAT als Aufnahme- sowie Wiedergabegerät angezeigt werden. Hierfür wird der Befehl *arecord -l* und *aplay -l* verwendet. Eine Alternative bietet der *ls -l /dev/snd/* Befehl, welcher alle Sound Geräte wie in Abbildung 5.14 auflistet.

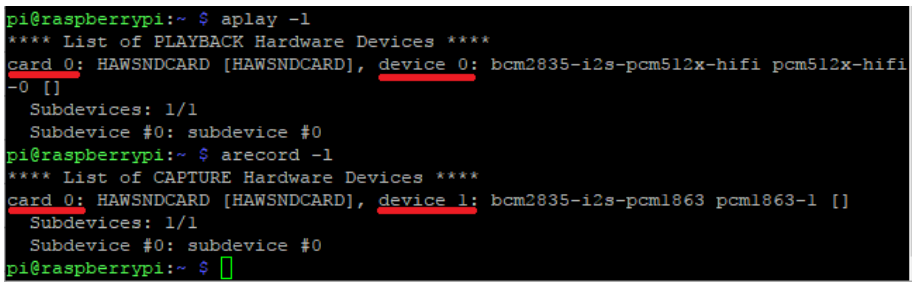

Abbildung 5.14.: Ausgabe der Audiogeräte

Im nächsten Schritt werden die ALSA Befehle zum Aufnahme als auch zur Wiedergabe verwendet. Folgende Flags sind hierfür erforderlich:

- -r zum Einstellen der Abtastrate (44100, 48000, 88200, 96000, 192000)
- -f zum Einstellen des Dateiformates (S16_LE, S24_LE)
- -c zum Einstellen der Kanäle (es werden nur zwei Kanäle unterstützt)
- -Dhw:x,x zum Einstellen des Gerätes (0,0 für das Wiedergabegerät und 0,1 für das Aufnahmegerät siehe 5.14, kann abweichen)
- -vvv für *verbose* somit ausführlicher Ausgabe, ist im Regelfall nicht relevant

Die Aufnahme kann in eine Datei erfolgen oder mit der Wiedergabe verknüpft werden. Das eingespeiste Signal wird in diesem Fall unverändert ausgegeben. Unter Abbildung A.5 und A.6 sind Aufnahmen und Wiedergaben mit ausführlichen Outputs dargestellt. Es wird der geschriebene Wert jeder zweiten Periodenlänge angezeigt sowie die Pegelauslenkung in Prozent aufgeführt. Da die Aufnahme in eine Datei geschrieben wird, muss die Wiedergabe identische Werte beziehungsweise Pegelauslenkungen aufweisen, was aus den Abbildungen A.5 und A.6 entnommen werden kann.

Zusätzlich besteht die Option über das */proc* Verzeichnis Informationen während des Betriebes wie in Abbildung 5.15 zu erfragen.

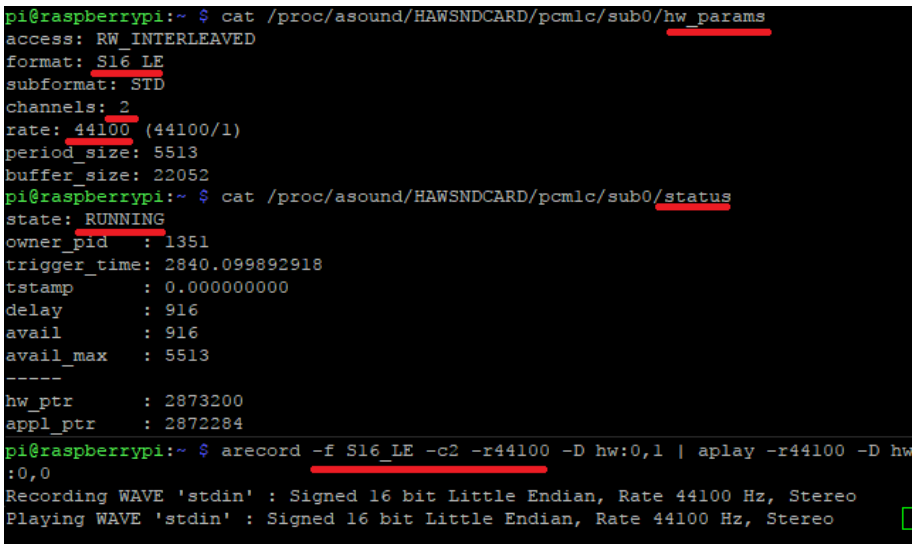

Abbildung 5.15.: Ausgabe der Hardware Parameter und des Statuses des Aufnahmegerätes in einem zweiten Terminal während des Durchreichen der Signale

5.4. Debugging

In diesem Kapitel werden Möglichkeiten, welche zum systematischen Vorgehen von Fehlerursachen benutzt werden, vorgestellt. Wenn möglich, wird anschaulich ein Beispiel präsentiert und wie mit den Informationen aus Fehlerausgabe das Problem behoben werden kann.

5.4.1. Kernel-Log und Videocore-Debug-Log

Bei unerwartetem Verhalten lassen sich zunächst Informationen aus dem Kernel Log gewinnen. Hierzu kann der Befehl *dmesg* benutzt werden. Die Funktion *dmesg* steht für *display message* und wird zur Kontrolle des Kernel Ring Speichers verwendet.

Für die Fehlersuche bei DT oder DTO wird bei einem RPI das Videocore-Debug-Log benutzt. Hier wird aufgezeichnet, wenn ein DT geladen wird. Kann ein DT nicht angezogen werden, so wird die Ursache des Fehlers aufgezeichnet.

5.4.2. Stacktraces

Ein Stacktrace beinhaltet die Ausgabe des Inhaltes vom Stack. Mit einem Stacktrace können die Aufrufe bis zu einem Programmabsturz aufgezeichnet, sowie die Ursachen rekonstruiert werden. Da diese Traces außerordentlich lang sind, empfiehlt es sich von unten zu beginnen. Oftmals wird der fehlerhafte Aufruf gefunden, da dieser unter Umständen mit falschen Parametern ausgeführt wird. Weiterhin muss ausgemacht werden, an welcher Stelle der fehlerhafte Parameter eingestellt worden ist, um in der Zukunft dieses Problem zu vermeiden.

```
pi@raspberrypi:~ $ strace arecord -fdat test.wav > strace.txt
open("/dev/snd/controlC0", O_RDWR|O_CLOEXEC) = 3
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
ioctl(3, SNDRV_CTL_IOCTL_PVERSION or USBDEVFS_CONTROL, 0x7eb22b34) = 0
ioctl(3, SNDRV_CTL_IOCTL_PCM_PREFER_SUBDEVICE, 0x7eb22b74) = 0
open("/dev/snd/pcmC0D0c", O_RDWR|O_NONBLOCK|O_CLOEXEC) = -1 ENOENT (No such file or directory)
close(3)                                = 0
write(2, "arecord: main:722: ", 19arecord: main:722: )     = 19
open("/usr/share/locale/en_GB.UTF-8/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or dire
ctory)
open("/usr/share/locale/en_GB.utf8/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or direc
tory)
open("/usr/share/locale/en_GB/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
open("/usr/share/locale/en.UTF-8/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or directo
ry)
open("/usr/share/locale/en.utf8/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or director
y)
open("/usr/share/locale/en/LC_MESSAGES/alsa-utils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
write(2, "audio open error: No such file o"..., 43audio open error: No such file or directory) = 43
write(2, "\n", 1
)                       = 1
exit_group(1)                           = ?
+++ exited with 1 +++
pi@raspberrypi:~ $ strace arecord -fdat -Dhw:0,1 test.wav &> strace.txt
open("/dev/snd/controlC0", O_RDWR|O_CLOEXEC) = 3
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
ioctl(3, SNDRV_CTL_IOCTL_PVERSION or USBDEVFS_CONTROL, 0x7eaf7d8c) = 0
ioctl(3, SNDRV_CTL_IOCTL_PCM_PREFER_SUBDEVICE, 0x7eaf7dcc) = 0
open("/dev/snd/pcmC0D1c", O_RDWR|O_NONBLOCK|O_CLOEXEC) = 4
fcntl64(4, F_SETFD, FD_CLOEXEC)         = 0
close(3)                                = 0
```

Abbildung 5.16.: Ausschnitte der Stacktraces zum *arecord* Befehl ohne und mit dem *-Dhw:0,1* Flag

Beispielhaft wird in Abbildung 5.16 vorgeführt, wie durch eine Standard Einstellung beim Starten der Aufnahme das falsche Gerät ausgewählt worden ist. Es muss zusätzlich ein *flag* gesetzt werden, welches bei der Soundkarte das zweite Gerät unter der Zahl eins setzt, da unter der Null das Wiedergabegerät ist.

5.4.3. Event-Tracing

Tracepoints bieten die Möglichkeit einen genauen Einblick in Betriebssystemabläufe zu erhalten. So ist es möglich einen Funktionsaufruf als ein Event zu setzen. Dies bewirkt, dass alle Prozesse, die diesen Funktionsaufruf ausführen, mit den dazugehörigen Parametern aufgezeichnet werden. Hierfür wird die *set_event* Schnittstelle mit folgenden Schritten benötigt.

- echo Beispiel_Event » /sys/kernel/debug/tracing/set_event
- Ausführen eines Programms, welches dieses Event beinhaltet
- cat /sys/kernel/debug/tracing/trace

Im Falle der Implementierung des *dapm_mux* kann der *regmap_write* Funktionsaufruf aufgezeichnet werden. Dies hilft zum Nachvollziehen der Werte, welche in das Register geschrieben werden.

```
root@raspberrypi:~# echo "regmap:regmap_reg_write" > /sys/kernel/debug/tracing/set_event
root@raspberrypi:~# alsamixer
root@raspberrypi:~# cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
# entries-in-buffer/entries-written: 6/6   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
      alsamixer-1106  [002] ....  1888.366824: regmap_reg_write: 1-004a reg=6 val=4
      alsamixer-1106  [000] ....  1888.600245: regmap_reg_write: 1-004a reg=6 val=2
      alsamixer-1106  [000] ....  1888.812684: regmap_reg_write: 1-004a reg=6 val=1
      alsamixer-1106  [000] ....  1889.860884: regmap_reg_write: 1-004a reg=6 val=2
      alsamixer-1106  [000] ....  1890.071722: regmap_reg_write: 1-004a reg=6 val=4
      alsamixer-1106  [000] ....  1890.281235: regmap_reg_write: 1-004a reg=6 val=8
root@raspberrypi:~#
```

Abbildung 5.17.: Eventtrace zum Funktionsaufruf *regmap_reg_write*

In Abbildung 5.17 wird das Event auf den Funktionsaufruf *regmap_reg_write* gesetzt. Mit dem Befehl *alsamixer* wird der Multiplexer durch die verschiedene Quellen durchgeschaltet. Bei der Ausgabe des Mitschnitts werden alle Funktionsaufrufe mit dem Prozessnamen, der Prozessnummer, dem Zeitstempel, der Adresse wohin geschrieben wird, dem Register sowie der Wert festgehalten.

5.4.4. ALSA xrun_debug

Für detailliertere ALSA Ausgaben, gibt es im */proc* Verzeichnis die *xrun_debug* Datei. Voraussetzung hierfür ist, dass der Kernel mit entsprechenden *flags* gemäß [23] und [24] compiliert wird. Im Folgenden muss das die tiefe des Logs definiert werden:

- 1 Grundlegendes Debugging - zeigt *xruns* in der *ksyslog*-Schnittstelle an
- 2 Ablegen des Stacks
- 4 Überprüfung der Jiffies

- 8 Ablegen der Position bei jedem Perioden Update Aufruf
- 16 Ablegen der Position bei jedem Hardware Zeiger Update Aufruf
- 32 Aufzeichnung der letzten zehn Ringspeicher Positionen
- 64 Anzeige der letzten zehn Ringspeicher Position, einmalig wenn der erste Fehler geschieht

Um mehrere Eigenschaften gleichzeitig zu aktivieren, müssen die jeweiligen Werte zusammen addiert werden. In der Praxis eignen sich folgende Kombinationen für den Befehl.

echo X > /proc/asound/cardx/pcmxx/xrun_debug

Wird eine Drei geschrieben, wird grundlegendes Debugging und die Ablage des Stack Speichers aktiviert, welches für die Suche eines Grundes ist, wenn ein Datenstrom beendet wird. Beim Wert elf und 27 können zusätzlich die Werte des Treibers überprüft werden.

6. Funktionstest

Im Anschluss einer erfolgreichen Implementierung muss ein Funktionstest erfolgen. Zielsetzung ist die Beweisführung einer Ordnungsgemäßen Funktion der Aufsteckplatine.

6.1. Testaufbau

Es wird ein RPI im Headless Modus mit der Aufsteckplatine in Betrieb genommen. Die Verbindung vom Notebook zum RPI wird über eine SSH-Verbindung ermöglicht, welche über ein Ethernet Kabel im gleichen Netzwerk sind.

Über den **HAMEG HMF2525** Funktionsgenerator werden Testsignale eingespeist. Für die Auswertung steht ein **Tektonix TDS 2024C** Oszilloskop sowie der **Rohde und Schwarz UPV 66** Audio Analyzer zur Verfügung. Darüber hinaus wird für die Verzögerungszeit ein Desktop-Computer mit der Anwendung **Audacity 2.1.2** verwendet.

6.1.1. Loopback-Test 1 - Ausgabe der Soundkarte wird aufgenommen und dargestellt im Zeitbereich

Mit der Funktion *speaker-test* von ALSA wird ein Sinussignal mit einer Frequenz von 440 Hz wiedergegeben und mit *arecord* in eine Datei gespeichert.

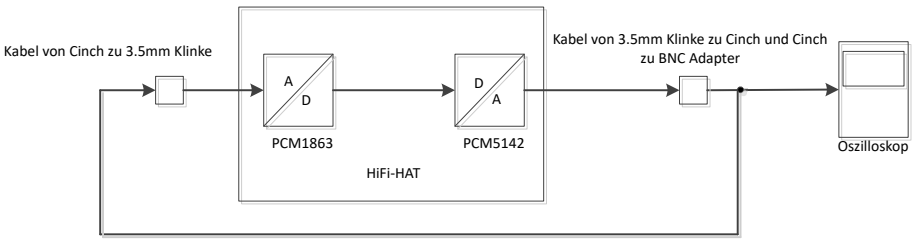

Abbildung 6.1.: Versuchsaufbau zur Aufnahme der Wiedergabe des Raspberry Pis

Abbildung 6.1 zeigt den Aufbau der Messung. Auf dem RPI wird die Ausgabe des Sinus Signals ausgeführt. Dieses Signal wird über ein T-Stück auf dem Oszilloskop dargestellt sowie auf die Aufnahme Buchse der Platine geführt. Wird die Einspeisung des Signals beendet, kann das aufgenommene Signal erneut wiedergegeben werden.

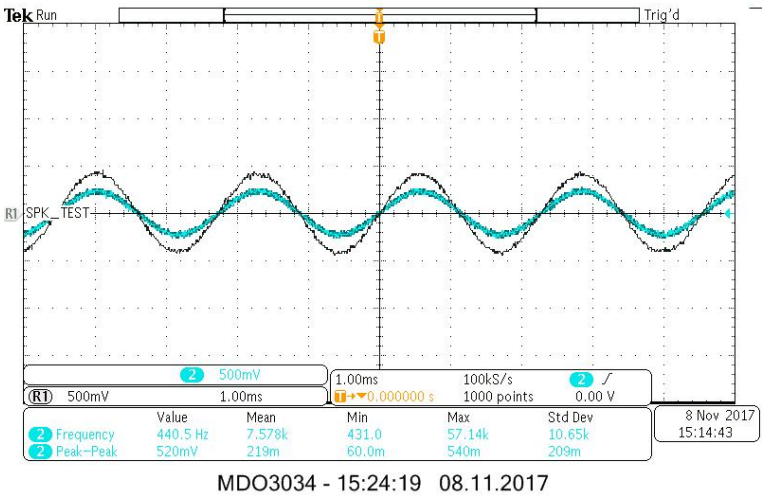

Abbildung 6.2.: Signalverläufe des ausgegebenen (schwarz) und aufgenommenen (türkis) Signals

Der türkise Graph in Abbildung 6.2 zeigt den Signalverlauf der Wiedergabe vom aufgenommenen Signal. Um einen Vergleich zur vorherigen Ausgabe herzustellen, wird es gespeichert und steht im schwarzen Verlauf zur Verfügung. Die Wiedergabe erfolgt mit einer um etwa Faktor zwei reduzierten Amplitude.

6.1.2. Loopback-Test 2 - Durchschleifen des Eingangssignals

Im zweiten Test wird, durch das Verknüpfen der Aufnahme sowie der Wiedergabe, der Verlauf eines weitergereichten Sinus Test Signals beobachtet. Hierfür werden in Kombination *arecord* und *aplay* von ALSA verwendet.

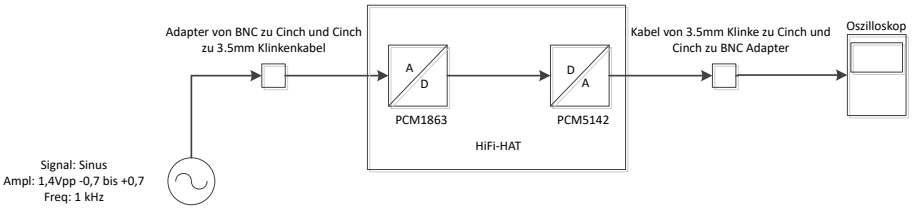

Abbildung 6.3.: Versuchsaufbau zur Auswertung der gesamten Messkette

Die Speisung des Sinussignals erfolgt durch den in Abbildung 6.3 dargestellten Funktionsgenerator. Dieses Signal wird innerhalb der Messkette nicht modifiziert und von den weiteren Komponenten des Messaufbaus nur durchgeschliffen.

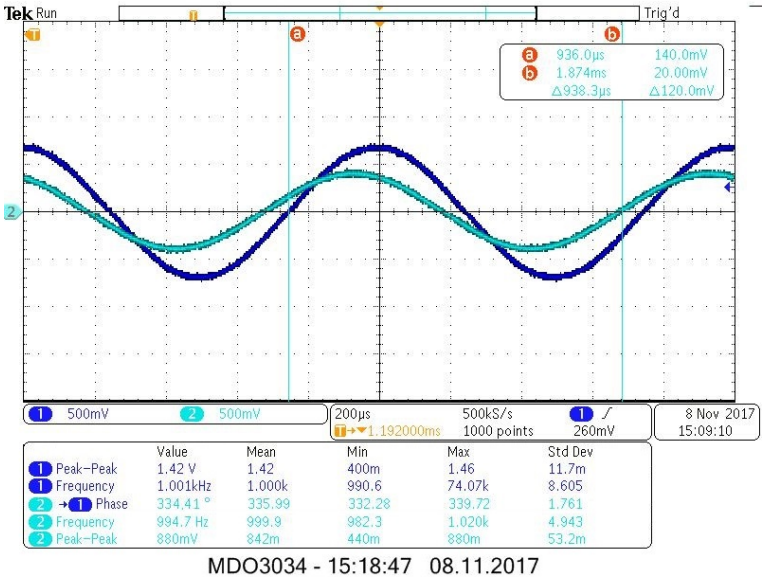

Abbildung 6.4.: Signalverläufe des eingespeisten (dunkleblau) sowie durchgeschliffenen (türkis) Signals

Das Oszillogramm aus Abbildung 6.4 zeigt in dunkelblau das eingespeiste Signal sowie in türkis die Ausgabe der Soundkarte. Außerdem wurden die Spitze-Spitze-Werte, die Frequenz und die Phasenverschiebung gemessen. Mit der Verwendung von Markern kann die Zeitdifferenz bestimmt werden.

6.1.3. Loopback-Test 2 - Durchschleifen des Eingangssignals im Frequenzbereich

Im zweiten Versuchsaufbau wird (siehe 6.3) die Amplitudengangsmessung aufgenommen. Um den Amplitudengang des Ausgangssignals zu Messen, wird der **Rohde und Schwarz UPV 66** Audio Analyzer verwendet.

Bei der Messung des Amplitudenganges wird ein Sinus mit 1V RMS von 20Hz bis 20kHz durchlaufen und die Antwort des Systems aufgenommen.

Als Eingangssignal wird ein Sinus mit 1V RMS mit den Frequenzen von 20Hz bis 20kHz durchlaufen, verwendet. Über den Audio Analyzer wird der Amplitudengang des Ausgangssignals betrachtet.

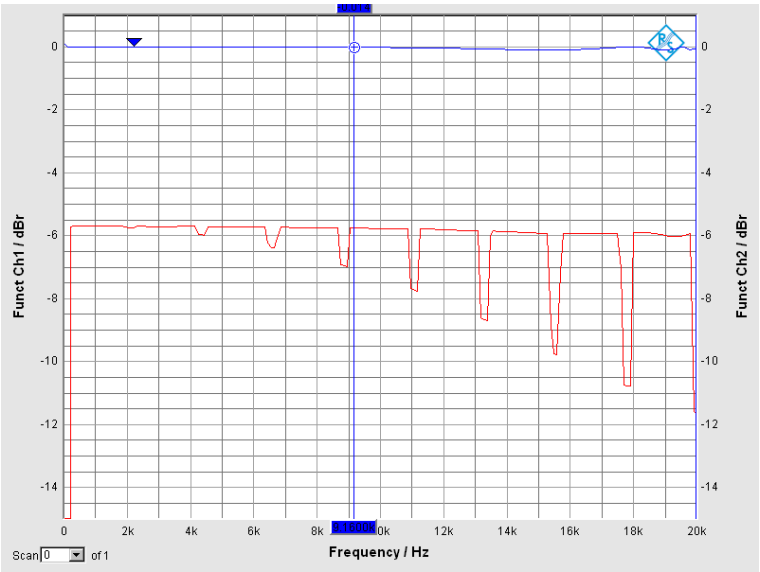

Abbildung 6.5.: Amplitudengang des HiFi-Hardware-Attached-on-Top (in rot) mit einer Abtastrate von f=44,1kHz, in blau wird die Referenz betrachtet

Die Abbildung 6.5 zeigt periodische Einbrüche, welche immer weiter zunehmen. Hierfür wurde die Messung mit einem RPI 2 wiederholt und ergab ein identisches Verhalten im Frequenzbereich. Die Messung wird mit höheren Abtastfrequenzen wiederholt, siehe A.9 und A.8. Bei der Messung mit einer Abtastfrequenz von f=96kHz fallen die Einbrüche häufiger, jedoch weniger stark aus. Mit 192 kHz sieht der Verlauf annähernd konstant aus.

In der Amplitudengangsmessung mit einer Abtastfrequenz von f=44,1kHz fallen besonders bei höheren Frequenzen die Einbrüche höher aus. Aus diesem Grund wird das Zeitsignal erneut bei einer Frequenz von f=16 kHz aufgenommen in Abbildung 6.6.

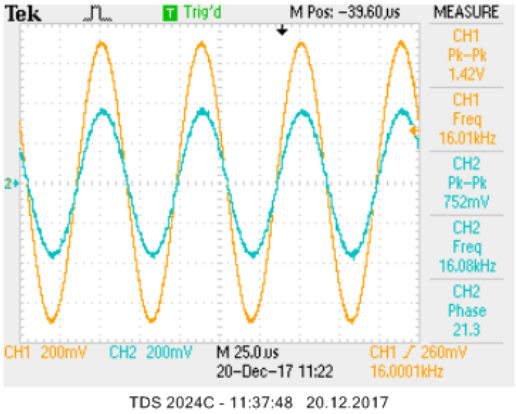

Abbildung 6.6.: Zeitsignal des HiFi-Hardware-Attached-on-Top eines Sinus bei f=16kHz mit einer Abtastrate von f=44,1kHz

Wird etwa 4 Sekunden gewartet, dann ändert sich die Ausgabe des Zeitsignals für einen Moment auf das folgende Oszillogramm 6.7. Darüber hinaus wird über einem Lautsprecher der Ton unterbrochen und es wird ein „Klopf“ Geräusch wahrgenommen.

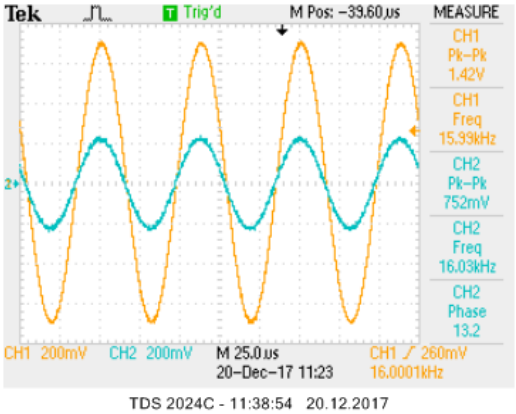

Abbildung 6.7.: Zeitsignal des Hifi-Hardware-Attached-on-Top eines Sinus bei f=16kHz mit einer Abtastrate von f=44,1kHz

6.1.4. Loopback-Test 2 - Messung der Verzögerung des Gesamtsystems

Für die Messung der Verzögerung wird der in Abbildung 6.8 dargestellte Versuchsaufbau verwendet.

Die dritte Messung wird gemäß des in Abbildung 6.8 gezeigten Aufbaus aufgenommen. Mit der Anwendung Audacity wird ein Audio Signal erzeugt, welches für eine Sekunde einen Sinus abspielt und vorher beziehungsweise nachher eine Signalpause von zwei Sekunden einlegt. Um einen Mittelwert bilden zu können, wird diese Sequenz drei mal wiederholt. Insgesamt wird mit drei verschiedenen Sequenzen, welche einen Sinus mit Frequenzen von 100Hz, 1000Hz und 10000Hz beinhalten, als auch unter verschiedenen Abtastfrequenzen 44,1kHz, 48kHz, 192kHz die Verzögerungszeit gemessen.

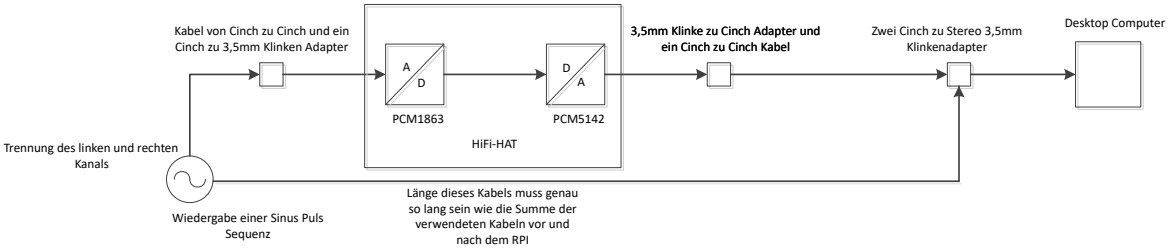

Abbildung 6.8.: Versuchsaufbau zur Messung der Verzögerung des Gesamtsystems

Mit einem Y-Adapter wird das Signal vom Computer auf zwei Cinch Kanäle aufgeteilt. Dieses wird über ein 1,5 Meter langes Kabel mit dem RPI verbunden. Der zweite Kanal wird mit einem drei Meter langem Kabel mit einem weiteren Y-Adapter verbunden. Die Ausgabe des RPI wird über ein 1,5 Meter langes Kabel mit der freien Cinch Buchse des zweiten Y-Adapters verbunden. Beide Signale durchlaufen somit eine identische Kabellänge und können am Computer aufgenommen werden. Es ergibt sich die folgende Messung:

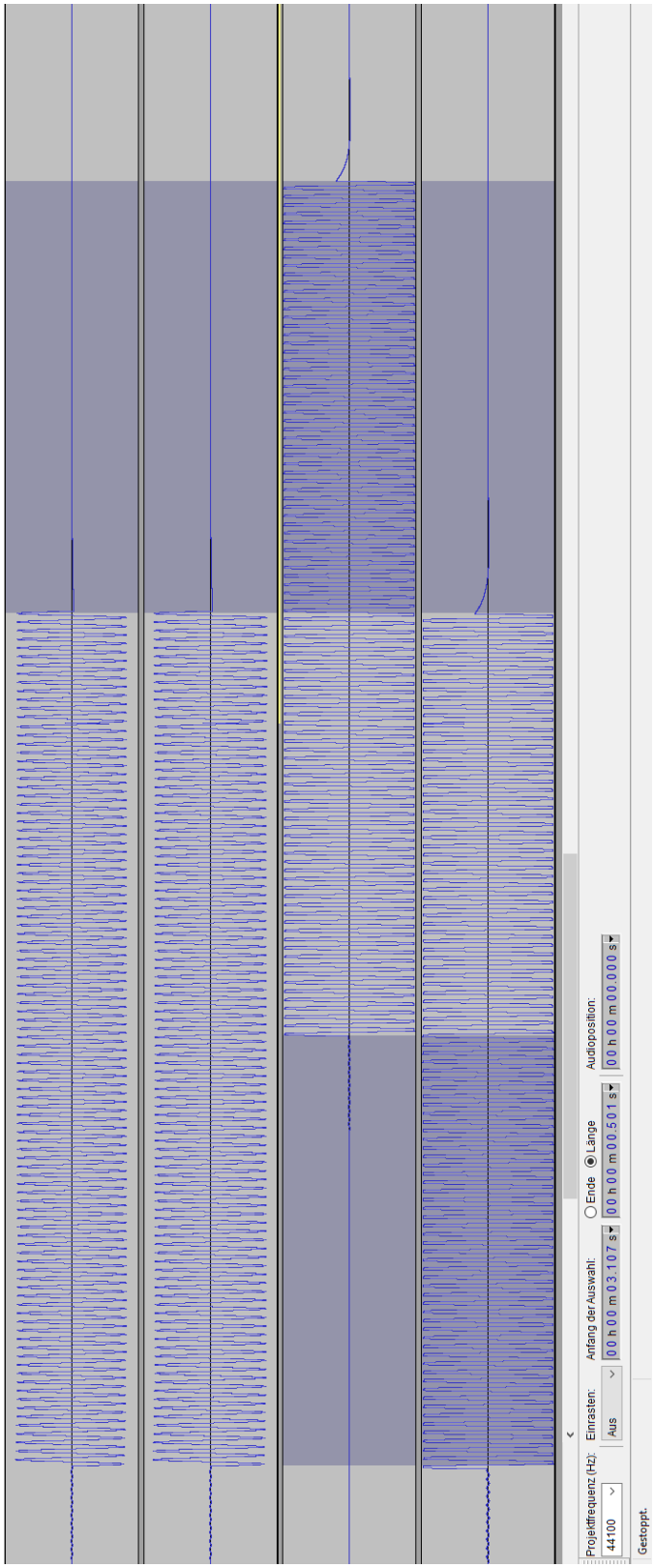

Abbildung 6.9.: Messung der Verzögerungszeit der gesamten Kette bei einer Abtastfrequenz von f=44,1kHz

Die Abbildung 6.9 zeigt die Anwendung Audacity. In der oberen Stereo Tonspur wird die abgespielte Sequenz angezeigt. In der zweiten Tonspur wird im ersten Kanal das durchgereichte Signal vom RPI dargestellt. Innerhalb des zweiten Kanals wird das Theorie-Signal verzögert aufgrund der Leitungslängen aufgenommen. In jeder Messung wurde eine Verzögerungszeit von t=500ms gemessen.

6.2. Ergebnisse

Innerhalb beider Tests wird das Eingangssignal im Zeitbereich, gedämpft um ungefähr den Faktor zwei, übertragen.

Das Verhalten beider Soundkarte weist in Verbindung mit dem RPI Probleme im Frequenzbereich auf. Erwartungsgemäß wird ein konstanter Verlauf, abhängig vom Eingangspegel, angestrebt damit jede Frequenz gleich stark verstärkt beziehungsweise gedämpft wird. Die dargestellten Einbrüche sind frequenzunabhängig. Wenn die Anzahl der Messpunkte variiert wird, erscheinen bei weniger Messpunkten eine geringere Anzahl an Einbrüchen, da die Messung schneller durchlaufen wird. Werden die Messpunkte erhöht, erhöht sich die Anzahl der Einbrüche. Die Periodizität ist abhängig von der Signalfrequenz als auch von der Abtastfrequenz des RPIs.

Die Messung der Verzögerungszeit der Gesamtkette ergibt bei allen drei Sequenzen sowie innerhalb aller Abtastfrequenzen eine Verzögerungszeit von t=500ms.

7. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Gerätetreiber für die HiFi-HAT Audio Aufsteckplatine von Sebastian Albers entwickelt und getestet. Erst mit diesem Treiber ist eine Verwendung der Platine außerhalb des Demo Programms möglich. Im Raspbian Betriebssystem wird die Soundkarte entsprechend als Wiedergabe- sowie Aufnahmegerät angezeigt. Außerdem sind Parameter wie die Bitzahl und Abtastrate der Aufsteckplatine einstellbar.

Darüber hinaus ist gemäß der HAT-Spezifikation eine automatische Treibererkennung implementiert. Das DTO, welches im EEPROM abgelegt ist, wird beim Hochfahren auf dem I2C-0 Bus ausgelesen. Zusätzlich ist für die reibungslose Inbetriebnahme ein Installationsskript entwickelt worden. Es lädt die aktuellen *kernel header* Quellen aus dem Internet, baut aus den Klartext Dateien die entsprechenden Module, kopiert sie in den korrekten Ordner und erneuert die Liste der Modulabhängigkeiten. Damit ist gewährleistet, dass die Soundkarte bei jedem RPI mit der Betriebssystem Raspbain Wheezy funktioniert sowie bei einer Einhaltung der Ordnerstrukturen für die nächsten Versionen erhalten bleibt.

Im Anschluss wurde die Aufsteckplatine zwei Loopback Tests unterzogen. Innerhalb dieser ist überprüft worden ob die Soundkarte die Signale unverändert aufnimmt sowie wiedergibt. Die Aufsteckplatine erweitert den RPI um weitere Schnittstellen und soll auf die Signalform kaum bis gar keinen Einfluss auf den Signalverlauf haben. In den Messungen wurde eine Dämpfung um den Faktor zwei und eine Verzögerungszeit von 500 ms gemessen.

Darüber hinaus wird ein unerwünschtes Verhalten im Frequenzbereich festgestellt sowie im Zeitbereich nachgestellt. Die Ausgabe des HiFi-HAT bricht zeitabhängig ein. Bisherige Erkenntnisse der Fehlersuche schließen den Messaufbau und Unter- sowie Überläufe aus.

8. Ausblick

Im derzeitigen Stand wird die Soundkarte vom Betriebssystem erkannt sowie verwendet. Wenn die HiFi-HAT als Standardgerät verwendet wird, können Audio-Signale aus Applikationen wiedergegeben werden. Somit stellt der Treiber eine Grundlage für weitere Software-Lösungen zur Signalverarbeitung. Dem ungeachtet besteht in einigen Bereichen Potenzial für Erweiterungen von Funktionalitäten.

Der Codec-Treiber wird weitestgehend für eine schnellstmögliche Funktionalität entwickelt. Somit folgt, dass bei weitem nicht alle über 100 einstellbare Register im Code abgebildet werden können. Darunter fällt zum Beispiel bei der Aufnahme die Option zwischen zwei bis acht Kanälen wählen zu können. Die Platine kann hingegen nur bauartbedingt im Stereo Format aufnehmen. Daher sind vorwiegend die Funktionen mit einer hohen Relevanz implementiert worden. Im ASOC Subsystem existieren Codec Treiber mit über 5000 Zeilen Code, welche nicht alle Funktionen abdecken.

Ein weiteres Problem stellt der Maschinentreiber dar. Für die Entwicklung ist der *simple-card* Maschinentreiber verwendet worden. Dieser ist allgemein entwickelt und kann keine Eigenheiten beherbergen. Im Fall der HiFi-HAT Platine muss der DAU zur Änderung der Taktquelle einen GPIO-Pin auf High oder Low schalten, um somit auf der gleichen Taktfrequenz, wie des ADUs mit dem entsprechend eingestellten Taktteiler zu arbeiten. Diese Besonderheit wird in beiden Codec-Treibern abgebildet, was gegen die Grundidee des ASOC Layers, den Codec-Treiber plattformunabhängig zu implementieren, spricht.

Optimierungsbedarf besteht im gegenwärtigen *Makefile*. Aktuell werden die Module in eigenen Ordnern gebaut und der *make* Befehl beinhaltet viele Variablen, welche im *Makefile* gesetzt werden können. Darauf aufbauend könnten *clean* und *install* Optionen implementiert werden, welche das Installationsskript nahe zu vollständig obsolet machen würden.

Den größten Schwerpunkt wird die Fehlersuche der Einbrüche beim Weitergeben des eingespeisten Signals einnehmen. Bisher konnte der Versuchsaufbau sowie Unter- beziehungsweise Überläufe ausgeschlossen werden. Die Vermutung liegt nahe, dass ein anderer Sound Prozess das Ausgabesignal beeinflusst. Da eine höhere Priorisierung mit dem *renice* Befehl kein Erfolge brachte, müssen die Zugriffsrechte angepasst werden.

Literaturverzeichnis

[1] Jürgen Quade, Eva-Katharina Kunst, *Linux-Treiber entwickeln*, dpunkt-verlag, (2015), ISBN 978-3-86490-288-8

[2] Sreekrishnan Venkateswaran, *Essential Linux Drivers*, Prentice Hall, (2015), ISBN 978-0-13-239655-4

[3] Alessandro Jonathan Corbet, Greg Kroah-Hartman Rubini, *Linux Device Drivers*, O'Reilly, (2005), ISBN 0-596-00590-3

[4] Jan Newmarch, *Linux Sound Programming*, apress, (2017), ISBN 978-1-4842-2495-3

[5] Klaus Dembowski, *Raspberry Pi - Das technische Handbuch*, Springer Vieweg, (2015), ISBN 978-3-658-08710-4

[6] Ralf Jesse, *Embedded Linux mit Raspberry Pi und Co.* , mitp, (2016), ISBN 978-3-95845-063-9

[7] Jürgen Quade, Eva-Katharina Kunst,*Kern-Technik*, Linux Magazin, 8 (2017), 76-78

[8] Takashi Iwai, *Sound Systems on Linux: From the Past To the Future*, UKUUG Linux 2003, 2003

[9] Florian Meier, *The JamBerry - A Stand-Alone Device for Networked Music Performance Based on the Raspberry Pi*, Linux Audio Conference 2014, 2014

[10] Sebastian Albers, *Entwicklung und Realisierung einer Raspberry Pi-Aufsteckplatine zur Bereitstellung von hochwertigen analogen Audioschnittstellen*, Bachelorthesis am Department Informations- und Elektrotechnik der HAW-Hamburg (2017)

[11] Jeff Tranter, *Introduction to Sound Programming with ALSA*, http://www.linuxjournal.com/article/6735 [10.12.2017]

[12] Dave Phillips, *A User's Guide to ALSA*, http://www.linuxjournal.com/node/8234/print [10.12.2017]

[13] *How it works: Linux audio explained*, http://tuxradar.com/content/how-it-works-linux-audio-explained [10.12.2017]

[14] *Linux ALSA sound notes*, http://www.sabi.co.uk/Notes/linuxSoundALSA.html [10.12.2017]

[15] Alexandre Belloni, *ASoC: Supporting Audio on an Embedded Board*, http://free-electrons.com/pub/conferences/2016/elce/belloni-alsa-asoc/belloni-alsa-asoc.pdf [10.12.2017]

[16] *Audio in embedded Linux systems*, http://free-electrons.com/doc/legacy/audio/embedded_linux_audio.p [10.12.2017]

[17] *Device Tree Specification*, https://www.devicetree.org/downloads/devicetree-specification-v0.1-20160524.pdf [10.12.2017]

[18] *About the Device Tree*, http://www.ofitselfso.com/BeagleNotes/AboutTheDeviceTree.pdf [10.12.2017]

[19] *Device Trees, overlays, and parameters*, https://www.raspberrypi.org/documentation/configuration/devic tree.md [10.12.2017]

[20] Thomas Petazzoni, *Device Tree for Dummies*, http://free-electrons.com/pub/conferences/2014/elc/petazzoni-device-tree-dummies/petazzoni-device-tree-dummies.pdf [10.12.2017]

[21] *Device Tree Usage*, https://elinux.org/Device_Tree_Usage [10.12.2017]

[22] *Linux Kernel and Driver Development Training*, http://free-electrons.com/doc/training/linux-kernel/linux-kernel-slides.pdf [10.12.2017]

[23] *Kernel building*, https://www.raspberrypi.org/documentation/linux/kernel/building.md [10.12.2017]

[24] *Configuring the kernel*, https://www.raspberrypi.org/documentation/linux/kernel/configuring.md [10.12.2017]

A. Anhang

A.1. Auf der CD beiliegende Daten

- Installationsordner
- Messordner
- Bachelorthesis

A.2. Abbildungen

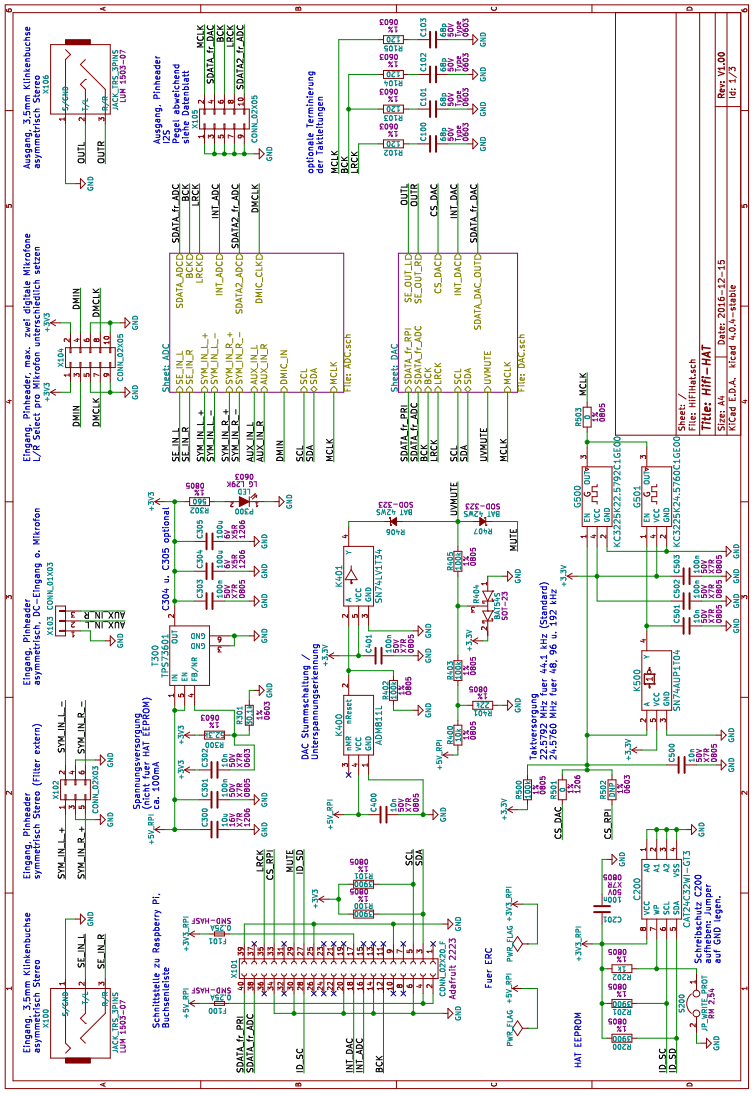

Abbildung A.1.: HiFI-Hardware-Attached-on-Top Schaltplan

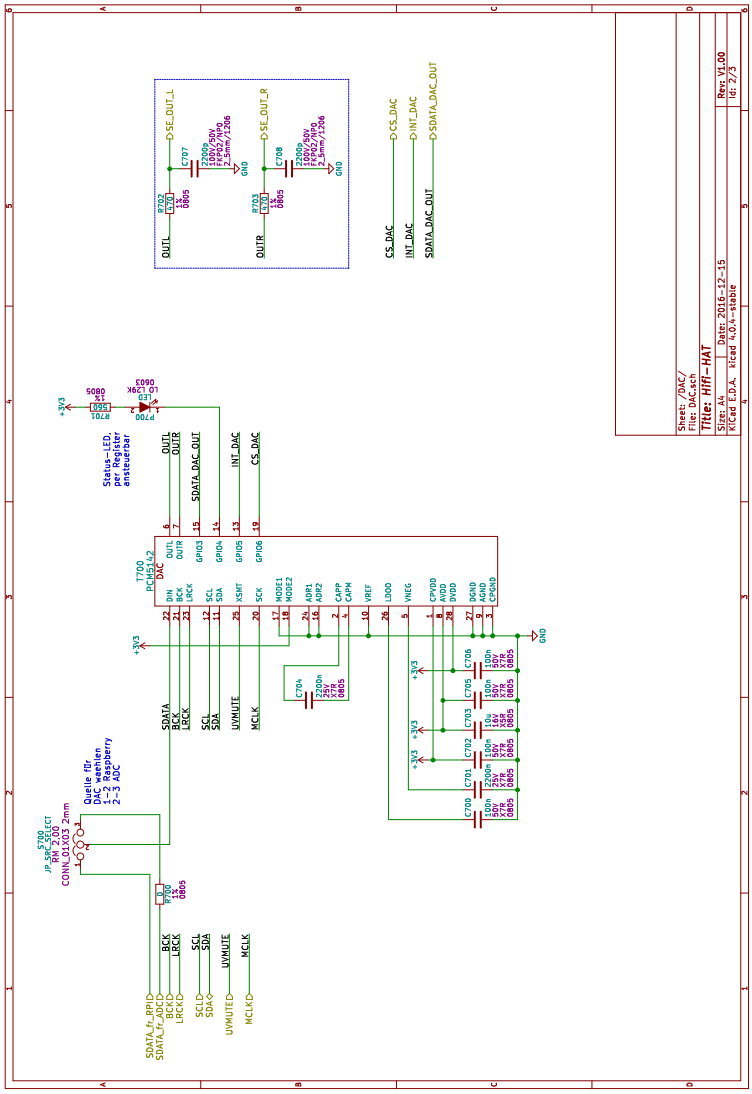

Abbildung A.2.: Digital-Analog-Umsetzer Schaltplan

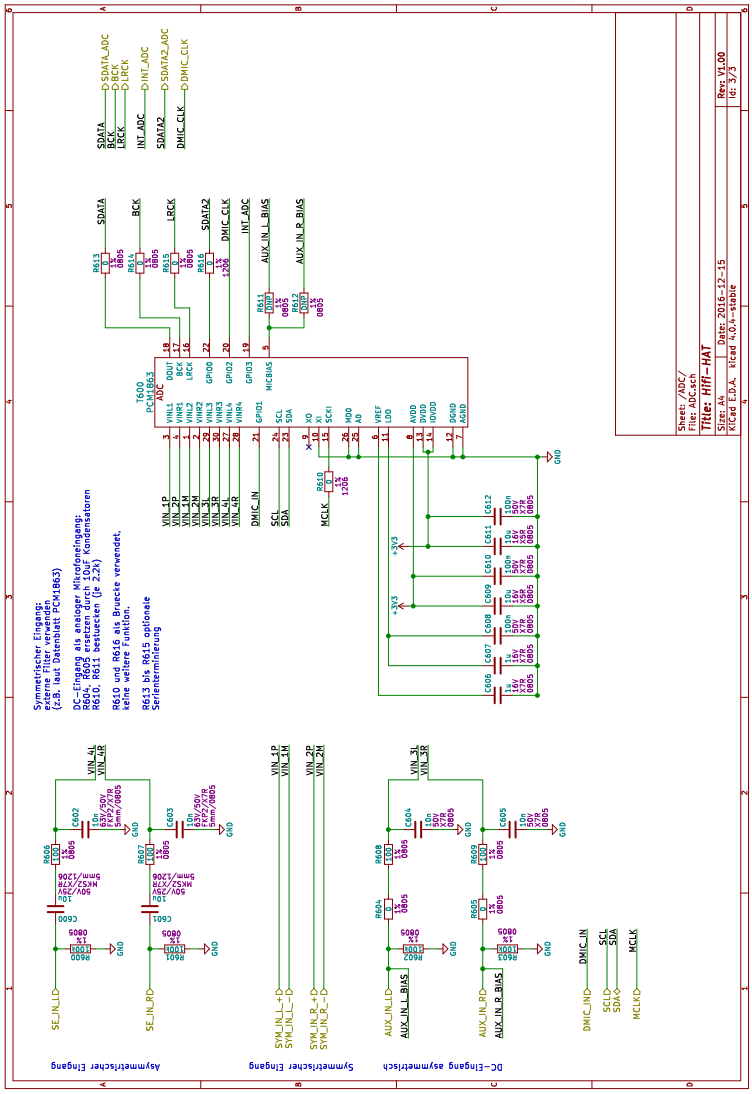

Abbildung A.3.: Analog-Digital-Umsetzer Schaltplan

```
pi@raspberrypi:~ $ ls -la /lib/modules/4.9.30-v7+/kernel/drivers/
total 164
drwxr-xr-x 41 root root 4096 May 30 20:32 .
drwxr-xr-x 11 root root 4096 May 30 20:32 ..
drwxr-xr-x  2 root root 4096 May 30 20:32 bcma
drwxr-xr-x  5 root root 4096 May 30 20:32 block
drwxr-xr-x  2 root root 4096 May 30 20:32 bluetooth
drwxr-xr-x  2 root root 4096 May 30 20:32 cdrom
drwxr-xr-x  3 root root 4096 May 30 20:32 char
drwxr-xr-x  2 root root 4096 May 30 20:32 clk
drwxr-xr-x  2 root root 4096 May 30 20:32 connector
drwxr-xr-x  2 root root 4096 May 30 20:32 extcon
drwxr-xr-x  2 root root 4096 May 30 20:32 gpio
drwxr-xr-x  3 root root 4096 May 30 20:32 gpu
drwxr-xr-x  2 root root 4096 May 30 20:32 hid
drwxr-xr-x  2 root root 4096 May 30 20:32 hwmon
drwxr-xr-x  5 root root 4096 May 30 20:32 i2c
drwxr-xr-x  8 root root 4096 May 30 20:32 iio
drwxr-xr-x  8 root root 4096 May 30 20:32 input
drwxr-xr-x  3 root root 4096 May 30 20:32 leds
drwxr-xr-x  3 root root 4096 May 30 20:32 md
drwxr-xr-x 14 root root 4096 May 30 20:32 media
drwxr-xr-x  2 root root 4096 May 30 20:32 mfd
drwxr-xr-x  4 root root 4096 May 30 20:32 misc
drwxr-xr-x  3 root root 4096 May 30 20:32 mmc
drwxr-xr-x  7 root root 4096 May 30 20:32 mtd
drwxr-xr-x 14 root root 4096 May 30 20:32 net
drwxr-xr-x  2 root root 4096 May 30 20:32 nvmem
drwxr-xr-x  2 root root 4096 May 30 20:32 of
drwxr-xr-x  3 root root 4096 May 30 20:32 power
drwxr-xr-x  3 root root 4096 May 30 20:32 pps
drwxr-xr-x  2 root root 4096 May 30 20:32 pwm
drwxr-xr-x  2 root root 4096 May 30 20:32 regulator
drwxr-xr-x  2 root root 4096 May 30 20:32 rtc
drwxr-xr-x  2 root root 4096 May 30 20:32 scsi
drwxr-xr-x  2 root root 4096 May 30 20:32 spi
drwxr-xr-x  2 root root 4096 May 30 20:32 ssb
drwxr-xr-x  9 root root 4096 May 30 20:32 staging
drwxr-xr-x  3 root root 4096 May 30 20:32 tty
drwxr-xr-x  2 root root 4096 May 30 20:32 uio
drwxr-xr-x 10 root root 4096 May 30 20:32 usb
drwxr-xr-x  4 root root 4096 May 30 20:32 video
drwxr-xr-x  4 root root 4096 May 30 20:32 wl
pi@raspberrypi:~ $
```

Abbildung A.4.: Aufbau des driver Ordners

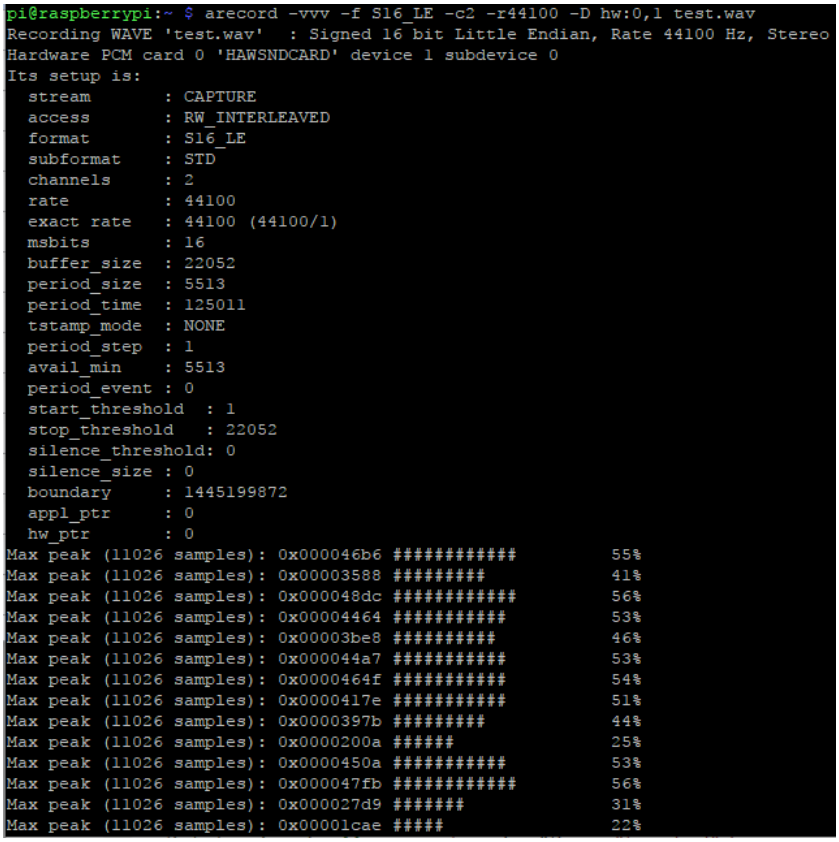

Abbildung A.5.: Ausgabe des *arecord* Befehls mit ausführlichem Output

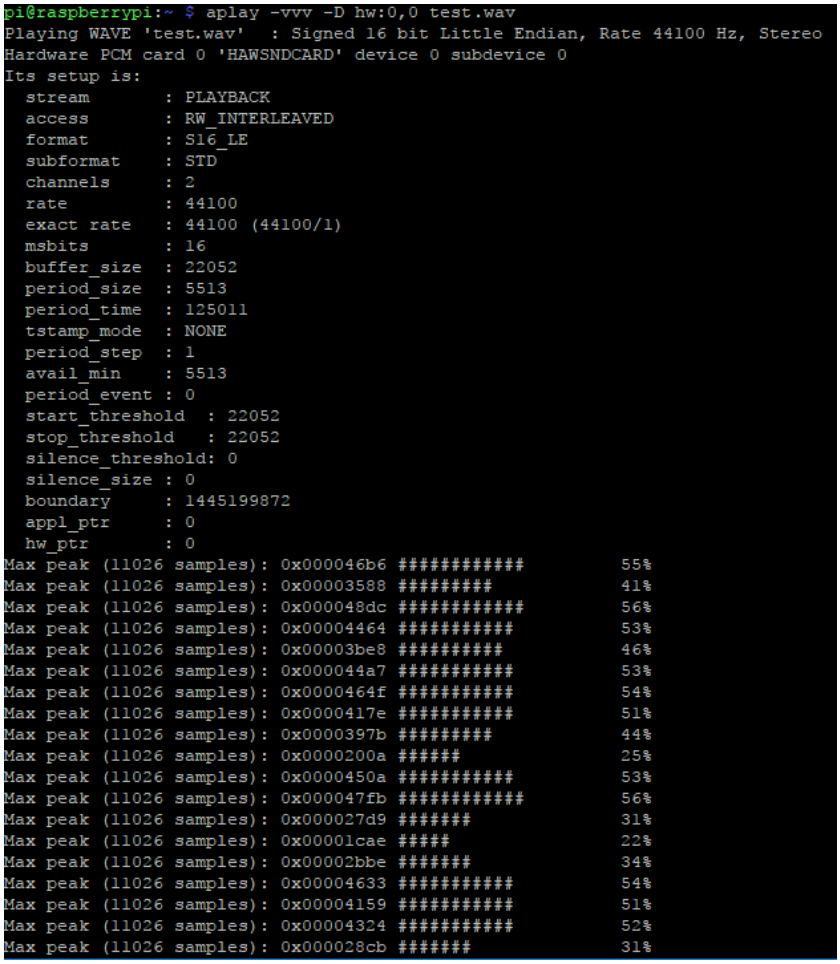

Abbildung A.6.: Ausgabe des *aplay* Befehls mit ausführlichem Output

```
pi@raspberrypi:~ $ cat /proc/asound/HAWSNDCARD/pcm1c/sub0/hw_params
access: RW_INTERLEAVED
format: S16_LE
subformat: STD
channels: 2
rate: 44100 (44100/1)
period_size: 5513
buffer_size: 22052
pi@raspberrypi:~ $ cat /proc/asound/HAWSNDCARD/pcm1c/sub0/status
state: RUNNING
owner_pid   : 1351
trigger_time: 2840.099892918
tstamp      : 0.000000000
delay       : 916
avail       : 916
avail_max   : 5513
-----
hw_ptr      : 2873200
appl_ptr    : 2872284
pi@raspberrypi:~ $ arecord -f S16_LE -c2 -r44100 -D hw:0,1 | aplay -r44100 -D hw
:0,0
Recording WAVE 'stdin' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'stdin' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

Abbildung A.7.: Aufbau des driver Ordners

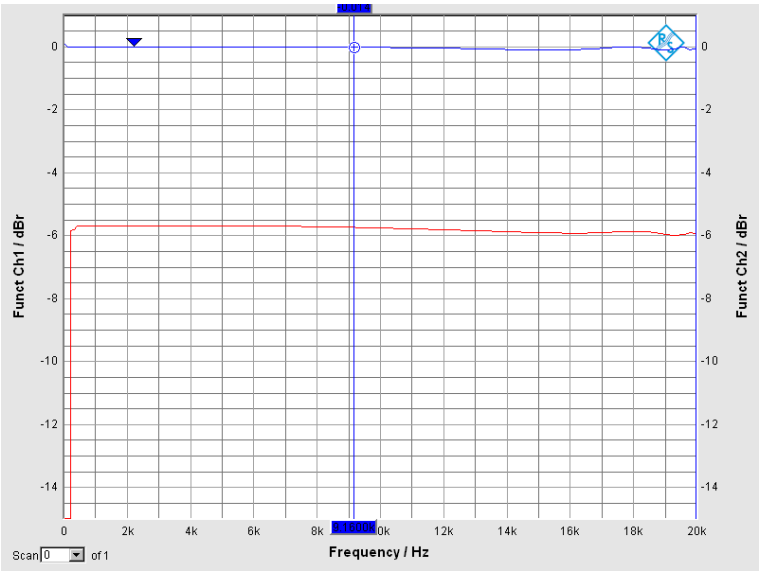

Abbildung A.8.: Amplitudengang des HiFi-Hardware-Attached-on-Top mit einer Abtastrate von f=192kHz

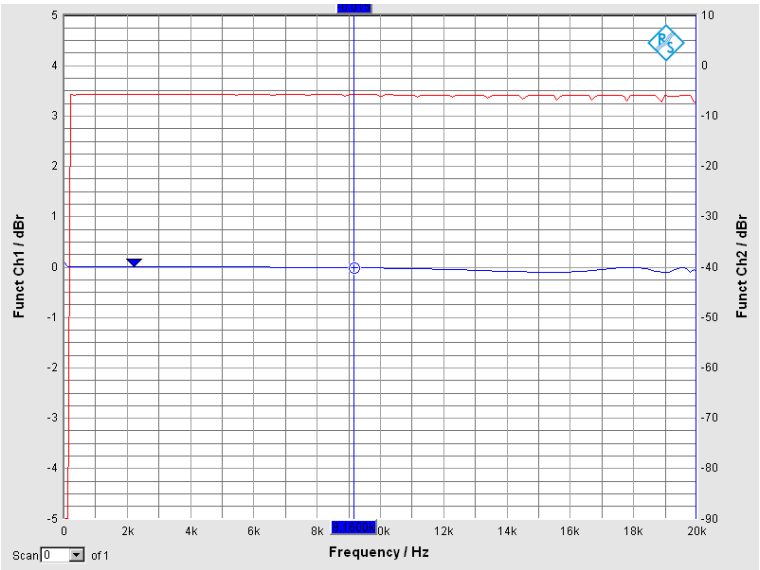

Abbildung A.9.: Amplitudengang des HiFi-Hardware-Attached-on-Top mit einer Abtastrate von f=96kHz

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 9. Januar 2018	
Ort, Datum	Unterschrift